

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
КАФЕДРА ЕЛЕКТРОННОЇ ТЕХНІКИ ТА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

СХЕМОКОНСПЕКТ ЛЕКЦІЙ

з навчальної дисципліни

«Комп'ютерні технології та програмування. Частина 2»

для студентів денної та заочної форм навчання
галузі знань G Інженерія, виробництво та будівництво

Дрогобич, 2025

УДК 004.42(042.3)

С 92

Схемоконспект лекцій з навчальної дисципліни «Комп'ютерні технології та програмування. Частина 2» для студентів денної та заочної форм навчання галузі знань G Інженерія, виробництво та будівництво/ уклад. Т.В. Алтухова, Е.А. Петелін. – Дрогобич: ДонНТУ, 2025. – 351 с.

Навчальна дисципліна «Комп'ютерні технології та програмування. Частина 2» є актуальною і затребуваною в контексті підготовки висококваліфікованих фахівців у галузі знань G Інженерія, виробництво та будівництво. Схемоконспект лекцій охоплює основні питання в галузі комп'ютерних технологій та дозволить забезпечити студентам необхідну допомогу під час вивчення ними ключових положень базових тем курсу. Схематична форма конспекту акцентує увагу студентів на закріпленні основних фахових знань, формуванні відповідних умінь та навичок і в цілому отриманні ними кваліфікаційних компетентностей.

Укладачі: Тетяна АЛТУХОВА, канд. техн. наук, доцент кафедри ПМІ ДВНЗ «ДонНТУ»

Едуард ПЕТЕЛІН, канд. техн. наук, доцент, доцент кафедри ЕТКІ ДВНЗ «ДонНТУ»

Рецензент: Валерій КАЛИНИЧЕНКО, канд. техн. наук, доцент, доцент кафедри ПМ ДВНЗ «ДонНТУ»

Відповідальний за випуск: Сергій КОВАЛЬОВ, канд. техн. наук, доцент, завідувач кафедри ЕТКІ ДВНЗ «ДонНТУ»

Затверджено навчально-методичним відділом ДонНТУ, протокол №6 від 29.04.2025 р.

Розглянуто на засіданні кафедри ЕТКІ ДВНЗ «ДонНТУ» протокол №9 від 03.04.2025 р.

© ДВНЗ «ДонНТУ», 2025

ЗМІСТ

ВСТУП	6
Тема №1. МАСИВИ	7
1.1 Загальне поняття масиву	8
1.2 Оголошення одновимірних масивів	10
1.3 Виконання введення-виведення одновимірних масивів	15
1.4 Програмування базових алгоритмів опрацювання одновимірних масивів	22
1.5 Опрацювання одновимірних масивів у функціях	28
1.6 Оголошення та ініціалізація багатовимірних масивів	32
1.7 Звертання до елементів багатовимірних масивів через індекси	36
1.8 Матриці як параметри функцій	37
1.9 Реалізація алгоритмів обробки багатовимірних масивів	38
1.10 Методи сортування масивів	50
Тема № 2. ЗАСТОСУВАННЯ ВКАЗІВНИКІВ ПРИ РОБОТІ З МАСИВАМИ	55
2.1 Поняття про вказівники	56
2.2 Оператори для роботи з вказівниками	58
2.3 Вирази з вказівниками	60
2.4 Динамічне виділення і вказівники	65
2.5 Вказівники на константи	67
2.6 Вказівники і масиви	68
2.7 Вказівники на символічні масиви	70
2.8 Масиви вказівників	71

<i>2.9 Вказівники на вказівники</i>	72
<i>Тема №3. СИМВОЛЬНИЙ ТИП ДАНИХ ТА РЯДКИ СИМВОЛІВ</i>	73
<i>3.1 Загальні поняття щодо символічний тип даних</i>	74
<i>3.2 Визначення рядків символів</i>	82
<i>Тема №4. МОДУЛЬНА ОРГАНІЗАЦІЯ ПРОГРАМ</i>	105
<i>4.1 Основні правила організації функцій</i>	106
<i>4.2 Визначення способів передавання параметрів до функцій</i>	117
<i>4.3 Формування параметрів зі значенням за замовчуванням та функції зі змінною кількістю параметрів</i>	127
<i>4.4 Основні поняття рекурсивних функцій</i>	134
<i>4.5 Можливі перевантаження функцій</i>	145
<i>Тема №5. ОСНОВНІ ТИПИ КОРИСТУВАЧІВ</i>	105
<i>5.1 Перейменовування типів (typedef)</i>	151
<i>5.2 Структури (struct)</i>	154
<i>5.3 Об'єднання (union)</i>	160
<i>5.3 Перерахування (enum)</i>	162
<i>5.4 Множини (Set)</i>	164
<i>Тема №6. РОБОТА З ФАЙЛАМИ</i>	169
<i>6.1 Загальне визначення поняття файлів</i>	170
<i>6.2 Робота з текстовими файлами на мові програмування C/C++</i>	172
<i>6.3 Робота з текстовими файловими потоками на мові програмування C/C++</i>	190
<i>6.4 Виконання послідовного та довільного записування та зчитування інформації у файлах</i>	195

6.5 Забезпечення опрацювання текстових файлів за допомогою дескрипторів	203
6.6 Основи роботи з бінарними файлами	212
6.7 Основи роботи з бінарними файловими потоками	221
6.8 Виконання опрацювання бінарних файлів із застосуванням дескрипторів	228
Тема №7. ОСНОВИ РОБОТИ З ДИНАМІЧНИМИ СТРУКТУРАМИ ДАНИХ	240
7.1 Загальне поняття списку	241
7.2 Визначення стеку та черги в динамічних структурах	249
7.3 Бінарні дерева	273
7.4 Загальне поняття автоматів та робота з ними	293
Тема №8. РОЗШИРЕНІ МОЖЛИВОСТІ РОБОТИ З КОНСОЛЬНИМ ВІКНОМ	310
8.1 Огляд можливостей роботи з консольним вікном	311
Тема №9. НАЛАГОДЖЕННЯ ПРОГРАМ	335
9.1 Визначення помилок компіляції	336
9.2 Попередження і підказки	342
9.3 Компонування	343
9.4 Помилки етапу виконання	344
СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ	350

ВСТУП

Сучасний стан інформаційних технологій (ІТ) характеризується швидким розвитком та трансформацією багатьох галузей, тому питання застосування комп'ютерних технологій є на даний час досить актуальним. З огляду на це, **метою** викладання даної дисципліни **є формування теоретичних знань та вмінь в розробці, налагодженню і тестуванню програм на мові програмування Сі, а також отримання практичних навичок при роботі з програмними продуктами.**

В результаті **опанування** навчальної дисципліни «Комп'ютерні технології та програмування. Частина 2» студент повинен отримати:

Знання: розширенні положення та можливості мови програмування С/С++, програмні засоби реалізації алгоритмів.

Уміння: на підставі розроблених алгоритмів засобами різних середовищ програмування та іншими засобами розробляти, тестувати та налагоджувати програмні коди, що засновані на розширених положеннях мови програмування С++; використовувати отримані знання та практичні навички з роботи в прикладних програмних засобах для програмування.

При вивченні курсу «Комп'ютерні технології та програмування. Частина 2» передбачено опанування студентами 9 тем, що поділені на декілька змістовних тематичних питань. Стисло, але вичерпне подання матеріалу забезпечить студентам можливість зосередитися на основних ключових поняттях в галузі комп'ютерних технологій та програмування та в цілому сприятиме формуванню в них необхідних для кваліфікаційного рівня «Бакалавр» знань, умінь і компетентностей.

Тема №1. МАСИВИ

Тематичний план

- 1.1 Загальне поняття масиву*
- 1.2 Оголошення одновимірних масивів*
- 1.3 Виконання введення-виведення одновимірних масивів*
- 1.4 Програмування базових алгоритмів опрацювання одновимірних масивів*
- 1.5 Опрацювання одновимірних масивів у функціях*
- 1.6 Оголошення та ініціалізація багатовимірних масивів*
- 1.7 Звертання до елементів багатовимірних масивів через індекси*
- 1.8 Матриці як параметри функцій*
- 1.9 Реалізація алгоритмів обробки багатовимірних масивів*
- 1.10 Методи сортування масивів*

1.1 ЗАГАЛЬНЕ ПОНЯТТЯ МАСИВУ

Масив – це упорядкована сукупність однотипних елементів.

Ім'я масиву (ідентифікатор) добирають за тими самими правилами, що й для змінних.

Індекси визначають місцезнаходження елемента в масиві. Індексами можуть бути лише змінні, константи чи вирази цілого типу.

Відмінності масиву від звичайних змінних:

- спільне ім'я для всіх значень;
- доступ до конкретного значення за його номером (індексом);
- можливість опрацювання у циклі.

