МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря Сікорського»

ОСНОВИ АЛГОРИТМІЗАЦІЇ ТА ПРОГРАМУВАННЯ
Частина 1. Основи алгоритмізації

Комп'ютерний практикум

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра
за освітньою програмою «Системи керування літальними апаратами та комплексами»
спеціальності 173 «Авіоніка»

Укладач М.М.Чепілко

Електронне мережне навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2022

Рецензент: *Смірнов С.А., канд.фіз.-мат.наук, доцент, навчально - науковий фізико - технічний інститут*

Відповідальний редактор: *Пономаренко С.О., кандидат технічних наук, доцент*

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(*Протокол № 6 від 24.06.2022 р.*)
за поданням Вченої ради навчально - наукового інституту аерокосмічних технологій
(*Протокол 5/22 від 31.05.2022 р.*)

Видання навчального посібника "Основи алгоритмізації та програмування. Частина 1. Основи алгоритмізації" для комп'ютерного практикуму призначене для закріплення та поглибленого розуміння студендами теоретичних положень та методів структурного програмування на мові C++, набуття досвіду побудови практично значимих програмних додатків, моделюванні та аналізі актуальних фізико - технічних процесів шляхом виконання спеціально сформульованих комплексних завдань для самостійного виконання. Виконання завдань комп'ютерного практикуму вимагає активного використання студентом набутих знань з вищої математика, фізики та основ алгоритмізації і програмування.

Ритм виконання завдань комп'ютерного практикуму синхронізовано з освоєнням студентами теоретичного матеріалу та набуттям практичних умінь стосовно структурного програмування, що розглядається під час лекцій.

У більшості завдань комп'ютерного практикуму розглянуто певний фізико - технічний сценарій, описана його математична модель, наведено приклад екранної форми та програмного коду комп'ютерного додатку, де робиться акцент на тому чи іншому методі структурного програмування на мові C++. Надаються методичні рекомендації, що до виконання завдання. Оцінювання виконаних студентом завдань комп'ютерного практикуму передбачає аналіз функціоналу відкомпільованого виконавчого файлу та якості написання програмного коду.

Реєстр. №21/22-602. Обсяг 4.8 авт.арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Перемоги, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК №5354 від 25.05.2017 р.

© КПІ ім. Ігоря Сікорського, 2022

Зміст

Вступ 6

1 Комп'ютерний практикум №1. Розробка базових алгоритмів та їх програмна реалізація 7

1.1 Теоретичні відомості та рекомендації 7

1.2 Завдання для аудиторної роботи 21

1.3 Завдання для самостійної роботи 21

1.4 Варіанти завдань для аудиторної та домашньої роботи студентів 22

1.5 Питання для самоконтролю знань 26

2 Комп'ютерний практикум №2. Алгоритмізація та програмування арифметичного калькулятора 26

2.1 Теоретичні відомості та рекомендації 26

2.2 Завдання для аудиторної роботи 30

2.3 Завдання для самостійної роботи 30

2.4 Питання для самоконтролю знань 30

3 Комп'ютерний практикум №3. Алгоритмізація та програмування калькулятора для розв'язку алгебраїчних рівнянь 1,2 та 3 порядку 31

3.1 Теоретичні відомості та рекомендації 31

3.2 Завдання для аудиторної роботи 40

3.3 Завдання для самостійної роботи 40

3.4 Питання для самоконтролю знань 41

4 Комп'ютерний практикум №4. Алгоритмізація та програмування додатку для дослідження динаміки двовимірного маятника 41

4.1 Теоретичні відомості та рекомендації 42

4.2 Завдання для аудиторної роботи 45

4.3 Завдання для самостійної роботи 45

4.4 Варіанти завдань для аудиторної та домашньої роботи студентів 46
4.5 Питання для самоконтролю знань 48
5 Комп'ютерний практикум №5. Алгоритмізація та програмування опрацювання статичних одновимірних та двовимірних масивів 49
5.1 Теоретичні відомості та рекомендації 49
5.2 Завдання для аудиторної роботи 56
5.3 Завдання для самостійної роботи 57
5.4 Варіанти завдань для аудиторної та домашньої роботи студентів 58
5.5 Питання для самоконтролю знань 61
6 Комп'ютерний практикум №6. Алгоритмізація та програмування опрацювання числових одновимірних та двовимірних динамічних масивів з використанням показчиків 62
6.1 Теоретичні відомості та рекомендації 62
6.2 Завдання для аудиторної роботи 74
6.3 Завдання для самостійної роботи 74
6.4 Варіанти завдань для аудиторної та домашньої роботи студентів 74
6.5 Питання для самоконтролю знань 78
7 Комп'ютерний практикум №7. Функції та правила їх алгоритмізації і програмування 78
7.1 Теоретичні відомості та рекомендації 78
7.2 Завдання для аудиторної роботи 91
7.3 Завдання для самостійної роботи 91
7.4 Варіанти завдань для аудиторної та домашньої роботи студентів 92
7.5 Питання для самоконтролю знань 96
8 Комп'ютерний практикум №8. Алгоритмізація та програмування додатку для аналізу траєкторій польоту ракети в межах земної атмосфери без врахування опору повітря 96
8.1 Теоретичні відомості стосовно предметного середовища . 97
8.2 Завдання для аудиторної роботи 104
8.3 Завдання для самостійної роботи 104

8.4 Питання для самоконтролю знань 105
Рекомедована література 105

Вступ

Теоретичні питання застосування мови програмування C++ та супутні питання розглядаються під час лекцій з навчальної дисципліни "Основи алгоритмізації та програмування" та рекомендованій літературі [1]-[10].

Що стосується комп'ютерного практикуму то, як показує досвід, на першому етапі освоєння технологій програмування на мові C++ найбільш підходящими є програмні продукти **Embarcadero C++ Builder Community Edition** або **Borland C++ Builder Enterprise v6.0**, в залежності від потужності комп'ютера. Ці продукти є добре відпрацьованими, найбільш гнучкі і швидкі, інтуїтивно зрозумілі і, розом з тим займають менше ресурсів компю'тера ніж Microsoft Visual Studio. З огляду на ці обставини при вивченні мови програмування високого рівня C++ на практичному рівні ми будемо використовувати саме Borland C++ Builder Enterprise v6.0 [5], [6]. Отримавши досвід програмування в середовищі Borland C++ Builder Enterprise v6.0 можна буде без проблем перенести свої проекти до більш сучасного середовища Embarcadero C++ Builder, яке має дещо розширені можливості, але, разом з тим, є більш ресурсоємним.

Середовище C++ Builder поєднує засоби мови програмування C++ та компонентно - орієнтований підхід до створення програм. Поєднання простоти освоєння візуального середовища та підтримки широкого спектру технологій робить C++ Builder універсальним інструментом створення програмних проектів будь якого рівня складності. Навіть початкові програмісти зможуть швидко створювати програмні проекти в C++ Builder, які матимуть професійно виконаний інтерфейс найрізноманітнішої спрямованості — від суто обчислювальних та логічних до графічних та мультимедійних. Програмний код середовища C++ Builder є універсальним і майже не відрізнюється від інших програмних засобів розроблення програм мовою C++.

При побудові комп'ютерного додатку рекомендується дотримуватися такого порядку його реалізації

- Увімкнути Windows-комп'ютер та запустити програмне середовище Embarcadero C++ Builder Community Edition або Borland C++ Builder Enterprise v6.0 в залежності від потужності комп'ютера.
- На меню інструментальної панелі вибрати File->New->Application; File->Save Project As; У діалоговому вікні збереження файлів проекту послідовно перейменувати та зберегти у довільно вибраній (створеній) папці файли Unit1.cpp -> Unit01.cpp, Project1 -> ProjectN, де Unit01.cpp — головний файл проекту, N — номер проекту).
- Відкомпілювати проект натиснувши на клавіатурі одночасно клавіши

Ctrl+F9. В папці з файлами проекту повинен з'явитися виконавчий файл ProjectN.exe.

- Перейменувати головну форму Form1->frmUnit01 і знову відкомпілювати проект.
- Розташувати на головній формі проекту колекцію компонентів, які необхідні для реалізації графічного інтерфейсу, та надати їм відповідне оформлення. Компоненти знаходяться на інструментальній панелі програмного середовища, яке використовується для програмування додатку.
- За допомогою вкладки Event на панелі Object Inspector приєднати до актуальних компонентів відповідні їм функції та заповнити їх блоки кодами, які реалізують функціонал додатку.
- Відкомпілювати проект та перевірити працездатність додатку. Якщо компіляція виявила помилки у написанні коду — перевірити його, виправити помилки і знову виконати процедуру компіляції проекту.
- Виконати тестові розрахунки на прикладах з відомою відповіддю.
- Підготувати звіт про самостійно виконану роботу, який в обов'язковому порядку, повинен ключати блок-схему алгоритму, лістінг програмного коду та відкомпільований виконавчий файл запрограмованого додатку.

1. Комп'ютерний практикум №1. Розробка базових алгоритмів та їх програмна реалізація

Мета практикуму. Отримати досвід розробки базових алгоритмів та програмування простих додатків з викристанням лінійних, розгалужених та циклічних операторів мови програмування C++.

1.1. Теоретичні відомості та рекомендації

Базові алгоритми організації обчислень поділяють на три основні види:

- лінійні (послідовності);
- розгалужені;
- циклічні.

Переважно вони є окремими частинами обчислювального процесу, тоді як загальний обчислювальний процес має складнішу (комбіновану) структуру. Зазвичай при написанні програми ці базові алгоритми поєднуються в такій структурі.

У лінійних алгоритмах усі операції виконуються послідовно у порядку їхнього запису. Типовим прикладом такого алгоритму є стандартна обчислювальна схема, яка складається з трьох етапів:

- введення початкових даних;
- обчислення за формулами;
- виведення результату.

У розгалужених алгоритмах для здобуття кінцевого результату передбачається вибір одного з кількох можливих напрямів обчислень (гілок) залежно від результату перевірки певної умови. Напрям обчислень обирається перевіркою, внаслідок якої можливі два випадки:

- "Так" – умову виконано;
- "Ні" – умову не виконано.

Циклом називається послідовність дій, яка багаторазово повторюється; алгоритм, який містить цикл, має назву циклічного. Цикли застосовують, коли для розв'язання задачі виникає потреба повторення низки кроків. Це призводить до багаторазового виконування одних і тих самих операторів у програмі за різних значень змінних. Керування повторенням циклу здійснюється за допомогою змінної, яка називається параметром циклу. Спочатку цьому параметру присвоюється певне початкове значення. Потім цикл виконується зі зміною параметру за кожного повторення від початкового до кінцевого значень на величину, що називається кроком циклу. Крок циклу може бути додатним чи від'ємним. Залежно від цього параметр циклу зростає чи зменшується. Цикл припиняється, якщо параметр циклу має значення, яке лежить поза межами діапазону між початковим і кінцевим значеннями.

Програмування лінійних алгоритмів

Лінійним називається алгоритм, в якому всі дії, від першої до останньої, виконуються послідовно. Іноді таку структуру називають просто послідовністю. Зазвичай у програмній реалізації таких обчислень використовують оператори

введення початкових даних, оператори обчислення та оператори виведення результатів.

Приклад 1.1. Розробити алгоритм і створити програмний додаток в C++ Builder для обчислення значень функції

$$y = \frac{0.2x^2 - x}{(\sqrt{3} + x)(1 + 2x)} + \frac{2(x-1)^3}{\sin^2(x) + 1}, \tag{1.1}$$

де x — довільна змінна, яку слід ввести.

Розв'язок. Алгоритм розв'язування цього прикладу та головна форма додатку зображені на рис. 1.1. Для побудови графічного інтерфейсу додатку на

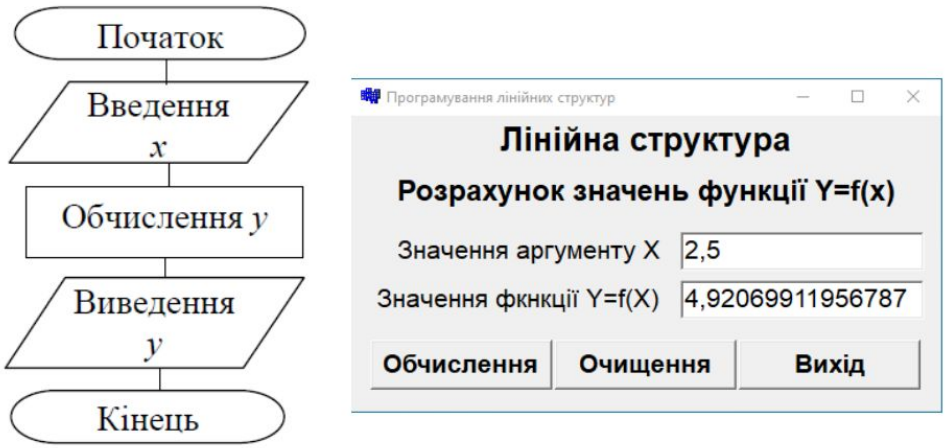


Рис. 1.1 — Блок-схема алгоритму і екранна форма додатку для обчислення значень функції

головній формі проекту (див. рис. 1.1), розташовується колекція компонентів

$$\begin{cases} TForm - 1 \text{ шт.}; \\ TPanel - 2 \text{ шт.}; \quad TLabel - 2 \text{ шт.}; \\ TEdit - 2 \text{ шт.}; \quad TButton - 3 \text{ шт.}, \end{cases} \tag{1.2}$$

які переносяться на форму з інструментальної панелі середовища програмування, та надається їм відповідне оформлення.

Програмний код головного файлу, який реалізує інтерфейс додатку (див. рис. 1.1) та його функціонал має такий вигляд

```
/*- Unit01.cpp ------------------------------------------------*/
#include <vcl.h>
#pragma hdrstop
#include <math.h> // Долучення математичної бібліотеки
#include "Unit01.h"
/*-----------------------------------------------------------------*/
```

```
#pragma package(smart_init)
#pragma resource "*.dfm"
TfrmUnit01 *frmUnit01;
/*---------------------------------------------------------------*/
__fastcall TfrmUnit01::TfrmUnit01(TComponent* Owner)
        : TForm(Owner)
{
/*---------------------------------------------------------------*/
} void __fastcall TfrmUnit01::FormShow(TObject *Sender)
{
Button1Click((TObject *)Sender);
}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button1Click(TObject *Sender)
{
float y, x=StrToFloat(Edit1->Text); // Ввведення x з Edit1
y=(0.2*x*x-x)/((sqrt(3)+x)*(1+2*x))+2*pow(x-1,3)/(pow(sin(x),2)+1);
Edit2->Text=FloatToStr(y); // Виведення y в Edit2
}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button2Click(TObject *Sender)
{
Edit1->Clear(); Edit2->Clear(); // Очищує вікна Edit1 та Edit2
Edit1->SetFocus(); // Встановлює курсор (фокус) в Edit1
}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button3Click(TObject *Sender)
{
frmUnit01->Close();
}
/*---------------------------------------------------------------*/
```

Програмування розгалужених алгоритмів

Алгоритм називається розгалуженим якщо певні його команди (інструкції) можуть не виконуватись жодного разу. Це відрізнює такі алгоритми від лінійних. Вочевидь, деякі команди не виконуються за певних вхідних умов, але можуть виконуватися за інших умов. Якщо за аналогією проаналізувати повсякденні життєві ситуації, то можна віднайти численну кількість умов, які розпочинаються словом "якщо". Отже, розгалуженням називається вибір програмою тієї чи іншої низки команд залежно від того, чи виконується певна

умова. При цьому спрацьовує лише одна з гілок алгоритму.

Для програмної реалізації таких обчислень слід використовувати оператори передавання керування, котрі дозволяють змінювати порядок виконування операторів програми. У мові C++ для цього передбачено інструкції: безумовного переходу — goto, умовного переходу — if та вибору варіанта — switch. Для записування умови переходу слід використовувати логічні (булеві) вирази.

Приклад 1.2. Розробити алгоритм з вкладеною перевіркою умов (порівняння в умовах не повторювати), подати його блок-схемою й написати програму обчислення значення заданої логічною залежністю функції

$$y = \begin{cases} \dfrac{9}{ax} \quad \text{при} \quad x \in [1, 3) \\ |ax^2 + x + b| \quad \text{при} \quad x = 3) \end{cases} \tag{1.3}$$

при довільних значеннях параметрів a і b і незалежної змінної x. В алгоритмі передбачити перевірку, чи немає ділення на нуль. Вважається, що функція визначена на заданому проміжку, а поза ним не визначена. Видачу повідомлень про помилки здійснювати зразу ж при їхньому виявленні, а видачу результату обчислення значення функції $y = f(x)$ — у кінці програми (щоб реалізувати таку видачу повідомлень, треба використати робочу змінну -індикатор, яка вказує на те, чи значення функції було обчислене).

Розв'язок. Алгоритм розв'язування цього прикладу та головна форма додатку зображені на рис. 1.2, рис. 1.3.

Для побудови графічного інтерфейсу додатку на головній формі проекту (див. рис. 1.3), розташовується колекція компонентів

$$\begin{cases} TForm - 1 \text{ шт.}; \quad TPanel - 2 \text{ шт.}; \\ TLabel - 2 \text{ шт.}; \quad TEdit - 2 \text{ шт.}; \\ TMemo - 1 \text{ шт.}; \quad TButton - 1 \text{ шт.}, \end{cases} \tag{1.4}$$

Програмний код головного файлу, який реалізує інтерфейс додатку (див. рис. 1.3) та його функціонал має такий вигляд

```
/*- Unit01.cpp ------------------------------------------------*/
#include <vcl.h>
#pragma hdrstop
#include "Unit01.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
#include <math.h>
```

```
TfrmUnit01 *frmUnit01;

float x, // значення аргумента
y, // значення функції
a,b; // параметри функції
int ind;
/*---------------------------------------------------------------*/
__fastcall TfrmUnit01::TfrmUnit01(TComponent* Owner)
        : TForm(Owner)
{
}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::FormShow(TObject *Sender)
{
Button1Click((TObject *)Sender);
}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button1Click(TObject *Sender)
{
ind = 0;
a = (Edit1->Text=="") ? 0 : StrToFloat(Edit1->Text);
b = (Edit2->Text=="") ? 0 : StrToFloat(Edit2->Text);;
x = (Edit3->Text=="") ? 0 : StrToFloat(Edit3->Text);
if (fabs(a*x) > 0) {
if ((x >= 1) && (x < 3)) {
ind = 1;
y = 9/(a*x);}
else
if (x >= 3) {
ind = 1;
y = fabs(a*x*x+x+b);};
}
else {
ShowMessage("Ділення на 0");
Edit4->Text = "";
};
if (ind == 1) {Edit4->Text = FloatToStr(y);}
else
{Edit4->Text = ""; ShowMessage("Функція не визначена");};
}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button2Click(TObject *Sender)
```

```
{
ind = 0;
Edit1->Text = "";
Edit2->Text = "";
Edit3->Text = "";
Edit4->Text = "";
}
/*-----------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button3Click(TObject *Sender)
{
frmUnit01->Close();
}
/*-----------------------------------------------------------------*/
```

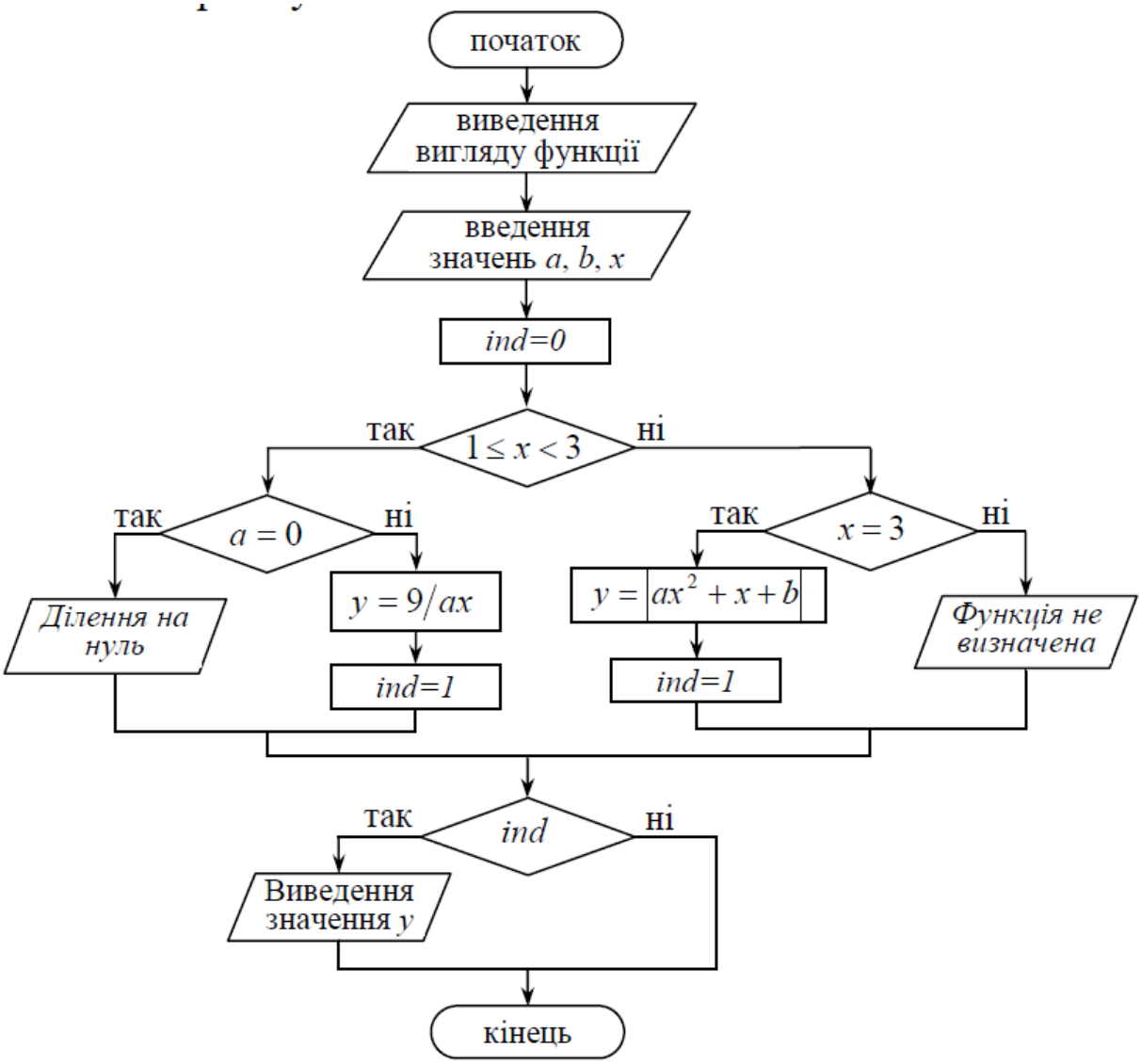


Рис. 1.2 — Блок-схема алгоритму для обчислення значень функції заданої логічною залежністю

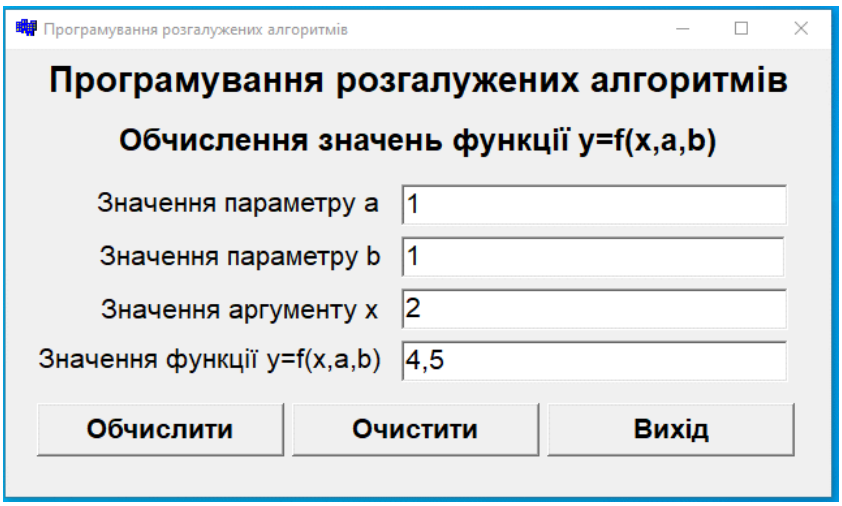


Рис. 1.3 — Екранна форма додатку для обчислення значень функції заданої логічною залежністю

Приклад 1.3. Розробити алгоритм обчислення значення функції

$$L = \begin{cases} x^{2+a} + 1 \quad \text{при} \quad k = 1; \\ \dfrac{x+1}{a} \quad \text{при} \quad k = 2; \\ e^{x-a} - a^x 1 \quad \text{при} \quad k = 3; \\ \log_{10} |x - a| \quad \text{при} \quad k = 4 \end{cases} \tag{1.5}$$

для всіх значень k з використанням оператора switch та подати його блок-схему. На її основі запрограмувати додаток для візуплізації результатів обчислень усіх чотирьох значень функції L у компоненті Memo (багаторядкове текстове вікно для введення чи виведення значень даних тієї чи іншої програми).

Розв'язок. Алгоритм розв'язування цього прикладу та головна форма додатку зображені на рис. 1.4.

Для побудови графічного інтерфейсу додатку на головній формі проекту (див. рис. 1.4), розташовується колекція компонентів

$$\begin{cases} TForm - 1 \text{ шт.}; \quad TPanel - 2 \text{ шт.}; \\ TLabel - 2 \text{ шт.}; \quad TEdit - 2 \text{ шт.}; \\ TMemo - 1 \text{ шт.}; \quad TButton - 3 \text{ шт.}, \end{cases} \tag{1.6}$$

які переносяться на форму з інструментальної панелі середовища програмування, та надається їм відповідне оформлення.

