Міністерство освіти і науки України
Донбаська державна машинобудівна академія (ДДМА)

О. Ю. Мельников

ПРОГРАМУВАННЯ ТА АЛГОРИТМІЧНІ МОВИ

Навчальний посібник

для здобувачів вищої освіти за спеціальностями «Системний аналіз та наука про дані» та «Інформаційні системи і технології»

Видання 2-е, зі змінами

Затверджено
на засіданні вченої ради
Протокол № 9 від 24.04.2025

Краматорськ
ДДМА
2025

УДК 004.021+004.43
М 48

Рецензенти:

Манзюк Е. А., д-р техн. наук, професор, Хмельницький національний університет;

Шевченко І. В., канд. техн. наук, доцент, Національний аерокосмічний університет «Харківський авіаційний інститут».

Мельников О. Ю.

М 48 Програмування та алгоритмічні мови : ***навчальний посібник [для здобувачів вищої освіти за спеціальностями «Системний аналіз та наука про дані» та «Інформаційні системи і технології»]*** / О. Ю. Мельников. – Краматорськ : ДДМА, 2025. – 240 с.

ISBN 978-617-7893-01-0

Містить основні теоретичні відомості; рекомендації до виконання й вимоги до оформлення робіт з дисципліни «Програмування й алгоритмічні мови»; матеріали про роботу з мовами програмування: Паскаль (Free Паскаль, ABC-Паскаль), Сі (Турбо Сі, Dev C++), Object-Pascal (середовище Lazarus), – а також основи об'єктно-орієнтованого програмування з прикладами на C++. Пропонується як методичні матеріали для виконання лабораторних і самостійних робіт, а також як посібник для самостійного вивчення й підготовки до складання заліку та іспиту.

УДК 004.021+004.43

ISBN 978-617-7893-01-0

ЗМІСТ

ВВЕДЕННЯ

Цій навчальний посібник містить матеріали щодо основи робіт з мовами програмування Паскаль і Сі з прикладами в реалізаціях Free Паскаль, ABC Паскаль, Object Pascal (середовище візуального програмування Lazarus), Турбо Сі й Турбо C++ (Dev-C++).

Передбачається, що з основами інформатики здобувачі були ознайомлені в процесі одержання середньої освіти, тому поняття різновидів обчислювальних процесів докладно не розглядаються. Також відсутні докладні інструкції побудови блок-схем алгоритмів.

Структурно посібник складається із чотирьох розділів, що відповідають змістовним модулям однойменної дисципліни ОПП «Системний аналіз» та ОПП «Інформаційні системи та технології»:

1. Основи програмування мовою Паскаль.
2. Основи програмування мовою Сі.
3. Основи візуального програмування.
4. Основи об'єктно-оріентованого програмування.

Останній розділ як обов'язковий компонент містить введення в об'єктну модель за відомим підручником Граді Буча (Grady Booch).

Кожен розділ містить блок основних теоретичних відомостей, кілька лабораторних робіт (від 4 до 9) і завдання для самостійної роботи. Передбачається, що лабораторні роботи здобувач повинен виконувати в аудиторний час, під керівництвом викладача, а завдання, винесені на самостійну роботу, – у позааудиторний. Кожна лабораторна робота супроводжується одним або декількома докладними прикладами написання програм з коментарями.

Звіт за завданням повинен містити:

– текст завдання;
– текст програми;
– результати роботи програми.

Здобувачі вищої освіти заочної форми навчання можуть використати цей посібник як методичні матеріали для самостійної роботи при підготовці до заліково-екзаменаційній сесії (точний перелік завдань, призначених саме для заочної форми навчання, визначається викладачем).

Хоча посібник призначений для здобувачів вищої освіти за спеціальностями «Системний аналіз» та «Інформаційні системи та технології», він може виявитися корисним і для інших спеціальностей.

1 ОСНОВИ ПРОГРАМУВАННЯ МОВОЮ ПАСКАЛЬ

1.1 Основні відомості про мову програмування Паскаль

Відповідно до одного з визначень, *алгоритм* – це перелік деталізованих інструкцій, що однозначно описують процес перетворення вихідних даних на результат розв'язання задачі.

Для опису алгоритму можуть використатися різні способи:

– вербальний (словесний запис: «якщо умова виконана, то перейти до кроку 5, інакше повернутися до кроку 2...»);

– графічний (блок-схеми);

– псевдокод (формальні алгоритмічні мови).

Як правило, сучасні програми створюються на так званих мовах програмування високого рівня, більшість яких розроблені на основі одного із чотирьох базових мов: Фортрану, Бейсика, Паскаля й Сі.

У першому модулі розглянемо основи мови програмування Паскаль, створеного швейцарським математиком Нікласом Віртом у 70-х роках XX століття. Паскаль створювався як універсальна мова програмування, придатний і для вивчення здобувачами основ програмування, і для розроблення складних прикладних програм.

Розвиток комп'ютерної техніки й вихід на ринок нових операційних систем викликало появу низки модифікованих версій Паскаля: Turbo-Pascal, Borland-Pascal, ABC-Pascal, Free-Pascal і багатьох інших. Часто вибір програмістом середовища розробки визначається не особливостями реалізації мови програмування, а умовами використання цього середовища (необхідність наявності ліцензії). Оскільки всі транслятори мов із префіксом «Турбо» та «Free» офіційно можна використати для навчальних цілей безоплатно, наведені в цьому посібнику приклади програм перевірялися саме на цих версіях Паскаля, що не забороняє здобувачеві використати будь-яку іншу версію мови відповідно до власних можливостей і під особисту відповідальність.

Алфавіт мови Паскаль містить:

– латинські літери (великі й малі при цьому не відрізняються);

– арабські цифри;

– розділові знаки: крапка (.), крапка з комою (;), двокрапка (:), кома (,), апостроф (');

– знаки арифметичних операцій: плюс (+), мінус (-), множення (*), ділення (/);

– знаки операцій відносини: більше (>), менше (<), дорівнює (=);

– дужки: ( ), { }, [ ];

– пробіл.

У тексті, взятому в лапки, а також у коментарях можна використати будь-які символи, наявні на клавіатурі комп'ютера.

Коментар – це взята в особливі дужки будь-яка послідовність символів, крім дужки, що закриває. Можна використати {фігурні дужки} або (*комбінацію круглої дужки й зірочки*).

Усі дані в програмі можуть бути або константами, або змінними. Значення констант визначаються перед виконанням програми й не можуть змінюватися під час її виконання; значення змінних можуть бути змінені скільки завгодно раз. У Турбо Паскалі було додане поняття типізованої константи, однак це, по суті, та сама змінна із присвоєнням початкового значення до початку виконання програми.

Дані – як константи, так і змінні – завжди мають бути віднесені до якого-небудь типу. Під типом даного розуміється перелік його припустимих значень; тип також визначає перелік припустимих дій.

У Паскалі існує кілька груп типів, які можна поділити, з одного боку, на прості й структуровані, а з іншого боку – на стандартні (визначені) і створені програмістом. Усі нові типи даних обов'язково повинні бути визначені або в розділі опису типів, або в розділі опису змінних.

У таблиці 1.1 наведені базові й часто використовувані типи даних.

Таблиця 1.1 – Базові й часто використовувані типи даних

Тип	**Ім'я типу**	**Зміст**
Цілий	ShortInt	Цілі числа в діапазоні –128…127
	Byte	Цілі числа в діапазоні 0…255
	Integer	Цілі числа в діапазоні –32768…32767
	Word	Цілі числа в діапазоні 0…65535
	LongInt	Цілі числа в діапазоні ± 2 млрд
Дійсний (речовинний)	Real	Речовинні числа в діапазоні $\pm 10^{\pm 38}$
Логічний	Boolean	Логічні дані: можуть набувати значень TRUE (істина) або FALSE (неправда)
Символьний	Char	Можуть набувати значення однієї літери з набору символів комп'ютера
Перелічний		Набір значень, яких може набувати дане значення
Діапазон		Підмножина впорядкованих значень, обумовлена мінімальним і максимальним значеннями

Ім'я будь-якого елемента програми (змінної, константи, типу даних, мітки) називається *ідентифікатором*. Ідентифікатор може включати літери, цифри й символ підкреслення (пробіли неприпустимі), при цьому починатися повинен тільки з літери. Великі й малі літери в ідентифікаторі не відрізняються (NAME й name – це одне ім'я). Довжина ідентифікатора теоретично може бути будь-якою, але транслятором приймаються не всі символи (наприклад, у Турбо Паскалі істотними є тільки перші 63 символи).

Розглянемо правила опису міток, констант, типів даних і змінних.

Мітка – ціле число без знака (у ранніх версіях Паскаля) або звичайний ідентифікатор, від оператора відокремлюється двокрапкою. Використається для визначення оператора, на який здійснюється перехід. Усі використовувані в програмі мітки повинні бути визначені у відповідному розділі опису:

Label <Ім'я мітки>;

Наприклад:

Label 1;
Label Abc;

Для переходу на мітку використовується оператор безумовного переходу **Goto**, наприклад:

Goto Abc;

Часте використання міток у програмі показує невисокий рівень здатностей програміста: правильно написана програма, як правило, не містить жодної мітки.

Опис констант має вигляд

Const <Ім'я константи> = Значення;

Тип константи однозначно визначається її значенням та явно не описується. Ім'я константи – звичайний ідентифікатор. Наприклад:

Const a=1.92; c='S'; k=7;

Тут описані два числа (речовинне й ціле) і один символ.

За замовчуванням у всіх версіях Паскаля визначені спеціальні константи:

Pi=3.14159265...

Maxint – найбільше ціле число, яке дорівнює 32767.

У більш новітніх версіях Паскаля це вже не константи, а особливі функції, однак це не впливає на принципи їхнього використання.

Опис типів має вигляд:

Type <Ім'я типу> = Зміст;

Приклади опису нових типів («створених програмістом») будуть наведені в третьому підрозділі.

Опис змінних починається із ключового слова **Var,** після якого перелічуються змінні і їхні типи. Загальний вид:

Var <Ім'я змінної> : <тип змінної>;

Якщо описується декілька змінних одного типу, їх можна перелічити через кому:

Var k: integer; x, y: real; c: char;

Структурно програма мовою Pascal складається із заголовка, розділу описів, розділу операторів і закінчується крапкою. *Заголовок* – це службове слово **Program** й ім'я програми (через пробіл). Розділ операторів містить у собі послідовність операторів, відокремлених крапкою з комою (;) і обмежених операторними дужками – службовими словами **Begin** й **End**. Усі описи й оператори (дії) у Паскалі обов'язково відокремлюються один від одного крапкою з комою (;).

Program <ім'я>;
Label <мітка>,...,<мітка>;
Const <ім'я константи> = <константа>;
Type <ім'я типу> = <тип>;
Var <ім'я змінної>,...,<ім'я змінної> : <тип>;
Procedure <заголовок процедури>;
<тіло процедури>;
Function <заголовок функції>;
<тіло функції>;
Begin
<оператор>;
<оператор>
End.

Обов'язковим є тільки розділ операторів. Узагалі, всі інші елементи програми можуть бути відсутніми. З формального погляду наведений нижче приклад не містить помилок:

Begin End.

Практично у всіх версіях Паскаля порядок розміщення розділів описів довільний, єдине правило, яке необхідно дотримати: можна використати лише ті ідентифікатори, які вже були визначені раніше.

Розглянемо поняття виразу. Розрізняють вирази: *арифметичні, строкові й відносини.*

Арифметичні вирази визначають послідовність обчислення значення. Можуть містити в собі константи, змінні, стандартні функції, які відокремлюються дужками й знаками операцій.

Знаки арифметичних операцій: «+» (додавання), «–» (віднімання), «*» (множення), «/» (ділення), DIV (ділення на ціле), MOD (визначення залишку від ділення).

Дії виконуються ліворуч-праворуч із дотриманням такого старшинства:

а) вирази в дужках;

б) мультиплікативні операції (* , / , DIV , MOD);

в) адитивні операції (+ , -).

Тип результату виразу залежить від типів операндів, що беруть участь в операції. Тип результату додавання, віднімання й множення є **Integer**, якщо обидва операнди мають тип **Integer,** і **Real** в іншому випадку. Результатом операції ділення завжди є тип **Real**, а результат операцій **Mod** й **Div** завжди має цілий тип, тому що аргументи можуть бути тільки цілі.

Вираз відносин припускають відносини між двома числовими або строковими значеннями: «=» (дорівнює), «< >» (не дорівнює), «<» (менше), «>» (більше), «<=» (менше або дорівнює), «>=» (більше або дорівнює).

Логічні операції: NOT – заперечення, AND – множення (логічне І), OR – додавання (логічне АБО).

Приклад: A = B, (A > 0) AND (B > 0).

Вираз (як і будь-яка послідовність операторів) може бути записаний як в один рядок, так і на декількох рядках.

Оператори – це опис яких-небудь дій над даними (змінними, константами). Оператори відокремлюються один від одного крапкою з комою. В одному рядку програми можна записати декілька операторів, і навпаки: один оператор може розміщатися на декількох рядках.

Складний оператор – це сукупність послідовно виконуваних операторів, укладених в операторні дужки **Begin** й **End**. Усередині операторних дужок оператори також відокремлюються один від одного крапкою з комою. **Begin** й **End** – це не оператори, тому після **begin** і перед **end** крапки з комами, як правило, не ставляться. Складний оператор може знадобитися в тих випадках, коли відповідно до правил побудови конструкцій мови припустиме використання тільки одного оператора, а потрібно виконати низку операторів. Таким єдиним оператором може бути складний оператор, до якого входить низка операторів, що виконують необхідні дії.

Оператор присвоювання служить для обчислення значення виразу й присвоювання його змінної. Формат оператора:

<ім'я змінної> := <вираз>;

Спочатку обчислюється значення виразу, записаного праворуч, а потім змінній привласнюється це значення. Ім'я змінної й результат повинні належати одному типу. Виключення може становити випадок, коли тип результату більше широкий, чим тип виразу (наприклад, вираз цілого типу, а результат – дійсного типу).

У процесі вирішення завдань часто виникає необхідність в обчисленні елементарних функцій. Для звертання до функції необхідно у виразі записати ідентифікатор функції й у круглих дужках – аргумент. Аргументами можуть бути константи, змінні, вирази або інші функції. Для тригонометричних функцій кут треба задавати тільки в радіанах. Розрізняють стандартні арифметичні функції (табл. 1.2) і стандартні функції перетворення (табл. 1.3).

Таблиця 1.2 – Стандартні функції

Функція	**Математичний запис**	**Запис в Pascal**
Синус	***Sin x***	**Sin (X)**
Косинус	***Cos x***	**Cos (X)**
Арктангенс	***Arctg x***	**Arctan (X)**
Абсолютне значення	$\|x\|$	**Abs (X)**
Корінь квадратний	$\sqrt{x}$	**Sqrt (X)**
Квадратування	x^2	**Sqr (X)**
Обчислення експоненти	e^x	**Exp (X)**
Натуральний логарифм	***Ln x***	**Ln (X)**
Присвоєння знака	**–**	**Sgn (X)**

Оскільки в Паскалі всього три тригонометричні функції, інші необхідно розраховувати за відомими формулами, наприклад: тангенс – це синус, поділений на косинус:

$$\mathrm{tg}\ \frac{x}{y} \rightarrow \sin(x/y)/\cos(x/y);$$

Для піднесення змінної до деякого степеня використовується відомий з математики вираз $x^a = e^{a\ln x}$. Мовою Паскаль піднесення числа X до степеня A запишеться так:

exp(a * **ln**(x));

Логарифм числа за заданою основою також розраховується з використанням відомої формули

$$\log_a b = \frac{\ln b}{\ln a}.$$

Таблиця 1.3 – Стандартні функції перетворення й визначення

Trunc (x)	Обчислює цілу частину аргументу X
Round (x)	Визначає округлене значення X
Ord (x)	Визначає порядковий номер аргументу X у впорядкованому списку значень, обумовлений типом X
Chr (x)	Визначає символ, порядковий номер якого дорівнює аргументу X
Succ (x)	Видає значення, що розташоване за аргументом X у списку значень, обумовленому для типу X
Pred (x)	Видає значення, що передує аргументу X у списку значень, обумовленому для типу X

Функції ord / chr будуть корисні при роботі із символьними даними, а функції succ / pred – як із символьними даними, так і з даними перелічених типів.

Уведення даних до програми здійснюється за допомогою процедур:

Read(b1, b2,..., bn);
Readln(b1, b2,... , bn);
Readln;

де b1,b2,..., bn – імена змінних, значення яких уводяться.

Процедура **Read** просто здійснює уведення даних, **Readln** здійснює уведення й забезпечує перехід до початку нового рядка (має сенс при роботі з масивами символів – рядками). Дані вводяться послідовно в одному рядку й відокремлюються один від одного пробілами.

Readln забезпечує пропуск одного рядка й перехід до початку нового рядка. Але фактично це процедура затримки роботи програми до натискання користувачем клавіші Enter.

Для виводу даних на екран використовуються процедури:

Write(b1, b2,..., bn);
Writeln(b1, b2,..., bn);
Writeln;

де bl,b2,...,bn – вираз, значення якого виводяться (це можуть бути змінні або константи).

Write виконує вивід значень, розміщаючи виведені значення в одному рядку, **Writeln** виконує вивід значень і після виводу останнього значення здійснює перехід до нового рядка. **Writeln** без параметрів забезпечує пропуск рядка й перехід до початку нового рядка – це часто буває корисно при роботі з масивами.

Значення виразів у списку виводу можуть бути різних типів. Загальний вид запису виводу значень цілого типу:

Write (b: m);

а для виводу дійсного типу:

Write (b: m: n);

де b – арифметичний вираз;
m – поле, що відводить під значення;
n – частина поля, що відводить під дробову частину числа.

Наприклад:

Writeln (k:4, a:6:2, b:0:2);

У першому випадку виводиться ціла змінна на 4 позиції, у другому й третьому – речовинні змінні з 2 позиціями після коми, однак змінна **a** виводиться максимум на 6 позиціях, а кількість позицій змінної **b** розраховується програмою. «Зайві» позиції заповнюються пробілами перед числом, праворуч – нулями. Якщо k = 2, a = 2.35, b = 1.1, то результат попереднього прикладу набуде вигляду

```
__ __ __ 2
__ __ 2  .  3  5
1  .  1  0
```

Узагалі, числа із дробовою частиною записуються у двох формах: основний і логарифмічній (з порядком). Замість коми ставиться десяткова крапка. Замість основи степеня 10 ставиться літера E, що дозволяє відокремити мантису від показника степеня й записати всі символи числа в одному рядку. Незначний нуль перед десятковою крапкою може бути відкинутий. Приклади:

```
212.356
2.12356E+02
```

Лабораторна робота 1. Лінійний обчислювальний процес

Завдання. Скласти програму для обчислення функцій: b = f(x,y,z), де z = w(x,y) при постійних значеннях x і в. Варіанти завдань у вигляді значень *x*, *y* і функцій *f* й *w* задані в таблиці 1.4.

Таблиця 1.4

Вар.	$f(x, y, z)$	$w(x, y)$	x	y
1	*2*	*3*	*4*	*5*
0	$\sin(x)\cdot e^{z}$	$\mathrm{sn}\,x+\cos y$	-π	π
1	$e^{-3x}(\mathrm{ctg}\,z+3y)$	$\sqrt{\cos^2\lvert x\rvert+y}$	–3.22	0.15
2	$\dfrac{\sqrt[3]{x}+\cos 3y}{z+x^{z}}$	$\dfrac{2xy}{x-\sin y}$	3.27	–1.84
3	$\dfrac{(y+z)/(y+x)}{(x+z)^2}$	$\dfrac{2\cdot\sqrt{x+y}}{y+\mathrm{tg}\ x}$	–1.32	9.35
4	$x^{y}+\sqrt{\lvert x+y\cdot z\rvert}$	$\dfrac{5\sqrt{x}}{x^3+y^2}$	0.75	–2.55
5	$\lg\left(\lvert x\rvert+\sqrt{y}+\lvert z\rvert\right)$	$\dfrac{\cos x}{5y^2}$	–5.24	3.35
6	$z+\dfrac{y\,\mathrm{tg}\,x}{x\,\mathrm{ctg}\,y}$	$x+\cos y$	0.32	–5.75
7	$\dfrac{z}{x^2+y^2}$	$\dfrac{\sin x+\cos y}{x+y}$	–2.54	3.16
8	$\dfrac{z}{\ln x+y}+e^{x}$	$\dfrac{2x}{\sin^2 x+\cos^2 y}$	1.28	–2.82
9	$\sin z+\dfrac{\mathrm{tg}\,x}{\mathrm{ctg}\,y}$	$\dfrac{x+\lg y}{y+\lg\lvert x\rvert}$	–0.72	3.29
10	$\ln\lvert z\rvert+\dfrac{\lg x}{\cos y}$	$\sqrt{2x+0.5y}$	11.42	–2.76
11	$\dfrac{z^3}{x+y}+\mathrm{tg}^2 x$	$\dfrac{e^{y}}{\ln\lvert x\rvert+\ln y}$	–5.43	1.87
12	$\sin z+\dfrac{e^{x+y}}{\mathrm{arctg}\,x}$	$\dfrac{x}{\sin x}+\dfrac{y}{\cos y}$	3.57	–0.32
13	$z^{x/y}+\sin x$	$\cos\left(5x+6y+e^{x}\right)$	–1.42	12.1
14	$\dfrac{e^{x}}{2x+\ln\lvert x+y\rvert}+\sin z$	$\sin^2 x+\cos^2 y$	1.34	–0.65

Продовження таблиці 1.4

1	*2*	*3*	*4*	*5*
15	$\sqrt{\lvert x+y\rvert}+\operatorname{arctg} z$	$\frac{x+y}{\ln\lvert x+y\rvert}$	–3.12	1.78
16	$\frac{4y^3+5x^2}{z}+\operatorname{tg} z$	$\frac{x+\sqrt{x}}{\operatorname{tg}(x+y)}$	0.72	–3.47
17	$\frac{y\sqrt{x}+x\cdot e^y}{\ln\lvert z\rvert}$	$\frac{e^x+e^y}{x+y}$	–2.65	5.32
18	$z+\sin x+\ln\lvert x+y+z\rvert$	$\frac{2x+\sqrt[3]{\lvert x+y\rvert}}{\lg x}$	0.32	–1.32
19	$\frac{x+z^2+y^3}{\operatorname{tg}^2(x+y)}$	$\frac{x+\sqrt{\lvert x+y^2\rvert}}{\sin(x+y)}$	–4.54	0.45
20	$\operatorname{tg} z+\operatorname{tg} x+\operatorname{tg} y$	$\frac{x+y}{\ln\lvert x+y\rvert}$	2.52	–8.12
21	$\frac{\sin(x+y)}{z+x+y}$	$\frac{x-y^2}{\sqrt[3]{\lvert x+y\rvert}}$	–0.73	3.28
22	$\frac{\ln\lvert x+z\rvert}{\sqrt[3]{\lvert y+z\rvert}}$	$e^{x+y}+\cos(x+y)$	1.76	–0.75
23	$\frac{\sqrt[3]{\lvert x+y+z\rvert}}{\ln\lvert x+z\rvert}$	$\frac{\operatorname{tg}(x+y)}{\operatorname{ctg}(x-y)}$	–0.62	2.45
24	$\lg\lvert z\rvert+\lg\lvert x+y\rvert+\operatorname{tg} z$	$\sin x+\operatorname{tg} y$	2.65	–4.36
25	$\frac{e^{x+y}+\sin z}{e^{x-y}+\cos z}$	$\frac{\operatorname{tg}(x+y)}{\operatorname{tg}(x-y)}$	–0.9	1.2
26	$e^z\sin x$	$\sin x+\cos y$	π	$-\pi$
27	$\ln\lvert z\rvert\cdot\sin x$	$\sin^2 x+\cos^2 y$	$-\pi$	π

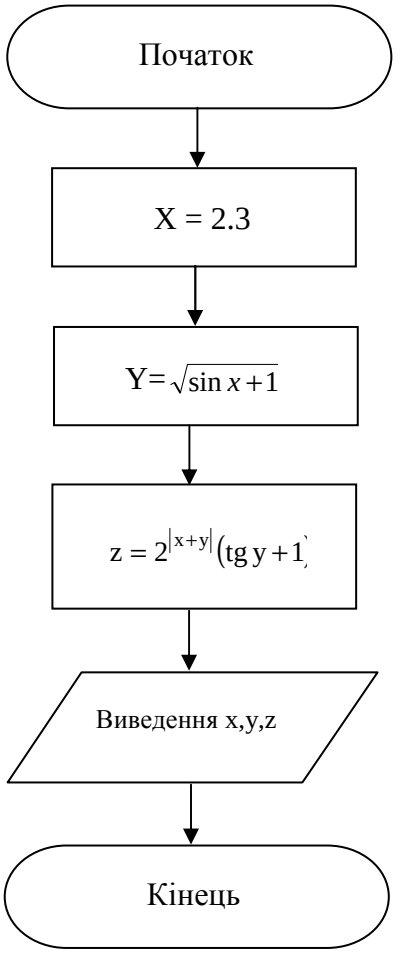
Приклад виконання завдання

Скласти програму для обчислення функції

$$z=2^{\lvert x+y\rvert}(\operatorname{tg} y+1),$$

де $y=\sqrt{\sin x+1}$.

Вихідні дані: x = 2.3.




```
Program lab1;
Const x = 2.3;
Var y,z:real;
Begin
     y := sqrt(sin(x)+1);
     z := exp(abs(x-y)*ln(2))*(sin(x)/cos(x)+1);
     Writeln('При x = ',x:3:1, ' y =',y:6:2, ' z =',z:6:2)
End.
```

1.2 Різновиди обчислювальних процесів

У прикладі попереднього підрозділу був наведений найпростіший вид алгоритму – лінійний обчислювальний процес. У такому алгоритмі дані або вводяться з клавіатури на початку програми, або задаються у вигляді констант, потім відбуваються певні розрахунки, і наприкінці – виведення результатів на екран.

Реально такі процеси майже не зустрічаються. Дуже часто використовуються циклічні обчислювальні процеси й процеси, що розгалужуються. Розглянемо їх по черзі.

Розгалуження в програмі виникають, якщо необхідно вибрати один з декількох можливих шляхів у вирішенні завдання. Оператор умовного переходу дозволяє змінити порядок виконання операторів у програмі залежно від певних умов. Загальний вид оператора умовного переходу:

If <умова> **then** <оператор1> **else** <оператор 2>;

Якщо умова, задана в операторі **If**, істинна, то виконується **then**-гілка, тобто оператор (простий або складовий), що розташований після **then** (<оператор 1>), інакше виконується **else**-гілка, тобто оператор, що розташований після **else** (<оператор 2>). Після виконання однієї з гілок робота програми продовжується з оператора, що розташований за **If**.

Усічений формат оператора умовного переходу:

If <умова> **then** <оператор>;

Оскільки «If-then-else» – це єдиний оператор, усередині нього (тобто перед else) крапки з комою не ставляться.

Усередині оператора умови можуть бути розташовані інші оператори умови. Формально в деяких трансляторах задається межа вкладеності умов, але реально цієї межі не досягти.

Циклічним називається обчислювальний процес, що містить багаторазові обчислення за тими самими залежностями (як правило, математичними), але для різних значень вхідних у нього змінних. Паскаль – одна з небагатьох мов програмування, у якої існує три види циклів, і єдина мова, де всі ці види задаються по-різному.

1. Цикл з передумовою. Дії (блок операторів) повторюються, поки умови, наведені перед початком циклу, виконуються:

While <умова продовження циклу> **do**
<оператори>;

Якщо в циклі необхідно виконати понад один оператор, то їх варто взяти в операторні дужки **begin – end**, тобто утворити з них складний оператор. Доти, поки дотримується умова, виконується тіло циклу (<оператори>). Якщо умова не виконується з самого початку, то виконання програми продовжується, починаючи з оператора, що розташований за циклом (у цьому випадку оператори циклу не виконаються жодного разу).

Серед операторів циклу обов'язково повинен бути присутнім оператор зміни умови продовження циклу (як правило, зміна певної змінної на величину кроку), інакше цикл перетвориться на нескінченний.

Приклад обчислення суми простого ряду:

```
s:=0; x:=0;
while x <=5 do begin
        s:=s+x*x;
        x:=x+0.5
              end;
```

2. Цикл з постумовою. Дії повторюються, поки не виконаються умови, наведені після закінчення циклу.

Repeat
<Оператор 1>;
. . .
<Оператор n>
Until <умова виходу із циклу>;

Особливості такого циклу:
– тіло циклу виконається як мінімум один раз;
– не потрібні операторні дужки (begin – end).
Приклад:

```
s:=0; x:=0;
repeat
        s:=s+x*x;
        x:=x+0.5
until x > 5;
```

3. Цикл із параметром (з лічильником, ітераційний). Дії повторюються, поки змінна-лічильник не досягне останнього значення:

For <змінна>:=<поч. знач.> **to** <кін. знач.> **do** <оператори>;

Особливості такого циклу:
– не потрібно примусово змінювати значень змінної циклу;
– змінна циклу може бути тільки цілого типу;
– крок зміни змінної дорівнює 1 або –1.
Найчастіше цикл із лічильником використовується при роботі з масивами.
Прості приклади:

```
s:=0; for i:=0 to 10 do s:=s+sqr(i/2);

s:=0; for i:=10 downto 0 do s:=s+sqr(i/2);
```

Лабораторна робота 2. Циклічний обчислювальний процес, що розгалужується

Завдання. Скласти програми для вирішення завдань.

Варіанти 0–10. Знайти суму $y = \sum \frac{F_1}{F_2}$, де a ≤ x ≤ b, x змінюється з кроком h = c. Варіанти завдань у вигляді значень F_1, F_2, a, b, c наведені в таблиці 1.5. Завдання вирішити, використовуючи цикли: а) WHILE (для непарних варіантів); б) REPEAT (для парних варіантів).

Варіанти 11–20. Обчислити таблицю значень функції

$$y = \begin{cases} F_1(x), \textit{якщо } x \le a; \\ F_2(x), \textit{якщо } x > a, \end{cases}$$

для значень аргументу *x* в інтервалі від xn до xk із кроком hx. Варіанти завдань у вигляді вихідних даних наведені в таблиці 1.6.

Варіанти 21–27. Обчислити таблицю значень функції

$$y = \begin{cases} \text{f1(x)}, & \textit{якщо} \quad x < 0, \\ \text{f2(x)}, & \textit{якщо} \quad 0 \le x \le 1, \\ \text{f3(x)}, & \textit{якщо} \quad x > 1, \end{cases}$$

де f_1, f_2 й f_3 наведені в таблиці 1.7. Методом перебору знайти екстремуми даної функції на відрізку. Початкове й кінцеве значення відрізка, а також крок табуляції задавати довільно.

Таблиця 1.5

Вар.	**F1**	**F2**	**a**	**b**	**c**
1	*2*	*3*	*4*	*5*	*6*
0	$\sin x$	x	$-\pi$	π	$\pi/10$
1	$2\sqrt{x^5}\cos x^2$	$x^3 + 3x^2 - x$	0.2	2.2	0.2
2	$\sqrt[3]{\cos x + x^3}$	$\frac{3x+5}{x^2}$	2.5	7.5	0.5
3	$\ln\lvert 3x - x^2 \rvert$	$2x^3 - \operatorname{tg} x$	0.5	4.5	0.3
4	$2^{x-1}\cos 3x$	$\sqrt{\lvert 1 + x - 2x^2 \rvert}$	–2	6	0.5
5	$\ln\lvert x - 5 \rvert + \sqrt[3]{5x}$	$x^{-2.5}\sin(2x+1)$	2.4	6.4	0.4
6	$e^{2x-1} + \cos x$	$\lg(2x+1)$	1.6	4.8	0.3

Продовження таблиці 1.5

1	*2*	*3*	*4*	*5*	*6*
7	x^3+2x^2-2	$x^{2x-1}+\cos x$	3.6	7.2	0.2
8	$\ln\frac{x+1}{2x-1}$	$\frac{x}{\operatorname{tg}x+\sin 2x}$	$-\pi$	π	$\pi/10$
9	$x^2+\ln x$	$\frac{x}{1+\operatorname{tg}x}$	0.3	3.3	0.3
10	$1+x^2-\operatorname{tg}x$	$\frac{x+\sin x}{x^2-\cos x}$	1.2	13.2	0.6

Таблиця 1.6

Вар.	**$F_1(x)$**	**$F_2(x)$**	**xn**	**xk**	**hx**	**a**
11	$2\sqrt{x^5}\cos x^2$	x^3+3x^2-x	0.2	2.2	0.2	1.2
12	$\sqrt[3]{\cos x+x^3}$	$\frac{3x+5}{x^2}$	2.5	7.5	0.5	5.0
13	$\ln\lvert 3x-x^2\rvert$	$2x^3-\operatorname{tg}x$	0.5	4.5	0.3	3.0
14	$2^{x-1}\cos 3x$	$\sqrt{\lvert 1+x-2x^2\rvert}$	–2	6	0.5	3
15	$\ln\lvert x-5\rvert+\sqrt[3]{5x}$	$x^{-2.5}\sin(2x+1)$	2.4	6.4	0.4	5
16	$e^{2x-1}+\cos x$	$\lg(2x+1)$	1.6	4.8	0.3	3
17	x^3+2x^2-2	$x^{2x-1}+\cos x$	3.6	7.2	0.2	4.8
18	$\ln\frac{x+1}{2x-1}$	$\frac{x}{\operatorname{tg}x+\sin 2x}$	$-\pi$	π	$\pi/10$	$\pi/5$
19	$x^2+\ln x$	$\frac{x}{1+\operatorname{tg}x}$	0.3	3.3	0.3	2.3
20	$1+x^2-\operatorname{tg}x$	$\frac{x+\sin x}{x^2-\cos x}$	1.2	13.2	0.6	4.2

Таблиця 1.7

Вар.	**$F_1(x)$**	**$F_2(x)$**	**$F_3(x)$**
1	*2*	*3*	*4*
21	$2\sqrt{x^5}\cos x^2$	x^3+3x^2-x	$\ln\lvert 3x-x^2\rvert$
22	$\frac{3x+5}{x^2}$	$\sqrt{\lvert 1+x-2x^2\rvert}$	$2x^3-\operatorname{tg}x$
23	$2^{x-1}\cos 3x$	$\ln\lvert x-5\rvert+\sqrt[3]{5x}$	$x^{-2.5}\sin(2x+1)$

Продовження таблиці 1.7

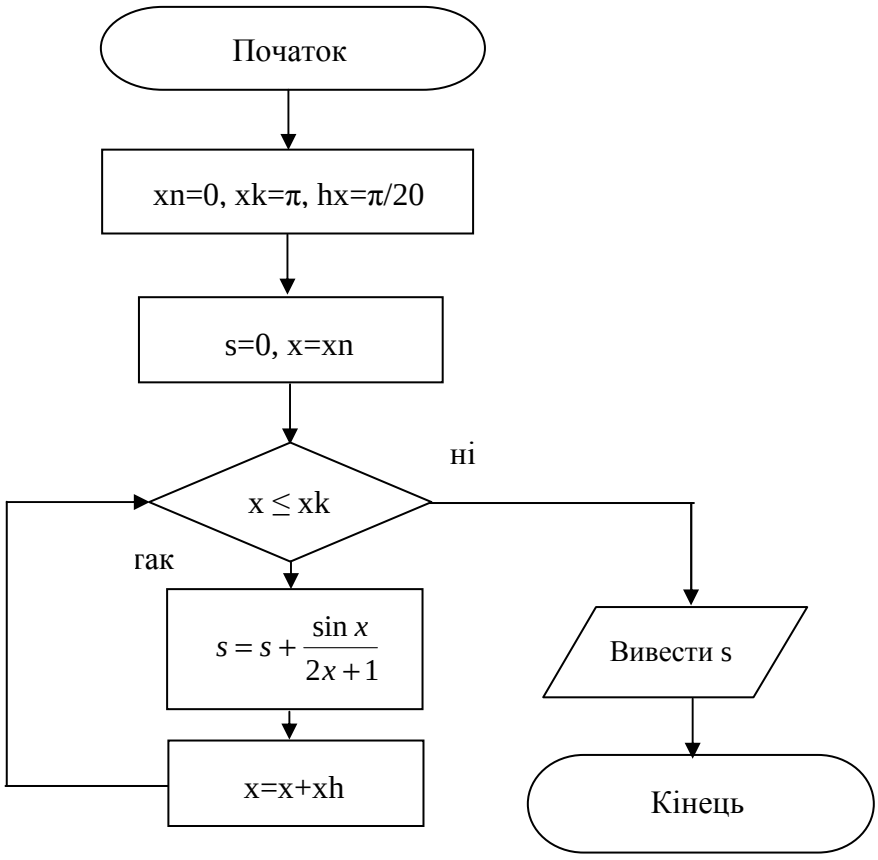
1	*2*	*3*	*4*
24	x^3+2x^2-2	$x^{2x-1}+\cos x$	$x^2+\ln x$
25	$\ln\frac{x+1}{2x-1}$	$\frac{x}{\operatorname{tg} x+\sin 2x}$	$\frac{x}{1+\operatorname{tg} x}$
26	$\ln\lvert 3x-x^2\rvert$	$2x^3-\operatorname{tg} x$	$\frac{x+\sin x}{x^2-\cos x}$
27	$1+x^2-\operatorname{tg} x$	$\frac{3x+5}{x^2}$	x^3+3x^2-x

Приклади виконання завдання

1. Знайти суму

$$S=\sum\frac{\sin(x)}{2x+1},\ \text{де } 0\le x\le\pi \text{ з кроком } \pi/20,$$

використовуючи цикл з передумовою.

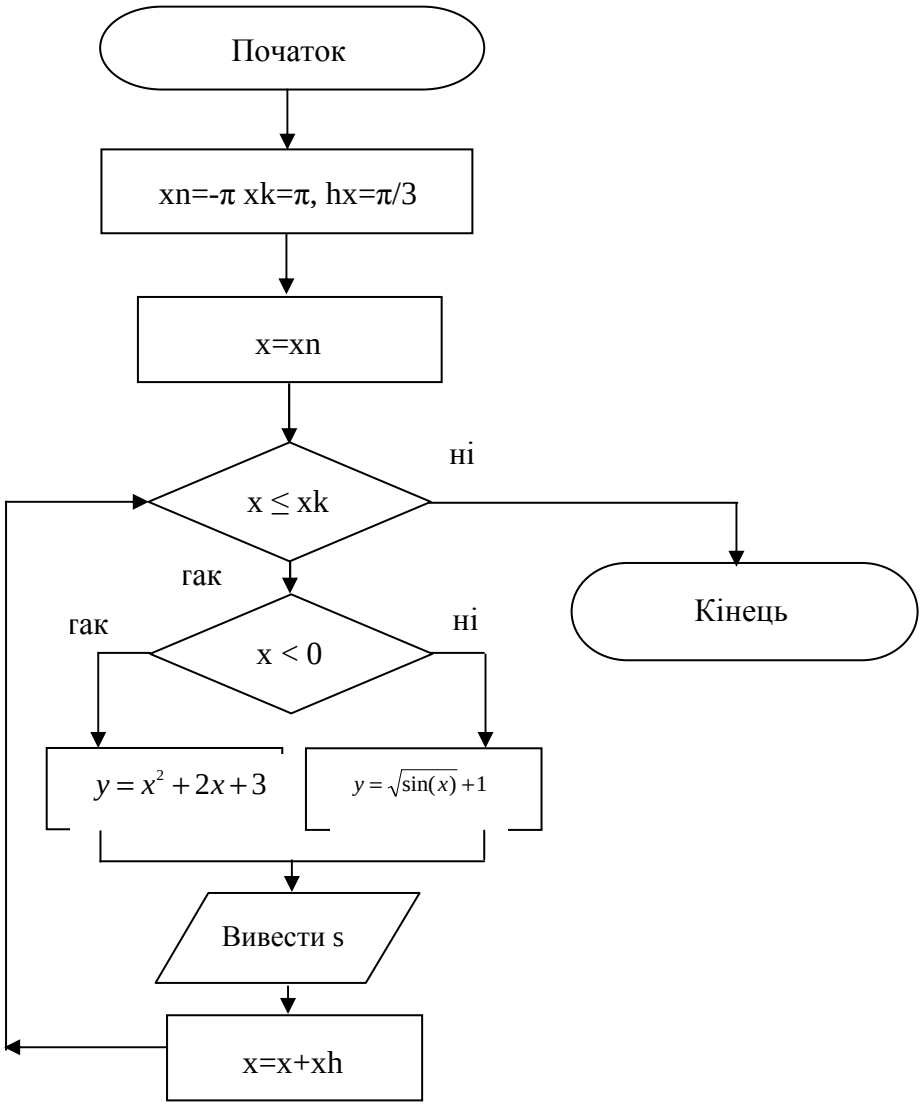


```
Program lab2a; { знаходження суми ряду }
Var x,s:real;
Const xn=0; xk=pi; hx=pi/20;
Begin
     s := 0; x := xn;
     while x <= xk do
     begin
          s := s + sin(x) / (2*x + 1);
          x := x + hx
     end;
     writeln ('s =', s:7:4)
End.
```

2. Обчислити таблицю значень функції

$$y = \begin{cases} \sqrt{\sin(x)} + 1, \textit{якщо } x \geq 0; \\ x^2 + 2x + 3, \textit{якщо } x < 0, \end{cases}$$

для значень x в інтервалі від –π до π з кроком π /3.



```
Program lab2b;   { табулювання функції }
Var x,y:real;
Const xn=-pi; xk=pi; hx=pi/3;
Begin
     writeln('ТАБЛИЦЯ ЗНАЧЕНЬ ФУНКЦІЇ');
     writeln('*************************');
     writeln('*     X     *     Y     *');
     writeln('*************************');
     x := xn;
     while x <= xk do
        begin
           if x >= 0 then y := sqrt(sin(x))+1
                     else y := sqr(x)+2*x+3;
              writeln('*', x:10:5,' *', y:10:5,' *');
              x := x + hx
        end;
     writeln('*************************')
End.
```

3. Знайти екстремуми функції

$$y = \begin{cases} \sqrt{\sin(x)} + 1, \textit{якщо } x \geq 0; \\ x^2 + 2x + 3, \textit{якщо } x < 0, \end{cases}$$

на інтервалі зміни аргументу від –π до π.

```
Program lab2c;   { екстремуми }
Var x,y,min,max,xmin,xmax:real;
Const xn=-pi; xk=pi; hx=pi/3; { крок задаємо довільно }
Begin
     max:=-10E+5; min:=10E+5; x:=xn;
     while x <= xk do
     begin
          if x >= 0 then y:=sqrt(sin(x))+1
                    else y:=sqr(x)+2*x+3;
          if y > max then begin max:=y; xmax:=x end else
          if y < min then begin min:=y; xmin:=x end;
          writeln('x=', x:10:5,' y=', y:10:5);
          x := x + hx
     end;
writeln('Мінімум дорівнює ', min:10:5,' при x=', xmin:10:5);
writeln('Максимум дорівнює ', max:10:5,' при x=', xmax:10:5)
End.
```

1.3 Нестандартні й обмежені типи даних, множини, оператор вибору варіанта

Як вже було наведено раніше, нові типи описуються в спеціальному розділі типів або визначаються безпосередньо при описі змінних.

До простих нестандартних типів даних, які описує програміст, відносяться перелічений й обмежений діапазони. Перелічений тип задається списком значень, яких може набувати змінна. Загальна форма завдання переліченого типу така:

Type Tname = (Al, A2, ... , AN);

Тут Tname – ім'я нового типу; A1,A2,...,AN визначають константи-значення нового типу даних. Послідовність значень типу упорядкована. Наприклад:

```
Type Season = (winter, spring, summer, autumn);
Var S: Season;
…
        If  S = Summer then S:=Autumn;
```

Відповідно, для значень типів, що перелічують, визначені стандартні функції **succ** й **pred:**

Succ(winter) → spring
Pred(autumn) → summer

Ще раз уточнимо, що значення переліченого типу – це не рядок тексту, а визначені константи.

Відрізок значень будь-якого порядкового типу може бути визначений як обмежений тип (інтервальний, тип діапазон):

Type Tname = MIN..MAX;

Тут Tname – ім'я типу; MIN, MAX – ліва й права межі діапазону, відокремлені двома крапками. Наприклад:

```
Type Days=1..31; Year=1900..2050;
Var d: days; y: year; m: 1..12;
```

При правильних параметрах транслятора спроба присвоїти змінній значення, що виходить за межі діапазону, призводить до повідомлення про помилку.

Поняття множини в Паскалі ґрунтується на математичному визначенні множини: це обмежена сукупність різних елементів. Для побудови конкретного множинного типу може бути використаний перелічений або інтервальний тип даних. Тип елементів, що становлять множину, називається базовим типом.

Множинний тип описується за допомогою службових слів Set of, наприклад:

```
Type M = Set of ‘A’..'D';
Var MS: M;
      m2: set of byte;
      C: Set of 0..7;
```

Константи множинного типу записуються у вигляді укладеної у квадратні дужки послідовності елементів або інтервалів базового типу, розділених комами, наприклад: ['A','C'] [0,2,7] [3,7,11..14]. Константа виду [ ] означає порожню множину.

Множина містить у собі набір елементів базового типу, всі підмножини даної множини, а також порожня підмножина. Якщо базовий тип, на якому будується множина, має К елементів, то кількість підмножин, що входять до цієї множини, дорівнює 2 у степені К. Змінна Т множинного типу

```
var T: Set of 1..3;
```

може набувати вісім різних значень:

[ ] [1,2]
[1] [1,3]
[2] [2,3]
[3] [1,2,3]

Порядок перелічення елементів базового типу в константах не має значення.

Значення змінної множинного типу може бути задано конструкцією виду [T], де T – змінна базового типу.

До змінних й констант множинного типу застосовні операції присвоювання (:=), об'єднання (+), перетинання (*) і віднімання (-):

['A','B'] + ['A','D'] (['A','B','D']
['A'] * ['A','B','C'] (['A']
['A','B','C'] - ['A','B'] (['C'].

Результат виконання цих операцій – величина множинного типу. До множинних величин застосовні такі операції: тотожність (=), нетотожність (<>), утримується в (<=), містить (>=). Результат виконання цих операцій має логічний тип, наприклад:

['A','B'] = ['A','C'] –> FALSE
['A','B'] <> ['A','C'] –> TRUE
['B'] <= ['B','C'] –> TRUE
['C','D'] >= ['A'] –> FALSE.

Для роботи з величинами множинного типу в Паскалі також використається оператор **in**, що перевіряє приналежність елемента базового типу, який розташований ліворуч від знака операції, множині, яка розташована праворуч від знака операції. Результат виконання – логічний:

A in ['A', 'B'] –> TRUE,
2 in [1, 3, 6] –> FALSE.

Величини множинного типу не можуть бути елементами списку вводу-виводу.

У кожній конкретній реалізації транслятора Паскаля кількість елементів базового типу, на якому будується множина, обмежена. У Турбо-Паскалі, наприклад, кількість базових елементів не повинна перевищувати 256.

Оператор вибору варіанта є узагальненням умовного оператора: він надає змоги виконати один з декількох операторів залежно від значення деякого виразу, називаного селектором. У загальному випадку оператор має вигляд

case <селектор> **of**
    <список 1>:<оператор 1>;
    <список 2>:<оператор 2>;
…
    <список n>: <оператор n>;
else <оператор>
end;

де <селектор> – вираз певного типу;

<оператор> – будь-який оператор мови, у тому числі й складовий;

<список> – список значень виразу селектора, відокремлених комами або у вигляді діапазону, або одне його значення.

Оператор варіанта вибирає для виконання того оператора, одна з міток якого дорівнює поточному значенню виразу селектора. Якщо значення виразу не збігається з жодною з міток, то виконується оператор, що відповідає **else**. Гілка **else** необов'язкова. Після закінчення виконання вибраного оператора керування передається в кінець оператора **case**. Селектор і мітки повинні бути одного типу.

Приклад:

```
case k of
        1: y:=k;
        2..5: y:=k*k;
        6, 7, 9: y:=ln(k);
        else y:=0
end;
```

Приклад ілюструє різні варіанти подання списку значень селектора, а також необхідність крапки-коми перед **else**.

Лабораторна робота 3. Інші типи даних і вибір варіанта

Завдання. Обчислити таблицю значень функції

$$Y=\begin{cases} F_1(X), \text{ якщо } X\in X_1 \\ F_2(X), \text{ якщо } X\in X_2 \\ \\ F_3(X), \text{ якщо } X\in X_3 \\ F_4(X), \text{ якщо } X\in X_4 \end{cases}$$

для цілочислених значень аргументу X в інтервалі [Xn .. Xk]. Множини X_1, X_2, X_3 і X_4, а також функції F_1, F_2, F_3, F_4 задані в таблицях 1.8 й 1.9. Варіанти наведені у вигляді «Варіант mn».

Таблиця 1.8

M	**Xn**	**Xk**	**X_1**	**X_2**	**X_3**	**X_4**
0	-10	15	[-2,5]	Парні числа з інтервалу [6,10]	Непарні числа з інтервалу [6,10]	Інші
1	0	30	Парні числа з інтервалу [10,20]	Непарні числа з інтервалу [10,20]	[1,9]	Інші
2	1	25	Числа, кратні 3, з інтервалу [10,20]	Числа, не кратні 5, з інтервалу [10,20]	[2,7] й [21,23]	Інші

Таблиця 1.9

n	$F_1(x)$	$F_2(x)$	$F_3(x)$	$F_4(x)$
1	$2\sqrt{x^5}\cos x^2$	x^3+3x^2-x	$\sqrt[3]{\cos x+x^3}$	$\frac{3x+5}{x^2}$
2	$2^{x-1}\cos 3x$	$\sqrt{\lvert 1+x-2x^2\rvert}$	$\ln\lvert 3x-x^2\rvert$	$2x^3-\operatorname{tg} x$
3	$\ln\lvert x-5\rvert+\sqrt[3]{5x}$	$x^{-2.5}\sin(2x+1)$	$e^{2x-1}+\cos x$	$\lg(2x+1)$
4	x^3+2x^2-2	$x^{2x-1}+\cos x$	$\ln\frac{x+1}{2x-1}$	$\frac{x}{\operatorname{tg} x+\sin 2x}$
5	$x^2+\ln x$	$\frac{x}{1+\operatorname{tg} x}$	$1+x^2-\operatorname{tg} x$	$\frac{x+\sin x}{x^2-\cos x}$
6	$\ln\lvert 3x-x^2\rvert$	$2x^3-\operatorname{tg} x$	$2\sqrt{x^5}\cos x^2$	x^3+3x^2-x
7	$\sqrt[3]{\cos x+x^3}$	$\frac{3x+5}{x^2}$	$\ln\lvert x-5\rvert+\sqrt[3]{5x}$	$x^{-2.5}\sin(2x+1)$
8	$e^{2x-1}+\cos x$	$\lg(2x+1)$	x^3+2x^2-2	$x^{2x-1}+\cos x$
9	$\ln\frac{x+1}{2x-1}$	$\frac{x}{\operatorname{tg} x+\sin 2x}$	$x^2+\ln x$	$\frac{x}{1+\operatorname{tg} x}$
0	$\ln\lvert 3x-x^2\rvert$	$2x^3-\operatorname{tg} x$	$2^{x-1}\cos 3x$	$\sqrt{\lvert 1+x-2x^2\rvert}$

Приклад виконання завдання

Для цілочислених значень аргументу обчислити таблицю значень функції

$$z=\begin{cases}\sin(x)+1, \text{ якщо } x\in X_1\\ \ln^2(x), \text{ якщо } x\in X_2\\ e^{x-4}, \text{ якщо } x\in X_3\\ 2\cos(x), \text{ якщо } x\in X_4\end{cases}$$

де X_1 – множина цілих чисел [0,3]; X_2 – множина парних чисел з інтервалу [4,10]; X_3 – множина непарних чисел з інтервалу [4,10]; X_4 – інші цілі числа інтервалу [–2,10].

```
Program lab3; { табулювання функції }
Type arg = -2..10;
Var x:arg; z:real;
Begin
    for x := -2 to 10 do begin
      case x of
              0..3: z := sin(x)+1;
              4..10: if x mod 2 = 0 then z := sqr(ln(x))
                                    else z := exp(x-4);
              else z := 2*cos(x)
      end; {case}
      writeln ('При x =',x:3,' z =',z:6:2)
    end
End.
```

1.4 Складні типи даних – масиви

До складних (структурованих) типах даних у Паскалі відносяться масиви, записи й файли. Докладніше робота зі структурами даних вивчатиметься в курсі «Алгоритми й структури даних», однак ніяке вивчення основ програмування не може обійтися без поняття масиву.

Масив складається з фіксованого числа компонентів одного типу, характеризується ім'ям і розмірністю.

Загальний вид опису типу масив:

Type T = **array** [T1, T2, … ,Tk] **of** tc;

де T – ім'я масиву; T1,..., Tk – індекси; TC – базовий тип.

Кількість індексів k визначає розмірність масиву. Індекси задаються у вигляді обмеженого типу даних (діапазону).

Масиви можна описувати безпосередньо в розділі описів змінних, наприклад:

Var m: **array** [1..10] **of integer**;

Кількість індексів, необхідних при звертанні до елемента масиву, визначає k-мірність масиву. При k = 1 масив називається одномірним, при k = 2 – двомірним. Одномірний масив відповідає поняттю вектора (лінійної таблиці, рядка), двомірний масив – поняттю матриці (набору векторів, прямокутної таблиці). Кількість вимірів масиву в Турбо Паскалі обмежена 7, в інших реалізаціях може бути іншою, однак на практиці рідко використовуються масиви із кількістю вимірів понад 4.

Доступ до будь-якого елемента масиву здійснюється записом ідентифікатора масиву, за яким у квадратних дужках знаходиться індексний вираз (номер елемента). Індексний вираз має бути в діапазоні, обумовленому типом індексу. Для оголошеного в прикладі масиву M у програмі доступні такі індексні змінні: M[1], M[2], ... , M[10].

Селективна обробка масиву – це відокремлення елементів, що задовольняють умовам, і обробка цих фрагментів. Відокремлені елементи перераховують, перемножують, підсумують або формують новий масив. Часто зустрічаються такі умови обробки елементів масиву:

парні	A [i] mod 2 = 0
непарні	A [i] mod 2 < > 0
кратні k	A [i] mod k = 0
не кратні k	A [i] mod k < > 0
на парних місцях	i mod 2=0
на непарних місцях	i mod 2 < > 0
позитивні	A [i] > 0
негативні	A [i] < 0
ненегативні	A [i] >= 0
в інтервалі [x1,x2]	(A [i] >= x1) and (A [i] <= x2)

Якщо необхідно знайти суму певних елементів, то перед початком циклу змінної суми обов'язково привласнюється 0, а усередині циклу до цієї змінного по черзі додаються знайдені елементи.

Якщо необхідно знайти добуток певних елементів, то перед початком циклу змінної добутку обов'язково привласнюється 1, а усередині циклу ця змінна по черзі множиться на знайдені елементи.

Якщо необхідно знайти кількість певних елементів, то перед початком циклу змінної кількості обов'язково привласнюється 0, а усередині циклу до цієї змінної щоразу додається 1.

При формуванні нового масиву необхідно передбачити другий індекс, що спочатку буде відігравати роль числа елементів у новому масиві (спочатку – 0, потім у циклі збільшується на 1).

Знаходження найбільшого й найменшого значень масиву виконується в циклі, у якому поточний елемент масиву порівнюється з найбільшим (найменшим) із всіх попередніх елементів масиву. Якщо поточний елемент виявиться більше (менше) найбільшого (найменшого) з попередніх, то його треба вважати новим найбільшим (найменшим). У протилежному разі найбільше (найменше) зберігає старе значення:

$$Y_{max} = \begin{cases} Y_i, \text{ якщо } Y_i > Y_{max}, \\ Y_{max}, \text{ якщо } Y_i \le Y_{max}; \end{cases}$$

$$Y_{min} = \begin{cases} Y_i, \text{ якщо } Y_i < Y_{min}, \\ Y_{min}, \text{ якщо } Y_i \ge Y_{min}. \end{cases}$$

Якщо тілом циклу є інша циклічна структура, то такі цикли називають вкладеними або складними. Цикл, що містить у собі інший цикл, називається *зовнішнім*. Цикл, що знаходиться в тілі іншого циклу, називається *внутрішнім*.

Двомірний масив характеризується ім'ям і розмірністю, де перший індекс визначає рядок, а другий – стовпець. Елемент масиву – змінна, розташована на перетинанні рядка й стовпця. При обробці таких масивів звичайно й застосовують вкладені цикли.

Приклад опису двомірного масиву:

Var m: **array** [1..3,1..3] **of integer**;

Посилання на елемент матриці M, що лежить на перетинанні *і-го* рядка й *j-го* стовпця, має вигляд M[i,j].

При обробленні матриць часто доводиться виділяти елементи:

– k-го рядка	A[k, j]	j=l,...,m
– k-го стовпця	A[i, k]	i=l,...,n

Для квадратних матриць (m=n):

– головна діагональ	A[i,i]	i=l,...,n
– побічна діагональ	A[i, n+l-i]	i=l,…,n
– наддіагональні	A[i, j]	i>j
– піддіагональні	A[i, j]	i<j

Лабораторна робота 4. Обробка одномірних масивів

Завдання. Скласти програму для вирішення задач, варіанти яких наведені в таблиці 1.10.

Таблиця 1.10

Вар.	**Завдання**
1	*2*
0	Знайти кількість позитивних і суму непарних елементів масиву B (15)
1	Знайти суму позитивних і кількість непарних елементів масиву A (10)
2	Обчислити середнє арифметичне елементів масиву T(15), що задовольняють умові 5<=T[i]<=15
3	Обчислити середнє геометричне парних і суму непарних елементів масиву З(10)
4	Знайти кількість елементів масиву B(16), кратних 4 і не більших заданого числа A
5	Знайти суму елементів одномірного масиву розміром 5. Розділити кожен елемент вихідного масиву на отримане значення. Результат зберегти в тому самому масиві. Надрукувати в одному рядку
6	Знайти середнє значення елементів заданого масиву розміром 6. Перетворити вихідний масив, віднімаючи з кожного елемента середнє значення
7	Обчислити довжину вектора X розміром 7
8	Визначити середнє значення елементів масиву. Потім знайти індекс елемента масиву, найбільш близького до середнього значення
9	Задано масив розміром 10. Якщо сума елементів виявиться понад 10, то знайти кількість парних елементів, інакше – добуток непарних
10	Задано масив розміром 10. Якщо добуток елементів виявиться понад 100, то знайти суму позитивних елементів, інакше – кількість негативних
11	Задано масив розміром 10. Якщо кількість парних елементів виявиться понад 5, то підрахувати кількість позитивних елементів, інакше – суму непарних
12	Визначити середнє значення елементів масиву. Потім підрахувати кількість елементів масиву, що перевищують середнє значення
13	Знайти номер найбільшого позитивного елемента масиву B(10)
14	Знайти різницю максимального і мінімального позитивних парних чисел масиву A(12)
15	Знайти суму квадратів максимального й мінімального чисел масиву B(14)

Продовження таблиці 1.10

1	*2*
16	Знайти максимальний і мінімальний елементи масиву B(20) і поміняти їх місцями
17	Знайти суму мінімального позитивного елемента масиву A(14) і його номера
18	У масиві B(10) поміняти місцями другий й найбільший позитивний елементи
19	Записати +1 замість максимального парного, а число –1 замість мінімального непарного елементів масиву A(10)
20	Знайти мінімальний елемент і його номер масиву B(10), значення елементів якого лежать в інтервалі від –10 до 10
21	У масиві з 10 чисел знайти найбільший елемент і поміняти його місцями з першим елементом
22	У масиві з 10 чисел знайти найменший елемент і поміняти його місцями з останнім елементом
23	Дано масив цілих чисел, що містить 15 елементів. Записати в новий масив спочатку всі негативні числа й нулі вихідного масиву, а потім – усі позитивні, зберігаючи порядок їхнього проходження
24	Задано масив розміром 10. Сформувати два масиви розміром 5, включаючи до першого елементи вихідного масиву з парними індексами, а до другого – з непарними
25	Задано масив розміром 10. Сформувати два нових масиви, включаючи до першого парні елементи вихідного масиву, а до другого – непарні
26	Задано масив розміром 10. Сформувати два нових масиви, включаючи до першого позитивні елементи вихідного масиву, а до другого – негативні
27	Обчислити середнє геометричне елементів масиву A(10), що задовольняють умові 5<=A[i]<=10

Приклади виконання завдання

1. Обчислити середнє арифметичне позитивних елементів масиву A розміром 10.

```
Program lab4_1;
Uses Crt;
Const n=10;
Type mas = array [1..n] of real;
Var a:mas; i,kol:integer; sum,sa:real;
Begin
    clrscr;
    writeln('Уведіть ',n:2,' елементів масиву:');
    for i:=1 to n do read(a[i]);
    writeln('Вихідний масив:');
```

```
    for i:=1 to n do write(a[i]:6:2);
    writeln;
    sum:=0; kol:=0;
    for i:=1 to n do if a[i] > 0 then
      begin
            sum := sum+a[i];
            kol:=kol + 1
      end;
    writeln('Сума позитивних = ',sum:6:2);
    writeln('Кількість позитивних = ',kol:2);
    if kol > 0 then begin sa := sum / kol;
        writeln('Середнє арифметичне = ',sa:6:2)
        end else writeln('Позитивних чисел немає.')
End.
```

2. Обчислити значення й порядковий номер найбільшого елемента масиву B (10).

```
Program lab4_2;
Uses Crt;
Const n=10;
Type mas = array [1..n] of real;
Var b:mas; i,imax:integer; bmax:real;
Begin
    clrscr;
    writeln('Уведіть ',n:2,' елементів масиву:');
    for i:=1 to n do read(b[i]);
    writeln('Вихідний масив:');
    for i:=1 to n do write(b[i]:6:2);
    writeln;
    imax:=1; bmax:=b[1];
    for i:=2 to n do if b[i] > bmax then
        begin
              bmax := b[i];
              imax := i
        end;
    writeln('Найбільший елемент масиву = ',bmax:6:2);
    writeln('Його номер = ',imax:2);
End.
```

3. У масиві B (10) поміняти місцями перший і найменший елементи.

```
Program lab4_3;
Uses Crt;
Const n=10;
Type mas = array [1..n] of real;
Var b:mas; i,imin:integer; bmin:real;
Begin
    clrscr;
    writeln('Уведіть ',n:2,' елементів масиву:');
    for i:=1 to n do read(b[i]);
    writeln('Вихідний масив:');
    for i:=1 to n do write(b[i]:6:2);
    writeln;
    imin:=1; bmin:=b[1];
    for i:=2 to n do if b[i] < bmin then
      begin
            bmin := b[i];
            imin := i
      end;
    writeln('Найменший елемент масиву = ',bmin:6:2);
    writeln('Його номер = ',imin:2);
    if imin <> 1 then
        begin
            b[imin]:=b[1];b[1]:=bmin;
            writeln('Новий масив :');
            for i:=1 to n do write(b[i]:6:2);
            writeln
        end else writeln('Перший елемент є мінімальним.')
End.
```

4. Сформувати масив, що складається з позитивних елементів масиву A (10).

```
Program lab4_4;
Uses Crt;
Const n=10;
Type mas = array [1..n] of integer;
Var a,b:mas; i,k:integer;
Begin
    clrscr;
    writeln('Уведіть ',n:2,' елементів масиву:');
    for i:=1 to n do read(a[i]);
    writeln('Вихідний масив:');
    for i:=1 to n do write(a[i]:3);writeln;
    k:=0;
    for i:=1 to n do if a[i] > 0 then
      begin
          k:=k+1;
          b[k]:=a[i]
      end;
```

```
    writeln('Новий масив :');
    for i:=1 to k do write(b[i]:3);
    writeln
End.
```

1.5 Підпрограми

Практично в кожній мові програмування передбачена можливість об'єднання послідовності операторів (як повторюваного в різних місцях програми фрагмента, так і просто логічно самостійного фрагмента) в окрему підпрограму, до якої можна багаторазово звертатися (викликати) з будь-якого місця основної програми, звільняючи останню від винесених у підпрограму фрагментів.

Звертання до підпрограми здійснюється за її ім'ям. При цьому виконання основної програми буде тимчасово припинено, виконаються всі дії підпрограми, і виконання основної програми знову буде продовжено, починаючи з оператора, що розташовний за оператором виклику підпрограми.

Таким чином, за ім'ям підпрограми ховаються деякі винесені із програми дії, які ініціюються автоматично з появою в програмі звертання до цієї підпрограми.

Будь-яка підпрограма, своєю чергою, може звертатися до інших підпрограм, а також до самої себе (явище рекурсії).

Структура підпрограми аналогічна структурі всієї програми, тобто в ній є заголовок, розділ описів і тіло підпрограми. Підпрограма повинна бути описана до того, як вона буде використана в програмі або іншій підпрограмі.

Усі змінні, використовувані в підпрограмі, можна поділити на три категорії:

– глобальні змінні, які описуються в основній програмі, можуть використатися як у програмі, так і у всіх її підпрограмах;

– локальні змінні, які описуються й використаються тільки усередині підпрограми;

– формальні параметри – зарезервовані в заголовку підпрограми змінні, до яких при звертанні до програми будуть занесені із зовнішньої програми конкретні дані або їхньої адреси (посилання).

У Паскалі є два різновиди підпрограм – процедури й функції.

У найпростішому випадку процедура може являти собою лише групу операторів:

```
Procedure Switch1;
begin
     r:=x;    x:=y;     y:=r
end;
```

Така процедура не має ні параметрів, ні локальних змінних. Усі використовувані в ній змінні (у нашому прикладі – x, y і r) є глобальними, тобто вони мають бути описані в зовнішній (основний) програмі. У прикладі процедура здійснює обмін значеннями змінних x і y. Ясно, що ці змінні мають бути доступні зовнішній програмі, тобто оголошення їх глобальними виправдано. А от змінна r – це робоча змінна, використовувана для тимчасового зберігання:

```
Procedure Switch2;
var  r: real;
begin
     r:=x;    x:=y;     y:=r
end;
```

Однак отримана процедура «жорстко» прив'язана до глобальних змінних x і y. Тому дані до процедури (а також із процедури) потрібно передавати як параметри:

```
Procedure Switch3 (var x,y: real);
var  r: real;
begin
     r:=x;    x:=y;     y:=r
end;
```

Приклад. Опишемо процедуру для уведення N цілих чисел. Нехай в основній програмі для масиву визначений тип mas:

```
Type mas=array [1..100] of integer;
Procedure input (var m: mas; n: integer);
{заголовок процедури}
var i: integer; {локальна змінна - параметр циклу}
begin
     writeln('уведіть', n:3, ' цілих чисел');
     for i:=1 to n do read(m[i]);
end;
```

Для виклику процедури в основній програмі або іншій підпрограмі треба вказати ім'я процедури з узятими в круглі дужки фактичними параметрами, які повинні збігатися за кількістю й типом з формальними параметрами процедури.

Виклик Input (m, 10) означає, що викликається процедура для уведення 10 цілих чисел до масиву M.

Функція призначена для обчислення якого-небудь значення. У цієї підпрограми дві основних відмінності від процедур:

– заголовок складається зі слова Function, за яким розташоване ім'я функції, список формальних параметрів і тип функції (тобто тип значення, що повертається);

– у тілі функції хоча б один раз її імені повинне бути привласнене значення (тобто значення, що повинне бути передане до основної програми як результат роботи функції).

Формальні параметри підпрограми вказують, з якими параметрами треба звертатися до цієї підпрограми. При звертанні до підпрограми формальні параметри заміщаються на еквівалентні фактичні параметри головної програми або підпрограми.

Усі формальні параметри можна поділити на дві категорії:

– параметри-значення (підпрограмою не змінюються);

– параметри-змінні (ці параметри підпрограма може змінити, і всі зміни передаються до основної програми).