Основні особливості структури даних (масиву):

- масив складається з елементів, які мають однаковий тип;
- елементи масиву послідовно розташовані в одній ділянці оперативної пам'яті без проміжків між елементами;
- кожен з елементів масиву має свій порядковий номер, що зветься індексом;
- нумерація елементів починається з 0;
- до елементів масиву можна звертатися використовуючи ім'я масиву і індекс;.
- масив може бути одновимірним, або багатовимірним, тобто таким у якого кожний елемент є також масивом;
- у мові C, C++ ім'я масиву зберігає адресу першого елементу цього масиву, тобто є вказівником на його початок.

1.2 ОГолоШЕННЯ ОДНОВИМІРНИХ МАСИВІВ

Одновимірний масив (вектор)

<тип_даних> <ім'я_масиву> [<розмір_масиву>]

Значення *розмір_масиву* при оголошенні масиву може бути не вказано в наступних випадках:

- при оголошенні масив ініціалізується;
- масив оголошено як формальний параметр функції;
- масив оголошено як посилання на масив, явно визначений в іншому модулі.

Звертання до елементів масиву:

<ім'я_масиву> [<значення_індексу>]

Приклад 1.1: `int A[10]` оголошує масив з ім'ям A, який містить 10 цілих чисел

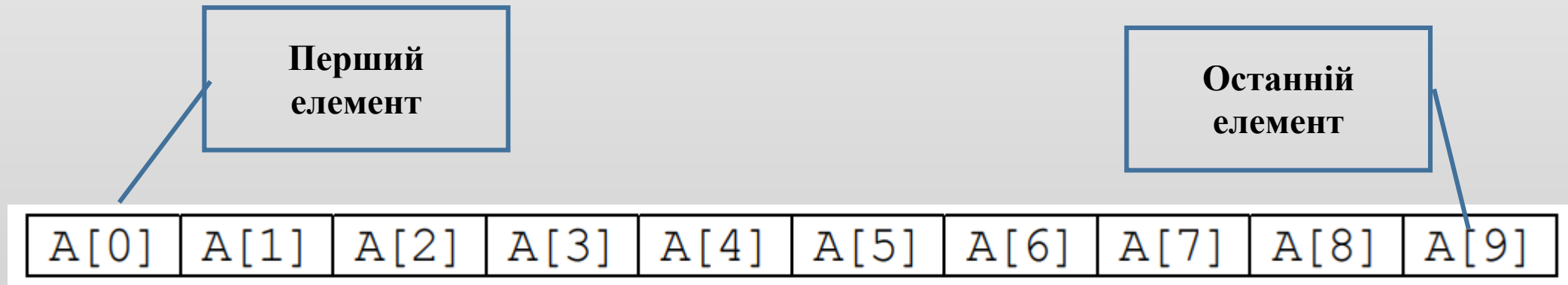


Рисунок 1.1

Приклад 1.2: для розміщення елементів одновимірного масиву `int B[5]` виділяється по 4 байти під кожен з 5-ти елементів масиву

Елементи	B[0]				B[1]				B[2]				B[3]				B[4]			
Байти	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Приклад 1.3: оголошення одновимірних масивів

```
char N[20];  
int grades[125];  
float mass[30];  
double vector[1500];
```

`N[0], N[1], ..., N[19]`

`grades[0], grades[1], ..., grades[124]`

`mass[0], mass[1], ..., mass[29]`

`vector[0], vector[1], ..., vector[1499]`

Приклад 1.4

```
int a[5]={9,33,-23,8,1};    //a[0]=9, a[1]=33, a[2]=-23, a[3]=8, a[4]=1
float b[10] = {1.5, -3.8, 10};
                        // b[0]=1.5, b[1]=-3.8, b[2]=10, b[3]=b[4]=...=b[9]=0
char s[5]="Фото";        // s[0]='Ф', s[1]='о', s[2]='т', s[3]='о', s[4]='\0'
char code[]="abc";        //code[0]='a', code[1]='b', code[2]='c', code[3]='\0'
```

Масиви в програмах можна оголошувати також зі створенням типу користувача

```
typedef <тип_даних> <ім'я_типу> [<розмір_масиву>];
<ім'я_типу> <ім'я_масиву>;
```

Приклад 1.5

```
typedef unsigned short mass[10];
mass a, b;
```

Для оголошування масивів можна використовувати *типізовані констант-масиви*

```
const <тип_даних> <ім'я_масиву> [<розмір_масиву>] =  
                                {<значення_елементів_масиву>};
```

Приклад 1.6: створення констант-масивів

```
const int arr[5]={9,3,-7,0,123}; //масив з 5-ти цілих чисел  
const float f[5]={1.5,6,8,7.4,-1.125}; //масив з 5-ти дійсних чисел  
const C[5]={15,-6,546}; //масив з 5-ти цілих чисел типу int,  
                        // де C[0]=15, C[1]=-6, C[2]=546, C[3]=0 і C[4]=0
```

1.3 ВИКОНАННЯ ВВЕДЕННЯ-ВИВЕДЕННЯ ОДНОВИМІРНИХ МАСИВІВ

Виведення елементів одновимірних масивів до різних форм за допомогою компонентів Edit, Label, Memo, ListBox та функції ShowMessage()

Виведення до компонента *Edit* елементів одновимірних масивів можна організовувати, відокремлюючи елементи пробілами (" ") чи іншими символами.

Приклад 1.7: фрагмента програми виведення масиву A

```
st = "" ;           // Очищення рядка st;  
for(i=0; i<=9; i++) // Початок циклу за індексами масиву  
                    // для накопичення в рядку значень масиву.  
{ st += FormatFloat("0.00", A[i])+" "; }  
Edit1->Text=st;     // Виведення до компонента Edit1 сформованого рядка.
```

```
for(i=0; i<=9; i++)  
    Edit1->Text = Edit1->Text+FormatFloat("0.00", A[i])+" ";
```

Виводити до компонента *Label* можна одновимірний масив, відокремлюючи його елементи пробілами (" ") чи символами переходу до нового рядка ('\n'). Виведення одновимірного масиву у рядок організовують за тими самими правилами, що і до компонента Edit, лише в програмі замість *Edit1->Text* слід записати *Label1->Caption* (приклад, *Label1->Caption=st;*). Для виведення одновимірного масиву у стовпчик слід властивість WordWrap компонента Label встановити в true замість false, що встановлено за замовчуванням.

Виведення одновимірного масиву у вікно повідомлень за допомогою функції *ShowMessage()* організовують так само, як і в попередніх прикладах, лише замість оператора присвоювання слід записати оператор виклику функції. *Приклад*, замість оператора *Edit1->Text = st;* слід записати *ShowMessage(st);*

Виводити до багаторядкового компонента **Мето** можна одновимірні масиви з якою завгодно кількістю елементів. В такому разі є доцільним використання смуги прокручування (надати властивості **ScrollBars** значення **ssBoth** чи **ssVertical**).

Приклад 1.8: фрагмента програми виведення масиву А (у стовпчик)

```
Memol.Clear();           // Очищення компонента.  
for(i=0; i<=9; i++)      // Початок циклу за індексами масиву  
                        // для почергового виведення елементів масиву.  
Memol->Lines->Add(FormatFloat("0.00", A[i]));
```

Виведення масивів до компонента **ListBox**.

Приклад 1.9: замість оператора

```
Memol->Lines->Add(FormatFloat("0.00", A[i]));
```



```
ListBox1->Items->Add(FormatFloat("0.00", A[i]));
```

Виведення елементу з масиву за допомогою оператора циклу while

У даній функції елементи масиву перебираються, поки не буде знайдено заданий елемент. Після цього усі наступні елементи зсуваються вліво на одну позицію, таким чином займаючи місце елементу, що видаляється. Розмір масиву зменшується на 1. Після цього пошук триває, поки не буде досягнутий кінець масиву.

Для перебору елементів масиву у функції використовується цикл *while*, тому що індекс елементу у разі зсуву елементів ліворуч не потрібно змінювати.

```
void delElement(int element, int ar[], int &size){  
    int i=0;  
    while(i < size){  
        if(ar[i] == element) {  
            size = size - 1;  
            for(int j = i; j<size; j++)  
                ar[j] = ar[j+1];  
        }  
        else i++;  
    }  
}
```

Рисунок 1.2

Введення елементів одновимірних масивів з різних компонентів

За допомогою компонента ***Memo*** можна вводити масиви як при виконуванні програми, так і при конструюванні форми проекту програми через вікно властивості ***Lines***.

Приклад 1.10: фрагмент програми введення значень елементів одновимірного масиву *A*

```
for (i=0; i<=9; i++) A[i]=StrToFloat (Memol->Lines->Strings[i]);
```

За допомогою компонента ***ListBox*** можна вводити масиви так само, як і з компонентом ***Memo***, лише замість властивості ***Lines*** використовувати властивість ***Items***. Компонент ***StringGrid*** також можна використовувати для введення-виведення одновимірних масивів.