Алгоритм та програмний код головного файлу, який реалізує інтерфейс додатку (див. рис. 1.4) та його функціонал, мають таке наповнення

```
/*- Unit01.cpp ------------------------------------------------*/
#include <vcl.h>
#pragma hdrstop
#include <math.h>
#include "Unit01.h"
/*---------------------------------------------------------------*/
#pragma package(smart_init)
#pragma resource "*.dfm"
TfrmUnit01 *frmUnit01;
/*---------------------------------------------------------------*/
__fastcall TfrmUnit01::TfrmUnit01(TComponent* Owner)
        : TForm(Owner)
{
}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::FormShow(TObject *Sender)
{
Button1Click((TObject *)Sender);
}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button1Click(TObject *Sender)
{
Memo1->Clear();
float x, a, L; int k = 1;
a = (Edit1->Text==""") ? 0 : StrToFloat(Edit1->Text);
x = (Edit2->Text==""") ? 0 : StrToFloat(Edit2->Text);
if (fabs(a*x) > 0) {
Memo1->Lines->Add("Результати обчислення функції L=f(x):");
M1: switch (k)
{
case 1: L = pow(x, 2 + a) + 1; break;
case 2: L = (x + 1) / a; break;
case 3: L = exp(x - a) - pow(a, x); break;
case 4: L = log10(fabs(x - a)); break;
}
Memo1->Lines->Add("L" + IntToStr(k) + " = " + FloatToStr(L));
k++;
if(k<=4) goto M1;
}
}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button2Click(TObject *Sender)
```

```
{
Edit1->Text = "";
Edit2->Text = "";
Memo1->Clear();
}
/*-------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button3Click(TObject *Sender)
{
frmUnit01->Close();
}
/*-------------------------------------------------------------*/
```

Оператор **if(k <= 4) goto M1** у цій програмі організовує чотириразове повторення виконання групи операторів, починаючи від мітки **M1**, що фактично утворює псевдоцикл.

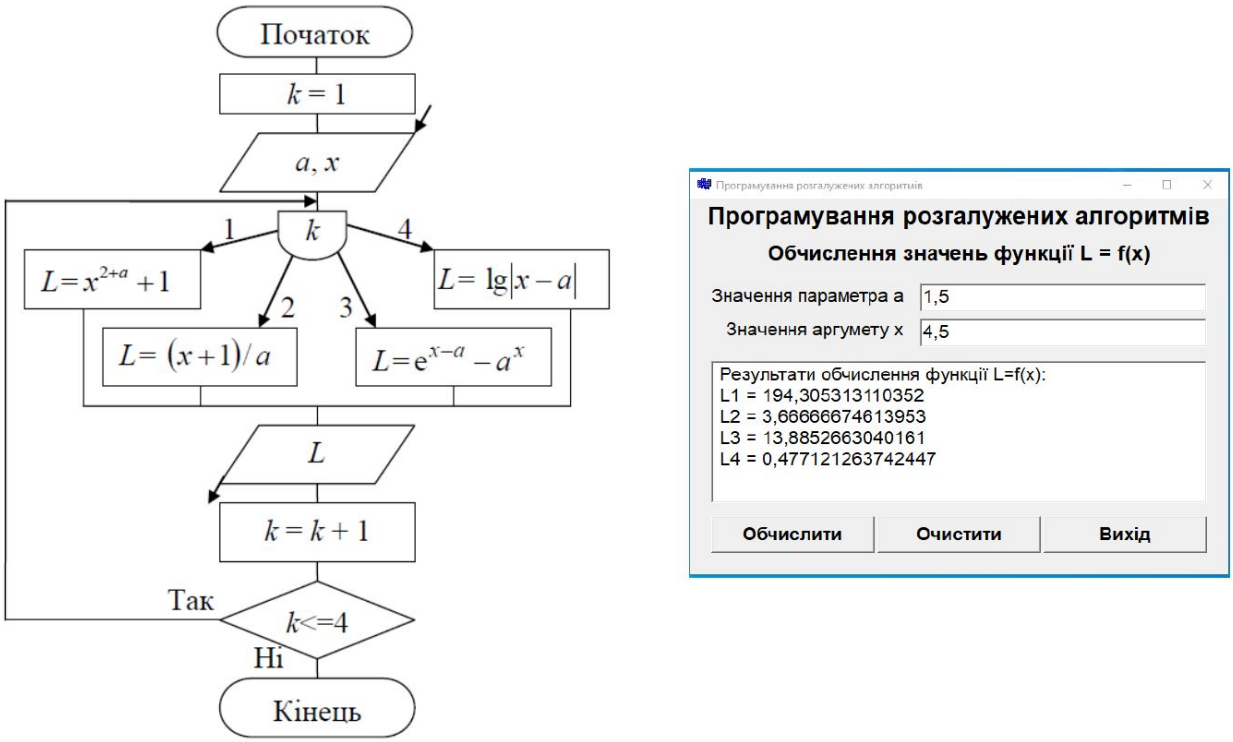


Рис. 1.4 — Блок-схема алгоритму та екранна форма додатку для обчислення значень функції заданої з використанням інструкції вибору варіанту *switch*

Програмування циклічних алгоритмів

Обчислювальний процес називається циклічним, якщо він неодноразово повторюється, допоки виконується певна задана умова. Блок повторюваних операторів називають тілом циклу.

Існують три різновиди операторів циклу:

- оператор циклу for;
- оператор циклу з передумовою while;
- оператор циклу з післяумовою do-while.

Всі оператори циклу неодмінно містять такі складові частини:

- присвоювання початкових значень (ініціалізація);
- умова продовження циклу;
- тіло циклу;
- зміна параметра циклу.

Приклад 1.4. Обчислити суму знакозмінного ряду

$$S = \sum_{i=1}^{5} \frac{(-1)^i x^{i+1}}{i!} \tag{1.7}$$

трьома варіантами, використовуючи оператори циклу **for**, **while**, **do-while**.

Розв'язок. Алгоритм розв'язування цього прикладу та головна форма додатку зображені на рис. 1.5, рис. 1.6.

Для побудови графічного інтерфейсу додатку на головній формі проекту (див. рис. 1.6), розташовується колекція компонентів

$$\begin{cases} TForm - 1 \text{ шт.}; \quad TPanel - 2 \text{ шт.}; \\ TLabel - 1 \text{ шт.}; \quad TEdit - 4 \text{ шт.}; \\ TButton - 5 \text{ шт.}, \end{cases} \tag{1.8}$$

які переносяться на форму з інструментальної панелі середовища програмування, та надається їм відповідне офомлення.

Програмний код головного файлу, який реалізує інтерфейс додатку (див. рис. 1.6) та його функціонал має такий вигляд

```
/*- Unit01.cpp ------------------------------------------------*/
#include <vcl.h>
#pragma hdrstop
#include <math.h>
#include "Unit01.h"
//---------------------------------------------------------------*/
```

```
#pragma package(smart_init)
#pragma resource "*.dfm"
TfrmUnit01 *frmUnit01;
float x;
//---------------------------------------------------------------*/
__fastcall TfrmUnit01::TfrmUnit01(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------*/


void __fastcall TfrmUnit01::FormShow(TObject *Sender)
{
x=StrToFloat(Edit1->Text);
Button1Click((TObject *)Sender);
}
//---------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button1Click(TObject *Sender)
{ int i, f, k; float S=0, x=StrToFloat(Edit1->Text);
for(i=1; i<=5; i++)
{ for(k=1, f=1; k<=i; k++) f *= k;
S += pow(-1,i)*pow(x,i+1)/f;
}
Edit2->Text = FloatToStr(S);
Edit3->Text = "";
Edit4->Text = "";
}
//---------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button2Click(TObject *Sender)
{ int i=1, f, k;
float S=0, x=StrToFloat(Edit1->Text);
while(i<=5)
{ f=1; k=1;
while(k<=i) { f *= k; k++; }
S += pow(-1, i) * pow(x, i+1) / f;
i++;
}
Edit2->Text = "";
Edit3->Text = FloatToStr(S);
Edit4->Text = "";
}
//---------------------------------------------------------------*/
```

```
void __fastcall TfrmUnit01::Button3Click(TObject *Sender)
{ int i=1, f, k;
float S=0, x=StrToFloat(Edit1->Text);
do
{ f=1; k=1;
do
{ f *= k;
k++;
}
while(k <= i);
S += pow(-1, i) * pow(x, i+1) / f;
i++;
}
while(i <= 5);
Edit2->Text = "";
Edit3->Text = "";
Edit4->Text = FloatToStr(S);
}
//---------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button5Click(TObject *Sender)
{
Edit1->Text = "0";
Button1Click((TObject *)Sender);
}
//---------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button6Click(TObject *Sender)
{
frmUnit01->Close();
}
//---------------------------------------------------------------*/
```

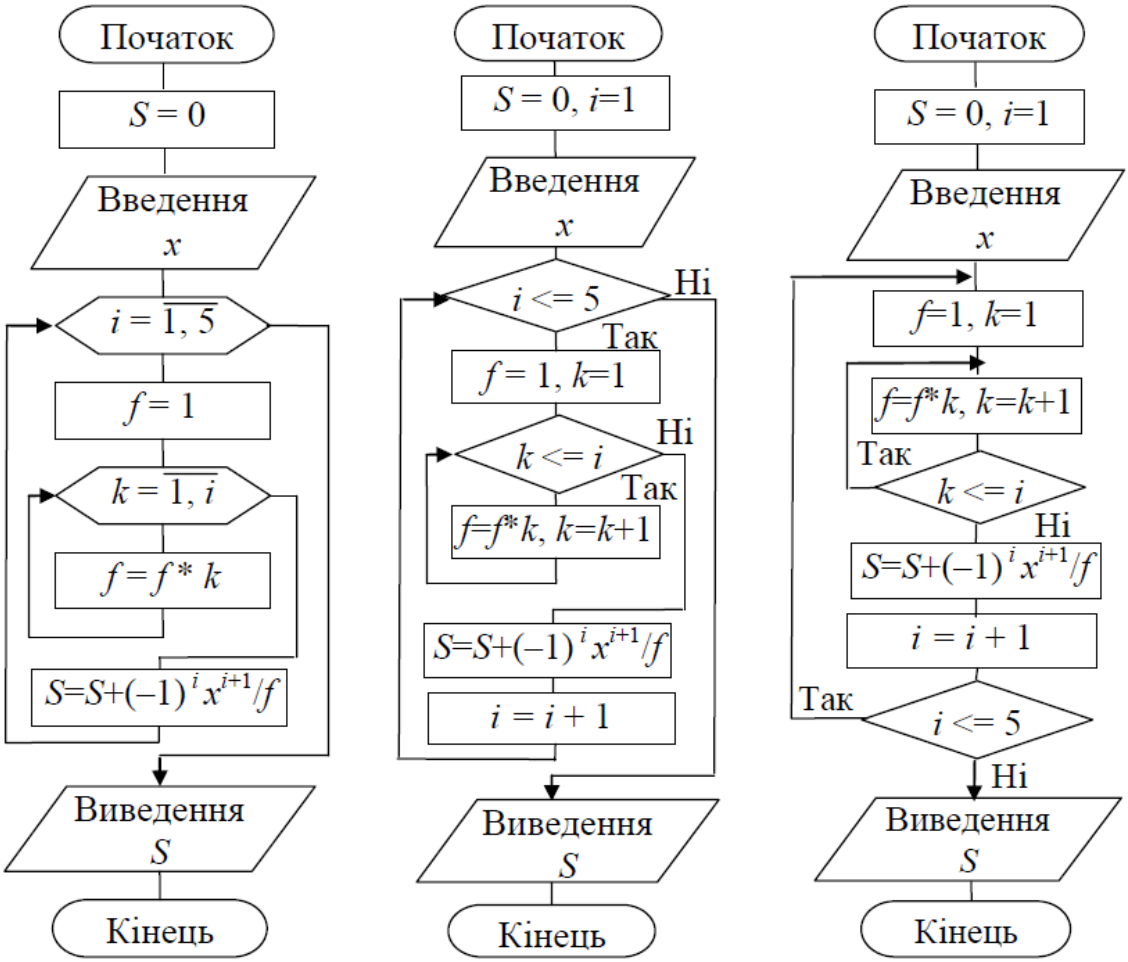


Рис. 1.5 — Блок-схеми алгоритмів використання операторів циклу **for**, **while**, **do-while** для обчислення суми знакозмінного ряду

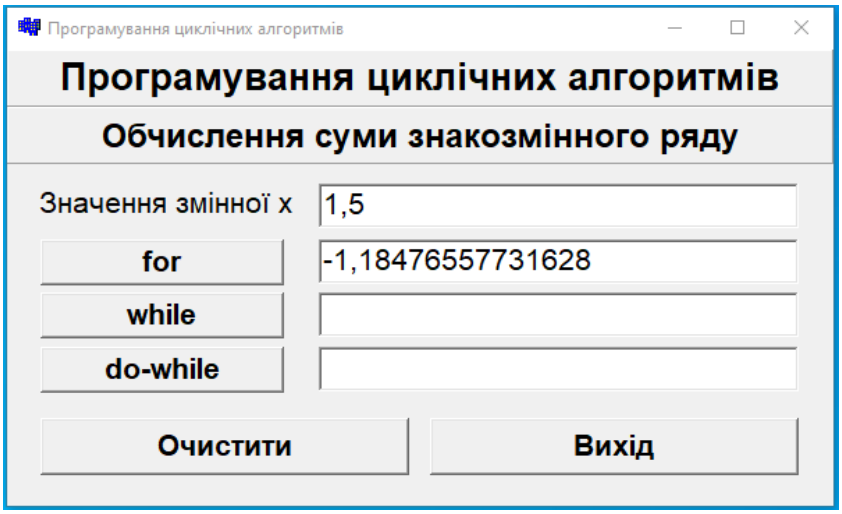


Рис. 1.6 — Екранна форма додатку для обчислення суми знакозмінного ряду з використанням операторів циклу **for**, **while**, **do-while**

1.2. Завдання для аудиторної роботи

- Розробити алгоритм і написати програмний код для обчислення значень функції
$$y = \frac{0.5x^2 - e^x}{(\sqrt{7} + x^4)(1 + 2\cos^2(x))} - \frac{2(x-1)^3}{\sin^2(x) + 1}, \tag{1.9}$$
де x — довільна змінна, яку слід ввести як аргумент функції.

- Розробити алгоритм з вкладеною перевіркою умов і написати програмний код обчислення значення заданої логічною залежністю функції
$$y = \begin{cases} \dfrac{b}{a\sin(x)} & \text{при} \quad x \in [1, 5) \\ |a\cos(x) + x + b| & \text{при} \quad x = 5) \end{cases} \tag{1.10}$$
при довільних значеннях параметрів a і b і незалежної змінної x. В алгоритмі передбачити перевірку, чи немає ділення на нуль.

- Розробити алгоритм і написати програмний код обчислення значення функції
$$y = \begin{cases} a(1 - e^{-x}) & \text{при} \quad k = 1; \\ \dfrac{\mathrm{tg}(x)}{a+2} & \text{при} \quad k = 2; \\ e^{a-x} - a^x & \text{при} \quad k = 3; \\ \ln|x^2 - a^2| & \text{при} \quad k = 4 \end{cases} \tag{1.11}$$
при довільних значеннях параметра a і незалежної змінної x для всіх значень k з використанням оператора switch.

- Розробити алгоритмі і написати програмний код обчислення суми знакозмінного ряду
$$y = \sum_{i=1}^{7} \frac{(-1)^i (x-a)^{i+1}}{i!} \tag{1.12}$$
при довільних значеннях параметра a і незалежної змінної x трьома варіантами, використовуючи оператори циклу **for**, **while**, **do - while**.

1.3. Завдання для самостійної роботи

Вивчити основи технології програмування [5], [6] в середовищі Borland C++ Builder Enterprise v6.0.

Уважно проаналізувати зміст прикладів теоретичної частини комп'ютерного практикуму №1 та на їх основі реалізувати програмні коди, які були під-

готвлені під час виконання аудиторних завдань у вигляді запрограмованих додатків у середовищі Borland C++ Builder Enterprise v6.0.

Передати викладачу на перевірку звіт про виконане завдання із зображенням блок-схем алгоритмів, лістінгів програмного коду та відкомпільованих виконавчих файлів запрограмованих додатків. Термін виконання завдання для самостійної роботи 2 тижні.

1.4. Варіанти завдань для аудиторної та домашньої роботи студентів

Обчислити і вивести на екранній формі у табличному вигляді значення функції $f(x)$ (див. далі) на заданому інтервалі зміни значень аргумента x від x_{min} до x_{max} із заданим кроком h. Коефіцієнти a, b, c – дійсні числа.

За допомогою інструкцій циклу написати програму, яка обчислює значення суми скінченного ряду S, якщо задано його формулу (див. далі) і кількість членів ряду n.

1.
$$f(x) = \begin{cases} \dfrac{\sqrt{x} - 3a}{x^2 + c} \text{ при } x > 0, \quad c < 0; \\ -(ax + c) \text{ при } x < 0, \quad c > 0; \\ \dfrac{x}{c + a} + \dfrac{b}{x} \text{ в інших випадках.} \end{cases}$$

2.
$$f(x) = \begin{cases} a + \dfrac{2x}{cx + 5a} \text{ при } x < 0, \quad b + c > 5; \\ \dfrac{2x + a}{3x - b} \text{ при } x > 0, \quad b - c < 0; \\ \dfrac{3}{x} - \dfrac{c}{2x^2} \text{ в інших випадках.} \end{cases}$$

3.
$$f(x) = \begin{cases} -\dfrac{2x + c}{cx + 2a} \text{ при } x < 0, \quad b + c > 5; \\ \dfrac{x - a}{\sqrt{x} + c} \text{ при } x > 0, \quad b - c < 0; \\ \dfrac{3x}{c} - \dfrac{b}{2x(x^2 + 1)} \text{ в інших випадках.} \end{cases}$$

4.

$$f(x)=\begin{cases}\dfrac{\sqrt{x+bx^2}}{2x}+\dfrac{a}{3x} \text{ при } x-b>0, \quad c<0;\\ a(bx^2+c)+x \text{ при } x-b<0, \quad c>0;\\ \dfrac{\sqrt{x+bc}}{x\sqrt{x^2+1}+b} \text{ в інших випадках.}\end{cases}$$

5.

$$f(x)=\begin{cases}\dfrac{1}{ax}-b \text{ при } x+10<0, \quad c>2;\\ \dfrac{x-a}{2b+x} \text{ при } x+10>0, \quad b-c<0;\\ \dfrac{5x}{c+b} \text{ в інших випадках.}\end{cases}$$

6.

$$f(x)=\begin{cases}\dfrac{ax^2+bx}{c} \text{ при } x-c<0, \quad b>0;\\ \dfrac{\sqrt{x}}{2x-a}+\dfrac{a}{3x+c} \text{ при } x-c>0;\\ -\dfrac{x\sqrt{x+bc}}{c+b} \text{ в інших випадках.}\end{cases}$$

7.

$$f(x)=\begin{cases}\dfrac{x(x+1)}{3x-c} \text{ при } x>2, \quad b-c<0;\\ ax^3-bx+c \text{ при } x<2, \quad b+c>0;\\ \dfrac{x}{15c}+\sqrt{ax+b} \text{ в інших випадках.}\end{cases}$$

8.

$$f(x)=\begin{cases}x+\dfrac{a}{x-c} \text{ при } x>0, \quad b<0;\\ ax^2-bx \text{ при } x<0, \quad b>0;\\ \dfrac{x}{5c}-x^2 \text{ в інших випадках.}\end{cases}$$

9.

$$f(x)=\begin{cases}x^2+\dfrac{ax}{x^3-c} \text{ при } x>3, \quad ac<0;\\ \sqrt{x^2-b}+cx \text{ при } x<0, \quad ac>0;\\ \dfrac{\sqrt{x}}{1+x}-(c-x)^2 \text{ в інших випадках.}\end{cases}$$

10.

$$f(x) = \begin{cases} x^3 + \dfrac{ax}{x^2 - ac} \text{ при } x < 0, \quad b + c > 5; \\ x\sqrt{x+b} + cx^2 \text{ при } x < 0, \quad b + x > 0; \\ \dfrac{\sqrt{x}}{1+x} - \dfrac{2}{3x} \text{ в інших випадках.} \end{cases}$$

11.

$$f(x) = \begin{cases} 3x + \dfrac{ax}{x^2 + 2c} \text{ при } x > 0, \quad b < 0; \\ -ax + b \text{ при } x < 0, \quad b > 0; \\ \dfrac{2x}{c - a} - bx^2 \text{ в інших випадках.} \end{cases}$$

12.

$$f(x) = \begin{cases} -ax^3 + bcx \text{ при } x < 0, \quad b > 0; \\ \sqrt{x^3} + \dfrac{ax}{cx + b} \text{ при } x > 0, \quad b < 0; \\ \dfrac{4x^2}{cx - a} - bx^3 \text{ в інших випадках.} \end{cases}$$

13.

$$f(x) = \begin{cases} a\sqrt{x+c} - bx^2 \text{ при } x < 4, \quad b + c > 10; \\ x + \dfrac{a}{x - c} \text{ при } x > 4, \quad b < 10; \\ x(x - b) - cx^2 \text{ в інших випадках.} \end{cases}$$

14.

$$S = \sum_{k=1}^{n} \frac{(-1)^{k+2}}{k(k+2)}$$

15.

$$S = \sum_{k=1}^{n} \frac{1}{k^2}$$

16.

$$S = \sum_{k=1}^{n} \frac{k!}{1+k}$$

17.

$$S = \sum_{k=1}^{n} \frac{\sqrt{k}}{k^5}$$

18.

$$S = \sum_{k=1}^{n} \frac{1}{\sum_{i=1}^{k} \sin(i)}$$

19.

$$S = \sum_{k=1}^{n} \left(1 + \frac{1}{k^3}\right)$$

20.

$$S = \sum_{k=1}^{n} \left(1 - \frac{1}{k^2}\right)$$

21.

$$S = \sum_{k=1}^{n} \frac{1}{\sum_{i=1}^{k} \cos(i)}$$

22.

$$S = \sum_{k=1}^{n} \frac{k!}{\sum_{i=1}^{k} \sin(i)}$$

23.

$$S = \sum_{k=1}^{n} \frac{\sum_{i=1}^{k} \sqrt{i}}{\sum_{i=1}^{k} \sin(i)}$$

24.

$$S = \sum_{k=1}^{n} \frac{(-1)^{k+1} \sum_{i=1}^{k} i}{2k+1}$$

25.

$$S = \sum_{k=1}^{n} \frac{\sqrt{\sum_{i=1}^{k} i}}{k^2}$$

1.5. Питання для самоконтролю знань

- Який процес називають лінійним?
- Який процес називають розгалуженим?
- Які оператори в C++ використовуються для організації розгалужень?
- Перелічіть базові логічні операції.
- Вкажіть послідовність дій при виконанні оператора **bool w=2*5<=17%3;**.
- Наведіть результат виконання логічного виразу **(7+3 > 16–4*3)**.
- Обчисліть логічний вираз **(–3 >= 5) || (7 < 9) && (0 < 3)**.
- Вкажіть значення **w** після виконання оператора **bool w=2*5<=17%3;**.
- Назвіть оператор, який записано без помилок:
  а) **if(x <= 6) y = 2*x; else y = cos(x);**
  в) **if(a <> 0) if(b <> 0) y = 2*x;**
  с) **if y <= x then y := exp(x*y);**
  d) **if(x > 0) y = ln(x) else y = x;**
- Вкажіть значення **s** після виконання операторів
  for(s = 0 ,k = 1; k <= 2; k++)
  { m = k;
  do {m++; s += m; } while (m <= 2);
  }

2. Комп'ютерний практикум №2. Алгоритмізація та програмування арифметичного калькулятора

Мета практикуму. Набуття досвіду розробки простого програмного коду комп'ютерних додатків мовою програмування C++ з використанням лінійних, розгалужених та циклічних операторів, змінних типу **string**, **float** та вбудованих функцій для їх опрацювання.

Розробити візуальний додаток аналогічний гаджету "Калькулятор" операційної системи Windows.

2.1. Теоретичні відомості та рекомендації

При розробці додатку передбачити виконання арифметичних операцій над дійсними дробовими числами, піднесення до степеня, комірку пам'яті, очистку компонентів із значенням того чи іншого числа (див. рис.2.1).

При програмуванні функціоналу додатку необхідно підключити математичну бібліотеку <**Math.hpp**> і використовувати як можна в більшій мірі лінійні, розгалужені та циклічні оператори, а також вбудовані функції для роботи із змінними типу **string**. Зокрема **Trim, SubString** та **Pos**. При програмуванні використовувати принцип повторного використання коду. Для побудо-

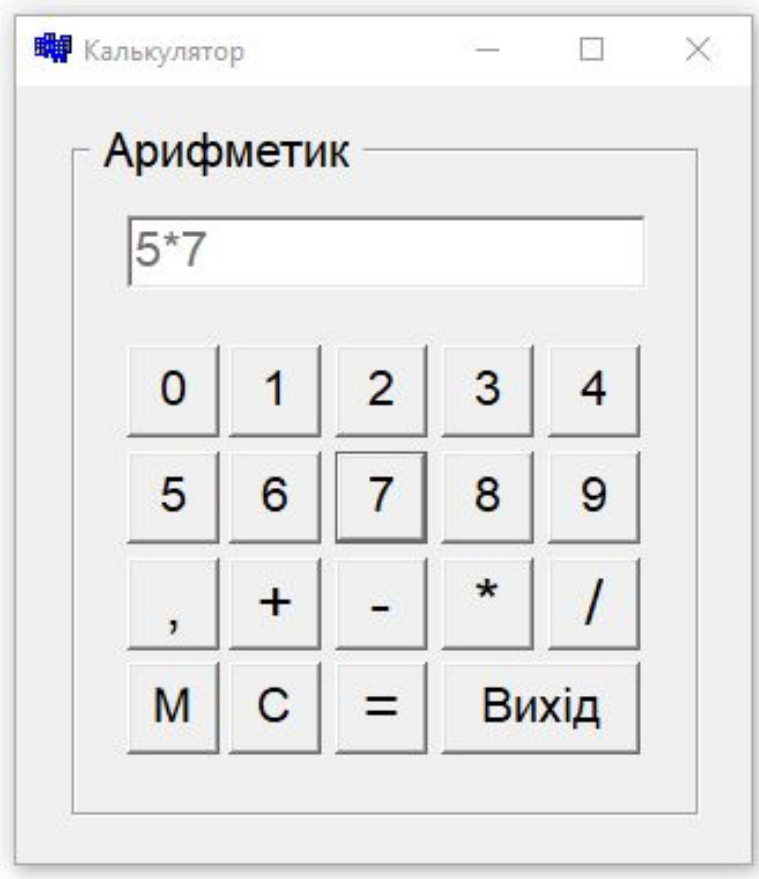


Рис. 2.1 — Екранна форма додатку "Калькулятор"

ви графічного інтерфейсу додатку на головній формі проекту (див. рис.2.1), розташовується колекція компонентів

$$\begin{cases} TForm - 1 \text{ шт.}; \quad TGroupBox - 1 \text{ шт.}; \\ TEdit - 5 \text{ шт.}; \quad TButton - 20 \text{ шт.} \end{cases} \tag{2.1}$$

Нижче приведено приклад програмного коду, який реалізує інтерфейс додатку (див. рис.2.1) та його функціонал

```
/*- Unit01.cpp ------------------------------------------------*/
#include <vcl.h>
#include <Math.hpp>
#include <cmath>
#pragma hdrstop
#include "Unit01.h"
```

```
/*---------------------------------------------------------------*/
#pragma package(smart_init)
#pragma resource "*.dfm"
    TfrmUnit01 *frmUnit01;
    int i, n, indx, flag;
    float x, y;
    AnsiString SP[4] = {'+','-','*','/'};
    AnsiString Data0, q, mem = '0';
/*---------------------------------------------------------------*/
__fastcall TfrmUnit01::TfrmUnit01(TComponent* Owner)
        : TForm(Owner)
{
    frmUnit01->Caption = "Калькулятор";
}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button18Click(TObject *Sender)
{
    frmUnit01->Close();
}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button0Click(TObject *Sender)
{
    Data0 = Trim(Edit0->Text);
    q = Data0.SubString(1,1);
    if (q == '0') {Edit0->Text = "";}
    Edit0->Text = Edit0->Text+((TButton *)Sender)->Caption;
}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button16Click(TObject *Sender)
{
    Edit0->Text = '0';
}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button17Click(TObject *Sender)
{
    Data0 = Trim(Edit0->Text);
    n = Data0.Length();
    q = Data0.SubString(1,1);
    flag = (q == '-') ? -1 : 1;
    indx = -1;
    if ((q == '*')||(q == '/')) {
        ShowMessage("Помилка введення даних!"); }
```

```
    else {
        if ((q == '+')||(q == '-')) {
            Data0 = Data0.SubString(2,n);
        }
        for (int i = 0; i < 4; ++i) {
            if (Data0.Pos(SP[i]) > 1) {
                indx = Data0.Pos(SP[i]);
                try {
                    x = StrToFloat(Data0.SubString(1,indx-1))*
                    flag;
                    y = StrToFloat(Data0.SubString(indx+1,n));
                    float z[4] = {x+y,x-y,x*y,x/y};
                    Edit0->Text =
                    FloatToStr(SimpleRoundTo(z[i],-2));
                } catch(...) {
                    ShowMessage("Помилка введення даних!");
                }
break;
}
}
    }
}
/*-----------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button15Click(TObject *Sender)
{
    Data0 = Trim(Edit0->Text);
    n = Data0.Length();
    indx = -1;

    for (int i = 0; i < 4; ++i) {
if (Data0.Pos(SP[i]) > 1) { indx = i;}
}

    if (mem == '0') {
        if (indx >= 0) { ShowMessage(
"Помилка запису даних в комірку пам'яті!"); }
        else {mem = Data0; Button15->Caption = "M!";}
    }
    else {
        if (indx >= 0) {
            Edit0->Text = Edit0->Text + mem;
```

```
            mem = '0';
            Button15->Caption = "M";
        }

        if (mem != '0') {
            ShowMessage("Введіть оператор арифметичної дії!");
        }
    }
}
/*-----------------------------------------------------------*/
```

2.2. Завдання для аудиторної роботи

- Розробити функціональний алгоритм арифметичного калькулятора.
- Написати програмний код на мові C++, який реалізує функціональний алгоритм арифметичного калькулятора.