Параметр-значення передається до підпрограми у вигляді свого значення й, отже, підпрограмою змінитися не може, точніше, усі зміни параметра-значення, виконані в підпрограмі, при виході з неї губляться й до основної програми не повертаються. Параметр-значення вказується в заголовку підпрограми своїм ім'ям і через двокрапку – типом. Якщо параметрів-значень одного типу декілька, їх можна перелічити через кому, а потім уже вказати загальний тип. Окремі групи параметрів відокремлюються один від одного крапкою з комою.

Як фактичний параметр на місці параметра-значення при виклику підпрограми може виступати будь-який вираз сумісного для присвоювання типу. Цей вираз обчислюється, і отримане значення передається підпрограмі.

Параметри-змінні передаються до підпрограми своїми адресами. Отже, підпрограма має доступ до цих параметрів і може їх змінювати. При виході з підпрограми нове значення параметра-змінної повертається основній програмі. Параметри-змінні вказуються в заголовку підпрограми аналогічно параметрам-значенням, але тільки перед ім'ям параметра записується слово Var. Дія Var поширюється до найближчої крапки з комою. При виклику підпрограми як фактичний параметр на місці параметра-змінної повинна використатися тільки змінна ідентичного типу. Використання виразу тут неприпустиме.

Наприклад, в описаній вище процедурі Input параметр M є параметром-змінною, а параметр N – параметром-значенням. Тому до цієї процедури припустимі такі записи:

Input (M, 10); Input (M, k+2); Input (A, k+SQR(m))

Лабораторна робота 5. Обробка різних масивів з використанням підпрограм

Завдання. У наведених в таблиці 1.11 завданнях уведення вихідних даних й їхнє контрольне виведення оформити як процедури, а обробку – у вигляді функції або процедури. Навести як мінімум два приклади.

Таблиця 1.11

Вар.	**Завдання**
1	*2*
0	Знайти кількості парних елементів у масивах A, B, C, а потім – їх середнє арифметичне
1	Обчислити відсоток парних елементів у масивах A, B, C. Визначити максимальний з них
2	Знайти кількості позитивних елементів у масивах A, B, C, а потім – їх середнє арифметичне
3	Визначити суму максимальних парних елементів масивів A, B, C
4	Знайти середнє геометричне непарних елементів кожного з масивів A, B, C. Визначити їхню суму
5	Знайти середнє арифметичне елементів головних діагоналей матриць X, B, Z і визначити найбільше з них
6	Підрахувати кількість точок, що перебувають усередині кола радіусом R = 2 і із центром на початку координат, координати точок увести до масивів X(20) і Y(20). Відстань від центра до точки обчислювати в підпрограмі-функції
7	Визначити максимальний з периметрів десяти трикутників, вершини яких A, B, C задані координатами (x, y) у масивах: AX(10), AY(10), BX(10), BY(10), CX(10), CY(10) відповідно. Периметр сторін трикутника обчислювати в підпрограмі-функції
8	Дано три матриці X, Y, Z. Роздрукувати ту з них, у якій більше нульових елементів
9	Дано три матриці A, B, C. Обчислити !!A!!+!!B!!+!!C!!, де !!X!! - максимальний за модулем елемент матриці
10	Довжини сторін 10 трикутників задані в масивах A[10], B[10], C[10]. Знайти суму довжин медіан кожного із трикутників і визначити максимальну з них (довжина медіани, проведеної до сторони **a**, дорівнює: 0.5 / 2b*b+2c*c–a*a)
11	У кожному з масивів A, B, C знайти максимальний за модулем елемент і відняти його з кожного елемента відповідного масиву
12	У кожній з матриць X,Y, Z знайти суми елементів, що лежать нижче головної діагоналі. Обчислити добуток отриманих значень
13	Дано три квадратні матриці x, y, z. Для кожної з них знайти суму елементів, що лежать вище головної діагоналі, визначити максимальну з них

Продовження таблиці 1.11

1	*2*
14	Задано три масиви A, B і C. Обчислити: $t = \begin{cases} \min(B) + \max(C), \text{ якщо } \min(A) < \max(B); \\ \max(A) + \min(B), \text{ у противному випадку.} \end{cases}$ Тут min(X) – мінімальний, а max(X) – максимальний елемент масиву X
15	Обчислити середнє арифметичне добутків позитивних, кратних 3, елементів масивів A, B, C
16	У кожній матриці X,Y, Z знайти номер стовпця, що містить найбільшу кількість позитивних елементів
17	У кожному масиві A, B, C знайти індекс максимального елемента й скласти ці індекси
18	Знайти максимальне значення середніх геометричних парних елементів масивів A, B, C
19	Знайти середню арифметичну кількість позитивних, кратних 3, елементів масивів A, B, C
20	Знайти найбільше значення серед сум позитивних елементів побічних діагоналей квадратних матриць X, Y, Z
21	Визначити кількість точок, що потрапили усередину верхньої частини кола радіусом R = 4 і із центром на початку координат. Координати точок увести до масивів X і Y
22	Дано три масиви A, B, C. Сформувати масиви A1, B1 і C1, що містять позитивні непарні елементи масивів A, B і C відповідно
23	У масивах A, B, C кожен елемент поділити на середнє арифметичне елементів масиву
24	У кожній із трьох матриць X, Y, Z знайти мінімальний елемент і поділити на нього кожен позитивний елемент матриці
25	Підрахувати кількості елементів у матрицях X, Y, Z, значення яких перебувають в інтервалі [–3, 15]
26	У кожній із трьох матриць X, Y, Z знайти максимальний елемент і поділити на нього кожен негативний елемент матриці
27	У кожній матриці X, Y, Z знайти номер рядка, що містить найменшу кількість негативних елементів

Приклад виконання завдання

Дано три масиви – A (15), B (10), C (7). Знайти суму їх максимальних парних елементів.

```
Program lab5;
Uses Crt;
Type mas = array [1..15] of integer;
Var a,b,c:mas; mc:integer;
```

```
Procedure InMas (Var m:mas; n:integer; mas_name:char);
Var i:integer;
Begin
    writeln('Уведіть масив ',
               mas_name, ' з ',n,' елементів:');
    for i:=1 to n do read(m[i])
End;
Procedure OutMas (m:mas; n:integer; mas_name:char);
Var i:integer;
Begin
    writeln('Масив ',mas_name, ':');
    for i:=1 to n do write(m[i],' ');
    writeln
End;
Function MaxChet (m:mas; n:integer):integer;
Var i,max:integer;
Begin
    max:=-maxint;
    for i:=1 to n do
     if (m[i] > max) and (m[i] mod 2 = 0) then max:=m[i];
    if max=-maxint then MaxChet:=0 else MaxChet:=max
End;
Begin
    clrscr;
    InMas(a,15,'A');
    InMas(b,10,'B');
    InMas(c,7,'C');
    OutMas(a,15,'A');
    OutMas (b,10,'B');
    OutMas (c,7,'C');
    mc:=MaxChet(a,15)+ MaxChet(b,10)+ MaxChet(c,7);
    writeln('Сума: ',mc:3);
End.
```

1.6 Оброблення символьних і рядкових даних

У мові Паскаль змінна типу Char (символьна змінна) може набувати значення з деякої впорядкованої сукупності символів комп'ютера (з послідовності ASCII-символів):

...<0<1<...<9<...<A<B<...<Z<...<a<b<c<...<z<....

…<А<Б<В...<Я<...<а<б<...<я<...

Кожному символу із цієї сукупності відповідає деякий код (так званий код ASCII) – порядковий номер символу в послідовності.

Для символьних даних визначені операції відносини =, <, >, < > і присвоєння := , а також функції, наведені ще в таблиці 1.3:

Ord(c) – повертає код символу c;

Chr(i) – повертає *i-й* символ послідовності ASCII, тобто за кодом *i* визначає символ;

Succ(i) і Pred(i) – повертають наступний або попередній (відповідно) символи в ASCII-послідовності.

Значенням символьної змінної (типу char) є тільки один символ.

Тип String (рядок) – послідовність символів довільної довжини (у Турбо Паскалі – не більше 255 символів), яку можна розглядати як одномірний масив символів array [0...n] of char. Відмінною рисою змінної цього типу є можливість працювати з нею як зі звичайною змінною, так і як з масивом символів:

```
var st: string;
…
for i:=1 to length(st) do if st[i]= '+' then st[i]:= '-'
```

При оголошенні типу String[n] визначається максимальна кількість символів у рядку. Якщо N не зазначено, приймається максимально можливе значення (у Турбо Паскалі – 255).

Перший байт рядка (індекс 0) містить поточну довжину рядка. Перший значущий символ рядка займає другий байт і має індекс 1.

До рядків можна застосовувати операції зчеплення (конкатенації) за допомогою символу «+». Рядки також можна порівнювати, причому рядки при цьому можуть мати різну довжину, а порівняння виконується зліва праворуч відповідно до кодів символів, уважається, що відсутній символ у більше короткому рядку має код менший, чим будь-який код дійсного символу. Наприклад: 'x' менше, ніж 'xs'.

При обробці рядкових даних використовуються такі процедури й функції:

1. Copy (St, N, K) – скопіювати (тобто створити новий рядок) з рядка ST довжиною K символів, починаючи із символу з номером N; якщо N більше довжини рядка ST, повертається порожній рядок.

2. Pos (St1, St) – знайти в рядку St перше входження підрядка St1 і повернути номер позиції, з якої він починається; якщо підрядок не знайдений, повертається нульове значення.

3. Length (St) – повернути поточну довжину рядка St.

4. Delete (St, N, K) – процедура, що видаляє з рядка St підрядок у K символів, починаючи із символу N.

5. Insert (St1, St, N) – процедура, що вставляє підрядок St1 у рядок St, починаючи із символу з номером N; якщо отриманий рядок містить понад 255 символів, усі символи після 255-го відкидаються.

6. Str (X, St) – процедура, що перетворює число x на рядкову змінну, результат заноситься до рядка ST.

7. Val (St, X, Code) – процедура, що перетворює символьну величину типу String з рядка на її чисельне значення й заносить результат до змінної X. Якщо рядок записаний з помилкою, то індекс (позиція) невірного символу заноситься до змінної Code, інакше Code дорівнює нулю.

Приклади використання підпрограм для роботи з рядками подані в таблиці 1.12. Припускаємо, що процедури й функції викликаються послідовно. Вихідний рядок: st:= 'abcdef123';

Таблиця 1.12 – Приклади використання підпрограм

Підпрограма	Результат
St2 := copy (st,2,4)	St2 → 'bcde'
K := pos ('cd',st)	K → 3
K := length (st)	K → 9
delete (st, 2, 4)	St → 'af123'
insert ('klm', St, 5)	St → 'af12klm3'
str (2.456, st)	St → '2.456'
val ('2.456', x, code)	Code → 0, x → 2.456
val ('2,456', x, code)	Code → 2, x не визначений

Лабораторна робота 6. Оброблення рядків символів

Завдання. Скласти програму, що вводить рядок символів, виконує його оброблення відповідно до таблиці 1.13 й потім виводить результати.

Таблиця 1.13

Вар.	Умова оброблення
1	*2*
0	Видалити всі символи – цифри
1	Видалити всі символи, що не є цифрами
2	Видалити парні цифри
3	Видалити всі символи від «I» до «N»
4	Видалити всі знаки «+» й «–»
5	Видалити всі літери «X» й «Y»
6	Видалити всі знаки «+», після яких знаходиться цифра
7	Видалити всі літери «B», після яких знаходиться літера «C»
8	Замінити все пари «AB» на «C»
9	Видалити всі символи, що не є латинськими літерами
10	Видалити знаки «+» й «–»
11	Підрахувати, скільки разів зустрічаються символи «+» й «–»
12	Замінити всі знаки оклику («!») на символ «*», а символ «крапка» («.») – крапками (три крапки «...»)

Продовження таблиці 1.13

1	*2*
13	Знайти позицію (номер першого символу) сполучення «МММ»
14	З'ясувати, чи є в рядку послідовність «ЧОТИРИ»
15	Визначити, чи входить до рядка літера «А», і підрахувати кількість пробілів
16	З'ясувати, чи входить до рядка пари символів, що знаходяться поряд, «АЛЕ» або «ВІН»
17	З'ясувати, є чи в рядку подвоєні символи (пари однакових символів, що знаходяться поряд) , надрукувати їх
18	З'ясувати, чи є в рядку пари символів – кома й двокрапка («, :»)
19	Видалити всі символи «+», а символи, що не є «+», подвоїти
20	Послідовності наступних один за одним пробілів замінити одним пробілом (тобто видалити всі пробіли, які знаходяться безпосередньо за пробілом)
21	Підрахувати загальну кількість входжень до рядка символів «А», «а», «В» й «в»
22	Видалити з рядка всі здвоєні, строєні тощо символи
23	Знайти позицію (номер символу), у якій знаходиться перша кома, і номер позиції з останньою комою
24	Видалити всі символи «*», а символи, що не є «*», подвоїти
25	Вставити пробіл після кожного символу «.» «,» «!» або «?», якщо за цими символами не розташовано пробілу (тобто будь-який символ, крім пробілу)
26	Замінити всі крапки (три крапки «...») одними крапками
27	Вставити після кожного символу крапки («.») один символ пробілу («_»), якщо після крапки немає пробілу

Приклади виконання завдання

1. Визначити, скільки разів у заданому рядку символів зустрічається словосполучення «SA».

```
Program lab6_1;
Uses crt;
Var st: string; i, k: integer; c: char;
Begin
    writeln('Уведіть рядок символів:');
    readln(st); k := 0;
    for i:=1 to length(st)-1 do
        if copy(st,i,2) = 'SA' then k := k+1;
writeln('Задане словосполучення зустрілося ',k,' раз.');
    c:=readkey
End.
```

2. Видалити із заданого рядка символів усі цифри.

```
Program lab6_2;
Uses crt;
Var st: string; i: integer; c:char;
Begin
    writeln('Уведіть рядок символів:');
    readln(st); i:=1;
    while i <= length(st) do
          if (st[i] >= '0') and (st[i] <= '9')
                          then delete(st,i,1)
                          else i := i + 1;
    writeln('Отриманий рядок:');
    writeln(st);
    c:=readkey
End.
```

1.7 Тип запис, файли й файлові типи даних

Запис – це структурований тип даних, що складається з фіксованої кількості компонент, які називають полями. В одному полі дані мають той самий тип, а в різних полях можуть мати різні типи.

У записі кожне поле має своє ім'я (ідентифікатор). Кількість полів, тип й ідентифікатор кожного поля фіксуються у визначенні типу, до якого ставиться запис. Визначення починається із ключового слова Record, за яким міститься перелічення всіх полів із вказівкою через двокрапку їхніх типів.

Поля відокремлюються один від одного крапкою з комою. Поля запису можуть бути будь-якого типу. Якщо кілька полів мають той самий тип, то їхні імена можна перелічувати через кому й потім указати загальний тип. Завершує визначення типу запису ключове слово End.

Пояснимо на конкретному прикладі. Необхідно зберігати інформацію про студентів (здобувачів освіти), при цьому запис про кожну людину містить прізвище, ім'я, по батькові, дату народження, групу й середній бал.

```
Type Date = Record
          Day, Month, Year: Integer
          End;
     Student = Record
          Fam, Name, Parent: String [20];
          DR: Date;
          Group: String [10];
          SR: Real
          End;
Var S: Student;
     SM: array [1..25] of Student;
```

Приклад також ілюструє можливість вкладеності записів: типом поля записи може бути інший запис.

Доступ до полів змінної типу запису здійснюється вказівкою імені змінної й імені поля, записуваного через крапку:

```
S.Fam:='Іванов'; S.DR.Day:=1;
```

Для того щоб не записувати щоразу ім'я змінної при звертанні до полів запису, можна використати оператор об'єднання With:

```
For i:=1 to 25 do
    With SM[i] do Begin
        Readln(Fam);Readln(Name);
        Readln(Parent);Readln(Group);Readln(SR);
    End;
```

Більш докладно тема записів (з виконанням лабораторної роботи) буде розкрита в рамках дисципліни «Алгоритми і структури даних».

Поняття файлу тісно пов'язане з розміщенням інформації на зовнішніх пристроях (магнітних дисках, флеш-накопичувачах і т. п.).

Файлом на комп'ютері називається іменована область пам'яті на внутрішньому або зовнішньому носії інформації. Файлом у мовах програмування називається послідовність компонентів одного типу. Компоненти файлу можуть бути будь-якого типу, за винятком типу файла. Кількість компонентів у файлі теоретично не обмежена.

Над файловими змінними не можна виконувати ніяких операцій (привласнювати значення, порівнювати й т. д.). Файлові змінні можна лише використовувати для читання інформації з файлу й запису інформації у файл. Загальний вид опису файлового типу:

```
Type T = File of Tk;
Var F: T ;
```

де T – ідентифікатор типу файлу;
Tk – тип компонента файлу;
F – змінна файлового типу.

Файл може бути заданий безпосередньо при описі змінної типу файл:

```
Var F: File of real;
```

У Паскалі введення-виведення інформації в зовнішній файл здійснюється тільки через файлові змінні. Перед тим, як здійснювати введення-виведення, файлова змінна повинна бути пов'язана з конкретним зовнішнім файлом за допомогою стандартної процедури Assign. Потім файл має бути відкритий для читання або запису. Після цього можна здійснювати організацію введення-виведення.

У кожен конкретний момент для оброблення доступний тільки один компонент файлу (поточний), тобто на цей компонент установлений «поточний покажчик» файлу.

Зазвичай усі файли вважаються файлами послідовного доступу. Це означає, що почати писати у файл потрібно тільки із самого його початку, дописуючи нові компоненти послідовно, один за одним. Для читання також треба переглядати файл із самого початку, однак за допомогою процедури Seek можна встановити режим довільного доступу, установлюючи покажчик поточної позиції файлу на необхідний компонент.

Для того щоб визначити готовність файлу до запису або до читання інформації, існує стандартна функція EOF(F). Якщо покажчик поточної позиції файлу перемістився за кінець файлу, то ця функція набуває значення True (істина), в інших випадках – False (неправда). Функцію EOF можна використати в умовному операторі If або в операторі циклу While:

```
while not Eof (F) do
begin
.....
     Read (F, A);
.....
end;
```

У цій конструкції здійснюється послідовне читання (і оброблення) компонентів файлу доти, поки покажчик поточної позиції вказує на черговий компонент (тобто у файлі є ще не оброблені компоненти). Цикл завершиться, як тільки покажчик поточної позиції переміститься за кінець файлу (тобто будуть оброблені все компоненти).

Після роботи з файлом його необхідно закрити за допомогою процедури Close.

Для роботи з файлами використаються такі стандартні процедури (скрізь мається на увазі, що F – ідентифікатор файлу).

1. Assign (F, St) – зв'язує файлову змінну F із зовнішнім файлом, ім'я якого задається в рядковому виразі (типу String) St. Ім'я файлу має відповідати вимогам операційної системи (для Турбо Паскаля – MS-DOS) і може містити маршрут (ім'я пристрою й імена каталогів). Усі подальші операції з F будуть діяти над зовнішнім файлом з ім'ям, заданим в St.

2. Rewrite (F) – створює й відкриває для запису новий файл з ім'ям, пов'язаним зі змінною F процедурою Assign. Якщо файл із таким ім'ям існує, він знищується, і замість нього створюється новий порожній файл. Якщо файл із таким ім'ям відкритий, він закривається, знищується й створюється наново. Покажчик поточної позиції файлу встановлюється в початок порожнього файлу (на ознаку кінця файлу), функції EOF(F) привласнюється значення True.

3. Write (F, A) – записує значення змінної A у файл F і переміщає покажчик поточної позиції файлу на наступний компонент.

4. Reset (F) – відкриває наявний файл з ім'ям, пов'язаним зі змінною F. Якщо файл не існує, видається повідомлення про помилку. Якщо файл уже відкритий, він спочатку закривається, а потім відкривається знову. Покажчик поточної позиції файлу встановлюється в початкове положення.

5. Read (F, A) – зчитує з файлу поточний компонент, привласнює його значення змінної A і переміщає покажчик поточної позиції файлу на наступний компонент. Якщо EOF (F) = True, то при спробі читання виникає помилка.

6. Seek (F, n) – переміщає покажчик поточної позиції файлу на компонент із номером n. Перший компонент файлу має номер 0. Для переміщення на кінець файлу можна використати процедуру Seek(F, FileSize(F)).

7. Close (F) – закриває відкритий файл, пов'язаний зі змінною F.

Також при роботі з файлами можна використати такі функції:

1. Eof (F) – повертає статус кінця файлу. Результат функції True (істина), якщо покажчик поточної позиції файлу перебуває на ознаці кінця файлу, тобто за останнім компонентом файлу, і False (неправда) – в протилежному разі.

2. FileSize (F) – повертає поточний розмір файлу, тобто кількість компонентів у файлі. Якщо файл порожній, FileSize (F)=0.

3. FilePos (F) – повертає поточне значення покажчика файлу. Якщо поточне значення покажчика відповідає початку файлу, FilePos повертає 0, якщо кінцю файлу (тобто EOF(F) =True), то FilePos (F) = FileSize (F).

При формуванні файлу (виведенні у файл) зазвичай використовується така послідовність стандартних процедур й операторів:

Assign – зв'язати файлову змінну із зовнішнім файлом.

Rewrite – відкрити файл для запису.

Write – вивести (у циклі) компоненти у файл.

Close – закрити файл.

Для читання з файлу зазвичай застосовується така послідовність:

Assign – зв'язати файлову змінну із зовнішнім файлом.

Reset – відкрити файл для читання.

Read – вивести (у циклі) компоненти з файлу.

Close – закрити файл.

Якщо в одній і тій самій програмі здійснюється й виведення у файл і читання з нього, то Assign досить записати один раз.

На практиці дуже часто компонентами файлу є записи. У цьому випадку файлову змінну можна описати, наприклад, у такий спосіб:

```
Var F: File of Student;
```

Крім типізованих файлів, у Паскалі використовуються ще два види файлів. Для виведення текстової інформації це текстові файли, виведення у які аналогічне виведенню на екран. Текстовий файл не дозволяє довільно встановити поточний покажчик або обчислити розмір файлу, але дозволяє використати процедуру Append (відкрити файл для додавання). Читання

з текстового файлу аналогічно читанню даних із клавіатури:

```
Var G: Text;
Begin
     Assign (G, 'Out.txt ');
     Reset (G);
     Append (G);
     …
     Writeln(G, 'x= ',x:6:2, ' y= ',y:6:2);
     …
     Close (G);
End.
```

Безтипові (нетипізовані) файли дозволяють записувати на диск довільні ділянки пам'яті комп'ютера й зчитувати їх з диска у пам'ять. Операції обміну з безтиповими файлами здійснюються за допомогою процедур BlockRead й BlockWrite. Крім того, уводиться розширена форма процедур Reset й Rewrite. В іншому принципи роботи залишаються такими самими, як і з компонентними файлами.

Перед використанням логічний файл

```
var f: File;
```

має бути пов'язаний з фізичним за допомогою процедури Assign. Далі файл має бути відкритий для читання або для запису процедурою Reset або Rewrite, а після закінчення роботи закритий процедурою Close.

При відкритті файлу довжина буфера встановлюється за замовчуванням у 128 байт. Турбо Паскаль дозволяє змінити розмір буфера введення-виведення, для чого варто відкривати файл розширеними параметрами процедур:

```
Reset (var F: File; BufSize: Word);
Rewrite (var F: File; BufSize: Word);
```

Параметр BufSize задає кількість байтів, зчитувальних з файлу або записуваних у нього за один обіг. Мінімальне значення BufSize – 1 байт, максимальне – 64 КБ (65536 байт).

Читання даних з бестипового файлу здійснюється процедурою

```
BlockRead (var F: File; var X; Count: Word; var QuantBlock: Word);
```

Ця процедура за один виклик здійснює читання в змінну X кількість блоків, задану параметром Count, при цьому довжина блоку дорівнює довжині буфера. Значення Count не може бути менше ніж 1. За один обіг не можна прочитати більше ніж 64 КБ.

Необов'язковий параметр QuantBlock повертає кількість блоків (буферів), прочитаних поточною операцією BlockRead. У випадку успішного завершення операції читання QuantBlock = Count, у випадку аварійної ситуації параметр QuantBlock буде містити кількість вдало прочитаних блоків. Зрозуміло, що за допомогою параметра QuantBlock можна контролювати правильність виконання операції читання.

Запис даних до бестипового файлу виконується процедурою

```
BlockWrite (var F: File; var X; Count: Word; var QuantBlock: Word);
```

Ця процедура здійснює за один обіг запис у файл зі змінної X кількості блоків, задану параметром Count, при цьому довжина блока дорівнює довжині буфера. Необов'язковий параметр QuantBlock повертає кількість блоків (буферів), записаних успішно поточною операцією BlockWrite.

Лабораторна робота 7. Робота із простими файлами

Завдання до варіантів 0–8. Сформувати файл із модулів цілих чисел, знайти умову з таблиці 1.14.

Таблиця 1.14

Вар.	Умова для оброблення
0	Суму компонентів файлу
1	Кількість парних чисел серед компонентів
2	Кількість непарних чисел серед компонентів
3	Суму квадратів непарних чисел
4	Суму квадратів парних чисел
5	Середнє арифметичне значення компонентів з парними номерами
6	Найбільше зі значень компонентів з парними номерами
7	Найменше зі значень компонентів з непарними номерами
8	Добуток квадратів компонентів

Завдання до варіантів 9–16. Прийнявши, що координати точок на площині задаються двома числами x й y, скласти програму, що зчитує із клавіатури координати точок і записує їх послідовно до файлу: спочатку x, а потім y. Після завершення уведення здійснюється перегляд файлу та його оброблення відповідно до таблиці 1.15.

Таблиця 1.15

Вар.	Завдання для оброблення
9	Підрахувати кількість точок, що попадають у коло радіуса 4 із центром на початку координат
10	Знайти суму відстаней кожної точки від центра координат
11	Підрахувати кількість точок, що потрапляють до прямокутника, утвореного осями координат і прямими x = 2 й в = 4
12	Знайти середнє відхилення (відстань) точок від осі OX
13	Знайти середнє відхилення (відстань) точок від центра координат
14	Підрахувати кількість точок, що лежать поза колом радіуса 2 з центром у точці (2, 2)
15	Знайти середнє відхилення (відстань) точок від осі OY
16	Підрахувати кількість точок, що лежать поза трикутником, утвореним осями координат і прямою y = 2x + 1

Завдання до варіантів 17–27. Сформувати файл із послідовності $(-1)^k * 0.3^k/(k+1)$. Знайти умову з таблиці 1.16.

Таблиця 1.16

Вар.	Умова для оброблення
17	Суму компонентів файлу
18	Добуток компонентів файлу
19	Суму квадратів компонентів файлу
20	Модуль суми компонентів файлу
21	Квадрат добутку компонентів файлу
22	Найбільший з компонентів файлу
23	Найбільший з компонентів з непарними номерами
24	Суму найбільшого й найменшого зі значень компонентів файлу
25	Середнє арифметичне модулів компонентів файлу
26	Квадрат максимального з компонентів файлу
27	Квадратний корінь із суми компонентів файлу

Приклад виконання завдання

Сформувати файл із квадратів цілих чисел; знайти суму парних чисел і кількість непарних чисел серед компонентів файлу. Ознакою кінця уведення інформації у файл вважати уведення числа «нуль».

```
Program lab7;
Var f: file of integer; x,s,k: integer; c: char;
Begin
    assign(f, 'lab7.dat');
    rewrite(f);
    writeln('Уведіть цілі числа. Ознака кінця - 0');
    repeat
        read(x);
        x := sqr(x);
        write(f,x)
    until x = 0;
    reset(f);
    writeln('Зміст файлу: ');
    s:=0; k:=0;
    while not eof(f) do
        begin
            read(f,x); write(x:4);
            if x mod 2 = 0 then s := s + x else k := k + 1
        end;
    close(f);
    writeln;
    writeln ('Сума парних чисел: ',s);
    writeln ('Кількість непарних чисел: ',k);
End.
```

1.8 Модулі

Щоб зробити основну програму більш зрозумілою для прочитання, звичайно рекомендується всі використовувані в ній підпрограми виносити в окремий файл, називаний модулем. Модуль має таку структуру:

```
Unit <ім'я>;
Interface
     <перший розділ описів>
Implementation
     <другий розділ описів>
     <розділ змісту>
Begin
     <секція ініціалізації>
End.
```

Unit – заголовок модуля, що повинен збігатися з ім'ям файлу модуля;

Interface – секція опису констант, змінних і заголовків підпрограм, які доступні як безпосередньо в модулі, так і програмам, що звертаються до нього;

Implementation – секція опису констант, змінних і заголовків підпрограм, які доступні тільки в межах модуля, а також тіла підпрограм, описаних у секції Interface.

Секція ініціалізації, що може бути відсутньою, містить оператори, що виконуються перед початком роботи головної програми. Якщо секція ініціалізації відсутня, то слово Begin також відсутнє.

Варто звернути увагу, що слова Interface й Implementation – це не оператори й не описи, тому вони не закінчуються крапками з комами, і в їхніх рядках не можуть розміщатися інші оператори або описи програми.

Підключення модуля здійснюється командою Uses <ім'я>:

```
Uses Unit1, Unit2, Unit3;
```

Якщо в інтерфейсних секціях двох модулів виявляться однойменні ідентифікатори, то за замовчуванням буде використаний ідентифікатор з модуля, наведеного в списку пізніше (праворуч). Припустимо запис через крапку: «ім'я модуля . ім'я ідентифікатора».

Бібліотеки підпрограм Free Pascal містять кілька убудованих модулів для роботи з екраном як у текстовому, так й у графічному режимах. Наприклад, при використанні модуля Crt (після заголовка програми необхідно написати «uses crt») можна викликати процедуру очищення екрана (clrscr), затримки виконання програми до натискання будь-якої клавіші (readkey) або задана кількість мілісекунд (delay).

Модулі для роботи із графікою Graph (у Free Pascal) і GraphABC (в ABC-Паскалі) містять низку процедур і функцій, що дозволяють малювати на екрані графічні об'єкти. Частина з них наведена в таблиці 1.17.

Необхідно пам'ятати, що будь-яка робота із графікою в Free Pascal починається з ініціалізації відповідного режиму процедурою InitGraph(Gd,Gm,Path), де Gd,Gm:integer (номера режимів), Path:string (шлях до графічних драйверів), а закінчується процедурою CloseGraph. Лівий верхній кут екрана має координати (0,0), правий нижній – (639,479) або (799,599). Водночас в ABC-Паскалі необхідність у відкритті й закритті графічного режиму відсутня.

Повний перелік підпрограм модуля Graph (GraphABC) можна одержати, нажавши Ctrl+F1 на кожному з набраних імен.

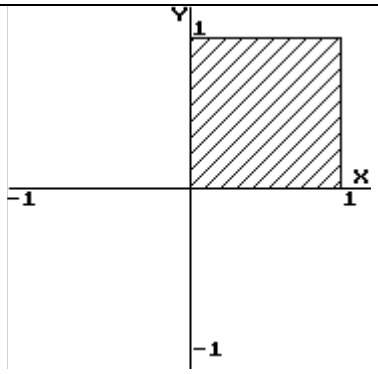
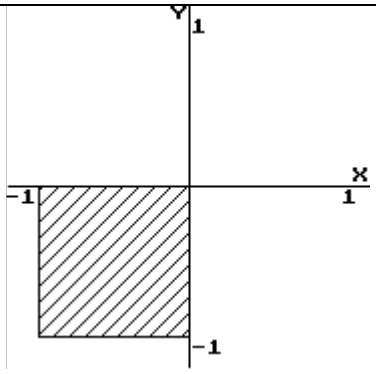
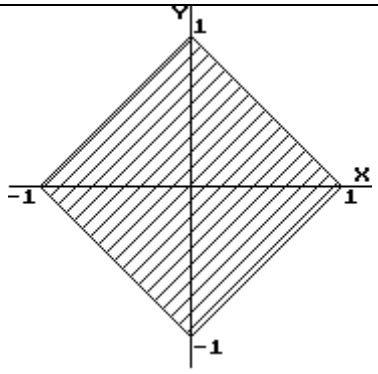
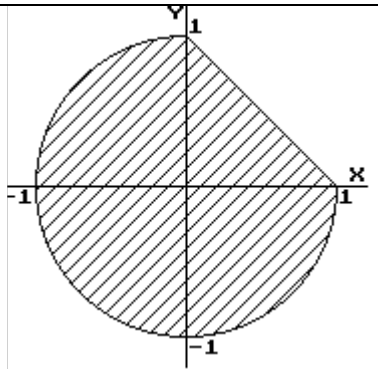
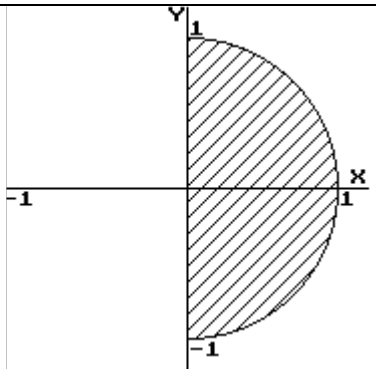
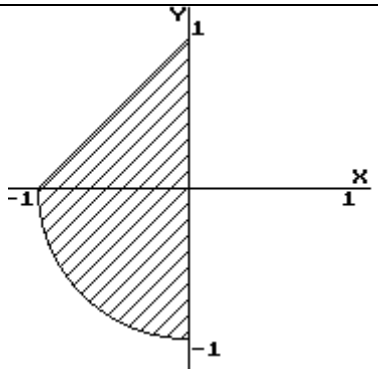
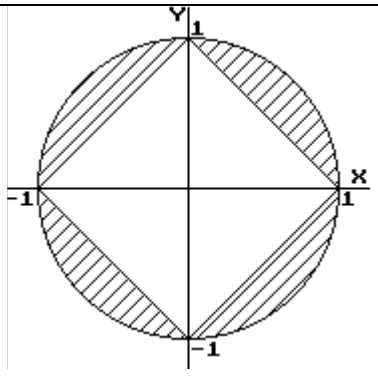
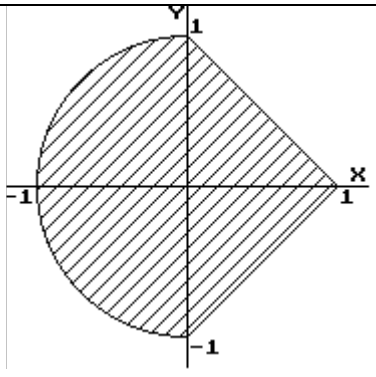
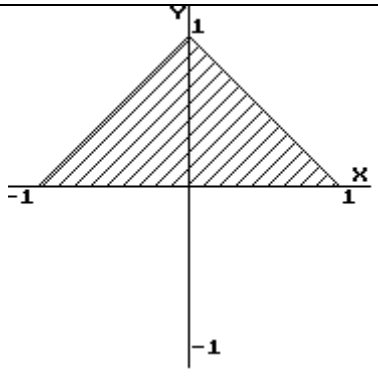
Таблиця 1.17 – Деякі процедури й функції модулів Graph й GraphABC

Назва в Free	**Назва в ABC**	**Опис**
InitGraph(Gd,Gm, '')	Немає	Ініціалізація граф. режиму
Rectangle (x1,y1,x2,y2)	DrawRectangle (x1,y1,x2,y2)	Прямокутник (задані координати двох вершин)
Bar(x1,y1,x2,y2)	FillRectangle (x1,y1,x2,y2)	Зафарбований прямокутник
Circle(x,y,r)		Окружність з центром у (x,y) і радіусом r
Arc(x,y,a1,a2,r)	Arc(x,y,r,a1,a2)	Дуга окружності від кута a1 до a2, причому кути задаються в градусах від 0 до 360
PieSlice(x,y,a1,a2,r)	FillPie (x,y,r,a1,a2)	Зафарбована дуга окружності
SetFillStyle(t,c)	SetBrushHatch(bh) SetBrushStyle(bs)	Встановити заливання номером t (типом bh, bs) і кольорами c
Var fills:array[1..4] of PointType; FillPoly (SizeOf(fills) div SizeOf(PointType), fills)	Var fills: array of Point; FillPolygon (fills)	Заливання ділянки попередньо заданим типом заливання
Line(x1,y1,x2,y2)		Пряма лінія із точки (x1,y1) до (x2,y2)
LineTo(x2,y2)		Пряма лінія з поточного місця до точки (x2,y2)
MoveTo(x2,y2)		Перенести поточний покажчик до точки (x2,y2)
PutPixel(x,y,color)		У точці (x,y) намалювати точку кольорами color (0..7)
C:=GetPixel(x,y)		Номер кольорів у заданій точці
SetColor(c)	SetPenColor(c)	Установити кольори ліній
C:=GetColor	C:=PenColor	Прочитати кольори ліній
SetBkColor(c)	SetBrushColor(c)	Установити кольори фону
C:=GetBkColor	C:=BrushColor	Прочитати кольори фону
C:=GetMaxColor		Номер білого кольору
X:=GetMaxX		Максимальна координата за X
Y:=GetMaxY		Максимальна координата за Y
OutText(st)	Немає	Виведення рядка в поточне місце
OutTextXY(x,y,st)	TextOut(x,y,st)	Виведення рядка в (x,y)
CloseGraph	Немає	Закриття графічного режиму

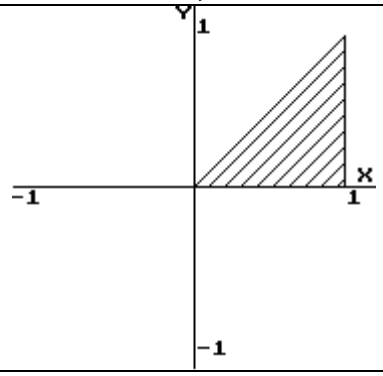
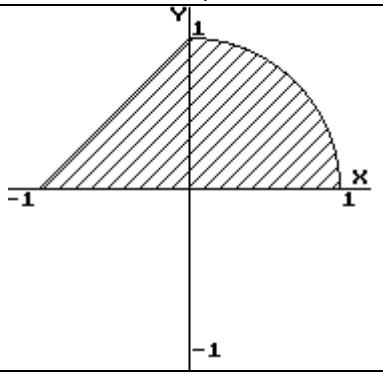
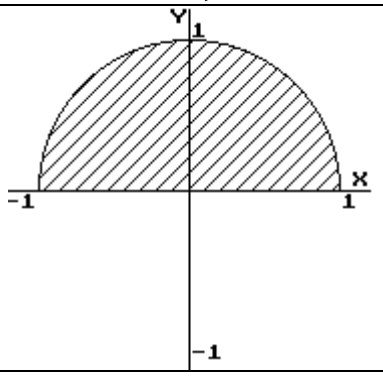
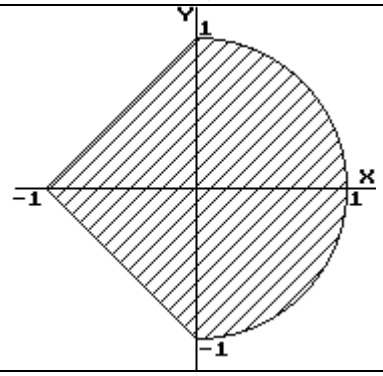
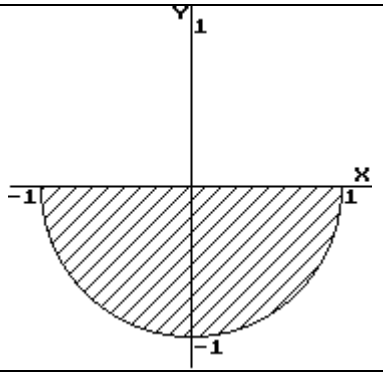
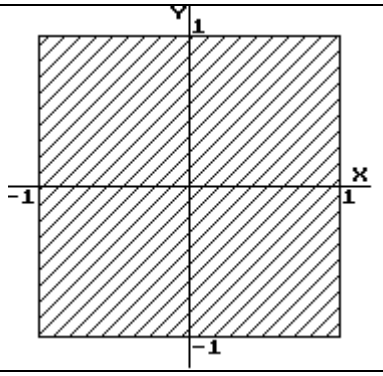
Лабораторна робота 8. Побудова графічних зображень

Завдання. З використанням модуля роботи із графікою скласти програму для малювання на екрані трьох зображень із таблиці 1.18 відповідно до варіанта (наприклад, якщо ваш варіант № 11, то необхідно намалювати зображення № 11, 12 й 13).

Таблиця 1.18

№ 0, 15	№ 1, 16	№ 2, 17
		
№ 3, 18	№ 4, 19	№ 5, 20
		
№ 6, 21	№ 7, 22	№ 8, 23
		

Продовження таблиці 1.18

№ 9, 24	№ 10, 25	№ 11, 26
		
№ 12, 27	№ 13	№ 14
		

Приклад виконання завдання

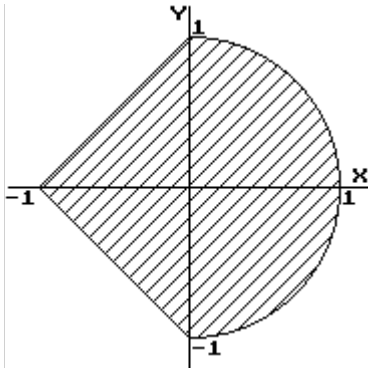


Рисунок 1.1

У середовищі Free Паскаля:

```
Program graphics;
Uses graph;
Var gd,gm,xc,yc:integer;
    fills:array[1..3] of PointType;
begin
     gd:=detect;initgraph(gd,gm,'');
     if graphresult <> 0 then halt(1);
     xc:=100;yc:=100;
     SetFillStyle(3,GetMaxColor);
```

```
     pieslice(xc,yc,270,360,75);
     pieslice(xc,yc,0,90,75);
     fills[1].x:=xc;fills[1].y:=yc-75;
     fills[2].x:=xc-75;fills[2].y:=yc;
     fills[3].x:=xc;fills[3].y:=yc+75;
FillPoly(SizeOf(fills) div SizeOf(PointType), fills);
     line(xc-90,yc,xc+90,yc);line(xc,yc-90,xc,yc+90);
     line(xc,yc-75,xc-75,yc);line(xc-75,yc,xc,yc+75);
     moveto(xc-75-16,yc+2);outtext('-1');
     moveto(xc+75+2,yc+2);outtext('1');
     moveto(xc+2,yc-75-8);outtext('1');
     moveto(xc+2,yc+75+2);outtext('-1');
     moveto(xc+90-8,yc-9);outtext('X');
     moveto(xc-9,yc-90);outtext('Y');
     readln;cleardevice;
     closegraph
End.
```

У середовищі АВС-Паскаля:

```
Program graphics;
Uses graphabc;
Var xc,yc:integer;
    fills: array of Point;
Begin
     xc:=100;yc:=100;
     SetPenColor(clBlack);
     SetBrushStyle(bsHatch);
     SetBrushHatch(bhDiagonalCross);
     DrawPie(xc,yc,75,270,360);
     DrawPie(xc,yc,75,0,90);
     FillPie(xc,yc,75,270,360);
     FillPie(xc,yc,75,0,90);
     SetLength(fills,3);
     fills[0].x:=xc;fills[0].y:=yc-75;
     fills[1].x:=xc-75;fills[1].y:=yc;
     fills[2].x:=xc;fills[2].y:=yc+75;
     FillPolyGon(fills);
     SetBrushStyle(bsSolid);
     line(xc-90,yc,xc+90,yc);line(xc,yc-90,xc,yc+90);
     line(xc,yc-75,xc-75,yc);line(xc-75,yc,xc,yc+75);
     textout(xc-75-16,yc+2,'-1');
     textout(xc+75+2,yc+2,'1');
     textout(xc+2,yc-75-8,'1');
     textout(xc+2,yc+75+2,'-1');
     textout(xc+90-8,yc-9,'X');
     textout(xc-9,yc-90,'Y');
End.
```

1.9 Базові положення ООП й їхнє використання в Паскалі

Завдання даних разом із процедурами й функціями їхнього оброблення перетворює їх на новий тип даних – об'єкт. Структурно об'єкт нагадує тип запис, полями якого можуть бути підпрограми. При цьому поля об'єкта називаються властивостями, а процедури й функції для їхнього оброблення – методами.

Як приклад опишемо об'єкт, що дозволяє відкрити й закрити графічний режим (у середовищі Free Pascal).

```
program test1;
uses graph;
type gr=object
     gd,gm:integer;
     procedure graphinit;
     procedure graphclose;
end;
procedure gr.graphinit;
begin
     gd:=detect;initgraph(gd,gm,'');
     if graphresult <> 0 then begin
     writeln('Неможливо відкрити графічний режим!');
     halt(1) end
end;
procedure gr.graphclose;
begin
     closegraph
end;
var p:gr;
{p – це змінна типу object, або екземпляр об'єкта}
{програма просто малює точку в центрі екрана}
begin
     with p do begin
     graphinit;
     putpixel(round(getmaxx/2), round(getmaxy/2),7);
     readln; {очікування натискання клавіші Enter}
     graphclose
     end;
end.
```

Принципи об'єктно-орієнтованого програмування дозволяють використати будь-який об'єкт для породження ієрархії «об'єктів-нащадків» зі спадкуванням доступу кожного з них до коду й даних предка. Тобто, наприклад, для малювання точки в центрі екрана ми можемо перетворити описану вище програму шляхом внесення нового об'єкта як нащадка об'єкта gr:

```
program test2;
uses graph;
type gr=object
     gd,gm:integer;
     procedure graphinit;
     procedure graphclose;
end;
procedure gr.graphinit;
begin
     gd:=detect;initgraph(gd,gm,'');
     if graphresult <> 0 then halt(1)
end;
procedure gr.graphclose;
begin
     closegraph
end;
point=object(gr)
     x,y:integer; {координати точки}
     procedure setd(px,py:integer); {завдання координат}
     procedure show; {намалювати точку}
     procedure hide; {сховати - намалювати кольорами фону}
     end;
procedure point.setd(px,py:integer);
begin
     x:=px;y:=py;
end;
procedure point.show;
begin
     putpixel(x,y,getmaxcolor);
end;
procedure point.hide;
begin
     putpixel(x,y,getbkcolor);
end;
var p:point; {p - це екземпляр об'єкта point}
{програма малює точку в центрі екрана, а потім видаляє}
begin
     with p do begin
     graphinit; {метод використається як свій власний}
     setd(round(getmaxx/2), round(getmaxy/2));
          {задаємо координати точки}
     show; readln;
     {малюємо точку й чекаємо натискання клавіші Enter}
     hide; readln;
{приховуємо точку й знову чекаємо натискання клавіші Enter}
     graphclose
     end;
end.
```

Лабораторна робота 9. Переміщення графічних об'єктів

Завдання. Скласти програму, що виконує дії, описувані в індивідуальному завданні (табл. 1.19). Програма повинна складатися як мінімум із двох частин – основної програми й модуля з описом деяких об'єктів – і працювати на основі технології об'єктно-орієнтованого програмування.

Таблиця 1.19

Вар.	**Зміст**
1	*2*
0	Намалювати в центрі екрана десять концентричних окружностей
1	Намалювати в центрі екрана десять вкладених один в одного прямокутників
2	Переробити програму «кіл на воді» (див. приклад) так, щоб кола спочатку розходилися з однієї точки, а потім до неї сходилися
3	Переміщати коло за горизонталлю із заданим кроком і затримкою в одну секунду
4	Переміщати коло за вертикаллю із заданим кроком і затримкою в одну секунду
5	Переміщати прямокутник за горизонталлю із заданим кроком й затримкою в одну секунду
6	Переміщати прямокутник за вертикаллю із заданим кроком і затримкою в одну секунду
7	Переробити варіант номер 2, замінивши кола на прямокутники.
8	Переміщати коло за діагоналлю із заданим кроком і затримкою в півтори секунди
9	Переміщати прямокутник за діагоналлю із заданим кроком і затримкою в півтори секунди
10	Розмістити на екрані дві розташовані поруч «» окружності, що переморгуються
11	Переробити варіант номер 3 для кола, уписаного в прямокутник
12	Переробити варіант номер 4 для кола, уписаного в прямокутник
13	Переробити варіант номер 8 для кола, уписаного в прямокутник
14	Переміщати коло за периметром розташованого в центрі екрана прямокутника
15	Переміщати коло по вершинах розташованого в центрі екрана прямокутника
16	Намалювати трикутник довільної форми й переміщати коло за його периметром
17	Намалювати трикутник довільної форми й переміщати коло по його вершинах
18	Переробити варіант номер 3 для трикутника
19	Переробити варіант номер 4 для трикутника

Продовження таблиці 1.19

1	*2*
20	Переробити варіант номер 8 для трикутника
21	Переміщати коло екраном випадковим образом (функція random)
22	Переміщати прямокутник екраном випадковим образом (функція random)
23	Переміщати трикутник екраном випадковим образом (функція random)
24	Зробити так, щоб на екрані по черзі виникали й зникали коло, прямокутник і трикутник
25	Намалювати в центрі екрана десять концентричних окружностей
26	Намалювати в центрі екрана десять вкладених один в одного прямокутників
27	Переміщати прямокутник за діагоналлю із заданим кроком і затримкою в півтори секунди

Приклад виконання завдання

Написати програму зображення на екрані розбіжних концентричних окружностей («кіл на воді»).

```
unit obj_un; {Модуль}
Interface
uses graph;
type gr=object
     gd,gm:integer;
     procedure graphinit;
     procedure graphclose;
     end;
     point=object(gr)
     x,y:integer;
     procedure setd(px,py:integer);
     procedure show;
     procedure hide;
     end;
Implementation
procedure gr.graphinit;
begin
     gd:=detect;initgraph(gd,gm,'');
     if graphresult <> 0 then begin
     writeln('Неможливо відкрити графічний режим!');
      halt(1) end
end;
procedure gr.graphclose;
begin
     closegraph
end;
```

```
procedure point.setd(px,py:integer);
begin
     x:=px;y:=py;
end;
procedure point.show;
begin
     putpixel(x,y,getmaxcolor);
end;
procedure point.hide;
begin
     putpixel(x,y,getbkcolor);
end;
end.

Program lab9; {Програма}
uses crt,graph,obj_un;
type krug=object(point)
     r:integer;
     procedure setd(px,py,rad:integer);
     procedure show;
     procedure hide;
     end;
procedure krug.setd(px,py,rad:integer);
begin
     r:=rad;point.setd(px,py)
end;
procedure krug.show;
begin
     setcolor(getmaxcolor);circle(x,y,r);point.show
end;
procedure krug.hide;
begin
     setcolor(getbkcolor);circle(x,y,r);point.hide
end;
var p:krug;i,k:integer;
begin
     with p do begin graphinit;
     for i:=1 to 10 do begin setd(100+i*20,100,1);show;
     for k:=1 to 10 do begin
          hide;setd(100+i*20,100,k*10);show;
          delay(1000) {затримка в 1 секунду}  end
     end;
     readln; {очікування натискання Enter}
     graphclose
     end;
end.
```

При реалізації завдання в середовищі ABC-Паскаля необхідно видалити об'єкт gr (ініціалізація графічного режиму) і по-іншому задавати параметри деяких процедур і функцій.

1.10 Завдання для самостійної роботи

Задача 1
Оператори присвоювання, уведення й виведення інформації

Таблиця 1.20

Вар.	**Завдання**
1	*2*
1	Складіть програму для рішення системи двох лінійних рівнянь: $\begin{cases} ax+by=c, \\ dx+ey=f. \end{cases}$ з двома невідомими *x, в.* Значення невідомих знаходять за формулами: $\Delta = ae-bd,\ x=\frac{ce-bf}{\Delta},\ y=\frac{af-cd}{\Delta}$. Вважати $\Delta \neq 0$
2	Підрахуйте, скільки очок набрала команда «Донбас» у першому колі чемпіонату з хокею, якщо відомо, що *m* зустрічей вона виграла, *n* зустрічей програла, *k* зустрічей закінчилися нічиєю, якщо за виграш команда одержує 2 очки, за нічию – 1 очко, за програш – 0 очок
3	Нехай відомі довжини сторін *a, b, c* трикутника. Обчислите висоти цього трикутника за формулами: $h_a = \frac{2}{a}\sqrt{p(p-a)(p-b)(p-c),}$ $h_b = \frac{2}{b}\sqrt{p(p-a)(p-b)(p-c),}$ $h_c = \frac{2}{c}\sqrt{p(p-a)(p-b)(p-c),}$ *где* $p=\frac{a+b+c}{2}$.

Продовження таблиці 1.20

1	*2*
4	Знайдіть x із пропорції $\frac{a+b}{x}=\frac{b-c}{a+c}$.
5	Скільки відсотків від A + B – C припадає на A? B? C?
6	Складіть програму обчислення ідеальної ваги людини за її зростом за умови, що *Ідеальна вага (кг) = Ріст (см) – 100*
7	Складіть програму для обчислення часу t зустрічі автомобілів, що рухаються рівноприскорено назустріч один одному, якщо відомі їхні швидкості V_1 й V_2, прискорення a_1 й a_2 і початкова відстань S між ними. Відстань S_1, пройдена першим автомобілем, обчислюється за формулою $S_1 = V_1 t + \frac{a_1 t^2}{2}$; відстань S_2, пройдена другим автомобілем, обчислюється за формулою $S_2 = V_2 t + \frac{a_2 t^2}{2}$. Час t зустрічі автомобілів визначається з рівняння $V_1 t + \frac{a_1 t^2}{2} = S - \left(V_2 t + \frac{a_2 t^2}{2} \right)$, *звідки* $t = \frac{-(V_1+V_2) + \sqrt{(V_1+V_2)^2 + (a_1+a_2)2S}}{a_1+a_2}$.
8	Ви поклали гроші в банк на терміновий депозит на квартал з розрахунку 12 % річних. Складіть програму, що обчислить суму, яка вам належить через 4 місяці
9	Роздрібна ціна чоловічого костюма становить R гривень. Торговельна націнка магазину становить T % від оптової ціни. Складіть програму визначення оптової ціни костюма
10	Зарплата співробітника приватної фірми становить r гривень на місяць. Скільки грошей він одержить за пів року після відрахування податків у розмірі t % щомісяця й s % за пів року?
11	Дано координати вершин деякого трикутника. Обчислите його периметр
12	Нехай змішано V_1 літрів води температури t_1 з V_2 літрами води температури t_2. Складіть програму обчислення об'єму й температури утвореної суміші
13	Визначите вартість набору, до якого входять такі цукерки (вартість упакування становить U грн): **Назва** **Вага** **Вартість 1 кг** Петровські 200 г K грн Львівські 300 г P грн Чарівниця 250 г R грн Факел 150 г B грн Ластівка 200 г L грн

Продовження таблиці 1.20

1	*2*
14	Скільки часу у хвилинах знадобиться школяру на дорогу від школи до стадіону, якщо ця відстань становить *S* км, а середня швидкість руху школяра – V км/год?
15	Підрахуйте, скільки очок набрала команда «Шахтар» у першому колі чемпіонату з футболу, якщо відомо, що *m* зустрічей вона виграла, *n* зустрічей програла, *k* зустрічей закінчилися нічиєю, якщо за виграш команда одержує 3 очки, за нічию – 1 очко, за програш – 0 очок
16	Скільки відсотків від A – B + C припадає на A? B? C?
17	Дано координати вершин деякого трикутника. Обчислите його площу.
18	Дано координати вершин деякого трикутника. Обчислите висоти цього трикутника за формулами, наведеним у варіанті 3.
19	Ви поклали гроші в банк на терміновий депозит на півроку з розрахунку 12 % річних. Складіть програму, що обчислить належну вам суму через 6 місяців
20	Обчислите довжину окружності, площа кола, об'єм кулі заданого радіуса
21	Дано натуральне число *n*, що складається із шести цифр. Визначте кількість сотень і тисяч у ньому
22	Дано число *f* – кут у градусах. Визначте суміжний до нього кут у радіанах
23	Обчислите відстань між двома точками на площині з координатами (x_1, y_1) і (x_2, y_2)
24	Визначте число, отримане виписуванням у зворотному порядку цифр заданого цілого тризначного числа *x*. Привласніть це число змінній *m*
25	Поміняйте місцями значення речовинних змінних *x* й *y*

Задача 2
Умовний оператор

Таблиця 1.21

Вар.	**Завдання**
1	*2*
1	Уведіть три числа. Якщо вони можуть бути довжинами сторін прямокутного трикутника, виведіть їх у порядку зростання й обчислите площу отриманого трикутника
2	Уведіть три числа. Якщо вони можуть бути довжинами сторін гострокутного трикутника, виведіть їх у порядку убування й обчислите площу отриманого трикутника

Продовження таблиці 1.21

1	*2*
3	Уведіть три числа. Якщо вони можуть бути довжинами сторін тупокутного трикутника, виведіть їх у порядку убування й обчислите площу отриманого трикутника
4	Уведіть три числа. Якщо вони можуть бути сторонами рівностороннього трикутника, обчислите його площу й висоту. Виведіть сторони, площу й висоту в порядку зростання
5	Уведіть три числа. Якщо вони можуть бути довжинами сторін рівнобедреного трикутника, обчислите його висоти. Виведіть довжину основи й висоти в порядку зростання
6	Уведіть три числа. Якщо вони можуть бути довжинами сторін різностороннього тупокутного трикутника, виведіть їх у порядку зростання й обчислите площу отриманого трикутника
7	Уведіть три числа. Якщо вони можуть бути довжинами сторін рівнобедреного тупокутного трикутника, обчислите його площу. Виведіть довжини сторін і площу в порядку зростання значень
8	Уведіть три числа. Якщо вони можуть бути довжинами сторін рівнобедреного гострокутного трикутника, обчислите його площу. Виведіть довжини сторін і площу в порядку зростання значень
9	Уведіть три числа. Якщо вони можуть бути довжинами сторін різностороннього гострокутного трикутника, виведіть їх у порядку зростання й обчислите площу отриманого трикутника
10	Нехай є координати трьох точок на площині. Якщо вони можуть бути вершинами прямокутного трикутника, обчислите його площу
11	Нехай є координати трьох точок на площині. Якщо вони можуть бути вершинами гострокутного трикутника, обчислите його площу
12	Нехай є координати трьох точок на площині. Якщо вони можуть бути вершинами тупокутного трикутника, обчислите його площу. Виведіть довжини сторін у порядку убування
13	Нехай є координати трьох точок на площині. Якщо вони можуть бути вершинами рівностороннього трикутника, обчислите його площу й висоту. Виведіть довжини сторін, площу й висоту в порядку зростання значень
14	Нехай є координати трьох точок на площині. Якщо вони можуть бути вершинами рівнобедреного трикутника, обчислите його висоти. Виведіть довжини основи й висот у порядку зростання значень
15	Нехай є координати трьох точок на площині. Якщо вони можуть бути вершинами різностороннього тупокутного трикутника, обчислите його площу
16	Нехай є координати трьох точок на площині. Якщо вони можуть бути вершинами рівнобедреного тупокутного трикутника, обчислите його площу. Виведіть довжини сторін і площу в порядку зростання значень

Продовження таблиці 1.21

1	*2*
17	Нехай є координати трьох точок на площині. Якщо вони можуть бути вершинами рівнобедреного гострокутного трикутника, обчислите його площу. Виведіть довжини сторін і площу в порядку зростання значень
18	Нехай є координати трьох точок на площині. Якщо вони можуть бути вершинами різностороннього гострокутного трикутника, обчислите його площу
19	Нехай є три числа. Якщо вони можуть бути довжинами сторін трикутника, визначте його вид (різносторонній, рівнобедрений, рівносторонній). Обчислите його висоти й надрукуйте їх у порядку убування
20	Нехай є три числа. Якщо вони можуть бути довжинами сторін трикутника, визначите його вид (прямокутний, тупокутний, гострокутний). Обчислите його висоти й надрукуйте їх у порядку убування
21	Нехай є координати трьох точок на площині. Якщо вони можуть бути вершинами трикутника, визначте його вид (різносторонній, рівнобедрений, рівносторонній). Обчислите його висоти й надрукуйте їх у порядку убування
22	Нехай є координати трьох точок на площині. Якщо вони можуть бути вершинами трикутника, визначте його вид (прямокутний, тупокутний, гострокутний). Обчислите його висоти й надрукуйте їх у порядку убування
23	Складіть програму, що визначала б вид трикутника (рівносторонній, рівнобедрений, різносторонній, прямокутний, тупокутний, гострокутний), якщо за трьома відрізками його можна побудувати
24	Нехай є координати трьох точок на площині. Складіть програму, що визначала б вид трикутника (рівносторонній, рівнобедрений, різносторонній, прямокутний, тупокутний, гострокутний), якщо дані координати вершин дозволяють його побудувати
25	Нехай є координати вершин чотирикутника. Складіть програму, що визначала б, чи є цей чотирикутник прямокутником

Задача 3
Оператори циклу

Таблиця 1.22

Вар.	**Завдання**
1	*2*
1	Нехай $a_0 = a_1 = 1; a_k = a_{k-1} + \frac{a_{k-1}}{2^{k-1}}$, *где* $k = 2,3,...$ Знайдіть добуток $a_0 * a_1 * \ldots * a_n$

Продовження таблиці 1.22

1	*2*
2	Обчислите довжину кривої, що відповідає функції $y = f(x)$ на відрізку $[a,b]$, приблизно замінивши криву ламаної, отриманої в результаті поділу відрізка $[a,b]$ на n рівних частин
3	Нехай є речовинні числа a, b $(b > a)$, натуральне n. Отримати $(f_1 + f_2 + \ldots + f_n)*h$, де $h = \frac{b-a}{n}$, $f_i = \frac{a + (i - 0{,}5)h}{1 + (a + (i - 0{,}5)h)^2}, i = 1,2,..., n$
4	Нехай дане ціле число $m > 1$. Отримати найбільше ціле k, при якому $4^k < m$
5	Визначите, до якого найбільшого цілого позитивного степеня можна піднести число b, щоб результат не перевершував заданої величини a $(b < a)$
6	Нехай дане речовинне число x і натуральне число n. Обчисліть: $\frac{(x-2)(x-4)...(x-2n)}{(x-1)(x-3)...(x-2n-1)}$
7	Нехай є речовинні числа a, h, натуральне число n. Обчисліть: $f(a)+f(a+h)+f(a+2h)+...+f(a+nh)$, де $f(x)=(x^2+1)\cos x$.
8	Корінь деякого рівняння знаходиться послідовними наближеннями за формулою $x_{n+1} = \frac{2 - x_n^3}{5}$. Напишіть програму для знаходження такого наближення кореня, при якому різниця за модулем між двома сусідніми наближеннями не перевершує 10^{-5}, а початкове наближення $X_0 = 1$
9	Сторона правильного вписаного багатокутника з подвоєною кількістю сторін виражається через сторону початкового багатокутника a_n, а радіус описаної окружності R у вигляді формули $a_{2n} = \sqrt{2R^2 - 2R\sqrt{R^2 - \frac{a_n^2}{4}}}$. Обчисліть довжину сторони правильного вписаного 48-кутника
10	Циліндр об'єму 1 має висоту h. Визначте радіус основи циліндра для значень h, які дорівнюють 0,5; 1; 1,5; ...; 4,5; 5
11	Спосіб послідовних наближень дозволяє знаходити корінь п'ятого степеня з додатного числа a приблизно за формулою $x_{n+1} = \frac{4}{5}x_n + \frac{a}{5x_n^4}$. При цьому різниця між x_n і $\sqrt[5]{a}$ за абсолютною величиною не перевершує $\frac{5}{4}a\left\|x_{n+1} - x_n\right\|$. Складіть програму обчислення кореня п'ятого степеня із числа a з точністю до 10^{-k} із заданим значенням k, беручи за $x_0 = \begin{cases} \min(2a, 0.95), & a \le 1, \\ a/5, & 1 \le a \le 25, \\ a/25, & a > 25. \end{cases}$

Продовження таблиці 1.22

1	*2*
12	Нехай дане натуральне число *n*. Отримати найменше число виду 2^k, що перевершує *n*.
13	Нехай дане натуральне число *n*. Обчисліть: 1 • 2 + 2 • 3 • 4 +... + *n* • ... • 2*n*.
14	Напишіть програму для обчислення кореня *n-го* степеня з додатного числа *a*, користуючись послідовними наближеннями $x_{k+1} = \frac{n-1}{n} x_k + \frac{a}{n x_k^{n-1}}$, $k = 0,1,2,\ldots$ до збігу сусідніх наближень із точністю ε, якщо задано початкове наближення x_0.
15	Нехай дане натуральне число *n*. Вилучіть із запису цього числа цифри 3 й 7, залишивши колишнім порядок інших цифр. Наприклад, із числа 3 171 507 377 має вийти 1150.
16	Нехай дане натуральне число *n*. Знайдіть найменше серед чисел $k^3 \sin\left(n + \frac{k}{n}\right)$, $k = 1,2,\ldots, n$.
17	Нехай дане натуральне число *n*. Знайдіть $a_1b_1 + a_2b_2 + \ldots + a_nb_n$, де $a_1 = b_1 = 1$, $a_k = \frac{1}{2}\left(\sqrt{b_{k-1}} + \frac{1}{2}a_{k-1}\right)$, $b_k = 2a_{k-1}^2 + b_{k-1}$, $k = 1,2,\ldots, n$.
18	Нехай є цілі числа x_1, x_2, x_3, …, x_{55}... Обчисліть величину: $x_1(x_2 + x_3)(x_4 + x_5 + x_6)\ldots(x_{46}+x_{47}+\ldots+x_{55})$...
19	Нехай $x_1 = 0,3$; $x_2 = -0,3$; $x_i = I + \sin(x_i - 1)$, $i = 3,4,\ldots$ Серед чисел x_1, x_2, … , x_{100} знайдіть найближче до якого-небудь цілого числа
20	Нехай $a_i = \frac{i-1}{i+1} + \sin\frac{(i-1)^3}{i+1}$, $i = 1,2,\ldots, n$, де *n* задано. Знайдіть суму всіх додатних чисел a_i
21	Нехай дане натуральне число *n* і речовинне число *x*. Серед чисел: $e^{\cos(x^{2k})}\sin(x^{3/k})$, *где* $k = 1,2,\ldots, n$, знайдіть найближче до якого-небудь цілого числа
22	Послідовність чисел a_1, a_2, …, a_{100} задана формулою $a_k = \sin^2(3k + 5) - \cos(k^2 - l5)$, $k = 1,2,\ldots,100$. Визначте, скільки членів послідовності з номерами 1, 2, 4, 8, 16, ... мають значення менше 0,25
23	Дано речовинне додатне число *b*. Послідовність a_1, a_2, a_3, … утворена за законом: $a_1 = b, a_i = a_{i-1} - \frac{1}{\sqrt{i}}, i = 2,3,\ldots$ Знайдіть перший від'ємний член послідовності
24	Дано речовинне від'ємне число *b*. Послідовність a_1, a_2, a_3, … утворена за законом: $a_1 = b, a_i = \frac{a_{i-1}+1}{1-\sin^2 i}$. Знайдіть перший від'ємний член послідовності

Продовження таблиці 1.22

1	*2*
25	При деяких заданих x, N й E, обумовлених уведенням, обчисліть суму N доданків заданого виду, а також суму тих доданків, які за абсолютною величиною більше E. Для другого випадку виконайте підсумовування для двох значень E, що відрізняються на порядок, і при цьому визначте кількість доданків, включених до суми. Порівняйте результати з точним значенням функції, для якої дана сума визначає наближене значення при x, що знаходиться в інтервалі $(-R, R)$. $\frac{\sin(x)}{x} = 1 - \frac{x^2}{3!} + \frac{x^4}{5!} - \frac{x^6}{7!} + \ldots \quad (R = \infty)$.