Введення-виведення масиву у консолі

Виведення масиву у консолі за допомогою **cout** у стовпчик:

```
for(int i=0; i<n; i++) cout << a[i] << endl;
```

у рядок через пробіл:

```
for(int i=0; i<n; i++) cout << a[i] << "  ";  
cout << endl;
```

Введення масиву за допомогою **cin**:

```
for (int i=0; i<n; i++)  cin >> a[i];
```

Для введення-виведення символьних масивів у консолі використовуються функції **gets()** і **puts()**:

```
char s[50];  
gets(s);      // Введення символьного масиву s  
puts(s);      // Виведення символьного масиву s
```


Приклад 1.11

Масив з 7-ми цілих чисел	7-ми окремих цілих змінних
Оголошення	
<code>int a[7];</code>	<code>int a, b, c, d, e, f, g;</code>
Введення до Memo1	
<code>for(int i=0; i<7; i++) a[i]=StrToInt(Memo1->Lines-> Strings[i]);</code>	<code>a = StrToInt(Memo1->Lines->Strings[0]); b = StrToInt(Memo1->Lines->Strings[1]); c = StrToInt(Memo1->Lines->Strings[2]); d = StrToInt(Memo1->Lines->Strings[3]); e = StrToInt(Memo1->Lines->Strings[4]); f = StrToInt(Memo1->Lines->Strings[5]); g = StrToInt(Memo1->Lines->Strings[6]);</code>
Обчислення суми додатних елементів	
<code>float s=0; for(int i=0; i<7; i++) if (a[i]>0) s+=a[i];</code>	<code>if (a>0) s=a; if (b>0) s+=b; if (c>0) s+=c; if (d>0) s+=d; if (e>0) s+=e; if (f>0) s+=f; if (g>0) s+=g;</code>

1.4 ПРОГРАМУВАННЯ БАЗОВИХ АЛГОРИТМІВ ОПРАЦЮВАННЯ ОДНОВИМІРНИХ МАСИВІВ

Базові алгоритми опрацювання масивів:

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

Приклад 1.12 Розробити схему алгоритму та проект програми для введення одновимірного масиву з 8-ми дійсних чисел та обчислення суми усіх елементів масиву.

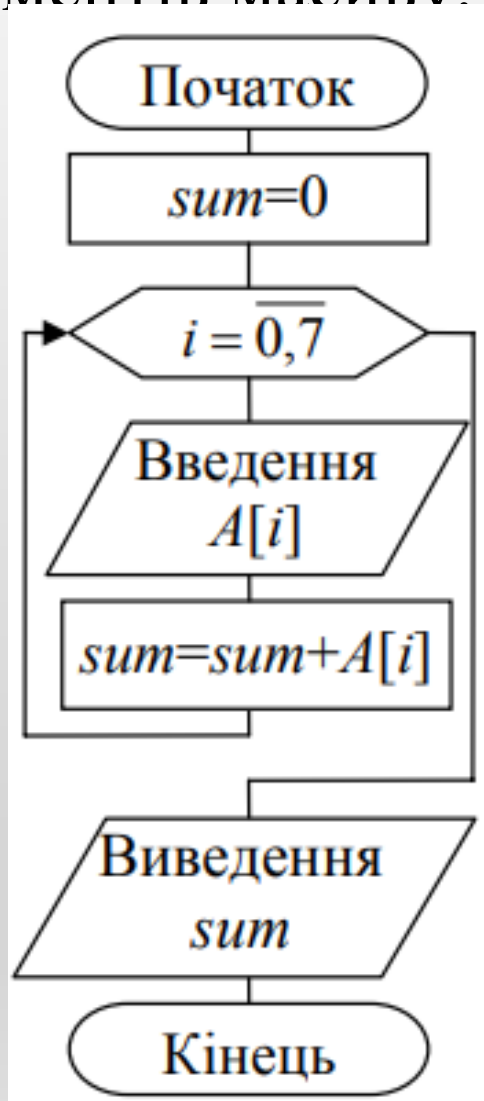


Рисунок 1.3

```

void __fastcall
 TForm1::Button1Click(TObject*Sender) // Сума
{ float A[8], sum = 0;
  for(int i = 0; i < 8; i++)
  { A[i]=StrToFloat(Memo1->Lines->Strings[i]);
    sum += A[i];
  }
  Edit1->Text = FormatFloat("0.000", sum);
}

void __fastcall
 TForm1::Button3Click(TObject*Sender) //Очищення
{ Memo1->Clear();
  Edit1->Clear();
}

void __fastcall
 TForm1::Button2Click(TObject*Sender) //Вихід
{ Close();
}
  
```

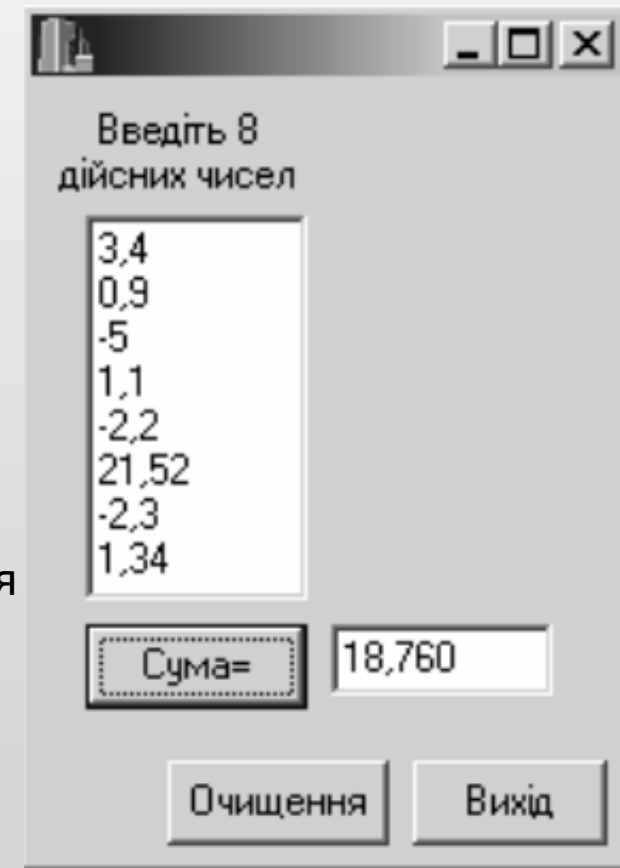
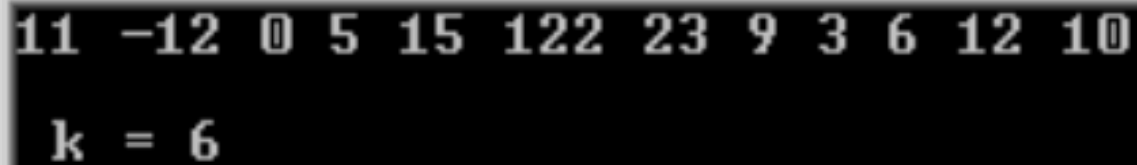


Рисунок 1.4

Приклад 1.13 Скласти консольну програму, яка вводить масив з 12-ти цілих чисел і обчислює кількість непарних елементів з проміжку $(-10, 30)$.

```
#include <conio.h>
#include <iostream.h>
int main(int argc, char* argv[])
{ int a[12],i,k=0;
  for(i=0; i<12; i++)
  { cin >> a[i];
    if(a[i]%2 != 0 && a[i] > -10 && a[i] < 30) k++;
  }
  cout << "k=" << a[i] << endl;
  getch(); return 0;
}
```



```
11 -12 0 5 15 122 23 9 3 6 12 10
k = 6
```

Рисунок 1.5

Приклад 1.14 Скласти схему алгоритму і проект програми для обчислення елементів одновимірного масиву з 15-ти елементів за формулою $A_i = \lg(i) + \operatorname{tg}(2^i)$, де $i = 1, 2, \dots, 15$, та їхнього виведення на форму, а також визначення мінімального елемента та його порядкового номера.