2.3. Завдання для самостійної роботи

Вивчити технологію програмування змінних типу string, float та вбудованих функцій для їх опрацювання [5], [6] в середовищі Borland C++ Builder Enterprise v6.0.

Уважно проаналізувати зміст прикладу теоретичної частини комп'ютерного практикуму №2.

На основі функціонального алгоритму та програмного коду на мові C++, які були підготвлені під час виконання аудиторного завдання комп'ютерного практикуму №2, запрограмувати візуальний додаток "Калькулятор" у середовищі Borland C++ Builder Enterprise v6.0 для виконання арифметичних операцій над дійсними дробовими числами, піднесення до степеня, комірку пам'яті, очистку компонентів із значенням того чи іншого числа.

Передати викладачу на перевірку звіт про виконане завдання із зображенням блок-схеми алгоритму, лістінгу програмного коду та відкомпільований виконавчий файл запрограмованого візуального додатку "Калькулятор". Термін виконання завдання для самостійної роботи 2 тижні.

2.4. Питання для самоконтролю знань

- Які основні властивості змінних типу string та float?

- Яким чином програмується взаємна конвертація даних типу string та float у програмі?
- Які функціональні властивості має компонент **TGroupBox**>?
- Яке значення має бібліотека <**Math.hpp**> для програмування функціоналу додатку?
- Яку роль відіграє комірка пам'яті в роботі додатку?

3. Комп'ютерний практикум №3. Алгоритмізація та програмування калькулятора для розв'язку алгебраїчних рівнянь 1,2 та 3 порядку

Мета практикуму. Набуття досвіду розробки складного алгоритму та програмного коду комп'ютерних додатків мовою програмування C++ з використанням лінійних, розгалужених та циклічних операторів, змінних типу **string**, **double** та вбудованих і незадокументованих функцій для їх опрацювання.

Розробити візуальний додаток для знаходження як дійсних так і комплексних розв'язків алгебраїчних рівнянь 1,2 та 3 порядку. При програмуванні додатку використовувати принцип повторного використання коду.

3.1. Теоретичні відомості та рекомендації

Лінійним рівнянням називається рівняння вигляду

$$ax + b = 0 \tag{3.1}$$

Воно може мати лише один корінь

$$x = -\frac{b}{a} \tag{3.2}$$

Квадратним рівнянням називається рівняння вигляду

$$ax^2 + bx + c = 0 \tag{3.3}$$

Воно може мати один корінь, два або жодного (в полі дійсних чисел). Спочатку потрібно обчислити дискримінант

$$D = b^2 - 4ac \tag{3.4}$$

Якщо:

- $D > 0$ то рівняння має два дійсні корені;
- $D = 0$ то рівняння має один корінь;
- $D < 0$ то рівняння має два комлексно-спряжені корені.

Розв'язки квадратного рівняння знаходимо по формулам:

$$\begin{cases} x_1 = \dfrac{-b + \sqrt{D}}{2a} \\ x_2 = \dfrac{-b - \sqrt{D}}{2a} \end{cases} \tag{3.5}$$

Кубічним рівнянням називається рівняння вигляду

$$ax^3 + bx^2 + cx + d = 0 \tag{3.6}$$

a, b, c, d - постійні коефіцієнти, а x - змінна величина. Ми розглянемо випадок, коли коефіцієнти є дійсними числами.

Кубічне рівняння завжди має не більше трьох коренів (над комплексним полем завжди три корені, з урахуванням кратності.) і завжди має хоча б 1 (дійсний) корінь.

Рівняння 3.6 очевидно можна звести до наступного вигляду

$$x^3 + ax^2 + bx + c = 0 \tag{3.7}$$

a, b, c - постійні коефіцієнти, а x - змінна величина.

Для аналітичного розв'язку кубічних рівнянь зазвичай використовують тригонометричну формулу Вієта. Для її використання введемо до розгляду наступні величини

$$\begin{cases} Q = \dfrac{1}{9}(a^2 - 3b) \\ R = \dfrac{1}{54}(2a^3 - 9ab + 27c) \\ D = Q^3 - R^2 \end{cases} \tag{3.8}$$

Всі можливі випадки типу коренів легко визначити за допомогою знаку дискримінанта кубічного рівняння D; Очевидно можливі тільки 3 випадки для D коли

- $D > 0$ тоді рівняння має 3 різні дійсні корені

$$\begin{cases} \varphi = \dfrac{1}{3}\arccos\left(\dfrac{R}{Q^{3/2}}\right) \\ x_1 = -2Q^{1/2}\cos(\varphi) - \dfrac{a}{3} \\ x_2 = -2Q^{1/2}\cos\left(\varphi + \dfrac{2\pi}{3}\right) - \dfrac{a}{3} \\ x_3 = -2Q^{1/2}\cos\left(\varphi - \dfrac{2\pi}{3}\right) - \dfrac{a}{3} \end{cases} \tag{3.9}$$

- $D = 0$ тоді рівняння має 3 дійсні корені, причому як мінімум два із них є виродженими

$$\begin{cases} x_1 = -2R^{1/3} - \dfrac{a}{3} \\ x_2 = x_3 = R^{1/3} - \dfrac{a}{3} \end{cases} \tag{3.10}$$

- $D < 0$ тоді рівняння має 3 різні корені, причому один із них буде дійсним, а два інші будуть комплексно спряженими

$$\begin{cases} \varphi = \dfrac{1}{3}\mathrm{arch}\left(\dfrac{|R|}{|Q|^{3/2}}\right) \\ x_1 = -2\mathrm{sign}(R)|Q|^{1/2}\,\mathrm{ch}(\varphi) - \dfrac{a}{3} \\ x_2 = \mathrm{sign}(R)|Q|^{1/2}\,\mathrm{ch}(\varphi) - \dfrac{a}{3} + (3|Q|)^{1/2}\,\mathrm{sh}(\varphi)i \\ x_3 = \mathrm{sign}(R)|Q|^{1/2}\,\mathrm{ch}(\varphi) - \dfrac{a}{3} - (3|Q|)^{1/2}\,\mathrm{sh}(\varphi)i \end{cases} \tag{3.11}$$

де

$$\begin{cases} \mathrm{ch}(x) = \dfrac{1}{2}(\exp(x) + \exp(-x)) \\ \mathrm{sh}(x) = \dfrac{1}{2}(\exp(x) - \exp(-x)) \\ \mathrm{arch}(x) = \ln(x + (x^2 - 1)^{1/2}) \end{cases} \tag{3.12}$$

Приклад програмування візуального додатку

При програмуванні функціоналу додатку необхідно підключити математичну бібліотеку <**Math.hpp**> і використовувати як можна в більшій мірі лінійні, розгалужені та циклічні оператори, а також вбудовані функції для роботи із

змінними типу **string**. Зокрема **Trim, SubString** та **Pos**.

При програмуванні додатку виникають проблеми з обмеженістю колекції вбудованих функцій мови програмування C++ та наявністю в C++ корисних, але незадокументованих функцій. З огляду на цей стан справ потрібно ввести додаткові функції, а саме int sign(Double x), double Round(double X, int precision), double fn_Acosh(double x) і використати незадокументовану функцію string String(double X).

При програмуванні використовувати принцип повторного використання коду.

Для побудови графічного інтерфейсу додатку на головній формі проекту (див. рис.3.1), розташовується колекція компонентів

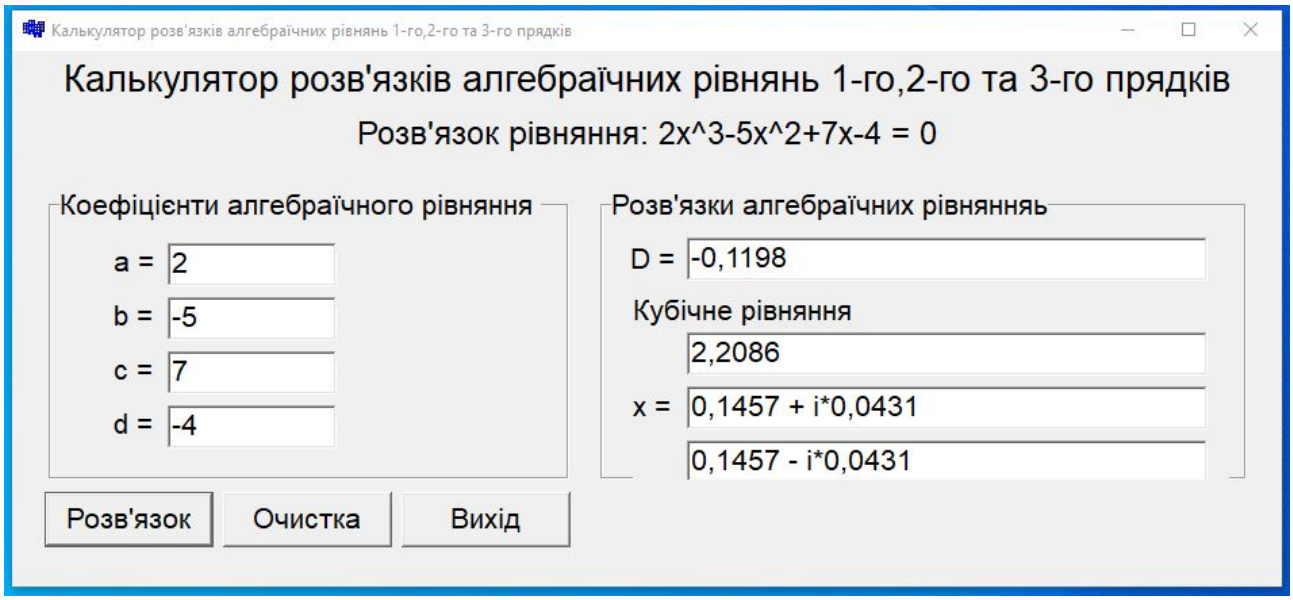


Рис. 3.1 — Екранна форма додатку для розв'язку алгеборічних рівнянь

$$\begin{cases} TForm - 1 \text{ шт.}; \quad TPanel - 2 \text{ шт.}; \\ TGroupBox - 2 \text{ шт.}; \quad TLabel - 7 \text{ шт.}; \\ TEdit - 8 \text{ шт.}; \quad TButton - 3 \text{ шт.} \end{cases} \tag{3.13}$$

Нижче приведено приклад програмного коду, який реалізує інтерфейс додатку (див. рис.3.1) та його функціонал

```
/*- Unit01.cpp ------------------------------------------------*/
#include <iostream>
#include <string>
#include <Math.hpp>
#include <math.h>
#include <vcl.h>
#pragma hdrstop
```

```
#include "Unit01.h"
/*---------------------------------------------------------------*/
#pragma package(smart_init)
#pragma resource "*.dfm"
TfrmUnit01 *frmUnit01;
double a,b,c,d,D2,X0,ReX0,ImX0,Q,R,D3,g,
Z1,Z2,ReZ2,ImZ2,Z3,ReZ3,ImZ3,pi;
AnsiString EdtA, EdtB, EdtC, EdtD, I2;

int sign(double x) {
if (x > 0) return  1;
if (x < 0) return -1;
return 0;
}

double Round(double X, int precision)
{
AnsiString k, Data0 = FloatToStr(X);
bool sign = false;
double Q;
int pos = Data0.Pos(DecimalSeparator);
int len = Data0.Length();
if ((pos == 0) || ((len - pos) <= precision))
return X;
AnsiString prom = Data0.SubString(1,1);
if (prom == '-') sign = true;
k = Data0.SubString(pos+precision+1,1);
int i = StrToInt(k);
if (i >= 5){
if (sign)
Q = floor(X*Power(10,precision))/
Power(10,precision);
else
Q = ceil(X*Power(10,precision))/
Power(10,precision);
} else {
if (!sign)
Q = floor(X*Power(10,precision))/
Power(10,precision);
else
Q = ceil(X*Power(10,precision))/
```

```
Power(10,precision);
}
return Q;
}

double fn_Acosh(double x) {
    return log(x + pow(x*x - 1,2));
}

/*---------------------------------------------------------------*/
__fastcall TfrmUnit01::TfrmUnit01(TComponent* Owner)
    : TForm(Owner)
{
}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::FormResize(TObject *Sender)
{
frmUnit01->Width  = 955;
frmUnit01->Height = 435;
GroupBox0->Height = 215;
GroupBox1->Height = GroupBox0->Height;
Panel0->Visible = true;
Panel1->Visible = false;
Panel2->Visible = false;
Panel0->Left = 24;
Panel0->Top = 78;
Panel1->Left = 24;
Panel1->Top = Panel0->Top;
Panel2->Left = 24;
Panel2->Top = Panel0->Top;
EditX0->Text = String(X0);
EditD2->Text = "Дискримінант не визначено";
}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button2Click(TObject *Sender)
{
frmUnit01->Close();
}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button1Click(TObject *Sender)
{
```

```
int i;
for( i=0; i<ComponentCount; i++ ) {
TEdit *Edit =
dynamic_cast<TEdit*>(Components[i]);
if( Edit ) {
Edit->Text = "1";
}
}
EditD2->Text = "Дискримінант не визначено";
}
/*-----------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button0Click(TObject *Sender)
{
    Panel0->Visible = false;
    Panel1->Visible = false;
    Panel2->Visible = false;
    Panel0->Left = 24;
    Panel0->Top = 78;
    Panel1->Left = 24;
    Panel1->Top = Panel0->Top;
    Panel2->Left = 24;
    Panel2->Top = Panel0->Top;
    EditX0->Text = "0";
    EditX1->Text = "0";
    EditX2->Text = "0";
    EditZ1->Text = "0";
    EditZ2->Text = "0";
    EditZ3->Text = "0";
    AnsiString X3 = "";
    AnsiString X2 = "";
    AnsiString X1 = "";
    AnsiString X0 = "";
    AnsiString Z0 = "";
    EditD2->Text = "Дискримінант не визначено";
    pi = 3.141592653589793;

    if (EditA->Text == "") {EditA->Text = "1";};
    if (EditB->Text == "") {EditB->Text = "1";};
    if (EditC->Text == "") {EditC->Text = "1";};
    if (EditD->Text == "") {EditD->Text = "1";};
```

```
    EdtA = EditA->Text;
    EdtB = EditB->Text;
    EdtC = EditC->Text;
    EdtD = EditD->Text;

    if (EdtA != "0") {X3 = EdtA+"x^3";}
    if (EdtB != "0") {X2 = EdtB+"x^2";}
    if (EdtC != "0") {X1 = EdtC+"x";}
    if (EdtD != "0") {X0 = EdtD;}
    X2 = (EdtB.SubString(1,1) == "-") ? X2 :
(EdtA != "0")&&(EdtB != "0") ? "+"+X2 : X2;
    X1 = (EdtC.SubString(1,1) == "-") ? X1 :
(EdtB != "0")&&(EdtC != "0") ? "+"+X1 : X1;
    X0 = (EdtD.SubString(1,1) == "-") ? X0 :
(EdtC != "0")&&(EdtD != "0") ? "+"+X0 : X0;
    a = EdtA.ToDouble();
    b = EdtB.ToDouble();
    c = EdtC.ToDouble();
    d = EdtD.ToDouble();
    if ((sign(a) != 0)||(sign(b) != 0)||(sign(c) != 0)) {
Header1->Caption =
"Розв'язок рівняння: "+X3+X2+X1+X0+" = 0";}
    else {
Header1->Caption =
"Рівняння "+X3+X2+X1+X0+" = 0"+
" записене некоректно!";}

    if ((sign(a) == 0)&&(sign(b) == 0)&&
(sign(c) != 0)&&(sign(d) != 0)) {
X0 = Round(-d/c,4);
EditX0->Text = String(X0);
} else
    if ((sign(a) == 0)&&(sign(b) != 0)&&(sign(d) != 0)) {
D2 = c*c-4.0*b*d;
EditD2->Text = String(D2);

ReX0 = Round(-c/(2.0*b),4);
ImX0 = Round(pow(fabs(D2),0.5)/(2.0*b),4);

if (sign(D2) == -1) {
EditX1->Text =
```

```
String(ReX0)+" +
i*"+String(fabs(ImX0));
EditX2->Text =
String(ReX0)+" -
i*"+String(fabs(ImX0));
} else {
X0 = ReX0+ImX0;
EditX1->Text = String(X0);
X0 = ReX0-ImX0;
EditX2->Text = String(X0);
}
} else
    if ((sign(a) != 0)&&(sign(d) != 0)) {
double k = a;
a = b/k; b = c/k; c = d/k;
Q = (pow(a,2)-3*b)/9;
R = (2*pow(a,3)-9*a*b+27*c)/54;
D3 = Round(pow(Q,3.0)-pow(R,2.0),4);

EditD2->Text = String(D3);

if (sign(D3) > 0) {
g = acos(R/pow(Q,3.0/2.0));
Z1 = Round(-2*pow(Q,1.0/2.0)*
cos(g)-a/3.0,4);
Z2 = Round(-2*pow(Q,1.0/2.0)*
cos(g+2.0*pi/3)-a/3.0,4);
Z3 = Round(-2*pow(Q,1.0/2.0)*
cos(g-2.0*pi/3)-a/3.0,4);
EditZ1->Text = String(Z1);
EditZ2->Text = String(Z2);
EditZ3->Text = String(Z3);
} else
if (sign(D3) == 0) {
Z1 = Round(-2.0*
pow(R,1.0/3.0)-a/3.0,4);
Z2 = Round(pow(R,1.0/3.0)-a/3.0,4);
Z3 = Z2;
EditZ1->Text = String(Z1);
EditZ2->Text = String(Z2);
EditZ3->Text = String(Z3);
```

```
} else {
g = fn_Acosh(fabs(R)/
pow(fabs(Q),3.0/2.0))/3.0;
Z1 = Round(-2*sign(R)*
pow(fabs(Q),1.0/2.0)*
cosh(g)-a/3.0,4);
ReZ2 = Round(sign(R)*
pow(fabs(Q),1.0/2.0)*
cosh(g)-a/3.0,4);
ImZ2 = Round(pow(3*fabs(Q),1.0/2.0)*
sinh(g),4);
ReZ3 = ReZ2;
ImZ3 = -ImZ2;
EditZ1->Text = String(Z1);
EditZ2->Text = String(ReZ2)+" +
i*"+String(fabs(ImZ2));
EditZ3->Text = String(ReZ3)+" -
i*"+String(fabs(ImZ3));
}
}

if ((a==0)&&(b==0)) {Panel0->Visible = true;}
else if ((a==0)&&(b!=0)) {Panel1->Visible = true;}
else if (a!=0) {Panel2->Visible = true;};
}
/*-----------------------------------------------------------------*/
```

3.2. Завдання для аудиторної роботи

- Розробити функціональний алгоритм калькулятора для розв'язку алгебраїчних рівнянь 1,2 та 3 порядку.
- Написати програмний код на мові C++, який реалізує функціональний калькулятора для розв'язку алгебраїчних рівнянь 1,2 та 3 порядку.

3.3. Завдання для самостійної роботи

Вивчити технологію повторного використання коду, взаємної конвертації змінних типу **string**, **double** та використання вбудованих (в тому числі і незадокументованих) функцій для їх опрацювання в середовищі [5], [6] Borland C++ Builder Enterprise v6.0.

Уважно проаналізувати приклад, який був розглянутий під час комп'ютерного практикуму №3.

На основі функціонального алгоритму та програмного коду на мові C++, які були підготвлені під час виконання аудиторного завдання комп'ютерного практикуму №3, запрограмувати візуальний додаток для знаходження як дійсних так і комплекснх розв'язків алгебраїчних рівнянь 1,2 та 3 порядку у середовищі Borland C++ Builder Enterprise v6.0.

Передати викладачу на перевірку звіт про виконане завдання із зображенням блок-схеми алгоритму, лістінгу програмного коду та відкомпільований виконавчий файл запрограмованого візуального додатку для знаходження як дійсних так і комплекснх розв'язків алгебраїчних рівнянь 1,2 та 3 порядку. Термін виконання завдання для самостійної роботи 2 тижні.

3.4. Питання для самоконтролю знань

- Які основні властивості змінних типу **string**, **double**?
- Яку роль відіграє незадокументована фунеція string String(double X)?
- Яким чином програмується взаємна конвертація даних типу **string**, **double** у програмі?
- Який алгоритм візуалізації розв'язків алгебраїчних рівнянь різного порядку?
- Який алгоритм формування та візуалізації комплексних роз'язків алгебраїчних рівнянь?

4. Комп'ютерний практикум №4. Алгоритмізація та програмування додатку для дослідження динаміки двовимірного маятника

Мета практикуму. Набуття досвіду розробки алгоритмів та програмного коду комп'ютерних додатків мовою програмування C++ з використанням лінійних, розгалужених та циклічних операторів і побудови динамічних графіків для ілюстрації розвитку фізико - технічних процесів.

Отримати досвід розробити візуального додатку, який вирішує питання побудови динамічного графіку на канві компонента **Image** на прикладі траєкторій суперпозиції ортогональних гармонічних коливань двовимірного математичного маятника.

4.1. Теоретичні відомості та рекомендації

Всі траєкторії, по яких може рухатися тіло, що бере участь одночасно в двох взаємно перпендикулярних коливаннях

$$\begin{cases} x = A_x(t)\cos(\varphi_x(t)); \\ y = A_y(t)\cos(\varphi_y(t)), \end{cases} \tag{4.1}$$

називаються фігурами Ліссажу. Тут $A_{x,y}$ — амплітуди, $\varphi_{x,y}$ — фази коливань, t — параметр еволюції (час).

В більшості випадків у курсах загальної фізики розглядають лінійну залежність фаз коливань від параметра еволюції при постійних амплітудах

$$\begin{cases} x = A_x\cos(\omega_x t + \varphi_x); \\ y = A_y\cos(\omega_y t + \varphi_y), \end{cases} \tag{4.2}$$

де $\omega_{x,y}$ — циклічні частоти, а $\varphi_{x,y}$ — початкові фази коливань.

Фігури Ліссажу можуть бути замкнутими і незамкнутими. Під замкнутою фігурою Ліссажу розуміють таку фіксовану сукупність точок (траєкторію), через кожну з яких тіло проходить багато разів в процесі свого руху. У лінійному випадку замкнуті фігури Ліссажу виникають, якщо відношення частот перпендикулярних коливань ω_x, ω_y, що складаються, можливо представити як відношення цілих чисел.

Якщо відношення частот перпендикулярних коливань ω_x, ω_y, що складаються, неможливо представити як відношення цілих чисел, то навіть якщо в якийсь момент часу тіло і повернеться в деяку точку простору, де воно побувало раніше, то подальший його рух буде проходити через точоки, в яких тіло раніше не спостерігалося. Такі траєкторії ніколи повністю не повторюються в часі, а відповідні їм фігури Ліссажу вважаються незамкнутими.