Задача 4
Регулярні типи даних. Одномірні масиви

Таблиця 1.23

Вар.	**Завдання**
1	*2*
1	Нехай є речовинні числа $x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n, r_1, r_2, \ldots, r_n$... З'ясуйте, чи є на площині точка, що належить усім колам $c_1, c_2, \ldots, c_n$, де c_i має центр із координатами x_i, y_i і радіус r_i
2	Нехай є речовинні числа $a_1, a_2, \ldots, a_{2n}$... Ці точки визначають n інтервалів числової осі (a_1, a_2), (a_3, a_4), …, (a_{2n-i}, a_{2n})... Чи є інтервалом об'єднання цих інтервалів? Якщо так, то вказати кінці об'єднаного інтервалу
3	Нехай є речовинні числа $a_1, a_2, \ldots, a_{2n}$... Ці точки визначають n інтервалів числової осі (a_1, a_2), (a_3, a_4), …, (a_{2n-i}, a_{2n})... Чи є точки числової осі, що належать принаймні трьом яким-небудь із даних інтервалів? Якщо так, то вказати яку-небудь із цих точок
4	Нехай є цілі числа $a_1, a_2, \ldots, a_n$... Нехай M – найбільше, m – найменше з них. Отримати у порядку зростання всі цілі числа з інтервалу (m, M), які не входять до послідовності $a_1, a_2, \ldots, a_n$
5	Нехай є координати центрів n окружностей й їхні радіуси. Визначте кількість окружностей, що перетинаються
6	Привласніть змінній t значення true, якщо в деякому масиві немає нульових елементів і при цьому позитивні елементи чергуються з негативними, інакше привласніть значення false
7	Нехай є десять гир вагою $a_1, a_2, \ldots, a_{10}$... Позначимо через c_k число способів, якими можна скласти вагу k, тобто c_k – це число рішень рівняння $a_1x_1 + a_2x_2 + \ldots + a_{10}x_{10} = k$, де x_i може набувати значення 0 або 1 (i = 1,..., 10). Отримати $c_1, c_2, \ldots, c_{10}$...

Продовження таблиці 1.23

1	*2*
8	Пряма на площині може бути задана рівнянням $ax + by = z$, де a, b одночасно не дорівнюють нулю, a, b, c – цілі. Нехай є коефіцієнти декількох прямих a_1, b_1, c_1, a_2, b_2, c_2, ..., a_n, b_n, c_n. Визначте, чи є серед цих прямих співпадаючі або паралельні
9	Пряма на площині може бути задана рівнянням $ax + by = z$, де a, b одночасно не дорівнюють нулю, a, b, c – цілі. Нехай є коефіцієнти декількох прямих a_1, b_1, c_1, a_2, b_2, c_2, ..., a_n, b_n, c_n. Визначите, чи є серед цих прямих три, що перетинаються в одній крапці
10	Нехай є натуральне число n, цілі числа a, x_1, x_2, …, x_n... Якщо в послідовності x_1, x_2, …, x_n є хоча б один член, який дорівнює a, то отримати суму всіх членів, що випливають за першим таким членом, інакше – знайдіть мінімальний серед непарних чисел послідовності x_1, x_2, …, x_n
11	Нехай є цілі числа a_1, a_2, …, a_n, серед яких можуть бути повторювані. Складіть новий масив із чисел, які входять до послідовності по одному разу
12	Нехай є цілі числа a_1, a_2, …, a_n, серед яких можуть бути повторювані. Складіть новий масив із чисел, узятих по одному з кожної групи рівних членів даної послідовності
13	Нехай є натуральні числа k, n, речовинні числа a_1, a_2, …, a_{kn}... Отримати послідовність $\min(a_1, a_2, \ldots, a_k)$, $\min(a_{k+1}, a_{k+2}, \ldots, a_{2k})$, $\min(a_{k(n-1)+1}, \ldots, a_{kn})$
14	Нехай є натуральні числа k, n, речовинні числа a_1, a_2, …, a_{kn}... Отримати послідовність $\max(a_1, a_2, \ldots, a_k)$, $\max(a_{k+1}, a_{k+2}, \ldots, a_{2k})$, $\max(a_{k(n-1)+1}, \ldots, a_{kn})$
15	Нехай є натуральні числа k, n, речовинні числа a_1, a_2, …, a_{kn}... Одержіть $\min(a_1 + a_2 + \ldots + a_k, a_{k+1}, a_{k+2}, \ldots, a_{2k}, \ldots, a_{k(n-1)+1}, \ldots, a_{kn})$
16	Нехай є натуральні числа k, n, речовинні числа a_1, a_2, …, a_{kn}... Отримати $\max(a_1 + a_2 + \ldots + a_k, a_{k+1}, a_{k+2}, \ldots, a_{2k}, \ldots, a_{k(n-1)+1}, \ldots, a_{kn})$
17	Нехай є натуральні числа k, n, речовинні числа a_1, a_2, …, a_{kn}... Отримати послідовність $\min(\max(a_1, a_2, \ldots, a_k), \max(a_{k+1}, a_{k+2}, \ldots, a_{2k}), \max(a_{k(n-1)+1}, \ldots, a_{kn}))$
18	Нехай є натуральні числа k, n, речовинні числа a_1, a_2, …, a_{kn}... Отримати послідовність $\max(\min(a_1, a_2, \ldots, a_k), \min(a_{k+1}, a_{k+2}, \ldots, a_{2k}), \min(a_{k(n-1)+1}, \ldots, a_{kn}))$
19	Нехай дана послідовність чисел. Усі її елементи, які не дорівнюють нулю, перенесіть, зберігаючи їхній порядок, у початок даної послідовності, а нульові – у кінець
20	Нехай числова пряма поділена на довільні відрізки точками a_1, a_2, …, a_n... З'ясуйте, якому з відрізків належить дана точка x

Продовження таблиці 1.23

1	*2*
21	У масиві з n елементів підрахуйте кількість четвірок a_i, a_{i+1}, a_{i+2}, a_{i+3}, членів, що йдуть один за одним, з яких усі члени рівні
22	У масиві з n елементів підрахуйте кількість четвірок a_i, a_{i+1}, a_{i+2}, a_{i+3}, членів, що йдуть один за одним, з яких усі члени різні
23	Довільний опуклий багатокутник заданий координатами своїх вершин на площині. Знайдіть саму довгу діагональ даного багатокутника
24	Довільний опуклий багатокутник заданий координатами своїх вершин на площині. Знайдіть саму коротку діагональ даного багатокутника
25	Обчислите площу довільного опуклого багатокутника, заданого координатами своїх вершин на площині, поділивши багатокутник на трикутники

Задача 5
Двовимірні масиви. Процедури й функції

У кожному із запропонованих нижче завдань (табл. 1.24) використайте процедури уведення й виведення елементів матриці по рядках.

Таблиця 1.24

Вар.	**Завдання**
1	*2*
1	Перевірте властивість $(A^T)^T = A$, де A – вихідна матриця (n x n), T означає транспонування. Використайте процедуру транспонування
2	Нехай задана речовинна матриця A (n x n). Упорядкувати елементи матриці: a) за неспаданням значень максимальних елементів у рядках; b) за неспаданням сум елементів рядків
3	У заданій матриці A (n x n) визначте кількість рядків, які впорядковані за зростанням. Використайте підпрограму перевірки впорядкованості рядка
4	У матриці A (n x n) визначте кількість рядків, елементи якого утворять арифметичну прогресію. Використайте підпрограму перевірки рядка
5	У заданій матриці A (n x n) знайдіть максимум із всіх мінімальних елементів матриці по стовпцях
6	У заданій матриці A (n x n) знайдіть мінімум усіх сум абсолютних величин елементів матриці по стовпцях. Для знаходження суми абсолютних величин стовпця використайте підпрограму-функцію

Продовження таблиці 1.24

1	*2*
7	Підрахуйте кількість рядків матриці A (n x n), елементи яких утворять монотонну послідовність. Для визначення факту монотонності використайте підпрограму
8	Ущільните матрицю A (n x n) уліво й нагору. Для виявлення нульових рядків і стовпців використайте підпрограму
9	Елементи матриці A (n x n) обчислюються за формулою $A_{i,j} = \sin(i*j)$ (i,j = 1,2,3, …, n)... Потрібно: а) сформувати матрицю B: $B_{i,j} = \frac{1}{j}\sum_{k=1} A_{i,k}$,(i,j = 1,2,3, …, n); б) надрукувати матриці A й B
10	Перевірте, є чи в матриці A (n x n) рядки, що не містять понад два негативні елементи. Для перевірки рядка використайте підпрограму
11	Нехай дана матриця A (n x n). Побудуйте вектор, кожен елемент якого містить найменший за абсолютною величиною елемент рядка
12	Складіть програму пошуку мінімального елемента, розташованого під головною діагоналлю, і максимального елемента, розташованого над головною діагоналлю заданої речовинної матриці A (n x n)
13	Нехай задана речовинна матриця. Розглядаючи її як вектор рядків, впорядкуйте її за кількістю непарних елементів у кожному рядку
14	Визначте номера рядків у матриці, у яких елементів, що належать відрізку [A, B], більше, ніж елементів, що належать відрізкам $[-\infty; A]$, $[B; \infty]$
15	Нехай дана матриця A (n x n). Побудуйте логічний вектор, кожен елемент якого набуває значення true, якщо серед елементів *i-й* рядка матриці A є хоча б два рівних, і значення false – у противному випадку. Скористайтеся логічною функцією, що для *i*-го рядка виконує зазначену перевірку
16	Нехай дана матриця A (n x n). Побудуйте вектор, кожен елемент якого дорівнює найбільшій кількості рівних елементів у відповідному до рядка матриці A. Скористайтеся функцією, що визначає цю кількість в *i*-му рядку матриці A
17	Нехай дана матриця A (n x n). Побудуйте логічний вектор, кожен елемент якого дорівнює true, якщо в рядку існує елемент, що поділяє весь масив на дві частини з однаковою сумою елементів у кожній
18	Перевірте, чи вірно, що кількість рядків матриці A (n x n), у якій всі числа непарні, кратно заданому числу x
19	Нехай дана матриця A (n x n). Побудуйте логічний вектор, кожен елемент якого дорівнює true, якщо серед елементів відповідного рядка матриці A є хоча б один елемент, що належить відрізку [0,5; 1], і false – в протилежному разі. Скористайтеся логічною функцією, що робить відповідну перевірку в *i*-му рядку

Продовження таблиці 1.24

1	*2*
20	Нехай дана матриця A (n x n). Побудуйте вектор, кожен елемент якого дорівнює кількості елементів в *i*-му рядку матриці A, не приналежному відрізку [0; 10]. Скористайтеся функцією, що робить відповідну перевірку в *i*-му рядку
21	Нехай дана матриця A (n x n). Побудуйте вектор, кожен елемент якого дорівнює сумі елементів *i*-го рядка матриці A, більших, ніж значення мінімального елемента в цьому рядку. Скористайтеся функцією, що визначає відповідну операцію в кожному рядку матриці A
22	Нехай дана матриця A (n x n). Впорядкуйте рядки за неспаданням сум цифр елементів цього рядка. Скористайтеся функцією, що визначає для кожного числа суму його цифр
23	Надрукуйте рядки й стовпці матриці W (n x n), на перетинанні яких знаходяться максимальні й мінімальні елементи, якщо елементи матриці обчислюються за формулами: $W_{i,j} = \begin{cases} i^2 + j, i < j, \\ 1/2, i = j, \\ i + j^2, i > j. \end{cases}$
24	Елементи матриці A (n x n) обчислюються за формулою $A_{i,j} = i*\sin(j) + \sin(i)$ (i,j = 1,2,3, …, n)... Потрібно: а) сформувати матрицю B: $B_{i,j} = \frac{A_{i,j}}{\sqrt{A_{i.j}^2 + i^2 + j^2}}$, (i,j = 1,2,3, …, n); б) надрукувати сформовані матриці A, B й їхній добуток A*B
25	Нехай дана матриця A (n x n), де $A_{i,j} = (-1)^i*i*j$. Побудуйте й надрукуйте матрицю B, елементи якої визначаються за правилом: $B_{i,j} = A_{i,j} * \max A_{i,k}$ $(1 \le k \le n)$

Задача 6
Рядки, записи, множини. Огляд усіх пройдених тем

Таблиця 1.25

Вар.	**Завдання**
1	*2*
1	Складіть програму обчислення суми номерів місць, на яких у слові S знаходяться літери, що позначають голосні звуки
2	Нехай уводиться послідовність чисел у діапазоні від 1 до 255. Ознака кінця послідовності – 0. Визначте змінні min й max як мінімальне й максимальне з уведених чисел. Надрукуйте по одному разу всі числа з інтервалу (min, max), які не були уведені

Продовження таблиці 1.25

1	*2*
3	Нехай уводиться послідовність символів. Ознака кінця послідовності – крапка. Надрукуйте всі латинські літери, які є в даній послідовності символів
4	Нехай задана довільна послідовність символів. Ознака кінця послідовності – крапка. Надрукуйте ті символи, які зустрічаються в даній послідовності більше одного разу
5	Нехай даний текст, що закінчується крапкою. Текст складається зі слів, відокремлених пробілами. Слова являють собою довільну послідовність символів, відмінних від пробілу. Надрукуйте всі слова, які складаються з тих самих літер, що й останнє слово тексту
6	Нехай даний текст, що закінчується крапкою. Текст складається зі слів, відокремлених пробілами. Слово являє собою послідовність латинських літер. Надрукуйте ті слова, до яких не входить жодна з літер першого слова
7	Нехай даний текст, що закінчується крапкою. Текст складається зі слів, відокремлених пробілами. Слово – послідовність російських літер (як малих, так і великих). Надрукуйте слова, що мають парний номер, які складаються тільки з повторюваних літер
8	Нехай даний текст, що закінчується крапкою. Текст складається зі слів, відокремлених пробілами. Слово – послідовність латинських літер. Надрукуйте слова тексту, що мають непарний номер, у яких немає ні однієї повторюваної літери
9	Нехай задана цілочисельна квадратна матриця розмірності n. Надрукуйте всі значення i ($1 < i < n$), при яких *i*-й рядок симетричний, а *i*-й стовпець упорядкований за спаданням
10	Нехай задана цілочисельна квадратна матриця розмірності n. Елементи матриці знаходяться в діапазоні від 1 до 100. Надрукуйте всі цифри із заданого діапазону, яких немає в жодному з рядків заданої матриці
11	Нехай задана цілочисельна квадратна матриця розмірності n. Відомо, що значення елементів матриці не менше 0 і не більше 30. Надрукуйте номера тих рядків матриці, які містять усі цілочисельні елементи, що лежать у діапазоні від мінімального елемента розглянутого рядка до максимального елемента. Наприклад, розглянемо рядки матриці розмірності 5. Якщо рядок складається з елементів 1, 3, 2, 5, 4, то її мінімальний елемент дорівнює 1, а максимальний – 5. Значення 2, 3, 4 саме становлять усі ті числа, які попадають у проміжок від 1 до 5, тому номер такого рядка необхідно надрукувати. Якщо ж рядок складається з елементів 7, 6, 3, 2, 5, то весь діапазон від мінімального елемента 2 до максимального елемента 7 не заповнюється наявними в рядку значеннями. Не вистачає значення 4. Тому номер такого рядка друкувати не слід
12	Нехай задана символьна квадратна матриця розмірності n. Надрукуйте елементи матриці, що лежать на її головній діагоналі, якщо всі вони відмінні від елементів, що належать побічній діагоналі. Якщо ця умова не виконується, то надрукуйте елементи побічної діагоналі даної матриці

Продовження таблиці 1.25

1	*2*
13	Нехай задана символьна матриця розмірності n x m. Надрукуйте всі символи, що перебувають у стовпцях, елементи яких симетричні
14	Надрукуйте всі цілі числа, що лежать у діапазоні від 5 до 2 500, які можуть бути у вигляді 5n+7m, де n й m – цілі числа (m, n ≥ 0)
15	Надрукуйте всі цілі числа в діапазоні від 1 до 3 600, які можуть бути у вигляді $n^2 + m^2$, але які не можна подати як 5L + 5k (m, n, k, l > 0)
16	Надрукуйте всі цілі числа в діапазоні від 1 до 1 600, які можуть бути у вигляді $x^2 + y^2$, але які не можна подати у вигляді $xy = c^2$, де с змінюється від 1 до 5
17	Нехай задані n відрізків із цілочисельними координатами кінців. Координати кінців знаходяться у діапазоні від 0 до 100 включно. Визначте, чи існує точка із цілочисельними координатами, що належить усім цим відрізкам
18	У класі навчаються 25 учнів. Кожному учневі були виставлені оцінки за чверть по 15 предметах. Визначте, скільки в класі відмінників, четвірочників і трієчників
19	Відомі результати анкетування ста осіб. Анкета складається зі 150 пунктів, на які пропонувалося відповісти ствердно, негативно або «немає певної думки з цього питання». Надрукуйте номера тих пунктів анкети, на які були отримані тільки стверджувальні й тільки негативні відповіді всіх опитаних (якщо, звичайно, такі пункти є)
20	У місті N є 100 кондитерських магазинів. Відомо, що в кожному із цих магазинів не більше 20 видів ласощів в асортименті. Які види ласощів є у всіх наявних магазинах? Чи існує магазин, що торгує унікальною продукцією? Перелічите 5 видів ласощів, які є в більшості магазинів міста N (асортименти кондитерських магазинів розглядайте як дані переліченого типу)
21	Нехай задані два речення, слова в яких відокремлені комами або пробілами. Кожне речення закінчується крапкою. Чи можна з літер першого речення скласти друге речення й навпаки? Якщо не можна ні те, ні інше, то перелічіть літери, яких не вистачає в першому (другому) реченні, щоб скласти друге (перше)
22	В їдальні є окремі меню на сніданок, обід і вечерю. Відомо, що в кожному із цих меню не більше 10 видів страв. Визначите, які види страв є й на сніданок, і на обід, і на вечерю, якщо такі є. Визначте види страв, які є тільки на сніданок, тільки на обід, тільки на вечерю (види страв розглядайте як дані переліченого типу)
23	Уводиться послідовність слів. Визначте, яке кількість слів буде потрібна, щоб залучити всі літери англійського (російського алфавіту). Уведення слів кінчається, коли залучені всі літери
24	Уводиться слово-зразок. Потім уводиться список слів (не більше 100). Визначте слова, у яких немає хоча б однієї літери зі слова-зразка. Виведіть такі слова, а також літери, яких немає в слові-зразку
25	Складіть програму обчислення суми номерів місць, на яких у слові S розміщені літери, що позначають приголосні звуки

2 ОСНОВИ ПРОГРАМУВАННЯ МОВОЮ СІ

2.1 Основні відомості про мову програмування Сі

При викладенні основ нової мови програмування передбачається, що здобувачі вищої освіти знайомі з основами програмування й уміють становити нескладні програми мовою Паскаль. Тому тут будуть наведені відмінності мови Сі від Паскаля:

- усі ідентифікатори регістрозалежні (a та A – це два різних ідентифікатори);
- крапка з комою (;) є не розмежуванням операторів й описів, а закінченням оператора або опису;
- замість операторних дужок begin – end використається блок {}
- присвоєння значення має вигляд = (знак «дорівнює»);
- порівняння даних: дорівнює (==), не дорівнює (!=);
- логічні операції: АБО (||), І (&&) і НІ (!), наприклад: ((x>0)||(y>0))&&(z>0);
- коментарі: /* коментар */ або // до кінця рядка.

Сі надає змоги скорочення запису скрізь, де це можливо (табл. 2.1).

Таблиця 2.1 – Можливість скорочення запису

Паскаль	Сі
k:=k+1	k++
k:=k-1	k--
k:=k+2	k+=2
k:=k-2	k-=2
k:=k*2	k*=2
k:=k/2	k/=2
k:=k div n	k%=n
m:=k; k:=k+1	m=k++
k:=k+1;m:=k	m=++k

Основні типи даних наведені в таблиці 2.2.

Опис змінних відбувається за шаблоном: тип ім'я;

Наприклад:

```
int k;
int m=5;
```

Перед типом можна вказати клас пам'яті, до якого відноситься змінна: auto, register, static, extern.

Таблиця 2.2 – Основні типи даних

Тип даних	Розмір, біт	Діапазон значень
void	0	Немає даних
char	8	-128…127
unsigned char	8	0…255
int	16	-32768…32767
unsigned int	16	0…65535
long	32	±2 млрд.
unsigned long	32	0..4 млрд.
float	32	±3.4E±38
double	64	±1.7E±308
long double	80	±3.4E±4932

Область дії змінних: уся програма, функція, блок (останньої можливості в Паскалі не було).

Перетворення типів відбувається так: (ім'я_типу) операнд або ім'я_типу (операнд). Наприклад: int (k) або (unsigned int) k.

Уведення даних: scanf(формат, список_адрес_змінних);

Наприклад: scanf("%i %f",&k,&a);

Виведення даних: printf(формат, список_змінних);

Наприклад: printf("k=%3i, a=%5.2f",k,a);

Формат містить кількість позицій і код типу даних.

Лабораторна робота 1. Лінійний обчислювальний процес

Завдання. Скласти програму для обчислення функції b = f(x,y,z), де z = w(x,y) при постійних значеннях x і в. Варіанти завдань у вигляді значень *x, y* і функцій *f* й *w* задані в таблиці 2.3.

Таблиця 2.3

Вар.	$f(x, y, z)$	$w(x, y)$	x	y
1	*2*	*3*	*4*	*5*
0	$\sin(x)\cdot e^{z}$	$\sin x + \cos y$	$-\pi$	π
1	$e^{-3x}(\operatorname{ctg} z + 3y)$	$\sqrt{\cos^2\lvert x\rvert + y}$	–3.22	0.15
2	$\dfrac{\sqrt[3]{x} + \cos 3y}{z + x^{z}}$	$\dfrac{2xy}{x - \sin y}$	3.27	–1.84
3	$\dfrac{(y+z)/(y+x)}{(x+z)^2}$	$\dfrac{2\cdot\sqrt{x+y}}{y + \operatorname{tg} x}$	–1.32	9.35
4	$x^{y} + \sqrt{\lvert x + y\cdot z\rvert}$	$\dfrac{5\sqrt{x}}{x^3 + y^2}$	0.75	–2.55

Продовження таблиці 2.3

1	*2*	*3*	*4*	*5*
5	$\lg\left(\lvert x\rvert+\sqrt{y}+\lvert z\rvert\right)$	$\frac{\cos x}{5y^2}$	–5.24	3.35
6	$z+\frac{y\operatorname{tg}x}{x\operatorname{ctg}y}$	$x+\cos y$	0.32	–5.75
7	$\frac{z}{x^2+y^2}$	$\frac{\sin x+\cos y}{x+y}$	–2.54	3.16
8	$\frac{z}{\ln x+y}+e^x$	$\frac{2x}{\sin^2 x+\cos^2 y}$	1.28	–2.82
9	$\sin z+\frac{\operatorname{tg}x}{\operatorname{ctg}y}$	$\frac{x+\lg y}{y+\lg\lvert x\rvert}$	–0.72	3.29
10	$\ln\lvert z\rvert+\frac{\lg x}{\cos y}$	$\sqrt{2x+0.5y}$	11.42	–2.76
11	$\frac{z^3}{x+y}+\operatorname{tg}^2 x$	$\frac{e^y}{\ln\lvert x\rvert+\ln y}$	–5.43	1.87
12	$\sin z+\frac{e^{x+y}}{\operatorname{arctg}x}$	$\frac{x}{\sin x}+\frac{y}{\cos y}$	3.57	–0.32
13	$z^{x/y}+\sin x$	$\cos\left(5x+6y+e^x\right)$	–1.42	12.1
14	$\frac{e^x}{2x+\ln\lvert x+y\rvert}+\sin z$	$\sin^2 x+\cos^2 y$	1.34	–0.65
15	$\sqrt{\lvert x+y\rvert}+\operatorname{arctg}z$	$\frac{x+y}{\ln\lvert x+y\rvert}$	–3.12	1.78
16	$\frac{4y^3+5x^2}{z}+\operatorname{tg}z$	$\frac{x+\sqrt{x}}{\operatorname{tg}(x+y)}$	0.72	–3.47
17	$\frac{y\sqrt{x}+x\cdot e^y}{\ln\lvert z\rvert}$	$\frac{e^x+e^y}{x+y}$	–2.65	5.32
18	$z+\sin x+\ln\lvert x+y+z\rvert$	$\frac{2x+\sqrt[3]{\lvert x+y\rvert}}{\lg x}$	0.32	–1.32
19	$\frac{x+z^2+y^3}{\operatorname{tg}^2(x+y)}$	$\frac{x+\sqrt{\lvert x+y^2\rvert}}{\sin(x+y)}$	–4.54	0.45
20	$\operatorname{tg}z+\operatorname{tg}x+\operatorname{tg}y$	$\frac{x+y}{\ln\lvert x+y\rvert}$	2.52	–8.12
21	$\frac{\sin(x+y)}{z+x+y}$	$\frac{x-y^2}{\sqrt[3]{\lvert x+y\rvert}}$	–0.73	3.28

Продовження таблиці 2.3

1	*2*	*3*	*4*	*5*
22	$\frac{\ln\lvert x+z\rvert}{\sqrt[3]{\lvert y+z\rvert}}$	$e^{x+y}+\cos(x+y)$	1.76	–0.75
23	$\frac{\sqrt[3]{\lvert x+y+z\rvert}}{\ln\lvert x+z\rvert}$	$\frac{\operatorname{tg}(x+y)}{\operatorname{ctg}(x-y)}$	–0.62	2.45
24	$\lg\lvert z\rvert+\lg\lvert x+y\rvert+\operatorname{tg} z$	$\sin x+\operatorname{tg} y$	2.65	–4.36
25	$\frac{e^{x+y}+\sin z}{e^{x-y}+\cos z}$	$\frac{\operatorname{tg}(x+y)}{\operatorname{tg}(x-y)}$	–0.9	1.2
26	$e^{z}\sin x$	$\sin x+\cos y$	π	$-\pi$
27	$\ln\lvert z\rvert\cdot\sin x$	$\sin^2 x+\cos^2 y$	$-\pi$	π

Приклад виконання завдання

Скласти програму для обчислення функції

$$z=2^{|x+y|}(tg\ y+1),$$

де $y=\sqrt{\sin x+1}$.

Вихідні дані: x = 2.3.

```
#include <stdio.h>
#include <math.h>
int main()
{
float x,y,z;
printf("Уведіть X:\n");
scanf("%f",&x);
y=sqrt(sin(x)+1);
z=pow(2,abs(x+y))*(tan(y)+1);
printf("Результат: x=%5.2f y=%5.2f z=%5.2f\n",x,y,z);
}
```

2.2 Різновиди обчислювальних процесів

На відміну від Паскаля, у Сі існують як операція вибору (умови), так й оператор вибору (умови). Операція вибору застосовується в простих випадках, коли варіантом є один оператор.

Операція вибору: вираз1 ? вираз2 : вираз3.

Наприклад:

y = x > 0 ? sin(x) : cos(x);

Оператор вибору має вигляд

```
if  (вираз) оператор1 else оператор2;
```

Як й у Паскалі, гілка else може бути відсутньою. Однак тут немає службового слова then – воно мається на увазі. Сама умова обов'язково має бути у круглих дужках.

Попередній приклад запишеться так:

```
if  (x > 0) y = sin(x) else y = cos(x);
```

Замість формули умови можна просто записати арифметичний вираз або ім'я змінної: нуль означає «неправда», ненульове значення – «істину».

Складні оператори мають бути в «операторних дужках», роль яких у мові Сі грають фігурні дужки.

Перемикач (оператор вибору одного варіанта з декількох) – один з небагатьох операторів у Сі, що поступається за своєю функціональністю Паскаль-аналогу:

```
switch  (вираз)
{
  case вираз_1: оператори_1;
  case вираз_2: оператори_2;
  …
  case вираз_n: оператори_n;
  default: оператори;
}
```

На відміну від case у Паскалі, коли вибір одного варіанта автоматично завершував роботу оператора, тут необхідно примусово його переривати, використовуючи службове слово break, інакше перевірка буде продовжена по всіх рядках, що залишилися.

Як й у Паскалі, у Сі використається три різновиди циклу, з тими самими назвами.

Цикл із передумовою: while (вираз-умова) {тіло_циклу}

Цикл із постумовою: do {тіло_циклу} while (вираз-умова);

Ітераційний цикл:

for (ініціалізація; вираз-умова; список) {тіло_циклу}

Вкрай відмінним від Паскаля є ітераційний цикл: крім особливої форми запису, він дозволяє використати в якості ітераційної змінну будь-якого типу, при цьому крок також може бути різним. Усередині кожного із трьох блоків циклу for може бути від нуля до декількох операторів, при цьому вони відокремлюються один від одного комами, а блоки – крапками з комами.

Приклади:

```
for (int x=1; x <= 10; x++) {s+=pow(x,2);} // сума ряду від 1 до 10
for (x=1; x <= 5; x+=0.5) {s+=pow(x,2);} // від 1 до 5 із кроком 0.5
for (s=0,x=1; x <= 5; x+=0.5) {s+=pow(x,2);}
for (s=0,x=1; x <= 5; s+=pow(x,2),x+=0.5); // тіло циклу порожнє
for (;;);                  // формально це не помилка
```

Для передачі керування використаються оператори:

– go to – перехід на мітку;

– return – повернення із блоку або функції до основної програми, припинення роботи основної програми;

– break – припинення роботи перемикача або циклу з передачею керування першому операторові, розташованому за перемикачем або циклом;

– continue – переривання поточної ітерації циклу з передачею керування останньому операторові циклу, тобто перехід до наступної ітерації.

```
        for (x=1; x <= 5; x+=0.5)
        {
          t1=sin(x);
          t2=2*x+1;
          if (t2 == 0) continue; // аналогічно goto 2;
          s+=t1/t2;
          if (x > 5) break; // аналогічно goto 3;
2:
         }
3:      printf("s=%5.2f",s);
```

Лабораторна робота 2. Обчислювальний циклічний процес, що розгалужується

Завдання. Скласти програми для розв'язання задач.

Варіанти 0–10. Знайти суму $y = \sum \frac{F_1}{F_2}$, де $a \le x \le b$, x змінюється із кроком h = c. Варіанти завдань у вигляді значень F_1, F_2, a, b, c наведені в таблиці 2.4. Задачу розв'язати, використовуючи цикли: а) із передумовою (для непарних варіантів); б) з постумовою (для парних варіантів).

Таблиця 2.4

Вар.	**F1**	**F2**	**a**	**b**	**c**
0	$\sin x$	x	$-\pi$	π	$\pi/10$
1	$2\sqrt{x^5}\cos x^2$	x^3+3x^2-x	0.2	2.2	0.2
2	$\sqrt[3]{\cos x + x^3}$	$\dfrac{3x+5}{x^2}$	2.5	7.5	0.5
3	$\ln\lvert 3x - x^2\rvert$	$2x^3 - \operatorname{tg} x$	0.5	4.5	0.3
4	$2^{x-1}\cos 3x$	$\sqrt{\lvert 1+x-2x^2\rvert}$	–2	6	0.5
5	$\ln\lvert x-5\rvert + \sqrt[3]{5x}$	$x^{-2.5}\sin(2x+1)$	2.4	6.4	0.4
6	$e^{2x-1}+\cos x$	$\lg(2x+1)$	1.6	4.8	0.3
7	x^3+2x^2-2	$x^{2x-1}+\cos x$	3.6	7.2	0.2
8	$\ln\dfrac{x+1}{2x-1}$	$\dfrac{x}{\operatorname{tg} x + \sin 2x}$	$-\pi$	π	$\pi/10$
9	$x^2+\ln x$	$\dfrac{x}{1+\operatorname{tg} x}$	0.3	3.3	0.3
10	$1+x^2-\operatorname{tg} x$	$\dfrac{x+\sin x}{x^2-\cos x}$	1.2	13.2	0.6

Варіанти 11–20. Обчислити таблицю значень функції

$$y = \begin{cases} F_1(x), \textit{ якщо } x \le a; \\ F_2(x), \textit{ якщо } x > a, \end{cases}$$

для значень аргументу *x* в інтервалі від xn до xk із кроком hx. Варіанти завдань у вигляді вихідних даних наведені в таблиці 2.5.

Варіанти 21–27. Обчислити таблицю значень функції

$$y = \begin{cases} f1(x), & \text{якщо} \quad x < 0, \\ f2(x), & \text{якщо} \quad 0 \le x \le 1, \\ f3(x), & \text{якщо} \quad x > 1, \end{cases}$$

де f_1, f_2 й f_3 задані в таблиці 2.6. Методом перебору знайти екстремуми даної функції на відрізку. Початкове й кінцеве значення відрізка, а також крок табуляції задавати довільно.

Таблиця 2.5

Вар.	$F_1(x)$	$F_2(x)$	xn	xk	hx	a
11	$2\sqrt{x^5}\cos x^2$	x^3+3x^2-x	0.2	2.2	0.2	1.2
12	$\sqrt[3]{\cos x + x^3}$	$\frac{3x+5}{x^2}$	2.5	7.5	0.5	5.0
13	$\ln\lvert 3x - x^2\rvert$	$2x^3 - \operatorname{tg} x$	0.5	4.5	0.3	3.0
14	$2^{x-1}\cos 3x$	$\sqrt{\lvert 1+x-2x^2\rvert}$	–2	6	0.5	3
15	$\ln\lvert x-5\rvert + \sqrt[3]{5x}$	$x^{-2.5}\sin(2x+1)$	2.4	6.4	0.4	5
16	$e^{2x-1}+\cos x$	$\lg(2x+1)$	1.6	4.8	0.3	3
17	x^3+2x^2-2	$x^{2x-1}+\cos x$	3.6	7.2	0.2	4.8
18	$\ln\frac{x+1}{2x-1}$	$\frac{x}{\operatorname{tg} x + \sin 2x}$	$-\pi$	π	$\pi/10$	$\pi/5$
19	$x^2+\ln x$	$\frac{x}{1+\operatorname{tg} x}$	0.3	3.3	0.3	2.3
20	$1+x^2-\operatorname{tg} x$	$\frac{x+\sin x}{x^2-\cos x}$	1.2	13.2	0.6	4.2

Таблиця 2.6

Вар.	$F_1(x)$	$F_2(x)$	$F_3(x)$
1	**2**	**3**	**4**
21	$2\sqrt{x^5}\cos x^2$	x^3+3x^2-x	$\ln\lvert 3x-x^2\rvert$
22	$\frac{3x+5}{x^2}$	$\sqrt{\lvert 1+x-2x^2\rvert}$	$2x^3-\operatorname{tg} x$
23	$2^{x-1}\cos 3x$	$\ln\lvert x-5\rvert+\sqrt[3]{5x}$	$x^{-2.5}\sin(2x+1)$
24	x^3+2x^2-2	$x^{2x-1}+\cos x$	$x^2+\ln x$
25	$\ln\frac{x+1}{2x-1}$	$\frac{x}{\operatorname{tg} x+\sin 2x}$	$\frac{x}{1+\operatorname{tg} x}$
26	$\ln\lvert 3x-x^2\rvert$	$2x^3-\operatorname{tg} x$	$\frac{x+\sin x}{x^2-\cos x}$
27	$1+x^2-\operatorname{tg} x$	$\frac{3x+5}{x^2}$	x^3+3x^2-x

Приклади виконання завдання

1. Знайти суму

$$S=\sum \frac{\sin(x)}{2x+1},$$

де $0 \le x \le \pi$ з кроком π /20,

використовуючи цикл з передумовою.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
int main()
{
float Pi=M_PI;
float x=0,s=0,xk=Pi,xh=Pi/20;
while (x<=xk) {
      s+=sin(x)/(2*x+1);
      x+=xh;
}
printf("Сума=%6.4f\n",s);
getche();
}
```

2. Обчислити таблицю значень функції

$$y=\begin{cases}\sqrt{\sin(x)}+1, \textit{якщо } x \ge 0;\\ x^2+2x+3, \textit{якщо } x<0\end{cases}$$

для значень X у інтервалі від $-\pi$ до π з кроком π /3.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
int main()
{
float Pi=M_PI;
float x,y,xn=-Pi,xk=Pi,xh=Pi/3;
printf("\n    X        Y\n");
for (x=xn;x <= xk;x+=xh) {
            if (x < 0) {y=pow(x,2)+2*x+3;}
               else {y=sqrt(sin(x)+1);}
            printf("%8.5f %8.5f\n",x,y);
}
getche();
}
```

3. Знайти екстремуми функції

$$y = \begin{cases} \sqrt{\sin(x)} + 1, якщо\ x \geq 0; \\ x^2 + 2x + 3, якщо\ x < 0 \end{cases}$$

на інтервалі зміни аргументу від $-\pi$ до π.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
int main()
{
float Pi=M_PI;
float x,y,xn=-Pi,xk=Pi,xh=Pi/3,min=1E+10,max=-
1E+10,xmin,xmax;
printf("\n    X        Y\n");
for (x=xn;x <= xk;x+=xh) {
    if (x < 0) {y=pow(x,2)+2*x+3;}
          else {y=sqrt(sin(x)+1);}
    printf("%8.5f %8.5f\n",x,y);
    if (y > max) {max=y; xmax=x;}
    if (y < min) {min=y; xmin=x;}
}
printf("Мінімум= %8.5f при x=%8.5f\n",min,xmin);
printf("Максимум= %8.5f при x=%8.5f\n",max,xmax);
getche();
}
```

2.3 Покажчики, функції, масиви

Покажчик – це змінна, котра вказує на певне місце в оперативній пам'яті, де розташовані необхідні дані.

Опис покажчика: **тип *ім'я**.

Наприклад: int *p;

NULL – покажчик, що нікуди не вказує (обов'язково великими літерами).

При роботі з покажчиком необхідно пам'ятати, що ім'я покажчика в програмі – це посилання, а ім'я «із зірочкою» – це звертання до вмісту покажчика. Тобто p – це покажчик, а *p – це значення в комірці пам'яті, на яке цей покажчик посилається. Якщо написати **p++**, то це означає зрушення покажчика на одну позицію «праворуч», а от ***p++** – це вже збільшення на одиницю вмісту комірки пам'яті.

Розглянемо приклад. Описано дані:

int *p=4;

int k=7;

Можливі дії подані в таблиці 2.7.

Таблиця 2.7 – Значення при роботі з покажчиками

Дії	*p	k
початкові значення	4	7
*p++	5	7
p++	7	7
*p++	8	8
p--	5	8

На відміну від Паскаля, де два різновиди підпрограм описуються різними службовими словами, у Сі визначений один вид підпрограми – функція:

тип ім'я (формальні_параметри) {тіло_функції}

Повернення з функції до місця виклику (основної програми або іншої функції) здійснюється оператором return, після якого вказується значення, що повертається (замість присвоєння цього значення імені функції). Заголовок функції зі списком параметрів називається *прототипом функції.*

Можна описати функцію типом void – така функція нічого не повертає в місце виклику й фактично є *процедурою* (у термінах Паскаля).

Особливістю синтаксису мови Сі є вимога наявності круглих дужок навіть у випадку відсутності параметрів – тоді дужки порожні.

На відміну від Паскаля, де транслятор перевіряв відповідність списку формальних параметрів списку фактичних параметрів (як послідовності, так і типів даних), у Сі ця відповідальність цілком полягає на програмісті. Таким чином, це можна використати для нетрадиційного розв’язання деяких задач, але частіше можна помилитися й одержати невірне розв’язання.

Певну практичну цінність мають так звані функції зі змінною кількістю параметрів:

```
int name (int a, ...) { }
```

Тут після першого формального параметра містяться три крапки, а при виклику функції кількість фактичних параметрів не обмежено.

Під час роботи із функціями зі змінними параметрами використається той факт, що всі параметри розташовуються в оперативній пам'яті послідовно. Існує два способи розв’язання проблеми. Перший формальний параметр може розглядатися або як перший елемент списку, що кінчається заздалегідь певним значенням (наприклад, нулем), або як число наступних за ним параметрів:

```
#include <stdio.h>
#include <conio.h>
int summa1(int a,...)
{
 int s=0;
 int *p;
 p=&a;
 while (*p)
 {
 s+=*(p++);
 }
 return s;
}
int summa2(int k,...)
{
 int s=0;
 int *p;
 p=&k;
 while (k)
 {
 s+=*(++p);
 k--;
 }
 return s;
}
int main()
{
 int sum;
 sum=summa1(5,2,4,0);
 printf("%5i\n",sum);
 sum=summa2(3,5,2,4);
 printf("%5i\n",sum);
 getche();
}
```

Стосовно до функцій у мові Сі існують також поняття перевантаження функцій (коли в програмі описується кілька функцій з однаковими іменами, але різними за числом й типом параметрами) і шаблонів функцій (можливість описати функцію, дійсну для будь-яких типів параметрів і значень, що повертаються). Докладно ці теми розглядаються в рамках об’єктно-орієнтованого програмування.

Як і функція, масив у мові Сі задається спрощено:

тип ім'я [константа];

«Константа» тут – кількість елементів у масиві (у цьому випадку – одномірному). Нумерація елементів масиву починається завжди з нуля.

Варіанти опису та завдання початкових значень:

```
int a[5];
int b[5]={1,2,3,4,5};
int c[5]={1,2,3};
int d[]={1,2,3,4,5};
extern int e[];
```

Написання **int e[]** буде помилковим – необхідно якимсь чином (явним або неявним) указати розмір масиву.

Ім'я масиву також є покажчиком на його перший елемент:

```
a[i] ~ *(a+i);
```

Синтаксис мови Сі дозволяє при написанні міняти місцями ім'я масиву й індекс його елемента, однак з практичного погляду робити це не рекомендується:

```
a[i] ~ i[a];
```

Опис багатомірного (двовимірного) масиву:

```
тип ім'я [константа1] [константа2];
```

«Константа1» – кількість рядків, «Константа2» – кількість стовпців. Як і при описі масиву, так і при звертанні до його елементів, кожен індекс розміщається в окремій квадратній дужці:

```
a[i][j];
```

Якщо ім'я масиву використається як покажчик на його перший елемент, то знайти елемент, розташований на перетинанні *і-го* рядка й *j-го* стовпця, можна за формулою

```
*(*(a+i)+j);
```

Варіанти опису двовимірних масивів:

```
int a[2][2];
int b[2][2]={1,2,3,4};
int c[2][2]={1,2,3};
int d[2][2]={{1},{2,3}};
```

Треба розрізняти масиви покажчиків (**int *a[5]**) і покажчик на масив (**int (*a)[5]**).

Лабораторна робота 3. Оброблення масивів з використанням функцій

Завдання. Скласти програму для рішення задач, варіанти яких наведені в таблиці 2.8, з обов'язковим використанням підпрограми для уведення матриці з екрана, її обробки й висновку на екран.

Таблиця 2.8

Вар.	**Завдання**
1	*2*
0	З кожного елемента матриці A(3,3) відняти суму її додатних елементів
1	З кожного елемента матриці A(3,3) відняти суму її парних додатних елементів
2	З кожного елемента матриці A(3,3) відняти суму її непарних додатних елементів
3	З кожного елемента матриці A(3,3) відняти суму її парних додатних елементів
4	З кожного елемента матриці A(3,3) відняти суму її непарних від'ємних елементів
5	З кожного елемента матриці A(3,3) відняти добуток її парних додатних елементів
6	З кожного елемента матриці A(3,3) відняти добуток її непарних додатних елементів
7	З кожного елемента матриці A(3,3) відняти добуток її парних від'ємних елементів
8	З кожного елемента матриці A(3,3) відняти добуток її непарних від'ємних елементів
9	Кожен елемент матриці A(3,3) помножити на суму її парних додатних елементів
10	Кожен елемент матриці A(3,3) помножити на суму її непарних додатних елементів
12	Кожен елемент матриці A(3,3) помножити на суму її непарних від'ємних елементів
13	Кожен елемент матриці A(3,3) помножити на добуток її парних додатних елементів
14	Кожен елемент матриці A(3,3) помножити на добуток її непарних додатних елементів
15	Кожен елемент матриці A(3,3) помножити на добуток її парних від'ємних елементів
16	Кожен елемент матриці A(3,3) помножити на добуток її непарних від'ємних елементів

Продовження таблиці 2.8

1	*2*
17	Кожен елемент матриці A(3,3) розділити на суму її парних додатних елементів.
18	Кожен елемент матриці A(3,3) розділити на суму її непарних додатних елементів
19	Кожен елемент матриці A(3,3) розділити на суму її парних від'ємних елементів
20	Кожен елемент матриці A(3,3) розділити на суму її непарних від'ємних елементів
21	Кожен елемент матриці A(3,3) розділити на добуток її парних додатних елементів
22	Кожен елемент матриці A(3,3) розділити на добуток її непарних додатних елементів
23	Кожен елемент матриці A(3,3) розділити на добуток її парних від'ємних елементів
24	Кожен елемент матриці A(3,3) розділити на добуток її непарних від'ємних елементів
25	До кожного елемента матриці A(3,3) додати суму її парних додатних елементів
26	До кожного елемента матриці A(3,3) додати суму її парних елементів
27	До кожного елемента матриці A(3,3) додати суму її додатних елементів

Приклади виконання завдання

1. До кожного елемента матриці M(3,3) додати суму її непарних від'ємних елементів.

```
#include <stdio.h>
#include <conio.h>
void vvod(int a[3][3])
{
int i,j;
printf("Уведіть матрицю:\n");
for (i=0;i<3;i++) {
for (j=0;j<3;j++)
{scanf("%i",&a[i][j]);}}
}
void vyvod(int a[3][3])
{
int i,j;
for (i=0;i<3;i++){
for (j=0;j<3;j++)
{printf("%4i",a[i][j]);}
printf("\n");}
```

```
}
int summa(int a[3][3])
{
int i,j,sum=0;
for (i=0;i<3;i++){
for (j=0;j<3;j++)
{if ((a[i][j] < 0) && (a[i][j]%2!=0))
{sum+=a[i][j];}
}}
return sum;
}
void work(int s,int a[3][3])
{
int i,j;
for (i=0;i<3;i++){
for (j=0;j<3;j++)
{a[i][j]+=s;}}
}
int main()
{
int m[3][3];
vvod(m);
printf("\nВихідна матриця:\n");
vyvod(m);
int s=summa(m);
printf("\nСума=%3i\n",s);
work(s,m);
printf("\nРезультат:\n");
vyvod(m);
getche();
}
```

2. Кожен елемент матриці M(3,3) поділити на суму її додатних елементів.

```
#include <stdio.h>
#include <conio.h>
void vvod(float a[3][3])
{
int i,j;
float b;
printf("Уведіть матрицю:\n");
for (i=0;i<3;i++) {
for (j=0;j<3;j++)
{scanf("%f",&b);a[i][j]=b;}}
}
void vyvod(float a[3][3])
{
int i,j;
for (i=0;i<3;i++){
for (j=0;j<3;j++)
{printf("%6.2f",a[i][j]);}
```

```
printf("\n");}
}
float summa(float a[3][3])
{
int i,j; float sum=0;
for (i=0;i<3;i++){
for (j=0;j<3;j++){
if (a[i][j] > 0) {sum+=a[i][j];}
}}
return sum;
}
void work(float s,float a[3][3])
{
int i,j;
for (i=0;i<3;i++){
for (j=0;j<3;j++)
{a[i][j]/=s;}}
}
int main()
{
float m[3][3];
vvod(m);
printf("\nВихідна матриця:\n");
vyvod(m);
float s=summa(m);
printf("\nСума = %5.2f\n",s);
work(s,m);
printf("\nРезультат:\n");
vyvod(m);
getche();
}
```

2.4 Оброблення символьних даних

На відміну від Паскаля, де змінна рядкового типу є зручним елементом програми, який, якщо необхідно, може бути розглянутий як масив символів, у Сі оброблення символьних даних реалізоване небагато складніше.

У мові Сі рядки можуть задаватися двома способами: як масив символів (**char s[n]**) і як послідовність символів (**char *s**).

У першому випадку рядок – це масив символів, що може оброблятися саме як масив; заздалегідь обмежується розмір пам'яті, який резервується для рядка; ніякі спеціальні функції для оброблення рядка незастосовні.

У другому випадку рядок – послідовність символів, що починається із зазначеного місця в пам'яті й закінчується спеціальним символом '\0' (зворотна коса риса й нуль). Ця послідовність уже не може оброблятися як масив, розмір необхідної пам'яті заздалегідь не визначений, однак для оброблення символів може бути використана низка спеціальних функцій.

Також не можна забувати, що будь-яка нова змінна в оперативній (динамічній) пам'яті має потребу в ініціалізації або за допомогою спеціальних функцій, або при самому описі змінної:

```
char *st="abcd";
```

У Сі використаються обидва види лапок (подвійні й одинарні), однак випадки їхнього застосування різні. В одинарних лапках записуються символи, а в подвійних – рядки:

```
char s1='a';        // s1 – це символьна змінна, 1 байт
char *s2="a";       // s2 – це послідовність, 2 байти
```

У таблиці 2.9 наведений перелік основних функцій для роботи з рядками (файли string.h, stdlib.h).

Таблиця 2.9 – Функції для роботи з рядками

Функція	**Прототип і стислий опис дій**
1	*2*
atof	double atof (char *str); Перетворить рядок str на речовинне число типу double.
atoi	int atoi (char *str); Перетворить рядок str на десяткове ціле число
atol	long atol (char *str); Перетворить рядок str на довге десяткове ціле число
itoa	char *itoa (int v, char *str, int baz); Перетворить ціле v на рядок str. При зображенні числа використається основа baz (2 < baz < 36). Для негативного числа й baz = 10 перший символ - "мінус" (-)
ltoa	char *ltoa (long v, char *str, int baz); Перетворить довге ціле v на рядок str. При зображенні числа використається основа baz (2 < baz < 36)
strcat	char *strcat (char *sp, char *si); Приписує рядок si до рядка sp (конкатенація рядків)
strchr	char *strchr (char *str, int c); Шукає в рядку str перше входження символу з
strcmp	int strcmp (char *strl, char *str2); Порівнює рядки strl й str2. Результат негативний, якщо strl < str2; дорівнює нулю, якщо strl == str2 і позитивний, якщо strl > str2 (порівняння беззнакове)
strcpy	char *strcpy (char *sp, char *si); Копіює байти рядка si у рядок sp

Продовження таблиці 2.9

1	*2*
strcspn	int strcspn (char *strl, char *str2); Визначає довжину першого сегмента рядка strl, що містить символи, які не входять до множини символів рядка str2
strdup	char *strdup (const char *str); Виділяє пам'ять і переносить до неї копію рядка str
strlen	unsigned strlen (char *str); Обчислює довжину рядка str
strlwr	char *strlwr (char *str); Перетворить літери верхнього регістра в рядку у відповідні літери нижнього регістра
strncat	char *strncat (char *sp, char *si, int kol); Приписує kol символів рядка si до рядка sp (конкатенація)
strncmp	int strncmp (char *strl, char *str2, int kol); Порівнює частини рядків strl й str2, причому розглядаються перші kol символів. Результат негативний, якщо strl < str2; дорівнює нулю, якщо strl == str2 і позитивний, якщо strl > str2
strncpy	char *strncpy (char *sp, char *si, int kol); Копіює kol символів рядка si у рядок sp («хвіст» відкидається або доповнюється пробілами)
strnicmp	char *strnicmp (char *strl, char *str2, int kol); Порівнює не більше kol символів рядка strl і рядка str2, не роблячи розходження регістрів (див. функцію strncmp ())
strnset	char *strnset (char *str, int c, int kol); Заміняє перші kol символів рядка str символом c
strpbrk	char *strpbrk (char *strl, char *str2); Шукає в рядку strl першу появу кожного із множини символів, що входять до рядка str2
strrchr	char *strrchr (char *str, int c); Шукає в рядку str останнє входження символу c
strset	int strset (char *str, int c); Заповнює рядок str заданим символом c
strspn	int strspn (char *strl, char *str2); Визначає довжину першого сегмента рядка strl, що містить тільки символи, із множини символів рядка str2
strstr	char *strstr (const char *strl, const char *str2); Шукає в рядку strl підрядки str2. Повертає покажчик на той елемент у рядку strl, з якого починається підрядок str2
strtod	double *strtod (const char *str, char **endptr); Перетворить рядок str на число подвійної точності. Якщо endptr не дорівнює null, то *endptr повертається як покажчик на символ, при досягненні якого припинене читання рядка str

Продовження таблиці 2.9

1	*2*
strtok	char *strtok (char *strl, const char *str2); Шукає в рядку strl лексеми, виділені символами з другого рядка
strtol	long *strtol (const char *str, char **endptr, int baz); Перетворить символьний рядок str до значення «довге число» з основою baz (2 < baz < 36). Якщо endptr не дорівнює NULL, то *endptr повертається як покажчик на символ, при досягненні якого припинене читання рядка str
strupr	char *strupr (char *str); Перетворить літери нижнього регістра в рядку str на літери верхнього регістра
ultoa	char *ultoa (unsigned long v, char *str, int baz); Перетворить беззнакове довге ціле v на рядок str

Лабораторна робота 4. Оброблення символьних даних

Завдання. Скласти програму, що вводить рядок символів, виконує його оброблення відповідно до таблиці 2.10 та виводить результати.

Таблиця 2.10

Вар.	**Умова оброблення**
1	*2*
0	Видалити всі символи – цифри
1	Видалити всі символи, що не є цифрами
2	Видалити парні цифри
3	Видалити всі символи від "I" до "N"
4	Видалити всі знаки "+" й "–"
5	Видалити всі літери "X" й "Y"
6	Видалити всі знаки "+", за яких треба цифра
7	Видалити всі літери "B", після яких знаходиться літера "C"
8	Замінити все пари "AB" на "C"
9	Видалити всі символи, що не є латинськими літерами
10	Видалити знаки "+" й "–"
11	Підрахувати, скільки разів зустрічаються символи "+" й "–"
12	Замінити всі знаки оклику ("!") на символ "*", а символ "крапка" (".") – крапками (три крапки "...")
13	Знайти позицію (номер першого символу) сполучення "DEV"
14	З'ясувати, чи є в рядку послідовність символів (підрядок) "DEV"
15	Визначити, чи входить до рядка літера "A", і підрахувати кількість пробілів
16	З'ясувати, чи входить до рядка пара символів, що знаходяться поруч, "NO" або "ON"

Продовження таблиці 2.10

1	*2*
17	З'ясувати, є чи в рядку подвоєні символи (пари однакових символів, що знаходяться поруч), надрукувати їх
18	З'ясувати, чи є в рядку пари символів, що знаходяться поруч, кома й двокрапка (", :")
19	Вставити після кожного символу крапки (".") один символ пробілу ("_"), якщо після крапки немає пробілу
20	Послідовності наступних один за одним пробілів замінити одним пробілом (тобто видалити всі пробіли, що знаходяться безпосередньо за пробілом)
21	Підрахувати загальну кількість входжень до рядка символів "A", "a", "B" й "в"
22	Видалити з рядка всі здвоєні, строєні й т.д. символи
23	Знайти позицію (номер символу), у якій знаходиться перша кома, і номер позиції з останньої коми
24	Видалити всі символи "*", а символи, що не є "*", подвоїти
25	Вставити пробіл після кожного символу "." "," "!" або "?", якщо за цими символами не знаходиться пробіл (тобто знаходиться будь-який символ, крім пробілу)
26	Замінити всі крапки (три крапки "...") одними крапками
27	Вставити після кожного символу крапки (".") один символ пробілу (" "), якщо після крапки немає пробілу

Приклади виконання завдання

1. Визначити, скільки разів у заданому рядку символів зустрічається словосполучення "SM".

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main()
{
int i,k;
char st[10];
printf("Уведіть рядок символів:\n");
scanf("%s",st);
printf("\nВхідний рядок:\n%s\n",st);
k=0;
for (i=0;i<strlen(st)-1;i++) {
if ((*(st+i)=='S')&&(*(st+i+1)=='M')) {k++;}}
printf("Задане словосполучення зустрілося %i раз\n",k);
getche();
}
```

2. Видалити із заданого рядка символів усі цифри.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main()
{
int i,k;
char st[20];
printf("Уведіть рядок символів:\n");
scanf("%s",st);
printf("\nВхідний рядок:\n%s\n",st);
i=0;
while (i < strlen(st)) {
      if ((*(st+i)>='0')&&(*(st+i)<='9')) {
         for (k=i;k<=strlen(st);k++) {
             *(st+k)=*(st+k+1);}} else {i++;}
}
printf("Результат:\n%s\n",st);
getche();
}
```

2.5 Складні типи даних

Як й у Паскалі, у Сі, крім масивів, існують типи даних, аналогічні записам і файлам.

До структурованих типів даних, що складаються з фіксованої кількості компонентів – полів, у Сі відносяться структури (struct) і об'єднання (union).

Структура аналогічна запису в Паскалі. Єдине доповнення – можливість використання покажчика на структуру:

```
ім'я->поле
```

Втім, даний вид запису може бути замінений стандартним:

```
(*ім'я).поле
```

Правила опису структури зрозумілі на наведеному далі прикладі даних про студентів (здобувачів вищої освіти).

```
struct stud {
          char fam[10], gr[10];
          int year, rt[5], max;
          };
stud ist[3]={{"Іванов","IST-25-1",2008,{70,55,55,60,75}},
     {"Петров","IST-25-2",2008,{61,62,63,74,55}},
     {"Сидоров","IST-25-3",2008,{98,99,97,90,91}}},  s1;
```

Якщо поля структури розташовуються в пам'яті послідовно, то об'єднання – паралельно. З цього випливає, що у структури можуть завжди бути доступні всі поля, а в об'єднання вони альтернативні. Розмір структури дорівнює сумі розмірів усіх її полів, розмір об'єднання визначається максимальним його полем.

Далі в прикладі описані два практично ідентичних структурованих типи: структура st1 й об'єднання st2.

```
struct st1 {
        char fm[10];
        int m[2];
        int k;
        };
union st2 {
        char fm[10];
        int m[2];
        int k;
        };
```

Розміщення в оперативній пам'яті набуде вигляду, як показано на рисунку 2.1. Передбачається, що змінна цілого типу займає в пам'яті 2 байти. Таким чином, розмір структури дорівнюватиме 10+2*2+2=16 байт, а розмір об'єднання – 10 байт.

Рисунок 2.1 – Розміщення в пам'яті структури й об'єднання

Структури й об'єднання можуть мати так звані бітові поля, тобто поля розміром менше одного байта:

```
struct st3 {
        int k:2;
        int m:4;
        };
```

У наведеному прикладі поле з ім'ям k має розмір 2 біти (тобто зможе набувати значення від 0 до 3), поле з ім'ям m – 4 біти (значення від 0 до 15). Іноді використання бітових полів має сенс.

Файл у мові Сі описується в програмі як покажчик на місце в оперативній пам'яті – буфер обміну з файлом на диску:

FILE *f;

Тут *f – ім'я файлу. Службове слово **FILE** завжди пишеться великими літерами.

Відкриття файлу:

ім'я = fopen (ім'я_на_диску, режим);

Тут режим – це літера "r" (читання) або "w" (запис).

Наприклад: **f=fopen("lab5.dat","w");**

Якщо відкриття файлу пройшло успішно, то файловій змінній привласнюється адреса (покажчик) місця в пам'яті, де розмістився буфер обміну, інакше значення файлової змінної дорівнюватиме NULL.

З файлами можна працювати у двох режимах – текстовому й двійковому, при цьому «включення» режиму відбувається автоматично, за використовуваними функціями уведення або виведення. Для перевірки статусу файлу (поточного режиму) використається убудована змінна **_fmode**, що може набувати значення **O_TEXT** або **O_BINARY**.

Читання й запис даних у двійковий файл:

fread (адреса_змінної, к_байт, скільки_раз, ім'я_файла);

Наприклад: **fread(&k,sizeof(k),1,f);**

Запис даних у двійковий файл:

fwrite (адреса_змінної, к_байт, скільки_раз, ім'я_файла);

Наприклад: **fwrite(&k,sizeof(k),1,f);**

Читання й запис даних у текстовий файл практично не відрізняється від читання й запису при роботі з екраном:

fscanf (ім'я_файла, формат, список_адрес_змінних);
fprintf (ім'я_файла, формат, список_змінних);

Закриття будь-якого файлу:

fclose(ім'я);

Можливе також потокове уведення й виведення даних (iostream.h):

cin >> ім'я;
cout << вираз;

У таблиці 2.11 наведений перелік функцій для роботи з файлами.

Таблиця 2.11

Функція	**Прототип і стислий опис дій**	**Файл**
1	*2*	*3*
chmod	int chmod (const char *path, int mode); Змінює режим доступу до файлу	io.h
chsize	int chsize (int handle, long size); Змінює розмір файлу	io.h
close	int close (int handle); Закриває файл	io.h
creatnew	int creatnew (const char *path, int mode); Створює новий файл	dos.h
eof	int eof (int handle); Визначає, чи досягнут кінець файлу	io.h
fclose	int fclose (FILE *stream); Закриває потік	stdio.h
fdopen	FILE *fdopen (int handle, char *type); Зв'язує потік з логічним номером файлу	stdio.h
feof	int feof (int handle); Визначає, чи досягнут кінець файлу	stdio.h
ferror	int ferror (FILE *stream); Виявляє помилки в потоці	stdio.h
fgetc	int fgetc (FILE *stream); Одержує символ з потоку	stdio.h

Продовження таблиці 2.11

1	*2*	*3*
fgetchar	int fgetchar (void); Одержує символ з потоку stdin	stdio.h
fgetpos	int fgetpos (FILE *stream, fpos_t *pos); Повертає положення покажчика поточної позиції у файлі	stdio.h
fgets	char *fgets (char s, int n, FILE *stream); Одержує рядок символів з потоку	stdio.h
fopen	FILE *fopen (char *filename, char *type); Відкриває потік	stdio.h
fprintf	int fprintf(FILE *stream, const char *format [argument,...]); Здійснює форматированный висновок у потік	stdio.h
fputc	int fputc (int c, FILE *stream); Виводить символ у потік	stdio.h
fputchar	int fputchar (int c); Виводить символ у потік stdout	stdio.h
fputs	int fputs (char *string, FILE *stream); Виводить рядок символів у потік	stdio.h
fread	size_t fread (void *ptr, size_t size, size_t n, FILE *stream); Зчитує дані з потоку	stdio.h
freopen	FILE *freopen (char *filename, char *mode, FILE *stream); Зв'язує з потоком новий файл	stdio.h
fscanf	int fscanf (FILE *stream, char *format [adress,...]); Виконує форматированный уведення з потоку	stdio.h
fseek	int fseek (FILE *stream, long offset, int fromwhere); Установлює покажчик файла в потоці	stdio.h
fsetpos	int fsetpos (FILE *stream, const fpos_t *pos); Позиціює покажчик поточної позиції у файлі, пов'язаному з потоком stream	stdio.h
ftell	long ftell (FILE *stream); Повертає положення покажчика поточної позиції файлу	stdio.h
fwrite	size_t fwrite(void *ptr, size_t size, size_t n, FILE *stream); Записує дані в потік	stdio.h
getc	int getc (FILE *stream); Уводить із потоку символ	stdio.h

Продовження таблиці 2.11

1	*2*	*3*
getchar	int getchar (void); Уводить символ з потоку stdin	stdio.h
gets	char *gets (char *s); Уводить рядок символів з потоку stdin	stdio.h
getw	int getw (FILE *stream); Уводить із потоку ціле число	stdio.h
lseek	long lseek (int handle, long offset, int fromwhere); Переміщає покажчик читання/запису файлу	io.h
_open	int _open (const char *filename, int oflags); Відкриває файл для читання або запису	fcntl.h
open	int open(const char *filename, int access [unsigned mode]); Відкриває файл для читання або запису	fcntl.h
putc	int putc (int c, FILE *stream); Виводить символ у потік	stdio.h
putchar	int putchar (int c); Виводить символ у потік stdout	stdio.h
puts	int puts (const char *s); Виводить рядок у потік stdout	stdio.h
putw	int putw (int w, FILE *stream); Поміщає в потік ціле значення	stdio.h
read	int read (int handle, void *buf, unsigned len); Зчитує дані з файлу	io.h
rewind	int rewind (FILE *stream); Установлює покажчик у початок потоку	stdio.h
setmode	int setmode (int handle, unsigned amode); Установлює режим відкриття файлу	fcntl.h
unlink	int unlink (const char *filename); Видаляє файл	io.h
vprintf	int vprintf (const char *format, va_list arglist); Здійснює форматований запис у стандартний потік stdout	stdarg.h
write	int write (int handle, void *buf, unsigned len); Записує дані у файл	io.h

Лабораторна робота 5. Робота з файлами

Завдання до варіантів 0–8. Сформувати файл із модулів цілих чисел, знайти задану умову для оброблення (табл. 2.12).

Таблиця 2.12

Вар.	Умова для оброблення
0	Суму компонентів файлу
1	Кількість парних чисел серед компонентів файлу
2	Кількість непарних чисел серед компонентів файлу
3	Суму квадратів непарних чисел
4	Суму квадратів парних чисел
5	Середнє арифметичне значення компонентів з парними номерами
6	Найбільше зі значень компонентів з парними номерами
7	Найменше зі значень компонентів з непарними номерами
8	Добуток квадратів компонентів файлу

Завдання до варіантів 9–16. Прийнявши, що координати точок на площині задаються двома числами x та y, скласти програму, що вводить з клавіатури координати точок і записує їх послідовно у файл: спочатку x, а потім y. Після завершення уведення переглядається файл і оброблюється відповідно до таблиці 2.13.

Таблиця 2.13

Вар.	Оброблення
9	Підрахувати кількість точок, що потрапляють до кола радіуса 4 із центром на початку координат
10	Знайти суму відстаней кожної точки від центра координат
11	Підрахувати кількість точок, що потрапляють до прямокутника, утвореного осями координат і прямими x = 2 й y = 4
12	Знайти середнє відхилення (відстань) точок від осі OX
13	Знайти середнє відхилення (відстань) точок від центра координат
14	Підрахувати кількість точок, що лежать поза колом радіуса 2 і із центром у точці (2, 2)
15	Знайти середнє відхилення (відстань) точок від осі OY
16	Підрахувати кількість точок, що лежать поза трикутником, утвореним осями координат і прямою y = 2x + 1

Завдання до варіантів 17–27. Сформувати файл із чисел послідовності $(-1)^k*0.3^k/(k + 1)$. Знайти умову (табл. 2.14).

Таблиця 2.14

Вар.	Умова для оброблення
17	Сума компонентів файлу
18	Добуток компонентів файлу
19	Сума квадратів компонентів файлу
20	Модуль суми компонентів файлу
21	Квадрат добутку компонентів файлу
22	Найбільший з компонентів файлу
23	Найбільший з компонентів з непарними номерами
24	Сума найбільшого й найменшого зі значень компонентів файлу
25	Середнє арифметичне модулів компонентів файлу
26	Квадрат максимального з компонентів файлу
27	Квадратний корінь із суми компонентів файлу

Приклад виконання завдання

Сформувати файл із модулів цілих чисел, знайти суму парних компонентів файлу. Виведення результати продублювати у текстовий файл.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
int main()
{
FILE *f,*g;
f=fopen("lab5.dat","w"); // запис даних у файл
printf("Введіть цілі числа, ознака кінця - 0:\n");
int k;
do {
     scanf("%i",&k);
     if (k!=0)
     {fwrite(&k,sizeof(k),1,f);} }
while (k);
fclose(f);
printf("Зміст файлу:\n"); // читання з файлу
if ((f=fopen("lab5.dat","r"))==NULL)
{
     printf("Помилка при читанні файлу!\n");
     return 1;
}
g=fopen("lab5.txt","w");
fprintf(g,"Зміст файла:\n");
int s=0;
```

```
do {
  fread(&k,sizeof(k),1,f);
  if (!feof(f)) {
          printf("%4i",k);
          fprintf(g,"%4i",k);
          if (!(k%2)) {s+=k;}
  }   } while (!feof(f));
printf("\nСума = %i\n",s);
fprintf(g,"\nСума = %i\n",s);
fclose(f);
fclose(g);
getche();
}
```

2.6 Робота препроцесора й виняткові ситуації

Перш ніж програму мовою Сі почне «переглядати» компілятор, з нею певні дії повинен провести так званий препроцесор. На стадії препроцесорного оброблення відбувається видалення коментарів, з'єднання операторів, розташованих на декількох рядках, в один і т.д. Препроцесор також здійснює такі дії:

– включення текстів з файлів;

– заміни в тексті програми;

– умовна компіляція;

– макропідстановки.

Усі команди (директиви) препроцесору записуються в окремих рядках і починаються з символу #:

#ім'я параметри

Параметри записуються, якщо необхідно. Крапки з комами в рядку директив не наводяться.

Розглянемо кожну групу команд докладніше.

Включення тексту з файлів:

```
#include <stdio.h>
#include "myprog.cpp"
```

Подвійні лапки використаються для включення програмних файлів, розташованих у тій самій теці, що й програма, що викликається. Кутові лапки – для заголовних файлів, розташовуваних, як правило, у спеціальних теках.