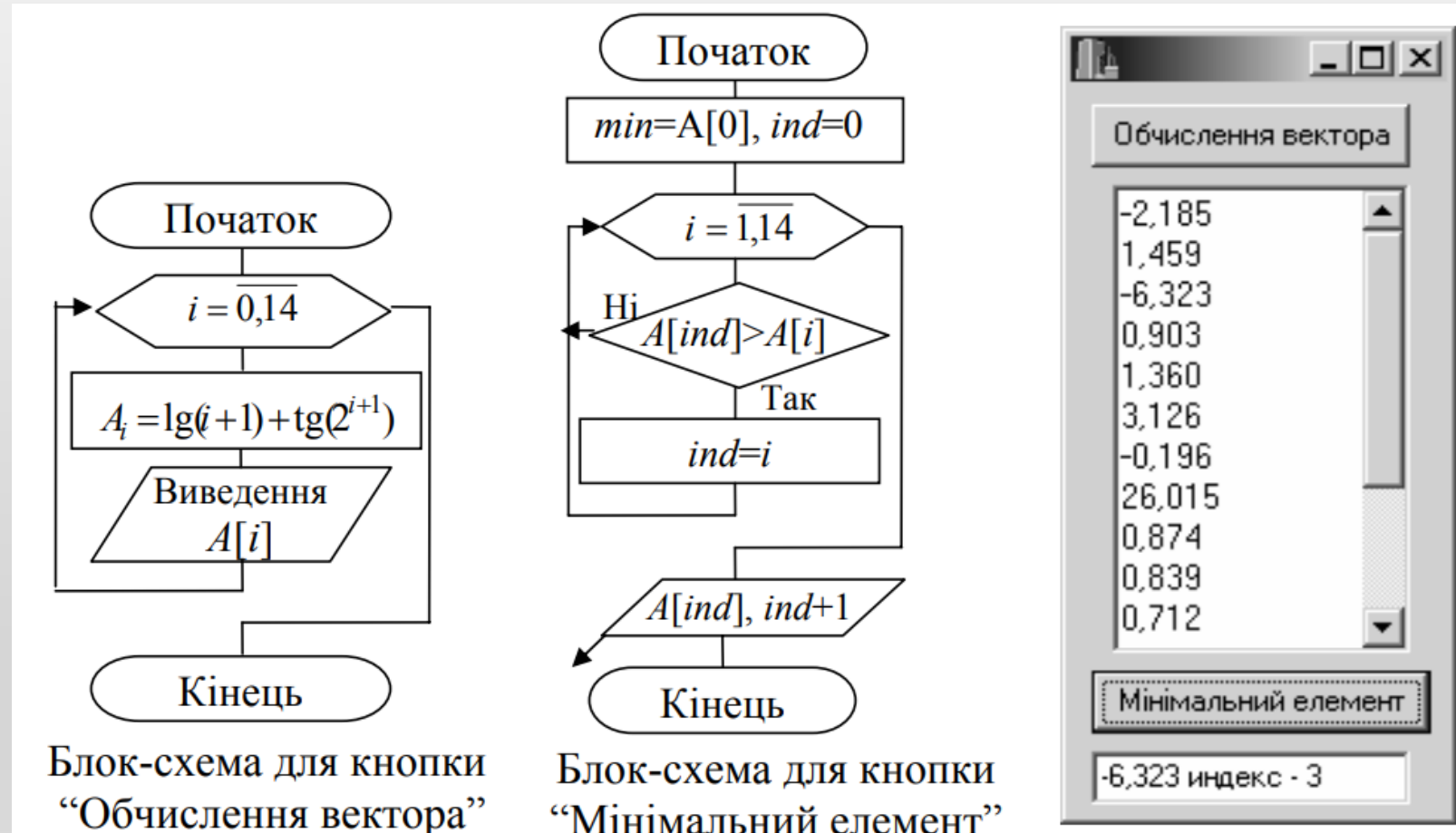


Рисунок 1.6

```

#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);
}

```

```
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);
}
```

```
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])+"индекс-" +IntToStr(ind+1);
}
```


1.5 ОПРАЦЮВАННЯ ОДНОВИМІРНИХ МАСИВІВ У ФУНКЦІЯХ

Функція — окремо організований поіменований блок команд програми, в якому розв'язується невелика і специфічна частина загального завдання.

Організація програмного проекту з розподілом великого складного завдання на набір більш простих задач і завдань, розв'язування яких виокремлено в логічні блоки — функції, дозволяє спростити читаність програми, її розуміння.

Наприклад, якщо результатом виконання функції є одне значення, яке повертається оператором **return**, то воно підставляється у точку виклику функції, тобто тип функції і є типом результату.

Способи зазначити адресу початку масиву:

```
float r (int a[10]);
```

явно зазначається кількість елементів масиву

```
float r (int a[]);
```

константний вираз у квадратних дужках є відсутній

```
float r (int *a);
```

показчик як посилення на масив, явно визначений в основній програмі

Приклад 1.15 Розробити схему алгоритму і проект програми для введення до 15-ти дійсних чисел із компонента Memo і впорядкувати їх за зростанням.

Перший спосіб:

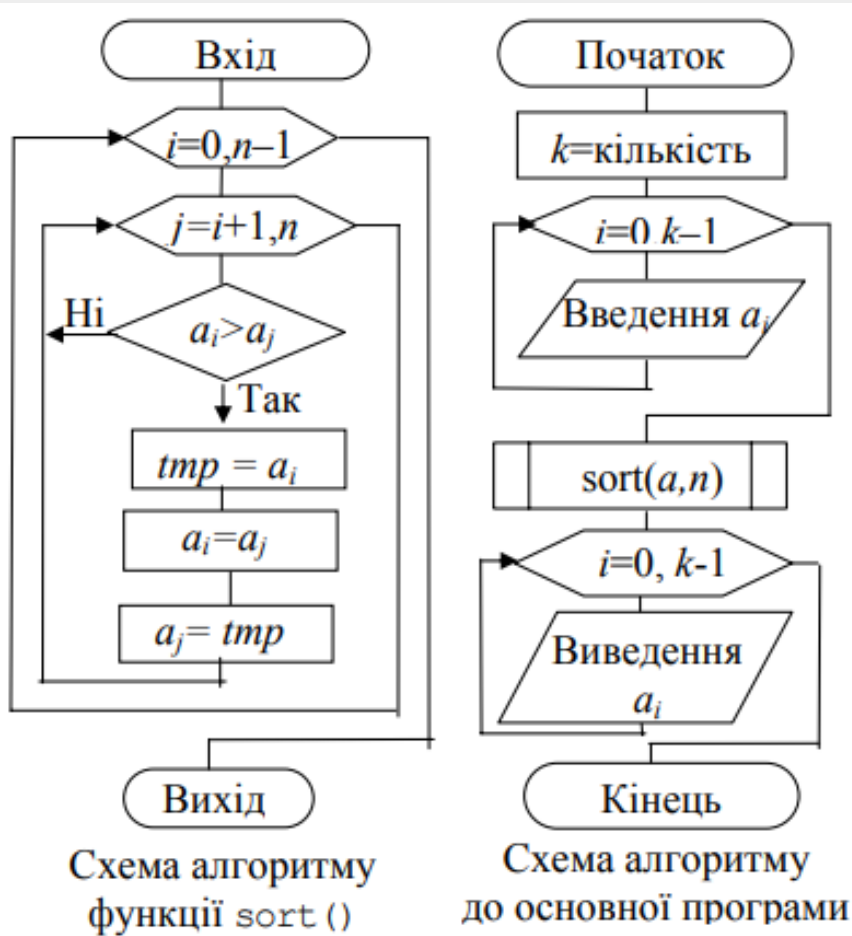


Рисунок 1.7

```

void sort(float a[], int n)
{ int i, j; // Параметри наступних циклів
  float tmp; // Тимчасова змінна для переставлянь елементів
  for(i=0; i<n-1; i++)
  for(j=i+1; j<n; j++)
  if(a[i] > a[j])
  { tmp = a[i];
    a[i] = a[j];
    a[j] = tmp;
  } }
//-----
void __fastcall TForm1::Button1Click(TObject *Sender)
{ int n,i; float a[15]; // Зчитування кількості заповнених
  рядків Memo1
  n = Memo1->Lines->Count;
  if(n>15)n=15;
  for(i=0; i<n; i++) a[i]=StrToFloat(Memo1->Lines->Strings[i]);
  sort(a, n); // Виклик функції
  Memo2->Clear();
  for(i=0; i<n; i++) Memo2->Lines->Add(FloatToStr(a[i]));
}

```

Другий спосіб:

```
void sort2(float a[], int n)
{ int i; float tmp; bool ok=1;
  while(ok)
  { ok=0;
    for(j=0; j<n-1; j++)
      if(a[j] > a[j+1])
      { tmp = a[j];
        a[j] = a[j+1];
        a[j+1] = tmp;
        ok = 1;
      }
  }
}
```

Третій спосіб:

```
void sort3(float a[], int n)
{ int i, j, imin; float tmp;
  for(i=0; i<n-1; i++)
  { imin=i;
    for(j=i+1; j<n; j++)
      if(a[imin] > a[j]) imin=j;
    tmp = a[i];
    a[i] = a[imin];
    a[imin] = tmp;
  }
}
```