Приклад програмування візуального додатку

Зосередимося на прикладі додатку для дослідження динаміки суперпозиції ортогональних гармонічних коливань двовимірного математичного маятника (див. рис.4.1). Будемо вважати амплітуди, циклічні частоти та зсув початкової фази параметрами задачі, значення яких задаються в компонентах **Edit**. Для надання динаміки в розрахунках траєкторії руху маятника використати контонент **Timer**. При розробці додатку необхідно підключити математичну бібліотеку <**Math.hpp**> і використовувати як можна в більшій мірі лінійні, розгалужені та циклічні оператори, а також вбудовані функції для роботи із змінними типу **String**. Зокрема **Trim, SubString** та **Pos**. При програму-

ванні додатку використовувати принцип повторного використання коду. Для

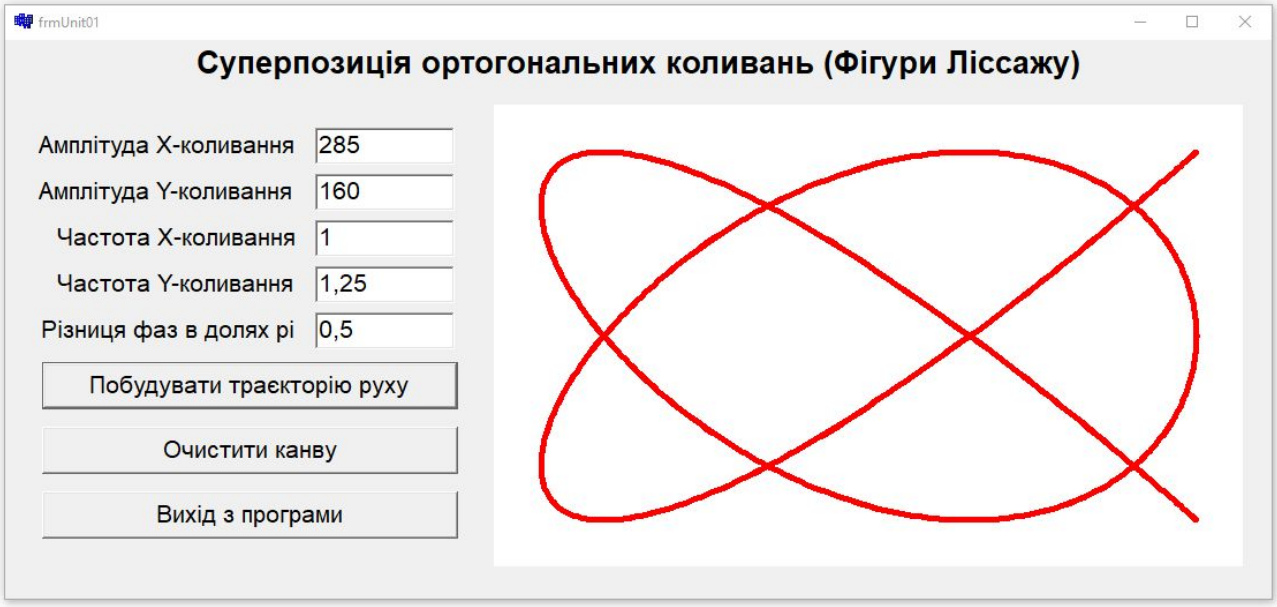


Рис. 4.1 — Екранна форма додатку для дослідження динаміки двовимірного маятника

побудови графічного інтерфейсу додатку на головній формі проекту (див. рис.4.1), розташовується колекція компонентів

$$\begin{cases} TForm - 1 \text{ шт.}; \quad TPanel - 2 \text{ шт.}; \\ TLabel - 5 \text{ шт.}; \quad TEdit - 5 \text{ шт.}; \\ TButton - 3 \text{ шт.} \quad TImage - 1 \text{ шт.}; \\ TTimer - 1 \text{ шт.} \end{cases} \tag{4.3}$$

Нижче приведено приклад програмного коду, який реалізує інтерфейс додатку (див. рис.4.1) та його функціонал

```
/*- Unit01.cpp ------------------------------------------------*/
#include <vcl.h>
#pragma hdrstop
#include "math.h"
#include "Unit01.h"
/*---------------------------------------------------------------*/
#pragma package(smart_init)
#pragma resource "*.dfm"
TfrmUnit01 *frmUnit01;
const float pi=3.141563;
int i,cx,cy,px,py;
float a,b,wx,wy,q;
TRect aRect;
/*---------------------------------------------------------------*/
```

```
__fastcall TfrmUnit01::TfrmUnit01(TComponent* Owner)
        : TForm(Owner)
{}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::FormShow(TObject *Sender)
{
aRect = Rect(0,0,ClientWidth,ClientHeight);
Image0->Canvas->Brush->Color = clWhite;
Image0->Canvas->FillRect(aRect);
Edit0->Text = FloatToStr(Image0->Width/2-40);
Edit1->Text = FloatToStr(Image0->Height/2-40);
Edit2->Text = FloatToStr(1);
Edit3->Text = FloatToStr(1);
Edit4->Text = FloatToStr(0.5);
}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button0Click(TObject *Sender)
{
cx=Image0->Width/2;
cy=Image0->Height/2;

a=StrToFloat(Edit0->Text);
b=StrToFloat(Edit1->Text);
wx=StrToFloat(Edit2->Text);
wy=StrToFloat(Edit3->Text);
q=pi*StrToFloat(Edit4->Text);

Image0->Canvas->MoveTo(cx,cy);
Image0->Canvas->Pen->Width=5;
Image0->Canvas->Pen->Color=clWhite;
i = -1;
Timer0->Enabled = true;
}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::Timer0Timer(TObject *Sender)
{
i++;
px=a*(cos(wx*i*pi/180));
py=b*(cos(wy*i*pi/180+q));
if (i==0) {
        Image0->Canvas->Pen->Color=clWhite; }
else {
```

```
Image0->Canvas->Pen->Color=clRed;
}
Image0->Canvas->LineTo(cx+px,cy+py);
}
/*-----------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button1Click(TObject *Sender)
{
Timer0->Enabled = false;
Image0->Canvas->FillRect(Rect(0,0,
Image0->Width,Image0->Height));
}
/*-----------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button2Click(TObject *Sender)
{
frmUnit01->Close();
}
/*-----------------------------------------------------------------*/
```

4.2. Завдання для аудиторної роботи

- Розробити функціональний алгоритм додатку для дослідження динаміки двовимірного математичного маятника.
- Написати програмний код на мові C++, який реалізує функціональний додатку для дослідження динаміки двовимірного математичного маятника.

4.3. Завдання для самостійної роботи

Вивчити технологію програмування у середовищі Borland C++ Builder Enterprise v6.0 зображень [5], [6] на канві компонента **TImage**.

Уважно проаналізувати приклад, який був розглянутий під час комп'ютерного практикуму №4.

На основі функціонального алгоритму та програмного коду на мові C++, які були підготвлені під час виконання аудиторного завдання комп'ютерного практикуму №4, запрограмувати візуальний додаток у середовищі Borland C++ Builder Enterprise v6.0, який вирішує питання побудови у динамічному режимі фігур Ліссажу для довільних значень параметрів $A_{x,y}$, $\omega_{x,y}$, $\varphi_{x,y}$.

Передати викладачу на перевірку звіт про виконане завдання із зображенням блок-схеми алгоритму, лістінгу програмного коду та відкомпільований

виконавчий файл запрограмованого візуального додатку для побудови у динамічному режимі фігур Ліссажу для довільних значень параметрів $A_{x,y}$, $\omega_{x,y}$, $\varphi_{x,y}$. Термін виконання завдання для самостійної роботи 2 тижні.

4.4. Варіанти завдань для аудиторної та домашньої роботи студентів

Побудувати додаток для дослідження у динамічному режимі фігур Ліссажу для таких коливальних систем

1. $$\begin{cases} x = A_x \cos((\omega_x t)^2 + \varphi_x); \\ y = A_y \cos(\omega_y t + \varphi_y) \end{cases}$$

2. $$\begin{cases} x = A_x \cos(\cos(\omega_x t) + \varphi_x); \\ y = A_y \cos(\omega_y t + \varphi_y) \end{cases}$$

3. $$\begin{cases} x = A_x \cos(\omega_x t + \varphi_x); \\ y = A_y e^{-\beta t} \cos(\omega_y t + \varphi_y) \end{cases}$$

4. $$\begin{cases} x = A_x \cos(\omega_x t + \varphi_x); \\ y = A_y \cos(\ln(\omega_y t) + \varphi_y) \end{cases}$$

5. $$\begin{cases} x = A_x e^{-\beta t} \cos(\omega_x t + \varphi_x); \\ y = A_y e^{+\beta t} \cos(\omega_y t + \varphi_y) \end{cases}$$

6. $$\begin{cases} x = A_x \cos(\exp(\omega_x t) + \varphi_x); \\ y = A_y \cos(\ln(\omega_y t) + \varphi_y) \end{cases}$$

7. $$\begin{cases} x = A_x \sin(\omega t) \cos(\omega_x t + \varphi_x); \\ y = A_y \cos(\omega_y t + \varphi_y) \end{cases}$$

8.
$$\begin{cases} x = A_x \cos(\omega t)\cos(\omega_x t + \varphi_x); \\ y = A_y e^{-\beta t}\cos(\omega_y t + \varphi_y) \end{cases}$$

9.
$$\begin{cases} x = A_x \cos(\omega_x t^2 + \varphi_x); \\ y = A_y \cos(\omega t)\cos(\omega_y t + \varphi_y) \end{cases}$$

10.
$$\begin{cases} x = A_x \cos(\sin(\omega_x t) + \varphi_x); \\ y = A_y \cos(\omega t)\cos(\omega_y t + \varphi_y) \end{cases}$$

11.
$$\begin{cases} x = A_x \cos(\mathrm{tg}(\omega_x t) + \varphi_x); \\ y = A_y \cos(\omega_y t + \varphi_y) \end{cases}$$

12.
$$\begin{cases} x = A_x e^{-\beta t^2}\cos(\omega_x t + \varphi_x); \\ y = A_y \cos(\mathrm{tg}(\omega_y t) + \varphi_y) \end{cases}$$

13.
$$\begin{cases} x = A_x \cos(\mathrm{tg}(\omega_x t) + \varphi_x); \\ y = A_y \cos(\sin(\omega_y t) + \varphi_y) \end{cases}$$

14.
$$\begin{cases} x = A_x e^{\sin(\omega t)}\cos(\omega_x t + \varphi_x); \\ y = A_y \cos(\mathrm{tg}(\omega_y t) + \varphi_y) \end{cases}$$

15.
$$\begin{cases} x = A_x \cos\left(\dfrac{\pi}{\omega_x t + \varphi_x}\right); \\ y = A_y \cos(\mathrm{ctg}(\omega_y t) + \varphi_y) \end{cases}$$

16.
$$\begin{cases} x = A_x \cos\left(\dfrac{\sin(\omega t)}{\omega_x t + \varphi_x}\right); \\ y = A_y \cos(\ln(\omega_y t) + \varphi_y) \end{cases}$$

17.
$$\begin{cases} x = A_x e^{-\beta t}\cos(\omega_x t + \varphi_x); \\ y = A_y \cos(\omega t)\cos(\omega_y t + \varphi_y) \end{cases}$$

18.
$$\begin{cases} x = A_x \cos(\ln(\omega_x t) + \varphi_x); \\ y = A_y \cos(\exp(\omega_y t) + \varphi_y) \end{cases}$$

19.
$$\begin{cases} x = A_x \cos\left(\dfrac{2\pi}{\sin(\omega_x t) + \varphi_x}\right); \\ y = A_y \cos(\ln(\omega_y t) + \varphi_y) \end{cases}$$

20.
$$\begin{cases} x = A_x e^{\cos(\omega t)} \cos(\omega_x t + \varphi_x); \\ y = A_y e^{-\beta t} \cos(\omega_y t + \varphi_y) \end{cases}$$

21.
$$\begin{cases} x = A_x \cos(e^{-\omega_x t} + \varphi_x); \\ y = A_y e^{\sin(\omega t)} \cos(\omega_y t + \varphi_y) \end{cases}$$

22.
$$\begin{cases} x = A_x \cos((\omega_x t)^{1.5} + \varphi_x); \\ y = A_y \cos((\omega_y t)^{0.75} + \varphi_y) \end{cases}$$

23.
$$\begin{cases} x = \dfrac{A_x}{2 + \cos(\omega t)} \cos(\omega_x t + \varphi_x); \\ y = A_y \cos(\omega_y t + \varphi_y) \end{cases}$$

24.
$$\begin{cases} x = A_x \cos(\omega t) \cos(\omega_x t + \varphi_x); \\ y = A_y (2 + \cos(\omega t)) \cos(\omega_y t + \varphi_y) \end{cases}$$

25.
$$\begin{cases} x = A_x \ln(\beta t) \cos(\omega_x t + \varphi_x); \\ y = A_y \cos(\omega_y t + \varphi_y) \end{cases}$$

4.5. Питання для самоконтролю знань

- Що таке фігури Ліссажу?
- Які функціональні властивості має компонент **TImage**?
- Якою інструкцією можна зобразити пряму на канві компонента **TImage**?

- Якою інструкцією можна змінити колір зображення прямої на канві компонента **TImage**?
- Яка роль компонента **TTimer** у побудові динамічних фігур Ліссажу?

5. Комп'ютерний практикум №5. Алгоритмізація та програмування опрацювання статичних одновимірних та двовимірних масивів

Мета практикуму. Набуття досвіду побудови базових алгоритмів для опрацювання масивів та їх програмної реалізації.

5.1. Теоретичні відомості та рекомендації

Опрацювання масиву полягає у виконанні операцій над його елементами. Окрім введення —виведення масивів існує перелік найпоширеніших базових алгоритмів опрацювання масивів:

- обчислення узагальнювальних характеристик (сум, добутків і кількості елементів);
- пошук максимального чи мінімального елемента;
- пошук заданих елементів;
- переставляння елементів;
- упорядкування масивів.

Базові алгоритми опрацювання масивів можуть бути складовими блоками при розв'язуванні більш складних задач.

Програмування базових алгоритмів опрацювання одновимірних масивів

Приклад 5.1. Скласти схему алгоритму і проект програми для обчислення елементів одновимірного масиву з 15-ти елементів за формулою

$$A_i = \lg(i) + \mathrm{tg}(2^i), \quad i = 1, ..., 15 \tag{5.1}$$

та їхнього виведення на форму, а також визначення мінімального елемента та його порядкового номера.

Розв'язок. Алгоритм розв'язування цього прикладу та головна форма додатку зображені на рис.5.1. У вигляді словесного алгоритму пошук мінімального елемента масиву і його індексу можна записати в такий спосіб:

- вважаємо перше число за найменше. Для цього присвоюємо перший елемент масиву $A[0]$ у змінну min, а змінній ind — значення 0 (тобто індекс першого елемента масиву);
- у циклі переглядаємо всі елементи. Якщо зустрічаємо число, яке є менше за min, запам'ятовуємо це число у min, а його індекс — в ind.

Коли цикл завершиться, у змінній min буде зберігатися найменший елемент масиву, а його індекс — у змінній ind. Оскільки нумерація індексів масиву розпочинається з 0, а не з 1, як у повсякденному житті, виводиться значення індексу, збільшене на 1.

```
/*- Unit01.cpp ------------------------------------------------*/
#include <math.h>
double A[15]; int i;
/*-----------------------------------------------------------------*/
// Обчислення елементів вектора
void__fastcall TForm1:: Button1Click(TObject *Sender)
{ Memo1->Clear();
for(i=0; i<15; i++)
{ A[i]=log10(i+1)+tan(pow(2.,i+1));
Memo1->Lines->Add(FormatFloat("0.000", A[i]));
}
}
//---------------------------------
// Мінімальний елемент
void __fastcall TForm1::Button2Click(TObject *Sender)
{ double min=A[0]; int ind=0;
for(i=1; i<15; i++)
if(min > A[i]) { min=A[i]; ind=i; }
Edit1->Text=FormatFloat("0.000", A[ind]) + " індекс - " +
IntToStr(ind+1);
}
// Якщо треба віднайти лише значення мінімального елемента масиву,
// то програма для Button2Click матиме спрощений вигляд:
void __fastcall TForm1::Button2Click(TObject *Sender)
{ double min=A[0];
for(i=1; i<15; i++)
if(min > A[i]) min=A[i];
```

```
Edit1->Text=FormatFloat("0.000", min);
}
// Можна відшукати лише індекс мінімального елемента,
// а потім за його допомогою визначити мінімальний елемент.
// У цьому способі мінімальним буде еле-мент з віднайденим індексом:
// A[ind]. Цей спосіб реалізовано у такій програмі.
void __fastcall TForm1::Button2Click(TObject *Sender)
{ int ind=0;
for(i=1; i<15; i++)
if(A[ind] > A[i]) ind=i;
Edit1->Text=FormatFloat("0.000", A[ind]) + " індекс - " +
IntToStr(ind+1);
}
/*-----------------------------------------------------------------*/
```

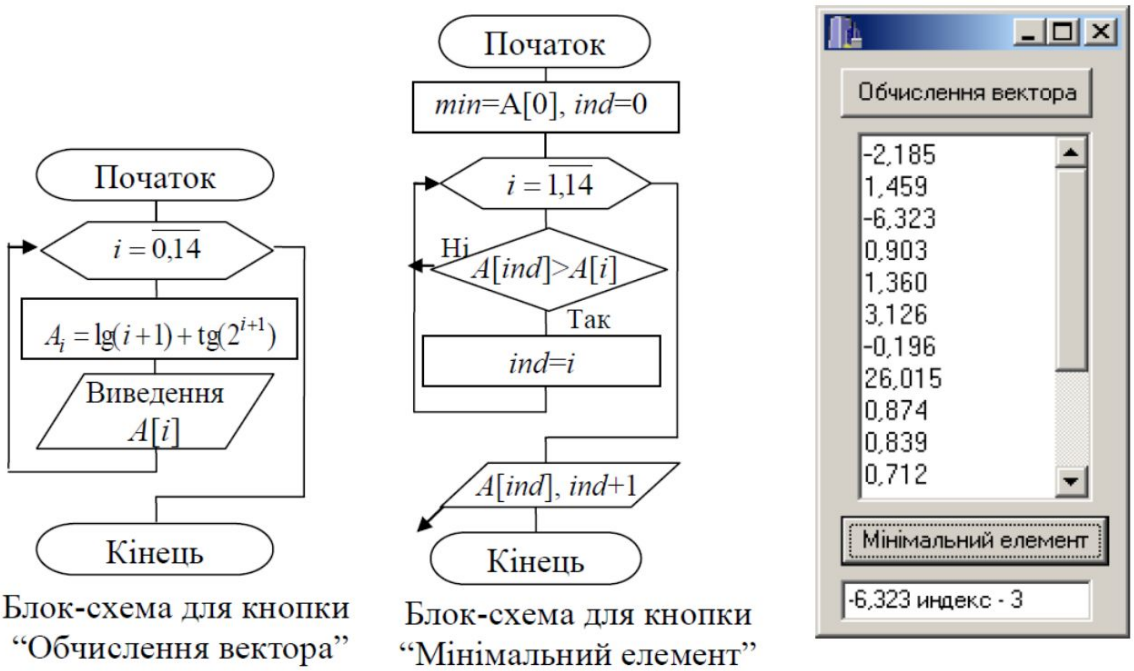


Рис. 5.1 — Блок-схема алгоритму та екранна форма додатку для обчислення елементів одновимірного масиву

Приклад 5.2. Розробити додаток (див. рис.5.2) для введення одновимірного масиву до 15-ти елементів та обчислення:

- добутку ненульових елементів;
- кількості від'ємних елементів;
- максимального елемента з переставлянням його з першим елементом вектора.

Розв'язок. При програмуванні потрібно звернути увагу на те, що чисел може бути введено менш за 15, але оголосити масив слід з максимально можливою кількістю елементів 15. У програмі при опрацюванні масиву індекс у циклі змінюється від 0 до **n**, де **n** — кількість реально введених чисел. Якщо введено менше за 15 чисел, зчитуються всі числа, які було введено. Якщо чисел є більше за 15, зайвими числами нехтують.

Нижче приведено приклад програмного коду, який реалізує інтерфейс додатку (див. рис.5.2) та його функціонал

```
/*- Unit01.cpp ----------------------------------------------*/
void __fastcall TForm1::
BitBtn1Click (TObject *Sender)
{double A[15]; int i, n;
n = StrToInt(Memo1->Lines->Count);
if (n>15) n=15;
for (i=0; i<n; i++)
A[i]=StrToFloat(Memo1->Lines->Strings[i]);
int k=0, ind=0;
double p=1, max=A[0];
for (i=0; i<n; i++)
{ // Кількість від"ємних елементів
if (A[i] < 0) k++;
// Добуток ненульових елементів
if (A[i] != 0) p *= A[i];
// Максимальний елемент та його індекс
if (A[i] > max)
{ max=A[i]; ind=i;
}
}
// Поміняти місцями максимальний елемент
// з першим елементом вектора
A[ind]=A[0]; A[0]=max;
Edit1->Text=FloatToStr(p);
Edit2->Text=IntToStr(k);
Edit3->Text=FormatFloat("0.000", max)+
" індекс "+IntToStr(ind);
Memo2->Clear();
for(i=0; i<n; i++)
Memo2->Lines->Add(A[i]);
}
/*-------------------------------------------------------------*/
```

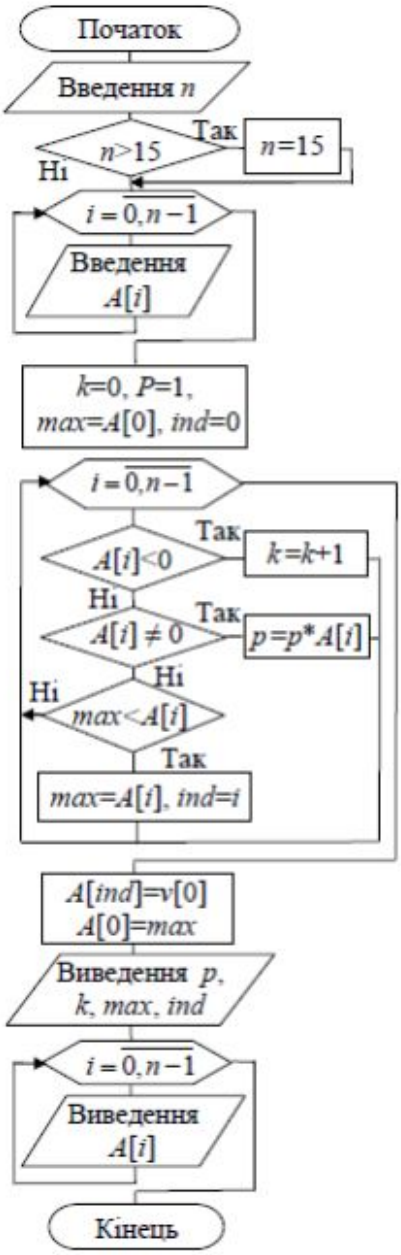
Програмування базових алгоритмів опрацювання двовимірних масивів

Приклад 5.3. Скласти схему алгоритму і розробити проект для введення матриці дійсних чисел розмірністю 3×5 і віднайти максимальний елемент матриці та його індекси.

Розв'язок. Програмний код головного файлу, який реалізує інтерфейс додатку (див. рис.5.3) та його функціонал має такий вигляд

```
/*- Unit01.cpp ------------------------------------------------*/
// Створення форми
void __fastcall TForm1::FormCreate(TObject *Sender)
{ for(int i=1; i<=3; i++)
SG1->Cells[0][i]=IntToStr(i)+"-й рядок";
for(int j=1; j<=5; j++)
SG1->Cells[j][0] = IntToStr(j)+"-й стовпчик";
}
/*-----------------------------------------------------------------*/
// Максимальний елемент та його індекси
void __fastcall TForm1::Button1Click(TObject *Sender)
{ float A[3][5], max; int i, j, ind_i, ind_j;
for (i=0; i<3; i++)
for (j=0; j<5; j++)
if (SG1->Cells[j+1][i+1] != "")
A[i][j] = StrToFloat(SG1->Cells[j+1][i+1]);
else
{ ShowMessage("Введіть [" + IntToStr(i+1) + "," +
IntToStr(j+1) + "] елемент");
break; }
max = A[0][0]; ind_i = 0; ind_j = 0;
for (i=0; i<3; i++)
for (j=0; j<5; j++)
if (max < A[i][j]) { max = A[i][j]; ind_i = i; ind_j = j; }
Edit1->Text = FormatFloat("0.00",max) + " " + IntToStr(ind_i+1)
+ "-й рядок і " + IntToStr(ind_j + 1) + "-й стовпчик";
}
/*-----------------------------------------------------------------*/
void __fastcall TForm1::Button2Click(TObject *Sender) // Очищення
{ for (int i=0; i<3; i++)
for (int j=0; j<5; j++) SG1->Cells[j+1][i+1] = "";
Edit1->Clear();
```

```
}
/*-----------------------------------------------------------------*/
void __fastcall TForm1::Button3Click(TObject *Sender)
// Вихід
{
Close();
}
/*-----------------------------------------------------------------*/
```



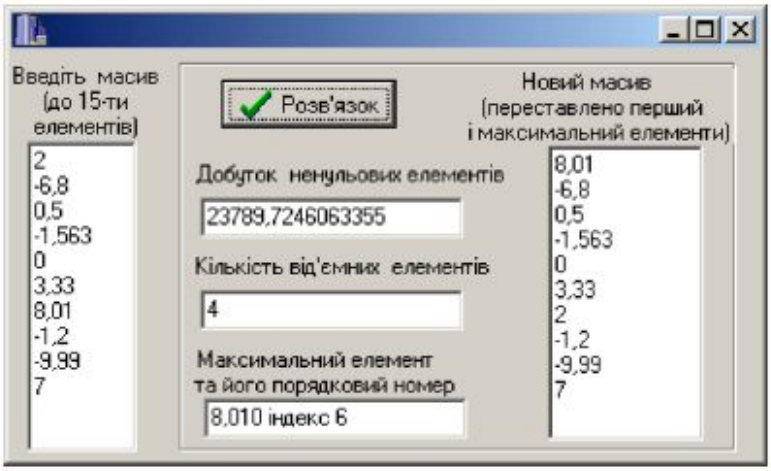


Рис. 5.2 — Блок-схема алгоритму та екранна форма додатку для опрацювання одновимірного масиву

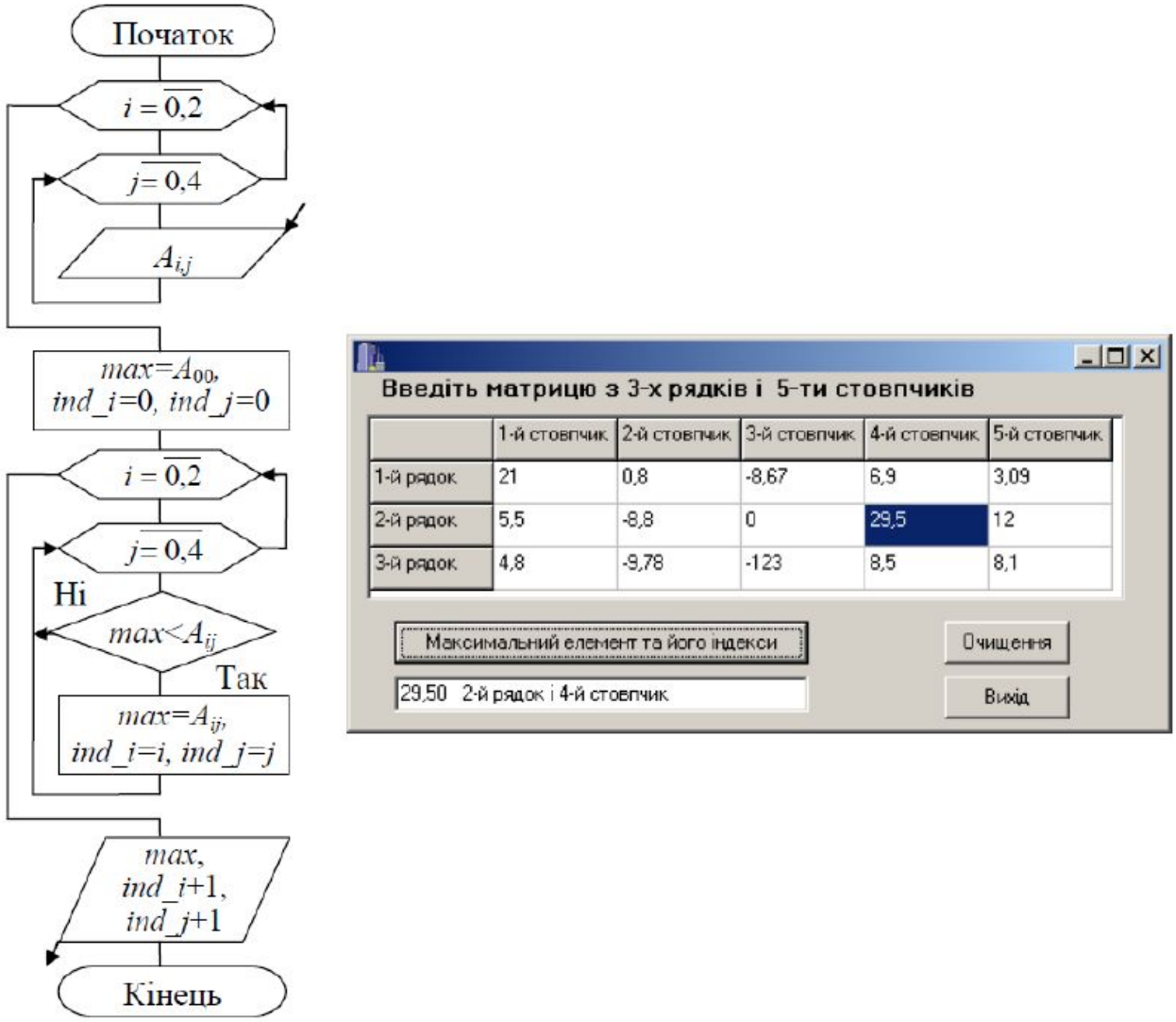


Рис. 5.3 — Блок-схема алгоритму та екранна форма додатку для пошуку максимального елементу матриці та його індексів

Приклад 5.4. Скласти схему алгоритму і розробити проект для введення матриці цілих чисел розмірністю 5×7 та обчислення елементів вектора сум від'ємних елементів стовпчиків матриці.