Заміни в тексті:

#define M 20
#define AAA for (int i=0; i < 10; i++)

Перший параметр – що заміняється, другий параметр – чим. Розділь-ник між параметрами – пробіл. Усі символи праворуч пробілу – рядок-замінник. Обмежень на кількість замін немає.

Умовна компіляція:

#if умова
#ifdef M
#ifndef M
#elif умова
#else
#endif

Якщо раніше за допомогою define був визначений параметр M, то блок (до однієї з наступних директив умовної компіляції) включається до компіляції, інакше – ні. Суть директив зрозуміла з їхніх назв.

Макропідстановки засобами препроцесора:

#define max(a,b) a>b?a:b

Також препроцесор може обробляти убудовані (заздалегідь визначені) макроімена: __LINE__, __FILE__, __DATE__, __TIME__ й інших.

Внесемо невеликі зміни до одного з попередніх прикладів для обро-блення рядків:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#define M 1
#define P printf
#define S scanf
#define AAA for (i=0;i<strlen(st)-1;i++) {
int main()
{
 int i,k;
 char st[20];
 clrscr();
 P("Enter string:\n");
 S("%s",st);
 P("String:\n%s\n",st);
```

```
 k=0;
#ifdef M
 AAA
 if ((*(st+i)=='S')&&(*(st+i+1)=='M')) {k++;}}
#endif
 P("Result: %i\n",k);
 getche();
}
```

Якщо в рядку умовної директиви замість «#ifdef M» написати «#ifdef A», то результат дорівнюватиме нулю.

Оброблення виняткових ситуацій у мові Сі здійснюється за допомогою операторів try, catch й throw:

```
try
{
оператор1;
оператор2;
…
throw вираз_генерації_виключення;
}
catch (тип_виключення ім'я) {оператори}
```

Докладно розділи оброблення виняткових ситуацій традиційно розглядаються в розділі об'єктно-орієнтованого програмування.

Для самостійної роботи пропонується вивчити графічну бібліотеку мови Сі.

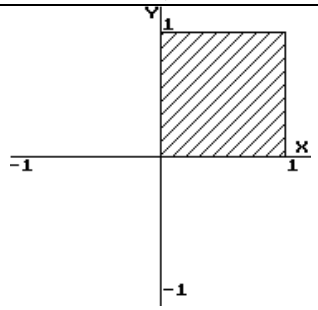
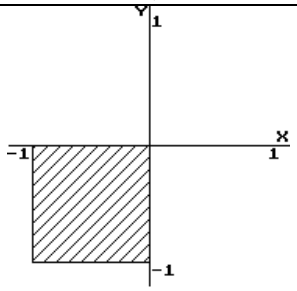
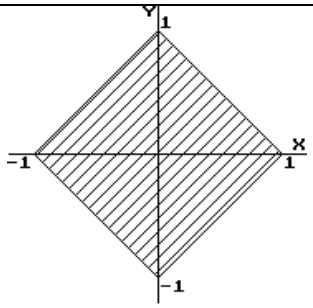
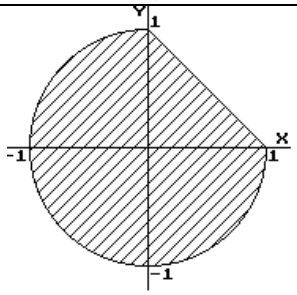
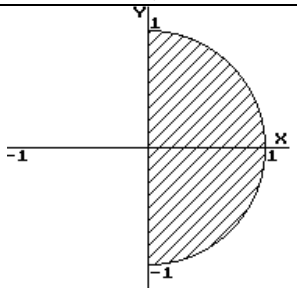
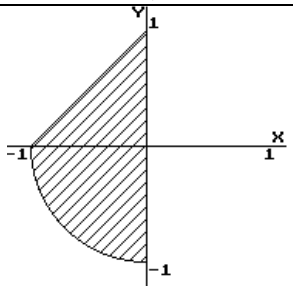
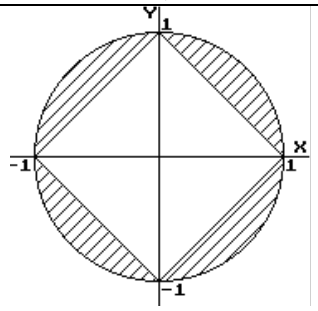
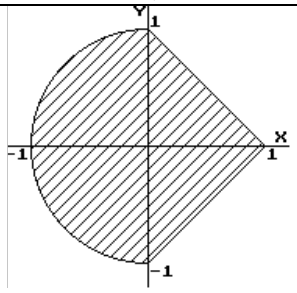
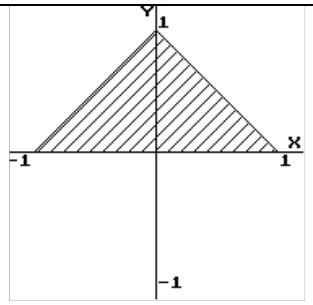
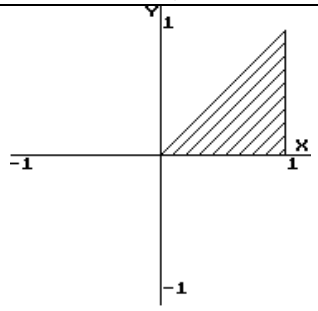
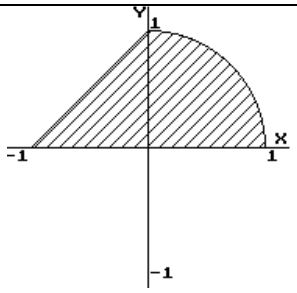
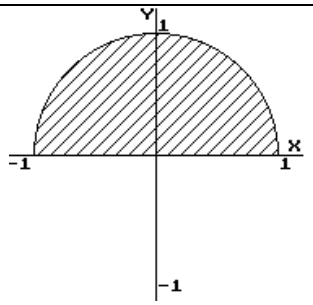
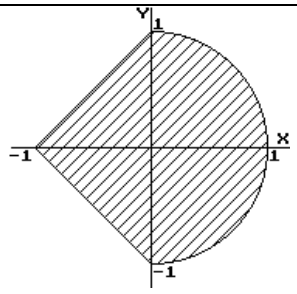
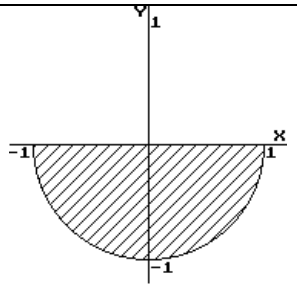
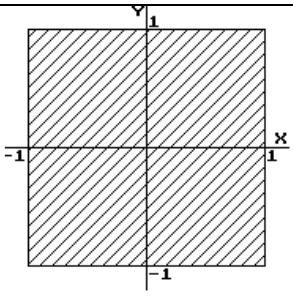
2.7 Завдання для самостійної роботи

Побудова графічних зображень у середовищі Сі

З використанням модуля роботи із графікою скласти програму для малювання на екрані трьох зображень відповідно до варіанта з таблиці 2.15 (наприклад, якщо ваш варіант – № 11, то необхідно намалювати зображення № 11, 12 й 13).

Зауваження: оскільки не всі сучасні компілятори мови Сі підтримують наведену бібліотеку для роботи із графікою, здобувач за згодою викладача може змінити завдання на самостійну роботу.

Таблиця 2.15

№ 1, 16	№ 2, 17	№ 3, 18
		
№ 4, 19	№ 5, 20	№ 6, 21
		
№ 7, 22	№ 8, 23	№ 9, 24
		
№ 10, 25	№ 11	№ 12
		
№ 13	№ 14	№ 15
		

Приклад виконання завдання

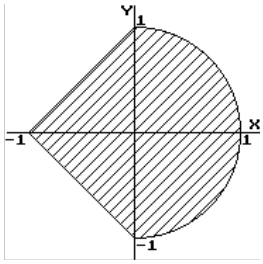


Рисунок 2.2

```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
void axes(int xc, int yc)
{
line(xc-100,yc,xc+100,yc);
line(xc+100,yc,xc+95,yc-2);
line(xc+100,yc,xc+95,yc+2);
line(xc,yc-100,xc,yc+100);
line(xc,yc-100,xc-2,yc-95);
line(xc,yc-100,xc+2,yc-95);
}
int main()
{
int xc,yc;
initwindow(400,400);
xc=200;yc=200;
axes(xc,yc);
setfillstyle(3,getmaxcolor());
pieslice(xc,yc,0,90,50);
pieslice(xc,yc,270,360,50);
line(xc,yc-50,xc-50,yc);
floodfill(xc-2,yc-2,getmaxcolor());
line(xc-50,yc,xc,yc+50);
floodfill(xc-2,yc+2,getmaxcolor());
getchar();
closegraph();
}
```

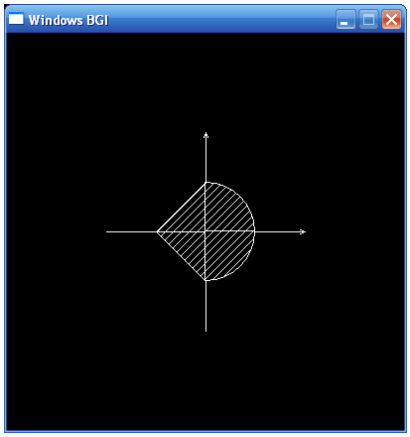


Рисунок 2.3

3 ОСНОВИ ПРОГРАМУВАННЯ МОВОЮ OBJECT PASCAL

3.1 Середовище візуального програмування Lazarus

Lazarus являє собою середовище для швидкісного розроблення додатків (RAD – rapid application development), ядром якого є модифікація створеної Ніклаусом Віртом мови Паскаль – Object Pascal. На відміну від свого «процедурно-орієнтованого» попередника Турбо Паскаля, програма на якому являла собою комп'ютерну реалізацію певного алгоритму, об'єктно-орієнтований Object Pascal дозволяє створювати повноцінні Windows-програми – по суті сукупності взаємодіючих між собою і користувачем об'єктів, які постійно очікують виникнення подій та реагують на них так, як потрібно розробнику.

Головна зручність середовища Lazarus полягає в тому, що розробник додатку (програма на Object Pascal є Windows-додатком, тому надалі будемо використовувати тільки цей термін) може конструювати його зовнішній вигляд шляхом додавання або видалення так званих візуальних компонент. Програміст може зосередитися на вирішенні поставленого завдання, не відволікаючись на вирішення супутніх проблем (введення і виведення даних, вибір, візуалізація результатів тощо).

Зовнішній вигляд і функціональність середовища Lazarus ідентичні Delphi версій 5–7. Головний недолік Lazarus: на відміну від Delphi, у нього немає оптимізатора програмного коду, і найпростіший виконуваний файл може зайняти 15 Мбайт дискового простору.

Різні версії Lazarus не дуже відрізняються одна від одної. Усі приклади в цьому посібнику виконані в середовищі Lazarus – 0.9.26.

На відміну від низки поширених Windows-додатків, побудованих за MDI-принципом (Multiple Document Interface: кілька дочірніх вікон розташовуються усередині одного великого вікна), середовище програмування Lazarus, як і Delphi, побудоване за специфікацією SDI (Single Document Interface: декількох окремо розташованих вікон). Перед початком роботи краще мінімізувати інші додатки, щоб вони не захаращували робочий простір. Далі наведені основні складові частини Lazarus (рис. 3.1):

1. Головне вікно: меню і палітра компонент.
2. Інспектор об'єктів.
3. Редактор коду.
4. Дизайнер форм – проєкт майбутнього додатку.
5. Вікно повідомлень.

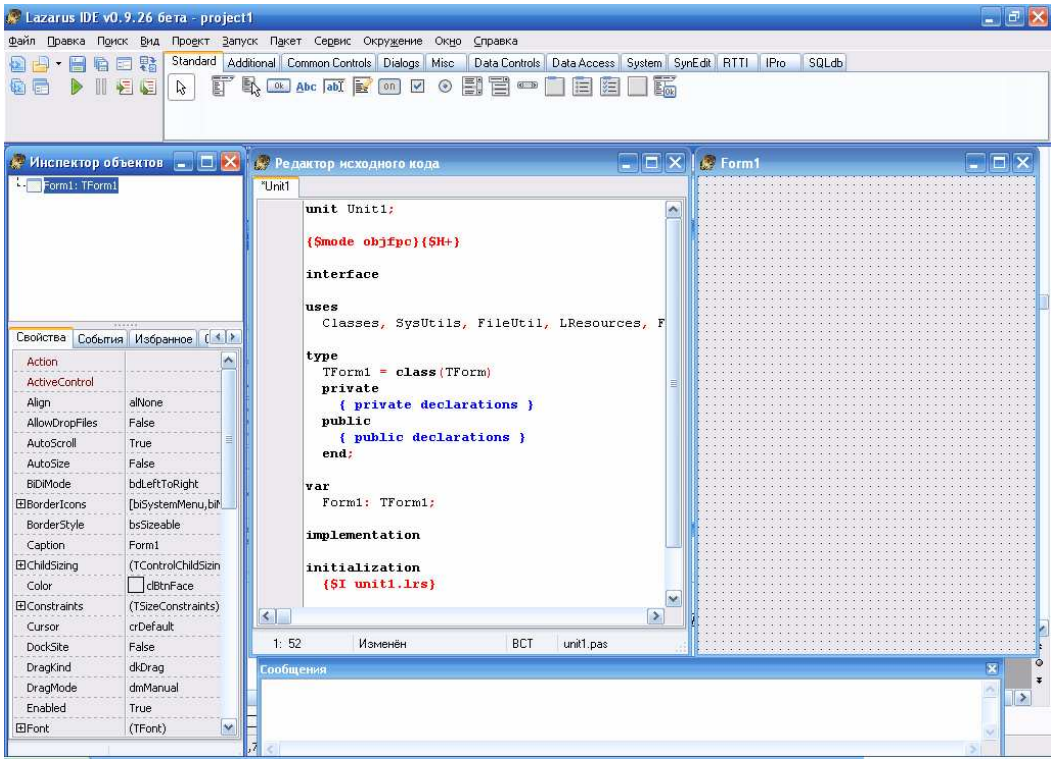


Рисунок 3.1 – Зовнішній вигляд середовища Lazarus

Загальний принцип створення додатків такий: з *Палітри компонент* вибирається потрібний об'єкт і розміщується на *Дизайнері форм* (перший раз потрібно клацнути мишкою на об'єкті, а потім – на *Дизайнері форм*); вибраний об'єкт з'явиться на формі проєкту, після чого ним можна буде маніпулювати за допомогою миші (переміщати, змінювати розміри, видаляти). За допомогою інспектора об'єктів можна змінювати практично будь-яку властивість вибраного об'єкта (колір, напис, розміри, положення тощо), а також перейти до створення методу-підпрограми, що викликається як реакція на будь-яку подію (клацання або переміщення миші, натискання клавіші і т. д.). Текст підпрограми набирається в *Редакторі коду*.

Розглянемо докладніше кожну із складових частин середовища.

Палітра компонент розташовується в так званому головному (або верхньому) вікні (рис. 3.2) разом з головним меню і супутнім йому набором піктограм. Вона використовує посторінкове угруповання об'єктів. Угорі *Палітри* знаходиться набір закладок: Standard, Additional, Dialogs тощо, причому порядок розташування закладок практично ідентичний своєму Delphi-аналогу.

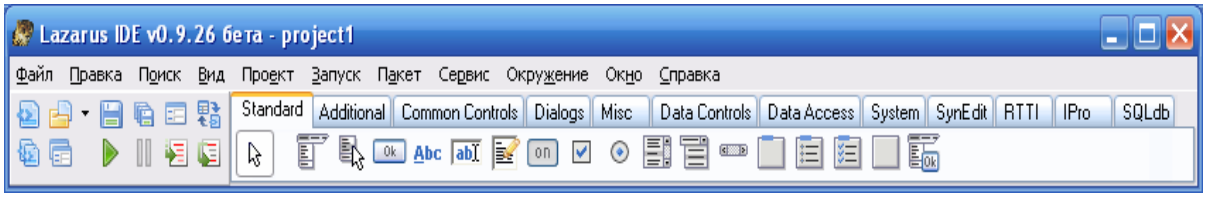


Рисунок 3.2 – Головне вікно

Дизайнер форм (рис. 3.3) являє собою проєкт вашого майбутнього додатку: на формі можна розмістити необхідні візуальні компоненти, наприклад: текстову мітку, поле введення тексту, кнопку і багато іншого. Розміщені об'єкти в будь-який момент можна перемістити в інше місце, видозмінити і навіть видалити.

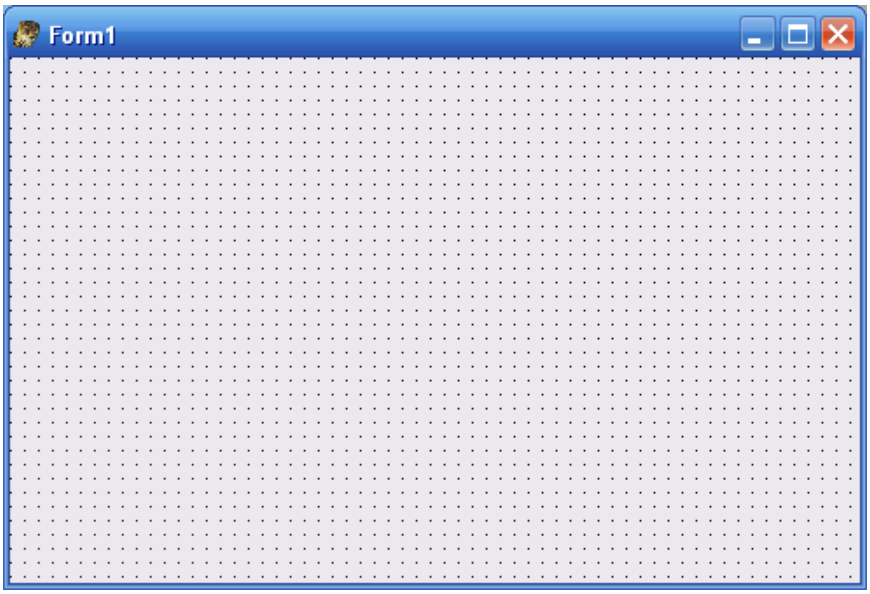


Рисунок 3.3 – Дизайнер форм

На рисунку 3.4 наведено приклад форми з розміщеними на ній кнопкою, текстовою міткою і полем введення. Як правило, написи на компонентах збігаються з їх іменами, а ті, у свою чергу, утворюються шляхом додавання до імені класу порядкового номера об'єкта даного класу на формі. Усе це можна змінити в *Інспекторі об'єктів*.

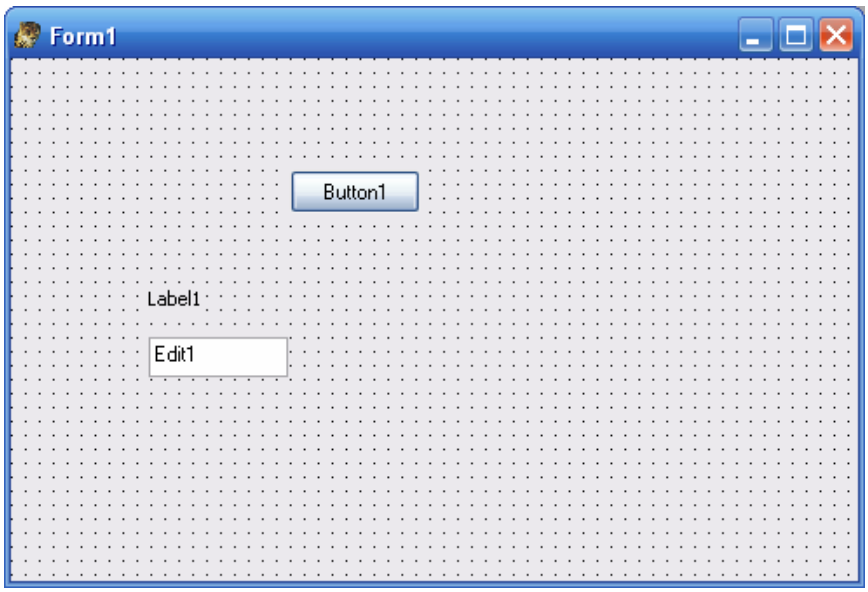


Рисунок 3.4 – Форма з розміщеними компонентами

На відміну від, наприклад, Турбо Паскаля, де вся робота над програмою проходить виключно в процесі редагування її тексту, у Lazarus вікно редактора (рис. 3.5) – одне з важливих, але, усе ж таки, не найголовніше. Бо створення будь-якого додатку починається, у першу чергу, з розміщення компонентів на формі, і тільки потім можна починати написання процедур – методів.

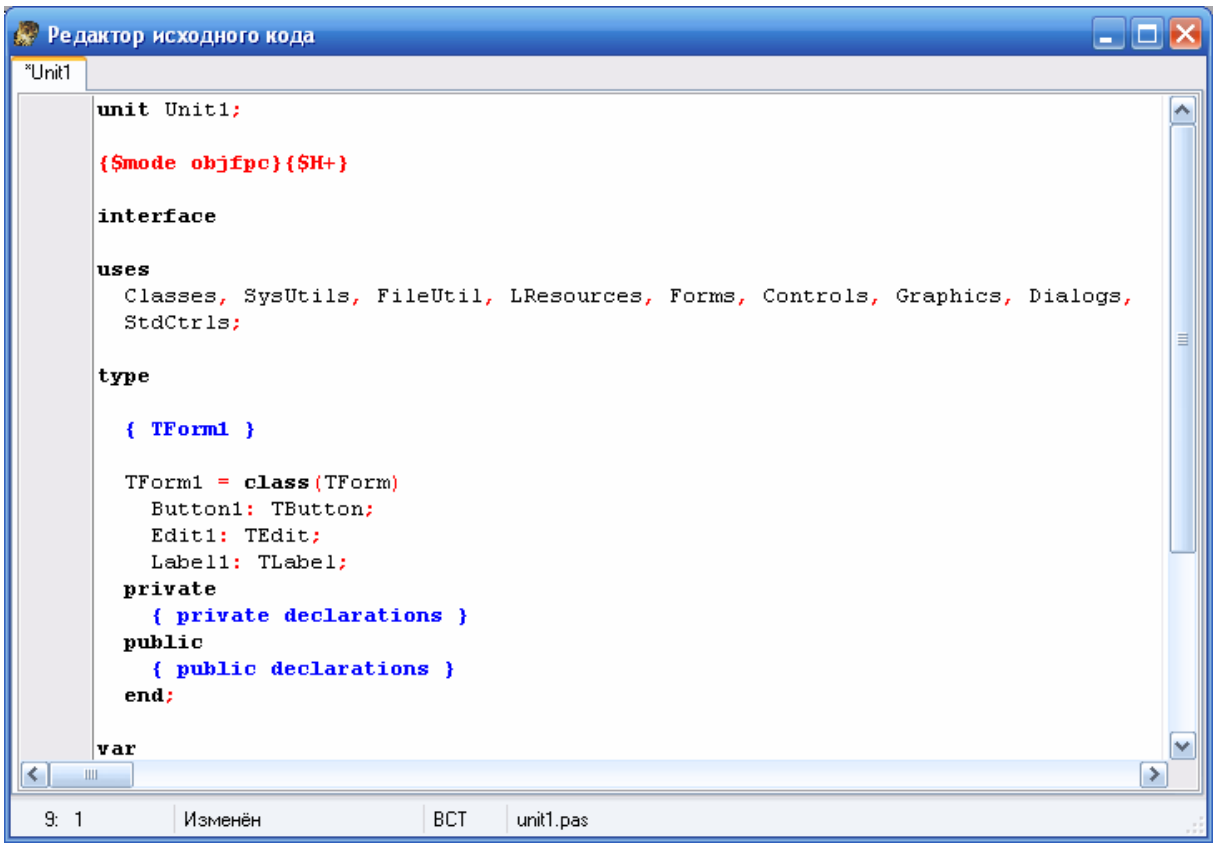


Рисунок 3.5 – Вікно редактора коду

Слід також зауважити ще одну істотну відмінність від Турбо Паскаля: проєкт програми створюється автоматично. При розміщенні на формі чергового об'єкта вся інформація про нього сама дописується в програму.

Розглянемо наш приклад трохи докладніше:

```
unit Unit1;
{$mode objfpc}{$H+}
interface
uses
  Classes, SysUtils, FileUtil, LResources, Forms, Controls, Graphics, Dialogs,  StdCtrls;
type
  { TForm1 }
  TForm1 = class(TForm)
    Button1: TButton;
```

```
    Edit1: TEdit;
    Label1: TLabel;
  private
    { private declarations }
  public
    { public declarations }
  end;
var
  Form1: TForm1;
implementation
initialization
  {$I unit1.lrs}
end.
```

Доступна частина програми являє собою модуль з ім'ям Unit1 (як і в будь-якому середовищі візуального програмування, кожна форма в Lazarus являє собою окремий модуль). У інтерфейсній частині модуля описується новий клас – нащадок стандартного класу TForm, а також нові компоненти, які він містить (кнопка, мітка і поле введення тексту):

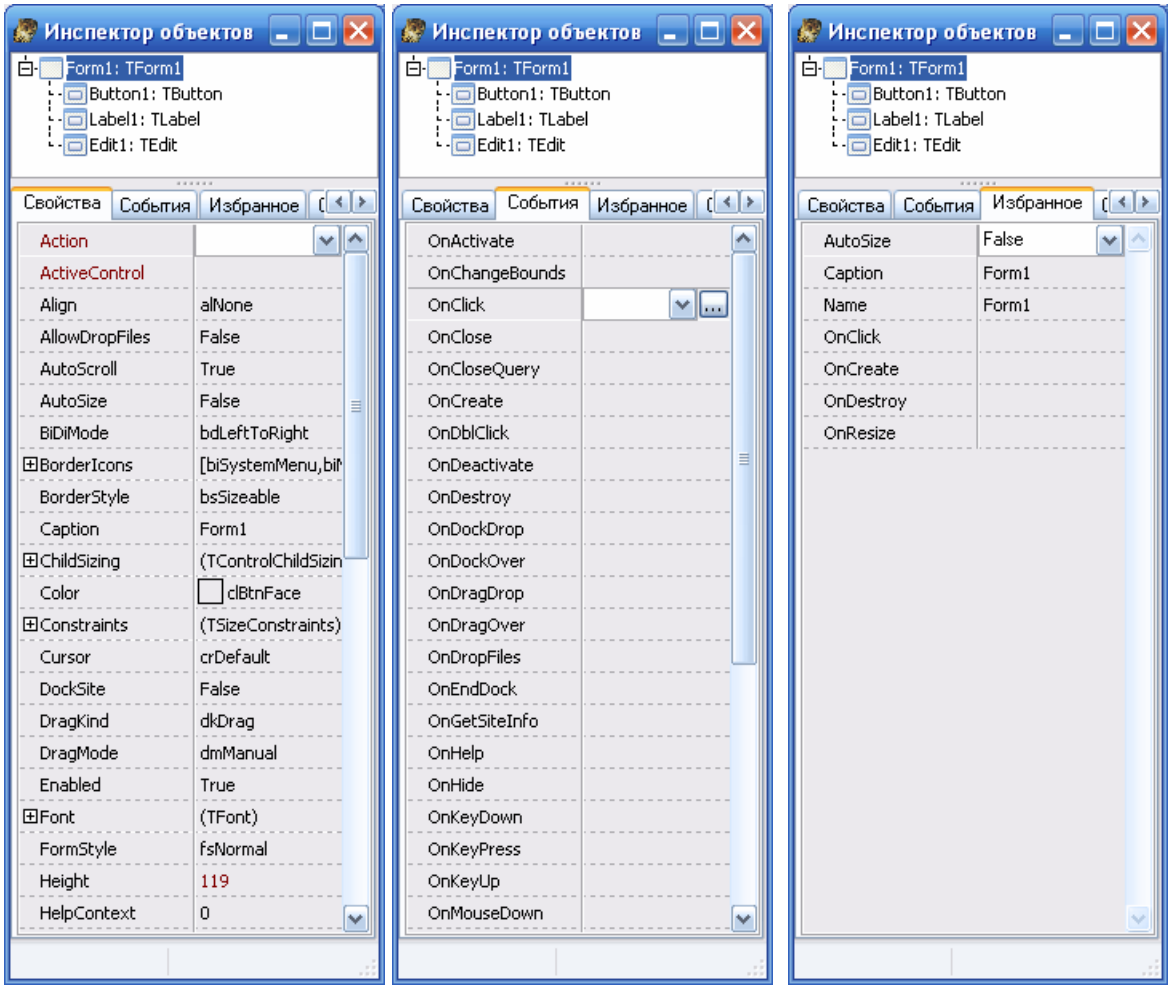
```
TForm1 = class(TForm)
    Button1: TButton;
    Edit1: TEdit;
    Label1: TLabel;
end;
```

Потім створюється змінна – об'єкт Form1 типу TForm1. Реалізаційна частина модуля поки порожня, ініціалізація передбачає включення коду з неіснуючого поки файла «unit1.lrs».

Інспектор об'єктів складається з чотирьох сторінок, кожна з яких відіграє свою роль при визначенні властивостей і поведінки даної компоненти. Перша сторінка – це список властивостей (рис. 3.6, *а*), друга – список подій (рис. 3.6, *б*), третя – вибрані властивості та події (рис. 3.6, *в*), четверта – обмеження. Використовуючи контекстне меню (викликається правою кнопкою миші) при наведенні на певну властивість або подію, можна додати його в «Вибране» або видалити звідти.

Якщо потрібно змінити що-небудь, пов'язане з певною компонентою, то зазвичай це робиться саме в *Інспекторі об'єктів*. Наприклад, можна змінити напис і розмір компоненти Label1, змінюючи властивості Caption, Left, Top, Height і Width.

Сторінка подій пов'язана з *Редактором*; якщо двічі клацнути мишкою за значенням якої-небудь події, то відповідний цій події код автоматично запишеться в *Редактор*, сам *Редактор* негайно отримає фокус, і ви відразу ж маєте змогу додати код обробника даної події.



а) *б)* *в)*

а – «Властивості»; б – «Події»; в – «Вибране»

Рисунок 3.6 – Інспектор об'єктів

Наприклад, нам потрібно зробити так, щоб після натискання кнопки введений в поле тексту рядок відобразився в текстовій мітці. Для цього спочатку зробимо активним вибраний об'єкт: або клацнемо мишею по кнопці на формі, або у верхньому віконці вибору інспектора об'єктів виберемо Button1. Потім на сторінці подій знаходимо подію «клацання» (OnClick) і два рази клацнемо лівою кнопкою миші по порожньому віконцю праворуч від нього. У редакторі текстів автоматично з'явиться таке:

```
procedure TForm1.Button1Click(Sender: TObject);
begin

end;
```

Все, що нам залишається, – це вписати між begin і end необхідну дію:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
        Label1.Caption := Edit1.Text
end;
```

Редактор оснащений зручним інструментом під назвою CodeTools: якщо при наборі точки після імені об'єкта підвести до неї покажчик миші і секунду почекати, то спливає меню зі списком усіх доступних властивостей і методів цього об'єкта (рис. 3.7).

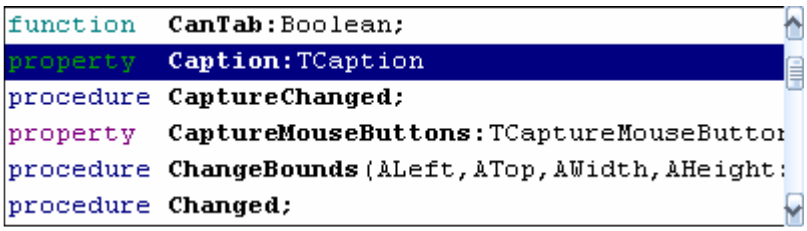


Рисунок 3.7 – Список властивостей і методів об'єкта Label

Після запуску на виконання (F9 або «Запуск» в меню) отримаємо спочатку повідомлення у відповідному вікні «Проєкт успішно зібраний», а потім – результат, поданий на рисунку 3.8. Усе, що б ми не ввели до вікна тексту, після натискання на кнопку буде скопійовано в текстову мітку.

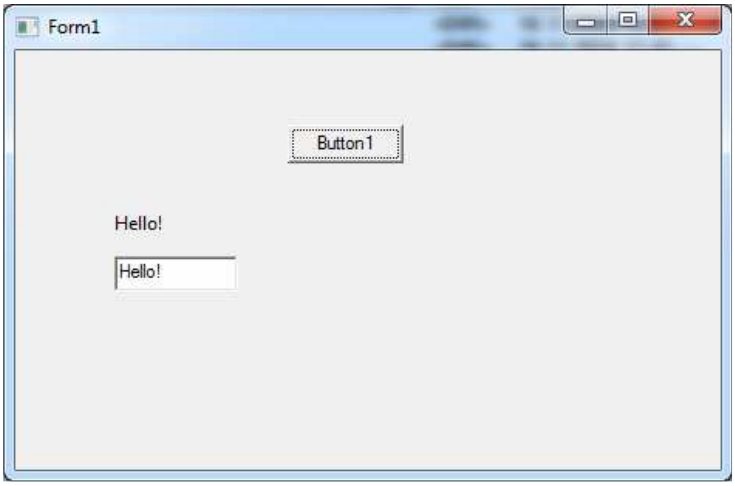


Рисунок 3.8 – Результат виконання проєкту

Якщо вибрати одразу декілька компонент, то вміст *Інспектора об'єктів* зміниться: він буде показувати тільки ті поля, які є загальними для об'єктів. Це означає те, що зроблені зміни у властивостях позначаться не на один, а на всі вибрані об'єкти.

Проєкт додатку містить, як правило, такі файли:

1) головний файл проєкту (LPR);
2) опис середовища проєкту у xml-вигляді (LPI);
3) модуль першої форми (PAS);
4) опис першої форми (LFM);
5) файл ресурсів першої форми (LRS).

Після першого вдалого запуску додатку з'являються ще кілька файлів:

1) виконуваний файл додатка (EXE);
2) параметри компіляції у xml-вигляді (COMPILED);
3) проміжний код проєкту (O);

4) відкомпільований модуль першої форми (PPU);

5) проміжний код модуля першої форми (O).

Таким чином, з проєктом, який містить одну форму, пов'язано 10 файлів. У разі двох і більше форм кількість файлів PAS-LFM-PPU збільшується відповідно. Виходячи з цього, настійно рекомендується створювати окремі папки для кожного свого проєкту, навіть невеликого!

При завантаженні і збереженні слід пам'ятати, що ці операції треба проробляти над головним файлом проєкту (LPR), використовуючи пункти меню «Відкрити проект» і «Зберегти все».

Докладний розгляд Палітри компонент наведений у навчальному посібнику «Робота в середовищі Lazarus», тому далі буде розглядатися винятково процес програмування в середовищі Lazarus.

3.2 Програмування в середовищі Lazarus

Розроблення програми в середовищі Lazarus є, по суті, розміщенням на формі (проєкт майбутнього додатка) потрібних компонент і настроювання їх властивостей. Для простого додатка цим можна обмежитися, але серйозні, т. зв. повнофункціональні додатки, вимагають більш серйозного підходу. Необхідно знати особливості мови Object Pascal, вміти користуватися спеціальною властивістю Canvas, створювати і знищувати компоненти в процесі роботи програми тощо.

Мова програмування Object Pascal є логічним продовженням створеної Ніклаусом Віртом на початку 70-х років XX століття мови Паскаль. Згідно з одними джерелами, вона створювалась програмістами фірми Borland саме для першої версії Delphi (1995), інші стверджують, що вона народжена у надрах Apple у 1986 році (тобто майже за 10 років до появи візуального програмування). Головною відмінністю «новонародженої» від своїх предків стало зняття більшості обмежень у програмі. Так, наприклад, програмісту тепер не треба думати про межу сегмента даних до 64 кілобайт, про максимальний розмір масиву і довжину рядка та багато іншого – усе це за нього зробить компілятор, що раціонально використовує доступну оперативну пам'ять. Ще низка нововведень була запозичена зі створеної Б. Страуструпом мови C++.

Одне з нововведень – «замовчувані» параметри підпрограм. У звичайному Паскалі заголовок процедури має вигляд

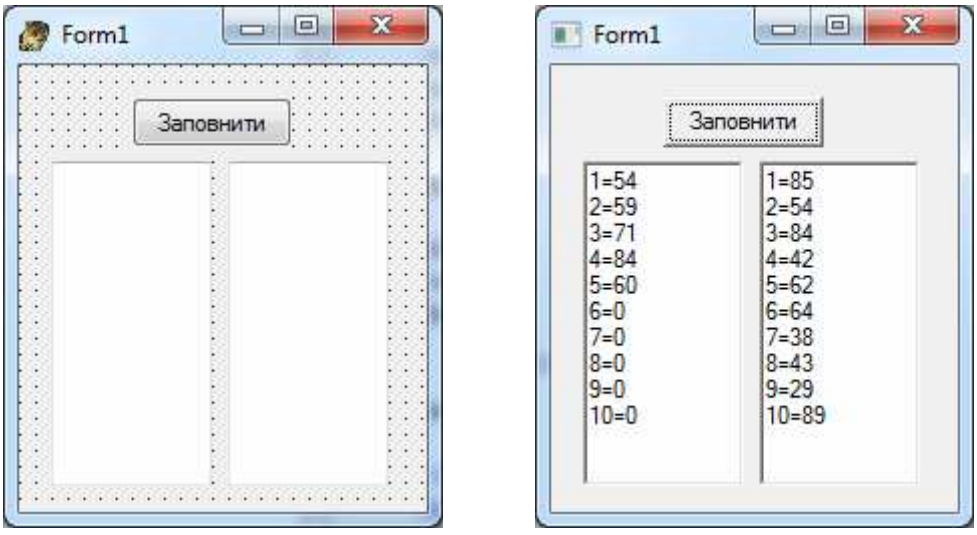
Procedure *ім'я (список формальних параметрів);*

При зверненні до підпрограми список фактичних параметрів має точно відповідати своїм формальним прототипам. Object Pascal, подібно C++, дозволяє опускати кілька (або навіть усі) фактичні параметри, при цьому в заголовку підпрограми мають бути вказані їх значення за замовчуванням:

Procedure *vvod (**var** a: mas; n: integer=10);*

Якщо тепер десь у програмі звернутися до підпрограми як vvod (a), то за замовчуванням буде введений масив з 10 елементів. У наведеному нижче прикладі (рис. 3.9) розглядаються два масиви: в одному заповнюються 5 елементів, у другому – усі 10 (за замовчуванням).

```
type mas=array[1..10] of integer;
var m1,m2:mas;
Procedure vvod (var a: mas; n: integer=10);
var i:integer;
begin
    for i:=1 to n do a[i]:=random(100)
end;
Procedure TForm1.Button1Click(Sender: TObject);
var i:integer;
begin
    vvod(m1,5);
    Memo1.Lines.Clear;
    for i:=1 to 10 do Memo1.Lines.Add(IntToStr(i)+'='+IntToStr(m1[i]));
    vvod(m2);
    Memo2.Lines.Clear;
    for i:=1 to 10 do Memo2.Lines.Add(IntToStr(i)+'='+IntToStr(m2[i]))
end;
```



а) *б)*

а – дизайн, б – робота

Рисунок 3.9 – Уведення і виведення двох масивів

Відкритий масив являє собою формальний параметр підпрограми, що описує базовий тип елементів масиву, але не визначає його розмірності та межі:

Procedure *Output (a:* ***array of*** *Integer);*

Усередині цієї процедури масив трактується як одномірний з нижнім індексом, який дорівнює нулю, і верхнім, який повертається функцією High. Змінимо попередній приклад: нехай два масиви будуть описані як такі, що мають різний тип даних, а процедура Output зможе виводити значення заданого масиву в задане Memo-поле (рис. 3.10).

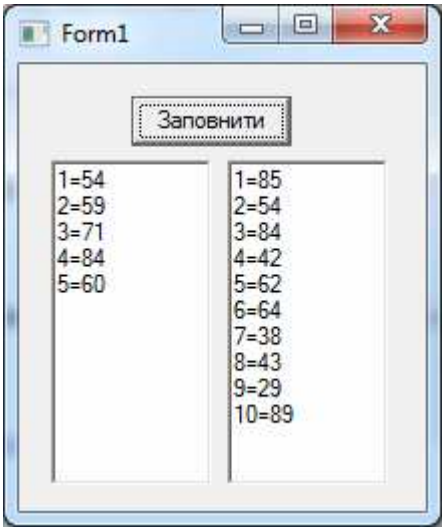


Рисунок 3.10 – Уведення і виведення двох масивів

Необхідно також трохи відкоригувати процедуру введення.

```
type mas=array[1..10] of integer;
    mas5=array[1..5] of integer;
var m1:mas5;m2:mas;

Procedure vvod (var a: array of Integer; n: integer=10);
var i:integer;
begin
    for i:=1 to n do a[i-1]:=random(100)
end;

Procedure Output (a: array of Integer; Memo: TMemo);
var i:integer;
begin
    Memo.Lines.Clear;
    For i:=0 to High(a) do
                Memo.Lines.Add(IntToStr(i+1)+'='+IntToStr(a[i]))
```

```
end;

Procedure TForm1.Button1Click(Sender: TObject);
var i:integer;
begin
    vvod(m1,5);
    Output(m1,Memo1);
    vvod(m2);
    Output(m2,Memo2);
end;
```

Для передавання двомірних масивів використовуються т. зв. варіантні дані (див. далі). Відкритий масив можна використовувати і незалежно від підпрограм: довжина такого масиву задається функцією SetLenght (ім'я, довжина). Зрозуміло, що використання цієї функції повинно передувати будь-якому зверненню до масиву.

Об'єктна модель, використовувана в Object Pascal, відрізняється від аналогічної в Turbo Pascal і багато в чому запозичена у C++. Об'єктний тип тут називається класом (class), а об'єкт являє собою конкретний екземпляр реалізації класу. На відміну від C++, тут не допускається множинне спадкування, тобто будь-який клас може мати рівно одного попередника. Не буває класів, які не мають у роді клас TObject, запис MyClass = class (TObject) ідентичний MyClass = class.

Також у Object Pascal розрізняються поняття ***поле*** і ***властивість***. Полями називаються дані, що містяться в класі, а властивість з'єднує поля з відповідними методами, які повинні використовуватися при вилученні з нього даних або запису в нього. Наприклад:

```
type
    MyClass = class
        CenterCoord: Integer;
        Function GetCenter: Integer;
        Procedure SetCenter (V: Integer);
        Property Center: Integer read GetCenter write SetCenter;
    end;
```

Тут запис «Center: = 2» автоматично викличе процедуру SetCenter, а «k: = Center» – функцію GetCenter.

Тип Variant уведений для тих випадків, коли ми не можемо напевно вирішити, якого типу дані будуть використовуватися у виразі або в якості параметрів при виклику підпрограм. Після присвоєння полю VType якого-небудь значення, відмінного від 0 (varEmpty), з варіантною змінною можна працювати, як зі звичайною змінною певного типу (varString, varInteger тощо). З практичної точки зору цікаві два шляхи використання варіантів – для роботи з варіантними масивами і OLE-об'єктами.

Відмітна особливість варіантного масиву (varArray) полягає в тому, що його елементи можуть мати різний тип даних. У наступному прикладі (рис. 3.11) створюється одномірний варіантний масив з 4 елементами різних типів:

```
procedure TForm1.Button1Click(Sender: TObject);
var V:Variant; i:integer;
begin
    V:=VarArrayCreate([0,3],varVariant);
    V[0]:=1;              // Цілий
    V[1]:=1.345;          // Реальний
    V[2]:='Привет!';      // Рядковий
    V[3]:=True;           // Логічний
    Memo1.Lines.Clear;
    for i:=0 to 3 do Memo1.Lines.Add(VarToStr(V[i]));
end;
```

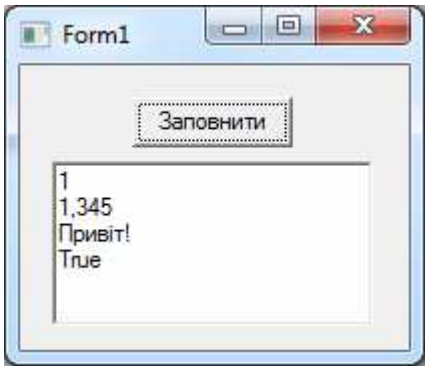


Рисунок 3.11 – Виведення масиву варіантів

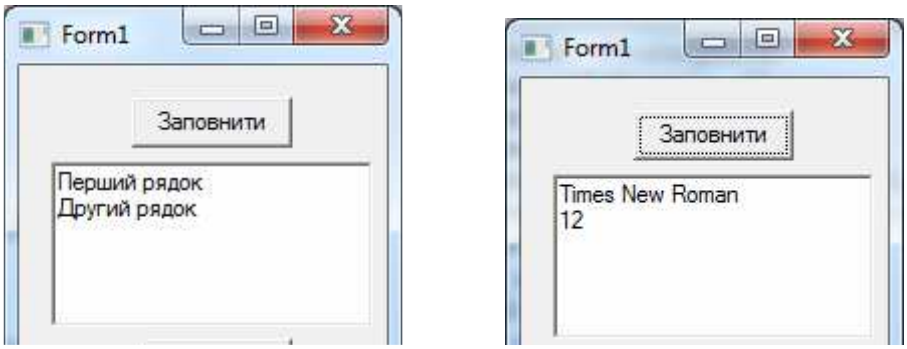
При роботі з варіантними масивами потрібно пам'ятати, що відлік елементів починається з нуля. Також не забудьте в списку модулів дописати Variants. У таблиці 3.1 наведені основні підпрограми для роботи з варіантними масивами.

Таблиця 3.1 – Підпрограми для роботи з варіантними масивами

Підпрограма	Виконувана дія
function VarArrayCreate (**const** Bounds: **array of** Integer; VarType: Integer): Variant;	Створює варіантний масив з елементів типу VarType з кількістю і межами вимірювань, що вказуються параметром Bounds
function VarArrayDimCount (**const** A: Variant) : Integer;	Повертає розмірність масиву A чи 0, якщо A – не масив
function VarArrayHighBound (**const** A: Variant; Dim: Integer) : Integer;	Повертає верхню межу індексу масиву A з вимірювання Dim
function VarArrayLowBound (**const** A: Variant; Dim: Integer) : Integer;	Повертає нижню межу індексу масиву A з вимірювання Dim
function VarArrayOf (**const** Values: **array of** Variant): Variant;	Створює одномірний варіантний масив за переліком значень, що містяться у відкритому масиві Values. Нижня межа при цьому дорівнює 0
function VarArrayRedim (**var** A: Variant; HighBound: Integer) : Integer;	Змінює верхню межу індексу масиву A на HighBound

У наступному прикладі ілюструються можливості варіантів для роботи з OLE-об'єктом Word (обов'язково потрібно в списку модулів дописати ComObj). Процедура запускає Word і виводить у новий документ вміст Memo-поля, після чого виводить у це саме поле назву і розмір шрифту (рис. 3.12):

```
procedure TForm1.ExportToWord1Click(Sender: TObject);
var MsWord:Variant;
begin
    MsWord:=CreateOleObject('Word.Application');
    MSWord.Documents.Open(FileName:='c:\111.doc');
    MSWord.Documents.Add;
    MsWord.Selection.TypeText(UTF8Decode(Memo1.Text));
    Memo1.Lines.Clear;
    Memo1.Lines.Add(MsWord.Selection.Font.Name);
    Memo1.Lines.Add(IntToStr(MsWord.Selection.Font.Size));
end;
```



а) *б)*

а – до завантаження, б – після

Рисунок 3.12 – Робота з текстовим процесором

Важливими відмінностями від реалізації цього підходу в середовищі Delphi є необхідність попереднього відкриття існуючого документа, обов'язкового декодування виведеного тексту (за замовчуванням він зберігається в кодуванні Unicode) і неможливість присвоєння значень ряду властивостей (наприклад, назва і розмір шрифту мають статус «тільки для читання»).

На жаль, робота с процесором електронних таблиць Excel у середовищі Lazarus не настільки проста, як у середовищі Delphi. Саме звернення до Excel відбувається аналогічно попередньому прикладу:

```
MsExcel:=CreateOleObject('Excel.Application');
MsExcel.Workbooks.Open(FileName:='c:\111.xls');
```

Однак будь-які спроби звернутися до осередків листа безпосередньо, як рекомендують в довіднику:

```
MsExcel.Range('A1'):='Hello!';
```

завжди призводять до виникнення виняткової ситуації. Єдиним виходом можна вважати поділ завдань: спочатку інформація виводиться у файл

формату XLS, а потім здійснюється завантаження Excel-додатків. Даний підхід реалізований у вільно поширюваній бібліотеці Fspreadsheet.

Далі наведені часто використовувані процедури і функції для роботи з рядками і виведення повідомлень.

Function Trim (**const** S: String): String – повертає рядок з видаленими провідними пробілами.

Function StrToFloat (St: String): Extended – перетворює рядок символів St на дійсне число, при цьому рядок не повинен містити ведучих і ведених пробілів.

Function FloatToStr (Value: Extended): String – перетворює дійсне значення Value на рядок символів.

Function FloatToStrF (Value: Extended; Format: TFloatFormat; Precision, Digits: Integer): String – перетворює дійсне значення Value на рядок символів з урахуванням формату Format і параметрів Precision і Digits.

Function StrToInt (St: String): Integer – перетворює рядок символів St на ціле число, при цьому рядок не повинен містити ведучих і ведених прогалин.

Function IntToStr (Value: Integer): String – перетворює ціле число на рядок символів.

Function Format (Format: String; список параметров): String – перетворює список параметрів на рядок символів з урахуванням формату Format (аналог printf в C++), наприклад: Memo1.Lines.Add (Format (’k = %5i, d = %5.2f’,k,d));

Procedure ShowMessage (**const** Msg: String) – виводить діалогове вікно з текстом повідомлення.

Function MessageDlg (**const** Msg: String; DlgType: TMsgDlgType; Buttons: TMsgDlgButtons; HelpCtx: Longint): Word – виводить діалогове вікно із заданими функціями, наприклад:

If MessageDlg ('Ви дійсно хочете зробити це?', mtConfirmation, [mbYes,mbNo], 0) = mrYes then begin … end;

MessageDlg(У вас немає прав користуватися цією програмою!', mtError, [mbOk], 0);

У Object-Pascal змінені назви деяких функцій і змінних у модулі System:

AssignFile замість Assign;

CloseFile замість Close;

TextFile замість Text;

В іншому робота з файлами не відрізняється від аналогічної у Турбо Паскалі.

Візуальне проєктування майбутнього додатку досить зручне, проте, по-перше, часто виникають ситуації, коли заздалегідь невідомі кількість і розташування деяких компонент, а по-друге, не всі компоненти винесені на однойменну палітру.

Якщо необхідно в будь-якому місці програми створити компоненту одного з існуючих класів, це робиться такою послідовністю операторів:

```
var Ed: TType;   // Описуємо змінну
begin
        …
        Ed := TType.Create (клас-батько);  // Створюємо змінну
        (* Працюємо зі змінною *)
        Ed.Free;  // Видаляємо компоненту
end;
```

У наступному прикладі створюється масив із розташованих послідовно 10 кнопок, при натисканні на кожну з яких напис на ній змінюється (рис. 3.13, 3.14). Розміри і розташування кнопок розраховуються безпосередньо у процедурі, після чого змінюються розміри самої форми. Зверніть увагу, що кнопки створюються в момент створення форми (подія Create), а обробник події необхідно задати «вручну».

```
var SB:array [0..9] of TSpeedButton;
procedure TForm1.SBClick(Sender: TObject);
begin
   (Sender as TSpeedButton).Caption:='Ok'
end;
procedure TForm1.FormCreate(Sender: TObject);
var i:integer;
begin
   for i:=0 to 9 do begin
   SB[i]:=TSpeedButton.Create(Form1);
   with SB[i] do
   begin
      Parent:=Form1;//Підтверджуємо розташування кнопки на формі
      if i=0 then Left:=1
      else Left:=SB[i-1].Left+SB[i-1].Width; // Зрушуємо нову кнопку
      Top:=1;Width:=50;Height:=50;
      Caption:=IntToStr(i);Font.Size:=12;
      onClick:=@SBClick; // Задаємо реакцію кнопки на натискання
   end;
   end;
   Width:=SB[0].Width*10+2;
   Height:=SB[0].Height+2;
end;
procedure TForm1.FormDestroy(Sender: TObject);
var i:integer;
begin
   for i:=0 to 9 do SB[i].Free
end;
```

Рисунок 3.13 – Створення кнопок: завантаження форми

Рисунок 3.14 – Створення кнопок: натиснуті кнопки 3 і 7

Абстрактний клас TString містить поля і методи для роботи з наборами рядків. Від нього породжені численні спеціалізовані нащадки, в тому числі найпоширеніший TStringList. Ця компонента дозволяє вводити, виводити, сортувати рядки, здійснювати пошук в масиві рядків і багато іншого.

Властивість Count визначає поточну довжину масиву; індекси змінюються від 0 до Count -1. Властивість Text інтерпретує вміст масиву у вигляді одного довгого рядка; властивість CommaText – у вигляді такого самого рядка, але з розподілами типу «Перший рядок», «Другий рядок» тощо. Властивість Sorted визначає необхідність сортування вмісту в алфавітному порядку. Функція Find (const S: String; var Index: Integer): Boolean шукає в масиві рядок S і в разі успіху (True) повертає його Index. Як і більшість компонент – наборів рядків – StringList може зчитувати інформацію з текстового файлу і записувати в нього.

У наступному прикладі з файлу зчитується деяка інформація, сортується і записується в інший файл.

```
procedure TForm1.WorkWithFile;
var S:TStringList;
begin
    S := TStringList.Create;
    if FileExists('input.txt') then begin  // Перевірка наявності файлу
    S.LoadFromFile('input.txt');           // Завантажуємо
    S.Sorted := True;                      // Сортуємо
    S.SaveToFile('output.txt');            // Зберігаємо
    end;
    S.Free                                 // Звільнюємо
end;
```

До стандартної бібліотеки візуальних компонентів LCL (Lazarus Component Library) входить декілька об'єктів, за допомогою яких можна надати своїй програмі абсолютно оригінальний вигляд. Це Image (DBImage), Shape, Bevel і деякі інші.

Image дозволяє розмістити графічне зображення на будь-якому місці форми. Тут слід пам'ятати, що зображення, поміщене на форму під час дизайну, додається до EXE-файлу, що збільшує його і без того чималий об'єм. У якості одного з можливих рішень проблеми можна здійснювати завантаження картинки під час виконання програми. Для цього у властивості Picture (яка є об'єктом зі своїм набором властивостей і методів) є спеціальний метод LoadFromFile. Наприклад:

```
if OpenDialog1.Execute then
    Image1.Picture.LoadFromFile(OpenDialog1.FileName);
```

Shape – найпростіші графічні об'єкти на формі типу «коло», «квадрат» тощо. Вид об'єкта, колір і вигляд меж, колір і вигляд заповнення об'єкта можна змінювати як під час дизайну, так і під час виконання програми.

Проте ні Image, ні Shape не надає таких змог і такої свободи творчості, як спеціальна властивість Canvas (канва), яке надає простого шляху для малювання на самому об'єкті.

Canvas є, у свою чергу, є об'єктом, що об'єднує у собі поле для малювання, олівець (Pen), пензлик (Brush) і шрифт (Font). Canvas має також низку графічних методів: Draw, TextOut, Arc, Rectangle та інші. Використовуючи Canvas, можна відтворювати на формі будь-які графічні об'єкти – картинки, багатокутники, текст тощо без використання компонент Image і Shape, однак при цьому потрібно обробляти подію OnPaint того об'єкта, на канві якого ми малюємо. Розглянемо докладніше властивості та методи об'єкта Canvas.

Brush – пензлик, є об'єктом зі своїм набором властивостей:

Bitmap – картинка розміром строго 8x8, використовується для заповнення (заливання) ділянки на екрані;

Color – колір заливання;

Style – зумовлений стиль заливання; ця властивість конкурує з Bitmap: яку властивість визначили останньою, та й буде визначати вид заливання;

CopyMode – властивість, яка визначає, яким чином буде відбуватися копіювання (метод CopyRect) на дану канву зображення з іншого місця: один до одного, з інверсією зображення та ін.

Font – шрифт, яким виводиться текст (метод TextOut).

Pen – олівець, визначає вид ліній; як і пензлик (Brush) є об'єктом з набором властивостей:

Color – колір лінії;

Mode – режим виводу: проста лінія, з інвертуванням та інші;

Style – стиль виводу: лінія, пунктир та ін;

Width – ширина лінії в пікселях;

PenPos – поточна позиція олівця, олівець рекомендується переміщати за допомогою методу MoveTo, а не прямою установкою цієї властивості.

Pixels – двомірний масив елементів зображення (пікселів), з його допомогою можна отримати доступ до кожної окремої точці зображення (рис. 3.15).

```
procedure TForm1.FormPaint(Sender: TObject);
var x,y:integer;
begin
    with Canvas do
    for x:=1 to Width do
    for y:=1 to Height do Pixels[x,y]:=x*y
end;
```

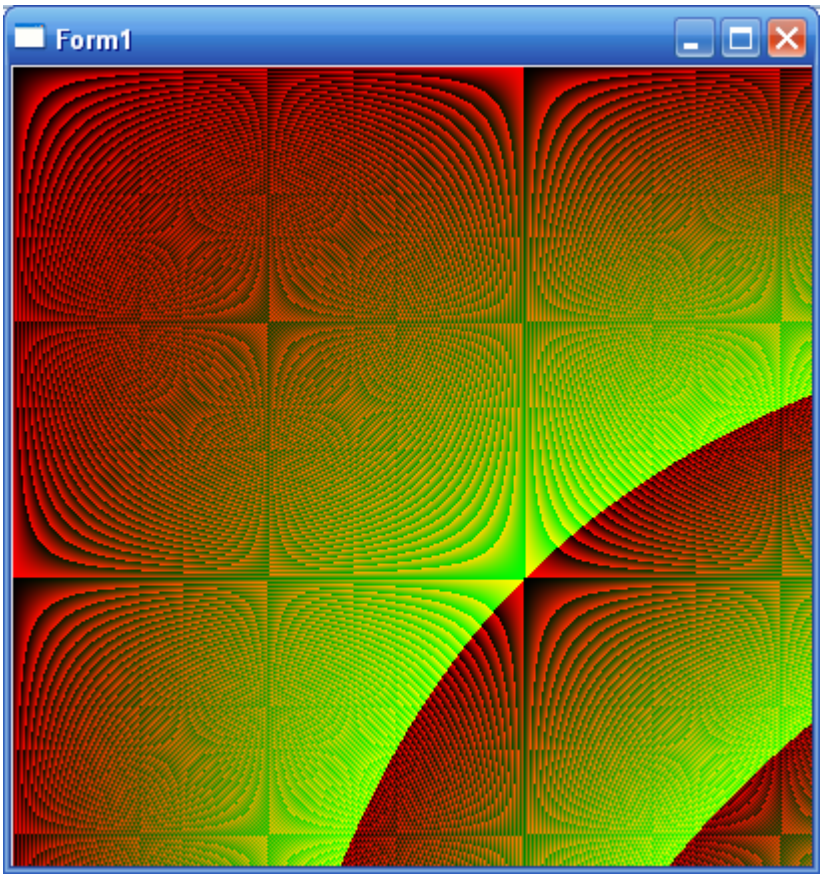


Рисунок 3.15 – Результат роботи з масивом пікселів

Змінивши формулу, наприклад, помноживши добуток координат на 100 або 1000, можна отримати абсолютно непередбачувані зображення (рис. 3.16, 3.17).

Усі методи об'єкта Canvas можна умовно поділити на три категорії.

1. Методи для малювання найпростішої графіки: Arc, Chord, LineTo, Pie, Polygon, PolyLine, Rectangle, RoundRect та інші. При промальовуванні ліній в цих методах використовуються олівець (Pen) канви, а для заповнення внутрішніх ділянок – пензлик (Brush).

2. Методи для виведення картинок на канву: Draw і StretchDraw; як параметри вказуються прямокутник і графічний об'єкт для виводу (TBitmap, TIcon або TMetafile). StretchDraw відрізняється тим, що розтягує або стискає картинку так, щоб вона заповнила весь зазначений прямокутник.

3. Методи для виведення тексту: TextOut і TextRect. При виведенні тексту використовується шрифт (Font) канви. При використанні TextRect текст виводиться тільки всередині зазначеного прямокутника. Довжину і висоту тексту можна дізнатися за допомогою функцій TextWidth і TextHeight.

Неповний перелік методів поданий у таблиці 3.2.

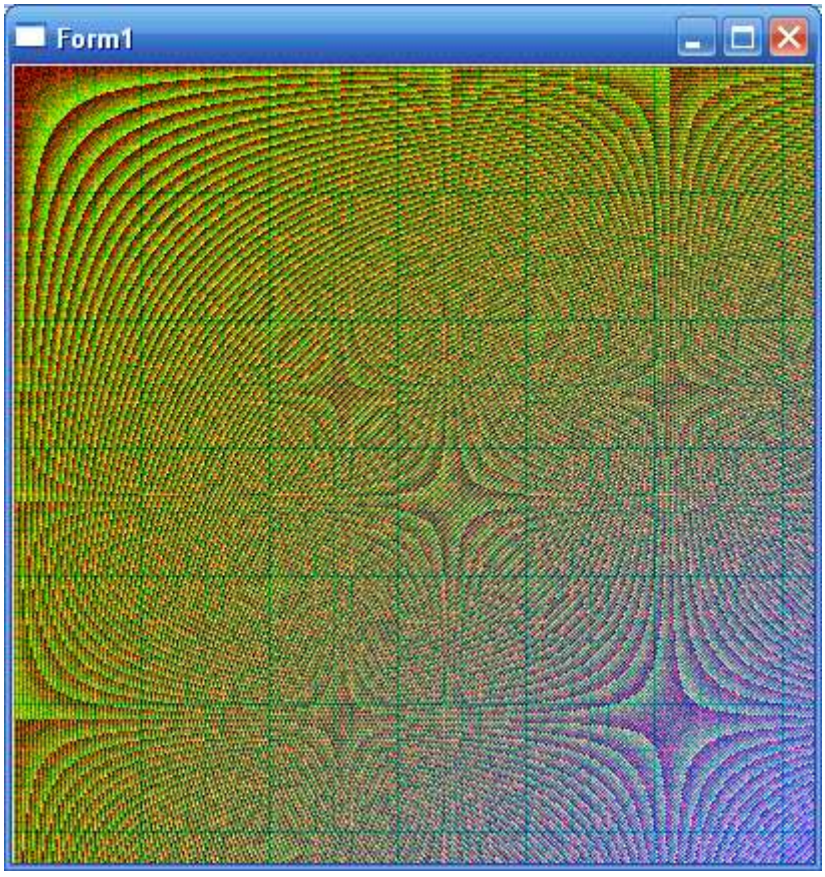


*Рисунок 3.16 – Результат роботи з масивом пікселів (*100)*

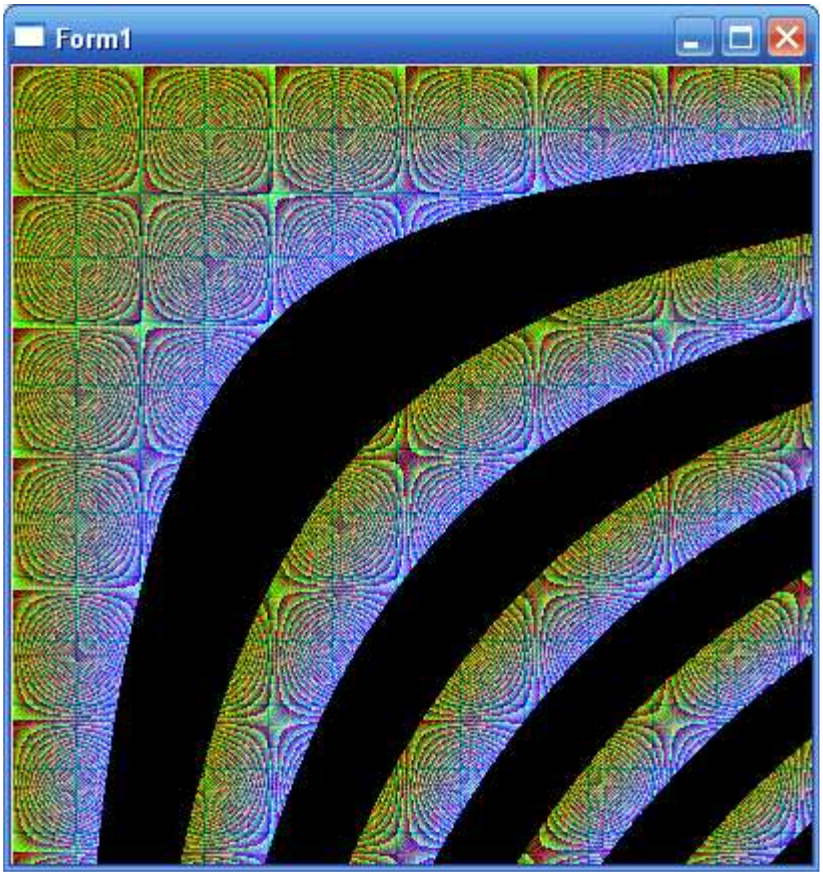


*Рисунок 3.17 – Результат роботи з масивом пікселів (*1000)*

Таблиця 3.2 – Методи класу TCanvas

Метод	**Назва**
1	*2*
Procedure Arc (X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer);	Креслить дугу еліпса в прямокутнику (X1, Y1) – (X2, Y2). Початок дуги лежить на перетині еліпса і променя, проведеного з його центра до точки (X3, Y3), а кінець – на перетині з променем із центра до точки (X4, Y4). Дугу креслить проти годинникової стрілки
procedure BrushCopy (const Dest: TRect; Bitmap: TBitmap; const Source: Trect; Color: TColor);	Копіює частину зображення *Source* на ділянку канви *Dest*. *Color* вказує колір у *Dest*, який повинен замінюватися на колір пензлика канви. Метод введений для сумісності з ранніми версіями Delphi. Замість нього слід користуватися класом *TImageList*
procedure Chord (XI, Yl, X2, Y2, X3, Y3, X4, Y4: Integer);	Креслить сегмент еліпса в прямокутнику (X1,Y1)–(X2,Y2). Початок дуги сегмента лежить на перетині еліпса і променя, проведеного з його центра до точки (X3, Y3), а кінець – на перетині з променем із центра до точки (X4, Y4). Дугу сегмента креслить проти годинникової стрілки, а початкова та кінцева точки дуги з'єднуються прямою
procedure CopyRect (Dest: TRect; Canvas: TCanvas; Source: TRect);	Копіює зображення *Source* канви *Canvas* у ділянку *Dest* поточної канви. При цьому різноманітні спеціальні ефекти досягаються за допомогою властивості *CopyMode*

Продовження таблиці 3.2

1	*2*
procedure Draw(X, Y: Integer; Graphic: Tgraphic);	Здійснює промальовування графічного об'єкта *Graphic* так, щоб лівий верхній кут об'єкта розташувався в точці (X, Y)
procedure DrawFocusRect (const Rect: TRect);	Промальовує прямокутник за допомогою операції XOR, тому повторне промальовування знищує раніше накреслений прямокутник. Використовується, в основному, для промальовування нестандартних інтерфейсних елементів при отриманні ними фокусу введення і при втраті його
procedure Ellipse (X1,Y1,X2,Y2: Integer);	Креслить еліпс у прямокутнику (X1, Y1) – (X2, Y2). Заповнює внутрішній простір еліпса поточним пензлем
procedure FillRect (const Rect: TRect);	Заповнює поточним пензлем прямокутну ділянку *Rect*, включаючи її ліву та верхню межі, але не зачіпаючи праву і нижню межі
procedure FloodFill (X, Y: Integer; Color: TColor; FillStyle: TfillStyle);	Заливає канву поточним пензлем. Заливання починається з точки (X, Y) і поширюється в усі сторони від неї. Якщо *FillStyle = fsSurface*, заливання поширюється на всі сусідні точки з кольором *Color*. Якщо *FillStyle = fsBorder*, навпаки, заливання припиняється на точках з цим кольором
procedure FrameRect (const Rect: TRect);	Окреслює межі прямокутника *Rect* поточним пензлем товщиною в 1 піксель без заповнення внутрішньої частини прямокутника.
procedure LineTo(X, Y: Integer) ;	Креслить лінію від поточного положення пера до точки (X, Y)
procedure Lock;	Блокує канву в багатопоточних додатках для запобігання використанню канви в інших потоках команд.
procedure MoveTo(X, Y: Integer) ;	Переміщує перо в положення (X, Y) без викреслювання ліній
procedure Pie (X1, Y1, X2, Y2, X3, Y3, X4, Y4: LongInt);	Малює сектор еліпса в прямокутнику (X1, Y1) – (X2, Y2). Початок дуги лежить на перетині еліпса і променя, проведеного з його центра до точки (X3, Y3), а кінець – на перетині з променем із центра до точки (X4, Y4). Дугу креслить проти годинникової стрілки. Початок і кінець дуги з'єднуються прямими з її центром
procedure Polygon (Points: array of TPoint);	Викреслює пером багатокутник по точках, заданих у масиві *Points*. Кінцева точка з'єднується з початковою, і багатокутник заповнюється пензлем. Для креслення без заповнення використовуйте метод *Polyline*
procedure Polyline (Points: array of TPoint);	Викреслює пером ламану лінію по точках, заданих у масиві *Points*.
procedure Rectangle (X1, Y1, X2, Y2: Integer);	Викреслює і заповнює прямокутник (X1, Y1) – (X2, Y2). Для креслення без заповнення використовуйте *FrameRect* або *Polyline*
procedure Refresh;	Встановлює в канві шрифт, перо і пензлик, прийняті за замовчуванням
procedure RoundRect(X1, Y1, X2, Y2, X3, Y3 : Integer);	Викреслює і заповнює прямокутник (X1, Y1) – (X2, Y2) з округленими кутами. Прямокутник (X1, Y1) – (X3, Y3) визначає дугу еліпса для округлення кутів

Продовження таблиці 3.2

1	*2*
procedure StretchDraw (const Rect: TRect; Graphic: TGraphic);	Викреслює і при необхідності масштабує графічний об'єкт *Graphic* так, щоб він повністю зайняв прямокутник *Rect*
function TextExtent (const Text: String): Tsize;	Повертає ширину і висоту прямокутника, що охоплює текстовий рядок *Text*
function TextHeight (const Text : String): Integer;	Повертає висоту прямокутника, що охоплює текстовий рядок *Text*
procedure TextOut(X, Y: Integer; const Text: String);	Виводить текстовий рядок *Text* так, щоб лівий верхній кут прямокутника, що охоплює текст, розташовувався в точці (X, Y)
procedure TextRect (Rect: Trect; X, Y: Integer; const Text : String);	Виводить текстовий рядок *Text* так, щоб лівий верхній кут прямокутника, що охоплює текст, розташовувався в точці (X, Y). Якщо при цьому будь-яка частина напису виходить за межи прямокутника *Rect*, вона відсікається і не буде видною
function TextWidth (const Text: String): Integer;	Повертає ширину прямокутника, що охоплює текстовий рядок *Text*
function TryLock: Boolean;	Намагається заблокувати канву. Якщо канву не було блоковано іншим потоком команд, повертає *True*, в іншому випадку нічого не робить і повертає *False*
procedure Unlock;	Зменшує на 1 лічильник блокувань канви

На сторінці Additional *Палітри компонент* є об'єкт PaintBox, який можна використовувати для побудови додатку типу графічного редактора або, наприклад, у якості місця побудови графіків. Ніяких ключових властивостей, крім Canvas, PaintBox не має; власне, цей об'єкт є просто канвою для малювання. Важливо, що координати покажчика миші, передані до обробників відповідних подій (OnMouseMove та інші), є відносними, тобто це зміщення миші щодо лівого верхнього кута об'єкта TPaintBox, а не відносно лівого верхнього кута форми (рис. 3.18).

```
procedure TForm1.PaintBox1Paint(Sender: TObject);
var x,y: Integer;
begin
    with PaintBox1.Canvas do
      begin
      Brush.Color := clRed;
      Ellipse(0, 0, Width-20, Height-20);
      Font.Name := 'Arial';
      Font.Size := Height div 5;
      Font.Style := [fsBold, fsItalic];
      Font.Color := clWhite;
      X:=(Width - TextWidth('Lazarus')) div 2;
      Y:=(Height - TextHeight('L')) div 2;
      TextOut(X,Y,'Lazarus');
      end;
```

Рисунок 3.18 – Результат виконання програми

На відміну від Delphi, де низка компонент забезпечена методом Print, що дозволяє виводити вміст компоненти на принтер, у компонент середовища Lazarus такого методу немає. Для друку необхідно використовувати системні засоби – скопіювати вміст екрана, обробити його в середовищі графічного редактора і тільки потім відправляти на принтер. Однак у випадку малювання на канві є можливість вивести малюнок на друк безпосередньо з програми, скориставшись модулем Printers і наявністю в об'єкта Printer власної канви. Змінимо попередній приклад, додавши до списку модулів Printers і перенісши роботу з канвою до окремої процедури з передачею канви у вигляді параметра:

```
procedure Draw(Canvas: TCanvas);
var x,y: Integer;
begin
    with Canvas do begin
      Brush.Color:=clRed;
      Ellipse(0,0,Width-20,Height-20);
      Font.Name:='Arial'; Font.Size:=Height div 5;
      Font.Style:=[fsBold, fsItalic];
      Font.Color:=clWhite;
      X:=(Width - TextWidth('Lazarus')) div 2;
      Y:=(Height - TextHeight('L')) div 2;
      TextOut(X,Y,'Lazarus');
      end;
end;
```

```
procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
   Draw(PaintBox1.Canvas)
end;
```

Додамо новий метод – обробник події кліка мишею по малюнку:

```
procedure TForm1.PaintBox1Click(Sender: TObject);
begin
   Printer.BeginDoc;
   Draw(Printer.Canvas);
   Printer.EndDoc;
end;
```

Перший метод відкриває можливість виведення на принтер (тобто активізує об'єкт «Принтер»), потім здійснюється виведення малюнка на канву принтера, а останній метод закриває виведення на принтер, здійснюючи власне друк.