1.6 ОГЛОШЕННЯ ТА ІНІЦІАЛІЗАЦІЯ БАГАТОВИМІРНИХ МАСИВІВ



1



2



3



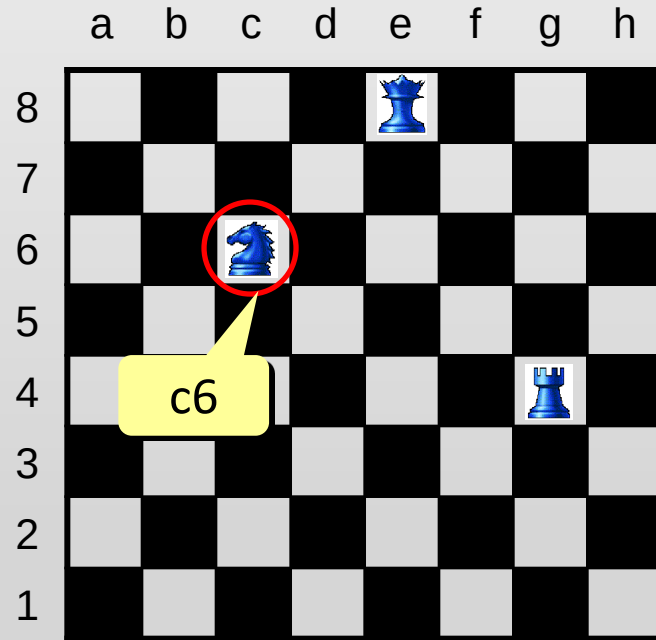
4



5



6



	0	1	2	3	4	5	6	7
7	0	0	0	0	2	0	0	0
6	0	0	0	0	0	0	0	0
5	0	0	3	0	0	0	0	0
4	0	0		0	0	0	0	0
3	0	0			0	0	4	0
2	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

A[5][2]

Матриця – це прямокутна таблиця однотипних елементів.

Матриця – це масив, в якому кожен елемент має два індекси (номер рядка і номер стовпця).

А

	0	1	2	3	4
0	1	4	7	3	6
1	2	-5	0	15	10
2	8	9	11	12	20

стовбец 2

рядок 1

комірка А [2] [3]

Рисунок 1.7

Властивості елементів матриці

- Якщо елемент рядка співпадає з номером стовпця ($i=j$), це означає, що елемент лежить на головній діагоналі матриці;
- 2. Якщо номер рядка більше номера стовпця ($i>j$), то елемент лежить нижче головної діагоналі;
- 3. Якщо номер рядка більше номера стовпця ($i<j$), то елемент лежить вище головної діагоналі;
- 4. Елемент лежить на допоміжній діагоналі, якщо його індекси задовольняють рівності: $i+j=n-1$;
- 5. Елементу, що лежить вище допоміжної діагоналі відповідає вираз: $i+j<n-1$;
- 6. Елементу, що лежить нижче допоміжної діагоналі відповідає вираз: $i+j>n-1$.

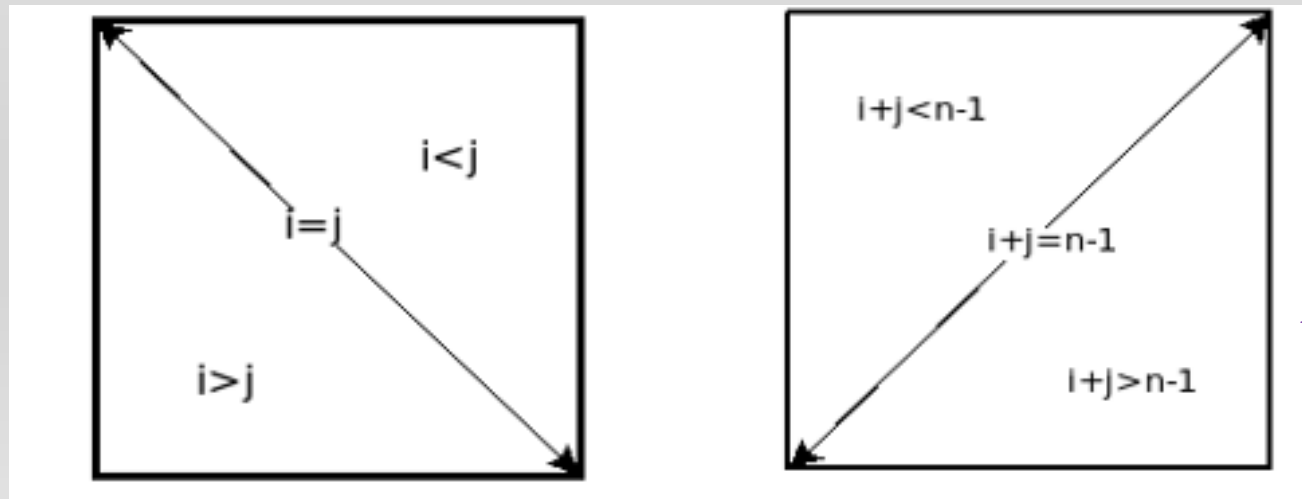


Рисунок 1.8

<тип елементів> <ім'я матриці> [<кількість рядків>] [<кількість стовпців>];

int A [10] [6];

int A [10][6] = {{2,5,10}, {6,12}, {7}};

int A [3][4] = {2, 5, 10, 3, 6, 0, 9, 4, 5, 7, 12, 1};

int A [][4] = {2, 5, 10, 3, 6, 0, 9, 4, 5, 7, 12, 1};

```
const int N = 3, M = 4; або #define N 3
                                #define M 4
```

```
int A[N][M];
```

```
float a[2][2] = {{3.2, 4.3}, {1.1, 2.2}};
```

```
char sym[2][2] = { 'a', 'b', 'c', 'd' };
```


1.7 ЗВЕРТАННЯ ДО ЕЛЕМЕНТІВ БАГАТОВИМІРНИХ МАСИВІВ ЧЕРЕЗ ІНДЕКСИ

$A[i][j] \quad i=0\dots2 \quad j=0\dots3$

$A[0][0]$ - це ім'я першого елемента, а $A[2][3]$ - ім'я останнього елемента.


 $A[0][0]=$
 $A[0][1]=$
 $A[0][2]=$
...
 $A[2][3]=$

1.8 МАТРИЦІ ЯК ПАРАМЕТРИ ФУНКЦІЙ

~~Тип_даних Ім'я [] []~~

Тип_даних Ім'я [] [Константа]

```
#define M 3
```

```
void PrintMatr(int a[ ][M], int n, int m);
```

```
void main ()  
{  
    int n, m;  
    int a[M][M];  
    ...
```

1.9 РЕАЛІЗАЦІЯ АЛГОРИТМІВ ОБРОБКИ БАГАТОВИМІРНИХ МАСИВІВ

*Введення елементів
матриці з клавіатури*

```
for ( i = 0; i < N; i ++ )  
    for ( j = 0; j < M; j ++ ) {  
        printf ( "A[%d][%d]=", i, j);  
        scanf ( "%d", &A[i][j] );  
    }
```

```
for ( j = 0; j < M; j ++ )  
    for ( i = 0; i < N; i ++ ) {
```