Розв'язок. Для заповнення вихідної матриці слід передбачити можливість заповнення комірок випадковими числами. Користувач може скористатись такою можливістю чи то заповнити комірки самостійно. Випадкові числа формуємо за допомогою функції генератора випадкових чисел **random(N)**, яка міститься в бібліотеці <**stdlib.h**>. Функція **random(N)** генерує випадкове ціле додатне число в діапазоні від 0 до $N-1$. Щоб формувались і від'ємні числа, використаємо конструкцію **(50-random(100))**. Функція **randomize()** ініціалізує генератор випадкових чисел.

Масив A та змінні i, j оголошено перед усіма функціями глобально, оскільки посилання до цих об'єктів є в різних функціях.

Програмний код головного файлу, який реалізує інтерфейс додатку (див.

рис.5.4) та його функціонал має такий вигляд

```
int A[5][7], i, j;
void __fastcall TForm1::Button1Click(TObject *Sender)
{ randomize(); // Заповнити випадковими числами
for (i=0; i<5; i++)
for (j=0; j<7; j++) SG1->Cells[j][i]=IntToStr(50-random(100));
}
//-------------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)//Розв"язок
{ int X[7]; Memo1->Clear();
for (i=0; i<5; i++)
for (j=0; j<7; j++) A[i][j] = StrToInt(SG1->Cells[j][i]);
for (j=0; j<7; j++)
{ X[j] = 0;
for(i=0; i<5; i++) if(A[i][j] < 0) X[j] += A[i][j];
Memo1->Lines->Add(IntToStr(X[j]));
} }
//-------------------------------------------------------------
void __fastcall TForm1::Button3Click(TObject *Sender)//Очищення
{ Memo1->Clear();
for (i=0; i<5; i++)
for (j=0; j<7; j++) SG1->Cells[j][i] = "";
}
//-------------------------------------------------
void __fastcall TForm1::Button4Click(TObject *Sender)
{ Close(); // Вихід - закриття форми
}
```

5.2. Завдання для аудиторної роботи

1. Скласти блок-схему алгоритму і написати програмний код для обчислення елементів одновимірного масиву з 10-ти елементів за формулою

$$A_i = \mathrm{ctg}(i) + i\cos(2^i), \quad i = 1, ..., 10 \tag{5.2}$$

та їхнього виведення на форму, а також визначення максимального елемента та його добутку на порядковий номер мінімального елементу.

2. Скласти блок-схему алгоритму і написати програмний код для введення одновимірного масиву з 12-ти елементів та обчислення:

- добутку позитивних елементів;

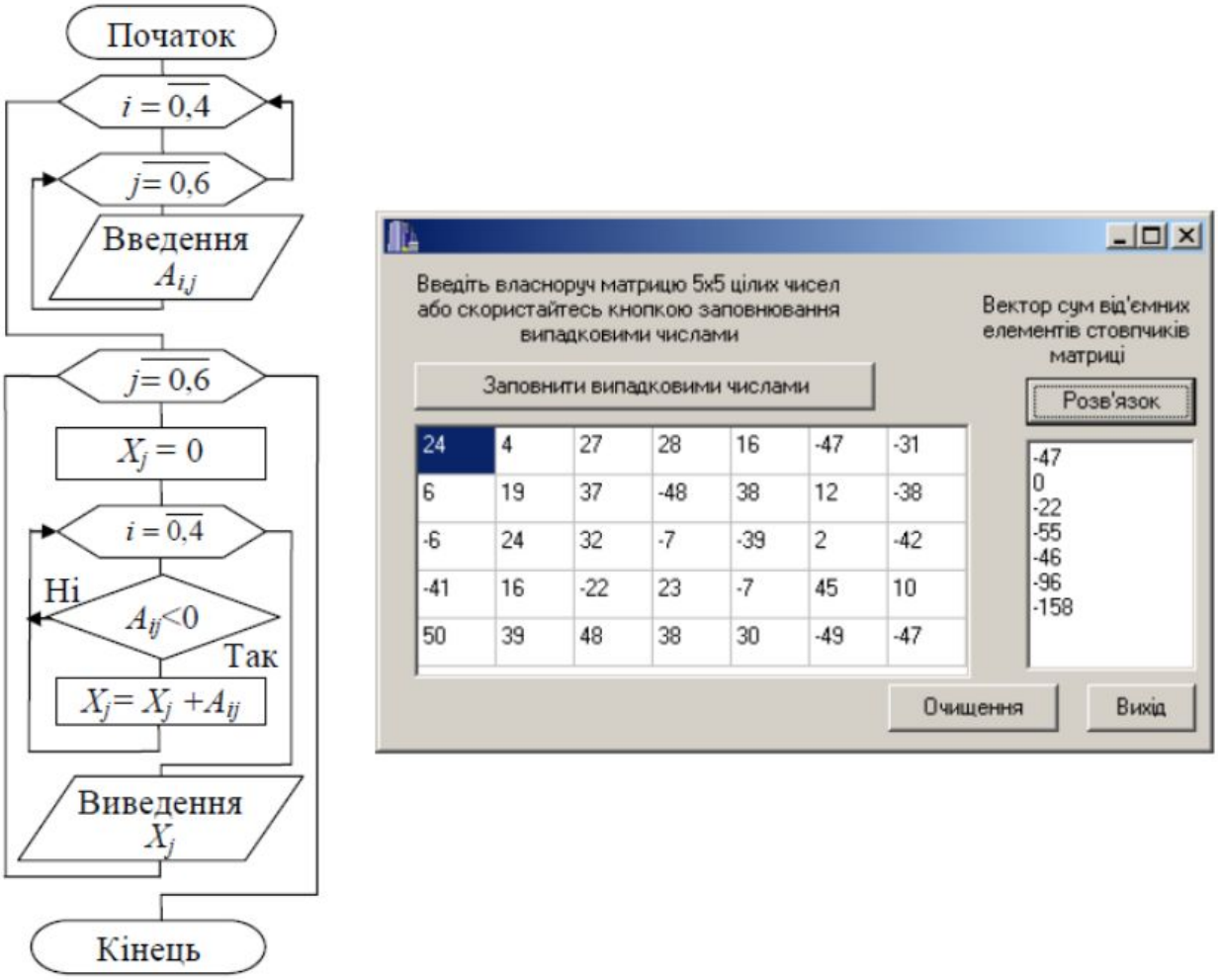


Рис. 5.4 — Блок-схема алгоритму та екранна форма додатку для обчислення елементів вектора сум від'ємних елементів стовпчиків матриці

- кількості від'ємних елементів;
- добутку максимального елемента на кількість позитивних елементів.

3. Скласти блок-схему алгоритму і написати програмний код для введення матриці дійсних чисел розмірністю 4×6 і віднайти мінімальний та максимальний елемент матриці та їх індекси.

4. Скласти блок-схему алгоритму і написати програмний код для введення матриці цілих чисел розмірністю 4×6 та обчислення елементів вектора сум позитивних елементів стовпчиків матриці.

5.3. Завдання для самостійної роботи

Вивчити основи технології програмування [5], [6] одновимірних та двовимірних масивів в середовищі Borland C++ Builder Enterprise v6.0 з використанням компонента TMemo та TStringGrid.

Уважно проаналізувати зміст прикладів теоретичної частини комп'ютерного практикуму №5 та на їх основі реалізувати програмні коди, які були підготвлені під час виконання аудиторних завдань у вигляді запрограмованих додатків у середовищі Borland C++ Builder Enterprise v6.0.

Передати викладачу на перевірку звіт про виконане завдання із зображенням блок-схем алгоритмів, лістінгів програмного коду та відкомпільованих виконавчих файлів запрограмованих додатків. Термін виконання завдання для самостійної роботи 2 тижні.

5.4. Варіанти завдань для аудиторної та домашньої роботи студентів

1. Скласти блок-схему алгоритму і написати програмний код для обчислення елементів одновимірного масиву з 20-ти елементів за формулою

$$A_i = \exp(i) + i\,\mathrm{tg}(2^i), \quad i = 1, ..., 20 \tag{5.3}$$

   та їхнього виведення на форму, а також визначення максимального елемента та його добутку на порядковий номер мінімального елементу.

2. Скласти блок-схему алгоритму і написати програмний код для введення одновимірного масиву з 8-ми елементів та обчислення:

   - добутку від'ємних елементів;
   - кількості позитивних елементів;
   - добутку мінімального елемента на кількість позитивних елементів.

3. Скласти блок-схему алгоритму і написати програмний код для введення матриці дійсних чисел розмірністю 6×5 і віднайти мінімальний та максимальний елемент матриці та їх індекси.

4. Скласти блок-схему алгоритму і написати програмний код для введення матриці цілих чисел розмірністю 6×5 та обчислення елементів вектора сум позитивних елементів стовпчиків матриці.

5. Скласти блок-схему алгоритму і написати програмний код для обчислення елементів одновимірного масиву з 15-ти елементів за формулою

$$A_k = \ln(k) + \cos(k\pi)\,\mathrm{tg}(2^k), \quad k = 1, ..., 15 \tag{5.4}$$

   та їхнього виведення на форму, а також визначення добутку максимального та мінімального елементів та їх порядкові номери.

6. Скласти блок-схему алгоритму і написати програмний код для введення одновимірного масиву з 14-ми елементів та обчислення:

- добутку квадратів від'ємних елементів;
- добутку квадратів позитивних елементів;
- добутку кількості від'ємних елемента на кількість позитивних елементів.

7. Скласти блок-схему алгоритму і написати програмний код для введення матриці дійсних чисел розмірністю 8×12 і віднайти мінімальний та максимальний елемент матриці, їх індекси та суму всіх позитивних елементів.

8. Скласти блок-схему алгоритму і написати програмний код для введення матриці цілих чисел розмірністю 12×7 та обчислення 12-вимірного вектора із сум рядків матриці.

9. Скласти блок-схему алгоритму і написати програмний код для обчислення елементів одновимірного масиву з 17-ти елементів за формулою

$$A_k = \frac{\sin(k\pi)}{1 + k^2}, \quad k = 1, ..., 17 \tag{5.5}$$

та їхнього виведення на форму, а також визначення добутку максимального та мінімального елементів та їх різницю.

10. Скласти блок-схему алгоритму і написати програмний код для введення одновимірного масиву з 18-ми елементів та обчислення:

- добутку кубів позитивних елементів;
- добутку кубів від'ємних елементів;
- добутку кількості кубів від'ємних елемента на кількість позитивних елементів.

11. Скласти блок-схему алгоритму і написати програмний код для введення матриці дійсних чисел розмірністю 14×18 і віднайти мінімальний та максимальний елемент матриці, їх індекси та середнє значення всіх позитивних елементів.

12. Скласти блок-схему алгоритму і написати програмний код для введення матриці цілих чисел розмірністю 4×9 та 9 вимірний вектор із сум елементів у колоноках матриці.

13. Скласти блок-схему алгоритму і написати програмний код для обчислення елементів одновимірного масиву з 13-ти елементів за формулою

$$A_k = \frac{\mathrm{tg}(k\pi)}{2 + \cos(k\pi)} + k^2, \quad k = 1, ..., 13 \tag{5.6}$$

та їхнього виведення на форму, а також визначення добутку максимального та мінімального елементів та їх середнє значення.

14. Скласти блок-схему алгоритму і написати програмний код для введення одновимірного масиву з 15-ми елементів та обчислення:
    - добутку позитивних елементів в 4-ій степені;
    - добутку від'ємних елементів в 4-ій степені;
    - добутку кількості від'ємних елементів в 4-ій степені на кількість позитивних елементів.
15. Скласти блок-схему алгоритму і написати програмний код для введення матриці дійсних чисел розмірністю 15×17 і віднайти мінімальний та максимальний елемент матриці, їх індекси та середнє значення всіх від'ємних елементів.
16. Скласти блок-схему алгоритму і написати програмний код для введення матриці цілих чисел розмірністю 7×5 та обчислення 7-вимірного вектора із сум елементів у рядках матриці.
17. Скласти блок-схему алгоритму і написати програмний код для обчислення елементів одновимірного масиву з 13-ти елементів за формулою

$$A_k = \frac{\cos(k\pi)}{2 + \cos^2(k\pi)} + k^3, \quad k = 1, ..., 13 \tag{5.7}$$

    та їхнього виведення на форму, а також визначення добутку максимального та мінімального елементів, їх середнє значення та кількість від'ємних елементів.
18. Скласти блок-схему алгоритму і написати програмний код для введення одновимірного масиву з 19-ми елементів та обчислення:
    - добутку позитивних елементів в 6-ій степені;
    - добутку від'ємних елементів в 6-ій степені;
    - добутку кількості від'ємних елементів в 6-ій степені на кількість від'ємних елементів.
19. Скласти блок-схему алгоритму і написати програмний код для введення матриці дійсних чисел розмірністю 8×15 і віднайти мінімальний та максимальний елемент матриці, їх індекси та середнє значення всіх позитивних елементів.
20. Скласти блок-схему алгоритму і написати програмний код для введення матриці цілих чисел розмірністю 13×12 та обчислення 12-вимірного вектора із сум елементів у солонках матриці.

21. Скласти блок-схему алгоритму і написати програмний код для обчислення елементів одновимірного масиву з 15-ти елементів за формулою

$$A_k = \frac{\mathrm{tg}(k\pi)}{2 + \sin^2(k\pi)} + \cos(k^3), \quad k = 1, ..., 15 \tag{5.8}$$

та їхнього виведення на форму, а також визначення добутку максимального та мінімального елементів, їх середнє значення та кількість позитивних елементів.

22. Скласти блок-схему алгоритму і написати програмний код для введення одновимірного масиву з 23-ми елементів та обчислення:

    - добутку позитивних елементів в 3-ій степені;
    - добутку від'ємних елементів в 3-ій степені;
    - добутку кількості позитивних елементів в 3-ій степені на кількість від'ємних елементів.

23. Скласти блок-схему алгоритму і написати програмний код для введення матриці дійсних чисел розмірністю 3×4 і віднайти мінімальний та максимальний елемент матриці, їх індекси та середнє значення всіх від'ємних елементів.

24. Скласти блок-схему алгоритму і написати програмний код для введення матриці цілих чисел розмірністю 9×8 та обчислення 9-вимірного вектора із сум елементів у рядках матриці.

25. Скласти блок-схему алгоритму і написати програмний код для обчислення елементів одновимірного масиву з 15-ти елементів за формулою

$$A_k = \frac{\mathrm{ctg}(k\pi)}{2 + \sin^3(k\pi)} + \ln(k^2), \quad k = 1, ..., 15 \tag{5.9}$$

та їхнього виведення на форму, а також визначення добутку максимального та мінімального елементів, їх середнє значення та кількість від'ємних елементів.

5.5. Питання для самоконтролю знань

- Дайте означення масиву у програмуванні?
- В який спосіб задають розмірність масиву?
- Яке призначення індексів масиву? Змінні яких типів можна використовувати для індексів? В який спосіб записують індекси масиву?

- Які з наведених оголошень одновимірного масиву з 10-ти елементів

  — int A[1..10],

  — int A[9],

  — float A[10],

  — float A[0,9]

  помилкові і чому?

- Запишіть оголошення одновимірного масиву з 25-ти дійсних чисел;

- Наведіть оператор, який присвоює першому елементу масиву A з 20-ти цілих чисел нульове значення;

- Запишіть фрагмент програми для обчислення середнього арифметичного, максимального і мінімального елементів масиву Z з 15-ти цілих чисел;

- Запишіть оголошення одновимірного констант-масиву з послідовності символів Вашого прізвища;

- Які компоненти **C++ Builder** для введення елементів одновимірних масивів на форму Ви знаєте?

6. Комп'ютерний практикум №6. Алгоритмізація та програмування опрацювання числових одновимірних та двовимірних динамічних масивів з використанням показчиків

Мета практикуму. Набуття досвіду побудови базових алгоритмів для опрацювання числових динамічних масивів та їх програмної реалізації з використанням показчиків.

6.1. Теоретичні відомості та рекомендації

Вказівники. Динамічна пам'ять.

Пам'ять комп'ютера являє собою сукупність комірок для зберігання інформації, кожна з яких має власний номер. Ці номери називаються адресами. Розмір однієї комірки пам'яті становить 1 байт. При оголошенні змінної в пам'яті виділяється ділянка обсягом, відповідним до типу змінної, наприклад 4 байти для типу int. Ця ділянка пам'яті пов'язується з іменем змінної. Можна

сказати, що адреса змінної — це адреса першої комірки цієї ділянки. Існують спеціальні змінні — вказівники, в яких можна зберігати адреси комірок пам'яті, тобто адреси змінних.

Адреса і значення змінної це різні поняття. Зазвичай адреси змінних не є важливими, найголовніше – значення цих змінних. Вказівники використовуються для прямого доступу до даних у пам'яті.

Вказівник оголошується за допомогою *:

<тип> *<ім'я>;

Тут тип — це базовий тип вказівника, котрим може бути який завгодно тип. Ім'я є ідентифікатором змінної-вказівника. Слід звернути увагу на те, що тип — це не тип вказівника, а тип даних, адресу яких буде записано у вказівнику.

У мові C++ визначено дві операції для роботи з вказівниками:

- * – розадресація, чи розіменування вказівника;
- & – адресація, тобто отримання адреси змінної.

Для прикладу у програмі оголосимо цілу змінну **X** зі значенням **7**:

```
int X = 7;
```

тоді, щоб її адресу записати до вказівника A, слід записати команду:

```
A = &X; // Значенням змінної A є адреса змінної X.
```

До значення **7**, яке зберігається у змінній **X**, можна звернутися не лише за ім'ям X, але й через вказівник. При цьому слід застосувати операцію розадресації вказівника (вона позначається символом '*'), наприклад: ***A**. Цей запис означає: 'значення, яке зберігається за адресою, записаною у змінній A'. Щоб збільшити на **2** оголошену раніше змінну **X**, можна записати так:

```
X = X + 2; /* За ім'ям X комп'ютер визначить адресу
ділянки пам'яті, в якій вона розташована, візьме значення,
що зберігається у пам'яті за цією адресою, збільшить
його на 2 і запише назад у ту ж саму ділянку пам'яті */
```

чи так:

```
*A = (*A) + 2; /* Комп'ютер з ділянки пам'яті за адресою,
записаною у змінній A, візьме значення, яке зберігається
в ній, збільшить це значення на 2 і запише в ту ж саму ділянку */
```

У першому разі відбувається звертання до змінної з ім'ям **X**, у другому — до змінної через її адресу, яку записано у змінній-вказівнику **A**.

Вказівники на одновимірні масиви

Масиви і вказівники у C++ тісно пов'язані і можуть використовуватись майже еквівалентно. Ім'я масиву можна сприймати як константний вказівник на перший елемент масиву. Його відмінність від звичайного вказівника полягає у тому, що його неможна модифікувати.

Для прикладу здійснимо оголошення масиву **mass** з п'яти цілих чисел з ініціалізацією значень елементів і вказівника на ціле число **ptr**:

```
int mass[5] = {10,-2,0,40,3}, *ptr;
```

При такому оголошенні масиву пам'ять виділяється не лише для п'яти елементів масиву, а й для вказівника з ім'ям **mass**, значення якого дорівнює адресі першого елемента масиву **mass[0]**, тобто сам масив залишається безіменним, а доступ до елементів масиву здійснюється через вказівник з ім'ям **mass**.

Для того, щоби звернутися до 3-го елементу цього масиву, можна записати чи то **mass[2]** чи ***(mass+2)**. При реалізації на комп'ютері перший з цих способів зводиться до другого, тобто індексний вираз перетвориться до адресного.

Оскільки ім'я масиву є вказівником на перший елемент масиву, можна надати вказівнику адресу першого елементу масиву за допомогою оператора:

```
ptr = mass;
```

Цей запис є еквівалентним присвоюванню адреси першого елемента масиву (тобто до елемента з нульовим індексом)у вигляді оператора

```
ptr = &mass[0];
```

Цей запис є еквівалентним присвоюванню адреси першого елемента масиву (тобто до елемента з нульовим індексом)у вигляді оператора

```
ptr = &mass[0];
```

Після цього звернутися до першого елемента масиву і надати йому значення **2** можна будь-яким з шести операторів:

```
* mass = 2; mass[0] = 2; *(mass+0) = 2;
* ptr = 2; ptr[0] = 2; *(ptr+0) = 2;
```

Усі ці оператори за дією є тотожними, але швидше за всі виконуватимуться присвоювання ***mass = 2** та ***ptr = 2**, оскільки в них не треба виконувати операції додавання.

Вказівники можна індексувати так само, як і масиви. Наприклад, вираз **ptr[3]** посилається до четвертого елемента масиву **mass[3]**.

Приклад 6.1. Запрограмувати додаток (див. рис.6.1) функціонал якого передбачає введення масиву **A** з 10-ти цілих чисел і створення з нього два нових нових підмасивів: перший підмасив **B** з непарних елементів, а другий **C** – з парних та їх візуалізацію.