Однак слід зауважити, що модуль роботи з принтерами в Lazarus не такий універсальний, як у Delphi, і вимагає від користувача певної участі в налаштуванні операційної системи, тому використовувати наведений вище приклад як єдиний засіб друку зображення не рекомендується.

Залежно від «природи», тобто внутрішнього устрою, властивості поділяються на прості, перелічувані і вкладені.

Значеннями ***простих властивостей*** є числа або рядки. Наприклад, властивості Left і Top набувають цілих значень, що визначають положення лівого верхнього кута компоненти або форми. Властивості Caption і Name являють собою рядки і визначають заголовок та ім'я компоненти або форми.

Перелічувані властивості можуть набувати значень з визначеного набору (списку). Найпростіший приклад – це властивість типу Boolean, яка може набувати значення True або False.

Вкладені властивості підтримують вкладені значення (або об'єкти); *Інспектор об'єктів* зображує знак «+» ліворуч від назви таких властивостей. У свою чергу, поділяються на множини і комбіновані значення.

Множини в *Інспекторі об'єктів* зображуються у квадратних дужках (якщо множина порожня, вона відображається як []). Установки для вкладених властивостей виду «множина» зазвичай мають значення типу Boolean. Найбільш поширеним прикладом такої властивості є властивість Style з вкладеною множиною булевих значень.

Комбіновані значення відображаються в *Інспекторі об'єктів* як колекція деяких величин, кожна зі своїм типом даних. Деякі властивості, наприклад Font, для зміни своїх значень мають можливість викликати діалогове вікно. Для цього досить клацнути маленьку кнопку з трьома крапками в правій частині рядка *Інспектора об'єктів*, що показує дану властивість.

Lazarus дозволяє легко маніпулювати властивостями компонент як в режимі проєктування (design time), так і в режимі виконання програми (run time).

Відкритість середовища Lazarus дозволяє отримувати і оперувати інформацією особливого роду, так званою інформацією періоду виконання (RTTI – run-time type information). Ця інформація є у вигляді декількох рівнів: верхнього, середнього і нижнього. Найбільшу практичну цінність має верхній рівень RTTI, поданий як засіб перевірки та приведення типів з використаних ключових слів is і as.

Ключове слово ***is*** надає програмісту змоги визначити, чи має певний об'єкт потрібний тип або є одним із спадкоємців даного типу, наприклад, таким чином:

```
if MyObject is TSomeObj then ...
```

Є можливість використовувати RTTI і для процесу приведення об'єктного типу, використовуючи ключове слово ***as***:

```
if MyObject is TSomeObj then (MyObject as TSomeObj).MyField:=...
```

або без нього:

```
TSomeObj(MyObject).MyField:=...
```

Нижче наводиться приклад практичного використання RTTI (спочатку виводиться ім'я компоненти, потім аналізується її тип, і, в разі виявлення кнопки, виводиться напис на кнопці):

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   ShowMessage('Ім'я: '+(Sender as TComponent).Name);
   if Sender is TButton then
   ShowMessage('Напис: '+(Sender as TButton).Caption);
end;
```

Список подій для певного об'єкта, на які він реагує, можна побачити в *Інспекторі об'єктів* на сторінці подій. Імена (назви) подій, як і імена методів, прийнято утворювати відповідно до принципу т. зв. угорської нотації, який дозволяє за іменем будь-якого ідентифікатора визначити його приналежність. Наприклад, OnClick відповідає одинарному клацанню миші, OnDblClick – подвійному, а OnKeyUp – події, коли натиснута клавіша була відпущена.

Увесь набір подій для різних об'єктів з LCL можна умовно поділити на дві нерівні частини:

– події операційної системи (Click, MouseMove, KeyDown);
– події, породжувані безпосередньо в програмі (Change).

Зрозуміло, що чим складніший додаток, тим більше в ньому обробників подій другої категорії.

Структурне оброблення виняткових ситуацій – це система, що дозволяє програмісту при виникненні помилки (виняткової ситуації) зв'язатися з кодом програми, підготовленим для оброблення такої помилки. Це виконується за допомогою мовних конструкцій, які як би «охороняють» фрагмент коду програми і визначають обробники помилок, які будуть викликатися, якщо щось піде не так у «охоронюваній» ділянці коду. У даному випадку поняття виняткової ситуації відноситься до мови, і не потрібно його плутати з системними виключними ситуаціями (hardware exceptions), такими як General Protection Fault. Ці виняткові ситуації зазвичай використовують переривання і особливий стан «заліза» для оброблення критичної системної помилки; виняткові ситуації в Lazarus ж незалежні від «заліза», не використовують переривань і використовуються для оброблення помилкових станів, з якими підпрограма не готова мати справу.

При традиційному обробленні помилок, помилки, виявлені в процедурі, зазвичай передаються назовні (де було викликано процедуру) у вигляді значення, що повертається функцією, параметрів або глобальних змінних («прапорців»). Кожна викликана процедура повинна перевіряти результат виклику на наявність помилки і виконувати відповідні дії. Часто це просто вихід ще вище, до вищої процедури, наприклад: функція A викликає B, B викликає C, C виявляє помилку і повертає код помилки до B, B перевіряє повернений код, бачить, що виникла помилка, і повертає код помилки до A, A перевіряє повернений код і видає повідомлення про помилку або вирішує зробити що-небудь ще, якщо перша спроба не вдалася. Така «пожежна бригада» для оброблення помилок трудомістка, вимагає написання великої кількості коду, в якому можна легко помилитися і який важко налагоджувати.

Структурне оброблення виняткової ситуації заміщає ручне оброблення помилок автоматичною, згенерованою компілятором системою повідомлення. У наведеному вище прикладі процедура A встановила б «охорону» зі зв'язаним обробником помилки на фрагмент коду, в якому викликається B. B просто викликає C. Коли C виявляє помилку, то створює (raise) виняткову ситуацію. Спеціальний код, що був згенерований компілятором і вбудований у Run-Time Library (RTL), починає пошук обробника даної виняткової ситуації. Якщо один з обробників помилок, які використовуються в A, підходить за типом для виниклої в C виняткової ситуації, то програма переходить на його виконання. При цьому виконання B і C процедур припиняється. Якщо в A немає відповідного обробника, то пошук продовжується далі; будь-яка виняткова ситуація, що залишилася не обробленою, призведе до припинення виконання додатку. Така система називається структурною, оскільки обробка помилок визначається областю «захищеного» коду; такі області можуть бути вкладеними.

Модель виняткових ситуацій у Object Pascal є ***невідновлюваною*** (non-resumable). При виникненні виняткової ситуації ми вже не можемо повернутися до точки, де вона виникла, для продовження виконання програми (це дозволяє зробити поновлювана (resumable) модель).

Розглянемо синтаксис оброблення виняткових ситуацій. Нове ключове слово, додане до мови Object Pascal, – try. Воно використовується для позначення першої частини, т. зв. захищеної ділянки коду.

Існує два типи захищених ділянок: *try .. except* і *try .. finally*. Перший тип використовується для оброблення виняткових ситуацій. Його синтаксис:

```
try
        дія 1;
        дія 2;
        ...
        except
        on Exception1 do дія1;
        on Exception2 do дія2;
        ...
        else
        дії; {за замовчуванням – якщо жодна умова не підходить}
end;
```

Exception тут – ім'я виняткової ситуації (помилки); деякі з них будуть наведені далі.

Для впевненості в тому, що ресурси, зайняті додатком, звільняться в будь-якому випадку, можна використовувати конструкцію другого типу. Код, розташований у частині finally, виконується в будь-якому випадку, навіть якщо виникає виняткова ситуація. Відповідний синтаксис:

```
try
        дія 1;
        дія 2;
        ...
        finally
        дії; {вони виконуються завжди}
end;
```

Нижче наведена довідкова інформація з деяких наперед визначених винятків.

EConvertError – відбувається при виклику функцій StrToInt і StrToFloat, коли конвертація рядка в числовий тип неможлива.

EInOutError – відбувається при помилках введення/виведення при включеній директиві {$ I +}.

EDivByZero – викликається у разі поділу на нуль, як результат RunTime Error 200.

ERangeError – викликається при спробі звернення до елементів масиву за індексом, що виходить за межі масиву, як результат RunTime Error 201 при включеній директиві {$ R}.

EInvalidGraphic – викликається при спробі передачі в LoadFromFile файлу несумісного графічного формату.

EInvalidGraphicOperation – викликається при спробі виконання операцій, що не застосовні для даного графічного формату (наприклад, Resize для TIcon).

EListError – викликається при зверненні до елемента спадкоємця TList за індексом, який виходить за межі допустимих значень (наприклад, об'єкт TStringList містить тільки 10 рядків, а відбувається звернення до одинадцятого).

EMathError – предок винятків, що трапляються при виконанні будь-яких операцій з плаваючою точкою.

EZeroDivide – викликається в результаті ділення на нуль.

EMenuError – викликається в разі будь-яких помилок при роботі з пунктами меню для компонент TMenu, TMenuItem, TPopupMenu та їх нащадків.

EOutOfMemory – відбувається у випадку викликів New(), GetMem() або конструкторів класів при неможливості розподілу пам'яті. Відповідає RunTime Error 203.

EPrinter – викликається у разі будь-яких помилок при роботі з принтером.

EGPFault – викликається, коли відбувається «загальне порушення захисту» – General Protection Fault. Відповідає RunTime Error 216.

EInvalidImage – викликається при спробі читання файлу, що не є ресурсом, або зруйнованого файлу ресурсу.

EReadError – відбувається в тому випадку, коли неможливо прочитати значення властивості або іншого набору байт з потоку (ресурсу).

EFOpenError – викликається, коли потік не може бути відкритим.

EStringListError – відбувається при помилках роботи з об'єктом TStringList (крім помилок, оброблюваних TListError).

Тут наведені не всі існуючі винятки, проте їх може виявитися цілком достатньо для створення нормально функціонуючого додатка, який коректно реагує на будь-які неадекватні дії користувача.

У розглянутому далі прикладі (рис. 3.19… 3.21) натисканням на кнопку здійснюється складання двох цілих чисел, уведених у поля введення тексту, і результат з'являється на самій кнопці. Приклад містить обидва види захищених ділянок коду: перший відповідає за оброблення помилки переведення рядка в число (видає повідомлення і встановлює фокус введення в текстове поле), а другий розміщує на кнопці певне повідомлення (у разі успіху – суму чисел, при невдачі – слово «Error»).

```
procedure TForm1.Button1Click(Sender: TObject);
var a,b,c:integer;st:string;
begin
    st:='Error!';
    try
    try
    a:=StrToInt(Edit1.Text);
```

```
    b:=StrToInt(Edit2.Text);
    c:=a+b;   st:=IntToStr(c);
    except
    on EConvertError do
      begin
          MessageDlg('Ви ввели невірні данні!',mtError,[mbOk],0);
          Edit1.SetFocus
      end
    end; //try..except
    finally
    Button1.Caption:=st;
    end; //try..finally
end;
```

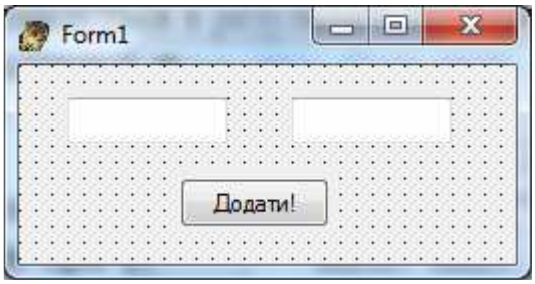


Рисунок 3.19 – Вид додатка на початку роботи

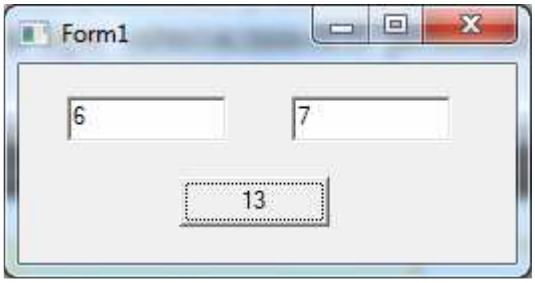


Рисунок 3.20 – Успішне виконання дій

а) *б)*

а – «Повідомлення»; б – «Вигляд додатка»
Рисунок 3.21 – Помилка введення даних

3.3 Робота з базами даних у середовищі Lazarus

Середовище Lazarus є потужним інструментом для розроблення інформаційних систем, за своїми можливостями перевершує низку існуючих систем управління базами даних. Поєднання готових компонент для роботи з базами даних з мовою Object Pascal ставить Lazarus поза конкуренцією.

Компоненти, що використовуються для роботи з базами даних, можна поділити на три групи:

– невізуальні: Dbf, SQLQuery тощо;
– візуальні: DBGrid, DBNavigator, DBEdit тощо;
– сполучні: DataSource.

Перша група включає невізуальні класи, які використовуються для управління таблицями і запитами. У палітрі компонент ці об'єкти розташовані на сторінках Data Access і SQLdb.

Друга важлива група класів – візуальні, які показують дані користувачеві і дозволяють йому переглядати і модифікувати їх. Ця група класів включає компоненти DBGrid, DBNavigator, DBEdit, DBImage, DBComboBox та інші. У Палітрі компонент ці об'єкти розташовані на сторінці Data Controls.

Є і третій тип, який використовується для того, щоб пов'язати попередні два типи об'єктів. До третього типу відноситься тільки невізуальна компонента DataSource.

Приблизна схема зв'язку між компонентами наведена на рисунку 4.1. Тут додаток містить сітку з керуючою панеллю і дві бази даних; з першою можна працювати за допомогою об'єкта Dbf1 (як встановлено за замовчуванням), з другою – як Dbf2, так і SQLQuery1. Перемикання між базами даних відбувається простою установкою властивості DataSet:

```
DataSource1.DataSet := Dbf2;
```

або

```
DataSource1.DataSet := SQLQuery1;
```

У наведеному вище прикладі до одного DataSource1 можна було додати другий і відразу пов'язати його з Dbf2 або SQLQuery1, а перемикання візуальних компонент проводити так:

```
DBGrid1.DataSource:=DataSource2;
DBNavigator1.DataSource:=DataSource2;
```

Можливі два режими роботи з базами даних: навігаційний і реляційний. Кожен з них має свої переваги і свої недоліки. Перший (компонента Dbf) розглядає базу даних як таблицю, по якій можна переміщатися в будь-які сторони (вперед, назад або по полях одного запису), і використовується, як правило, під час оброблення невеликих баз даних в індивідуальному режимі.

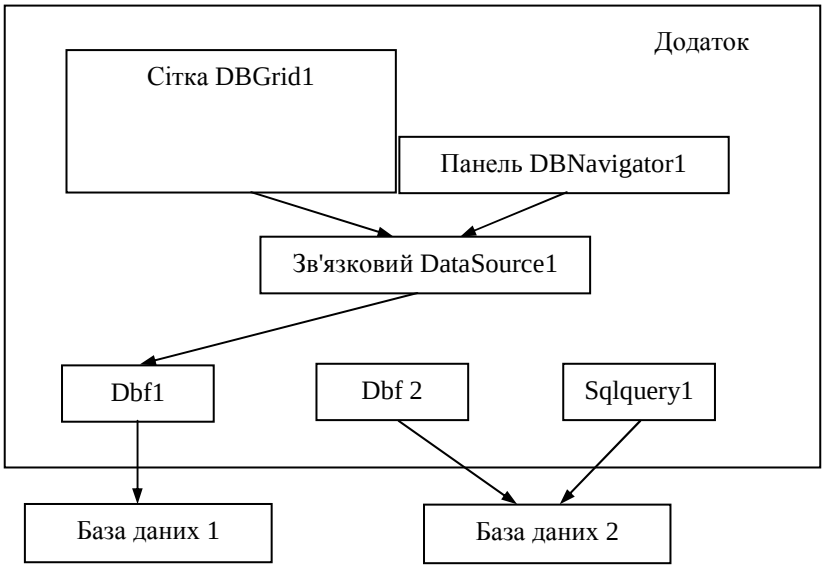


Рисунок 3.22 – Зв'язок компонент для роботи з базами даних

Під час оброблення великих баз даних навігаційний спосіб демонструє вкрай повільну швидкість роботи; в сітьовому режимі цей недолік посилюється необхідністю контролю над оновленням даних у реальному часі. Реляційний спосіб (компонента SQLQuery) заснований на використанні мови структурованих запитів SQL; результатом виконання запиту буде або яка-небудь зміна в базі даних (додавання запису, видалення, модифікація тощо), або набір даних, складений із записів, які містяться в базі даних. На жаль, використання реляційного способу в середовищі Lazarus пов'язане з необхідністю використання низки додаткових додатків, тому надалі будемо розглядати тільки навігаційний спосіб роботи з даними та пов'язані з ним компоненти.

Для роботи з базою даних необхідно, перш за все, помістити під час дизайну на форму об'єкт Dbf і вказати, з якою таблицею ми хочемо працювати: заповнити в інспектор об'єктів властивості FilePath і TableName.

У FilePath, якщо необхідно, можна вказати папку, у якій знаходяться таблиці у форматі dBase (наприклад, d:\mywork\mybase). TableName визначає ім'я файла бази даних на диску, наприклад: mybase.dbf.

Принциповою відмінністю середовища Lazarus від Delphi є відсутність можливості встановлювати формат (тип) бази даних. Для кожного формату необхідно використовувати свою компоненту. Компонента Dbf, як явно видно з її назви, працює тільки з базами даних відкритого формату «.dbf». На сторінці Data Access розташовані ще кілька компонент для роботи з іншими форматами, проте формату Paradox («.db») серед них немає.

Якщо база даних уже існує на диску, то можна на етапі дизайну властивість Active встановити у True, в іншому випадку слід передбачити кнопку «Відкрити базу даних», з якою пов'язати одну дію:

```
Dbf1.Active := True;
```

Зрозуміло, що закриття бази даних здійснюється в програмі присвоєнням Active значення False. Зміни значень цих властивостей еквівалентні викликам методів Open і Close відповідно.

Слід пам'ятати, що багато властивостей компоненти Dbf стають доступними тільки після відкриття бази даних, наприклад: якщо спробувати звернутися до властивості RecordCount (кількість записів) до переведення таблиці в активний стан, це призведе до виникнення виняткової ситуації (помилки).

Крім того, слід бути готовим до ситуації, коли таблиці, створені в додатках інших середовищ програмування, не зможуть бути прочитані в додатку, створеному в середовищі Lazarus.

Для «ручного» завдання полів бази даних виберемо в *Інспекторі об'єктів* редагування властивості FieldDefs компоненти Dbf (рис. 3.23).

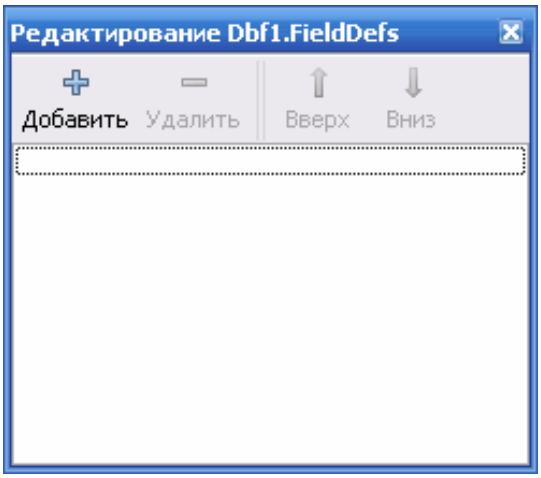
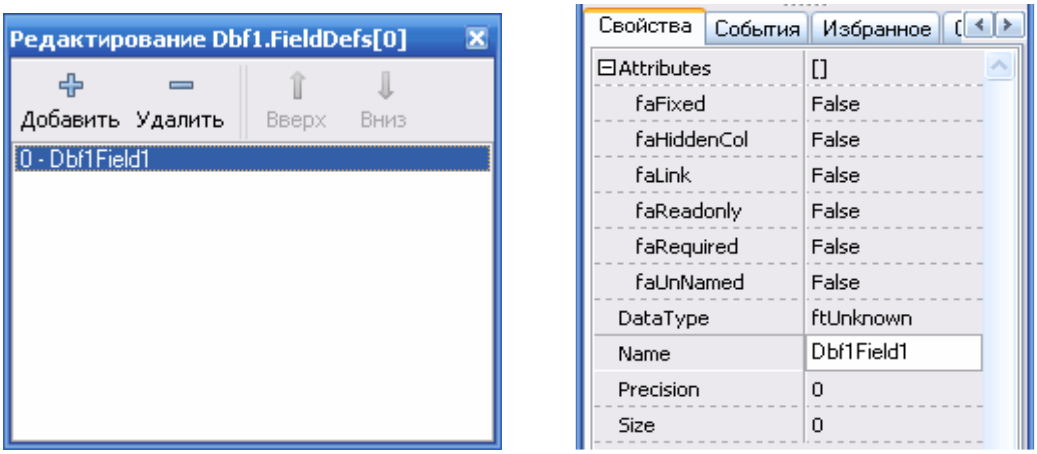


Рисунок 3.23 – Редактор полів бази даних

Вибір пункту «Додати» призведе до вставки нового поля з ім'ям Dbf1Field1 і властивостями, приведеними на панелі Інспектора об'єктів (рис. 3.24).

Задамо такі значення властивостям: Name – «Fam», DataType – «ftString», Size – «20», faRequired – «True». Аналогічно введемо інші поля: Name (Ім'я, ftString), GroupName (назва групи, ftString), DR (дата народження, ftDate), SR (середній рейтинг, ftFloat). Поле faRequired скрізь залишимо зі значенням «False» (тобто всі поля, крім прізвища, можуть тимчасово бути заповнені).

Для створення dbf-файла можна також скористатися одним з допоміжних додатків для роботи з базами даних (Database Desktop) або набрати дані в Excel-таблиці, а потім вибрати пункт «Зберегти як / dBase III +». В останньому випадку перший рядок буде списком імен полів, а типи даних визначаються автоматично.



а) *б)*

а) – «Редактор»; б) – «Властивості»
Рисунок 3.24 – Додавання нового поля

Припустимо, що у нас на диску вже існує створена в попередньому пункті база даних. Обширний набір методів і властивостей DataSet забезпечують все, що потрібно нам для доступу до будь-якого конкретного запису всередині таблиці:

```
procedure First;
procedure Last;
procedure Next;
procedure Prior;
property BOF: Boolean read FBOF;
property EOF: Boolean read FEOF;
procedure MoveBy (Distance: Integer);
```

Дамо короткий огляд їх функціональних можливостей:

– Dbf1.First переміщує поточний покажчик до першого запису таблиці;

– Dbf1.Last переміщує поточний покажчик до останнього запису таблиці;

– Dbf1.Next переміщує поточний покажчик на один запис вперед;

– Dbf1.Prior переміщує поточний покажчик на один запис назад;

– Dbf1.BOF перевіряє, чи не перебуває поточний покажчик на початку таблиці;

– Dbf1.EOF перевіряє, чи не перебуває поточний покажчик у кінці таблиці;

– Dbf1.MoveBy(N) переміщує поточний покажчик на N записів вперед або назад у таблиці. Немає ніякої функціональної відмінності між запитом Dbf1.Next і викликом Dbf1.MoveBy (1). Аналогічно, виклик Dbf1.Prior має той же самий результат, що і виклик Dbf1.MoveBy (–1).

Щоб почати використовувати ці навігаційні методи, потрібно помістити компоненти Dbf, DataSource і DBGrid на форму, приєднати DBGrid1 до DataSource1, а DataSource1 – до Dbf1. Потім встановити властивості таблиці:

FilePath – залишити порожнім (за замовчуванням таблиця розташовується в поточному каталозі);

TableName – вписати ім'я (назву) таблиці (наприклад, mybase.dbf).

Переміщатися по таблиці можна як за допомогою клавіш управління в компоненті DBGrid, так і програмним шляхом. Нижче наводиться приклад програми (рис. 3.25…3.27), яка містить всі перелічені вище компоненти, а також дозволяє виконувати навігацію програмним шляхом за допомогою низки кнопок.

```
procedure TForm1.ButtonOpenClick(Sender: TObject);
begin
   Dbf1.FilePath:='';
   Dbf1.TableName:='mybase.dbf';
   Dbf1.Open
end;

procedure TForm1.ButtonCloseClick(Sender: TObject);
begin
   Dbf1.Close
end;

procedure TForm1.ButtonFirstClick(Sender: TObject);
begin
   Dbf1.First
end;

procedure TForm1.ButtonLastClick(Sender: TObject);
begin
   Dbf1.Last
end;

procedure TForm1.ButtonNextClick(Sender: TObject);
begin
   if not Dbf1.EOF then Dbf1.Next
end;
procedure TForm1.ButtonPriorClick(Sender: TObject);
begin
   if not Dbf1.BOF then Dbf1.Prior
end;
```

Типове навігаційне оброблення бази даних має такий вигляд (передбачається, що набір даних уже відкритий):

```
Dbf1.First;          // Перенеслися на початок таблиці
while not Dbf1.EOF do begin          // Поки не дійдемо до кінця
      DoSomething;                   // … щось робимо
      Dbf1.Next;                     // Переходимо на запис вперед
end;
```

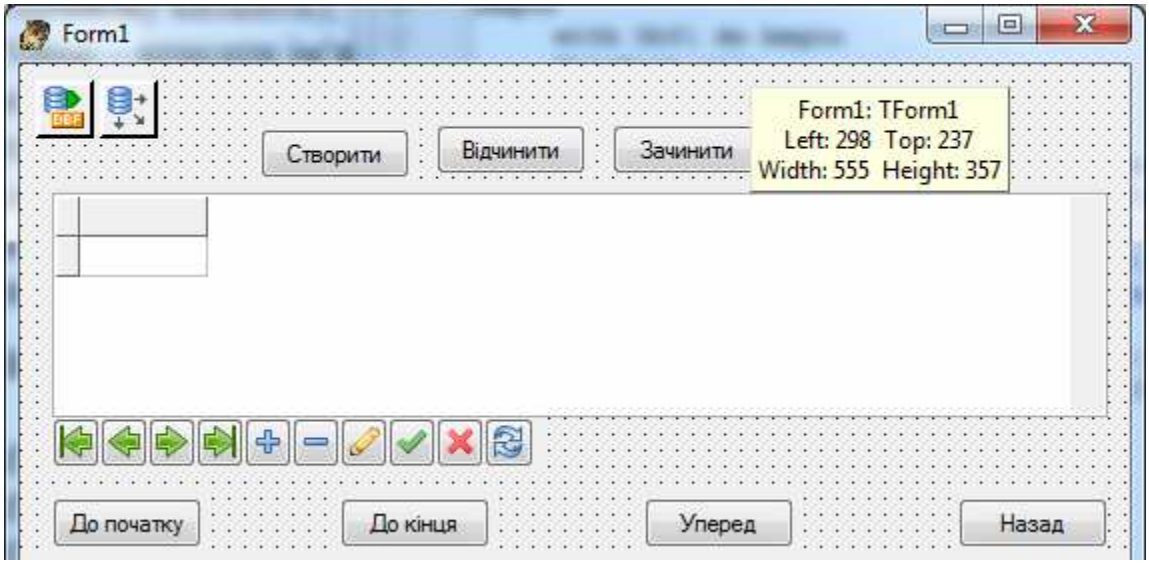


Рисунок 3.25 – Вигляд додатка на этапі дизайну

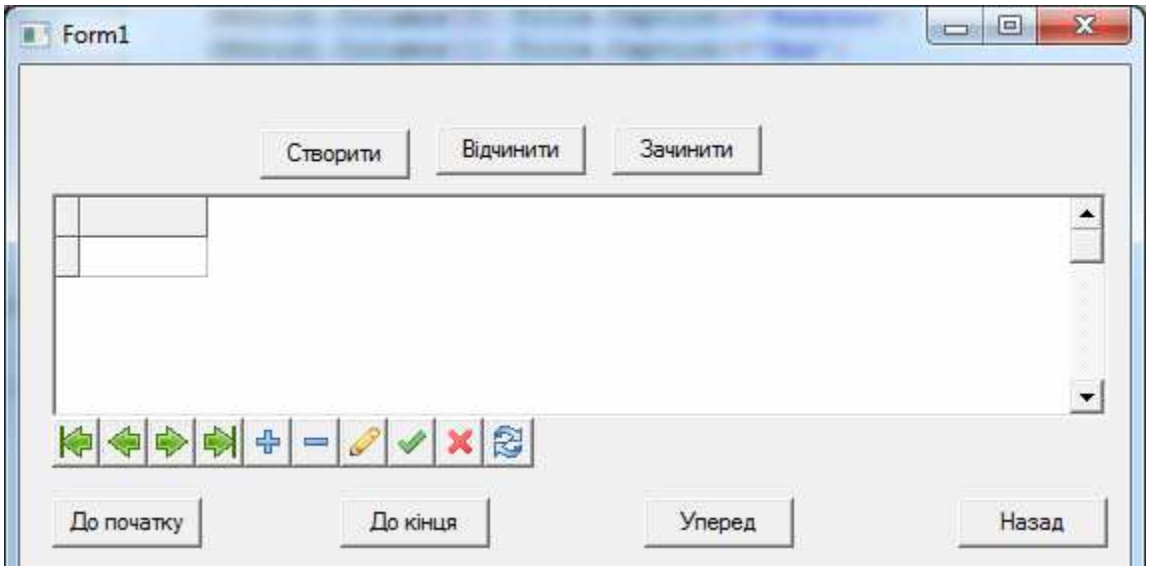


Рисунок 3.26 – Початок роботи додатка, база даних ще не відкрита

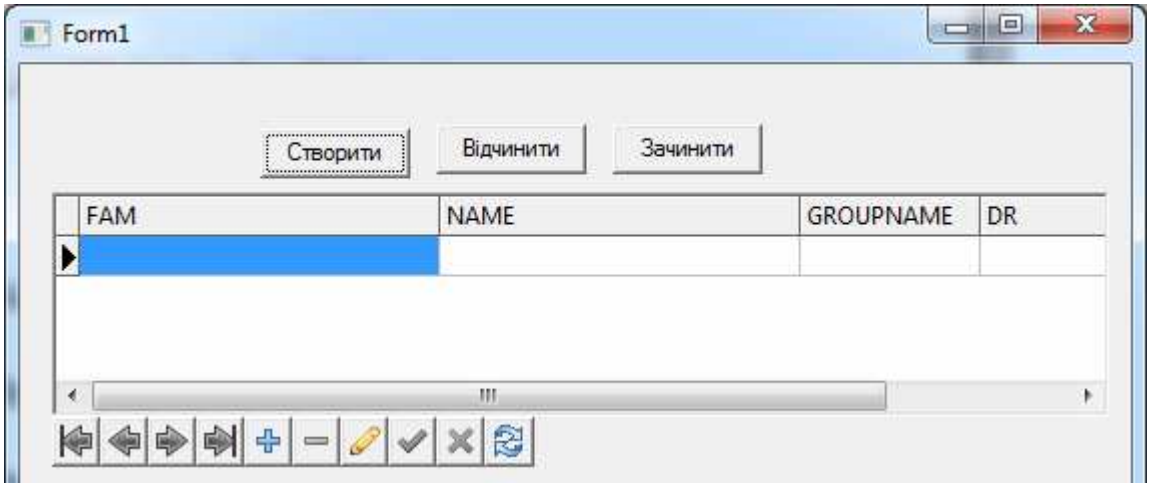


Рисунок 3.27 – Робота додатка після натискання кнопки «Відкрити»

Часто припустима помилка у випадках, подібних цьому: здійснюється вхід до циклу while або repeat, але Dbf1.Next пропущений:

```
while not Dbf1.EOF do begin DoSomething end;
```

Оскільки немає переходу на наступний запис, нескінченно буде оброблятися поточний запис – із циклу можна вийти, тільки вручну перервавши виконання програми.

У більшості випадків, коли нам потрібно отримати доступ з програми до полів запису, можна використовувати одну з таких властивостей або методів, що належать DataSet:

```
property FieldCount;
property Fields[Index: Integer];
function FieldByName(const FieldName: string): TField;
```

Властивість FieldCount повертає кількість полів до поточної структури запису. Для доступу до полів використовуються масив Fields і функція FieldByName. Принципова відмінність між ними полягає в тому, що Fields шукає потрібне поле за його порядковим номером у структурі даних, а FieldByName – за його іменем.

У будь-який час роботи програми можна отримати доступ до імен полів, наприклад: S: = Fields [0]. FieldName присвоює рядку S ім'я першого поля. Нижче наводиться приклад, у якому всі поля таблиці виводяться в Memo-поле (рис. 3.28):

```
procedure TForm1.Button1Click(Sender: TObject);
var i:integer;
begin
    Memo1.Clear;
    for i:=1 to Dbf1.FieldCount do
    Memo1.Lines.Add (Dbf1.Fields[i-1].FieldName)
end;
```

Якщо потрібно прочитати поточне утримання конкретного поля конкретного запису, то використовуються спеціальні властивості об'єкта Field:

```
S := Fields[0].AsString;
```

Властивість Fields (так само, як і метод FieldsByName) дозволяє вибрати тип результату, використовуючи такі характеристики:

```
property AsBoolean
property AsFloat
property AsInteger
property AsString
property AsDateTime
```

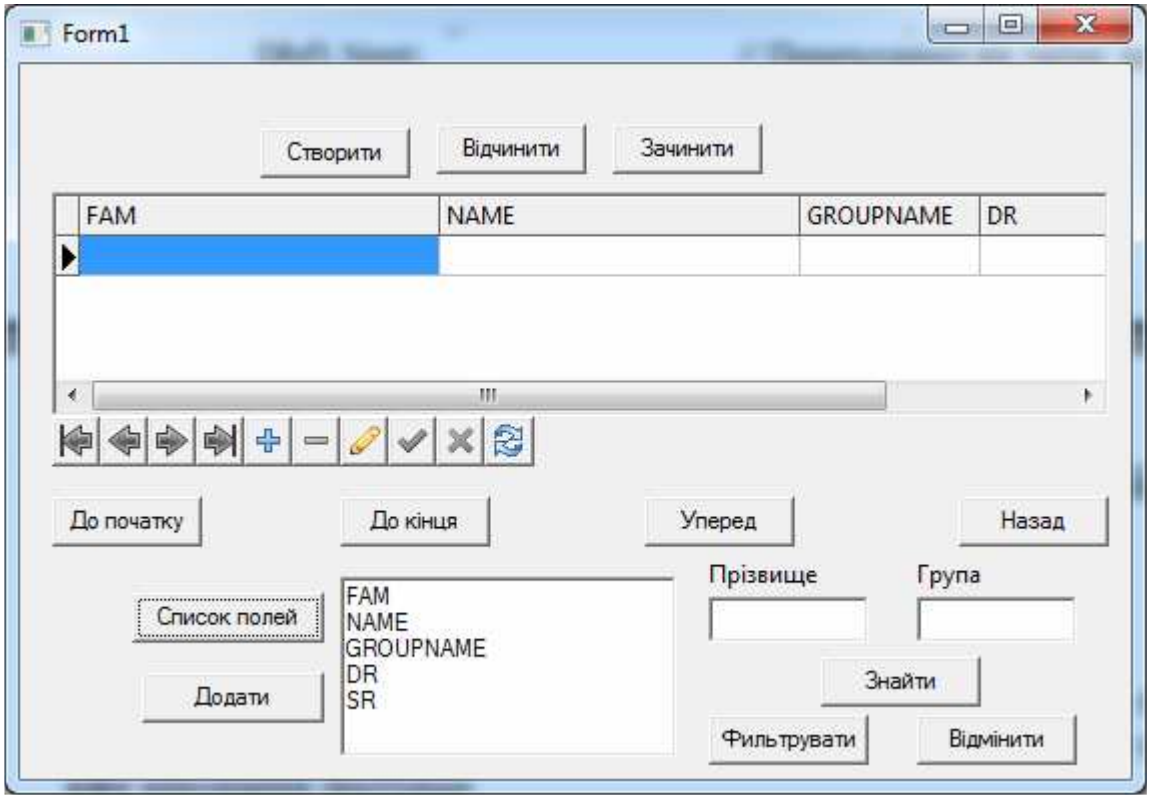


Рисунок 3.28 – Виведення ймен полів

Lazarus може робити перетворення, наприклад, перетворювати поле Boolean на Integer або Float, або поле Integer на String. Але не буде перетворювати String на Integer, хоча і може перетворювати Float на Integer. Для доступу до полів Date або DateTime можна використовувати AsString і AsFloat.

Такі методи дозволяють змінити дані, пов'язані з Dbf:

```
procedure Edit;
procedure Append;
procedure Insert;
procedure Post;
procedure Cancel;
procedure Delete;
```

Усі ці методи – частина DataSet, вони успадковані і використовуються Dbf і SQLQuery. Кожного разу, коли потрібно змінити дані, потрібно спочатку перевести DataSet у режим редагування. Більшість візуальних компонент роблять це автоматично, але в програмі доведеться використовувати перелічені вище функції.

Ось типова послідовність, яку можна використовувати при зміні значення поля поточного запису:

```
Dbf1.Edit;
Dbf1.FieldByName('Fam').AsString := 'Шевченко';
Dbf1.Post;
```

Перший рядок переводить БД у режим редагування. Наступний рядок привласнює значення 'Шевченко' полю 'Fam'. Нарешті, дані записуються на диск, коли викликається Post.

Якщо після здійснення редагування з яких-небудь причин воно виявляється непотрібним, до початкового стану запис повертає метод Cancel (зрозуміло, якщо не був викликаний метод Post).

Для додавання нового запису до набору даних використовуються методи Append і Insert: перший вставляє новий запис після останньої, а другий – у поточне місце. І той, і інший переводять вставлений порожній запис у режим редагування:

```
procedure TForm1.ButtonAppendClick(Sender: TObject);
begin
    with Dbf1 do begin
    Append;
    FieldByName('Fam').AsString := 'Шевченко';
    FieldByName('Name').AsString := 'Тарас';
    FieldByName('GroupName').AsString := 'СМ-15-2';
    FieldByName('DR').AsDateTime := StrToDate('12.01.95');
    FieldByName('SR').AsFloat := 3.5;
    Post
    end
end;
```

Якщо потрібно заборонити додавання або редагування записів, властивості ReadOnly присвоюється значення True.

При створенні таблиці, як правило, створюються так звані індекси, що визначають поля для сортування даних. Наприклад, якщо було передбачене сортування за прізвищем або за середнім рейтингом, то для його реалізації після відкриття (Open) таблиці додається рядок

```
Dbf1.IndexName:=’IndFam’;
```

або

```
Dbf11.IndexName:=’IndSR’;
```

Дані в таблиці автоматично будуть відсортовані за вибраним полем.

На відміну від середовища Delphi, де для пошуку необхідної інформації в базі даних існувало два способи (SetKey / GotoKey і Locate), у Lazarus не реалізовано пошук в індексованій базі даних. Недолік другого способу – повільна робота, однак тут ми можемо знайти запис, наприклад, за комбінацією «Прізвище – Дата народження».

Форма визову методу Locate:

```
Locate('ім'я поля', 'рядок для пошуку', [опції]);
```

Ось найпростіший приклад його використання:

```
if not Dbf1.Locate('GroupName', 'СМ-15-3', [loCaseInsensitive]) then
ShowMessage('Групи СМ-15-3 не виявлено!');
```

Опція loCaseInsensitive передбачає ігнорування різниці між великими та малими літерами.

Пошук за комбінацією значень має складніший вигляд:

Locate('им’я поля 1; им’я поля 2, …', VarArrayOf (['рядок для пошуку 1', "рядок для пошуку 2', …]), [опції])

Наприклад:

Dbf1.Locate('Fam;GroupName',VarArrayOf(['Шевченко','СМ-15-2']),[]);

Тут ми намагаємося знайти в групі СМ-15-2 студента (здобувача вищої освіти) на прізвище Шевченко.

У наведеному нижче прикладі показано результат пошуку запису за комбінацією значень полів «Прізвище» та «Група»:

```
procedure TForm1.ButtonSearchClick(Sender: TObject);
begin
    if not Dbf1.Locate('Fam;GroupName',
    VarArrayOf([LabeledEdit1.Text,LabeledEdit2.Text]),
    [loCaseInsensitive]) then
    ShowMessage('У групі '+LabeledEdit2.Text+
    ' здобувача на прізвище '+LabeledEdit1.Text+' не знайдено!');
end;
```

Значення вводяться користувачем за допомогою двох компонент LabeledEdit, пошук здійснюється без урахування регістра літер. Якщо задана комбінація ніде не зустрічається, видається повідомлення про помилку (рис. 3.29), у разі успіху поточний покажчик переміщається на рядок зі знайденим записом (рис. 3.30).

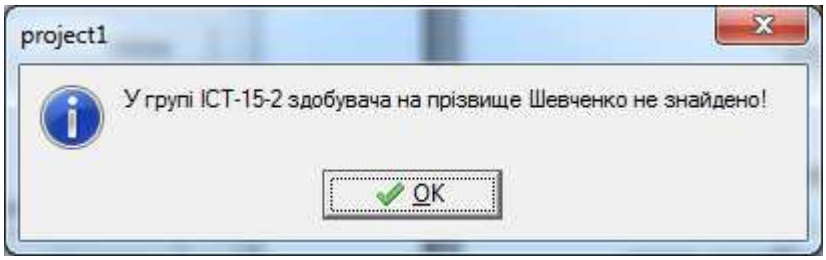


Рисунок 3.29 – Повідомлення про помилку

Властивість Filter дозволяє задати обмеження для записів. Слід пам'ятати, що фізично записи залишаються в базі даних. У наведеному нижче прикладі натискання кнопки призведе до того, що доступною залишиться інформація тільки про тих здобувачів, чиї рейтинги знаходяться в діапазоні від 4 до 5; натискання другої кнопки означатиме скасування фільтра, тобто показ на екрані всіх записів цієї бази даних (рис. 3.31).

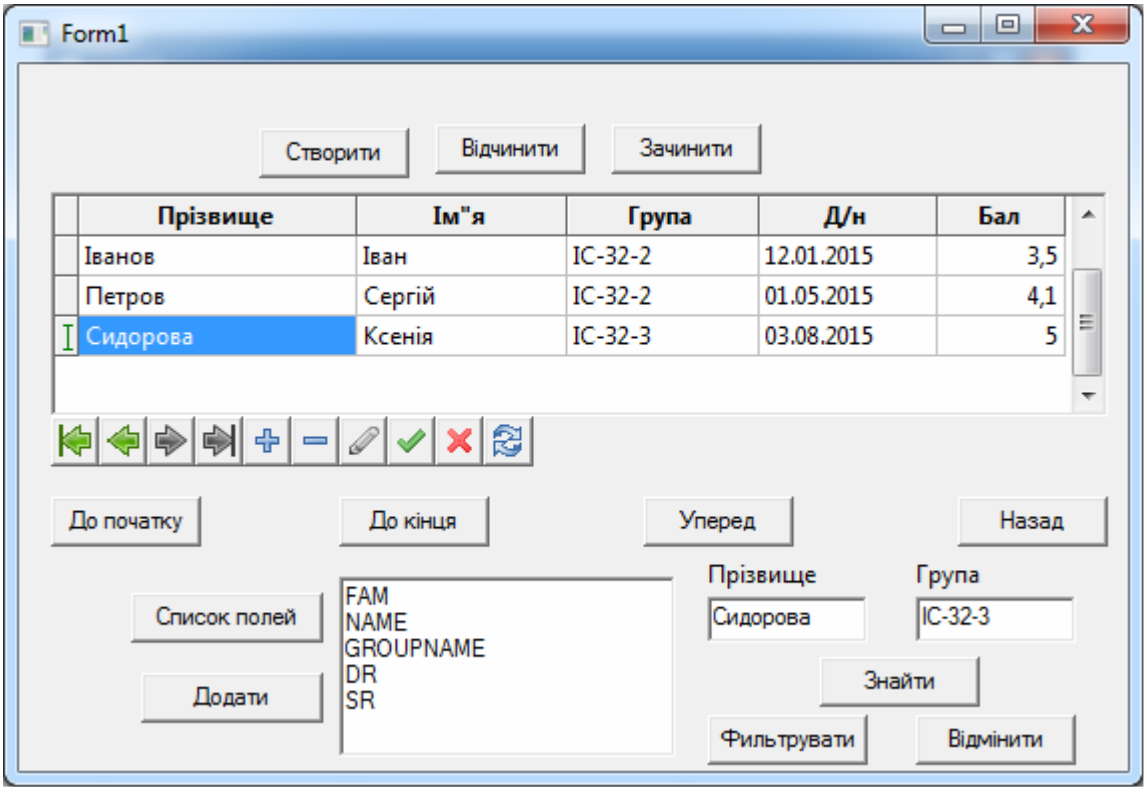


Рисунок 3.30 – Пошук запису за комбінацією значень полів

```
procedure TForm1.ButtonFilterOnClick(Sender: TObject);
begin
    with Dbf1 do begin
    Filter:='(SR >= 4) and (SR <= 5)';
    Filtered:=True
    end
end;

procedure TForm1.ButtonFilterOffClick(Sender: TObject);
begin
    with Dbf1 do begin
    Filtered:=False;
    Filter:=''
    end
end;
```

Будь-яка таблиця, з якою працює наша програма, завжди схильна до зміни. Навіть, якщо передбачається, що програма не буде працювати в мережі, завжди існує можливість того, що база даних може мати кілька шляхів зміни даних до таблиці. Ми повинні бути впевнені, що працюємо з «останнім» варіантом набору даних. Для цього існує метод Refresh.

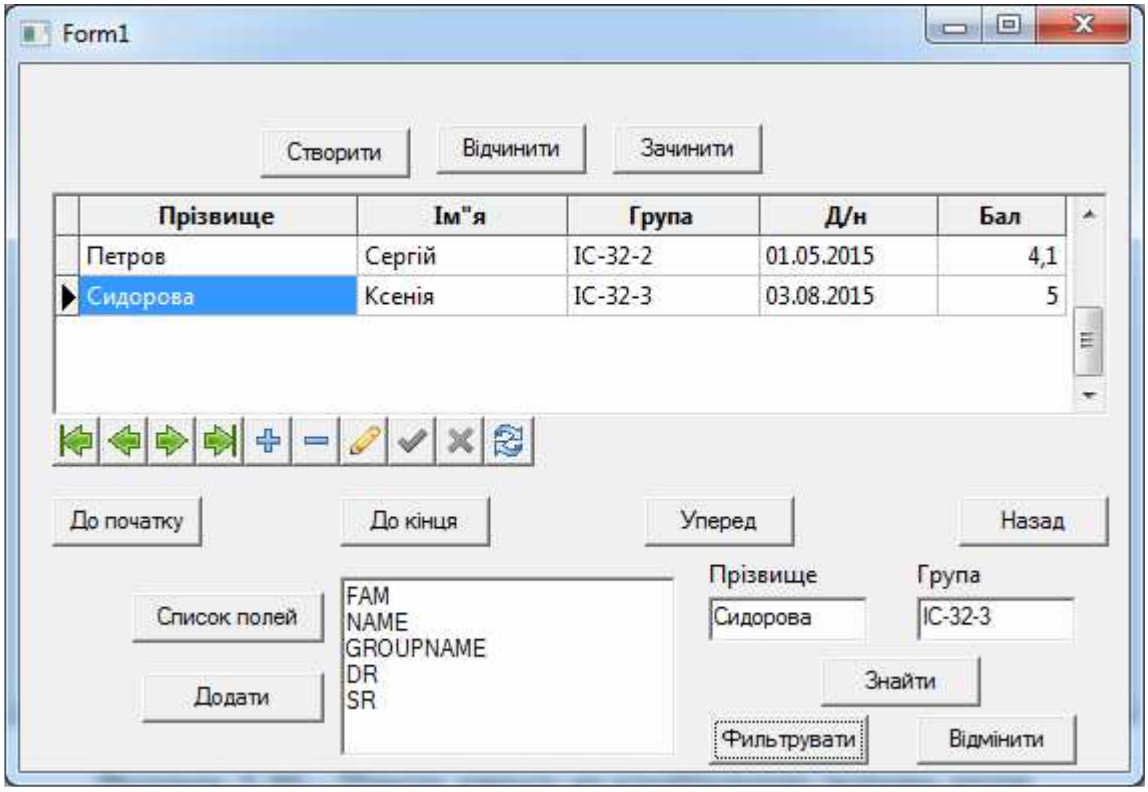


Рисунок 3.31 – Фільтрація даних

Проте слід мати на увазі, що оновлення Dbf може іноді призвести до несподіваних результатів. Наприклад, якщо користувач розглядає запис, який вже був видалений, то він зникне з екрана в той момент, коли буде викликаний Refresh. Аналогічно, якщо якийсь інший користувач редагував дані, то виклик Refresh призведе до динамічної зміни даних. Звичайно, малоймовірно, що один користувач буде змінювати або видаляти запис у той час, як інший переглядає його, але це можливо.

Часто буває корисно зазначити поточне місце розташування в таблиці так, щоб можна було швидко повернутися до цього місця надалі. Lazarus забезпечує цю функціональну можливість за допомогою трьох методів, які використовують поняття закладки:

```
functon GetBookmark: TBookmark; // встановлює закладку в таблиці
procedure GotoBookmark(Bookmark: TBookmark);//переходить на закладку
procedure FreeBookmark(Bookmark: TBookmark); // звільняє пам'ять
```

Виклик GetBookmark повертає закладку – змінну типу TBookmark. Така закладка містить достатню кількість інформації, щоб Lazarus міг знайти місце розташування, до якого вона належить. Тому можна просто передавати цю закладку функції GotoBookmark, і поточний покажчик буде переміщений до запису, пов'язаного з цією закладкою. Необхідно пам'ятати, що виклик GetBookmark виділяє пам'ять для закладки, тому потрібно викликати FreeBookmark до закінчення роботи програми і перед кожною спробою повторного використання оголошеної закладки.

У всіх раніше розглянутих випадках передбачалося, що база даних уже створена або іншим додатком, або нами ж на етапі дизайну програми. Однак досить часто доводиться перекладати цю роботу «на плечі» програми.

Для створення таблиць компонента Dbf має метод CreateTable, який може створювати локальні таблиці формату dBase (.dbf). Компонента Dbf, як правило, розміщується на формі на етапі проєктування програми, а власне створення таблиць відбувається десь у програмі.

Перед викликом методу CreateTable необхідно встановити значення таких властивостей:

FilePath – ім'я папки з файлами таблиці

TableName – ім'я таблиці

FieldDefs – масив описів полів

IndexDefs – масив описів індексів

Властивість FieldDefs для існуючої таблиці містить інформацію про всі поля таблиці. Ця інформація доступна тільки в режимі виконання і зберігається у вигляді масиву екземплярів класу TFieldDef, що зберігають дані про фізичні поля таблиці. Кількість полів визначається властивістю Count, а доступ до елементів масиву здійснюється через властивість Items:

property Items[Index: Integer]: TFieldDef;

При створенні таблиці, перед викликом методу CreateTable, потрібно сформувати ці елементи. Для цього у класі TFieldDefs є метод Add:

procedure Add(const Name: string; DataType: TFieldType; Size: Word; Required: Boolean);

Параметр Name, що має тип string, визначає ім'я поля (обов'язково складається з англійських літер, цифр і не містить ніяких спеціальних символів). Параметр DataType (тип TFieldType) позначає тип поля. Він може мати одне з таких значень, зміст яких ясний з їх найменування:

TFieldType = (ftUnknown, ftString, ftSmallint, ftInteger, ftWord, ftBoolean, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftBytes, ftVarBytes, ftBlob, ftMemo, ftGraphic);

Параметр Size (тип word) містить розмір поля. Цей параметр має сенс лише для полів типу ftString, ftBytes, ftVarBytes, ftBlob, ftMemo, ftGraphic, розмір яких може сильно варіюватися. Поля інших типів завжди мають строго фіксований розмір, так що даний параметр для них не береться до уваги. Четвертий параметр – Required – визначає, чи може поле мати пусте значення при записі до бази даних. Якщо значення цього параметра є true, то поле є «необхідним», тобто не може мати порожнього значення. В іншому випадку поле не є «необхідним» і, отже, допускає запис значення NULL.

Якщо необхідно проіндексувати таблицю за одним або декількома полями з метою подальшого автоматичного сортування за заданим полем, використовується метод AddIndex:

```
procedure AddIndex(const Name, Fields:string; Options:TIndexOptions);
```

Параметр Name визначає ім'я індексу, параметр Fields визначає імена поля або полів, які повинні бути індексовані. Складовий індекс, який використовує кілька полів, може бути заданий списком імен полів, відокремлених крапкою з комою «;», наприклад: 'Field1; Field2; Field4'. Останній параметр – Options – визначає тип індексу. Він може мати набір значень, описуваних типом TIndexOptions:

```
TIndexOptions = set of (ixPrimary, ixUnique, ixDescending, ixCaseInsensitive, ixExpression);
```

ixPrimary позначає первинний ключ, ixUnique – унікальний індекс, ixDescending – індекс, що передбачає сортування за зменшенням значень (для рядків – у порядку, зворотному алфавітному), ixCaseInsensitive – індекс, «нечутливий» до регістру літер (для російських слів непридатний), ixExpression – індекс за висловом. Для застосування останньої опції достатньо в параметрі Fields вказати бажаний вираз, наприклад: 'Field1 * Field2 + Field3'.

Головна відмінність від Delphi полягає в зміненій послідовності дій: у Lazarus необхідно спочатку описати поля, потім створити таблицю, відкрити її, додати всі індекси, а потім застосувати потрібний індекс. Ще одна важлива особливість: для індексів придатні тільки числові або символьні поля. Наявний індекс видаляється з таблиці методом DeleteIndex.

Нижче наводиться приклад створення таблиці, з якою ми вже працювали в багатьох попередніх прикладах:

```
procedure TForm1.ButtonCreateClick(Sender: TObject);
begin
    with Dbf1 do begin
    FilePath:='';                    // у поточній папці
    TableName:='mybase.dbf';
    with FieldDefs do begin
        Clear;
        Add('Fam',ftString,20,true);
        Add('Name',ftString,20,false);
        Add('GroupName',ftString,10,false);
        Add('DR',ftDate,0,false);
        Add('SR',ftFloat,0,false)
        end;{FieldDefs}
    CreateTable;
    Open;
    AddIndex('indFIO','Fam+Name',[ixExpression]);
```

```
    AddIndex('indSR','SR',[]);
    IndexName:='indFIO';
    end;
end;
```

При роботі з базами даних у середовищі Lazarus необхідно пам'ятати, що всі значення полів, введені на кириліці, Lazarus зберігає в кодуванні UTF8 (Unicode). Це означає:

– при роботі з таблицями, створеними в середовищі Lazarus, в іншому середовищі або системі управління базами даних, необхідно використовувати функції декодування;

– для зберігання кириличних рядків знадобиться рівно в 2 рази більше пам'яті (місця на диску), тому якщо ми описали поле «Прізвище» розміром 20 байт, то зможемо ввести прізвище українською мовою, що містить не більше ніж 10 літер.

Візуальна компонента DBGrid призначена для відображення на формі додатка вмісту зазначеної таблиці бази даних. Для найпростішої роботи з цією компонентою досить задати властивість DataSource, і при завантаженні будь-якої таблиці за допомогою відповідної компоненти (наприклад, Dbf) ця таблиця відразу з'явиться на екрані. Рекомендується також установити опцію dgEditing у True, а властивості ScrollBars надати значення ssAutoBoth.

Візуальна компонента DBNavigator об'єднує в собі низку кнопок для навігації по таблиці: вперед, назад тощо. Цю компоненту також необхідно пов'язати з відповідним джерелом даних DataSource.

Наведений далі приклад показує, як можна використовувати інші можливості компоненти DBGrid. Тут при відкритті бази даних заголовки таблиці змінюються (рис. 3.32):

```
procedure TForm1.ButtonOpenClick(Sender: TObject);
var i:integer;
begin
    with Dbf1 do begin
    FilePath:=''; TableName:='mybase.dbf';
    Open;
    DBGrid1.Columns[0].Title.Caption:='Прізвище';
    DBGrid1.Columns[1].Title.Caption:='Ім”я';
    DBGrid1.Columns[2].Title.Caption:='Група';
    DBGrid1.Columns[3].Title.Caption:='Д/н';
    DBGrid1.Columns[4].Title.Caption:='Бал';
    for i:=1 to FieldCount do begin
    DBGrid1.Columns[i-1].Title.Alignment:=taCenter;
    DBGrid1.Columns[i-1].Width:=DBGrid1.Columns[i-1].Width-1 end;
    DBGrid1.TitleFont.Style:=[fsBold];
    end;
end;
```

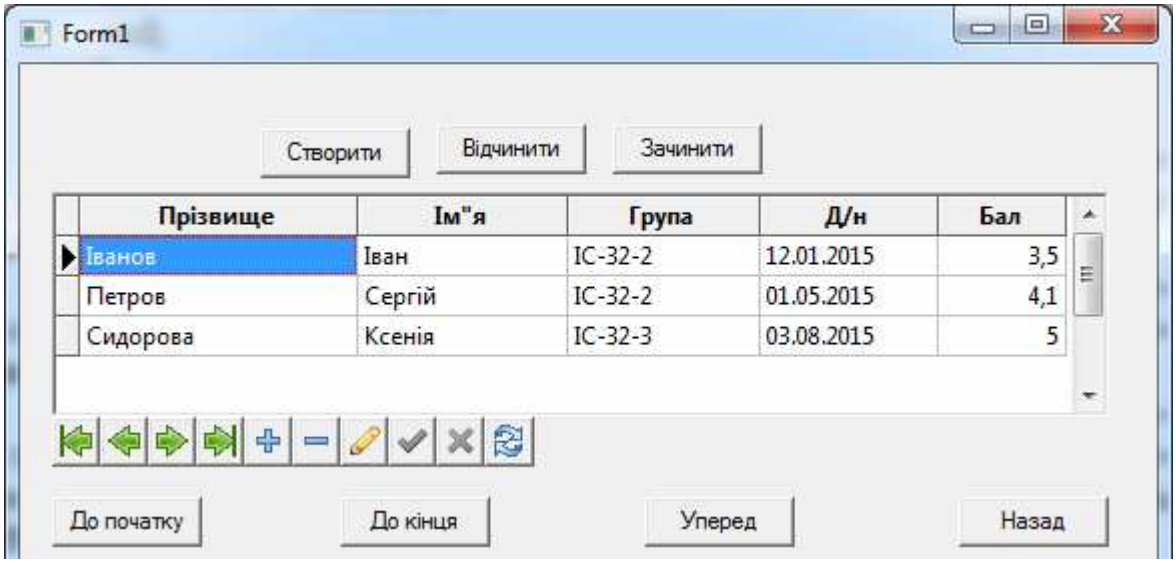


Рисунок 3.32 – Змінена таблиця

Кнопки у компоненті DBNavigator означають таке:

- First
- Prior
- Next
- Last
- Insert
- Delete
- Edit
- Post
- Cancel
- Refresh

Усі перелічені назви «спливають» у вигляді підказок при наведенні покажчика миші на відповідну кнопку; назви цих підказок можуть бути змінені в масиві Hints. Видимість кожної кнопки визначається компонентою VisibleButtons; якщо потрібно, наприклад, зробити недоступною кнопку «Видалити», записується так:

```
DBNavigator1.VisibleButtons:=DBNavigator1.VisibleButtons-[nbDelete]
```

Іноді масиви даних доцільніше зберігати й обробляти в текстовому файлі у вигляді полів, розділених заздалегідь позначеним символом (комою, пробілом, крапкою з комою тощо). У Lazarus для таких випадків є компонента SdfDataSet, значна частина властивостей і методів якої ідентичні властивостям і методам компоненти Dbf.

Відмінні властивості:

FileName – ім'я текстового файла;

Delimiter – символ роздільника полей;

FirstLineAsSchema – чи використовувати перший рядок як список імен полів.

При роботі з російським текстом необхідно пам'ятати, що він зберігається в кодуванні UTF-8 (Unicode), і при спробі звернутися до файла в кодуванні win-1251 або dos-866 такі поля просто не будуть відображатися на візуальних компонентах.

Для наведеного далі прикладу в системному текстовому редакторі «Блокнот» («AkelPad») був підготовлений текстовий файл, показаний на рисунку 3.33, і збережений у кодуванні UTF-8 («Зберегти у Unicode», Ctrl + Alt + U). Роздільники полів – пробіли, перший рядок являє собою перелік імен полів.

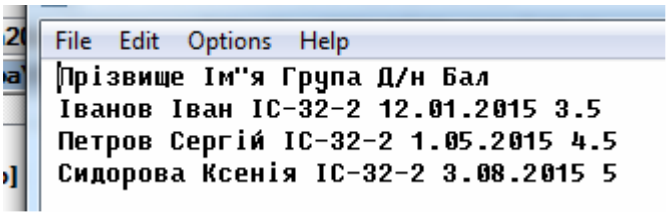


Рисунок 3.33 – Вхідні дані

Деяким недоліком візуальної компоненти DBGrid при роботі з текстовим файлом є відсутність автоматичного розрахунку ширини поля. Це необхідно зробити в програмі або просто задати ширину полів постійною. Дизайн форми і робота програми подані на рисунках 3.34, 3.35:

```
procedure TForm1.ButtonOpenClick(Sender: TObject);
var i:integer;
begin
    with SdfDataSet1 do begin
        if Active then Close;
        FileName:='mydata.txt';
        Delimiter:=#32;
        FirstLineAsSchema:=True;
        Open;
        for i:=1 to FieldCount do begin
        DBGrid1.Columns[i-1].Title.Alignment:=taCenter;
        DBGrid1.Columns[i-1].Width:=100;
        DBGrid1.Columns[i-1].Title.Caption:=Fields[i-1].FieldName
        end;
    end;
    DBGrid1.TitleFont.Style:=[fsBold];
end;
```

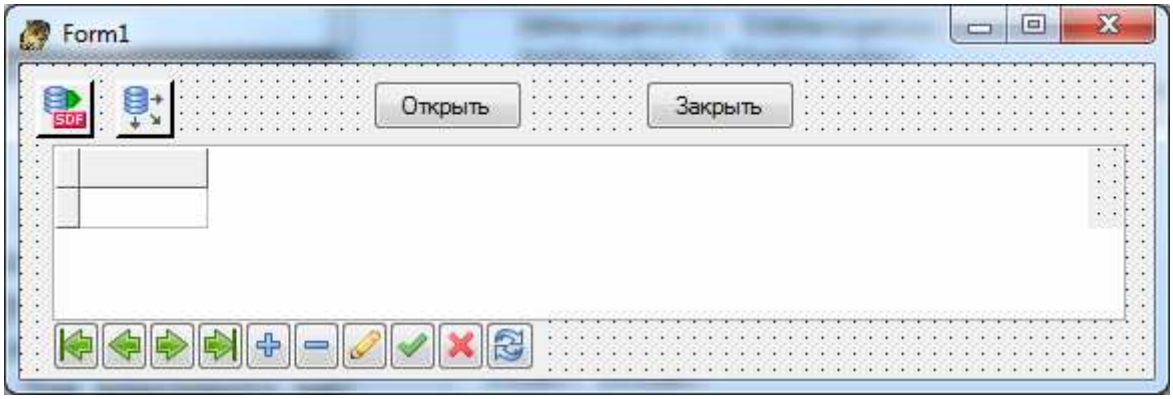


Рисунок 3.34 – Вигляд додатка на етапі дизайну

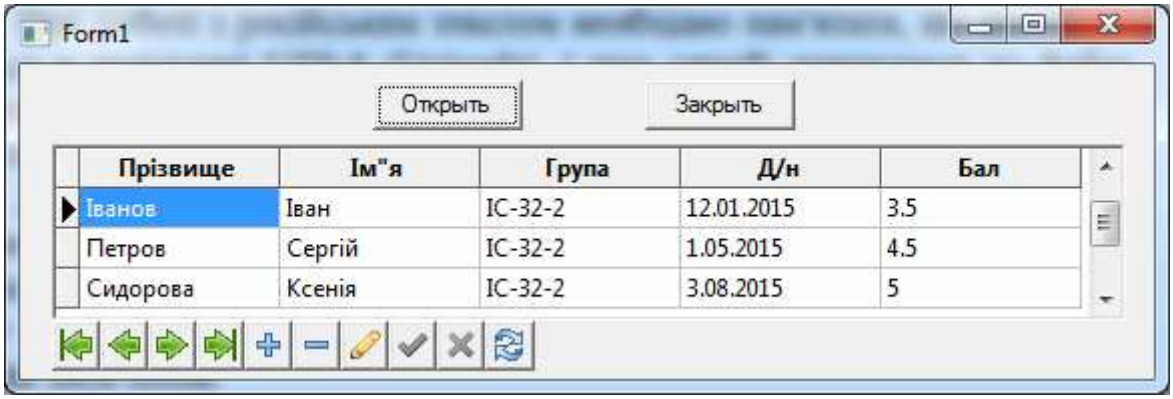


Рисунок 3.35 – Результат натискання кнопки «Відкрити»

Змінимо у здобувача Петрова середній бал з 4.5 на 4.1 (рис. 3.36), натиснемо спочатку кнопку Post (на панелі DBNavigator), а потім – «Закрити». Усі зміни збережуться в текстовому файлі (рис. 3.37).

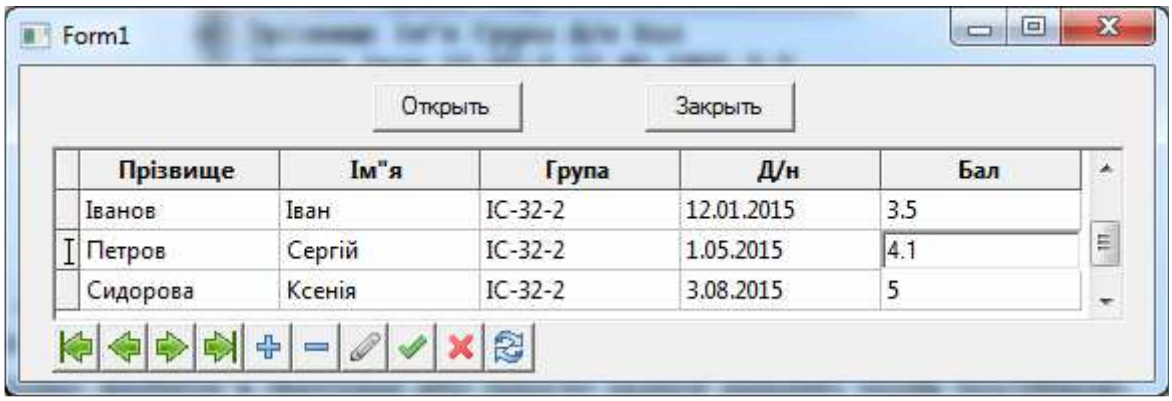


Рисунок 3.36 – Змінюємо значення поля

```
1  Прізвище Ім"я Група Д/н Бал
2  Іванов Іван ІС-32-2 12.01.2015 3.5
3  Петров Сергій ІС-32-2 1.05.2015 4.1
4  Сидорова Ксенія ІС-32-2 3.08.2015 5
```

Рисунок 3.37 – Змінений текстовий файл

Лабораторна робота 1. Компоненти середовища візуального програмування Lazarus

Мета роботи. Вивчити основні сторінки палітри компонент середовища візуального програмування Lazarus.

Завдання до роботи. Створити простий додаток у середовищі візуального програмування Lazarus. Форма повинна містити три взаємопов'язаних компоненти різних сторінок палітри компонент: дві вибираються з таблиці 5.1 згідно з номером варіанта, третю здобувач підбирає самостійно, виходячи з особистих переваг. Для кожної компоненти необхідно написати як мінімум по одному методу, яким ця компонента реагує на вибрану подію, при цьому хоча б один метод повинен змінювати стан іншої компоненти. У звіті про роботу перерахувати список властивостей кожної компоненти, які піддавалися зміні в процесі роботи програми.

Таблиця 3.3 – Список компонент

Вар.	**Компонента**	
	1	**2**
1	MainMenu	Memo
2	PopupMenu	ToggleBox
3	Button	CheckBox
4	Label	RadioButton
5	Edit	ListBox
6	Memo	Shape
7	ToggleBox	NoteBook
8	CheckBox	LabeledEdit
9	RadioButton	Button
10	ListBox	StatusBar
11	ComboBox	BitBtn
12	ScrollBar	Label
13	RadioGroup	SpeedButton
14	CheckGroup	Image
15	BitBtn	Edit
16	SpeedButton	ComboBox
17	Image	ScrollBar
18	Shape	ScrollBox
19	NoteBook	CheckListBox
20	LabeledEdit	RadioGroup
21	CheckListBox	CheckGroup
22	ScrollBox	ProgressBar
23	StringGrid	PopupMenu
24	TrackBar	SpinEdit
25	ProgressBar	Timer
26	StatusBar	MainMenu
27	SpinEdit	ColorBox
28	PageControl	StringGrid
29	ColorBox	PageControl
30	Timer	TrackBar

Приклад виконання роботи

Завдання: використати компоненти Shape, Timer і ProgressBar.

Можливе вирішення: додаток буде являти собою комп'ютерну гру, у процесі якої гравцеві потрібно клікнути мишею по випадково виникаючим на екрані кулькам (компонента Shape) з фіксованим контролем часу (компонента Timer) і фіксованою (максимально можливою) кількістю кульок. За переміщення зображень по екрану відповідає інша компонента Timer, хід часу і кількість (частку) «спійманих» кульок показують компоненти ProgressBar. Гра закінчується у разі закінчення відведеного на неї часу чи «упіймання» гравцем заданої кількості кульок.

Використовувані властивості компонент

Компонента	**Властивості, що визначаються відразу**	**Властивості, змінювані в процесі роботи**
Shape1	Shape (stCircle), Width (40), Height (40)	Top, Left
Timer1	Interval (1000)	
Timer2	Interval (1000)	
ProgressBar1	Width (по ширині форми)	Min, Max, Position
ProgressBar2	Width (по ширині форми)	Min, Max, Position

Текст модуля програми наведений далі, вигляд форми і працюючої програми – на рисунках 3.38…3.40:

```
unit Unit1;
{$mode objfpc}{$H+}
interface
uses
  Classes, SysUtils, FileUtil, LResources, Forms, Controls, Graphics, Dialogs, Menus, StdCtrls, Buttons, ExtCtrls, ComCtrls;
type
  { TForm1 }
  TForm1 = class(TForm)
    ProgressBar1: TProgressBar;
    ProgressBar2: TProgressBar;
    Shape1: TShape;
    Timer1: TTimer;
    Timer2: TTimer;
    procedure FormActivate(Sender: TObject);
    procedure Shape1MouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
    procedure Timer1Timer(Sender: TObject);
    procedure Timer2Timer(Sender: TObject);
  private
```

```
  public
  end;
var
  Form1: TForm1;
implementation
{ TForm1 }
var k:integer; // кількість спійманих кульок

procedure TForm1.FormActivate(Sender: TObject);
// процедура встановлення початкових значень
begin
    with ProgressBar1 do begin
    Min:=0;
    Max:=60; // 1 хвилина на гру
    Position:=0 end;
    with ProgressBar2 do begin
    Min:=0;
    Max:=5; // треба спіймати 5 кульок
    Position:=0 end;
    k:=0;
    randomize; // вмикаємо генератор випадкових чисел
    with Shape1 do begin
    Top:=random(Height-50); // розміщаємо першу кульку
    Left:=random(Width)
    end;
end;

procedure TForm1.Shape1MouseDown(Sender: TObject; Button:
TMouseButton;  Shift: TShiftState; X, Y: Integer);
// клік по кульці
begin
    k:=k+1;
    with ProgressBar2 do begin
    Position:=k;
    if Position=Max then begin // кінець гри
    ShowMessage('Вітаємо, ви виграли!');
    Close end
    end;
    Shape1.Top:=random(Height-50-Shape1.Height);
    Shape1.Left:=random(Width-Shape1.Width)
end;

procedure TForm1.Timer1Timer(Sender: TObject);
// контроль часу
begin
    with ProgressBar1 do begin
    Position:=Position+1; // секунда пройшла
```

```
    if Position=Max then begin // кінець гри
    ShowMessage('Час вичерпано!');  Close
    end
    end
end;

procedure TForm1.Timer2Timer(Sender: TObject);
// переміщення кульки
var nach:integer;
begin
    nach:=random(4); // випадкове число 0..3
    case nach of
    0: begin Shape1.Left:=Shape1.Left+10; // йдемо праворуч
      if Shape1.Left+Shape1.Width >= Width then Shape1.Left:=1
      end;
    1: begin Shape1.Top:=Shape1.Top+10; // йдемо вниз
      if Shape1.Top+Shape1.Height >= Height-50 // ProgressBar
      then Shape1.Top:=1
      end;
    2: begin Shape1.Left:=Shape1.Left-10; // йдемо ліворуч
      if Shape1.Left <= 10 then Shape1.Left:=Width-Shape1.Width-1
      end;
    3: begin Shape1.Top:=Shape1.Top-10; // йдемо вверх
      if Shape1.Top <= 10 then Shape1.Top:=Height-50-Shape1.Height
      end;
    end
end;
initialization
  {$I unit1.lrs}
end.
```

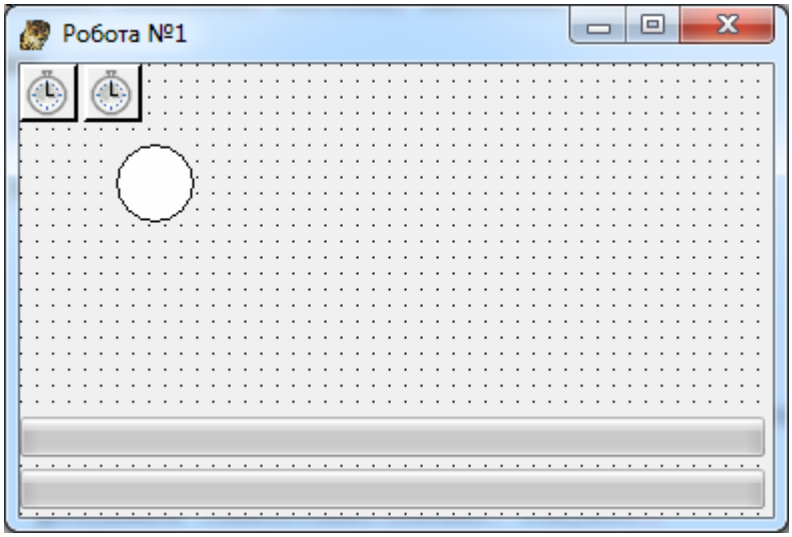


Рисунок 3.38 – Вигляд додатка на етапі дизайну

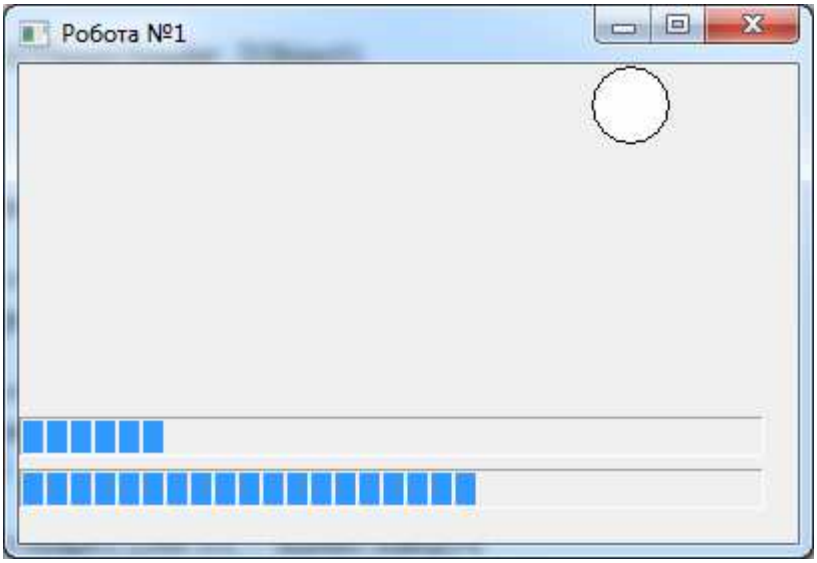
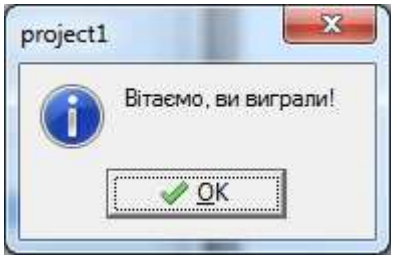
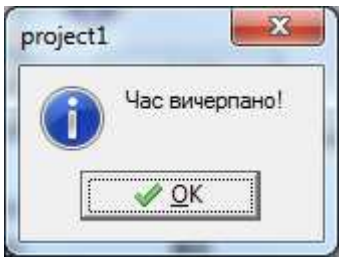


Рисунок 3.39 – Вигляд додатка під час роботи



а)



б)

а – «Успіх»; б – «За часом»
Рисунок 3.40 – Повідомлення про закінчення гри

Висновки. Під час лабораторної роботи були вивчені компоненти Shape, Timer і ProgressBar, їх основні властивості і призначення; був розроблений ігровий додаток, у якому використовувалися розглянуті компоненти.

Лабораторна робота 2. Розроблення простого додатка для Windows за допомогою засобу розроблення Lazarus

Мета роботи. Отримати навички використання різних компонент середовища візуального програмування Lazarus для вирішення поставленого завдання.

Завдання до роботи. Створити додаток згідно з індивідуальним завданням (табл. 3.4). Додаток повинен містити базовий набір керуючих елементів (головне меню, рядок стану тощо).

Таблиця 3.4

Вар.	**Зміст завдання**
1	*2*
1	Розробити додаток, що здійснює переведення температури в градусах за Цельсієм у температуру за Фаренгейтом або Кельвіном за такими залежностями: Фаренгейт = 32 + (Цельсій / 5) * 9 Кельвін = 273,15 + Цельсій
2	Розробити додаток, який здійснює введення 6 значень і знаходження або середнього арифметичного, або середнього геометричного значення при натисканні командної кнопки, а також видачу результату
3	Розробити додаток, який здійснює введення 5 значень і знаходження математичного очікування, або середньоквадратичного відхилення, або дисперсії при натисканні командної кнопки, а також видачу результату
4	Розробити додаток, який здійснює введення 7 значень і знаходження мінімального чи максимального з них при натисканні кнопки, а також видачу результату
5	Розробити додаток, який здійснює введення значень матриці 3х3 (за допомогою вікон введення тексту, розташованих у вигляді матриці) і обчислення її детермінантів
6	Розробити додаток, який здійснює вирішення квадратного рівняння за введеним користувачем коефіцієнтом. Вікна введення коефіцієнтів обов'язково повинні бути доповнені текстовими поясненнями
7	Розробити тестуючу програму, яка пропонує користувачеві два питання і по три відповіді на кожне з них (тільки одна правильна), а також виставляє оцінку за результатами тестування
8	Розробити додаток, у якому колір форми (властивість Color) змінювався б при кожному третьому натисканні відповідної кнопки, при цьому була б можливість цей режим обробки натиснення кнопки відключати

Продовження таблиці 3.4

1	*2*
9	Розробити додаток з вікном, у якому будуть розташовані всі компоненти сторінки «Стандартні». При наведенні на кожну з компонент покажчика миші повинна з'являтися підказка (властивість Hint при встановленій властивості ShowHint: = True) з описом можливостей компоненти. При цьому повинна бути передбачена можливість перемикання режиму показу підказок (короткі/повні)
10	Розробити додаток з вікном, у якому будуть розташовані всі компоненти сторінки «Додаткові». Робота програми повинна здійснюватися подібно попередньому варіанту
11	Розробити додаток з вікном, у якому будуть розташовані всі компоненти сторінки «Dialogs». Робота програми повинна здійснюватися подібно попередньому варіанту
12	Розробити додаток – електронний калькулятор для операцій +, -, *, / над числами
13	Розробити додаток, який здійснює піднесення до степеня числа. При цьому користувач повинен мати змогу ввести число, яке буде підноситися до степеня, і значення степеня, а програма повинна здійснювати перевірку символів, що вводять, і переведення негативного значення степеня на позитивне
14	Розробити додаток, у якому колір форми змінюється за допомогою трьох смуг прокрутки (компоненти ScrollBar) для кожного базового кольору – червоного, зеленого і синього. Отримані за допомогою смуг прокрутки числові значення повинні бути перетворені на значення кольору (за допомогою функції RGB), яке привласнюється властивості Color
15	Розробити додаток, який заповнює випадковими числами у випадковому порядку матрицю (4x4) полів введення тексту і очищає їх шляхом натиснення відповідної кнопки
16	Розробити «Ігровий автомат», який генерує випадковим чином виграшне число і порівнює його з уведеним користувачем числом; за результатами порівняння автомат повинен призначити розмір виграшу і повідомити про це користувачеві
17	Розробити додаток – шифрувальник текстів, який за заданим вами законом або переставляє місцями символи введеного користувачем тексту, або замінює одні символи в початковому тексті на інші, і відображає зашифрований варіант користувальницького тексту
18	Розробити додаток – емулятор пристрою зв'язку з можливістю набору телефонного номера з перевіркою формату уведених даних і зазначенням користувачеві помилок при введенні

Продовження таблиці 3.4

1	*2*
19	Розробити довідкову систему, яка видає у вікно виведення тексту інформацію щодо чотирьох операторів мови програмування Паскаль. Вибір інформації, що відображається, здійснити за допомогою залежних або незалежних перемикачів
20	Розробити довідкову систему, яка видає у вікно виведення тексту інформацію щодо чотирьох команд інтерфейсу засобу розробки Lazarus. Вибір інформації, що відображається, здійснити за допомогою залежних або незалежних перемикачів
21	Розробити додаток-гру під назвою «Сапер», яка зажадає від користувача включення масиву незалежних перемикачів таким чином, щоб не включити один або кілька перемикачів, що символізують міни. Інформація про близькість міни при включенні чергового перемикача повинна видаватися користувачеві
22	Розробити додаток – електронний калькулятор, який використовує тригонометричні функції
23	Розробити додаток для гри в хрестики-нулики двома гравцями. Для цього розставте CheckBox у вигляді матриці 3х3 для кожного гравця і надайте змоги вибору поточного гравця за допомогою залежного перемикача. Програма повинна видати результат гри
24	Розробити додаток – гру «Морський бій» (спрощена версія)
25	Розробити додаток для побудови гістограм (стовпчастих діаграм) за допомогою компоненти Chart. Дані повинні вводитися з текстового файла і відображатися за допомогою компоненти StringGrid
26	Розробити додаток для побудови гістограм (стовпчастих діаграм) за допомогою методів виведення прямокутників на канву. Дані повинні вводитися користувачем за допомогою компоненти StringGrid
27	Розробити додаток для побудови графіка заданої функції за допомогою компоненти Chart. Дані повинні вводитися з текстового файла і відображатися за допомогою компоненти StringGrid
28	Розробити додаток для побудови графіка заданої функції за допомогою методів малювання ліній на канві. Дані повинні вводитися користувачем за допомогою компоненти StringGrid
29	Розробити додаток, який забезпечений максимально можливою кількістю компонент для вибору властивостей форми (наприклад, компонента ColorBox дозволить змінювати колір форми, у полі введення тексту можна буде ввести заголовок форми тощо)
30	Розробити додаток, яке здійснює пошук екстремальних значень заданої функції на заданому інтервалі одним з відомих методів оптимізації. Передбачити висновок усієї проміжної інформації

Приклад виконання завдання

Завдання. Оснастити гру, яку було розроблено під час попередньої роботи, нормальним інтерфейсом користувача.

Опис вирішення. Перемістимо ігрове поле на окрему панель, додамо головне меню, статусний рядок і кнопки для управління, забезпечимо можливість зміни параметрів і їх збереження в текстовому файлі, а також збереження результатів гри на диску, встановимо контроль над можливістю зміни розмірів вікна.

Текст програми наведено нижче, екранні форми на етапі дизайну і роботи програми – на рисунках 3.41…3.49:

```
unit Unit1;
{$mode objfpc}{$H+}
interface
uses
  Classes, SysUtils, FileUtil, LResources, Forms, Controls, Graphics, Dialogs,
  Menus, StdCtrls, Buttons, ExtCtrls, ComCtrls, Grids;
type
  { TForm1 }
  TForm1 = class(TForm)
    MainMenu1: TMainMenu;
    MenuItem4: TMenuItem;
    MenuItem1: TMenuItem;
    MenuItem2: TMenuItem;
    MenuItem3: TMenuItem;
    MenuItem5: TMenuItem;
    MenuItem6: TMenuItem;
    MenuItem7: TMenuItem;
    Notebook1: TNotebook;
    Page1: TPage;
    Page2: TPage;
    Page3: TPage;
    Panel1: TPanel;
    ProgressBar1: TProgressBar;
    ProgressBar2: TProgressBar;
    Shape1: TShape;
    StatusBar1: TStatusBar;
    StringGrid1: TStringGrid;
    StringGrid2: TStringGrid;
    Timer1: TTimer;
    Timer2: TTimer;
    ToolBar1: TToolBar;
    ToolButton1: TToolButton;
    ToolButton2: TToolButton;
    ToolButton3: TToolButton;
```

```
    ToolButton4: TToolButton;
    procedure FormActivate(Sender: TObject);
    procedure FormClose(Sender: TObject; var CloseAction: TCloseAction);
    procedure FormCreate(Sender: TObject);
    procedure FormResize(Sender: TObject);
    procedure MenuItem1Click(Sender: TObject);
    procedure MenuItem2Click(Sender: TObject);
    procedure MenuItem3Click(Sender: TObject);
    procedure MenuItem5Click(Sender: TObject);
    procedure MenuItem6Click(Sender: TObject);
    procedure MenuItem7Click(Sender: TObject);
    procedure Shape1MouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure Timer1Timer(Sender: TObject);
    procedure Timer2Timer(Sender: TObject);
  private
    { private declarations }
  public
    { public declarations }
  end;
var
  Form1: TForm1;

implementation
{ TForm1 }
var k:integer; // кількість спійманих кульок
procedure TForm1.FormCreate(Sender: TObject);
// установка початкових значень
var S:TStringList;st:string;
    i:integer;
begin
     randomize; // вмикаємо генератор випадкових чисел
     with StringGrid1 do begin
     ColWidths[1]:=60;ColWidths[0]:=260;
     Cells[0,0]:='                          Параметр';
     Cells[1,0]:='Значення';
     Cells[0,1]:='Час на одну гру (секунд)';
     Cells[1,1]:='60';
     Cells[0,2]:='Скількі кульок треба спіймати';
     Cells[1,2]:='5';
     Cells[0,3]:='Затримка переміщення кульки (міллісекунд)';
     Cells[1,3]:='500';
     Cells[0,4]:='Им’я гравця';
     Cells[1,4]:='Admin';
     end;
```

```
    if FileExists('params.dat') then begin
      S:=TStringList.Create;
      S.LoadFromFile('params.dat');
      for i:=1 to 4 do StringGrid1.Cells[1,i]:=S[i-1];
      S.Free
    end;
    with StringGrid2 do begin
    Cells[0,0]:='   Дата і час';
    Cells[1,0]:='         Гравець';
    Cells[2,0]:='         Бали';
    end;
    if FileExists('gamers.dat') then begin
      S:=TStringList.Create;
      S.LoadFromFile('gamers.dat');
      StringGrid2.RowCount:=S.Count+1;
      for i:=1 to S.Count do begin
        st:=S[i-1];
        StringGrid2.Cells[0,i]:=copy(st,1,pos('//',st)-1);
        delete(st,1,pos('//',st)+1);
        StringGrid2.Cells[1,i]:=copy(st,1,pos('//',st)-1);
        delete(st,1,pos('//',st)+1);
        StringGrid2.Cells[2,i]:=st
        end;
      S.Free
    end;
    StatusBar1.Panels[0].Text:='Додаток працює'
end;
procedure TForm1.FormActivate(Sender: TObject);
// процедура встановлення початкових значень
begin
    with ProgressBar1 do begin
    Min:=0;
    Max:=StrToInt(StringGrid1.Cells[1,1]);
    Position:=0 end;
    with ProgressBar2 do begin
    Min:=0;
    Max:=StrToInt(StringGrid1.Cells[1,2]);
    Position:=0 end;
    Timer2.Interval:=StrToInt(StringGrid1.Cells[1,3]);
    k:=0;
    with Shape1 do begin
    Top:=random(Panel1.Height-Height); // розміщуємо першу кульку
    Left:=random(Panel1.Width-Width)
    end;
end;
```

```
procedure TForm1.FormClose(Sender: TObject; var CloseAction:
TCloseAction);
// зберігаємо параметри
var S:TStringList;
   i,j:integer;
begin
    S:=TStringList.Create;
    S.Clear;
    for i:=1 to 4 do S.Add(StringGrid1.Cells[1,i]);
    S.SaveToFile('params.dat');
    S.Clear;
    for i:=1 to StringGrid2.RowCount-1 do
    S.Add(StringGrid2.Cells[0,i]+'//'+
    StringGrid2.Cells[1,i]+'//'+StringGrid2.Cells[2,i]);
    if S.Count > 0 then S.SaveToFile('gamers.dat');
    S.Free
end;
procedure TForm1.FormResize(Sender: TObject);
// зміна розмірів форми
begin
    Notebook1.Left:=0;
    Notebook1.Top:=ToolBar1.Height+1;
    Notebook1.Width:=Width-1;
    Notebook1.Height:=Height-Notebook1.Top-StatusBar1.Height-21;
    ProgressBar2.Top:=Notebook1.Height-ProgressBar2.Height-26;
    ProgressBar1.Top:=ProgressBar2.Top-ProgressBar1.Height-2;
    ProgressBar2.Left:=1;ProgressBar2.Width:=Notebook1.Width-2;
    ProgressBar1.Left:=1;ProgressBar1.Width:=Notebook1.Width-2;
    Panel1.Left:=1;
    Panel1.Width:=Notebook1.Width-2;
    Panel1.Top:=1;
    Panel1.Height:=ProgressBar1.Top-Panel1.Top-1;
    StringGrid2.Left:=1;
    StringGrid2.Top:=1;
    StringGrid2.Height:=Notebook1.Height-22;
end;
procedure TForm1.MenuItem1Click(Sender: TObject);
// розпочати гру
begin
    FormActivate(Sender);
    Notebook1.PageIndex:=0;
    Timer1.Enabled:=True;
    Timer2.Enabled:=True;
    StatusBar1.Panels[0].Text:='Гра'
end;
```

```
procedure TForm1.MenuItem2Click(Sender: TObject);
// перервати гру
begin
    Timer1.Enabled:=False;
    Timer2.Enabled:=False;
    ProgressBar1.Position:=0;
    ProgressBar2.Position:=0;
    StatusBar1.Panels[0].Text:='Гру перервано';
    ShowMessage('Гру перервано з ініціативи гравця!')
end;

procedure TForm1.MenuItem3Click(Sender: TObject);
// закінчити роботу додатку
begin
    Close
end;

procedure TForm1.MenuItem5Click(Sender: TObject);
// про програму
begin
    ShowMessage('Гра "влуч по м'ячу"'+#13#13+
    'Автор: здобувач групи СМ-15-5 Шевченко Тарас Григорович')
end;

procedure TForm1.MenuItem6Click(Sender: TObject);
// змінити параметри
begin
    Notebook1.PageIndex:=1;
end;

procedure TForm1.MenuItem7Click(Sender: TObject);
// список гравців
begin
    Notebook1.PageIndex:=2;
end;

procedure TForm1.Shape1MouseDown(Sender: TObject; Button:
TMouseButton;  Shift: TShiftState; X, Y: Integer);
// натискання мишею по колу
begin
    if StatusBar1.Panels[0].Text <> 'Игра' then exit;
    k:=k+1;
    with ProgressBar2 do begin
    Position:=k;
```

```
    if Position=Max then begin // кінець гри
    Timer1.Enabled:=False;
    Timer2.Enabled:=False;
    with StringGrid2 do begin
        RowCount:=RowCount+1;
        Cells[0,RowCount-1]:=DateTimeToStr(Now);
        Cells[1,RowCount-1]:=StringGrid1.Cells[1,4];
        Cells[2,RowCount-1]:=IntToStr(k)+' circles / '+
        IntToStr(ProgressBar1.Position)+' seconds'
    end;
    ShowMessage('Поздоровляємо, ви виграли!');
    StatusBar1.Panels[0].Text:='Гру завершено перемогою гравця';
    Notebook1.PageIndex:=2;
    end
    end;
    Shape1.Top:=random(Panel1.Height-Shape1.Height);
    Shape1.Left:=random(Panel1.Width-Shape1.Width)
end;

procedure TForm1.Timer1Timer(Sender: TObject);
// контроль часуу
begin
    with ProgressBar1 do begin
    Position:=Position+1; // секунда пройшла
    if Position=Max then begin // кінець гри
    Timer1.Enabled:=False;
    Timer2.Enabled:=False;
    ShowMessage('Час вичерпано!');
    StatusBar1.Panels[0].Text:='Гру завершено після закінчення часу';
    end
    end
end;

procedure TForm1.Timer2Timer(Sender: TObject);
// переміщення кульки
var nach:integer;
begin
    nach:=random(4); // випадкове число 0..3
    case nach of
    0: begin
      Shape1.Left:=Shape1.Left+10; // йдемо праворуч
      if Shape1.Left+Shape1.Width >= Panel1.Width then Shape1.Left:=1
      end;
    1: begin
      Shape1.Top:=Shape1.Top+10; // йдемо вниз
```

```
      if Shape1.Top+Shape1.Height >= Panel1.Height-50 // ProgressBar
      then Shape1.Top:=1
      end;
    2: begin
      Shape1.Left:=Shape1.Left-10; // йдемо ліворуч
      if Shape1.Left <= 10 then Shape1.Left:=Panel1.Width-Shape1.Width-1
      end;
    3: begin
      Shape1.Top:=Shape1.Top-10; // йдемо верх
      if Shape1.Top <= 10 then Shape1.Top:=Panel1.Height-Shape1.Height
      end;
    end
end;

initialization
  {$I unit1.lrs}

end.
```

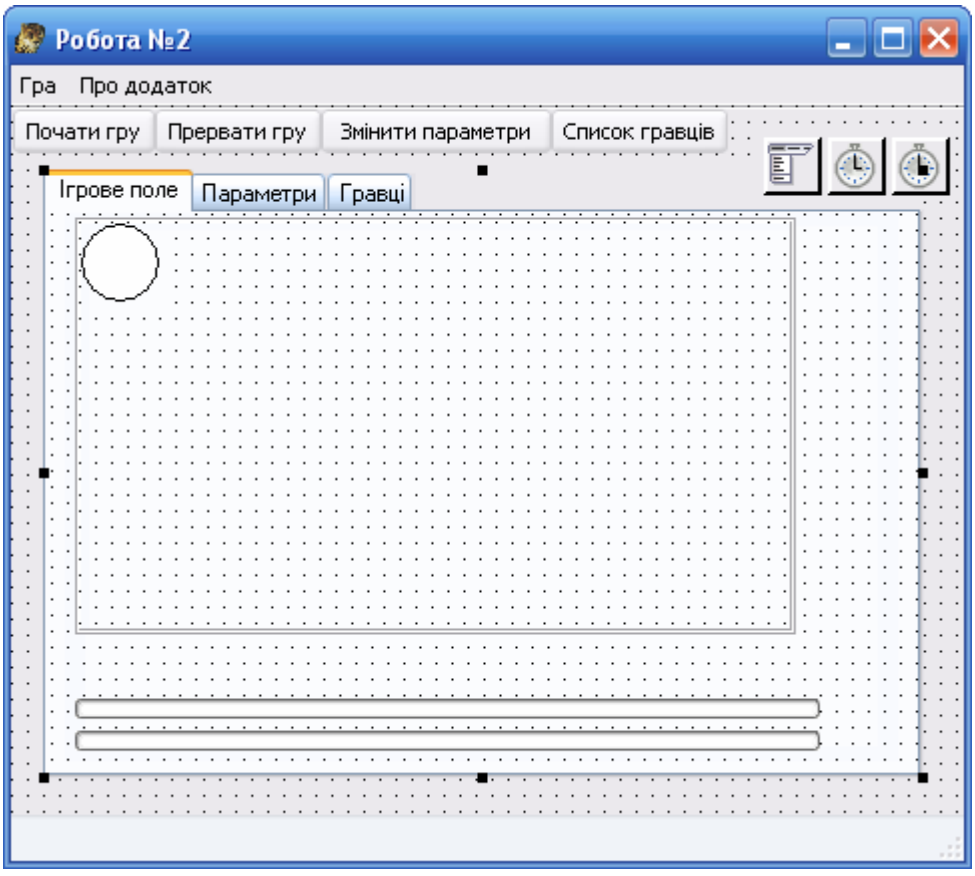


Рисунок 3.41 – Вигляд додатка на етапі дизайну («Ігрове поле»)

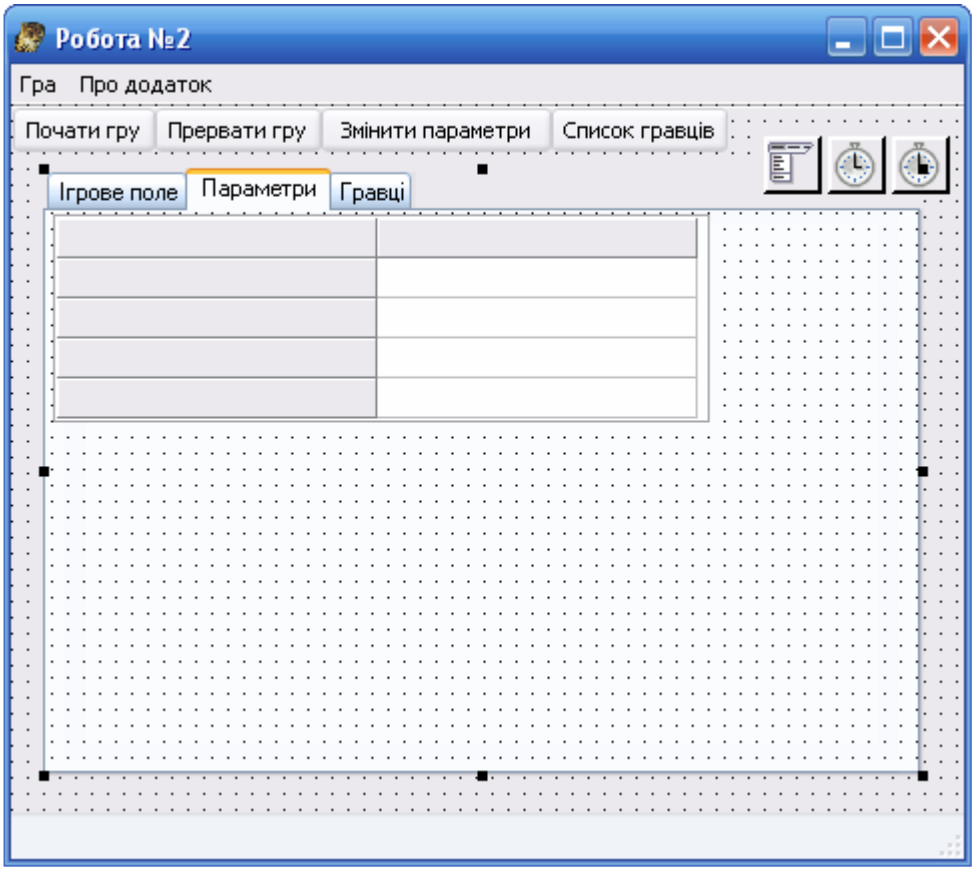


Рисунок 3.42 – Вигляд додатка на етапі дизайну («Параметри»)

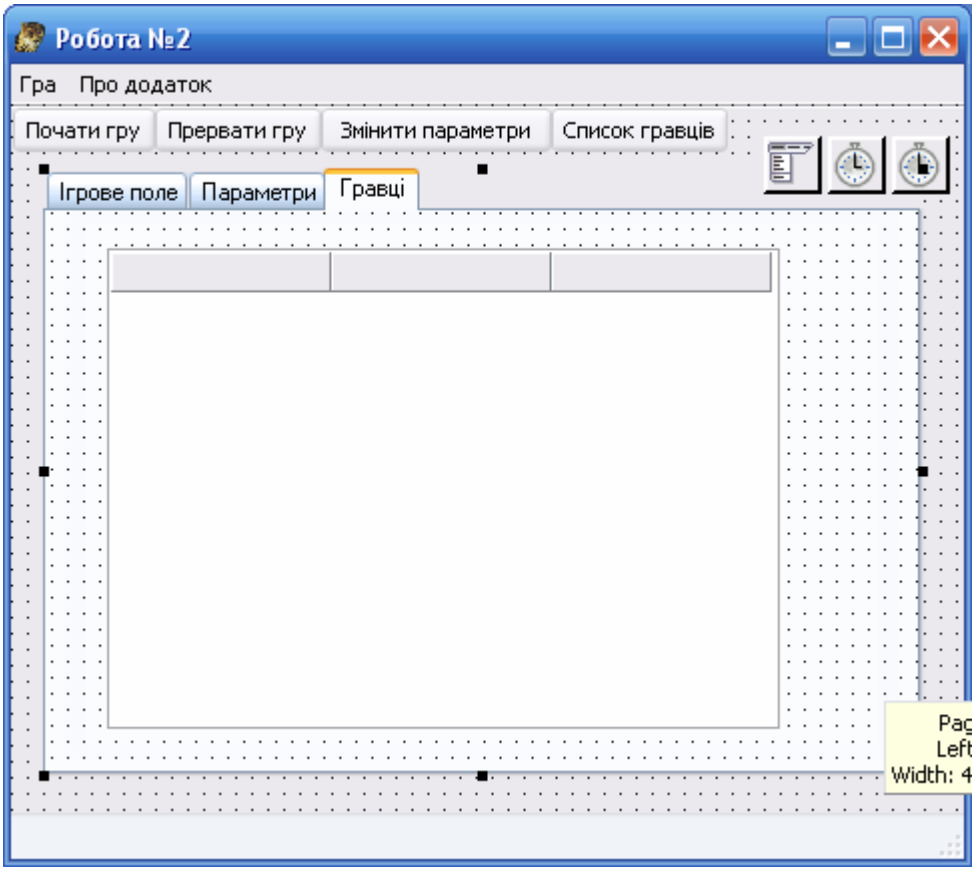


Рисунок 3.43 – Вигляд додатка на етапі дизайну («Гравці»)

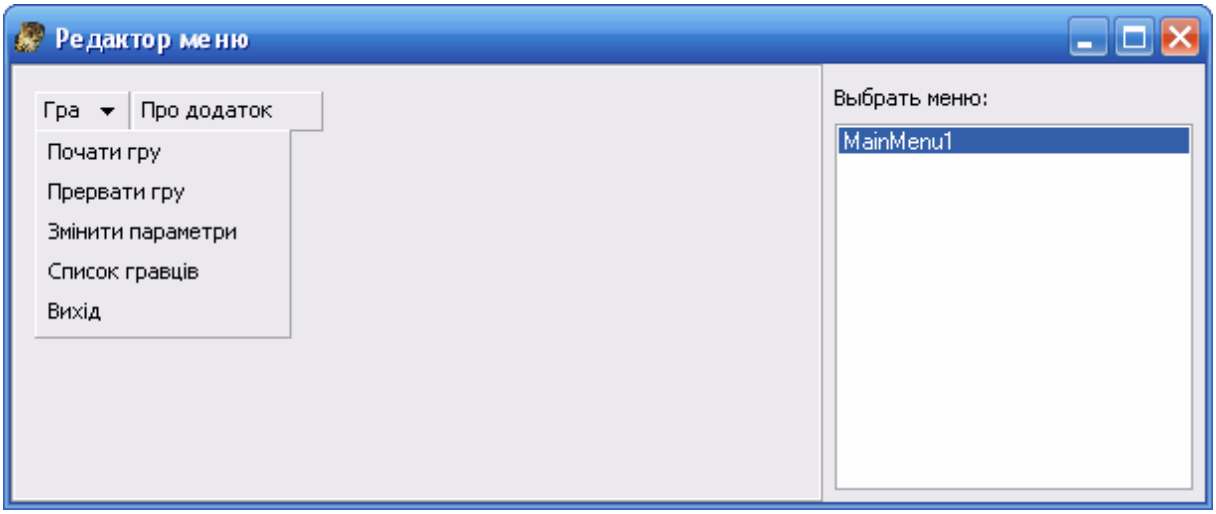


Рисунок 3.44 – Вигляд додатка на етапі дизайну (головне меню)

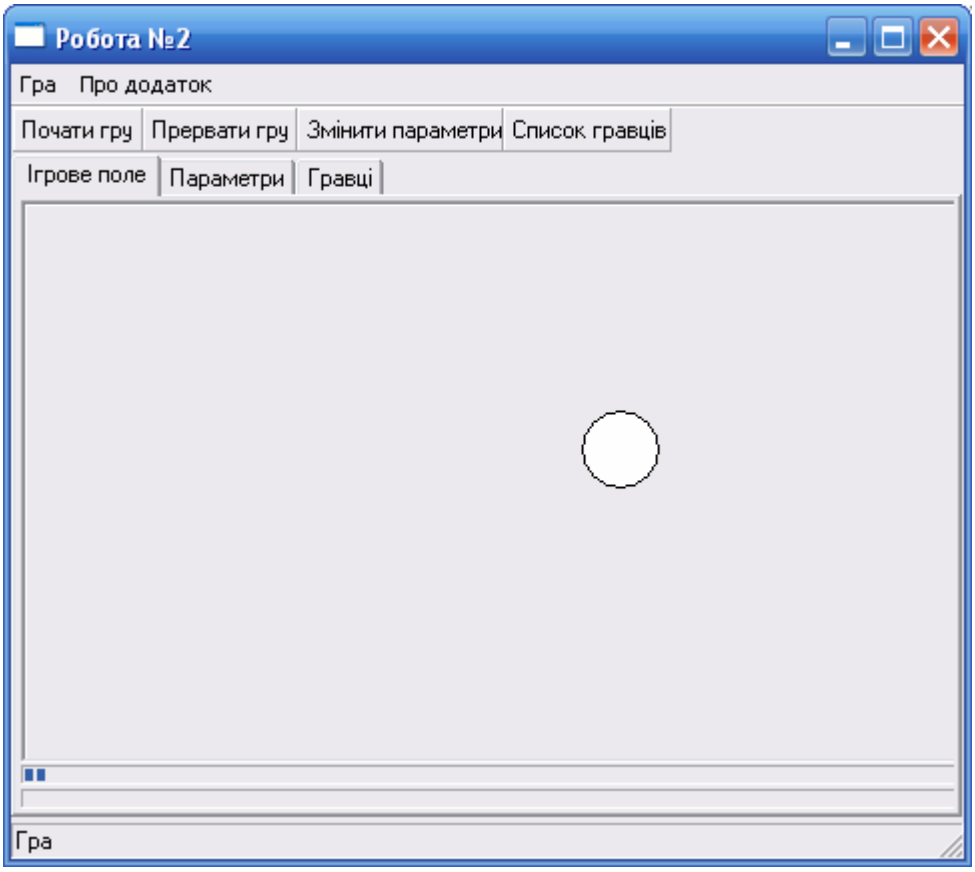


Рисунок 3.45 – Робота додатка: початок

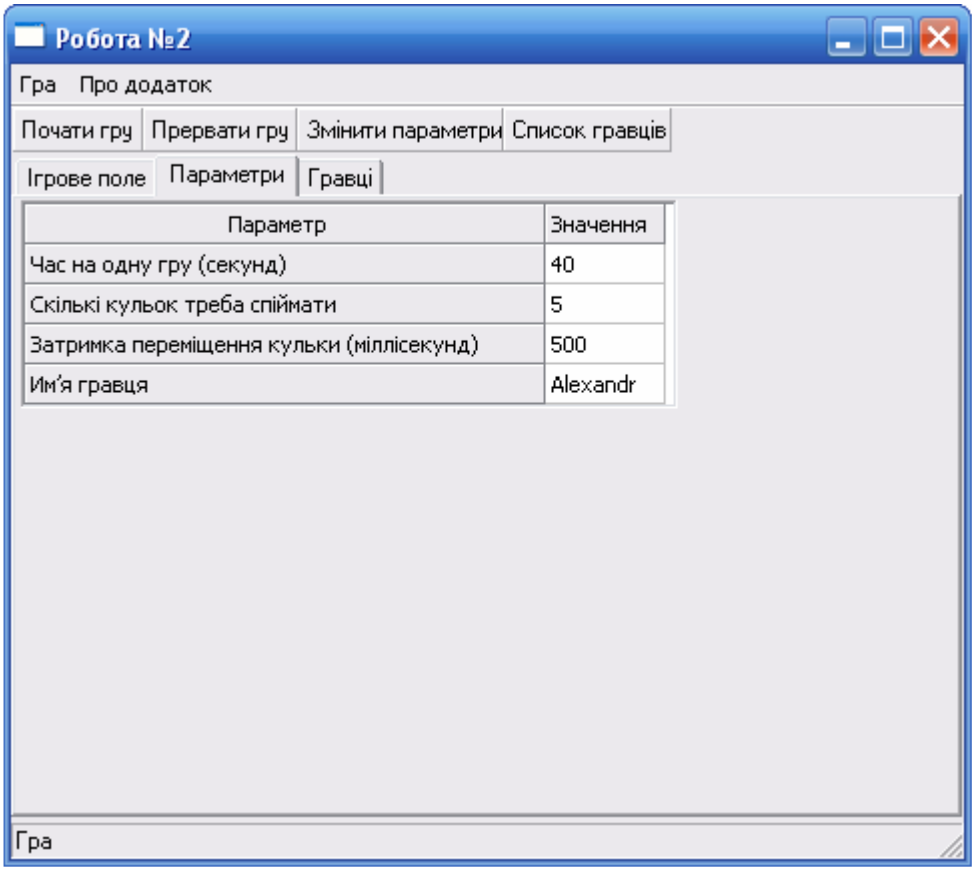


Рисунок 3.46 – Робота додатка: встановлення параметрів

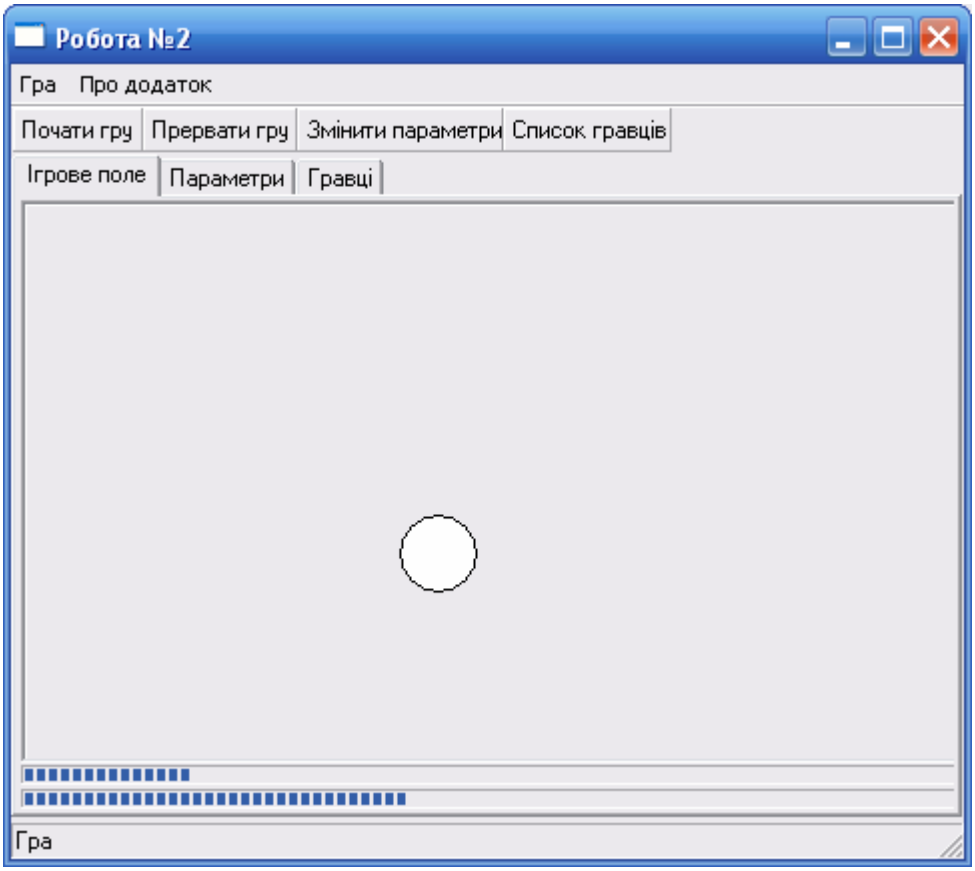


Рисунок 3.47 – Робота додатка: процес гри

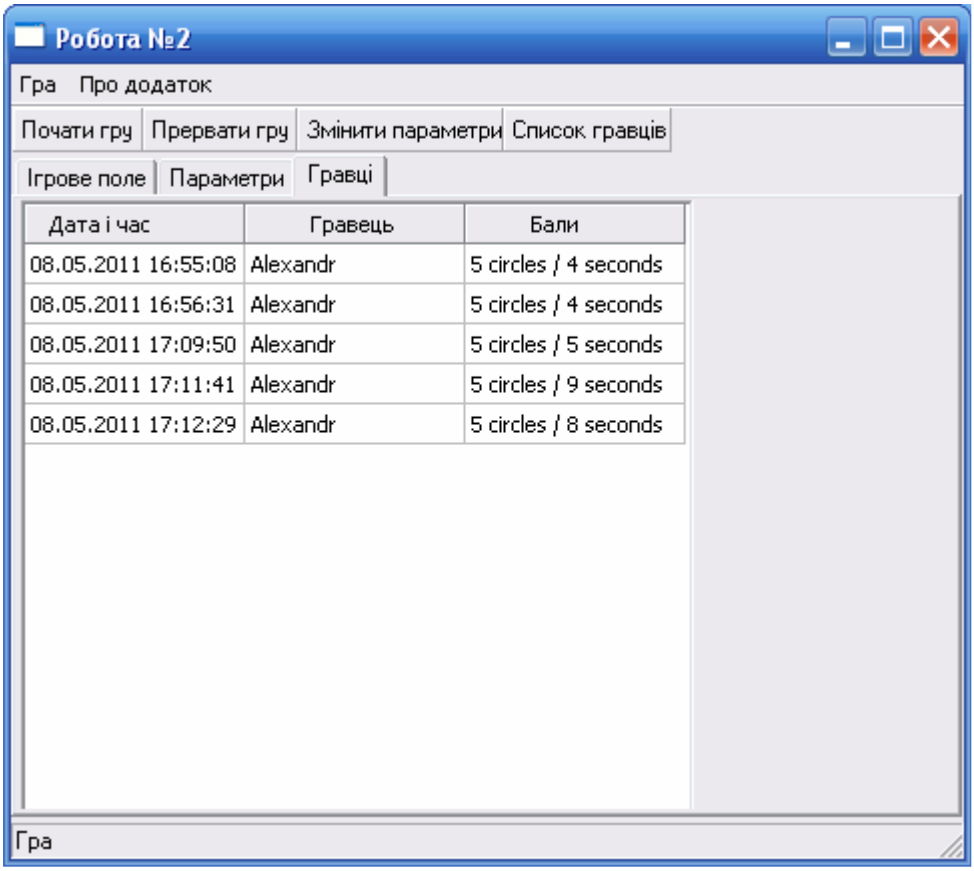


Рисунок 3.48 – Робота додатка: перелік гравців і досягнень

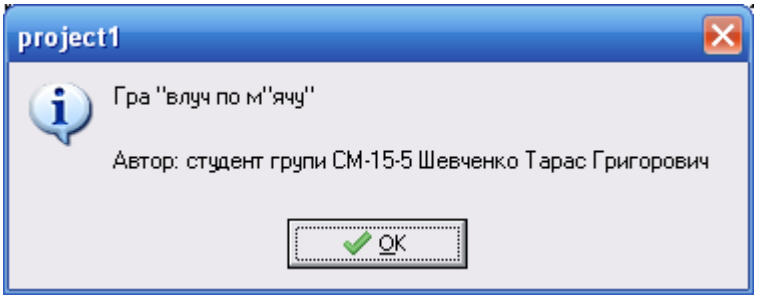


Рисунок 3.49 – Робота додатка: пункт меню «Про програму»

Висновки. Під час лабораторної роботи було створено повнофункціональний додаток для Windows – гра «Влуч по м'ячу»; вивчені компоненти (MainMenu, Notebook, Panel, ProgressBar, Shape, StatusBar, StringGrid, Timer, ToolBar), їх основні властивості і призначення; створені методи для оброблення подій FormCreate, FormActivate, FormResize, FormClose, MenuItemClick, ShapeMouseDown, TimerTimer. Додаток реалізовано у вигляді однієї форми, багатосторінковий інтерфейс забезпечується компонентою Notebook, що складається з трьох сторінок; вибір користувача здійснюється за допомогою головного меню і панельних кнопок.

Лабораторна робота 3. Побудова графіків функцій у середовищі Lazarus

Мета роботи. Отримати навички розроблення додатків, що дозволяють зобразити на екрані таблиці і графіки функцій однієї змінної.

Завдання до роботи. За вказаним літературним джерелом взяти дані про математичний опис функцій (кривих), виданих у якості індивідуального завдання (табл. 3.5). Розробити додаток, що дозволяє розраховувати таблицю значень своєї функції (компонента StringGrid) і будувати її графік (компонента Chart). Передбачити можливість коригування користувачем інтервалу і кроку зміни аргументу (вводяться у відповідних текстових полях), розрахунку значення функції при заданому значенні аргументу (задавати як явно, так і за допомогою «повзунка»), а також виведення результатів у текстовий файл.

Таблиця 3.5

Вар.	**Зміст завдання**
1	*2*
1	Лінійна функція; математичний опис, вимоги та обмеження, приклад графіків див. на с. 113 [6]
2	Квадратична функція; математичний опис, вимоги та обмеження, приклад графіків див. на с. 113 [6]
3	Функція третього степеня; математичний опис, вимоги та обмеження, приклад графіків див. на с. 113 [6]
4	Степенева функція; математичний опис, вимоги та обмеження, приклад графіків див. параграф 4 на с. 114, 116, 117 [6]
5	Дрібно-лінійна функція; математичний опис, вимоги та обмеження, приклад графіків див. на с. 114 [6]
6	Нелінійна дрібно-раціональна функція; математичний опис, вимоги та обмеження, приклад графіків див. на с. 115, п. 1 [6]
7	Нелінійна дрібно-раціональна функція; математичний опис, вимоги та обмеження, приклад графіків див. на с. 115, п. 2 [6]
8	Нелінійна дрібно-раціональна функція; математичний опис, вимоги та обмеження, приклад графіків див. на с. 115, п. 3 [6]
9	Напівкубічна парабола; математичний опис, вимоги та обмеження, приклад графіків див. на с. 123 [6]
10	Декартов лист; математичний опис, вимоги та обмеження, приклад графіків див. на с. 123 [6]
11	Цисоїда; математичний опис, вимоги та обмеження, приклад графіків див. на с. 123, 124 [6]
12	Строфоїда; математичний опис, вимоги та обмеження, приклад графіків див. на с. 124 [6]

Продовження таблиці 3.5

1	*2*
13	Равлик Паскаля; математичний опис, вимоги та обмеження, приклад графіків див. на с. 124 [6]
14	Кардіоїда; математичний опис, вимоги та обмеження, приклад графіків див. на с. 125 [6]
15	Епіциклоїда; математичний опис, вимоги та обмеження, приклад графіків див. на с. 126 [6]
16	Показова функція; математичний опис, вимоги та обмеження, приклад графіків див. на с. 119-120 [6]
17	Логарифмічна функція; математичний опис, вимоги та обмеження, приклад графіків див. на с. 119-120 [6]
18	Функція гіперболічного синуса; математичний опис, вимоги та обмеження, приклад графіків див. на с. 121–122 [6]
19	Функція гіперболічного косинуса; математичний опис, вимоги та обмеження, приклад графіків див. на с. 121–122 [6]
20	Функція гіперболічного тангенса; математичний опис, вимоги та обмеження, приклад графіків див. на с. 122 [6]
21	Функція гіперболічного котангенса; математичний опис, вимоги та обмеження, приклад графіків див. на с. 122 [6]
22	Звичайна циклоїда; математичний опис, вимоги та обмеження, приклад графіків див. на с. 125-126 [6]
23	Архімедова спіраль; математичний опис, вимоги та обмеження, приклад графіків див. на с. 128 [6]
24	Гіперболічна спіраль; математичний опис, вимоги та обмеження, приклад графіків див. на с. 128 [6]
25	Укорочена циклоїда; математичний опис, вимоги та обмеження, приклад графіків див. на с. 126 [6]
26	Подовжена циклоїда; математичний опис, вимоги та обмеження, приклад графіків див. на с. 126 [6]
27	Розгортка (евольвента) окружності; математичний опис, вимоги та обмеження, приклад графіків див. на с. 128, 129 [6]
28	Овали Кассіні; математичний опис, вимоги та обмеження, приклад графіків див. на с. 125 [6]
29	Лемніската; математичний опис, вимоги та обмеження, приклад графіків див. на с. 125 [6]
30	Трактриса; математичний опис, вимоги та обмеження, приклад графіків див. на с. 129 [6]

Приклад виконання завдання

Завдання: розрахувати значення і побудувати графік функції логарифмічної спіралі.

Опис вирішення: функція логарифмічної спіралі задається в полярній системі координат і має вигляд $\rho = A\, e^{k\varphi}$, φ = 0..360°. Якщо k = 0, то крива перетворюється на коло з радіусом A і центром на початку координат. Перетворення значень з полярної системи координат на декартову здійснюється за формулами:

$$\begin{cases} x = \rho \cos \varphi \\ y = \rho \sin \varphi \end{cases}$$

Інтервал і крок функції не задається, кут змінюється від 0 до 360°.

Текст програми наведено нижче, екранні форми на етапі дизайну і роботи програми – на рисунках 3.50…3.57:

```
unit Unit1;
{$mode objfpc}{$H+}
interface
uses
  Classes, SysUtils, FileUtil, LResources, Forms, Controls, Graphics,
Dialogs, Grids, ExtCtrls, TAGraph, StdCtrls, taseries, ComCtrls;
type
  { TForm1 }
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Chart1: TChart;
    Edit1: TLabeledEdit;
    Edit2: TLabeledEdit;
    StringGrid1: TStringGrid;
    StringGrid2: TStringGrid;
    TrackBar1: TTrackBar;
  procedure Button1Click(Sender: TObject);
  procedure Button2Click(Sender: TObject);
  procedure FormCreate(Sender: TObject);
  procedure Func(fi:real;var x,y:real);
  procedure TrackBar1Change(Sender: TObject);
  end;
var
  Form1: TForm1;
  S:TSerie; a,k:real;
implementation
```

```
procedure TForm1.Func(fi:real;var x,y:real);
var po:real;
begin
    po:=a*exp(k*fi);
    x:=po*cos(fi);
    y:=po*sin(fi);
end;
procedure TForm1.TrackBar1Change(Sender: TObject);
var i:integer;
begin
    i:=TrackBar1.Position;
    StringGrid2.Cells[0,0]:=StringGrid1.Cells[0,i+1];
    StringGrid2.Cells[1,0]:=StringGrid1.Cells[1,i+1];
    StringGrid2.Cells[2,0]:=StringGrid1.Cells[2,i+1];
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
    with StringGrid1 do begin
    RowCount:=362;ColCount:=3;
    Cells[0,0]:='fi';Cells[1,0]:='X';Cells[2,0]:='Y' end;
    S:=TSerie.Create(Chart1);
    S.ShowLines:=True;
    S.ShowPoints:=False;
    S.SeriesColor:=clBlack;
    Chart1.Title.Text.Text:= 'Логарифмічна спіраль';
    Chart1.Title.Visible:=true;
    Chart1.Series.Clear;
    Chart1.AddSerie(S);
    with TrackBar1 do begin
        Width:=Chart1.Width;
        Min:=0;Max:=360;
        Position:=0
    end
end;
procedure TForm1.Button1Click(Sender: TObject);
// розрахувати значення
var kod,fi:integer;x,y:real;
begin
    val(Edit1.Text,a,kod);
    if (a = 0) or (kod > 0) then begin a:=1;Edit1.Text:='1' end;
    val(Edit2.Text,k,kod);
    if kod > 0 then begin k:=1;Edit2.Text:='1' end;
    x:=0;y:=0;
    with StringGrid1 do
    for fi:=0 to Rowcount-2 do begin
```