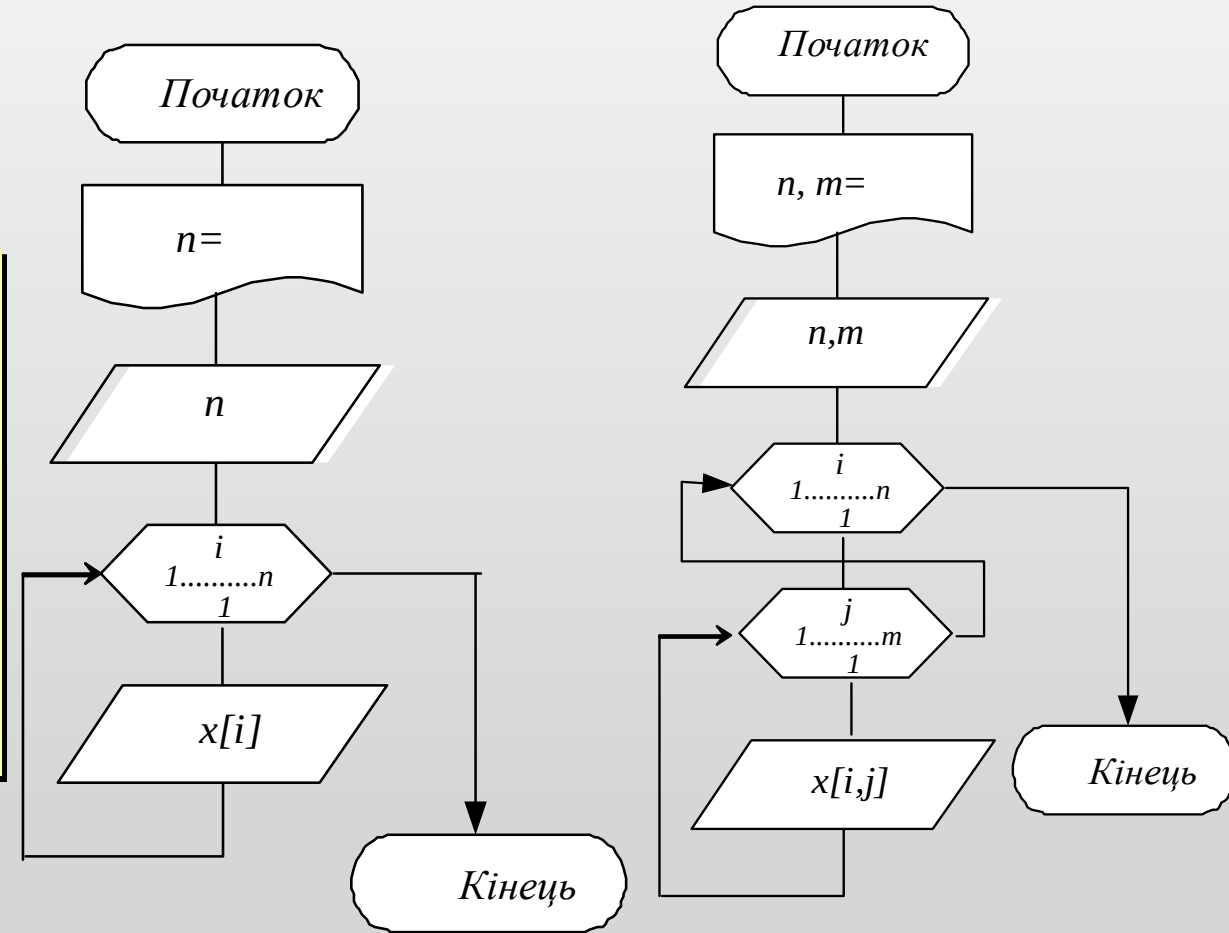


Рисунок 1.9

Формування та виведення матриць з використанням консолі

```
#define ROWS 10  
#define COLS 10  
...  
void printMatrix(int matr[][COLS], int nRow, int nCol) {  
    cout << endl;  
    for(int i = 0; i < nRow; i++) {  
        for(int j = 0; j < nCol; j++) {  
            cout << setw(6) << matr[i][j];  
        }  
        cout << endl;  
    }  
}
```

Рисунок 1.10

Тотальна обробка даних у матрицях

- заповнення матриці випадковими або іншими числами;
- пошук суми всіх елементів матриці;
- пошук максимального або мінімального елемента в матриці;
- множення матриці на число;
- отримання суми двох матриць однакового розміру.

```
#define ROWS 10
```

```
#define COLS 8
```

```
...
```

```
void matrixToZero(int matr[][COLS], int nRow, int  
nCol){
```

```
    for(int i = 0; i < row; i++)
```

```
        for(int j = 0; j < col; j++)
```

```
            matr[i][j]=0;
```

Заповнення випадковими числами

```
for ( i = 0; i < N; i ++ )  
    for ( j = 0; j < M; j ++ )  
        A[i][j] = random(25) - 10;
```

цикл по рядках

цикл по стовбцях

Виведення на екран

```
for ( i = 0; i < N; i ++ ) {  
    for ( j = 0; j < M; j ++ )  
        printf("%5d", A[i][j]);  
    printf("\n");  
}
```

виведення рядка

в тому ж рядку

перейти на
новий рядок

12	25	1	13
156	1	12	447
1	456	222	23

Приклад 1.16 Заповнити матрицю з 3 рядків і 4 стовпців випадковими числами і вивести її на екран. Знайти суму елементів матриці.

```
void main()
{
    const int N=3, M=4;
    int A[N][M], i, j, S=0;
    ... // заповнення матриці та виведення на екран
    for ( i=0; i<N; i++ )
        for ( j=0; j<M; j++ )
            S += A[i][j];
    printf("Сума елементів матриці S=%d", S);
}
```

Приклад 1.17 Знайти суму елементів, що стоять на головній діагоналі і нижче її.

рядок 0: $A[0][0]$

рядок 1: $A[1][0] + A[1][1]$

...

рядок i : $A[i][0] + A[i][1] + \dots + A[i][i]$

```
S = 0;  
for ( i = 0; i < N; i ++ )  
    for ( j = 0; j <= i; j ++ )  
        S += A[i][j];
```

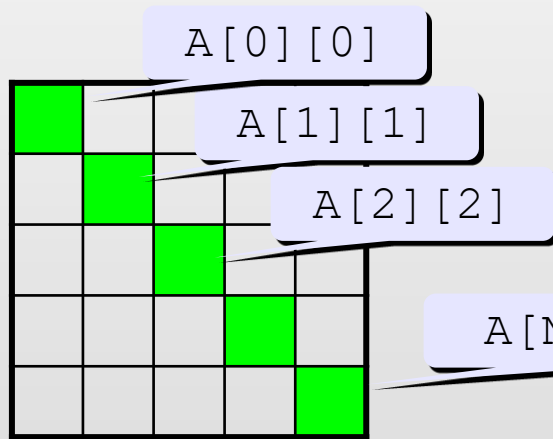
цикл по всім рядкам

Додаємо необхідні
елементи рядка i

Вибіркова обробка матриць

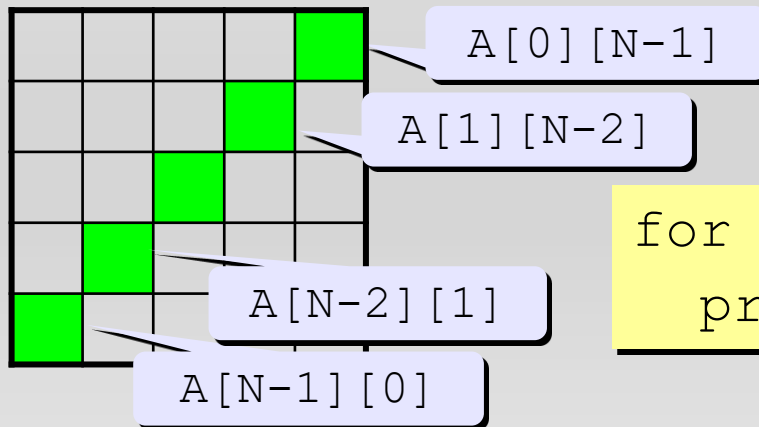
- елементи головної діагоналі квадратної матриці;
- елементи допоміжної діагоналі квадратної матриці;
- елементи деякого стовпця або рядка матриці;
- елементи матриці, розташовані по її кромці;
- елементи квадратної матриці, розташування яких відповідає розташуванню білих або чорних клітин шахової дошки.

Приклад 1.18 Вивести на екран головну діагональ квадратної матриці з N рядків і N стовпців.



```
for ( i = 0; i < N; i ++ )  
    printf ( "%5d", A[i][i] );
```

Приклад 1.19 Вивести на екран допоміжну діагональ.
сума номерів рядка і стовбця : $i+j=n-1$
 $N-1$



```
for ( i = 0; i < N; i ++ )  
    printf ( "%5d", A[i][ N-1-i ] );
```

Перестановки елементів матриці

- транспонування матриці (поворот навколо головної діагоналі);
- поворот навколо допоміжної діагоналі;
- поворот навколо горизонтальної осі;
- поворот навколо вертикальної осі.

Таблиця 1.1 –

Лінія симетрії у повороті	Зовнішній цикл, за рядками (індекси рядка)	Внутрішній цикл, за елементами рядка (індекси стовпця)	Індекси елементів	
			поточні	симетричні
Головна діагональ	від $i=1$; $i < n$	від $j = 0$; $j < i$	[i, j]	[j, i]
Допоміжна діагональ	від $i=0$; $i < n - 1$	від $j=0$; $j < n - 1 - i$	[i, j]	[n-1-j, n-1-i]
Горизон- тальна вісь	від $i=0$; $i < nRow/2$	від $j=0$; $j < nCol$	[i, j]	[nRow - 1 – i, j]
Вертикаль- на вісь	від $i=0$; $i < nRow$	від $j = 0$; $j < nCol / 2$	[i, j]	[i, nCol - 1- j]

Приклад 1.20 Перестановка рядків або стовпців. У матриці з N рядків і M стовпців переставити 1-ий і 3-ий рядки .

	j				
					A[1][j]
1	1	2	5	2	1
	↕	↕	↕	↕	↕
3	7	3	1	3	7
					A[3][j]

```
for ( j = 0; j < M; j ++ ) {  
    c = A[1][j];  
    A[1][j] = A[3][j];  
    A[3][j] = c;  
}
```

Приклад 1.21 До 3-го стовпця додати 6-й.

```
for ( i = 0; i < N; i ++ )  
    A[i][3] += A[i][6];
```

Видалення та вставка елементів матриці

Видалення стовпця або рядка матриці не можливо без порушення прямокутної структури

При видаленні потрібно зрушити наступні рядки вліво або стовпці вгору, на місце видаленого компонента, і зменшити відповідний розмір.