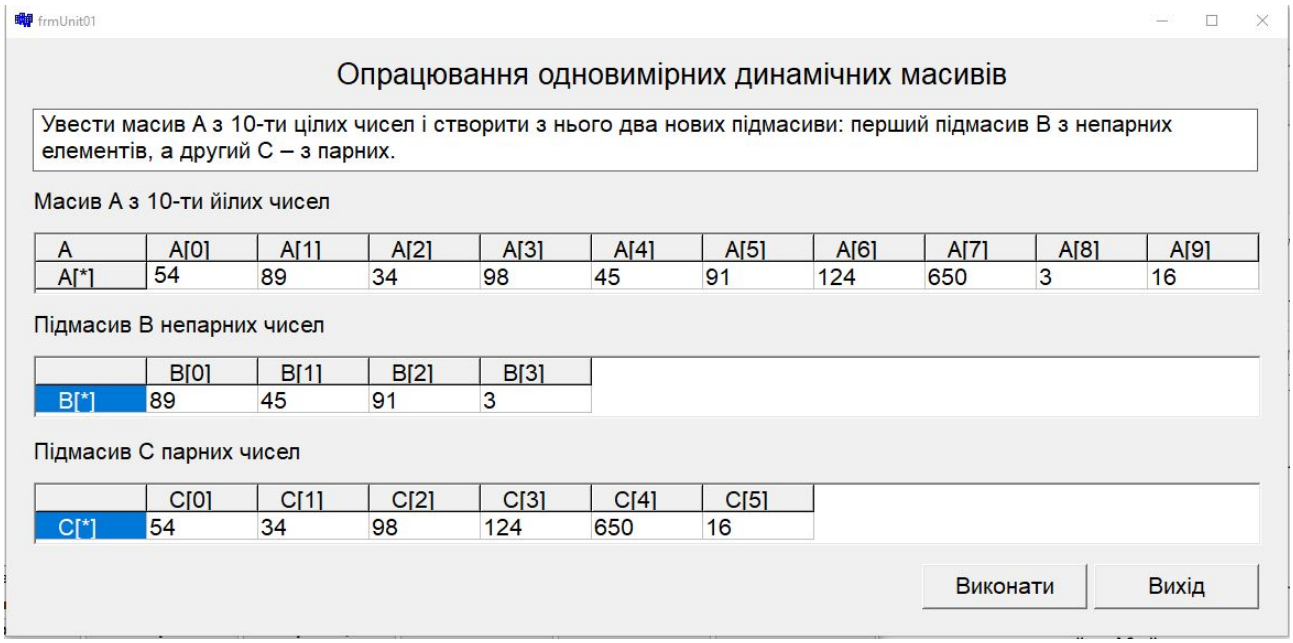


Рис. 6.1 — Екранна форма додатку для опрацювання одновимірних динамічних масивів

Для побудови графічного інтерфейсу додатку на головній формі проекту (див. рис.6.1), розташовати колекція компонентів

$$\begin{cases} TForm - 1 \text{ шт.}; \quad TPanel - 1 \text{ шт.}; \\ TMemo - 1 \text{ шт.}; \quad TLabel - 3 \text{ шт.}; \\ TStringGrid - 3 \text{ шт.}; \quad TButton - 2 \text{ шт.} \end{cases} \tag{6.1}$$

Оскільки кількість елементів вихідного масиву **A** є відома, тому цей масив у програмі доцільно зробити оголошеним у звичайний спосіб: **int A[10];**. Два нові масиви **B** та **C** доцільно створити динамічно, оскільки кількість елементів цих масивів **n1** та **n2** буде обчислено у програмі

Приклад програмного коду в головному файлі додатку має такий вигляд

```
/*- Unit01.cpp ------------------------------------------------------*/
#include <vcl.h>
#pragma hdrstop
```

```
#include "Unit01.h"
/*---------------------------------------------------------------*/
#pragma package(smart_init)
#pragma resource "*.dfm"
TfrmUnit01 *frmUnit01;
int A[10], i, n1=0, n2=0, j1=0, j2=0;
/*---------------------------------------------------------------*/
__fastcall TfrmUnit01::TfrmUnit01(TComponent* Owner)
        : TForm(Owner)
{}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::FormCreate(TObject *Sender)
{
        StringGrid0->Cells[0][0]= "   A";
        for (i=0; i<10; i++) {
        StringGrid0->Cells[i+1][0]= "   A["+IntToStr(i)+"]";
        StringGrid0->Cells[i+1][1] = "0"; }
        StringGrid0->Cells[0][1] = "   A[*]";
        StringGrid1->Cells[0][1] = "   B[*]";
        StringGrid2->Cells[0][1] = "   C[*]";
}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button0Click(TObject *Sender)
{
        for(i=0; i<10; i++) { A[i] = 0; }
        for(i=0; i<10; i++) {
                A[i] = StrToInt(StringGrid0->Cells[i+1][1]);
        }
        for (i=0; i<10; i++) {
                if (A[i]%2 != 0) {n1++;}
// Обчислення кількості непарних елементів n1
                else {if (A[i] > 0) {n2++;}}
// та кількості парних елементів n2.
        }
        StringGrid1->ColCount=n1+1;
        StringGrid2->ColCount=n2+1;
// Встановлення потрібної кількості комірок
// у компонентах для виведення створюваних масивів.
        int *B = new int [n1];
        int *C = new int [n2];
        for (i=0; i<10; i++)
```

```
// Формування масивів B та C
        if(A[i]%2 != 0) { B[j1] = A[i]; j1++; }
        else { C[j2]=A[i]; j2++; }
// Виведення масивів в окремих циклах,
// оскільки масиви мають різну кількість елементів
        for(i=0; i<n1; i++) {
           StringGrid1->Cells[i+1][0] =
           "    B["+IntToStr(i)+"]";
           StringGrid1->Cells[i+1][1] = IntToStr(B[i]);
        }
        for(i=0; i<n2; i++) {
                        StringGrid2->Cells[i+1][0]
                        = "    C["+IntToStr(i)+"]";
            if (C[i] > 0)
               StringGrid2->Cells[i+1][1] = IntToStr(C[i]);
        }
        n1 = 0; j1 = 0; n2 = 0; j2 = 0;
// Звільнення пам'яті від B та C
        delete []B;
        delete []C;
}
/*-----------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button1Click(TObject *Sender)
{
        Close();
}
/*-----------------------------------------------------------------*/
```

Динамічні двовимірні масиви (матриці)

Двовимірний динамічний масив з **m** рядків і **n** стовпчиків займає в пам'яті сусідні **m*n** комірок, тобто зберігається так само, як і одновимірний масив з **m*n** елементів. При розміщенні елементи двовимірних масивів розташовуються в пам'яті підряд один за одним з першого до останнього без проміжків у послідовно зростаючих адресах пам'яті.

У такому масиві перші п'ять елементів належать до першого рядка, наступні п'ять — до другого і останні п'ять — до третього рядка.

Оголосити дійсний динамічний масив 3×5 можна як одновимірний з 15-ти елементів:

```
float *a=new float [3*5];
```

чи то

```
float *a=(float *)calloc(3*5, sizeof(float));
```

Пам'ять від створеного в такий спосіб масиву очищується за допомогою операцій відповідно **delete** й **free**:

```
delete []a;
```

чи то

```
free(a);
```

Мождивий інший спосіб роботи з динамічним двовимірним масивом. Для цього розмістимо в пам'яті машини матрицю 3×5. При цьому буде виділено пам'ять під кожний рядок матриці окремо, тобто буде утворено три різні одновимірні масиви. Адреси нульових елементів цих масивів зберігатимуться в допоміжному масиві a (пам'ять під нього слід виділити заздалегідь). Елементами цього масиву будуть адреси дійсних чисел, тому вони матимуть тип "вказівник на дійсне число", тобто **float***.

Загальний вигляд оголошення вказівника на динамічний масив є такий:

```
<тип елемента> * <ім'я масиву>;
```

Тоді при оголошенні масиву a слід записати дві зірочки:

```
float **a = new float*[3];
// Оголошення й розміщення в пам'яті допоміжного масиву
// з 3-х елементів типу float*
for(int i=0; i<3; i++) a[i] = new float [5];
// У циклі виділяється пам'ять під 3 масиви по 5 елементів
// (рядки матриці) й адреси нульових елементів цих масивів
// записуються у відповідні елементи масиву a
```

Після цього можна працювати з матрицею як зі звичайним двовимірним масивом, звертаючись до кожного елемента за його індексом, наведеним у квадратних дужках: **a[i][j]**, що є більш природним і зручним, аніж попередній спосіб.

Аналогічне є оголошення за допомогою calloc():

```
float ** a = (float**) calloc (3, sizeof(float*));
for (int i=0; i<3; i++)
a[i] = (float*) calloc (5, sizeof(float));
```

Приклад 6.2. Запрограмувати додаток (див. рис.6.2) функціонал якого передбачає визначення на початку роботи програми кількості рядків і стовпчиків матриці **A** та чилових елементів самої матриці, обчислення добутку її додатних елементів.

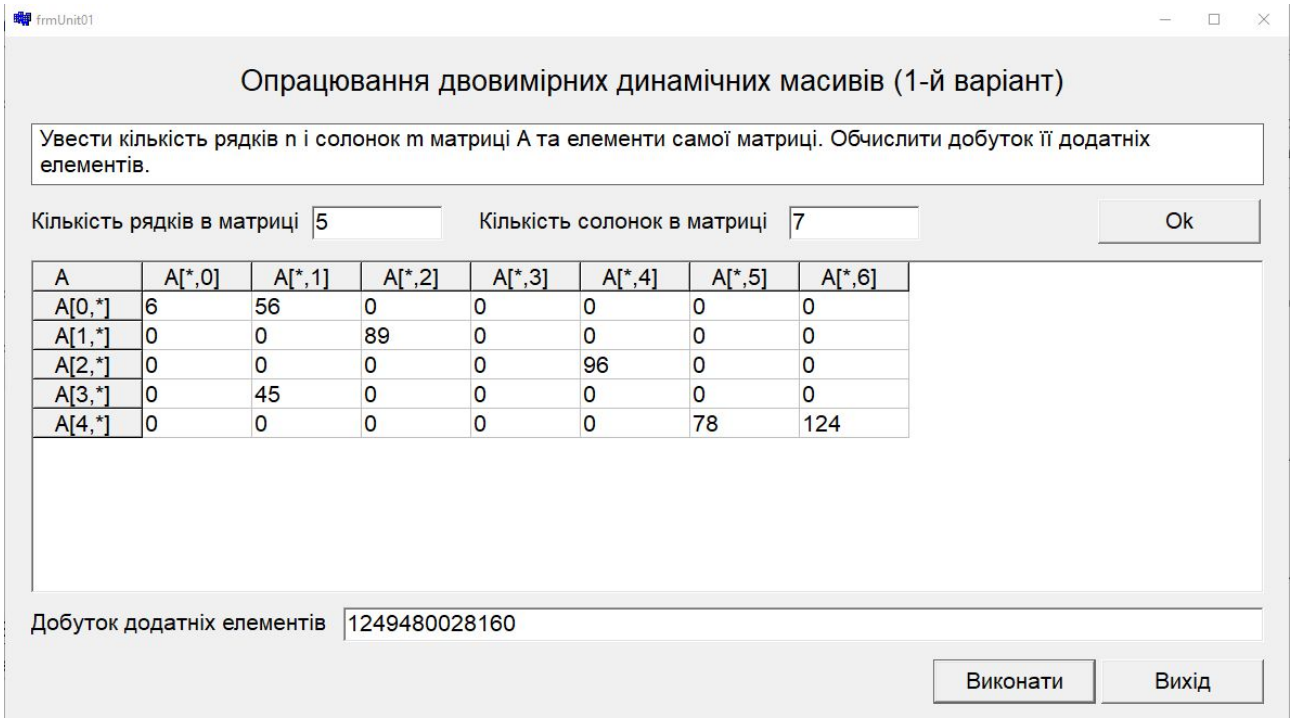


Рис. 6.2 — Екранна форма додатку для обчислення добутку додатніх елементів двохвимірного динамічного масиву

Для побудови графічного інтерфейсу додатку на головній формі проекту (див. рис.6.2), розташувати колекцію компонентів

$$\begin{cases} TForm - 1 \text{ шт.}; \quad TPanel - 1 \text{ шт.}; \\ TMemo - 1 \text{ шт.}; \quad TLabel - 3 \text{ шт.}; \\ TEdit - 3 \text{ шт.}; \quad TStringGrid - 1 \text{ шт.}; \\ TButton - 3 \text{ шт.} \end{cases} \tag{6.2}$$

Оскільки кількість рядків та стовпчиків матриці заздалегідь є невідомі, тут матрицю доцільно організувати динамічно.

Приклади двох способів виконання завдання наведено нижче.

Перший спосіб. Пам'ять виділяється під кожний рядок матриці окремо. При роботі з елементами матриці явно зазначаються обидва індекси (як при

роботі з елементами нединамічної матриці), що вважається більш зручним.

Приклад програмного коду має такий вигляд

```
/*- Unit01.cpp ------------------------------------------------------*/
#include <vcl.h>
#pragma hdrstop
#include "Unit01.h"
/*------------------------------------------------------------------*/
#pragma package(smart_init)
#pragma resource "*.dfm"
TfrmUnit01 *frmUnit01;
int n,m,i,j;
double p = 1;
/*------------------------------------------------------------------*/
__fastcall TfrmUnit01::TfrmUnit01(TComponent* Owner)
        : TForm(Owner)
{}
/*------------------------------------------------------------------*/
void __fastcall TfrmUnit01::FormCreate(TObject *Sender)
{
StringGrid0->Cells[0][0] = "   A";
StringGrid0->Cells[0][1] = "   A[0,*]";
StringGrid0->Cells[1][0] = "   A[*,0]";
StringGrid0->Cells[1][1] = "0";
}
/*------------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button0Click(TObject *Sender)
{
// Введення розмірності матриці
        m = StrToInt(Edit0->Text);
        n = StrToInt(Edit1->Text);
        StringGrid0->RowCount = m+1;
        StringGrid0->ColCount = n+1;
        for(int i=1; i<=m; i++) {
          StringGrid0->Cells[0][i] = "
          A["+IntToStr(i-1)+",*]";
        }
        for(int j=1; j<=n; j++) {
          StringGrid0->Cells[j][0] = "
          A[*,"+IntToStr(j-1)+"]";
        }
        for(int i=1; i<=m; i++) {
```

```
        for(int j=1; j<=n; j++) {
                        StringGrid0->Cells[j][i] = "0";
                        p = 1;
                        Edit2->Text = "0";
        }}
}
/*————————————————————————————————————————————————————————*/
void __fastcall TfrmUnit01::Button1Click(TObject *Sender)
{
try {
        p = 1;
        Edit2->Text = "0";
        float **A = new float* [m];
// Виділення пам'яті під допоміжний масив
        for (int i=0; i<m; i++) {
                A[i] = new float[n];
// Виділення пам'яті під кожний рядок матриці окремо
        }
        for (int i=0; i<m; i++) {
        for (int j=0; j<n; j++) {
           A[i][j] =
           StrToFloat(StringGrid0->Cells[j+1][i+1]);
           if (A[i][j] > 0) {p *= A[i][j];
// Обчислення добутку додатних елементів
        }
        }}
        Edit2->Text = FloatToStr(p);
        for (int i=0; i<m; i++) {
                delete []A[i];
// Очищення пам'яті від рядків матриці
        }
        delete []A;
// Очищення пам'яті від допоміжного масиву
}
catch (...)
        {ShowMessage("Помилка у виконанні програми!");}
}
/*————————————————————————————————————————————————————————*/
void __fastcall TfrmUnit01::Button2Click(TObject *Sender)
{
        Close();
```

```
}
/*---------------------------------------------------------------------*/
```

Другий спосіб. Тут змінюється лише код **Button1Click**.

Приклад програмного коду має такий вигляд

```
/*-- Unit01.cpp ---------------------------------------------------------*/
#include <vcl.h>
#pragma hdrstop
#include "Unit01.h"
/*---------------------------------------------------------------------*/
#pragma package(smart_init)
#pragma resource "*.dfm"
TfrmUnit01 *frmUnit01;
int n,m,i,j;
double p = 1;
/*---------------------------------------------------------------------*/
__fastcall TfrmUnit01::TfrmUnit01(TComponent* Owner)
        : TForm(Owner)
{}
/*---------------------------------------------------------------------*/
void __fastcall TfrmUnit01::FormCreate(TObject *Sender)
{
StringGrid0->Cells[0][0] = "   A";
StringGrid0->Cells[0][1] = "   A[0,*]";
StringGrid0->Cells[1][0] = "   A[*,0]";
StringGrid0->Cells[1][1] = "0";
}
/*---------------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button0Click(TObject *Sender)
{
// Введення розмірності матриці
        m = StrToInt(Edit0->Text);
        n = StrToInt(Edit1->Text);
        StringGrid0->RowCount = m+1;
        StringGrid0->ColCount = n+1;
        for(int i=1; i<=m; i++) {
            StringGrid0->Cells[0][i] = "
            A["+IntToStr(i-1)+",*]";
        }
        for(int j=1; j<=n; j++) {
            StringGrid0->Cells[j][0] = "
```

```
            A[*,"+IntToStr(j-1)+"]";
        }
        for(int i=1; i<=m; i++) {
        for(int j=1; j<=n; j++) {
                        StringGrid0->Cells[j][i] = "0";
                        p = 1;
                        Edit2->Text = "0";
        }}
}
/*-----------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button1Click(TObject *Sender)
{
try {
        double p=1;
        float *A = new float [m*n];
// Виділення пам'яті під увесь масив одразу;
// Інакше це можна записати так:
// float *A = (float*)calloc(m*n, sizeof(float));
        for (int i=0; i<m; i++) {
        for (int j=0; j<n; j++) {
            *(A+i*n+j)=
            StrToFloat(StringGrid0->Cells[j+1][i+1]);
// При звертанні до елементів використовується покажчик
// і обчислюється індекс
                        if (*(A+i*n+j) > 0) p *= *(A+i*n+j);
// За допомогою операції індексації це можна записати так:
// if (A[i*n+j]>0) p *= A[i*n+j];
        }}
        Edit2->Text = FloatToStr(p);
        free(A);
}
catch (...)
        {ShowMessage("Помилка у виконанні програми!");}
}
/*-----------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button2Click(TObject *Sender)
{
        Close();
}
/*-----------------------------------------------------------------*/
```

6.2. Завдання для аудиторної роботи

1. Скласти блок-схему алгоритму і написати програмний код для введення масиву **A** з 15-ти цілих чисел і створення з нього два нових нових підмасивів: перший підмасив **B** з непарних додатніх елементів, а другий **C** – з парних від'ємних елементів.

2. Скласти блок-схему алгоритму і написати програмний код для визначення на початку роботи програми кількості рядків і стовпчиків матриці **A** та числових елементів самої матриці, обчислення добутку її парних від'ємних елементів розділених на максимальне значення елементів матриці **A.**

6.3. Завдання для самостійної роботи

Вивчити основи технології програмування покажчиків та динамічних масивів [5], [6] в середовищі Borland C++ Builder Enterprise v6.0 та їх використання при побудові додатків.

Уважно проаналізувати зміст прикладів теоретичної частини комп'ютерного практикуму №6 та на їх основі реалізувати програмні коди, які були підготвлені під час виконання аудиторних завдань у вигляді запрограмованих додатків у середовищі Borland C++ Builder Enterprise v6.0.

Передати викладачу на перевірку звіт про виконане завдання із зображенням блок-схем алгоритмів, лістінгів програмного коду та відкомпільованих виконавчих файлів запрограмованих додатків. Термін виконання завдання для самостійної роботи 2 тижні.

6.4. Варіанти завдань для аудиторної та домашньої роботи студентів

1. Скласти блок-схему алгоритму і написати програмний код для введення масиву **A** з 17-ти цілих чисел і створення з нього два нових нових підмасивів: перший підмасив **B** з непарних додатніх елементів, а другий **C** – з парних від'ємних елементів.

2. Скласти блок-схему алгоритму і написати програмний код для визначення на початку роботи програми кількості рядків і стовпчиків матриці **A** та числових елементів самої матриці, обчислення добутку її непарних позитивних елементів розділених на мінімальне значення елементів матриці **A.**

3. Скласти блок-схему алгоритму і написати програмний код для введення масиву **A** з 13-ти цілих чисел і створення з нього два нових нових підма-

сивів: перший підмасив **B** з непарних від'ємних елементів, а другий **C** – з парних від'ємних елементів.

4. Скласти блок-схему алгоритму і написати програмний код для визначення на початку роботи програми кількості рядків і стовпчиків матриці **A** та числових елементів самої матриці, обчислення добутку її непарних позитивних елементів розділених на максимальне значення елементів матриці **A.**

5. Скласти блок-схему алгоритму і написати програмний код для введення масиву **A** з 9-ти цілих чисел і створення з нього два нових нових підмасивів: перший підмасив **B** з квадратів непарних від'ємних елементів, а другий **C** – з кубів парних від'ємних елементів.

6. Скласти блок-схему алгоритму і написати програмний код для визначення на початку роботи програми кількості рядків і стовпчиків матриці **A** та числових елементів самої матриці, обчислення добутку її непарних від'ємних елементів розділених на середнє значення елементів матриці **A.**

7. Скласти блок-схему алгоритму і написати програмний код для введення масиву **A** з 18-ти цілих чисел і створення з нього два нових нових підмасивів: перший підмасив **B** з кубів непарних від'ємних елементів, а другий **C** – з квадратів парних від'ємних елементів.

8. Скласти блок-схему алгоритму і написати програмний код для визначення на початку роботи програми кількості рядків і стовпчиків матриці **A** та числових елементів самої матриці, обчислення добутку її парних позитивних елементів розділених на середнє значення негативних елементів матриці **A.**

9. Скласти блок-схему алгоритму і написати програмний код для введення масиву **A** з 9-ти цілих чисел і створення з нього два нових нових підмасивів: перший підмасив **B** з непарних від'ємних елементів в 5-ій степені, а другий **C** – з парних позитивних елементів в 5-ій.

10. Скласти блок-схему алгоритму і написати програмний код для визначення на початку роботи програми кількості рядків і стовпчиків матриці **A** та числових елементів самої матриці, обчислення добутку середнього значення парних позитивних елементів розділених на середнє значення негативних елементів матриці **A.**

11. Скласти блок-схему алгоритму і написати програмний код для введення масиву **A** з 23-ти цілих чисел і створення з нього два нових нових підмасивів: перший підмасив **B** з парних позитивних елементів в 4-ій степені, а другий **C** – з парних від'ємних елементів в 4-ій степені.

12. Скласти блок-схему алгоритму і написати програмний код для визначення на початку роботи програми кількості рядків і стовпчиків матриці **A** та числових елементів самої матриці, обчислення добутку середнього значення непарних позитивних елементів розділених на середнє значення парних негативних елементів матриці **A.**

13. Скласти блок-схему алгоритму і написати програмний код для введення масиву **A** з 11-ти цілих чисел і створення з нього два нових нових підмасивів: перший підмасив **B** з парних позитивних елементів в 4-ій степені, які розділені на їх середнє значення, а другий **C** – з парних від'ємних елементів в 4-ій степені, які розділені на їх середнє значення.

14. Скласти блок-схему алгоритму і написати програмний код для визначення на початку роботи програми кількості рядків і стовпчиків матриці **A** та числових елементів самої матриці, обчислення ділення мінімального значення парних позитивних елементів на середнє значення парних позитивних елементів матриці **A.**

15. Скласти блок-схему алгоритму і написати програмний код для введення масиву **A** з 11-ти цілих чисел і створення з нього два нових нових підмасивів: перший підмасив **B** є транспонованою матрицею **A**, а другий **C** – складається з обернених єлементів матриці **A**.

16. Скласти блок-схему алгоритму і написати програмний код для визначення на початку роботи програми кількості рядків і стовпчиків матриці **A** та числових елементів самої матриці, обчислення суми діагональних елементів матриці **A.**

17. Скласти блок-схему алгоритму і написати програмний код для введення масиву **A** з 11-ти цілих чисел і створення з нього два нових нових підмасивів: перший підмасив **B** є транспонованою матрицею **A** помноженою на матрицю **A**, а другий **C** – є сумою транспонованою матрицею **A** та матриці **A**.

18. Скласти блок-схему алгоритму і написати програмний код для визначення на початку роботи програми кількості рядків і стовпчиків матриці **A** та числових елементів самої матриці, обчислення суми діагональних елементів матриці **A** помноженої на середнє значення елементів матриці **A.**

19. Скласти блок-схему алгоритму і написати програмний код для введення масиву **A** з 25-ти цілих чисел і створення з нього два нових нових підмасивів: перший підмасив **B** є квадратом матриці **A**, а другий **C** є кубом матриці **A**.

20. Скласти блок-схему алгоритму і написати програмний код для визначення на початку роботи програми кількості рядків і стовпчиків матриці **A** та числових елементів самої матриці, обчислення суми недіагональних елементів матриці **A** помноженої на максимальне значення елементів матриці **A.**

21. Скласти блок-схему алгоритму і написати програмний код для введення масиву **A** з 16-ти цілих чисел і створення з нього два нових нових підмасивів: перший підмасив **B** є сумою матриці **A** та одиничної матриці, а другий **C** є результатом множення матриці **A** на добуток діагональних елементів матриці **A**.

22. Скласти блок-схему алгоритму і написати програмний код для визначення на початку роботи програми кількості рядків і стовпчиків матриці **A** та числових елементів самої матриці, обчислення добутку недіагональних елементів матриці **A** помноженого на мінімальне значення елементів матриці **A.**

23. Скласти блок-схему алгоритму і написати програмний код для введення масиву **A** з 20-ти цілих чисел і створення з нього два нових нових підмасивів: перший підмасив **B** є сумою матриці **A** та діагональної матриці у якої в якості елементів виступає натуральний ряд чисел від 1 до 20, а другий **C** є результатом множення матриці **A** на добуток недіагональних елементів матриці **A**.

24. Скласти блок-схему алгоритму і написати програмний код для визначення на початку роботи програми кількості рядків і стовпчиків матриці **A** та числових елементів самої матриці, обчислення добутку недіагональних елементів матриці **A** помноженого на суму діагональних елементів **A.**

25. Скласти блок-схему алгоритму і написати програмний код для введення масиву **A** з 24-ти цілих чисел і створення з нього два нових нових підмасивів: перший підмасив **B** є сумою квадрата матриці **A** та діагональної матриці у якої в якості елементів виступає натуральний ряд чисел від 3 до 26, а другий **C** є результатом множення матриці **A** на добуток недіагональних елементів матриці **A**.

26. Скласти блок-схему алгоритму і написати програмний код для визначення на початку роботи програми кількості рядків і стовпчиків матриці **A** та числових елементів самої матриці, обчислення добутку недіагональних елементів квадрату матриці **A** помноженого на суму діагональних елементів кубу **A.**

6.5. Питання для самоконтролю знань

- Що являє собою пам'ять комп'ютера?
- Що таке адрес комірки?
- Що таке вказівник?
- Наведіть синтаксис оголошення вказівника?
- В який спосіб можна отримати адресу змінної?
- Яке призначення операції &?
- В який спосіб можна дізнатися значення, на яке посилається вказівник?
- Поясніть відмінності поміж змінними a та b, оголошення яких має вигляд:
  а) int a; int b;
  б) int *a; float *b;
- Як можна виділити пам'ять під цілу змінну?
- В який спосіб можна звільнити пам'ять, виділену за допомогою оператора new?
- Як можна ініціалізувати пам'ять, виділену за допомогою оператора new?
- В який спосіб можна звернутися до даних, для яких пам'ять було виділено динамічно?
- Поясніть відмінності динамічного масиву від звичайного.

7. Комп'ютерний практикум №7. Функції та правила їх алгоритмізації і програмування

Мета практикуму. Набуття досвіду побудови базових алгоритмів обробки числових даних з використанням різних варіантів організації функцій та їх програмної реалізації.

7.1. Теоретичні відомості та рекомендації

Функція — це незалежна іменована частина програми, яка може багаторазово викликатися з інших частин програми, маніпулювати даними та повертати результати. Кожна функція має власне ім'я, за яким здійснюють її виклик. Розрізнюють два основні різновиди функцій:

- стандартні вбудовані функції, які є складовою частиною мови програмування, наприклад: sin(), pow() тощо;
- функції, які створюються безпосередньо користувачем для власних потреб.

Створювана у програмі функція повинна бути оголошеною і визначеною. Оголошення функції має бути написаним у програмі раніш за її використання. Визначення може перебувати у будь-якому місці програми, за винятком тіла (середини) інших функцій. Оголошення функції складається з прототипу (чи заголовка) і має форму

```
[клас] <тип результату> <ім'я функції>
(<перелік формальних аргументів із зазначенням типу>);
```

Прототип функції може бути розташовано як безпосередньо у програмі, так і в заголовному файлі.

Окрім оголошення, кожна функція повинна мати визначення (реалізацію). Визначення функції, окрім заголовку, містить тіло функції (команди, які виконує функція) і команду повертання результату **return**:

```
<тип функції> <ім'я функції> (<перелік аргументів>)
{ <тіло функції>
return <результат обчислень>}
```

У разі, коли визначення функції розміщено у програмі раніш за точку виклику, писати окремо оголошення і окремо реалізацію цієї функції немає потреби і оголошення функції можна уникнути.