```
        func(fi/180*pi,x,y);
        Cells[0,fi+1]:=IntToStr(fi);
        Cells[1,fi+1]:=FloatToStr(x);
        Cells[2,fi+1]:=FloatToStr(y)
        end;
    StringGrid1.SetFocus
end;
procedure TForm1.Button2Click(Sender: TObject);
// побудувати график
var i:integer;
begin
    S.Clear;
    with StringGrid1 do for i:=1 to Rowcount-1 do
    S.AddXY(StrToFloat(Cells[1,i]),
    StrToFloat(Cells[2,i]),IntToStr(i),clBlack);
    TrackBar1.Width:=Chart1.Width;
    StringGrid1.SetFocus
end;
initialization
  {$I unit1.lrs}
end.
```

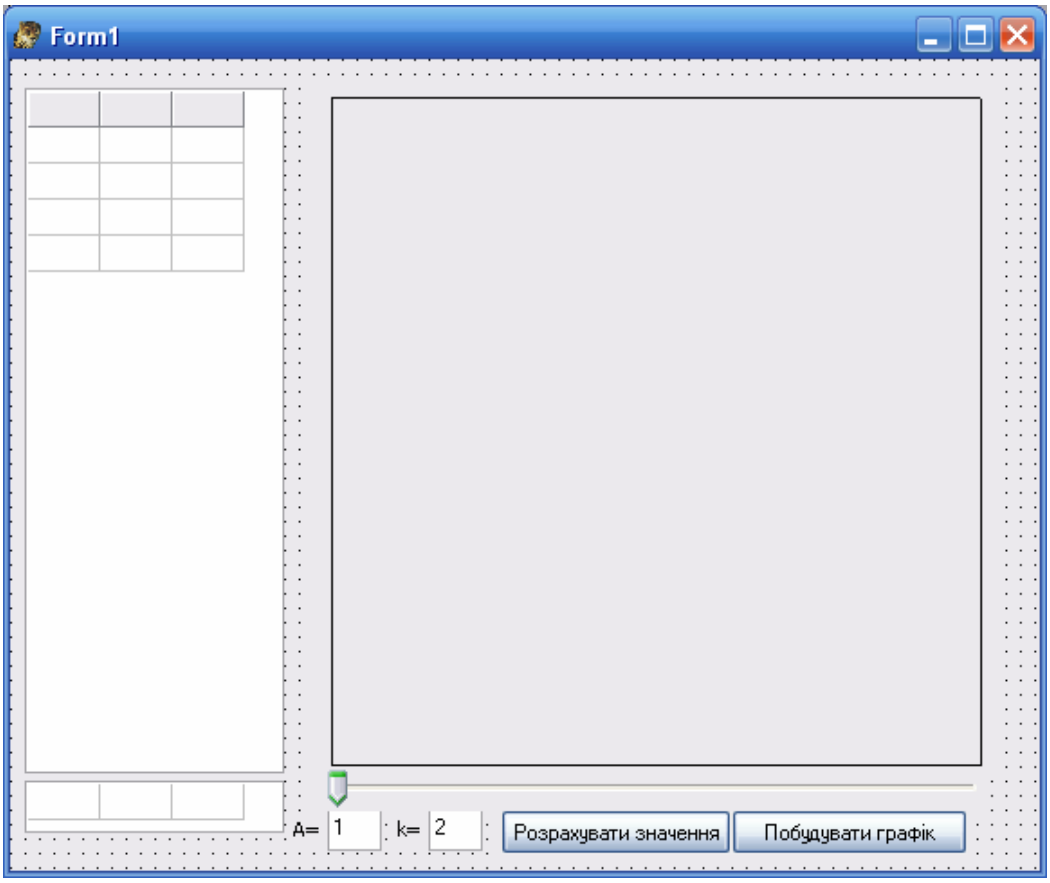


Рисунок 3.50

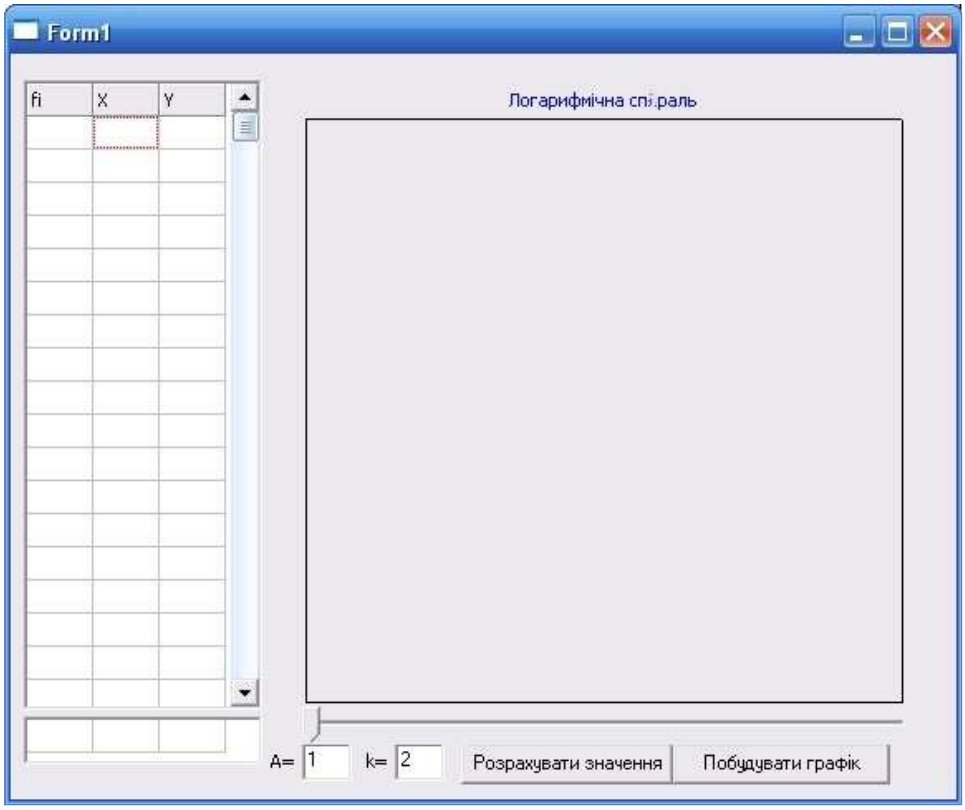


Рисунок 3.51

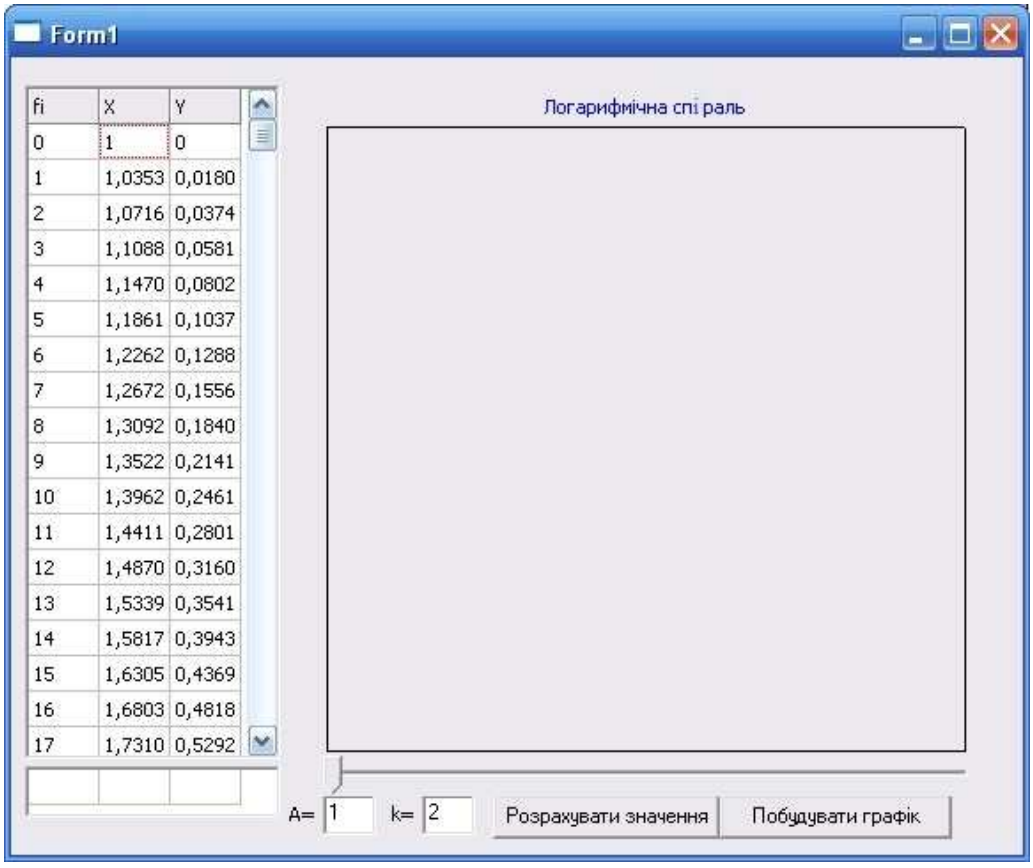


Рисунок 3.52 – Робота додатка: розрахунок значень

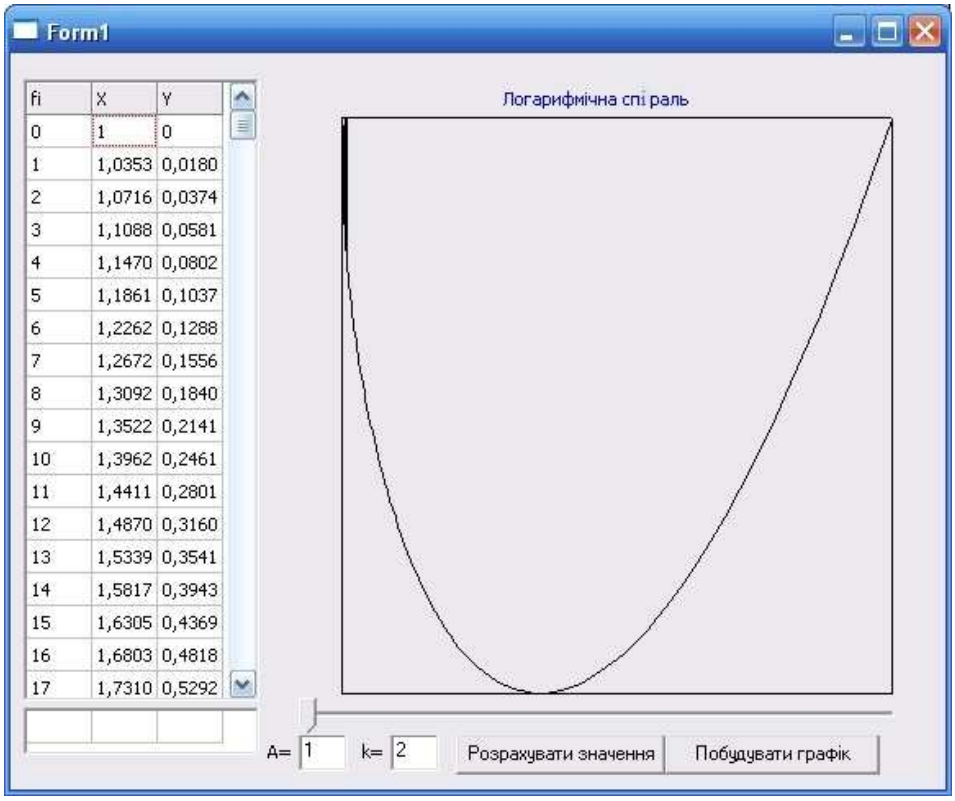


Рисунок 3.53 – Робота додатка: побудова графіка для A = 1, k = 2

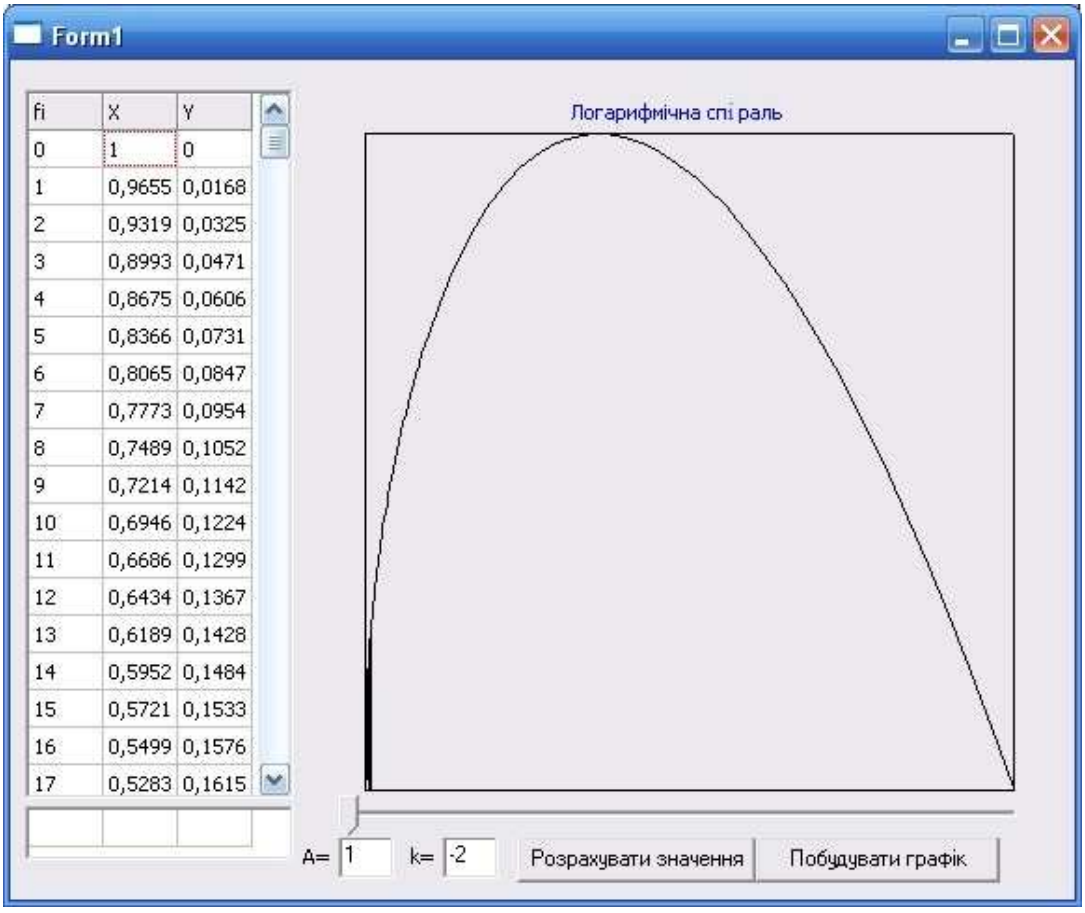


Рисунок 3.54 – Робота додатка: побудова графіка для A = 1, k =- 2

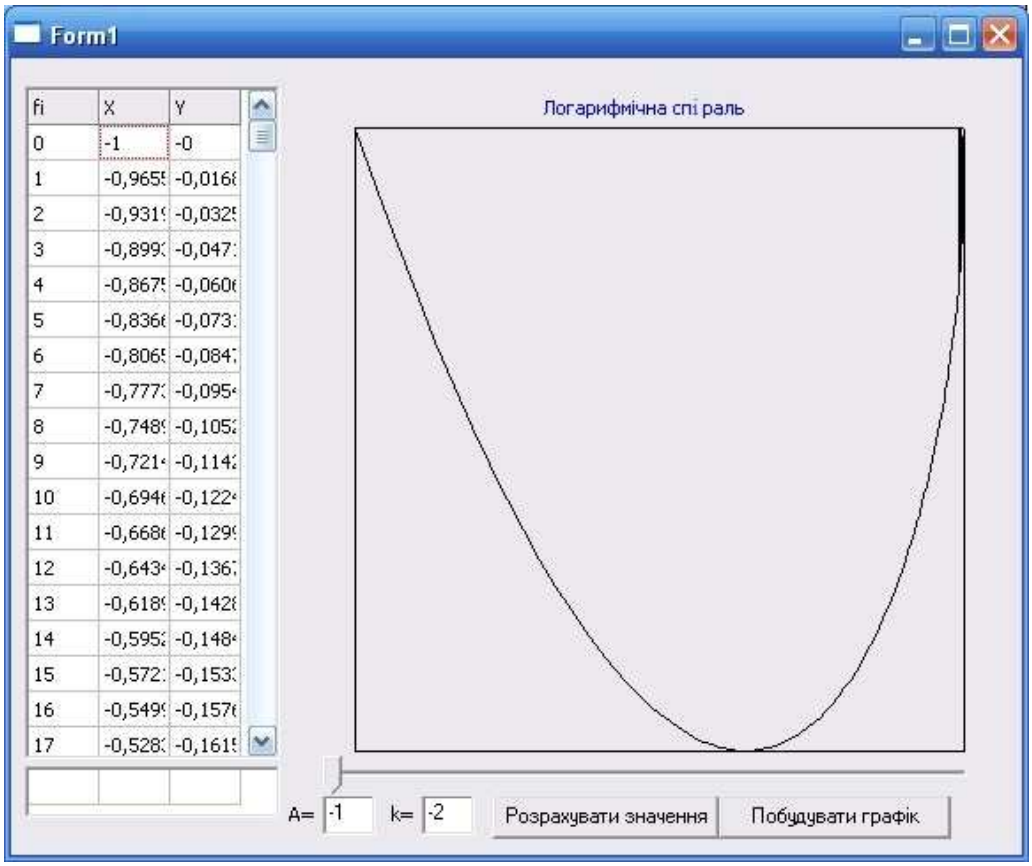


Рисунок 3.55 – Робота додатка: побудова графіка для A =-1, k =- 2

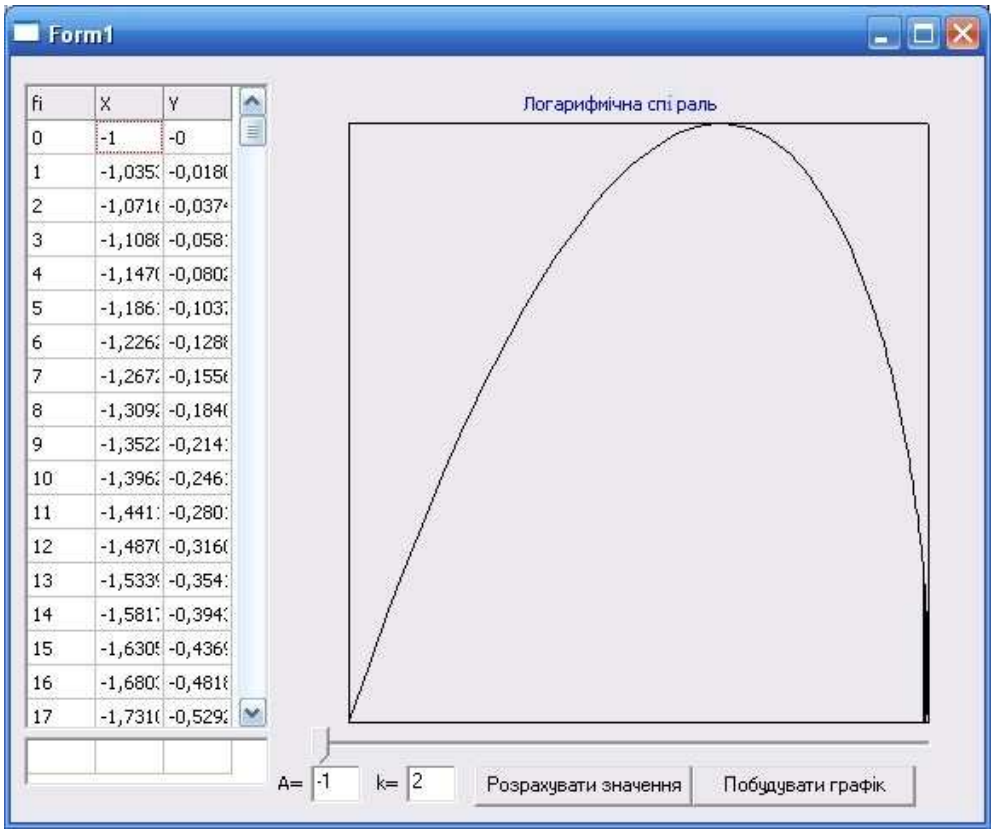


Рисунок 3.56 – Робота додатка: побудова графіка для A =-1, k = 2

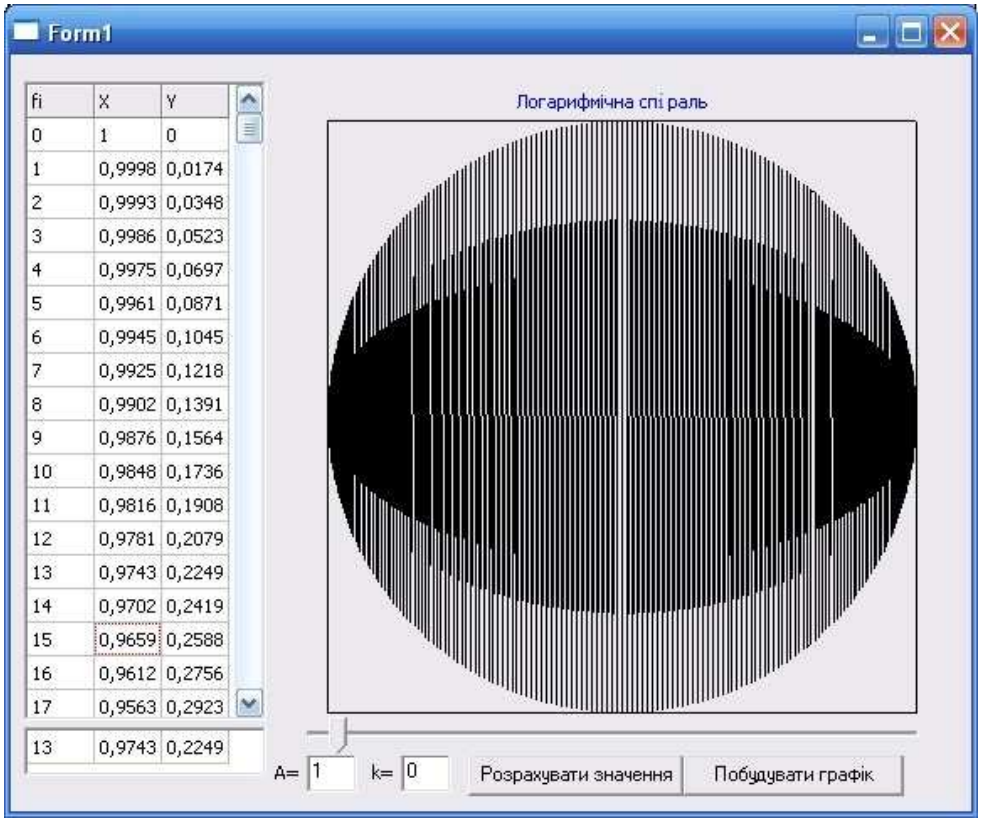


Рисунок 3.57 – Робота додатка: побудова графіка для A =1, k = 0

Висновки. Під час лабораторної роботи було створено програму для побудови графіка функції логарифмічної спіралі. Для цього використовувалися компоненти Button, LabeledEdit, StringGrid, Chart, TrackBar, Timer, ToolBar; створені методи для обробки подій FormCreate, FormPaint, ButtonClick, TrackBarChange. Додаток реалізований у вигляді однієї форми; вибір користувача здійснюється за допомогою двох стандартних кнопок і «повзунка».

Лабораторна робота 4. Робота з базами даних у середовищі Lazarus

Мета роботи. Отримати навички розроблення додатків, що дозволяють створювати й обробляти найпростіші бази даних.

Завдання до роботи. Розробити додаток, який створює базу даних про здобувачів вищої освіти у такому вигляді:

Прізвище Ім'я По батькові Група Дата_народження R1 R2 R3 R4 R5

(тут R1..R5 – рейтинги з п'яти предметів).

Наприклад: Шевченко Тарас Григорович СМ-15-5 9.09.99 77 75 81 85 92

Два записи повинні створюватися програмою автоматично, ще не менше п'яти додаються до бази безпосередньо при роботі додатка за допомогою візуальних компонент. Крім створення бази та візуальної роботи з нею, додаток повинен також надати змоги навігації базою даних. Передбачити індексування (сортування) бази даних за своїм варіантом (табл. 3.6) та її оброблення (табл. 3.7), а також фільтрацію даних (табл. 3.8).

Таблиця 3.6

Варіант	Поле для сортування
1..5	Прізвище
6..10	Ім'я
11.15	Група
16..20	Дата народження
21..25	R1
26..30	R5

Таблиця 3.7

Варіант	Умова для оброблення
1,6,11,16, 21,26	Додати нове поле – максимальний рейтинг RS з усіх предметів; розрахувати його для кожного здобувача
2,7,12,17, 22,27	Додати нове поле – мінімальний рейтинг RS з усіх предметів; розрахувати його для кожного здобувача
3,8,13,18, 23,28	Додати нове поле – середній рейтинг RS з усіх предметів; розрахувати його для кожного здобувача
4,9,14,19, 24,29	Додати нове поле – рік народження; розрахувати його для кожного здобувача шляхом аналізування поля «Дата народження»
5,10,15,20, 25,30	Додати нове поле – місяць народження; розрахувати його для кожного здобувача шляхом аналізування поля «Дата народження»

Таблиця 3.8

Вар.	Умова для фільтрації даних
1	*2*
1	Прізвища починаються з літер «А» – «К»
2	Народились пізніше 1995 року
3	Імена починаються з літер «А» – «К»
4	Мають хоча б один рейтинг «100»
5	Народились раніше 1995 року
6	Імена починаються з літер «Л» – «Я»
7	Навчаються в групах «СМ»
8	Не мають жодного рейтингу вище 89 балів
9	Вступили до вишу в 2011 році
10	Народились пізніше 1993 року
11	Не навчаються в групах «СМ»
12	Вступили до вишу не у 2011 році
13	Не мають жодного рейтингу нижче 75 балів
14	Прізвища починаються з літер «Л» – «Я»

Продовження таблиці 3.8

1	*2*
15	Не мають жодного рейтингу нижче 90 балів
17	По батькові починаються з літер «А» – «К»
18	Не мають жодного рейтингу понад 74 бали
19	По батькові починаються з літер «Л» – «Я»
20	Народились раніше 1993 року
21, 26	Мають хоча б один рейтинг нижче 75 балів
22, 27	Розрахований рейтинг RS понад 95 балів
23, 28	Розрахований рейтинг RS нижче 75 балів
24, 29	Розрахований рейтинг RS понад 74 бали
25, 30	Рік народження – 1999

Приклад виконання завдання

Завдання. Розробити додаток, який здійснює роботу з базою даних формату, що описаний вище, передбачивши індексування за прізвищем та ім'ям здобувача. Додати нове поле – середній рейтинг RS з усіх предметів; розрахувати його для кожного здобувача. Фільтрувати всіх відмінників (середній рейтинг не нижче 90 балів).

Текст програми наведено нижче, екранні форми на етапі дизайну і роботи програми – на рисунках 3.58…3.63:

```
unit Unit1;
{$mode objfpc}{$H+}
interface
uses
  Classes, SysUtils, FileUtil, LResources, Forms, Controls, Graphics,
Dialogs,  StdCtrls, Buttons, ExtCtrls, sqldb, dbf, DBGrids, db, variants, DbCtrls;
type
  { TForm1 }
  TForm1 = class(TForm)
    ButtonCalc: TButton;
    ButtonFilterOn: TButton;
    ButtonFilterOff: TButton;
    ButtonCreate: TButton;
    ButtonFirst: TButton;
    ButtonClose: TButton;
    ButtonNext: TButton;
    ButtonLast: TButton;
    ButtonPrior: TButton;
    ButtonOpen: TButton;
    Datasource1: TDatasource;
    Dbf1: TDbf;
    DBGrid1: TDBGrid;
```

```
    DBNavigator1: TDBNavigator;
    procedure ButtonCalcClick(Sender: TObject);
    procedure ButtonCloseClick(Sender: TObject);
    procedure ButtonCreateClick(Sender: TObject);
    procedure ButtonFilterOffClick(Sender: TObject);
    procedure ButtonFilterOnClick(Sender: TObject);
    procedure ButtonFirstClick(Sender: TObject);
    procedure ButtonLastClick(Sender: TObject);
    procedure ButtonNextClick(Sender: TObject);
    procedure ButtonOpenClick(Sender: TObject);
    procedure ButtonPriorClick(Sender: TObject);
  private
    { private declarations }
  public
    { public declarations }
  end;
var
  Form1: TForm1;
implementation
{ TForm1 }
procedure TForm1.ButtonOpenClick(Sender: TObject);
var i:integer;
begin
     with Dbf1 do begin
     FilePath:='';
     TableName:='mybase.dbf';
     Open; IndexName:='indFIO';
     DBGrid1.Columns[0].Title.Caption:='Прізвище';
     DBGrid1.Columns[1].Title.Caption:='І’мя';
     DBGrid1.Columns[2].Title.Caption:='По батькові';
     DBGrid1.Columns[3].Title.Caption:='Група';
     DBGrid1.Columns[4].Title.Caption:='Д/н';
     for i:=5 to 9 do DBGrid1.Columns[i].Title.Caption:='R'+IntToStr(i-4);
     for i:=1 to FieldCount do begin
     DBGrid1.Columns[i-1].Title.Alignment:=taCenter;
     case i of
     1..3: DBGrid1.Columns[i-1].Width:=
round(DBGrid1.Columns[i-1].Width*2/3);
     4,5: DBGrid1.Columns[i-1].Width:=DBGrid1.Columns[i-1].Width-1;
  else DBGrid1.Columns[i-1].Width:=DBGrid1.Columns[i-1].Width div 2
     end end;
     DBGrid1.TitleFont.Style:=[fsBold];
     end
end;
```

```
procedure TForm1.ButtonCloseClick(Sender: TObject);
begin
    Dbf1.Close
end;
procedure TForm1.ButtonCalcClick(Sender: TObject);
var i:integer;s:real;
begin
    with Dbf1 do begin
        First;
        while not EOF do begin
            s:=0;
            for i:=1 to 5 do
               s:=s+Fields[4+i].asInteger;
            s:=s/5;
            Edit;
            FieldByName('SR').asFloat:=s;
            Post;
            Next
        end
    end;
end;
procedure TForm1.ButtonCreateClick(Sender: TObject);
begin
    with Dbf1 do begin
    FilePath:='';                    // у поточній папці
    TableName:='mybase.dbf';
    with FieldDefs do begin
        Clear;
        Add('Fam',ftString,20,true);
        Add('Name',ftString,20,false);
        Add('Parent',ftString,20,false);
        Add('Group',ftString,10,false);
        Add('DR',ftDate,0,false);
        Add('R1',ftInteger,0,false);
        Add('R2',ftInteger,0,false);
        Add('R3',ftInteger,0,false);
        Add('R4',ftInteger,0,false);
        Add('R5',ftInteger,0,false);
        Add('SR',ftFloat,0,false)
        end;{FieldDefs}
    CreateTable;
    Open;
    AddIndex('indFIO','Fam+Name',[ixExpression]);
    IndexName:='indFIO';
    Append;
```

```
    FieldByName('Fam').AsString:='Іванов';
    FieldByName('Name').AsString:='Петро';
    FieldByName('Parent').AsString:='Сергійович';
    FieldByName('Group').AsString:='СМ-15-2';
    FieldByName('DR').AsDateTime:=StrToDate('12.01.95');
    FieldByName('R1').AsInteger:=55;
    FieldByName('R2').AsInteger:=75;
    FieldByName('R3').AsInteger:=80;
    FieldByName('R4').AsInteger:=65;
    FieldByName('R5').AsInteger:=81;
    Post;
    Append;
    FieldByName('Fam').AsString:='Сидорова';
    FieldByName('Name').AsString:='Ксенія';
    FieldByName('Parent').AsString:='Едуардівна';
    FieldByName('Group').AsString:='СМ-15-2';
    FieldByName('DR').AsDateTime:=StrToDate('9.09.95');
    FieldByName('R1').AsInteger:=75;
    FieldByName('R2').AsInteger:=75;
    FieldByName('R3').AsInteger:=76;
    FieldByName('R4').AsInteger:=77;
    FieldByName('R5').AsInteger:=75;
    Post;
    Close
    end;
    ButtonOpenClick(Sender)
end;
procedure TForm1.ButtonFilterOffClick(Sender: TObject);
begin
    with Dbf1 do begin Filtered:=False; Filter:='' end
end;
procedure TForm1.ButtonFilterOnClick(Sender: TObject);
begin
    with Dbf1 do begin Filter:='SR >= 90'; Filtered:=True end
end;
procedure TForm1.ButtonFirstClick(Sender: TObject);
begin
    Dbf1.First
end;
procedure TForm1.ButtonLastClick(Sender: TObject);
begin
    Dbf1.Last
end;
```

```
procedure TForm1.ButtonNextClick(Sender: TObject);
begin
    if not Dbf1.EOF then Dbf1.Next
end;
procedure TForm1.ButtonPriorClick(Sender: TObject);
begin
    if not Dbf1.BOF then Dbf1.Prior
end;
initialization
  {$I unit1.lrs}
end.
```

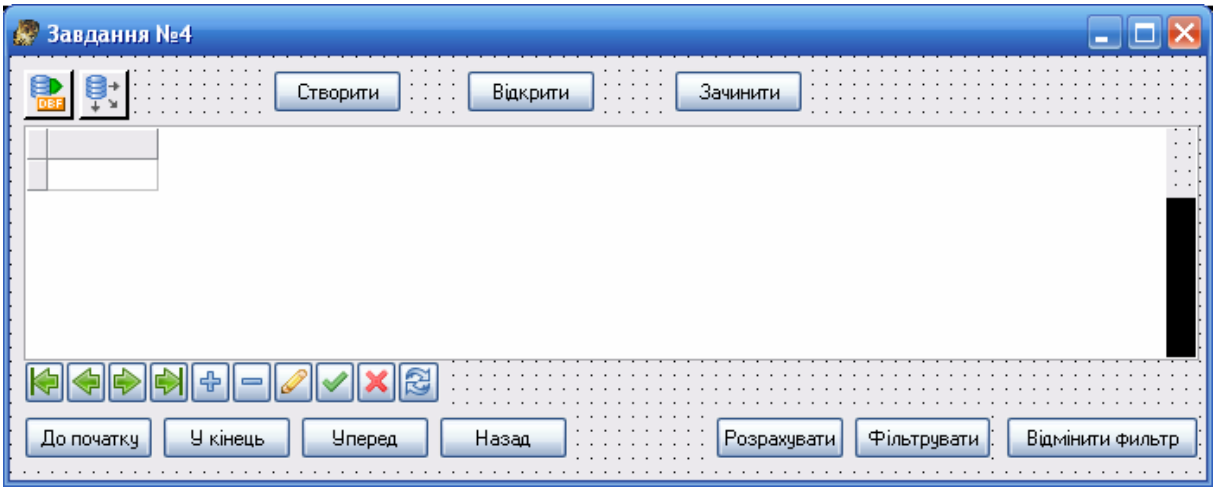


Рисунок 3.58

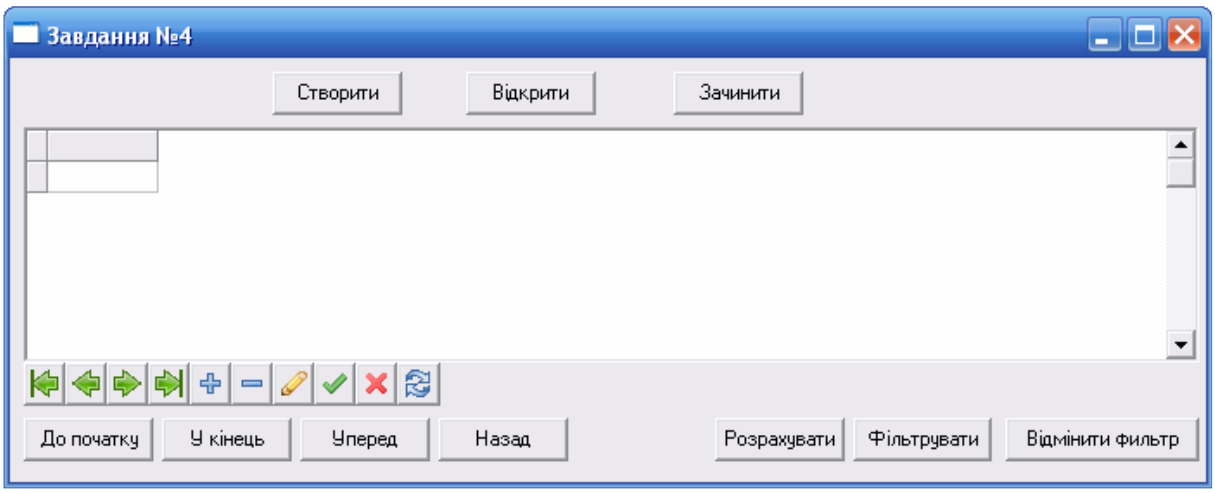


Рисунок 3.59

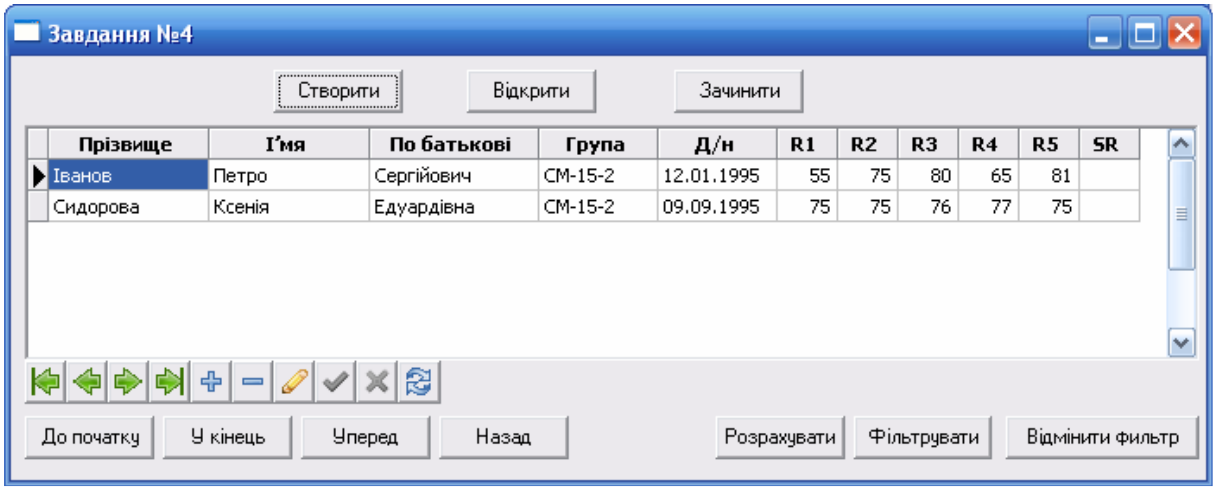


Рисунок 3.60 – Робота додатка: створення бази даних

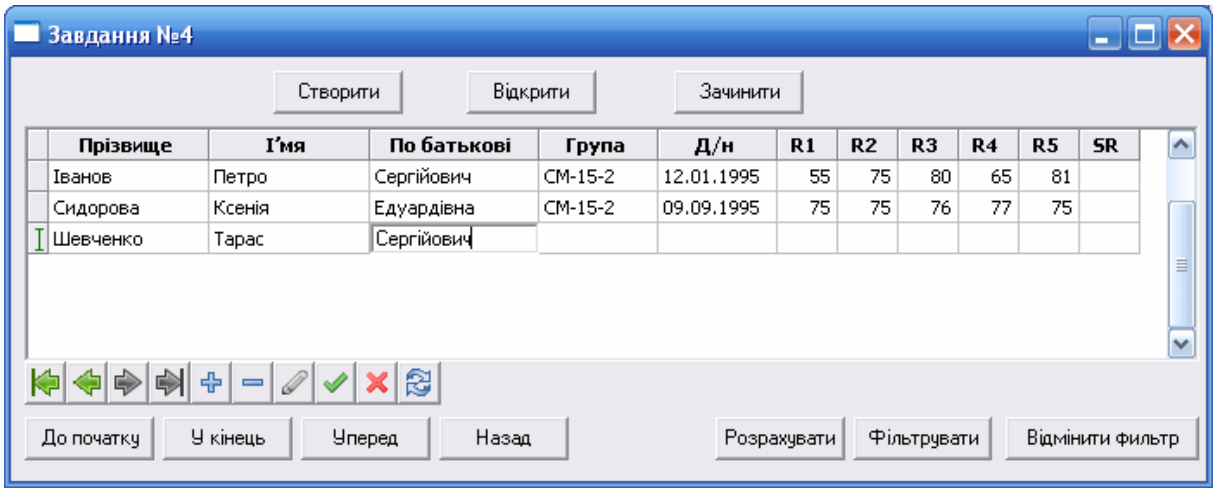


Рисунок 3.61 – Робота додатка: додавання нових записів

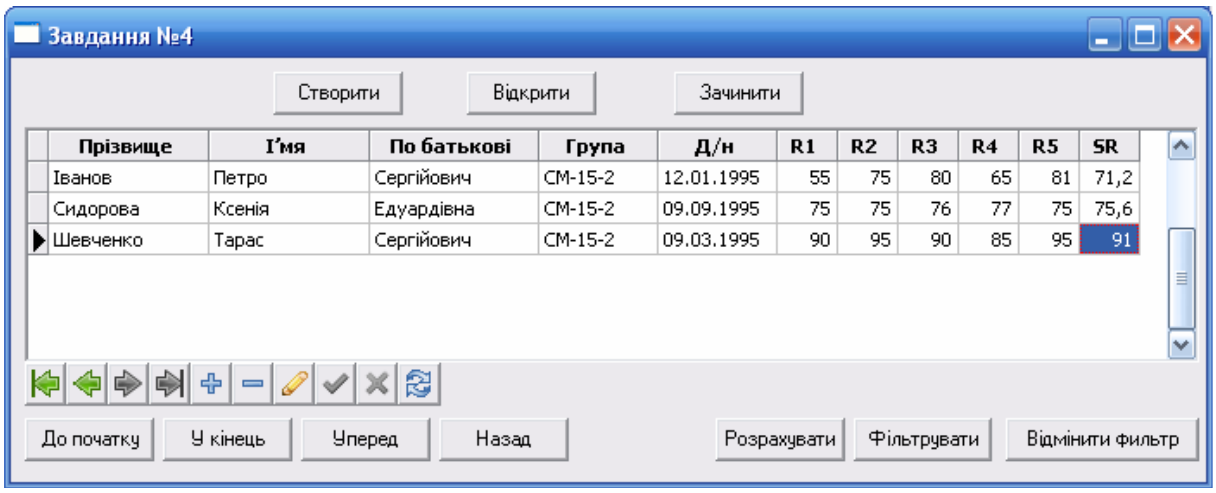


Рисунок 3.62 – Робота додатка: розрахунок середніх рейтингів

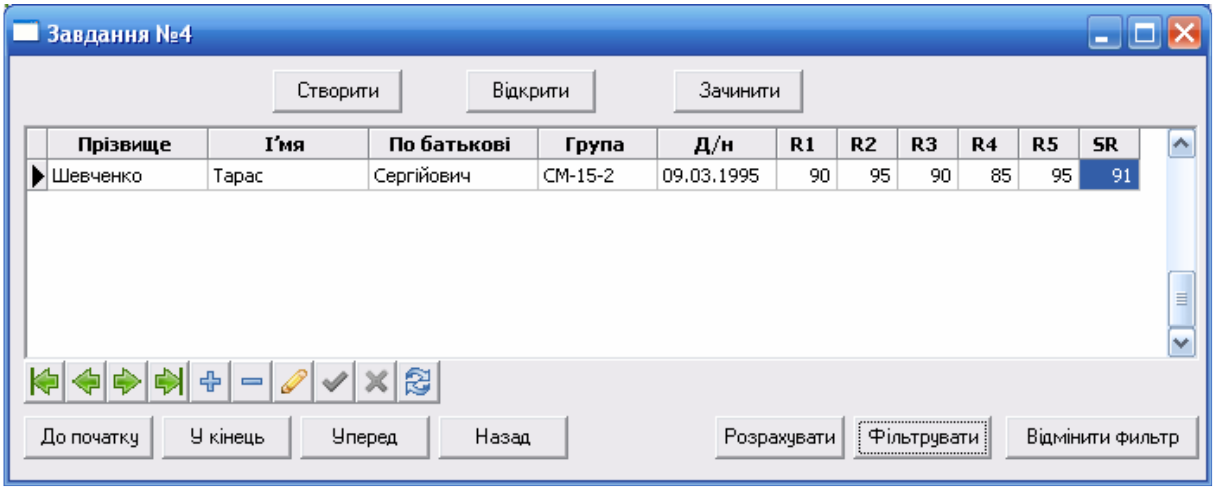


Рисунок 3.63 – Робота додатка: фільтрація

Висновки. Під час лабораторної роботи було створено програму для створення та обробки бази даних заданого формату. Для цього використовувалися компоненти Button, Datasource, Dbf, DBGrid, DBNavigator. Додаток реалізовано у вигляді однієї форми; вибір користувача здійснюється за допомогою низки стандартних кнопок. Користувачеві пропонується можливість створити нову базу даних, відкрити і закрити існуючу базу даних, переміщатися набором даних, вводити і модифікувати дані, проводити розрахунок додаткового поля (середній рейтинг), фільтрувати дані (відображати лише здобувачів-відмінників) і скасовувати фільтр.

Самостійна робота. Створення пакета для оброблення даних у середовищі Lazarus

Завдання до роботи:

1 Створити текстовий файл, що містить дані про здобувачів у такому вигляді: Прізвище Ім'я Група Рік_народження R1 R2 R3 R4 R5. Наприклад: Іванов Іван СМ-12-3 1995 77 75 81 85 92, де R1...R5 – рейтинги з п'яти предметів за 100-бальною шкалою; усього файл повинен містити не менше 10 рядків-записів.

2 Розробити повнофункціональний Windows-додаток у середовищі візуального програмування Lazarus, яке б зчитувало інформацію з файла в набір даних (компонента SdfDataSet) і здійснювало його оброблення. Додаток повинен містити, як мінімум, головне меню («Виведення даних на екран», «Виведення даних до текстового файла», «Розрахунок додаткового поля», «Сортування даних за заданим полем», «Побудова діаграм», «Вихід»), memo-поле для виведення результатів і PaintBox (Panel) для побудови діаграм. Додавання інших компонент для поліпшення інтерфейсу додатку схвалюється.

3 Імена файлів для завантаження/збереження даних повинні задаватися із застосуванням стандартних діалогових компонент (OpenDialog, SaveDialog). Виведення даних передбачається як на екран, так і в текстовий файл (за вибором користувача).

4 Додаткове поле обчислюється згідно з індивідуальним завданням (табл. 3.9). Масив сортується за заданим полем заданим методом (табл. 3.10, стов. 2 і 3).

5 Стовпчаста діаграма повинна відображати порівняння рейтингів заданих предметів і RS (RM) для кожного здобувача (номери предметів наведено в табл. 3.10, стов. 4; назви придумати самостійно). Необхідний вид діаграми наведено на рисунку 3.64. Побудова діаграм можлива як за допомогою властивості Canvas компоненти PaintBox, так і шляхом розміщення декількох компонент Shape (з автоматичним обчисленням розмірів).

Таблиця 3.9 – Варіанти для розрахунку додаткового поля

Варіант	**Умова**
1...8	Визначити максимальний рейтинг RM з усіх предметів у кожного здобувача
9…16	Визначити мінімальний рейтинг RM з усіх предметів у кожного здобувача
17...30	Визначити середній рейтинг RM з усіх предметів у кожного здобувача

Таблиця 3.10

Вар.	**Поле для сортування**	**Алгоритм сортування**	**Номер предмета**
1	*2*	*3*	*4*
1	Прізвище	Сортування обмінами	Четвертий
2	Ім’я	Сортування вставками	П’ятий
3	Група	Сортування вибором	Перший
4	Рік народження	Швидке сортування Хоора	Другий
5	Рейтинг	Піраміда Вільямса – Флойда	Третій
6	Рейтинг	Сортування обмінами	Четвертий
7	Прізвище	Сортування вставками	П’ятий
8	Ім’я	Сортування вибором	Перший
9	Група	Швидке сортування Хоора	Другий
10	Рік народження	Піраміда Вільямса – Флойда	Третій
11	Рік народження	Сортування обмінами	Четвертий
12	Рейтинг	Сортування вставками	П’ятий
13	Прізвище	Сортування вибором	Перший
14	Ім’я	Швидке сортування Хоора	Другий

Продовження таблиці 3.10

1	*2*	*3*	*4*
15	Група	Піраміда Вільямса – Флойда	Третій
16	Група	Сортування обмінами	Четвертий
17	Рік народження	Сортування вставками	П’ятий
18	Рейтинг	Сортування вибором	Перший
19	Прізвище	Швидке сортування Хоора	Другий
20	Ім’я	Піраміда Вільямса – Флойда	Третій
21	Ім’я	Сортування обмінами	Четвертий
22	Група	Сортування вставками	П’ятий
23	Рік народження	Сортування вибором	Перший
24	Рейтинг	Швидке сортування Хоора	Другий
25	Прізвище	Піраміда Вільямса – Флойда	Третій
26	Ім’я	Швидке сортування Хоора	Четвертий
27	Група	Піраміда Вільямса – Флойда	П’ятий
28	Рік народження	Швидке сортування Хоора	Перший
29	Рейтинг	Сортування обмінами	Другий
30	Рік народження	Сортування вставками	Третій

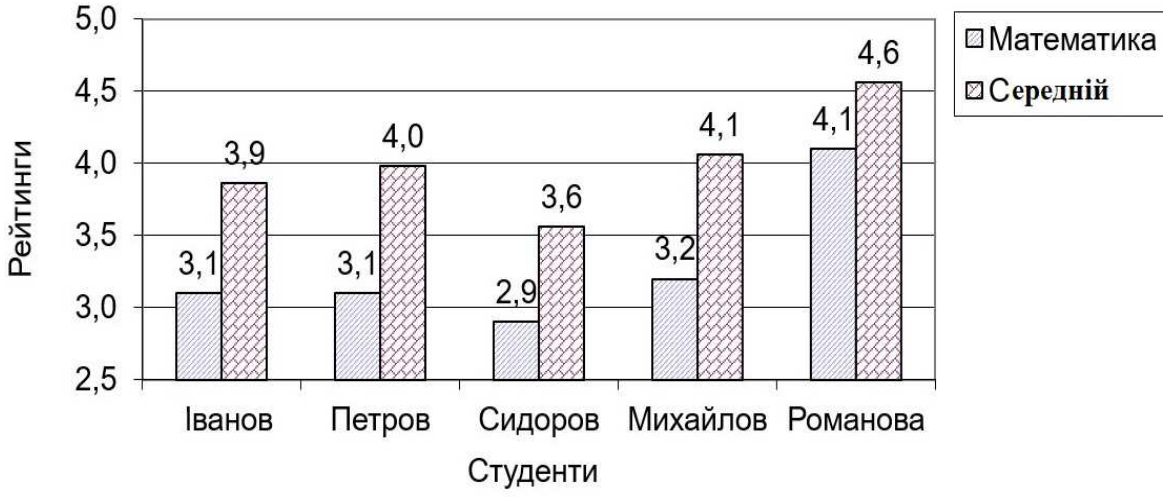


Рисунок 3.64 – Орієнтовний вигляд результатів розрахунків

4 ОСНОВНІ ПОНЯТТЯ ОБ'ЄКТНО-ОІЕНТОВАНОГО ПІДХОДУ Й ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ НА МОВАХ C++ ТА JAVA

4.1 Основні поняття об’єктно-орієнтованого підходу

Усі методи проєктування програмних систем можна об'єднати в три групи:

– структурне проєктування зверху донизу, в основі якого лежить алгоритмічна декомпозиція, і яке не може забезпечити створення складних систем і неефективне в об'єктно-орієнтованих мовах;

– метод потоків даних, коли система розглядається як перетворювач вхідних потоків на вихідні;

– об'єктно-орієнтоване проєктування.

Об'єктно-орієнтоване програмування (OOP – Object-oriented programming) – це методологія програмування, заснована на поданні програми у вигляді сукупності об'єктів, кожний з яких є екземпляром певного класу, а класи утворюють ієрархію спадкоємства. Слід розрізняти «об'єктні» («об'єктно-базовані») і «об'єктно-орієнтовані» мови програмування.

Об'єктно-орієнтоване проєктування (OOD – Object-oriented design) – це методологія проєктування, що сполучає в собі процес об'єктної декомпозиції і прийоми подання логічної і фізичної, а також статичної і динамічної моделей проєктованої системи.

Об'єктно-орієнтований аналіз (OOA – Object-oriented analysis) – це методологія, при якій вимоги до системи сприймаються з погляду класів і об'єктів, виявлених у наочній області.

У результаті об'єктно-орієнтованого аналізу формуються моделі, на яких ґрунтується об'єктно-орієнтоване проєктування, що створює фундамент для остаточної реалізації системи з використанням методології об'єктно-орієнтованого програмування. Усі три методології базуються на поняттях ***об'єктної моделі.***

Об'єктна модель

Об'єктна модель є концептуальною базою об'єктно-орієнтованого стилю. Вона складається з чотирьох головних елементів (абстрагування, інкапсуляція, модульність, ієрархія) і трьох додаткових (типізація, паралелізм, збереженність), які корисні, але не обов'язкові.

Абстрагування виділяє істотні характеристики деякого об'єкта, відрізняючи його від всіх інших видів об'єктів, і, таким чином, чітко визначає його концептуальні межі з погляду спостерігача. Головний принцип абстрагування – принцип якнайменшого здивування, згідно з яким абстракція

повинна охоплювати всю поведінку об'єкта, але не привносити сюрпризів або побічних ефектів, що лежать поза її сферою застосовності.

Вибір правильного набору абстракцій – найголовніша задача.

Абстракції можна об'єднати в 4 групи.

Абстракція суті: об'єкт є корисною моделлю якоїсь суті в наочній області.

Абстракція поведінки: об'єкт складається з узагальненої безлічі операцій.

Абстракція віртуальної машини: об'єкт групує операції, які або разом використовуються вищим рівнем управління, або самі використовують деякий набір операцій нижчого рівня.

Довільна абстракція: об'єкт уключає набір операцій, що не мають один із одним нічого спільного.

Приклади абстракцій у гідропонному господарстві: теплиця, температура, вогкість, освітлення, кислотність, датчики, культури, план вирощування.

Інкапсуляція – це процес відокремлення один від одного елементів об'єкта, що визначають його будову й поведінку; слугує для того, щоб ізолювати контрактні зобов'язання абстракції від їхньої реалізації.

Абстрагування та інкапсуляція доповнюють одне одного: перше направлене на спостережувану поведінку об’єкта, а друга займається його внутрішнім устроєм. *Інтерфейс* відображає зовнішню поведінку об’єкта, описуючи абстракцію поведінки всіх об'єктів даного класу; внутрішня *реалізація* описує подання цієї абстракції і механізми досягнення бажаної поведінки об’єкта.

(Слід звернути увагу: «Інкапсуляція не рятує від дурості; вона захищає від помилок, але не від шахрайства» – Б. Страуструп.)

Приклад: простий нагрівник має розташування (конструктор), умикач, вимикач і перевірку стану. Ускладнений нагрівник може бути пов'язаний із датчиком температури й планом вирощування.

Модульність – це властивість системи розкладатися на внутрішньо зв'язні, але слабо зв'язані між собою модулі. Логічне продовження інкапсуляції – її «практична проєкція».

Ієрархія – це впорядкування абстракцій, розташування їх за рівнями. Розрізняють два види ієрархічних структур: структура класів (is-a) і структура об'єктів (part of).

Структура класів є ієрархією спадкоємства типу «узагальнення-спеціалізація», у якій підклас (нащадок) є спеціалізованим окремим випадком свого суперкласу (предка). Приклад: «токар є окремий випадок робітника», «датчик температури є окремим випадком датчика».

Структура об'єктів є ієрархією типу «агрегація», у якій один об'єкт міститься усередині іншого. Приклад: «вікно містить смуги прокрутки, меню, рядок стану», «управляючий елемент у теплиці містить датчики температури, вогкості, кислотності й освітлення».

Типізація – це спосіб захиститися від використання об'єктів одного класу замість іншого або, принаймні, управляти таким використанням. Типізація примушує виражати абстракції так, щоб мова програмування підтримала дотримання ухвалених проєктних рішень. Розрізняють сильну й слабку типізації.

Паралелізм – це властивість, що відрізняє активні об'єкти від пасивних. Припускає паралельну (незалежну) роботу ряду активних об'єктів. При проєктуванні треба вживати заходів, які гарантують, що об'єкт не буде розтерзаний на частини декількома незалежними процесами.

Збереженість – здатність об’єкта існувати в часі, переживаючи процес, що його породив, та в просторі, переміщаючись зі свого первинного адресного простору.

Спектр збереженості об'єктів охоплює:

– проміжні результати обчислення виразів;

– локальні змінні в підпрограмах;

– власні (глобальні) змінні та динамічно створювані дані;

– дані, що зберігаються між сеансами виконання програми;

– дані, що зберігаються при переході на нову версію програми;

– дані, які взагалі переживають програму.

Традиційно, першими трьома рівнями займаються мови програмування, а останніми – бази даних. Можливі змішані рішення: програмісти розробляють спеціальні схеми для збереження об'єктів у період між запусками програми, а конструктори баз даних переінакшують свою технологію під короткоживучі об'єкти.

Існують так звані об'єктно-орієнтовані бази даних, які зберігають не тільки дані, але й класи. Ця технологія не прижилася, на практиці вважають за краще створювати об'єктно-орієнтовану оболонку для реляційних баз даних.

Переваги об'єктної моделі

1. Дозволяє повною мірою використовувати можливості об'єктних і об'єктно-орієнтованих мов програмування.

2. Підвищує рівень уніфікації розробки й придатність для повторного використання не тільки програм, але й проєктів. Використання попередніх розробок дає виграш у вартості та часі.

3. Спрощує процес внесення змін через наявність стабільних проміжних описів. Це надає змоги не переробляти систему цілком навіть у разі істотних змін початкових вимог.

4. Зменшує вірогідність допущення різноманітних помилок.

5. Орієнтована на людське сприйняття світу.

Деякі факти з історії

1969 – Кей (Kay) у своїй дисертації запропонував ідею об'єктного підходу;

1979 – Джонс (Jones) увів концепцію об'єктного підходу;

1981 – Буч (Booch) запропонував методи OOD;

1988 – Шлаєр і Меллор (Shlaer and Mellor) запропонували методи OOA;

1988 – Страуструп (Stroustrup) опублікував методичну статтю про OOP;

1991 – Румбах (Rumbaugh) об'єднав OOA і OOD.

Організації, що відповідають за стандарти за об'єктною технологією: Object Management Group (OMG) і комітет ANSI X3J7.

Класи та об'єкти

Одними з базових понять об'єктної моделі є поняття класів і об'єктів.

Об'єкт моделює частину навколишньої дійсності і, таким чином, існує в часі та просторі. Об'єкт має стан, поведінку й ідентичність; структура і поведінка схожих об'єктів визначають загальний для них клас; терміни «екземпляр класу» і «об'єкт» взаємозамінні.

Стан об’єкта характеризується переліком (зазвичай статичним) усіх властивостей певного об’єкта і поточними (зазвичай динамічними) значеннями кожної з цих властивостей. Приклад: автомат із продажу напоїв.

Поведінка визначає сукупність дій і реакцій об’єкта; виражається в термінах стану об’єкта й передачі повідомлень. Іншими словами, поведінка об’єкта – це спостережувана й перевіряльна ззовні діяльність. Стан об’єкта є сумарним результатом його поведінки.

Операцією називається певна дія одного об’єкта на інший із метою викликання відповідної реакції; це послуга, яку клас може надати своїм клієнтам. Типовий клієнт може здійснювати операції 5 видів:

Модифікатор – операція зміни стану об’єкта.

Селектор – операція, що прочитує стан об’єкта, але не змінює його.

Ітератор – операція, що дозволяє організувати доступ до всіх частин об’єкта в точній послідовності.

Конструктор – операція створення об’єкта і/або його ініціалізації.

Деструктор – операція очищення і/або руйнування об’єкта.

Сукупність усіх методів і вільних процедур, що відносяться до конкретного об’єкта, утворює *протокол* цього об’єкта.

Об'єкти можуть бути активними й пасивними. Активний об'єкт може проявляти свою поведінку без дії з боку інших об'єктів. Пасивний об'єкт, навпаки, може змінювати свій стан тільки під впливом інших об'єктів. Таким чином, активні об'єкти системи – джерела управляючих дій.

Ідентичність – це така властивість об’єкта, яка відрізняє його від усіх інших об'єктів. Слід уміти відрізняти ім'я об’єкта від самого об’єкта. Якщо абстракція є сукупністю властивостей без власної поведінки, це не об'єкт, а структура.

Система реалізується тільки в процесі взаємодії об'єктів. Відношення між об'єктами можуть бути двох типів: зв'язок і агрегація.

Зв'язок – це специфічне зіставлення, через яке клієнт запрошує послугу в об’єкта-сервера або через яке один об'єкт знаходить шлях до іншого.

Беручи участь у зв'язку, об'єкт може виконувати одну з трьох ролей:

- ***актор***: може впливати на інші об'єкти, але сам ніколи не піддається дії інших об'єктів (виключно активний об'єкт);
- ***сервер***: може тільки піддаватися дії з боку інших об'єктів, але сам ніколи не виступає в ролі впливаючого об’єкта (виключно пасивний об'єкт);
- ***агент***: може виступати як в активній, так і в пасивній ролі.

На рисунку 4.1 зображені чотири об'єкти, кожний з яких характеризує описані вище ролі.

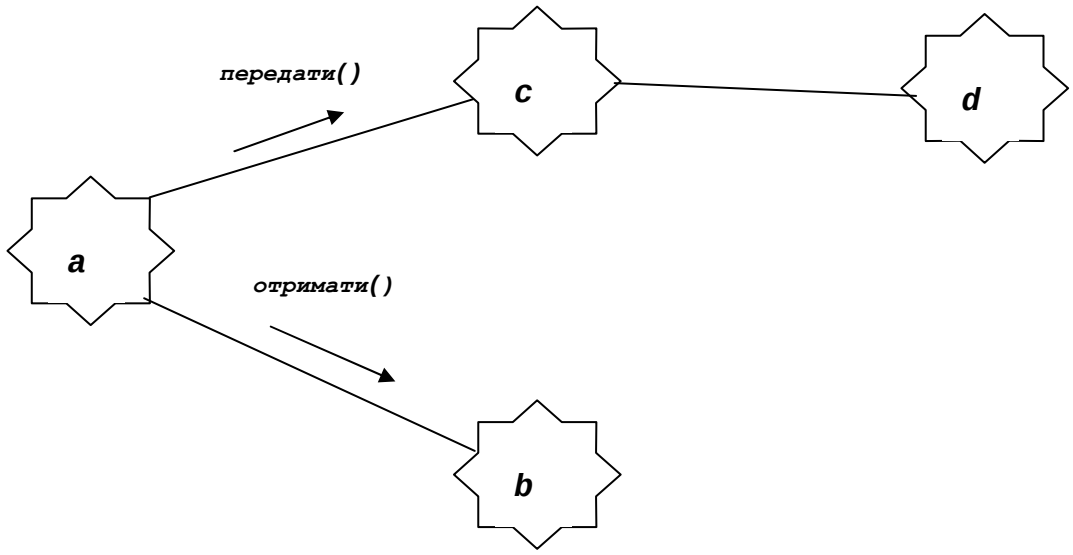


Рисунок 4.1 – Взаємодіючі об'єкти

Агрегація може означати фізичне входження одного об'єкта до іншого (автомобіль і двигун), але не обов'язково (акціонер й акції). Приклад агрегації з фізичним входженням поданий на рисунку 4.2.

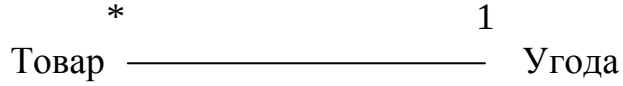
Рисунок 4.2 – Агрегація

Клас – це деяка множина об'єктів, що мають спільну структуру й спільну поведінку. У той час як об'єкт позначає конкретну сутність, клас

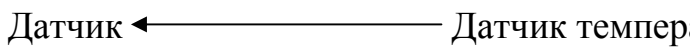
визначає лише абстракцію істотного в об'єкті. Будь-який конкретний об'єкт є просто екземпляром класу.

Підтримуються 6 (шість) видів відносин між класами.

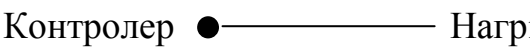
1 ***Асоціація*** – смисловий зв'язок (за замовчуванням – двосторонній), що фіксує учасників, їхні ролі й потужність відносини (1:1, 1:*, *:*):



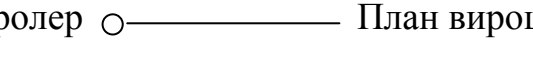
2 ***Спадкування*** – таке відношення між класами, коли один клас повторює структуру й поведінку іншого або інших класів. Установлює відношення загального й частки. Класи, екземпляри яких створюються, називаються конкретними, не створюються – абстрактними. Висновки: у будь-якого класу може бути два види клієнтів – екземпляри (об'єкти) і підкласи (спадкоємці). Зображується у вигляді стрілки, спрямованої від нащадка до предка:



3 ***Агрегація*** між класами має ту саму сутність, що й у випадку об'єктів. Розрізняють агрегацію *за значенням* (фізичне включення) і *за посиланням*. Якщо є сумніви у виборі між спадкуванням й агрегацією, краще вибрати другу. Зображується у вигляді лінії із зафарбованим колом:



4 Відношення ***використання*** між класами відповідає зв'язку між їхніми екземплярами (клієнт – серверні відносини). Зображується у вигляді лінії з порожнім колом:



5 ***Інстанціювання*** припускає використання параметризованих класів, коли як формальний параметр класу передається покажчик на якийнебудь клас, а при звертанні до його екземпляра формальний параметр заміщається фактичним. Є продовженням (або окремим випадком) відносини використання. Зображується у вигляді прямокутника-урізання в правому верхньому куті класу.

6 ***Метаклас*** – це клас, екземпляри якого є класами. Зображується у вигляді зафарбованого класу.

На етапі аналізування й ранніх стадій проєктування вирішуються два основні завдання:

- виявлення основних класів й об'єктів, що становлять словник предметної області;
- побудова структур, що забезпечують взаємодію об'єктів, при якому виконуються вимоги завдання.

Для оцінювання якості класів й об'єктів, зазначених у системі, можна запропонувати такі п'ять критеріїв: зачеплення, зв’язність, достатність, повнота й примітивність.

Зачеплення визначає степінь глибини зв'язків між окремими модулями (чим менше, тим краще). Приклад неправильного підходу: джерело живлення стереосистеми розміщений в одному зі стовпчиків.

Зв’язність – це степінь взаємодії між елементами окремого модуля (класу, об'єкта), характеристика його насиченості: «Клас Dog буде функціонально зв'язаним, якщо він описує поведінку собаки, і нікого більше, крім собаки».

Достатність – наявність у класі або модулі всього необхідного для реалізації логічної й ефективної поведінки.

Під ***повнотою*** мається на увазі наявність в інтерфейсній частині класу всіх характеристик абстракції; інтерфейс повинен гарантувати все для взаємодії з користувачами.

Примітивними є тільки такі операції, які вимагають доступу до внутрішньої реалізації абстракції.

Класифікація

Класифікація – це засіб упорядкування знань. Історично відомі тільки ***три*** підходи: класична категоризація, концептуальна кластерізація, теорія прототипів.

При ***класичному підході*** всі речі, що мають дану властивість або сукупність властивостей, формують деяку категорію. Причому наявність цих властивостей є необхідною й достатньою умовою, що визначає категорію.

Однак природні категорії нечітко відокремлені одна від одної. Більшість птахів літає, але не всі. Стілець може бути дерев'яним, металевим або пластмасовим, а кількість ніжок необов'язково дорівнює чотирьом.

Концептуальна кластерізація – сучасний варіант класичного підходу. Тут спочатку формуються концептуальні описи класів («кластерів об'єктів»), а потім ми класифікуємо сутності відповідно до їх описів. Це саме поняття, а не ознака або властивість: «Якщо пісня скоріше про любов, чим про щось інше, то ми відносимо її до категорії *любовна пісня*, хоча степінь любовності навряд чи можна виміряти».

Концептуальну кластерізацію можна зв'язати з теорією нечітких (багатозначних) множин, у якій об'єкт може належати до декількох категорій одночасно з різним степенем точності.

Однак існують деякі абстракції, які не мають ні чітких властивостей, ні чіткого визначення. ***Теорія прототипів*** визначає клас одним об'єктом-прототипом; новий об'єкт відноситься до класу за умови, що він наділений істотною подібністю із прототипом. Приклад: ігри.

На практиці ми спочатку ідентифікуємо класи й об'єкти за властивостями, важливими у даній ситуації, тобто намагаємося зазначити й відібрати структури й типи поведінки за допомогою словника предметної області.

Якщо таким шляхом не вдалося побудувати нормальної структури класів, ми пробуємо концептуальний підхід: приділяємо увагу поведінці об'єктів. Нарешті, пробуємо зазначити прототипи й асоціювати з ними об'єкти. Ці три способи класифікації становлять теоретичну основу об'єктно-орієнтованого аналізу.

Межі між стадіями аналізування й проєктування розмиті, однак у процесі аналізування ми моделюємо проблему, *виявляючи* класи й об'єкти, які становлять словник предметної області, у той час як на стадії проєктування ми *винаходимо* абстракції й механізми, що забезпечують необхідну поведінку.

Існує ***сім*** перевірених практикою підходів до аналізування об'єктно-орієнтованих систем.

1 ***Класичні підходи*** – спираються на класичну категоризацію.

Шлаєр і Меллор: відчутні предмети (датчики), ролі (робітник, студент), події (запит, переривання), взаємодія (зустріч, перетинання).

Росс (БД): люди (виконують певні функції), місця (області, пов'язані з людьми або предметами), предмети (відчутний матеріальний об'єкт), організації (формально організована сукупність людей і предметів, що має певну мету й не залежить від окремих індивідуумів), концепції (принципи й ідеї), події.

Коад і Йордан: структури (відносини «ціле – частина» й «загальне – частка»), інші системи (зовнішні), пристрою, події, ролі, місця, організаційні одиниці.

2 При ***аналізуванні поведінки*** саме динамічна поведінка розглядається як першоджерело класів й об'єктів; засноване на концептуальній кластерізації. Ініціатори певної поведінки його учасників розпізнаються як об'єкти й становляться відповідальними за певні ролі. Приклади: уведення/виведення, запит.

3 ***Аналізування предметної області*** – спроба зазначити ті об'єкти, операції й зв'язки, які експерти даної області вважають найбільш важливими.

Етапи: побудова базової моделі предметної області; вивчення існуючих у даній області систем; визначення подібності й розходжень між системами; уточнення загальної моделі для пристосування до потреб конкретної системи.

Приклад: предметна область – бухгалтерські звіти; визначаються абстракції й механізми, що обслуговують усі види звітів; мета вихідного завдання – створення системи звітів.

Експерт – майбутній користувач системи (бухгалтер, диспетчер).

4 Окремо всі перелічені вище підходи дуже залежать від індивідуальних здатностей і досвіду аналітика. ***Аналізування варіантів*** передбачає впорядковане послідовне їхнє використання. Заснований на перелічуванні й ретельному опрацюванні сценаріїв роботи системи.

5 ***Метод CRC-карток*** вдало реалізує на практиці попередній підхід (Class / Responsibilities / Collaborators – Клас / Відповідальності / Учасники). Зверху на картці пишеться назва класу, знизу в лівій половині –

за що він відповідає, у правій – з ким він співробітничає. Прохід сценарієм викликає дописування нових пунктів в існуючих картках або створення нових. Розташування карток може показати потік повідомлень між об'єктами й ієрархію класів.

6 ***Неформальний опис*** – радикальна альтернатива класичному аналізуванню (Аббот). Треба описати завдання або його частину на звичайній (людській) мові, а потім підкреслити іменники й дієслова. Іменники – кандидати на роль класів, а дієслова можуть стати іменами операцій. Метод можна автоматизувати. Використовується в Токійському технологічному інституті й в Fujitsu.

7 Друга альтернатива класичній техніці – використання ***структурного аналізу*** як основи об'єктно-орієнтованого проєктування. Після його проведення ми вже маємо модель системи, описану діаграмами потоків даних; виходячи із цього, можна почати визначення класів й об'єктів будь-якими іншими способами.

4.2 Клас як абстрактний тип даних

Клас – це похідний структурований тип, що задає деяку сукупність типізованих даних і дозволяє визначити набір операцій над цими даними.

Визначається за допомогою конструкції:

ключ_класу ім'я_класу { список_компонентів };

де ***ключ_класу*** – одне зі службових слів class, struct або union;

ім'я_класу – довільно вибраний ідентифікатор;

список_компонентів – визначення й опис типізованих даних і приналежних до класу функцій (***тіло класу***).

Компонентами класу можуть бути дані, функції, класи, перерахування, бітові поля, дружні функції, дружні класи й імена типів. Інакше кажучи, компоненти класу – це типізовані дані (базові й похідні) і функції.

Приналежні класу функції називаються ***методами класу*** або ***компонентними функціями***. Дані класу називають ***компонентними даними*** або ***елементами даних класу***.

Клас відрізняється від структури можливістю включення компонентних функцій:

```
struct complex1          // Клас «комплексне число»
{
 double real;            // Речовинна частина
 double imag;            // Уявна частина
 void define (double re=0.0, double im=0.0)
 { real=re; imag=im; }
```

```
 void display ()
 { printf("real=%5.2f, imag=%5.2f\n",real,imag); }
};
```

Клас має права типу, для опису об'єкта класу використовується запис:
ім'я_класу ім'я_об'єкта;

Наприклад:
```
complex1 x1,x2,D;
complex1 *p=&D;
complex1 dim[8];
```

До компонентів класу можна звертатися за допомогою:
1) уточнених імен: ***ім'я_об'єкта.ім'я_компонента;***
2) «кваліфікованих» імен: ***ім'я_класу::ім'я_компонента;***
3) покажчиків на об'єкт класу:
покажчик_на_об'єкт_класу -> ім'я_компонента

Приклади:
```
x1.define();              // Параметри вибираються за замовчуванням
x2.define(4.3,20.0);      // 4.3 + i 20.0
x2.display();             // Виведення на екран: real=  4.3, imag= 20.0
p->real=2.3;
p->imag=6.1;
p->display();
```

Визначення даних класу аналогічно опису об'єктів базових і похідних типів, за деякими виключеннями: при описі елементів класу не допускається їхня ініціалізація.

Помилка: struct complex1 { ... double real=0; ... };

Розглянемо клас, що описує товари на складі магазину. Компонентами класу будуть:
- назва товару;
- оптова (закупівельна) ціна;
- роздрібна (торговельна) націнка;
- функція уведення даних про товар;
- функція виводу відомостей про товар із вказівкою роздрібної ціни.