При вставці потрібно зрушити наступні рядки вправо або стовпці вниз, звільнивши місце для компонента, що вставляється, і збільшити відповідний розмір матриці.

Сортування елементів матриці

- сортування кожного рядка або кожного стовпця незалежно від інших;
- сортування однієї з діагоналей матриці;
- сортування рядків або стовпців матриці за значенням першого елементу групи;
- сортування рядків матриці по сумі елементів рядка;
- сортування стовпців за сумою елементів стовпця.

1.10 МЕТОДИ СОРТУВАННЯ МАСИВІВ

Сортування масиву – це процес перестановки елементів масиву з метою розміщення елементів масиву в певному порядку.

Методи сортування масивів:

- сортування методом вибору.
- сортування методом обміну (метод бульбашки);
- сортування методом вставки

Сортування методом вибору

Алгоритм сортування елементів масиву у зростаючому порядку за методом вибору можна описати так:

1. Серед усіх елементів масиву, починаючи з першого, шукають мінімальний елемент.
2. Знайдений мінімальний елемент міняють місцями з першим елементом.
3. Переглядають масив від другого елементу, і знаходять мінімальний серед цих елементів.
4. Знайдений елемент міняють місцями з другим елементом.
5. Далі те саме з третім елементом, і так далі до останнього елементу.

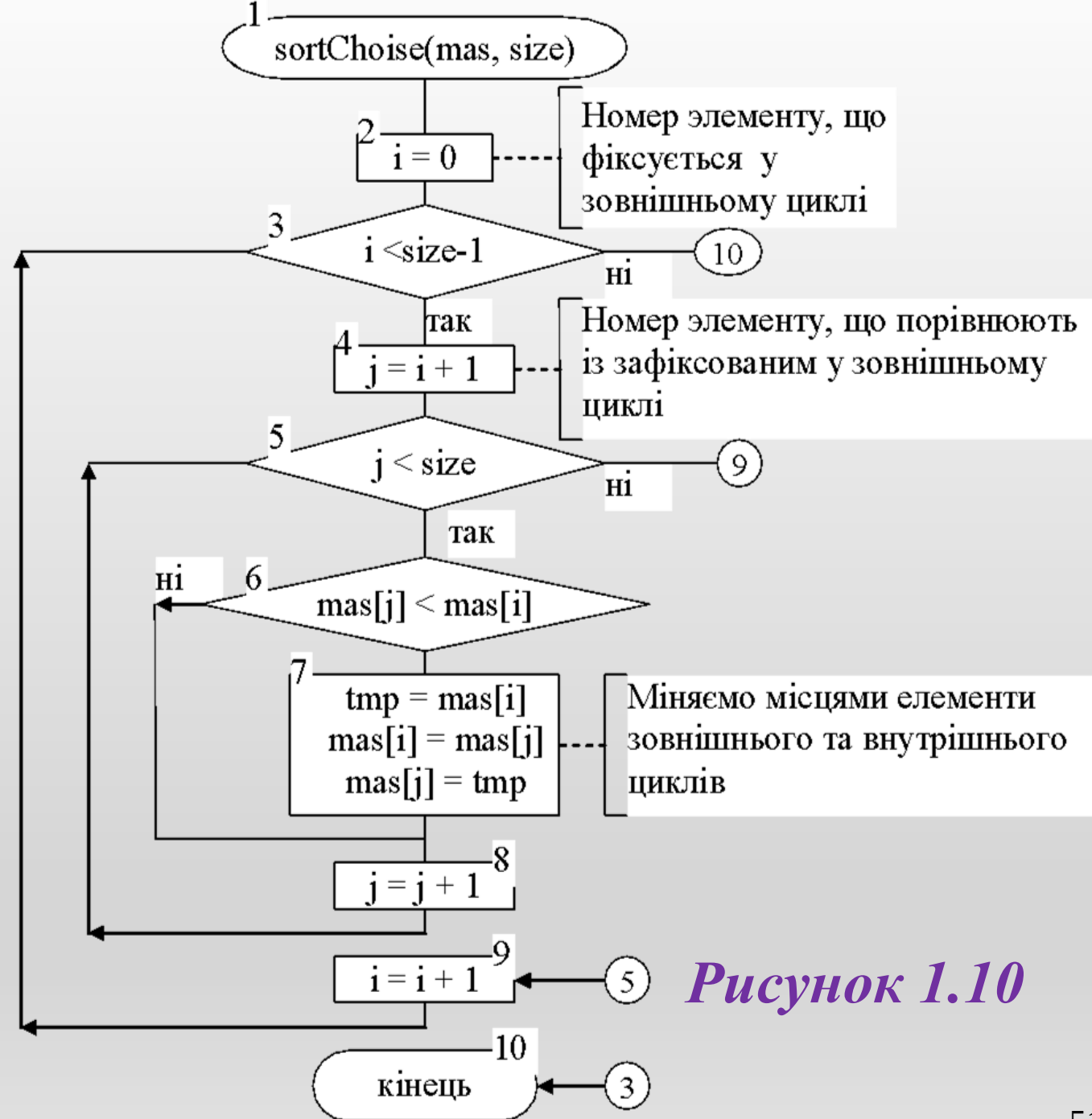


Рисунок 1.10

Функція сортування елементів масиву у зростаючому порядку за методом вибору

```
void sortChoice(int mas[], int size){  
    for(int i=0; i<size-1;i++){  
        for(int j = i + 1; j<size; j++) {  
            if (mas[j] < mas[i] ){  
                int tmp = mas[i];  
                mas[i] = mas[j];  
                mas[j] = tmp;  
            }  
        }  
    }  
}
```

Рисунок 1.11

Сортування методом обміну (метод бульбашки)

```
void sortBubl(int ar[], int size){  
    int last=size; bool ok;  
    do{  
        last=last-1;  
        ok = true;  
        for(int i= 0; i<last; i++){  
            if(ar[i] > ar[i+1 ]){  
                int x = ar[i];  
                ar[i] = ar[i+1];  
                ar[i+1] = x;  
                ok = false;  
            }  
        }  
    }while(!ok);  
}
```