Якщо функції виконують певні обчислення й дії, які не потребують повертання результатів, за їхній тип вказують тип **void** (тобто порожній, без типу). У таких функціях оператор **return** може бути відсутнім чи записуватись без значення, яке повертається.

Виконання функції розпочинається, коли у тексті програми зустрічається оператор виклику, тобто ім'я функції з фактичними параметрами у дужках. При цьому формальні параметри послідовно набувають значень фактичних: значення першого фактичного аргументу копіюється у перший формальний аргумент, другий — у другий і т. д. А тому дуже важливими чинниками є кількість, порядок записування і відповідність типів всіх аргументів, інакше це може призвести до серії помилок.

Якщо функцію оголошено з будь-яким типом, окрім типу **void**, тобто вона повертає певний результат, при викликанні такої функції її варто присвоїти змінній того ж самого типу, що й сама функція, чи то вивести на екран за допомогою оператора виведення.

Правила організації функцій

- Фактичними параметрами можуть бути константи, змінні чи вирази. В останньому разі спочатку обчислюється значення виразу, а потім результат передається у функцію.
- Імена змінних, які є фактичними параметрами, можуть не збігатися з іменами формальних параметрів.
- Типи змінних, які є фактичними параметрами, мають збігатися з типами відповідних формальних параметрів чи то мати можливість бути перетвореними до типів формальних параметрів.
- Якщо кількість чи типи фактичних параметрів не збігаються з формальними параметрами, це призведе до помилки з відповідним повідомленням.
- Послідовність змінних, які є фактичними параметрами, має збігатися з послідовністю формальних параметрів.
- У прототипі функції необов'язково задавати імена формальних параметрів, достатньо їх оголосити, наприклад у такий спосіб:

  float seredne(int, int);
- Якщо функція не отримує жодного параметра, слід у заголовку функції ставити порожні дужки (опустити дужки не можна) чи записати у дужках void:

  double func(); double func(void);
- Тип значення, яке повертає функція, може бути якимзавгодно, але не може бути масивом чи функцією. Наприклад, таке оголошення є помилкове:

  int [10] func(); // Помилка!

  Якщо функція має повернути масив, то можна оголосити її тип як вказівник на перший елемент масиву (кількість елементів цього масиву має бути відомою):

  int * func();
- C++ дозволяє існування у програмі функцій з однаковим ім'ям, але різними параметрами.

Функції використовують пам'ять зі стека програми. Певна область стека надається функції й залишається пов'язаною з нею до завершення її роботи.

Після завершення роботи надана для функції пам'ять звільнюється й може бути зайнята іншою функцією.

Усі величини, оголошені всередині функції, а також її параметри, є локальними. Областю їхньої дії є функція, тобто для решти функцій програми ці змінні та їхні значення залишаються невідомими. Глобальні змінні є відомими всім функціям програми, кожна з функцій може змінювати значення таких змінних. Використовування глобальних змінних часто призводить до помилок, які важко відшукати. Тому слід уникати використовування глобальних змінних, якщо це не є нагальною потребою.

При викликанні функції, як і при вході до кожного блока, у стеку виділяється пам'ять під локальні автоматичні змінні. При виході з функції пам'ять, яку було виділено під локальні змінні, звільнюється. За спільної роботи функції обмінюються даними. Це можна здійснити за допомогою глобальних змінних, через параметри й через значення, які повертає функція.

Глобальні змінні є видимими у всіх функціях, де не описані локальні змінні з тими ж іменами, тому використовувати їх для передаванні даних між функціями дуже легко. Якщо імена глобальної й локальної змінної збігаються, перевага надається локальній змінній. Проте використовувати змінні з однаковими іменами не рекомендовано, оскільки це ускладнює налагодження програми. Слід прагнути до того, щоб функції були максимально незалежними, а їхній інтерфейс повністю визначався би прототипом функції.

Приклад 7.1. Обчислити значення виразу

$$x = \frac{\sqrt{6}+6}{2} + \frac{\sqrt{13}+13}{2} + \frac{\sqrt{21}+21}{2} \tag{7.1}$$

Розв'язок. Усі три доданки заданої формули є схожі один з одним. Тому кожен з них можна записати за допомогою спільної формули: $\frac{\sqrt{a}+a}{2}$, де a – число, яке у першому доданку дорівнює 6, у другому – 13, а у третьому – 21. Слушно є створити функцію, параметром якої буде дійсне число a і яка обчислюватиме значення цієї формули. У програмі функцію буде викликано тричі для кожного з доданків: 6, 13, 21.

Текст функції та її виклику в основній програмі:

```
double f(double a)
{ double b=(sqrt(a)+a)/2;
return b;
}
void __fastcall TForm1::Button1Click(TObject *Sender)
{ double x=f(6)+f(13)+f(21);
```

```
Edit1->Text=FormatFloat("0.00", x);
}
```

де **Button1, Edit1** — компненти для запуску функції на обчислення та візуалізації її результату.

Приклад 7.2. Обчислити за допомогою функції значення виразу

$$z = \frac{2 \cdot 5! + 3 \cdot 8!}{6! + 4!}. \tag{7.2}$$

Розв'язок. У формулі чотири рази зустрічається факторіал. Тому доречно обчислити значення факторіалу у функції і викликати його відповідно чотири рази для різних параметрів.

Текст функції та основної програми:

```
long fact(int n)
{ int i; long c=1;
for(i=1; i<=n; i++) c*=i;
return c;
}
void __fastcall TForm1::Button1Click(TObject *Sender)
{ double z=(2.0*fact(5)+3*fact(8))/(fact(6)+fact(4));
Edit1->Text=FloatToStr(z);
```

де **Button1, Edit1** — компненти для запуску функції на обчислення та візуалізації її результату.

Способи передавання параметрів до функцій

Значення автоматичних локальних змінних між викликами однієї й тієї самої функції не зберігаються. Якщо значення певних локальних змінних треба зберігати, при їхньому оголошуванні слід використовувати специфікатор **static**:

```
void f(int a)
{ int m=0;
cout<<"n m p\n";
while (a--)
{ static int n=0;
int p=0;
cout<<n++<<' '<<m++<<' '<<p++<<'\n';
} }
int main()
```

```
{ f(3); f(2);
return 0;
}
```

Статична змінна n оголошується й ініціалізується одноразово за першого виконання оператора, який містить її визначення. Автоматична змінна m оголошується й ініціалізується за кожного входження до функції. Автоматична змінна p оголошується й ініціалізується за кожного входження до блока циклу. Програма виведе на екран:

```
n m p
0 0 0
1 1 0
2 2 0

n m p
3 0 0
4 1 0
```

При викликанні функції першою чергою обчислюються вирази, які стоять на місці фактичних параметрів; потім у стеку виділяється пам"ять під формальні параметри функції відповідно до їхнього типу і кожному з них присвоюється значення відповідного фактичного параметра. При цьому перевіряється відповідність типів і, за потреби, виконуються їхні перетворювання. За невідповідності типів видається діагностичне повідомлення.

Існує два способи передавання параметрів до функції: за значенням і за адресою.

За передавання за значенням до стека заносяться копії фактичних параметрів і оператори функції працюють з цими копіями. Доступу до вихідних значень параметрів у функції немає, а, отже немає й можливості їх змінити.

За передавання за адресою до стека заносяться копії адрес параметрів, а функція здійснює доступ до комірок пам'яті за цими адресами і може змінювати вихідні значення параметрів. Передавати параметри за адресою можна у два способи: за допомогою вказівника і посилання.

За обох способів до функції передається адреса параметра, який зазначено при викликанні. При передаванні вказівника всередині функції виникає потреба використовувати явно операції розіменування чи то отримання адреси, що робить менш зрозумілим текст коду функції і більш складним налагодження функції. За передавання за посиланням до функції передається посилання, тобто синонім імені, внаслідок чого всередині функції всі звертання до параметра неявно розіменовуються.

Отже, використовування посилань замість вказівників зоптимізовує читабельність програми, позбавляючи потреби застосовувати операції отримання адреси й розіменування. Використовування посилань замість передавання за значенням є більш ефективне, оскільки не потребує копіювання параметрів, а це є надто важливо при передаванні структур даних великого обсягу.

У поданому нижче фрагменті програми функція f має три параметри, один з яких передається за значенням, другий — за вказівником, а третій — за посиланням.

Тексти функції та її виклику в основній програмі:

```
void f(int i, int* j, int& k);
int main()
{ int i=1, j = 2, k = 3;
cout << "i j k\n";
cout <<i<<' '<<j<<' '<<k <<'\n';
f(i,&j,k);
cout<<i<<' '<<j<<' '<<k;
}
void f(int i, int* j, int& k)
{ i++; (*j)++; k++;
}
```

Результат роботи програми:

```
i j k
1 2 3
1 3 4
```

Перший параметр i передається до функції за значенням. Його змінення у функції не впливає на вихідне значення. Другий параметр j передається за адресою за допомогою вказівника, при цьому для передавання до функції адреси фактичного параметра використовується операція отримання адреси $\&j$, а для одержання його значення у функції потрібна операція розіменування (розадресації) $*j$. Третій параметр k передається за адресою за допомогою посилання. Це дозволяє передавати фактичний параметр k без операції отримання адреси і використовувати його у функції без розадресації.

Якщо виникає потреба уникнути змінювання параметра всередині функції, використовується специфікатор **const:**

```
int f(const char*);
char *t(char *a, const int *b);
```

На місце параметра типу **const&** може передаватися константа, а для змінної за потреби – виконуватися перетворення типу. Вихідні дані, які не повинні змінюватися у функції, більш оптимально є передавати їй за допомогою константних посилань.

За замовчуванням параметри будь-якого типу, окрім масиву й функції, передаються до функції за значенням.

При використовуванні масиву як параметра функції передається вказівник на його перший елемент, тобто масив завжди передається за адресою. При цьому інформація про кількість елементів втрачається і слід передавати його розмірність через окремий параметр. Якщо розмірність масиву є константою, проблем не виникає, оскільки можна зазначити її при оголошенні формального параметра і як верхню межу циклів при опрацюванні масиву всередині функції. У разі передавання до функції масиву символів, тобто рядка, його фактичну довжину можна визначити за розташуванням нуль-символу і передавати кількість елементів немає потреби.

У поданому нижче фрагменті програми оголошується функція sum(), параметрами якої є масив і кількість його елементів. Ця функція обчислює суму елементів масиву.

```
int sum (int* mas, int n) {
// Варіанти: int sum(int mas[ ], int n)
// чи int sum(int mas[n]) (якщо n - константа).
for(int i = 0, s = 0; i < n; i++) s += mas[i];
return s;
}
void main()
{ int marks[] = {3, 4, 5, 4, 4};
cout << "Сума елементів масиву: "<< sum(marks, 5);
}
```

При передаванні до функції багатовимірних масивів усі розмірності, як-що вони є невідомі на етапі компіляції, мають передаватися як параметри. Приміром, заголовок функції, яка обчислює суму елементів динамічного двовимірного масиву, може мати вигляд

```
int sum(const int **a, const int nr, const int nc);
```

Виклик цієї функції у програмі може бути таким:

```
cout << "Сума елементів a: " << sum((const int**)a,nstr,nstb);
```

Для звичайного статичного двовимірного масиву, коли обидві розмірності є відомі й є константами, заголовок функції матиме вигляд

```
int sum(int a[4][6]);
```

Приклад 7.3. Увести матрицю дійсних чисел розмірності 4×3 й визначити за допомогою функції мінімальний елемент матриці.

Текст функції та її виклику в основній програмі:

Розв'язок. Текст функції, її виклику в основній програмі головної форми (див. рис.7.1)

```
double matrix_min(double a[4][3]) {
double min=a[0][0];
for(int i=0; i<4; i++)
for(int j=0; j<3; j++)
if(a[i][j]<min)
min=a[i][j];
return min; }

void __fastcall TForm1::BitBtn1Click(TObject *Sender) {
double a[4][3], m;
for(int i=0; i<4; i++)
for(int j=0; j<3; j++)
a[i][j]=StrToFloat(StringGrid1->Cells[j][i]);
m=matrix_min(a);
Edit1->Text=FloatToStr(m); }
```

Рис. 7.1 — Екранна форма додатку для пошуку мінімального елементу матриці за допомогою функції

Приклад 7.4. Увести матрицю дійсних чисел розмірності 4×3 й обчислити за допомогою функції мінімальний елемент матриці та його індекси.

Розв'язок. Цей приклад є продовженням попереднього. Змінимо функцію в такий спосіб, щоб вона обчислювала і повертала три значення. У цьому разі слід передати до функції два додаткових параметри — ind_i та ind_j, — в які буде записано обидва індекси мінімального елемента. Ці параметри мають передаватися за посиланням (чи за покажчиком), щоб у тілі функції можна було їх змінити і повернути ці зміни у точку виклику.

Текст функції, її виклику в основній програмі головної форми (див. рис.7.2)

```
double matrix_min(double a[4][3], int& ind_i, int& ind_j)
{ double min=a[0][0];
ind_i=0; ind_j=0;
for(int i=0; i<4; i++)
for(int j=0; j<3; j++)
if(a[i][j]<min)
{ min=a[i][j];
ind_i=i;
ind_j=j;
}
return min;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{ double a[4][3], m; int i, j, ind_i=0, ind_j=0;
for(i=0;i<4;i++)
for(j=0;j<3;j++) a[i][j]=StrToInt(StringGrid1->Cells[j][i]);
m = matrix_min(a, ind_i, ind_j);
Edit1->Text = FloatToStr(m) + "(" + IntToStr(ind_i) + ", " +
IntToStr(ind_j) + ")";
}
```

Параметри зі значеннями за замовчуванням

Щоб спростити виклик функції, в її заголовку можна задати значення параметрів за замовчуванням. Ці параметри мають бути останніми у списку й можуть опускатися при виклику функції. Якщо при виклику параметр є опущений, має бути опущено й усі параметри, що стоять за ним. Як значення параметрів за замовчуванням можуть використовуватися константи, глобальні змінні й вирази.

Приклади прототипів функцій з параметрами за замовчуванням.

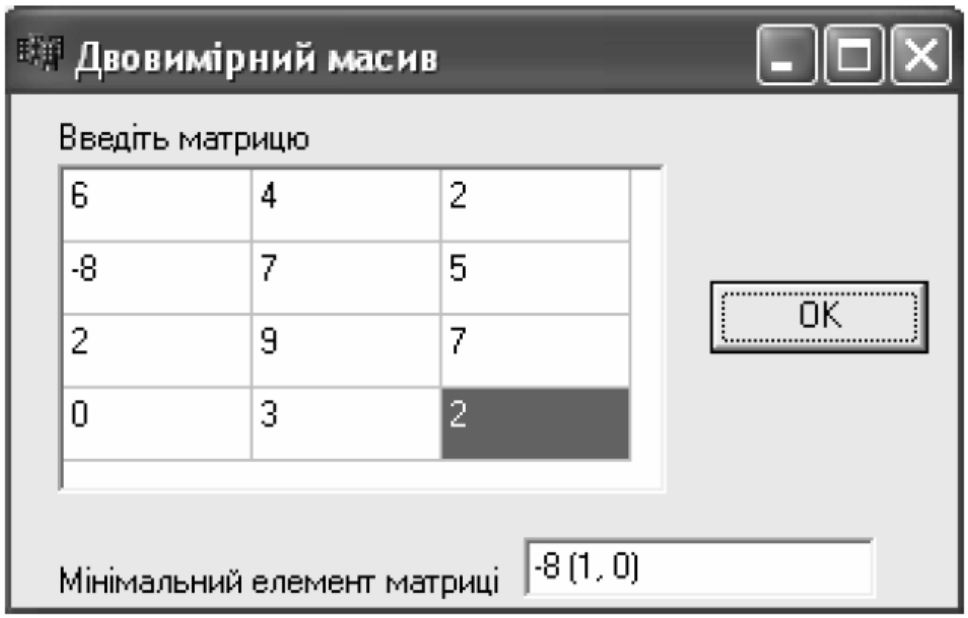


Рис. 7.2 — Екранна форма додатку для пошуку мінімального елементу матриці та його індексів за допомогою функції

```
int f(int a, int b=0); // Параметр b має значення
// за замовчуванням 0.
void f1(int, int=100, char* = 0);
// Зверніть увагу на пробіл між * й =,
// без нього б вийшла операція складного присвоювання *=
void err(int errValue = errno); // errno - глобальна змінна.
```

Варіанти виклику цих функцій:

```
f(100); // Виклик функції з першим параметром 100,
// другий за замовчуванням - 0.
f(A, 1); // Виклик функції з першим параметром -
// значенням змінної A, другим - 1.
f1(A); // Виклик функції з другим та третім параметрами
// за замовчуванням.
f1(a, 10); // Виклик функції зі значенням третього параметра
// за замовчуванням.
f1(a, 10, "Hello"); // Виклик без значень параметрів
// за замовчуванням.
```

Функції зі змінною кількістю параметрів

Якщо список формальних параметрів функції завершується трьома крапками, це означає, що при її виклику на цьому місці можна зазначити ще кілька параметрів. Перевірка відповідності типів для цих параметрів не виконується, char та short передаються як int, а float – як double. За приклад стандартної

функції зі змінним числом параметрів можна навести функцію printf(), прототип якої має вигляд

```
int printf (const char*, ...)
```

Це означає, що виклик функції має містити принаймні один параметр типу char* й може чи то містити, чи не містити інші параметри:

```
printf("Уведіть вихідні дані"); // Один параметр
printf("Сума: %5.2f рублів", sum); // Два параметри
printf("%d %d %d %d", a, b, c, d); // П'ять параметрів
```

Для роботи з функціями зі змінною кількістю аргументів використовуються величини va_list, va_start, va_arg і va_end, описані в заголовному файлі <stdarg.h>.

Тип va_list призначено для зберігання покажчика на черговий аргумент.

Для роботи з функціями зі змінною кількістю аргументів використовуються декілька стандартних макросів (дещо подібних до функцій наборів команд).

Макрос va_start ініціює покажчик на черговий елемент, він встановлює аргумент arg_ptr на початок списку необов'язкових параметрів і має вигляд функції з двома параметрами:

```
void va_start(arg_ptr, prav_param);
```

де параметр prav_param має бути останнім обов'язковим параметром викликуваної функції, а покажчик arg_ptr має бути оголошено у списку змінних типу va_list у вигляді

```
va_list arg_pt;
```

Макрос va_start має бути використано до першого використання макросу va_arg. Макрос va_arg повертає значення чергового аргументу, кожен його виклик призводить до просування покажчика, який зберігається в va_list. Макрокоманда va_arg забезпечує доступ до поточного параметра викликуваної функції й теж має вигляд функції із двома параметрами:

```
type_arg va_arg (arg_ptr, type);
```

Ця макрокоманда бере значення типу type за адресою, заданою покажчиком arg_ptr, збільшує значення покажчика arg_ptr на довжину використаного

параметра (довжина type) і в такий спосіб параметр arg_ptr вказуватиме на наступний параметр викликуваної функції. Макрокоманда va_arg використається стільки разів, скільки треба для отримання всіх параметрів викликуваної функції.

Після перебирання аргументів, але до виходу з функції зі змінним числом аргументів, необхідно звернутися до макросу va_end. Він встановлює покажчик списку необов'язкових параметрів на NULL.

Рекурсивні функції

Рекурсивною називається функція, яка викликає сама себе. Така рекурсія називається прямою. Існує ще непряма рекурсія, коли дві чи більше функцій викликають одна одну.

Класичним прикладом рекурсивної функції є обчислення факторіала (це не означає, що факторіал треба обчислювати лише в такий спосіб). Для того щоб визначити факторіал числа n, слід помножити n на факторіал числа $(n-1)$:

```
n! = n*(n-1)! // Це так зване рекурентне співвідношення.
```

Відомо також, що $0! = 1$ й $1! = 1$, тобто умовою зупинки рекурсії буде: якщо $n = 0$ чи $n = 1$, то факторіал дорівнює 1.

Дамо повне рекурсивне визначення факторіала:

$$n! = \begin{cases} (n-1)! \cdot n, & n > 0; \\ 1, & n = 0. \end{cases} \tag{7.3}$$

Отже, рекурсивна функція обчислення факторіала числа n виглядатиме так:

```
long fact (long n)
{ if(n ==0 || n==1) return 1;
return (n*fact(n-1)); }
```

Якщо у функції **main()** зустрінеться виклик функції **fact(3),** при виконуванні цієї функції буде викликано функцію **fact(2),** яка своєю чергою викличе **fact(1).** Для останньої спрацює умова зупинки рекурсії **(n==1)** і у функцію **fact(2)** буде повернуто значення **1** та обчислено значення **2 (2! = 2),** яке своєю чергою буде повернуто до функції **fact(3)** і буде використано для обчислення **3!.**

7.2. Завдання для аудиторної роботи

1. Скласти блок-схему алгоритму і написати програмний код для обчислення за допомогою функції значення виразу

$$x = \frac{\sin(0.1\pi) + 0.1}{3 \cdot \cos(0.1\pi)} + \frac{\sin(0.2\pi) + 0.2}{3 \cdot \cos(0.2\pi)} + \frac{\sin(0.3\pi) + 0.3}{3 \cdot \cos(0.3\pi)} \tag{7.4}$$

2. Скласти блок-схему алгоритму і написати програмний код для обчислення за допомогою функції значення виразу

$$z = \frac{2\cos(5\pi) \cdot 5! + 3\cos(8\pi) \cdot 8!}{\cos(6\pi) \cdot 6! + 4!}. \tag{7.5}$$

3. Скласти блок-схему алгоритму і написати програмний код для введення матриці дійсних чисел розмірності 4×5 і обчислення за допомогою функції добутку мінімального значення елементів матриці на максимальне значення елементів матриці.

4. Скласти блок-схему алгоритму і написати програмний код для введення матриці дійсних чисел розмірності 3×4 і знаходження за допомогою функції значень мінімального та максимального елементів матриці та їх індекси.

7.3. Завдання для самостійної роботи

Вивчити основи технології програмування функцій [5], [6] в середовищі Borland C++ Builder Enterprise v6.0 та їх використання при побудові додатків.

Уважно проаналізувати зміст прикладів теоретичної частини комп'ютерного практикуму №7 та на їх основі реалізувати програмні коди, які були підготвлені під час виконання аудиторних завдань у вигляді запрограмованих додатків у середовищі Borland C++ Builder Enterprise v6.0.

Передати викладачу на перевірку звіт про виконане завдання із зображенням блок-схем алгоритмів, лістінгів програмного коду та відкомпільованих виконавчих файлів запрограмованих додатків. Термін виконання завдання для самостійної роботи 2 тижні.

7.4. Варіанти завдань для аудиторної та домашньої роботи студентів

1. Скласти блок-схему алгоритму і написати програмний код для обчислення за допомогою функції значення виразу

$$x = \frac{\mathrm{tg}(0.1\pi) + 0.1}{3 \cdot \mathrm{ctg}(0.1\pi)} + \frac{\mathrm{tg}(0.2\pi) + 0.2}{3 \cdot \mathrm{ctg}(0.2\pi)} + \frac{\mathrm{tg}(0.3\pi) + 0.3}{3 \cdot \mathrm{ctg}(0.3\pi)} \tag{7.6}$$

2. Скласти блок-схему алгоритму і написати програмний код для обчислення за допомогою функції значення виразу

$$z = \frac{2\,\mathrm{ctg}(5\pi) \cdot 5! + 3\,\mathrm{ctg}(8\pi) \cdot 8!}{\mathrm{ctg}(6\pi) \cdot 6! + 4!}. \tag{7.7}$$

3. Скласти блок-схему алгоритму і написати програмний код для обчислення за допомогою функції значення виразу

$$x = \frac{\mathrm{tg}(0.1\pi) + 0.1}{3 \cdot \cos(0.1\pi)} \cdot \frac{\mathrm{tg}(0.2\pi) + 0.2}{3 \cdot \cos(0.2\pi)} \cdot \frac{\mathrm{tg}(0.3\pi) + 0.3}{3 \cdot \cos(0.3\pi)} \tag{7.8}$$

4. Скласти блок-схему алгоритму і написати програмний код для обчислення за допомогою функції значення виразу

$$z = \frac{2\cos(5\pi) \cdot 5! \cdot 3\cos(8\pi) \cdot 8!}{\cos(6\pi) \cdot 6! \cdot 4!}. \tag{7.9}$$

5. Скласти блок-схему алгоритму і написати програмний код для обчислення функції

$$f(x) = \begin{cases} -\dfrac{2x + c}{cx + 2a} \text{ при } x < 0, \quad b + c > 5; \\ \dfrac{x - a}{\sqrt{x} + c} \text{ при } x > 0, \quad b - c < 0; \\ \dfrac{3x}{c} - \dfrac{b}{2x(x^2 + 1)} \text{ в інших випадках.} \end{cases},$$

де параметри a, b, c — дійсні позитивні числа.

6. Скласти блок-схему алгоритму і написати програмний код для обчисле-

ння функції

$$f(x) = \begin{cases} \dfrac{ax^2 + bx}{c} \text{ при } x - c < 0, \quad b > 0; \\ \dfrac{\sqrt{x}}{2x - a} + \dfrac{a}{3x + c} \text{ при } x - c > 0; \\ -\dfrac{x\sqrt{x + bc}}{c + b} \text{ в інших випадках.} \end{cases},$$

де параметри a, b, c — дійсні позитивні числа.

7. Скласти блок-схему алгоритму і написати програмний код для обчислення функції

$$f(x) = \begin{cases} x^2 + \dfrac{ax}{x^3 - c} \text{ при } x > 3, \quad ac < 0; \\ \sqrt{x^2 - b} + cx \text{ при } x < 0, \quad ac > 0; \\ \dfrac{\sqrt{x}}{1 + x} - (c - x)^2 \text{ в інших випадках.} \end{cases},$$

де параметри a, b, c — дійсні позитивні числа.

8. Скласти блок-схему алгоритму і написати програмний код для обчислення функції

$$f(x) = \begin{cases} -ax^3 + bcx \text{ при } x < 0, \quad b > 0; \\ \sqrt{x^3} + \dfrac{ax}{cx + b} \text{ при } x > 0, \quad b < 0; \\ \dfrac{4x^2}{cx - a} - bx^3 \text{ в інших випадках.} \end{cases},$$

де параметри a, b, c — дійсні позитивні числа.

9. Скласти блок-схему алгоритму і написати програмний код для обчислення суми ряду

$$S = \sum_{k=1}^{n} \frac{(-1)^{k+1} \sum\limits_{i=1}^{k} i}{2k + 1},$$

де аргументом є кількість членів ряду n.