```
#include <stdio.h>
class goods          // Клас «товари»
{
 char name[40];      // Назва
 float price;        // Закупівельна ціна
 static int percent; // Торговельна націнка в %
```

```
 void input ()
 {
   printf("Назва товару: "); scanf("%s", name);
   printf("Закупівельна ціна: "); scanf("%i", price);
 }
 void display ()
 { printf("%10s\t, ціна %5.2f\n",name,price*(1+goods::percent*0.01)); }
};
void main()
{
goods wares[5];
…
}
```

Торговельна націнка визначена як статичний компонент класу. Такі компоненти не дублюються при створенні об'єктів (екземплярів) класу, тобто кожен статичний компонент існує в єдиному екземплярі. Доступ до нього можливий тільки після ініціалізації:

тип ім'я_класу::ім'я_компонента=ініціалізатор;

Наприклад: int goods::percent=12;

Статичні компоненти класу мають практичний сенс при роботі зі зв'язними списками, указуючи на їхні початки:

```
class list
{
 …
 public:
 static list *begin;   // Початок списку
 …
};
list *list::begin=NULL;    // Ініціалізація статичного компонента
```

Головний недолік обох розглянутих прикладів: відсутність автоматичної ініціалізації створюваних об'єктів – необхідно явно викликати функції define(), input() або привласнювати значення компонентним даним об'єкта.

Для автоматичної ініціалізації компонентних даних об'єкта до визначення класу можна явно включати спеціальну компонентну функцію, яка має назву ***конструктор***:

ім'я_класу (список_формальних_параметрів)
{ оператори_тіла_конструктора };

Ім'я такої функції повинне збігатися з ім'ям класу. Вона буде автоматично викликатися при визначенні або розміщенні в пам'яті кожного об'єкта класу. Переробимо частину першого прикладу:

```
complex1 (double re=0.0, double im=0.0)
 { real=re; imag=im; }
```

Оголошення даних набуде вигляду

```
complex1 x1,x2(4.3,20.0);
```

Особливості конструкторів

1 Для конструктора не визначається тип значення, що повертається, (навіть *void*).

2 Конструктор завжди автоматично викликається при визначенні (створенні) об'єкта класу. При відсутності параметрів використаються значення за замовчуванням.

3 Конструктор існує для будь-якого класу, причому він може бути створений без явних указівок програміста. Наприклад, конструктор копіювання: class F { ... public: F (const F&); ... }

4 У класі може бути кілька конструкторів (перевантаження), але тільки один зі значеннями параметрів за замовчуванням.

5 Не можна одержати адресу конструктора (&complex1).

6 Параметром конструктора не може бути його власний клас, але може бути посилання на нього, як у конструктора копіювання.

7 Конструктор не можна викликати як звичайну компонентну функцію.

Існує два способи ініціалізації даних об'єкта за допомогою конструкторів.

Перший – передача значень параметрів до тіла конструктора (див. попередній приклад).

Другий – застосування списку ініціалізаторів даного об'єкта, що розміщується між списком параметрів і тілом конструктора:

ім'я_класу (список_формальних_параметрів): список_инициализаторов
{ тіло_конструктора };

Кожен ініціалізатор списку відноситься до конкретного компонента й має вигляд

ім'я_компонента_даних (вираз)

Наприклад:

```
class AZ
{ int ii; float ff; char cc;
  public:
  AZ(int in, float fn, char cn): ii(5), ff(ii*fn+in), cc(cn) {}
  …
}
AZ A(2,3.0,’d’);    // Одержуємо: A.ii=5, A.ff=17, A.cc='d'
```

Деструктор (руйнівник об'єктів) класу має формат
~имя_класу () { оператори_тіла_деструктора };

Особливості деструкторів

1 Назва складається зі значка «~» й імені класу (без пробілів).

2 У деструктора не може бути параметрів (навіть *void*).

3 Деструктор не має значення, що повертається (навіть *void*).

4 Виклик деструктора здійснюється неявно, автоматично, як тільки об'єкт класу знищується.

Застосування деструкторів має сенс при роботі з динамічною пам'яттю:
~stroka() { delete [] ch; }

Видимість (доступність) компонентів

Усі компоненти класу можуть мати три степені доступу («видимості»):

public («загальнодоступний») – доступні для інших компонентів даного класу, інших класів і всіх інших компонентів програми;

protected («захищений») – доступні для інших компонентів даного класу і його нащадків (при побудові ієрархії класів);

private («власний») – доступні тільки для інших компонентів даного класу.

Специфікатор доступу являє собою одне з указаних вище слів, за яким розміщено двокрапку. Усе компоненти від цього місця до кінця визначення класу або до іншого специфікатора змінюють статус на вказаний вище.

Основний принцип ООП – інкапсуляція (приховання) даних усередині об'єктів – не виконується для класу *struct*, оскільки всі його компоненти одержують статус *public*. Усі компоненти класу, описаного за допомогою службового слова *class*, завжди одержують статус *private*.

```
struct complex1     // Клас «комплексне число»
{
// За замовчуванням – public
complex1 (double re=0.0, double im=0.0)
 { real=re; imag=im; }
 void display ()
 { printf("real=%5.2f, imag=%5.2f\n",real,imag); }
private:            // Змінити статус доступу на «приватний»
double real;        // Речовинна частина
double imag;        // Уявна частина
};
```

Для доступу до компонентних даних з операторів, які виконуються поза визначенням класу, безпосереднє використання імен елементів неприпустиме. Формально така заборона можна обійти перевизначенням

степеня видимості на «public», але фактично це суперечить основному принципу ООП.

На статичні дані класу також поширюються правила статусу доступу. До моменту звертання до таких даних об'єкти можуть бути ще не визначені, або їх може бути декілька. Можливість обійтися без імені конкретного об'єкта при звертанні до статичних даних класу забезпечують статичні компонентні функції:

ім'я_класу::ім'я_статичної_функції

На відміну від звичайних (глобальних) функцій, компонентна функція має доступ до всіх компонентів класу, з будь-яким статусом доступу.

При визначенні класів їхні компонентні функції можуть бути специфіковані як ті, що підставляються (inline). За замовчуванням вважається функція як та, що підставляється, чиє визначення (не тільки заголовок, але й тіло) повністю розміщене в тілі класу (див. усі попередні приклади).

Перевага: швидкий виклик. *Недоліки*: така функція не може містити цикли й перемикачі, а також не може бути рекурсивною.

Звичайно використається інший підхід: у тілі класу розміщається тільки прототип (заголовок) компонентної функції, а її визначення – поза класом, як визначення звичайної програмної функції (з використанням «кваліфікованого запису»). Опис класу із зовнішнім визначенням його компонентних функцій надає змоги, не міняючи інтерфейс об'єктів класу, по-різному визначати його компонентні функції.

Наведений нижче приклад використовує клас «точка на екрані».

```
#include <graphics.h>
#include <conio.h>
class pixel
{
 protected:
  int x,y;                          // Координати точки
 public:
  point(int xi=0, int yi=0);        // Конструктор
  int getx();                       // Координата за X
  int gety();                       // Координата за Y
  void show();                      // Зобразити на екрані
  void move(int xn=0, int yn=0);// Перемістити в нове місце
 private:
  void hide();                      // Вилучити точку з екрана
};
pixel::pixel(int xi=0, int yi=0)
{ x=xi; y=yi; }
int pixel::getx() { return x; }
```

```
int pixel::gety() { return y; }
void pixel::show() { putpixel(x,y,getcolor()); }
void pixel::hide() { putpixel(x,y,getbkcolor()); }
void pixel::move(int xn=0, int yn=0)
{
 hide();
 x=xn; y=yn;
 show();
}
void main()
{
 pixel A(200,100), B;
 int gd=0,gm;
 initgraph(&gd,&gm,"");
 A.show(); getch();
 B.show(); getch();
 B.move(200,200); getch();
 A.move(); getch();
 closegraph();
}
```

Коли функція, що належить класу, викликається для оброблення даних конкретного об'єкта, цій функції автоматично й неявно передається покажчик на той об'єкт, для якого функція викликана (згадайте Sender:TObject в Delphi / Lazarus). Цей покажчик має фіксоване ім'я this і непомітно для програміста визначений у кожній функції класу в такий спосіб:

ім'я_класу * const this = адреса_об'єкта;

Ім'я this є службовим словом. Явно визначити або перевизначити цей покажчик не можна й не потрібно.

Виклик за посиланням:

```
{
this->real=5; this->imag=2;
}
```

Дружньою функцією класу називається функція, що, не будучи його компонентом, має доступ до його захищених (protected) і власних (private) компонентів. Функція не може стати другом класу без його згоди. Для одержання прав друга вона повинна бути описана в тілі класу зі специфікатором ***friend*** (тобто в тілі класу розташовується прототип функції). Використання механізму дружніх функцій дозволяє спростити інтерфейс між класами: із класів можна забрати деякі компонентні функції, призначені для доступу до певних «схованих» компонентів.

Особливості дружніх функцій

1 Дружня функція при виклику не одержує покажчика this.

2 На дружню функцію не поширюється дія специфікаторів доступу; місце розташування її прототипу усередині класу не має значення.

3 Дружня функція може бути як глобальною функцією програми, так і компонентною функцією іншого класу.

4 Функція може бути дружньою стосовно декількох класів.

Поширення дії (перевантаження) стандартних операцій

Мова C++ дозволяє поширити дію будь-якої стандартної операції на нові типи даних, що вводять користувачі. Така можливість називається ***перевантаженням стандартних операцій***. Процес здійснюється за допомогою спеціального ***«оператора функції»***:

тип operator знак_операції (специфікація_параметрів) { оператори_тіла }

де кількість параметрів залежить від «-арності» операції.

Наприклад, для поширення дії операції '*' (помножити) на об'єкти класу T, може бути уведена функція

T operator * (T x, T y)

Таким чином, якщо описані два об'єкти A і B класу T, то вираз A*B інтерпретується як виклик функції operator * (A, B).

Приклад перевантаження операції додавання для рядків:

stroka& operator + (stroka& A, stroka& B)

Приклад заміни операції додавання на множення:

```
int operator + (int a, int b)
                { return a*b;}
int k=5+2; // k=10
```

4.3 Спадкування та інші можливості класів

Обґрунтовано уведений до програми об'єкт покликаний моделювати властивості й поведінку деякого фрагмента розв'язуваного завдання, зв'язуючи в єдине ціле дані й методи, що відносяться до цього фрагмента. Об'єкти взаємодіють між собою й з іншими частинами програми за допомогою повідомлень. У кожному повідомленні об'єкту передається деяка інформація. У відповідь на повідомлення об'єкт виконує деяку дію, передбачену набором компонентних функцій того класу, якому він належить. Такою дією може бути зміна внутрішнього стану (зміна даних) об'єкта або передача повідомлення іншому об'єкту.

Кожен об'єкт є представником конкретного класу. Об'єкти різних класів і самих класів можуть перебувати у відносинах спадкування, при якому формується ієрархія об'єктів, що відповідає заздалегідь передбаченій ієрархії класів.

Ієрархія класів дозволяє визначати нові класи на основі вже наявних. Наявні класи звичайно називають ***базовими*** (***породжаючими***), а нові класи, формовані на основі базових, – ***похідними*** (***породженими***), ***класами-нащадками*** або ***спадкоємцями***. Похідні класи «одержують спадщину» – дані й методи своїх базових класів – і, крім того, можуть поповнюватися власними компонентами (даними й методами). Наслідувані компоненти не переміщаються в похідний клас, а залишаються в базових класах. Повідомлення, обробку якого не можуть виконати методи похідного класу, автоматично передається до базового класу. Якщо для обробки повідомлення потрібні дані, відсутні в похідному класі, то їх намагаються відшукати автоматично й непомітно в базовому класі.

Приклад: ієрархія «точка – коло – коло_у_квадраті».

При спадкуванні деякі імена методів (компонентних функцій) і (або) компонентних даних базового класу можуть бути по-новому визначені в похідному класі. У цьому випадку відповідні компоненти базового класу стають недоступні з похідного класу. Для доступу використається операція «::» – уточнення області видимості.

Спадкування в ієрархії класів може відображатися й у вигляді дерева, й у вигляді більш загального *спрямованого ациклічного графа*, у якому показані відносини між об'єктами.

C++ допускає *множинне спадкування* – можливість для деякого класу успадковувати компоненти декількох ніяк не зв'язаних між собою базових класів (рис. 4.3).

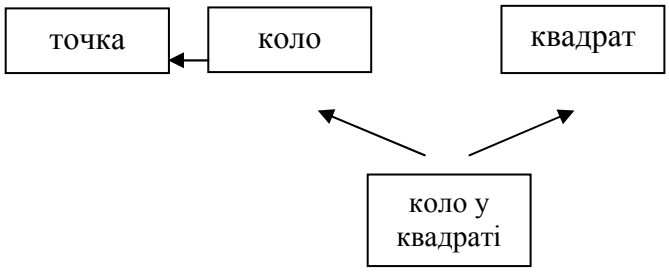


Рисунок 4.3

При описі похідного класу наводиться список усіх його базових класів:

```
class S: X, Y, Z {...};
```

Нащадкові доступні тільки ті компоненти предків, які мають статус доступу ***public*** й ***protected***. У породженому класі всі вони одержують

статус ***private***, якщо новий клас визначений словом *class*, і статус ***public***, якщо визначено словом *struct*. Зрозуміло, статус компонентів можна змінювати явною вказівкою нового статусу – як окремих компонентів, так і всього наслідуваного класу:

class S: X, protected Y, public Z {...};

Ні базовий клас, ні похідний не можуть бути оголошені за допомогою ключового слова *union*! Об'єднання не можуть використатися при побудові ієрархії класів.

Конструктори й деструктори не успадковуються, а створюються заново в кожному класі-нащадку. Конструктор базового класу завжди викликається й виконується до конструктора похідного класу. Деструктор похідного класу викликає деструктори базових класів, тобто порядок знищення об'єктів протилежний стосовно порядку його конструювання. Конструктори й деструктори автоматично одержують статус доступу ***public***.

Приклад: ~krug_square() {~krug(); ~square();}

У будь-якому класі можуть бути як компоненти визначені інші класи. У цих класах будуть свої деструктори, які при знищенні об'єкта охоплюючого (зовнішнього) класу виконуються після деструктора охоплюючого класу.

Множинне спадкування й віртуальні базові класи

Клас називають ***безпосереднім (прямим) базовим класом*** (прямою базою), якщо він входить до списка базових при визначенні класу. Водночас для похідного класу можуть існувати ***непрямі*** попередники, які служать базовими для класів, що входять до списку базових.

A (базовий клас – пряма база для B)

↑

B (похідний від A клас – пряма база для C)

↑

C (похідний клас – із прямою базою B і непрямою A)

Клас може мати кілька прямих базових класів, тобто може бути породжений з будь-якої кількості базових класів (див. приклад раніше). Така ситуація називається ***множинним спадкуванням***.

При множинному спадкуванні ніякий клас не може більше одного разу використатися в якості безпосереднього базового. Однак клас може скільки завгодно разів бути непрямим базовим класом:

class X {…; f (); …};

class Y: public X {…};

class Z: public X {…};

class D: public Y, public Z {…};

У даному прикладі клас X двічі опосередковано успадковується класом D (рис. 4.4).

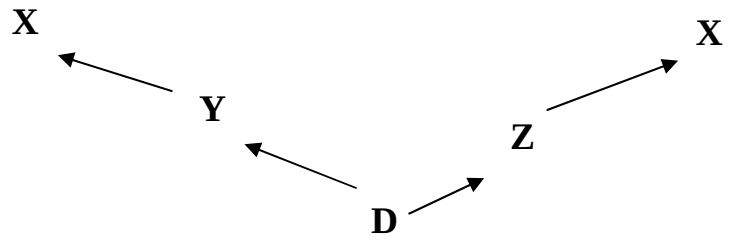


Рисунок 4.4

Проілюстроване дублювання класу відповідає включенню до похідного об'єкта декількох об'єктів базового класу. У такому випадку для звертання до компонентів класу X необхідна «повна кваліфікація», наприклад:

D::Y::X::f() или D::Z::X::f()

Щоб усунути дублювання об'єктів непрямого базового класу при множинному спадкуванні, цей базовий клас необхідно оголосити віртуальним:

```
class X {…; f (); …};
class Y: virtual public X {…};
class Z: virtual public X {…};
class D: public Y, public Z {…};
```

Тепер клас D буде включати тільки один екземпляр X, доступ до якого рівноправно мають класи Y й Z (рис. 4.5).

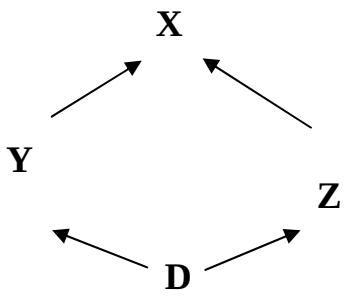


Рисунок 4.5

При множинному спадкуванні той самий базовий клас може бути включений до похідного класу одночасно кілька разів, причому як віртуальний, так і невіртуальний. Можливі будь-які комбінації віртуальних і невіртуальних базових класів.

Віртуальні функції та абстрактні класи

Іноді до базового класу необхідно помістити функцію, що повинна по-різному виконуватися в похідних класах. Точніше, у кожному похідному класі потрібен свій варіант цієї функції. Наприклад, клас «графічний об'єкт» може визначати якісь дії над фігурою без конкретизації її виду,

а нащадки цього класу – «квадрат», «трикутник» тощо – однозначно визначають форму й розміри цієї фігури. Класи, що включають такі функції, називаються ***поліморфними***.

Зазвичай вибір викликуваної функції визначається при написанні вихідного тексту програми й не змінюється після компіляції. Такий режим називається ***раннім*** або ***статичним зв'язуванням***. Механізм віртуальних функцій забезпечує ***пізніше*** або ***динамічне зв'язування***.

```
class gr_obj
{
public:
 gr_obj(int col=7) {_color=col;}
 int color() {  return _color; }
 virtual void draw() {}
 void show() {  setcolor(color());  draw(); }
 void hide()
{ cback=getcolor(); setcolor(getbkcolor()); draw(); setcolor(color()); }
private:
 int _color,cback;
};
```

Особливості віртуальних функцій

1 Віртуальною функцією може бути тільки нестатична функція.

2 Віртуальною не може бути глобальна функція.

3 Функція, що підмінює віртуальну, у похідному класі може бути описана як зі специфікатором virtual, так і без нього. В обох випадках вона буде віртуальною.

4 Віртуальна функція може бути дружньою в іншому класі.

Чистою віртуальною називається компонентна функція, що має таке визначення: virtual тип ім'я (параметри) = 0;

У нашому прикладі: virtual void draw()=0;

Чиста віртуальна функція «нічого не робить» і недоступна для викликів. Її призначення – бути основою для підмінюючих її функцій у похідних класах. Клас, у якому є хоча б одна чиста віртуальна функція, називається ***абстрактним***. Такий клас не може мати екземплярів (об'єктів); він служить винятково для побудови ієрархії класів.

Клас може бути визначений усередині блоку, наприклад, усередині тіла функції. Такий клас називається ***локальним***.

Локалізація класу припускає неприступність його компонентів поза областю визначення класу (поза тілом функції або блоку, у якому він описаний або визначений). Локальний клас не може мати статичних даних. Компонентні функції локальних класів можуть бути тільки убудованими.

4.4 Основи мови програмування Java

Мова Java розроблена компанією Sun Microsystems (творець Джеймс Гослинг працював в цієї компанії з 1984 до 2010 року) і є вільно розповсюджуваною. Транслятори цієї мови існують для більшості наявних операційних систем; програма транслюється не у файл, виконуваний системою (наприклад, .exe), а в спеціальний байт-код, що виконується так званою «віртуальною машиною Java» (JVM).

Якщо в C++ класи створювалися усередині програми (головної функції main), то в Java основною програмною одиницею є саме клас (class), усередині якого, як правило, описується компонентна функція main (якщо це необхідно). Розташування класу визначається середовищем розробки (як правило, класи зберігаються в т. зв. пакетах – packages).

Вихідний код класу зберігається у файлі з розширенням java, клас після перетворення на байт-код JVM – у файлі з розширенням class.

Для завдання ідентифікаторів Java дозволяє використати будь-які символи, наприклад: static final double π=3.14159;

На відміну від C++, поля можуть бути явно ініціалізовані. При відсутності ініціалізаторів полям привласнюються значення за замовчуванням (для чисел – 0, для рядків – символ кінця рядка, для логічних – false, для покажчиків – null).

Перед кожним ідентифікатором в java-програмі може бути скільки завгодно модифікаторів, які наведено через пробіл у будь-якій послідовності, наприклад: public static void main(String[] args) { }

Тут описується головна функція класу, що нічого не повертає (void), не міняється для різних екземплярів даного класу (static) та є загальнодоступною (public). Приклад статичних методів – функції класу Math (звертання йде не до екземпляра, а до самого класу: y=Math.sqrt(x)).

Можливі ще такі модифікатори:

final – описувані дані є константами (static final int max=50) або такий клас не допускає спадкування;

abstract – такий клас не допускає створення екземплярів;

private, package, protected, public – рівень доступу;

native – визначення методу, описаного іншою мовою програмування (як правило, C++).

Спадкування класів задається словом «extends»:

```
class Krug extends Point {}
```

Абстрактний базовий клас в Java має назву інтерфейс, однак він може мати атрибути, навіть з модифікаторами static і final. За замовчуванням усі компоненти інтерфейсу є загальнодоступними (public). Приклад:

```
interface gr_obj {
int x,y;
void draw(); }
```

Тут задаються координати об'єкта та ім'я функції для малювання об'єкта; саме функція буде описана в класах – нащадках.

Об'єкти створюються з використанням команди new:

```
Krug k1 = new Krug();
```

Кожному методу класу неявно передаються покажчики на об'єкт, що його викликав (this), і на предка цього об'єкта (super). При цьому допускається використання цих імен як явні виклики конструкторів: this(); super();

Приведення типів здійснюється вказівкою імені типу в дужках перед ім'ям змінної, наприклад: mref=(More)sref; (тут mref й sref – змінні, More – тип даних).

Перевірка приналежності об'єкта класу відбувається за допомогою оператора instanceof:

```
if (k2 instanceof Krug) Krug k3=(Krug)k2;
```

Усі стандартні класи в Java зберігаються в пакетах. При звертанні до компонентів пакета потрібно або щоразу повністю писати його ім'я:

```
java.util.Date now = new java.util.Date();
```

або попередньо (перед початком опису класу) імпортувати цей пакет:

```
import java.util.Date;
Date now = new Date();
```

На вершині ієрархії класів Java розташований клас Object, що містить такі методи:

boolean equals (Object obj2) – перевіряє збіг змісту поточного об'єкта та об'єкта, заданого як параметр obj2;

Object clone() – клонує поточний об'єкт, тобто повертає його копію;

final Class getClass() – повертає інформацію про клас поточного об'єкта;

void finalize() – деструктор;

String toString() – повертає рядкове подання об'єкта.

Останні два методи можуть (і повинні) бути перевизначеними в класах-нащадках.

Java дозволяє неявне перетворення типів: змінна «вузького» типу може бути автоматично перетворена до більше широкого:

```
int a=5;
System.out.println("k= "+a);
```

Лабораторна робота 1. Класи та об'єкти

Мета роботи. Закріпити знання про об'єктну модель, класи й об'єкти шляхом побудови основної структурної діаграми – діаграми класів.

Завдання. Для будь-якого додатку, розробленого під час попередніх розділів, побудувати діаграму класів.

Можливий вид діаграми класів для простого додатка поданий на рисунку 4.6. Тут на основній формі розташовані два компоненти для роботи з базами даних sdfDataSet1 й dBGrid1, два класи «Кнопка» й «Поле уведення тексту» пов'язані з основною формою відношенням агрегації, сама база даних (компонента разом з текстовим файлом) – відношенням асоціації. Крім того, указується, що основні компоненти є нащадками стандартних класів Object-Pascal – TForm, TButton й TEdit. Користувач пов'язаний з основною формою відношенням асоціації.

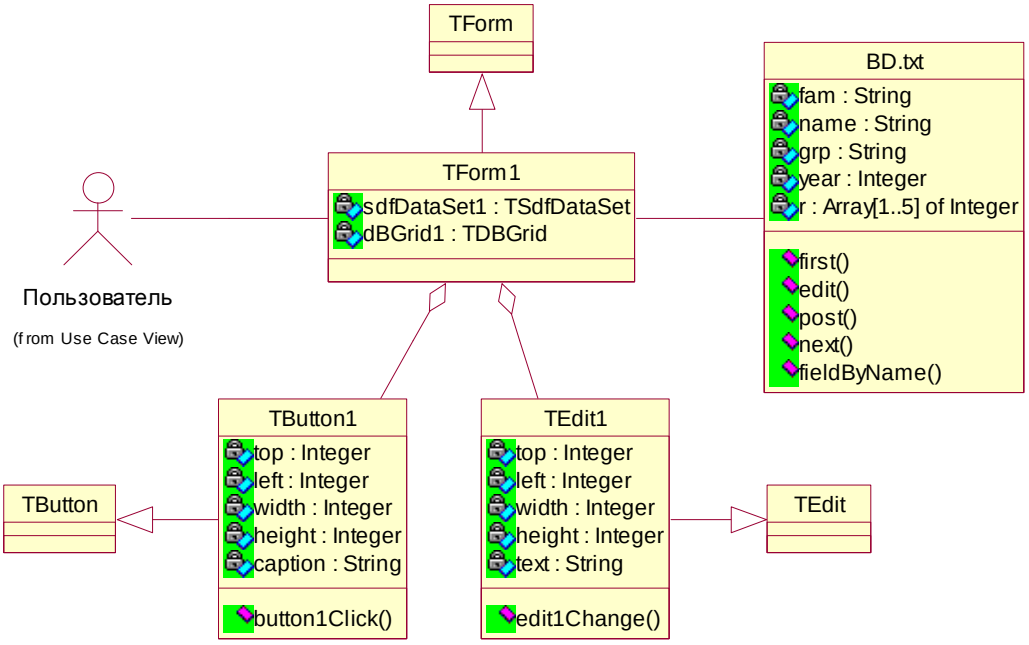


Рисунок 4.6 – Діаграма класів для простого додатка

Цей приклад можна реалізувати іншими способами. З одного боку, компоненти sdfDataSet1 й dBGrid1 також можна зобразити окремими класами, пов'язаними з формою відносинами агрегації. З іншого боку, наведені для кнопки й поля уведення тексту атрибути можна не показувати на діаграмі, тому що вони успадковуються від класів-предків. Конкретизація компонентів (насиченість діаграми) визначається залежно від складності додатка.

Лабораторна робота 2. Програмування мовою Сі

Мета роботи. Відновлення навичок створення програм мовою програмування Сі.

Завдання. Створити програми для вирішення завдань відповідно до індивідуального завдання (табл. 4.1).

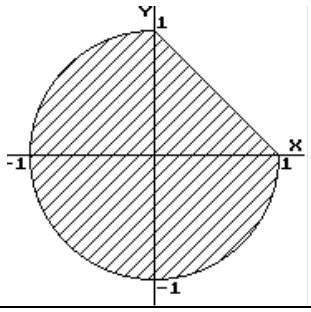
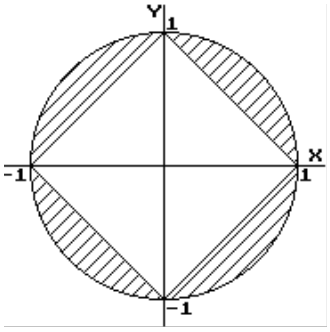
Таблиця 4.1

Вар.	**Завдання**
1	*2*
1	Використовуючи цикл з передумовою, знайти суму $y = \sum \frac{F_1}{F_2}$, де $a \le x \le b$, x змінюється із кроком h = c. F_1: $2\sqrt{x^3}\sin x^3$; F_2: x^4+2x^3-x; a = 0.3; b = 3.12; c = 0.15
2	Використовуючи цикл з післяумовою, знайти суму $y = \sum \frac{F_1}{F_2}$, де $a \le x \le b$, x змінюється із кроком h = c. F_1: $x^3-\ln x$; F_2: x^4-x^{2-x}; a = 1.2; b = 13.4; c = 0.6
3	Використовуючи цикл з параметром, знайти суму $y = \sum \frac{F_1}{F_2}$, де $a \le x \le b$, x змінюється із кроком h = c. F_1: x^3+2x^2-2; F_2: $x^{2x-1}+\cos x$; a = 3.6; b = 7.2; c = 0.2
4	Обчислити таблицю значень функції $y = \begin{cases} F_1(x), \text{якщо } x \le a; \\ F_2(x), \text{якщо } x > a, \end{cases}$ для значень аргументу *x* в інтервалі від xn до xk із кроком hx. F_1: $\arcsin\frac{X}{30}$; F_2: $\sqrt{\lvert\ln x^2\rvert}$; x_n = 2.3; x_k = 8.9; h_x = 0.4; a = 5.4
5	Дано функцію $y = \begin{cases} f1(x), & \text{якщо } x < 0, \\ f2(x), & \text{якщо } 0 \le x \le 1, \\ f3(x), & \text{якщо } x > 1, \end{cases}$ Методом перебору знайти екстремуми даної функції на відрізку. Початкове й кінцеве значення відрізка, а також крок табуляції задавати довільно. F_1: $x^5 ctg\, 2x^3$; F_2: $\ln(x+1)$; F_3: $e^{-2X} - \sqrt[3]{x}$

Продовження таблиці 4.1

1	*2*
6	Дано функцію $y = \begin{cases} f1(x), & якщо \quad x < 0, \\ f2(x), & якщо \quad 0 \le x \le 1, \\ f3(x), & якщо \quad x > 1, \end{cases}$ Методом перебору знайти екстремуми даної функції на відрізку. Початкове й кінцеве значення відрізка, а також крок табуляції задавати довільно. F_1: $ctg(3x-1)^2$; F_2: $2 + xe^{-x}$; F_3: $\sin^3 x^2$
7	Знайти суми парних позитивних елементів кожного рядка матриці A(3,3) і зберегти їх в одномірному масиві B
8	Знайти суми непарних негативних елементів кожного стовпця матриці A(3,3) і зберегти їх в одномірному масиві B
9	Скласти програму з обов'язковим використанням підпрограми для уведення матриці з екрана, її обробки й виводу на екран. Завдання: з кожного елемента матриці A(3,3) відняти суму її непарних позитивних елементів
10	Скласти програму з обов'язковим використанням підпрограми для уведення матриці з екрана, її обробки й виводу на екран. Завдання: кожен елемент матриці A(3,3) поділити на суму її парних позитивних елементів
11	Скласти програму з обов'язковим використанням підпрограми для уведення матриці з екрана, її обробки й виводу на екран. Завдання: кожен елемент матриці A(3,3) поділити на добуток її непарних негативних елементів
12	Скласти програму, що вводить рядок символів, виконує його обробку відповідно до завдання й виводить результати. Завдання: видалити всі символи, що не є цифрами
13	Скласти програму, що вводить рядок символів, виконує його обробку відповідно до завдання й виводить результати. Завдання: замінити всі знаки оклику («!») на символ «*», а символ «крапка» («.») – багатокрапкою (три крапки «…»)
14	Скласти програму, що вводить рядок символів, виконує його обробку відповідно до завдання й виводить результати. Завдання: видалити з рядка всі здвоєні, строєні тощо символи
15	Скласти програму, що вводить рядок символів, виконує його обробку відповідно до завдання й виводить результати. Завдання: вставити пробіл після кожного символу «.» «,» «!» або «?», якщо за цими символами не треба пробіл (тобто треба будь-який символ, крім пробілу)

Продовження таблиці 4.1

1	*2*
16	Сформувати файл із модулів цілих чисел, знайти суму квадратів парних компонентів
17	Прийнявши, що координати точок на площині задаються двома числами **x** й **y**, скласти програму, що вводить із клавіатури координати точок і записує їх послідовно у файл: спочатку **x**, а потім **y**. Після завершення уведення здійснюється перегляд файла і його оброблення: знайти суму відстаней кожної точки від центра координат
18	Сформувати файл із чисел послідовності $(-1)^k*0.2^k/k$. Знайти найбільший з компонентів файла
19	Створити файл, що містить таку структуру даних: «Прізвище здобувача; Найменування групи; Місце проживання; Місце народження; Кількість братів і сестер». Вибрати з файла й видати на екран список студентів, що проживають не там, де народилися (змінили місце проживання)
20	Створити файл, що містить таку структуру даних: «Прізвище здобувача; Найменування групи; Дата народження; Середній рейтинг». Вибрати з файла й видати на екран список студентів заданої групи, що мають середній рейтинг не нижче 4,5
21	Створити файл, що містить таку структуру даних: «Номер книги (код, шифр); Прізвище автора; Найменування книги; Рік видання; Кількість сторінок». Вибрати з файла й видати на екран список книг, виданих раніше заданого року
22	З використанням модуля роботи із графікою скласти програму для малювання зображення: 
23	З використанням модуля роботи із графікою скласти програму для малювання зображення: 

Продовження таблиці 4.1

1	*2*
24	Сформувати динамічний список «Стік» (LIFO) структур (не менш 5), що містить дані про здобувачів у такому вигляді: «Прізвище Ім'я Група РН RS», де РН – рік народження, RS – середній рейтинг. Вивести на екран дані про всіх студентів, чиї імена починаються з літери «Н»
25	Сформувати динамічний список «Черга» (FIFO) структур (не менш 5), що містить дані про здобувачів у такому вигляді: «Прізвище Ім'я Група РН RS», де РН – рік народження, RS – середній рейтинг. Вивести на екран дані про всіх здобувачів старше 18 років

Приклади виконання завдань

1. Знайти екстремуми функції

$$y = \begin{cases} \sqrt{\sin(x)} + 1, \textit{якщо } x \geq 0; \\ x^2 + 2x + 3, \textit{якщо } x < 0 \end{cases}$$

на інтервалі зміни аргументу від $-\pi$ до π.

```
#include <stdio.h>
#include <math.h>
int main()
{
 float Pi=M_PI;
 float x,y,xn=-Pi,xk=Pi,xh=Pi/3,min=1E+10,max=-1E+10,xmin,xmax;
 printf("\n    X        Y\n");
 for (x=xn;x <= xk;x+=xh)
 {
 if (x < 0) {y=pow(x,2)+2*x+3;} else {y=sqrt(sin(x)+1);}
 printf("%8.5f %8.5f\n",x,y);
 if (y > max) {max=y; xmax=x;}
 if (y < min) {min=y; xmin=x;}
 }
printf("Мінімум= %8.5f при x=%8.5f\n",min,xmin);
printf("Максимум= %8.5f при x=%8.5f\n",max,xmax);
}
```

2. Знайти мінімальні елементи кожного стовпця матриці A(3,3) і зберегти їх в одномірному масиві B.

```
#include <stdio.h>
#include <math.h>
int main()
```

```
{
 int min,I,j,b[3];
 int a[3][3]={4,3,1,6,2,7,1,9,3};
 printf("Вихідна матриця:\n");
 for (i=0;i<3;i++) {
 for (j=0;j<3;j++) {printf("%3i",a[i][j]);}
 printf("\n");}
 for (j=0;j<3;j++) { min=a[0][j];
 for (i=1;i<3;i++) {min=a[i][j]<min?a[i][j]:min;}
 b[j]=min;}
 printf("\nРезультат:\n");
 for (j=0;j<3;j++) {printf("%3i",b[j]);} printf("\n");
}
```

3. До кожного елемента матриці M(3,3) додати суму її непарних негативних елементів.

```
#include <stdio.h>
#include <conio.h>
void vvod(int a[3][3])
{
 printf("Уведіть матрицю:\n");
 for (int i=0;i<3;i++) {for (int j=0;j<3;j++) {scanf("%i",&a[i][j]);}}
}
void vyvod(int a[3][3])
{
 for (int i=0;i<3;i++) {
 for (int j=0;j<3;j++) {printf("%4i",a[i][j]);}
 printf("\n");}
}
int summa(int a[3][3])
{
 int sum=0;
 for (int i=0;i<3;i++) {
 for (int j=0;j<3;j++) {
 if ((a[i][j] < 0) && (a[i][j]%2!=0)) {sum+=a[i][j];}
 }}
 return sum;
}
void work(int s,int a[3][3])
{
 for (int i=0;i<3;i++) {
 for (int j=0;j<3;j++) {a[i][j]+=s;}}
}
int main()
{
 int m[3][3];
```

```
 vvod(m);
 printf("\nВихідна матриця:\n"); vyvod(m);
 int s=summa(m);
 printf("\nСума = %3i\n",s); work(s,m);
 printf("\nРезультат:\n"); vyvod(m);
 getche();
}
```

4. Видалити із заданого рядка символів усі цифри.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
 int main()
{
 int i,k;char *st="";
 printf("Уведіть рядок символів:\n"); scanf("%s",st);
 printf("\nВихідний рядок:\n%s\n",st); i=0;
 while (i<strlen(st)) {
 if ((*(st+i)>=’0’)&&(*(st+i)<=’9’)) {
 for (k=I;k<=strlen(st);k++) {
 *(st+k)=*(st+k+1);}}
 else {i++;}
 }
 printf("Результат:\n%s\n",st);
getche();
}
```

5. Сформувати файл із модулів цілих чисел, знайти суму парних компонентів файла. Вивід результатів продублювати у текстовий файл.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
int main()
{
FILE *f,*g;
f=fopen("lab8.dat","w");
 printf("Уводите цілі числа, ознаку кінця – 0:\n");
int k;
do { scanf("%i",&k); if (k!=0) {fwrite(&k,sizeof(k),1,f);}} while (k);
fclose(f);
 printf("Зміст файлу:\n");
if ((f=fopen("lab8.dat","r"))==NULL) {
printf("Помилка при читанні файла!\n");return;}
g=fopen("lab8.txt","w");
```

```
fprintf(g," Зміст файлу:\n");
int s=0;
do { fread(&k,sizeof(k),1,f);
if (!feof(f)) {printf("%4i",k);fprintf(g,"%4i",k);if (!(k%2)) {s+=k;}}
} while (!feof(f));
printf("\nСума = %i\n",s);fprintf(g,"\nСума = %i\n",s);
fclose(f);fclose(g);getche();
}
```

6. Сформувати файл «Список здобувачів» форматом «Прізвище – група – рік_народження – 5_рейтингів», попередньо обчисливши додаткове поле – максимальний рейтинг. Вивести на екран й у текстовий файл список здобувачів заданої групи.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main()
{
 struct stud {char fam[10],gr[10]; int year,rt[5],max;};
 stud ek[3]={{"Іванов","ЕК-03-1",1986,{70,55,55,60,75}},
             {"Петров","ЕК-03-2",1986,{61,62,63,74,55}},
             {"Сидоров","ЕК-03-2",1986,{98,99,97,90,91}}}, s1;
 // шукаємо максимальний рейтинг
 int i,j,max;
 for (i=0;i<3;i++) { max=ek[i].rt[1];
 for (j=0;j<5;j++) { if (ek[i].rt[j] > max) {max=ek[i].rt[j];}}
 ek[i].max=max;}
 FILE *f,*g;
 // запис у файл
 f=fopen("rgr2.dat","w");
 for (i=0;i<3;i++) { fwrite(&ek[i],sizeof(stud),1,f);} fclose(f);
 // читання
 printf("Зміст файлу:\n");
 if ((f=fopen("rgr2.dat","r"))==NULL) {
 printf("Помилка при читанні файлу!\n");return;}
 g=fopen("rgr2.txt","w"); fprintf(g, "Уміст файлу:\n");
 do { fread(&s1,sizeof(stud),1,f);
 if (!feof(f)) {
 printf("%12s%12s%6i  ",s1.fam,s1.gr,s1.year);
 fprintf(g,"%12s%12s%6i  ",s1.fam,s1.gr,s1.year);
 for (j=0;j<5;j++) {printf("%3i",s1.rt[j]);fprintf(g,"%3i",s1.rt[j]);}
 printf("%6i\n",s1.max);fprintf(g,"%3i\n",s1.max);
 }} while (!feof(f));
 // читання з пошуком
 char group[10];
 printf("Уведіть групу для пошуку:\n"); scanf("%s",&group);
```

```
 printf("Здобувачі групи %s:\n",group);
 fprintf(g," Здобувачі групи %s:\n",group); fseek(f,0L,SEEK_SET);
 do { fread(&s1,sizeof(stud),1,f);
 if ((!feof(f))&&(!strcmp(s1.gr,group))) {
 printf("%12s%12s%6i  ",s1.fam,s1.gr,s1.year);
 fprintf(g,"%12s%12s%6i  ",s1.fam,s1.gr,s1.year);
 for (j=0;j<5;j++) {printf("%3i",s1.rt[j]);fprintf(g,"%3i",s1.rt[j]);}
 printf("%6i\n",s1.max);fprintf(g,"%3i\n",s1.max);
 }} while (!feof(f));
 //printf("\nСума = %i\n",s);fprintf(g,"\nСума = %i\n",s);
 fclose(f);fclose(g);getche();
 }
```

7. Намалювати зображення

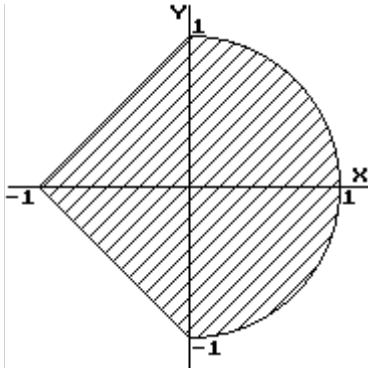


Рисунок 4.7

```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
void axes(int xc, int yc)
{
line(xc-100,yc,xc+100,yc); line(xc+100,yc,xc+95,yc-2);
line(xc+100,yc,xc+95,yc+2); line(xc,yc-100,xc,yc+100);
line(xc,yc-100,xc-2,yc-95); line(xc,yc-100,xc+2,yc-95);
}
 int main()
{
int xc,yc,*gd,*gm;
*gd=0;
initgraph(gd,gm,"");
if (graphresult() !=0 ) {printf("Error!");}
xc=200;yc=200;axes(xc,yc); setfillstyle(3,getmaxcolor());
pieslice(xc,yc,0,90,50); pieslice(xc,yc,270,360,50);
line(xc,yc-50,xc-50,yc);floodfill(xc-2,yc-2,getmaxcolor());
line(xc-50,yc,xc,yc+50);floodfill(xc-2,yc+2,getmaxcolor());
getch(); closegraph();
}
```

8. Сформувати зв'язний список структур, що містить дані про здобувачів. Вивести на екран список здобувачів, у яких прізвища починаються на літеру А.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
struct stud { char fam[10],name[10],group[10]; int gr,rs;};
struct dstud {
                stud data;
                dstud *pPrior;
                dstud *pNext;
                 };
void sread(stud &s)
{
 printf("Family: \n");scanf("%s",&s.fam);
if (s.fam[0] != ‘*’) {
printf("Name: \n");scanf("%s",&s.name);
printf("Group: \n");scanf("%s",&s.group);
printf("Year: \n");scanf("%i",&s.gr);
printf("Rating: \n");scanf("%i",&s.rs);
}}
void main(void)
{
dstud *pBegin=NULL,*pEnd=NULL,*pList=NULL;
stud s;
int k=0;
// Створення списку
clrscr();
pList=new(dstud);
(*pList).pPrior=NULL;
(*pList).pNext=NULL;
sread(s);(*pList).data=s;
pBegin=pList;
// Додавання даних
while (s.fam[0] != '*') {
sread(s);
if (s.fam[0] != '*') {
pEnd=new(dstud);
(*pEnd).pPrior=pList;
(*pEnd).pNext=NULL;
(*pEnd).data=s;
(*pList).pNext=pEnd;
pList=pEnd;
}}
printf("Увесь список:\n");
pList=pBegin;
```

```
while (pList) {
printf(“F=%s N=%s G=%s Y=%i Rs=%i\n”,(*pList).data.fam,
(*pList).data.name,(*pList).data.group,(*pList).data.gr,(*pList).data.rs);
pList=(*pList).pNext;
}
 printf("Необхідні здобувачі:\n");
pList=pBegin;
while (pList) {
if ((*pList).data.fam[0]== 'A') {
k++;
printf(“F=%s N=%s G=%s Y=%i Rs=%i\n”,(*pList).data.fam,
(*pList).data.name,(*pList).data.group,(*pList).data.gr,(*pList).data.rs);
}
pList=(*pList).pNext;
}
 printf("Усього знайдено %i здобувачів. ",k);
}
```

Лабораторна робота 3. Робота з функціями

Мета роботи. Вивчити таке поняття об’єктно-орієнтованого програмування, як «перевантаження функцій», навчитися використати функції з невизначеною кількістю параметрів.

Завдання. Створити підпрограму-функцію, що дозволяє виконувати дії відповідно до індивідуального завдання (табл. 4.2). У всіх випадках, де необхідно, використати перевантаження функцій.

Таблиця 4.2

№	Завдання
1	*2*
1	Знаходження суми трьох цілих чисел або двох речовинних
2	Знаходження суми трьох цілих чисел або різниці двох речовинних
3	Знаходження суми чотирьох цілих чисел або добутку трьох речовинних
4	Знаходження добутку трьох цілих чисел або двох речовинних
5	Знаходження добутку трьох цілих чисел або суми двох речовинних
6	Знаходження суми трьох цілих чисел або різниці двох
7	Знаходження суми чотирьох цілих чисел або добутку трьох
8	Знаходження добутку трьох цілих чисел або суми двох
9	Знаходження середнього арифметичного трьох цілих чисел або суми двох речовинних
10	Знаходження середнього арифметичного трьох цілих чисел або суми двох речовинних

Продовження таблиці 4.2

1	*2*
11	Знаходження середнього арифметичного двох цілих чисел або добутку трьох речовинних
12	Знаходження середнього арифметичного двох цілих чисел або добутку трьох речовинних
13	Знаходження середнього геометричного трьох цілих чисел або суми двох речовинних
14	Знаходження середнього геометричного трьох цілих чисел або добутку двох речовинних
15	Знаходження середнього геометричного трьох цілих чисел або суми двох речовинних
16	Знаходження середнього геометричного трьох цілих чисел або добутку двох речовинних
17	Знаходження суми довільного набору цілих чисел
18	Знаходження суми довільного набору речовинних чисел
19	Знаходження добутку довільного набору цілих чисел
20	Знаходження добутку довільного набору речовинних чисел
21	Знаходження середнього арифметичного довільного набору цілих чисел
22	Знаходження середнього арифметичного довільного набору речовинних чисел
23	Знаходження середнього геометричного довільного набору цілих чисел
24	Знаходження середнього геометричного довільного набору речовинних чисел
25	Знаходження суми довільного набору цілих чисел

Приклади виконання завдання

1. Створити функцію для знаходження мінімального із трьох цілих чисел або максимального із двох речовинних (з використанням «перевантаження»).

```
#include <stdio.h>
int minimax(int a, int b, int c)
{
 int m;
 m=a<b?a:b;
 m=m<c?m:c;
 return m;
}
float minimax(float a, float b)
{
 float m;
```

```
 m=a>b?a:b;
 return m;
}
 int main()
{
 int k;
 k=minimax(6,1,9);
 printf("Мінімальне ціле = %i\n",k);
 float x;
 x=minimax(6.27,5.98);
 printf("Максимальне речовинне = %5.2f\n",x);
}
```

2. Створити функцію для знаходження добутку довільного набору цілих чисел.

```
#include <stdio.h>
int proizv1(int a,...)
// 0 – ознака закінчення списку параметрів
{
 int pr=1,*p=&a;
 while (*p) {pr*=*(p++);}
 return pr;
}
int proizv2(int a,...)
// Перший параметр – кількість параметрів
{
 int pr=1,*p=&a;
 while (a--) {pr*=*(++p);}
 return pr;
}
int main()
{
 int p1=proizv1(2,3,4,0),p2=proizv2(3,2,3,4);
 printf("P1= %i\nP2= %i\n",p1,p2);
}
```

Лабораторна робота 4. Пересування графічних об'єктів на C++

Мета роботи. Вивчити основні визначення об’єктно-орієнтованого програмування на прикладі програми для переміщення графічних об'єктів

Завдання. Скласти програму для вирішення завдання (табл. 4.3). Програма має містити нові класи (включаючи абстрактні), конструктори й деструктори, підтримувати спадкування, поліморфізм і видимість компонентів.

Таблиця 4.3

Вар.	**Завдання**
1	*2*
1	Намалювати на екрані десять концентричних окружностей
2	Намалювати в центрі екрана десять вкладених один в одного прямокутників
3	Переробити програму «кіл на воді» (див. приклад) так, щоб кола спочатку розходилися з однієї точки, а потім назад до неї сходилися
4	Переміщати коло за горизонталлю із заданим кроком і затримкою в одну секунду
5	Переміщати коло за вертикаллю із заданим кроком і затримкою в одну секунду
6	Переміщати прямокутник за горизонталлю із заданим кроком і затримкою в одну секунду
7	Переміщати прямокутник за вертикаллю із заданим кроком і затримкою в одну секунду
8	Переробити програму «кіл на воді» (див. приклад), замінивши кола на прямокутники
9	Переміщати коло за діагоналлю із заданим кроком і затримкою в півтори секунди
10	Переміщати прямокутник за діагоналлю із заданим кроком і затримкою в півтори секунди
11	Розмістити на екрані дві розташовані поруч окружності, що «переморгуються»
12	Переміщати коло, уписане у прямокутник, за горизонталлю із заданим кроком і затримкою в одну секунду
13	Переміщати коло, уписане у прямокутник, за вертикаллю із заданим кроком і затримкою в одну секунду
14	Переміщати коло, уписане у прямокутник, за діагоналлю із заданим кроком і затримкою в півтори секунди
15	Переміщати коло за периметром розташованого в центрі екрана прямокутника
16	Переміщати коло по вершинах розташованого в центрі екрана прямокутника
17	Намалювати трикутник довільної форми й переміщати коло за його периметром
18	Намалювати трикутник довільної форми й переміщати коло по його вершинах
19	Переміщати трикутник за горизонталлю із заданим кроком і затримкою в одну секунду
20	Переміщати трикутник за вертикаллю із заданим кроком і затримкою в одну секунду
21	Переміщати трикутник за діагоналлю із заданим кроком і затримкою в півтори секунди

Продовження таблиці 4.3

1	*2*
22	Переміщати коло екраном випадковим чином
23	Переміщати прямокутник екраном випадковим чином
24	Переміщати трикутник екраном випадковим чином
25	Зробити так, щоб на екрані по черзі виникали й зникали коло, прямокутник і трикутник

Приклад виконання завдання

Зобразити на екрані розбіжні концентричні окружності («кола на воді»).

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>
class gr_init
{
public:
 gr_init(int driver=0)
 {
  *gd=driver;
  initgraph(gd,gm,"");
  if (graphresult() !=0 ) {printf("Error!");abort();}
 }
 ~gr_init()
 {
  closegraph();
 }
private:
 int *gd,*gm;
};
class gr_obj
{
public:
 gr_obj(int col=7)
 {_color=col;}
 virtual void draw()=0;
 int color()
 {
  return _color;
 }
public:
 void show()
 {
```

```
  setcolor(color()); draw();
 }
 void hide()
 {
  cback=getcolor(); setcolor(getbkcolor()); draw(); setcolor(color());
 }
private:
 int _color,cback;
};
class point: public gr_obj
{
public:
 point(int xp=0, int yp=0, int col=7): gr_obj(col)
 {setpx(xp); setpy(yp); show();}
 ~point()
 {
  hide();
 }
 void draw()
 {
  putpixel(x,y,color());
 }
 int getpx()
 {
  return x;
 }
 void setpx(int px)
 {
  x=px;
 }
 int getpy()
 {
  return y;
 }
 void setpy(int py)
 {
  y=py;
 }
private:
 int x,y;
};
class krug: public point
{
public:
 krug(int xc=0, int yc=0, int rad=0, int col=7): point(xc,yc,col)
 {r=rad; show();}
 ~krug()
```

```
{
        hide();
       }
       void draw()
       {
        circle(getpx(),getpy(),getrad());
       }
       void move(int newx, int newy, int newr)
       {
        hide();
        setpx(newx); setpy(newy); setrad(newr);
        show();
       }
       int getrad()
       {
        return r;
       }
       void setrad(int rad)
       {
        r=rad;
       }
      private:
       int r;
      };
      void main()
      {
      gr_init gr(0);
      krug kr(300,200,100);
      for (int i=1; i<=10; i++) {
        kr.move(300,200,10+i*15); delay(200); }
      getch();
      }
```

Лабораторна робота 5. Побудова діаграми класів

Мета роботи. Навчитися об'єктно-орієнтованому аналізуванню та закріпити навички побудови діаграми класів.

Завдання. Об'єктно-орієнтовано проаналізувати завдання з лабораторної роботи 4 і побудувати для неї діаграму класів.

Приклад виконання завдання

1 Об'єктно-орієнтоване аналізування завдання

У розглянутій області можна зазначити такі класи:

1. **Графічний режим**. Властивості: графічний драйвер (gd), графічний режим (gm). Методи: відкриття (конструктор), закриття (деструктор).

2. **Графічний об'єкт**. Властивості: кольори ліній, кольори фону. Методи: визначення кольорів ліній, намалювати об'єкт, показати об'єкт, сховати об'єкт. «Намалювати об'єкт» – чисто віртуальна функція, точний зміст якої буде визначено в спадкоємцях класу. «Показати об'єкт» установлює кольори ліній і викликає «Намалювати об'єкт»; «Сховати об'єкт» установлює кольори ліній ідентичним кольорам тла й знову викликає «Намалювати об'єкт». Оскільки даний клас є основою для створення ієрархії графічних класів, його можна вважати абстрактним (зображується курсивом).

3. **Точка** (спадкоємець «Графічного об'єкта»). Нові властивості: координати. Нові методи: повернення й установка значень координат, намалювати об'єкт.

4. **Коло** (спадкоємець «Точки»). Нова властивість: радіус. Нові методи: повернення й установка значень радіуса, намалювати об'єкт.

2 Побудова діаграми класів

Діаграма класів подана на рисунку 4.8.

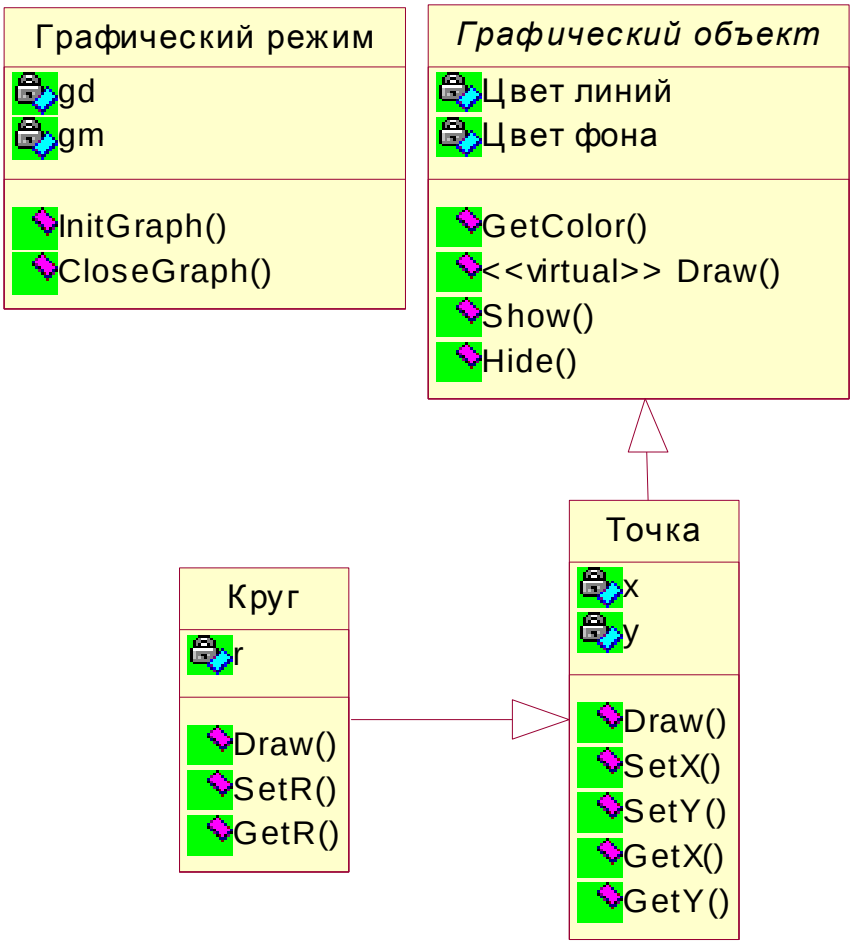


Рисунок 4.8 – Диаграма класів

Лабораторна робота 6. Програмування мовою Java

Мета роботи. Отримання навичок складання програм різної складності мовою програмування Java.

Завдання:

1. Вивчити середовище програмування Eclipse.
2. Створити клас, що дозволяє вирішити завдання відповідно до індивідуального завдання (табл. 4.1). При створенні не забути вказати правильні імена.
3. Виконати створений пакет.

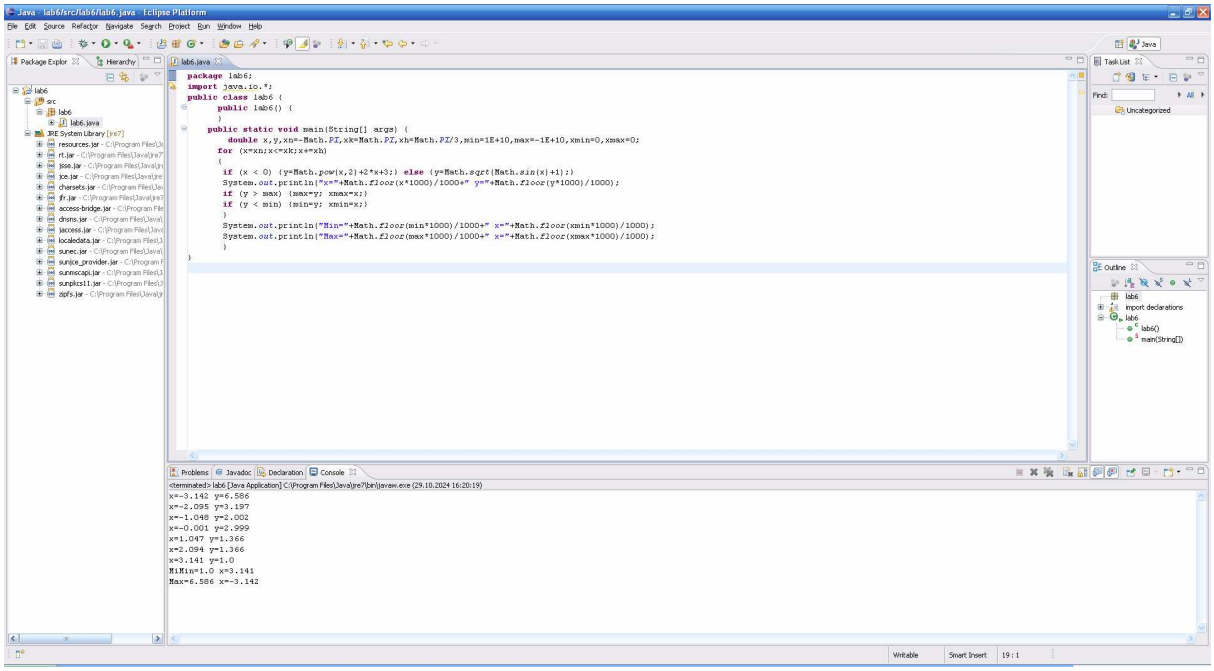
Приклад виконання завдання

Знайти екстремуми функції

$$y = \begin{cases} \sqrt{\sin(x)} + 1, \textit{якщо } x \geq 0; \\ x^2 + 2x + 3, \textit{якщо } x < 0 \end{cases}$$

на інтервалі зміни аргументу від $-\pi$ до π.

```
package lab6;
import java.io.*;
public class lab6 {
       public lab6() {
       }
     public static void main(String[] args) {
         double x,y,xn=-Math.PI,
xk=Math.PI,xh=Math.PI/3,min=1E+10,max=-1E+10,xmin=0,xmax=0;
       for (x=xn;x<=xk;x+=xh)
       {
        if (x < 0) {y=Math.pow(x,2)+2*x+3;} else
{y=Math.sqrt(Math.sin(x)+1);}
        System.out.println("x="+Math.floor(x*1000)/1000+"
y="+Math.floor(y*1000)/1000);
        if (y > max) {max=y; xmax=x;}
        if (y < min) {min=y; xmin=x;}
        }
        System.out.println("Min="+Math.floor(min*1000)/1000+"
x="+Math.floor(xmin*1000)/1000);
        System.out.println("Max="+Math.floor(max*1000)/1000+"
x="+Math.floor(xmax*1000)/1000);
        }
}
```




```
x=-3.142 y=6.586
x=-2.095 y=3.197
x=-1.048 y=2.002
x=-0.001 y=2.999
x=1.047 y=1.366
x=2.094 y=1.366
x=3.141 y=1.0
Min=1.0 x=3.141
Max=6.586 x=-3.142
```

ЛІТЕРАТУРА

1 Мельников О. Ю. Програмування та алгоритмічні мови : ***посібник [для студентів спеціальностей «Системний аналіз» та «Інформаційні системи та технології»]*** / О. Ю. Мельников. – Краматорськ : ДДМА, 2018. – 236 с. – ISBN 978-966-379-870-7.

2 Мельников О. Ю. Об'єктно-орієнтований аналіз і проєктування інформаційних систем : ***посібник [для студентів спеціальностей «Системний аналіз» та «Інформаційні системи та технології»]*** / О. Ю. Мельников. – Вид. 3-є, перероб. та доп. – Краматорськ : ДДМА, 2020. – 208 с. – ISBN 978-966-379-954-4.

3 Мельников О. Ю. Алгоритми і структури даних : ***навчальний посібник [для здобувачів вищої освіти за спеціальностями «Системний аналіз» та «Інформаційні системи та технології»]*** / О. Ю. Мельников. – Краматорськ : ДДМА, 2024. – 120 с. – ISBN 978-617-7889-78-5.

4 Ковалюк Т. В. Основи програмування / Т. В. Ковалюк. – Київ : Видавнича група BHV, 2005. – 384 с.

5 Шищук В. В**.** Основи програмування на алгоритмічній мові Pascal / В. В. Шищук. – Київ : Кондор, 2006. – 224 с.

6 Алгоритмічна мова Паскаль : ***навчал. посібник*** / [уклад. Д. Д. Татарчук]. – Київ : ІВЦ Політехніка, 2006. – 85 с.

7 Вакалюк Т. А. Програмування мовою Pascal : ***навчально-методичний посібник [для студентів фізико-математичного факультету]*** / Т. А. Вакалюк. – Житомир : ФОП Левковець Н. М., 2016. – 232 с.

8 Васильєв О. М. Програмування на С++ в прикладах і задачах / О. М. Васильєв. – Київ : Ліра-К,, 2019. – 382 с.

9 Підручник з мови програмування Pascal. – Режим доступу : https://uk.wikibooks.org/wiki/Pascal

10 Григорович В. Г. Алгоритмізація та програмування / В. Г. Григорович. – Львів : Магнолія, 2023. – 357 с.

11 Stroustrup B. Programming: Principles and Practice Using C++ [Електронний ресурс] / Stroustrup Bjarne. – 2014. – Режим доступу : https://www.pdfdrive.com/programming-principles-and-practice-using-c-e18903967.html.

12 Бублик В. В. Об'єктно-орієнтоване програмування : ***підручник*** / В. В. Бублик. – Київ : ІТ-книга, 2015. – 624 с. – Режим доступу : http://itknyga.com.ua/docs/OOP_final.pdf

13 Алхімова С. М. Обєктно-орієнтоване програмування : ***підручник***: **у 2 ч.** / С. М. Алхімова. – Київ : КПІ ім. Ігоря Сікорського, Вид-во «Політехніка», 2019.

Ч. 2 : Обєктно-орієнтований підхід до розробки програмного забезпечення. – 2019. – 192 с.

14 Петрик О. Об’єктно-орієнтоване програмування в середовищі C++. Лабораторний практикум : ***навчал. посібник*** / О. Петрик. – Тернопіль, ТНТУ імені Івана Пулюя, 2011. – 188 с.

15 Кащеєв Л. Б. Інформатика. Основи візуального програмування : ***навчал. посібник*** / Л. Б. Кащеєв, С. В. Коваленко, С. М. Коваленко. – Харків : Веста, 2011. – 192 с.

16 Безменов М. І. Основи програмування в середовищі Delphi : ***навчал. посібник*** / М. І. Безменов. – Харків : НТУ «ХПІ», 2010. – 608 с.

17 Мельников О. Ю. Робота у середовищі Borland-Delphi : ***навчальний посібник*** / О. Ю. Мельников, П. І. Сагайда. – Краматорськ : ДДМА, 2005. – 100 с. – ISBN 966-379-021-0.

18 Booch Grady. Object-Oriented Analysis and Design with Applications, 3rd Edition / Grady Booch, Robert Maksimchuk, Michael Engle, Jim Conallen, Kelli Houston, Bobbi Young Ph.D. – Addison-Wesley Professional Publ., 2007. – 694 p.

19 Кравченко В. І. Основи програмування та алгоритмічні мови : ***методичний посібник до лабораторних і самостійних робіт [для студентів спеціальності 7.080402]*** / В. І. Кравченко, О. В. Веремій, В. В. Зоненко. – Краматорськ : ДДМА, 2005. – Ч. 1. – 104 с.

20 Васильєв О. М. Програмування мовою Java / О. М. Васильєв. – Тернопіль : Навчальна книга – Богдан, 2020. – 696 с.

Навчальне видання

МЕЛЬНИКОВ Олександр Юрійович

ПРОГРАМУВАННЯ ТА АЛГОРИТМІЧНІ МОВИ

Навчальний посібник

для здобувачів вищої освіти за спеціальностями «Системний аналіз та наука про дані» та «Інформаційні системи і технології»

Редагування і комп’ютерне верстання І. І. Дьякова

57/2024. . Формат 60 х 84/16. Ум. друк. арк.14.00
Обл.-вид. арк. 8.32. Тираж 100 прим. Зам. № 8.

Видавець і виготівник
Донбаська державна машинобудівна академія
84313, м. Краматорськ, вул. Академічна, 72.
Свідоцтво суб’єкта видавничої справи
ДК № 1633 від 24.12.2003