10. Скласти блок-схему алгоритму і написати програмний код для обчисле-

ння суми ряду

$$S = \sum_{k=1}^{n} \frac{k!}{\sum\limits_{i=1}^{k} \sin(i)},$$

де аргументом є кількість членів ряду n.

11. Скласти блок-схему алгоритму і написати програмний код для обчислення суми ряду

$$S = \sum_{k=1}^{n} \frac{1}{\sum\limits_{i=1}^{k} \cos(i)},$$

де аргументом є кількість членів ряду n.

12. Скласти блок-схему алгоритму і написати програмний код для обчислення за допомогою функції об'єму та площу поверхні кулі, де аргументом є її радіус.

13. Скласти блок-схему алгоритму і написати програмний код для обчислення за допомогою функції об'єму та площу поверхні циліндра, де аргументом є його радіус та висота.

14. Скласти блок-схему алгоритму і написати програмний код для обчислення за допомогою функції об'єм та площу поверхні куба, де аргументом є довжина його ребра.

15. Скласти блок-схему алгоритму і написати програмний код для обчислення за допомогою функції силу електричного струму у розімкнутому однорідному електричному колі

$$I = \frac{\Delta\varphi}{R},$$

де аргументами є різниця потенціалів $\Delta\varphi$ та активний опір R.

16. Скласти блок-схему алгоритму і написати програмний код для обчислення за допомогою функції силу електричного струму у замкнутому неоднорідному електричному колі

$$I = \frac{\varepsilon}{R},$$

де аргументами є електрорушійна сила ε та активний опір R.

17. Скласти блок-схему алгоритму і написати програмний код для обчислення за допомогою функції силу електричного струму у розімкнутому

неоднорідному електричному колі

$$I = \frac{\Delta\varphi + \varepsilon}{R},$$

де аргументами є $\Delta\varphi$, електрорушійна сила ε та активний опір R.

18. Скласти блок-схему алгоритму і написати програмний код для обчислення за допомогою функції швидкості та координати вільно вертикально падаючого тіла, де аргументами є початкові координати та швидкість.

19. Скласти блок-схему алгоритму і написати програмний код для обчислення за допомогою функції циклічної частоти $\omega_1 = \sqrt{\frac{1}{LC} - \left(\frac{R}{2L}\right)^2}$ для RLC — контура, де аргументами є величини R, L, C.

20. Скласти блок-схему алгоритму і написати програмний код для обчислення за допомогою функції періоду коливань $T = \frac{2\pi}{\sqrt{\frac{1}{LC} - \left(\frac{R}{2L}\right)^2}}$ для RLC контура, де аргументами є величини R, L, C.

21. Скласти блок-схему алгоритму і написати програмний код для обчислення за допомогою функції добротності $Q = \frac{1}{R}\sqrt{\frac{L}{C}}$ для RLC контура, де аргументами є величини R, L, C.

22. Скласти блок-схему алгоритму і написати програмний код для обчислення за допомогою функції довжини хвилі λ залежної від циклічної частоти ω оптичного випромінювання за допомогою дисперсійного рівняння

$$\vec{k}^2 = \frac{\omega^2}{c^2}\varepsilon, \quad \lambda = \frac{2\pi}{|\vec{k}|},$$

де c — швидкість у вакуумі, ε —діелектрична проникливість середовища.

23. Скласти блок-схему алгоритму і написати програмний код для обчислення за допомогою функції енергію випромінювання та поглинання фотонів атомом водню

$$\hbar\omega_{nm} = R\Big(\frac{1}{m^2} - \frac{1}{n^2}\Big),$$

де $\hbar$ — постійна Планка, R — постійна Рідберга, n, m — цілі числа. за рахунок електронних переходів між їх енергетичними рівнями в атомі водню.

24. Скласти блок-схему алгоритму і написати програмний код для введення матриці дійсних чисел розмірності 8×9 і обчислення за допомогою

функції добутку максимального значення елементів матриці на середнє значення елементів матриці.

25. Скласти блок-схему алгоритму і написати програмний код для введення матриці дійсних чисел розмірності 5×3 і знаходження за допомогою функції значень мінімального і максимального елементів матриці, їх індекси та середнє значення елементів матриці.

7.5. Питання для самоконтролю знань

- У чому полягає різниця поміж оголошенням прототипу й визначенням функції?
- Який тип має функція, яка не повертає значення?
- Які параметри функції називають формальними, а які — фактичними?
- Чи повинні збігатися імена відповідних формальних і фактичних параметрів функції?
- В яких випадках слід використовувати статичні глобальні змінні?
- Що таке локальна змінна?
- Що таке область видимості змінної?
- Поясніть різницю поміж локальними та глобальними змінними.
- В яких випадках слід використовувати глобальні змінні?
- Які є способи передавання параметрів до функції?
- Що таке рекурсивна функція?

8. Комп'ютерний практикум №8. Алгоритмізація та програмування додатку для аналізу траєкторій польоту ракети в межах земної атмосфери без врахування опору повітря

Мета практикуму. Набуття практичного досвіду використання функцій при побудові комп'ютерних додатків з врахуванням особливостей предметного середовища.

Детально розглянути замкнутий цикл програмування комп'ютерного додатку для аналізу траєкторій польоту ракети в межах земної атмосфери без

врахування опору повітря, що, в свою чергу, розкриває значимість розуміння програмістом предметного середовища для розробки конкретного проекту.

8.1. Теоретичні відомості стосовно предметного середовища

Теоретичною основою даного проекту є рівняння Мещерського для руху матеріальної точки із змінною масою m

$$m\frac{d\vec{v}}{dt} = \vec{f} - \vec{v}_p\frac{dm}{dt},$$

де $\vec{f} = m\vec{g}$ — сила земного тяжіння, що діє на ракету, $\vec{v}_p = \vec{v}_g - \vec{v}$ — відносна швидкість витікання гарячих газів яка направлена в протилежному напрямі по відношенню до напряму руху ракети, а $\vec{v}_g$ — абсолютна швидкість витікання гарячих газів. Прицьому треба враховувати, що $dm < 0$ оскільки маса ракети зменшується. Що стосується величини $\vec{v}_p$ то вона із задовільною точністю вважається постійною величиною.

Далі зручно в рівнянні руху перейти до безрозмірних змінних, а саме

$$\tau = \frac{t}{t_0}, \quad \vec{V} = \frac{\vec{v}}{v_0}, \quad v_0 = gt_0, \quad \vec{V}_p = \frac{\vec{v}_p}{v_0}, \quad \vec{R} = \frac{\vec{r}}{r_0}, \quad \vec{r}_0 = \frac{g{t_0}^2}{2},$$

де t_0 — час роботи двигуна ракети. В нових змінних рівняння Мещерського набуває вигляду

$$\frac{d\vec{V}}{d\tau} = \frac{\vec{g}}{g} - \vec{V}_p\frac{1}{\mu}\frac{d\mu}{d\tau},$$

де $\mu = \dfrac{m}{m_0 + m_1}$ — безрозмірна маса ракети.

Розв'язок цього рівняння визначає швидкість руху ракети на активній ділянці траєкторії в наступному вигляді

$$\vec{V} = \frac{\vec{g}}{g}\tau + \vec{V}_p\ln(\mu).$$

Положення ракети на активній ділянці праєкторії руху $(0 < \tau < 1)$ буде визначатися інтегралом

$$\vec{R} = \vec{R}_0 + \frac{\vec{g}}{g}\frac{\tau^2}{2} + \vec{V}_p\int_0^\tau \ln(\mu)d\tau.$$

Тут $\vec{R}_0$ координати ракети в момент старту.

Параметр відповідальний за швидкість згорання палива α визначається з рівняння

$$m_0 = m_f\mu_1, \quad \mu_1 = \mu(\tau = 1)$$

якому можна надати вигляд

$$\mu_1 = \frac{1}{1+\chi}, \quad \chi = \frac{m_1}{m_0},$$

де χ — так названий параметр Ціолковського.

Залежність маси ракети від часу визначається конструкцією двигуна і при моделюванні руху ракети зазвичай використовують такі формули:

$$m = m_f(1-\alpha t), \quad \mu = (1-\alpha t) = (1-\alpha_1\tau),$$

$$m = m_f \exp(-\alpha t), \quad \mu = \exp(-\alpha t) = \exp(-\alpha_1\tau)$$

де $\alpha_1 = \alpha t_0$, $m_f = m_0 + m_1$ — повна маса ракети на старті, α — параметр що визначає швидкість витікання гарячих газів. В цих випадках прискорення ракети a буде визначатися формулами $\vec{a} = \vec{g} - \alpha\vec{v}_p\dfrac{m_f}{m}$ та $\vec{a} = \vec{g} - \alpha\vec{v}_p$ відповідно. 0чевидно перший випадок описує політ ракети з постійною реактивною силою, а другий — з постійнм прискоренням. За рахунок простих алгебраїчних перетворень знайдемо $\alpha = \dfrac{1}{t_0}\dfrac{\chi}{1+\chi}$ для першого випадку та $\alpha = \dfrac{1}{t_0}\ln(1+\chi)$ для другого випадку.

В першому випадку безрозмірні координати ракети на активній ділянці польоту визначаються як

$$\vec{R} = \vec{R}_0 + \frac{\vec{g}}{g}\frac{\tau^2}{2} - \frac{\vec{V}_p}{\alpha_1}((1-\alpha_1\tau)\ln(1-\alpha_1\tau) + \alpha_1\tau).$$

а в другому випадку будемо мати

$$\vec{R} = \vec{R}_0 + \frac{\vec{g}}{g}\frac{\tau^2}{2} - \vec{V}_p\frac{\alpha_1\tau^2}{2}.$$

Після вигорання палива ракета продовжує рух по закону вільного руху тіла кинутого з деякої висоти $Z_f = r_z(t_f)$ під певнм кутом θ до горизонту (шкільна задача з фізики). На ділянці пасивного польоту безрозмірні швидкість та координати ракети визначаються наступними формулами

$$\vec{V} = \vec{V}(\tau_1) + \frac{\vec{g}}{g}(\tau - \tau_1),$$

$$\vec{R} = \vec{R}(\tau_1) + \vec{V}(\tau_1)(\tau - \tau_1) + \frac{\vec{g}}{g}\frac{(\tau-\tau_1)^2}{2}$$

Де $\tau_1 = 1$. Максимальна висота підйому ракети Z_h визначається з умови

$$Z_h = R_z(\tau_h), \quad V_z(\tau_h) = V_z(\tau_1) - (\tau_h - \tau_1) = 0, \quad \tau_h = \tau_1 + V_z(\tau_1),$$

де τ_h — момент досягання ракетою найвищої точки підйому.

Момент падіння ракети на Землю t_m визначається як розв'язок рівняння

$$R_z(\tau_m) = R_z(\tau_1) + V_z(\tau_1)(\tau_m - \tau_1) - \frac{(\tau_m - \tau_1)^2}{2} = 0$$

що має фізичний зміст, а саме

$$\tau_m = \tau_1 + V_z(\tau_1) + \sqrt{V_z(\tau_1)^2 + \frac{R_z(\tau_1)}{2}}$$

Максимальне горизонтальге зміщення ракети L_m буде рівним

$$L_m = \sqrt{R_x(\tau_1)^2 + R_y(\tau_1)^2} + \sqrt{V_x(\tau_1)^2 + V_y(\tau_1)^2}(\tau_m - \tau_1).$$

Після виконання всіх розрахунків потрідно повернутися до фізичних змінних, а саме $t, \vec{r}, \vec{v}$.

При програмуванні вважати, що рух ракети відбувається в площині $(0, x, z)$, тобто координати і швидкість руху ракети відображаються векторами $\vec{r} = (r_x(t), 0, r_z(t)), \vec{v} = (v_x(t), 0, v_z(t))$. При цьому треба розуміти, що отримані кількісні результати не мають ніякого практичного значення, оскільки при тих швидкостях, які розвиває ракета, значну роль грають сили опору повітря, які, в свою чергу, якісно змінюють результати розрахунків. Тобто це завдання має сугубо навчальне значення в питанні розуміння предметного середовища для постаноки задачі програмісту та набуття навичок у використання функцій при програмуванні комп'ютерних додатків.

Приклад програмування візуального додатку

Розглянемо програмування візуального додатку для побудови траєкторій польоту ракети в межах земної атмосфери (див. рис.8.1) для наступних початковий умов: в момент $t = 0$ $\vec{r}(t = 0) = 0$, $\vec{v}(t = 0) = 0$, де $\vec{r} = (r_x, r_y, r_z), \vec{v} = (v_x, v_y, v_z)$ — кординати та швидкість руху ракети, а маса незаправленої паливом ракети m_0, стартова маса палива m_1 та відносна швидкість витікання гарячих газів $\vec{v}_p$ є параметрами задачі, які визначаються у компонентах **Edit**. Для побудови графічного інтерфейсу додатку на головній формі проекту (див. рис.8.1), розташувати колекцію компонентів

$$\begin{cases} TForm - 1 \text{ шт.}; \quad TPanel - 3 \text{ шт.}; \quad TRadioButton - 2 \text{ шт.}; \\ TGroupBox - 2 \text{ шт.}; \quad TLabel - 13 \text{ шт.}; \quad TEdit - 13 \text{ шт.}; \\ TImage - 1 \text{ шт.}; \quad TButton - 3 \text{ шт.} \end{cases} \tag{8.1}$$

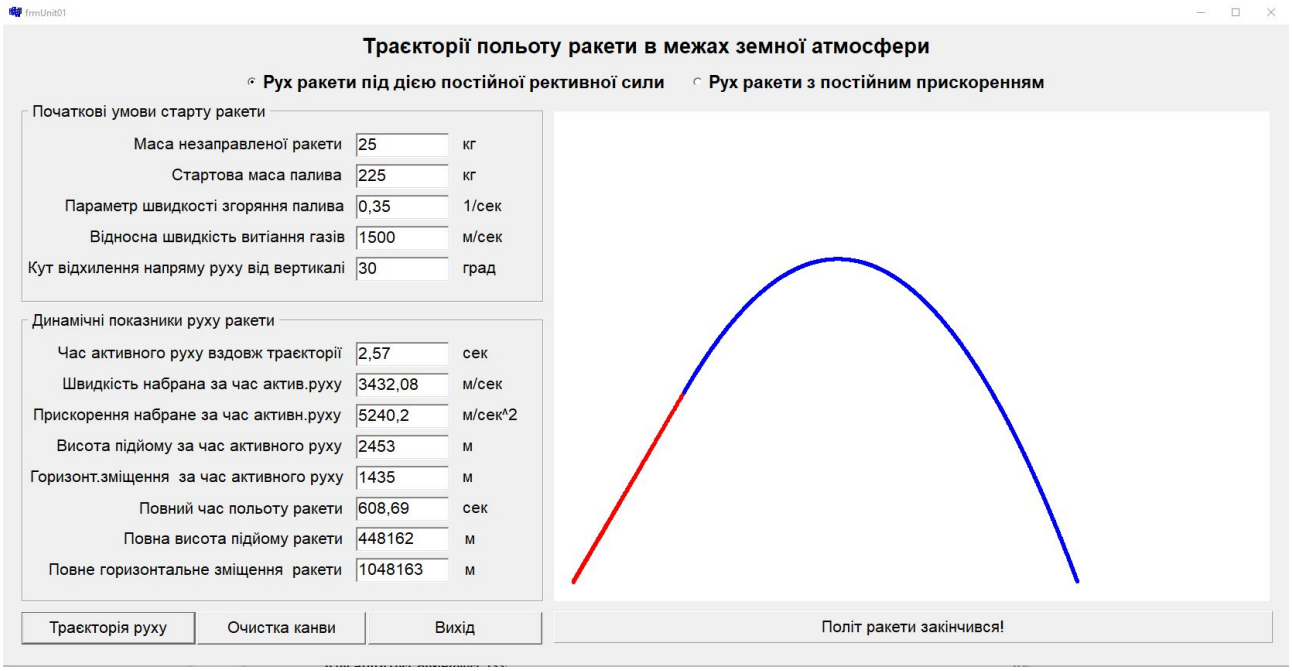


Рис. 8.1 — Екранна форма додатку для візуалізації траєкторій руху ракети

Приклад програмного коду, що реалізує поставлену задачу, приведено нижче.

```
/*- Unit01.cpp ------------------------------------------------*/
#include <vcl.h>
#pragma hdrstop
#include <Math.hpp>
#include "Unit01.h"
#include <math.h>
#include <stdio.h>
/*---------------------------------------------------------------*/
#pragma package(smart_init)
#pragma resource "*.dfm"
TfrmUnit01 *frmUnit01;
const float pi = 3.141563;
const float g = 9.8;
int flag,i,Cx,Cz;
float M,M0,Kc,K0,K1,Yz,T0,V0,R0,X0,Z0,V1,T1,F,dt,dX,dZ;
float U0[5],U1[5],U[5],Uz[5],Uh[5];
TRect aRect;
/*---------------------------------------------------------------*/
__fastcall TfrmUnit01::TfrmUnit01(TComponent* Owner)
        : TForm(Owner)
{ }
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::FormShow(TObject *Sender)
{
try {
```

```
        frmUnit01->Caption = frmUnit01->Header0->Caption;
        Timer0->Enabled = false;
        flag = (RadioButton0->Checked) ? 0 : 1;
        dt = 0.01;
        M0 = StrToFloat(Edit0->Text);
        Kc = StrToFloat(Edit1->Text);
        K0 = Kc/(1+Kc);
        Yz = pi*StrToFloat(Edit5->Text)/180;
        T0 = StrToFloat(Edit2->Text);
        V0 = g*T0;
        R0 = g*T0*T0/2;
        V1 = StrToFloat(Edit6->Text)/V0;
        dt = dt/T0;
        T1 = 1;
        X0 = StrToFloat(Edit3->Text)/R0;
        Z0 = StrToFloat(Edit4->Text)/R0;

        U0[0] = 0;
        U0[1] = 0.0*V0*sin(Yz);
        U0[2] = 0.0*V0*cos(Yz);
        U0[3] = X0;
        U0[4] = Z0;

        for(int k=0; k<5; k++) {Uz[k] = 0;}

        Button1Click((TObject *)Sender);
} catch(...) {
        ShowMessage("Помилка введення даних!");
}}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::RadioButton0Click(TObject *Sender)
{
FormShow((TObject *)Sender);
        Button1Click((TObject *)Sender);
}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::Edit0Change(TObject *Sender)
{
FormShow((TObject *)Sender);
}
/*---------------------------------------------------------------*/
```

```
void __fastcall TfrmUnit01::Button0Click(TObject *Sender)
{
RadioButton0Click((TObject *)Sender);
        i = 0;
        Timer0->Enabled = true;
        Panel2->Caption = "Ракета в польоті";
}
/*-----------------------------------------------------------------*/
void __fastcall TfrmUnit01::Timer0Timer(TObject *Sender)
{
        fn_Exec0((TObject *)Sender);
        fn_Data0((TObject *)Sender);
        fn_Image0((TObject *)Sender);
        if (U[4] < 0) {
                Timer0->Enabled = false;
                Panel2->Caption = "Ракета зірвалася!!!";
        };
}
/*-----------------------------------------------------------------*/
void __fastcall TfrmUnit01::fn_Exec0(TObject *Sender)
{
int q;
K1 = (flag==0) ? Kc/(1+Kc) : log(1+Kc);
if (i == 0) {
for(int k=0; k<5; k++) {U[k] = U0[k];}}
else
if (i > 0) {
for(int k=0; k<5; k++) {Uz[k] = U[k];}
U[0] = U[0]+dt;
if (U[0] <= 1) {
M = (flag==0) ? (1-K1*U[0]) : exp(-K1*U[0]);
F = (flag==0) ? V1*(K1*U[0]+(1-K1*U[0])*log(1-K1*U[0])) :
V1*K1*U[0]*U[0]/2; }
else {
M = K0/Kc;
F = 0;
}
if (U[0] <= 1) {
U1[1] = -V1*sin(Yz)*log(M);
U1[2] = -V1*cos(Yz)*log(M)-U[0];
U1[3] = X0+F*sin(Yz);
```

```
U1[4] = Z0+F*cos(Yz)-K1*U[0]*U[0]/2;
}
U[1] = U1[1];
U[2] = U1[2]+((U[0] > 1) ? -(U[0]-1) : 0);
U[3] = U1[3]+((U[0] > 1) ? U1[1]*(U[0]-1) : 0);
U[4] = U1[4]+((U[0] > 1) ? U1[2]*(U[0]-1)-(U[0]-1)*(U[0]-1)/2 : 0);
}
if (Uz[2]*U[2] < 0) {
for(int k=0; k<5; k++) {Uh[k] = Uz[k];}
}
i++;
}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::fn_Data0(TObject *Sender) {
        float T, H;
        if (U[2] > 0) { T = 0; H = 0;}
        else { T = Uh[0]*T0; H = Uh[4]*R0; }
        Edit9->Text = FloatToStr(SimpleRoundTo(T,-2));
        Edit10->Text = FloatToStr(SimpleRoundTo(H,-2));
        Edit11->Text = FloatToStr(SimpleRoundTo(U[0]*T0,-2));
        Edit12->Text =
FloatToStr(SimpleRoundTo(sqrt(U[1]*U[1]+U[2]*U[2])*V0,-2));
        Edit13->Text =
FloatToStr(SimpleRoundTo((U[1]*U[1]+U[2]*U[2])*V0*V0/2/M0,-2));
        Edit14->Text = FloatToStr(SimpleRoundTo(U[3]*R0,-2));
}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::fn_Image0(TObject *Sender) {
        dX =  U[3]*R0*0.25/(1+Kc);
        dZ = -U[4]*R0*2.50/(1+Kc);
        if (i <= 1) {Image0->Canvas->Pen->Color=clWhite;}
        else
        if (U[0] <= T1) {Image0->Canvas->Pen->Color=clRed;}
        else
        {Image0->Canvas->Pen->Color=clBlue;}
        Image0->Canvas->LineTo(Cx+dX,Cz+dZ);
}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button1Click(TObject *Sender)
{
        Timer0->Enabled = false;
```

```
        Edit9->Text  = FloatToStr(0);
        Edit10->Text = FloatToStr(0);
        Edit11->Text = FloatToStr(0);
        Edit12->Text = FloatToStr(0);
        Edit13->Text = FloatToStr(0);
        Edit14->Text = FloatToStr(0);
        Cx=25;
        Cz=Image0->Height-25;
        Image0->Canvas->Pen->Width=5;
        Image0->Canvas->Pen->Color=clWhite;
        aRect = Rect(0,0,ClientWidth,ClientHeight);
        Image0->Canvas->FillRect(aRect);
        Image0->Canvas->
FillRect(Rect(0,0,Image0->Width,Image0->Height));
        Image0->Canvas->MoveTo(Cx,Cz);
        Panel2->Caption = "Ракета на старті!!!";
}
/*---------------------------------------------------------------*/
void __fastcall TfrmUnit01::Button2Click(TObject *Sender)
{
frmUnit01->Close();
}
/*---------------------------------------------------------------*/
```

8.2. Завдання для аудиторної роботи

На основі відомостей про предметне середовище скласти блок-схему алгоритму і написати програмний код додатку для аналізу траєкторій польоту ракети в межах земної атмосфери без врахування опору повітря.

8.3. Завдання для самостійної роботи

Уважно проаналізувати зміст теоретичної частини комп'ютерного практикуму №8 та використовуючи алгоритм і програмний код, які були підготвлені під час виконання аудиторного завдання, побудувати у середовищі Borland C++ Builder Enterprise v6.0 візуальний додаток для аналізу траєкторій польоту ракети в межах земної атмосфери без врахування опору повітря.

Передати викладачу на перевірку звіт про виконане завдання із зображенням блок-схеми алгоритму, лістінгу програмного коду та відкомпільованого виконавчого файлу запрограмованого додатку. Термін виконання завдання для самостійної роботи 2 тижні.

8.4. Питання для самоконтролю знань

- Поясніть різницю поміж локальними та глобальними змінними.
- В яких випадках слід використовувати глобальні змінні?
- Що буде виведено на екран після виконання такої програми?

```
#include <iostream.h>
void swap(int i, int j)
int temp = i; i = j; j = temp;
int main()
int a = 10, b = 3;
swap (a, b);
cout « "a = «< a « "and b = «< b;
return 0;
```

- Що таке область видимості змінної?
- Чи можна у C++ оголошувати вкладені функції?
- Які є способи передавання параметрів до функції?
- Що таке перевантаження функцій?
- Яка проблема виникне з такими перевантаженими функціями?

```
void inc(int &i)
i = i + 1;
void inc(int &i, int diff = 1)
i += diff;
```

- Знайдіть помилку у функції

```
double volume(double length, double width = 1, double height)
return length * width * height;
```

- Яка помилку припущено у такій програмі? Як її виправити?

```
# include <iostream.h>
int main()
double x = 5.2;
cout « x « ” * 2 = ” « sqr(x);
return 0;
double sqr( double x)
return x * x;
```

Рекомедована література

Обов'язкова література

[1] **Вступ до програмування мовою C++. Організація обчислень: Навчальний посібник**, *Ю.А.Бєлов, Т.О.Карнаух, Ю.В.Коваль, А.Б.Ставровський*, - Київ: Видавничо-поліграфічний центр "Київський університет", 2012. – 175 с.

[2] **Основи сучасного програмування: Підручник** *О.С.Бичков*, - Київ: Видавничо-поліграфічний центр "Київський університет", 2008. - 273 с.

[3] **Алгоритмізація та програмування: Практикум. Навчальний посібник**, *Л.І.Кублій*, — Київ: КПІ ім. Ігоря Сікорського, 2019. — 209 с.

[4] **Уроки програмування на C++**, *Переклад з Ravesli / Уроки по C++* — WEB-ресурс, 2022.

Додаткова література

[5] **C/C++ и Borland C++ Builder для студента**, *Пахомов Б.И.*, — СПб.: БХВ-Петербург, 2006. — 448 с.

[6] **Самоучитель C++ Builder**, *Культин Н.Б.*, СПб.: БХВ-Петербург, 2004. - 320 с.

[7] **C/C++. Программирование на языке высокого уровня**, *Т.А.Павловская*, СПб.: Питер, 2003. — 461 с.

[8] **C/C++. Структурное и объектно-ориентированное программирование: Практикум.**, *Павловская Т.А., Щупак Ю.А.*, — СПб.: Питер, 2011. — 352 с.

[9] **C++. Объектно-ориентированное программирование: Практикум.**, *Павловская Т.А., Щупак Ю.А.*, — СПб.: Питер, 2006. — 265 с:

[10] **Язык программирования Си. Справочник: Пер. с англ.**, *М.И.Болски*, – М.: Радио и связь, 1988. – 96 с.