Функція сортування елементів масиву у зростаючому порядку за методом обміну

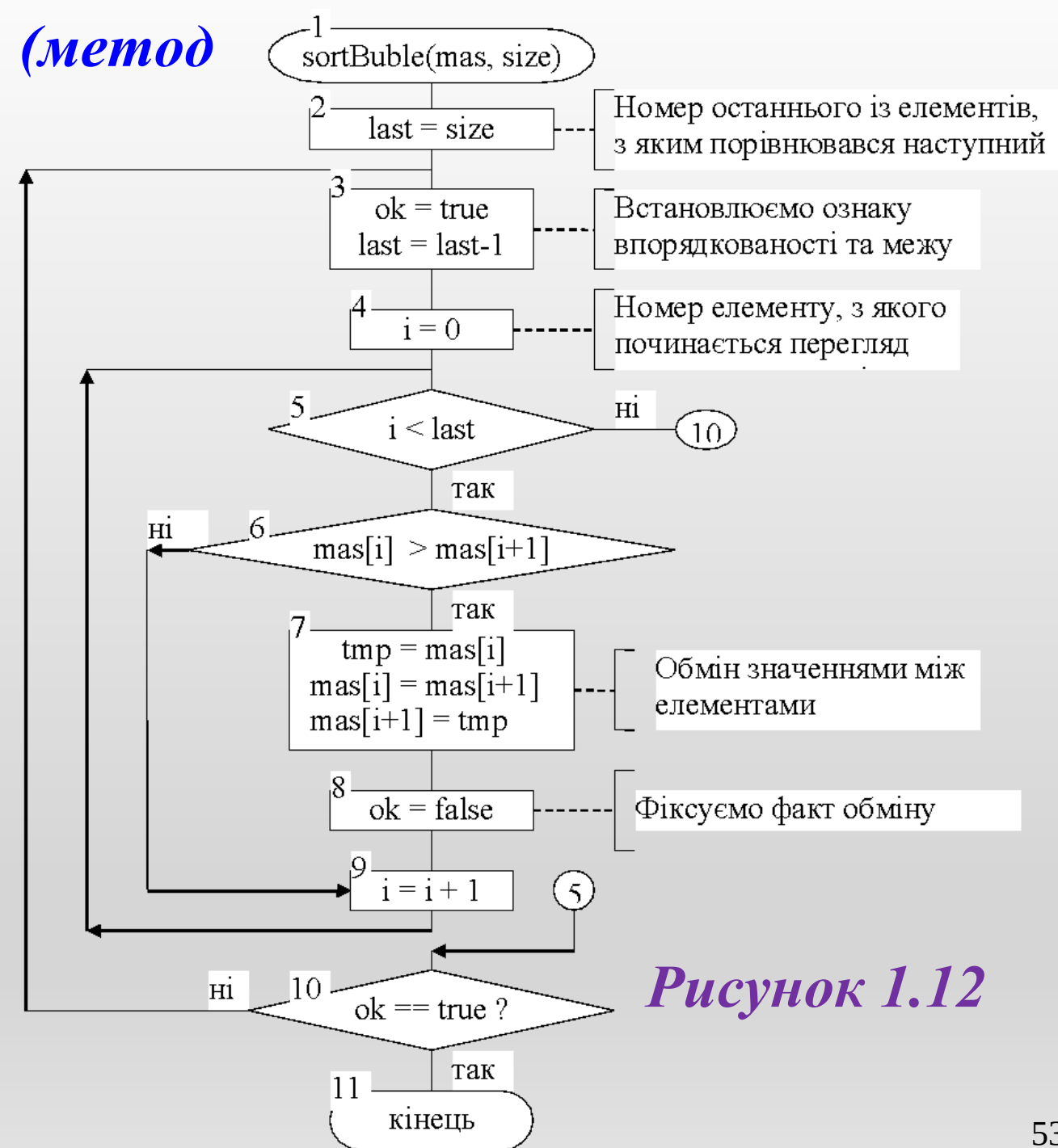


Рисунок 1.12

Сортування методом вставки

```
void sortInsert(int ar[], int size){  
    for(int i=1; i<size; i++){  
        //Порівнюємо елементи на межі між частинами масиву  
        if(ar[i] < ar[i-1]){  
            int buf = ar[i]; // Порядок порушено, переносимо i-й елемент у буфері  
            // Починаємо зсуви праворуч на місце i-го елементу  
            int j = i; // j – індекс вільного місця  
            do{  
                ar[j]=ar[j-1]; // зсув праворуч  
                j=j-1;  
                // поки ліворуч число більше, або дійшли до початку масиву  
            }while (j != 0 && buf<ar[j-1]);  
            //Вставляємо елемент, що мав номер i на нове місце з індексом j  
            ar[j] = buf;  
        }  
    }  
}
```

Функція сортування елементів масиву у зростаючому порядку за методом вставки

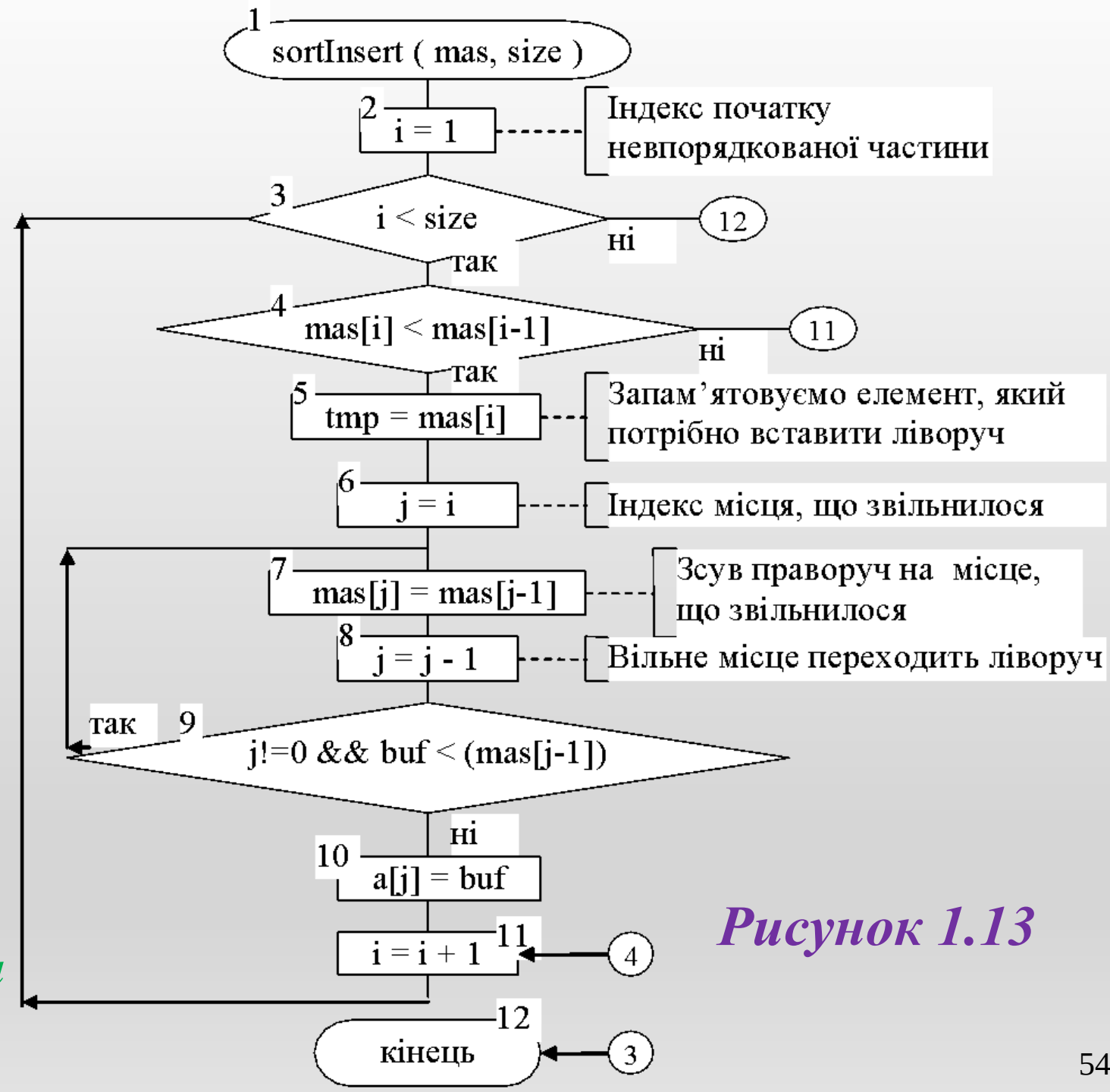


Рисунок 1.13

Тема № 2. ЗАСТОСУВАННЯ ВКАЗІВНИКІВ ПРИ РОБОТІ З МАСИВАМИ

Тематичний план

- 2.1 Поняття про вказівники***
- 2.2 Оператори для роботи з вказівниками***
- 2.3 Вирази з вказівниками***
- 2.4 Динамічне виділення і вказівники***
- 2.5 Вказівники на константи***
- 2.6 Вказівники і масиви***
- 2.7 Вказівники на символічні масиви***
- 2.8 Масиви вказівників***
- 2.9 Вказівники на вказівники***

2.1 ПОНЯТТЯ ПРО ВКАЗІВНИКИ

Основні причини ефективного застосування вказівників в C/C++:

1. Вказівники надають можливість функціям модифікувати передані аргументи
1. Вказівники використовують для підтримання системи динамічного виділення пам'яті
1. Використання вказівників може підвищити ефективність роботи деяких підпрограм
1. Вказівники використовуються для підтримки таких структур даних як зв'язані списки і бінарні дерева

	Адреса комірки пам'яті	Вміст комірки	
p	1000	1004	
q	1002	134	
num	1004	255	

Одна змінна вказує на іншу змінну

```
тип* ім'я //один варіант;
тип *ім'я //другий варіант.
```

2.2 ОПЕРАТОРИ ДЛЯ РОБОТИ З ВКАЗІВНИКАМИ

*Спеціальні оператори для роботи з
вказівниками – * та &*

Оператор & – це унарний оператор, який повертає адресу операнда.

```
p = &num;
```

*Оператор ** – це унарний оператор, який повертає значення змінної, яка знаходиться за вказаною адресою.

```
q = *p
```

Приклад 2.1: програми, яка демонструє сказане вище

```
#include <stdio.h>
int main(void)
{
    int* p;
    int q, num;

    num = 255; //присвоєно
255 num
    p = &num;    //p отримує
адресу num
    q = *p; //q присвоюється
255 з використанням p
    printf("%d",
q); //виводить 255
    return 0;
}
```

```
#include <stdio.h>

int main(void)
{
    double x, y;
    int* p;
    x = 100.123;           //x
присвоєно 100.123
    p = &x; //p отримує адресу x
— неправильно
    y = *p; //y присвоюється
100.123 з використанням p
    printf("%f", y);
    //неправильно
    return 0;
}
```


2.3 ВИРАЗИ З ВКАЗІВНИКАМИ

Присвоєння вказівників

```
#include <stdio.h>

int main(void)
{
    int x;
    int *p1, *p2;
    p1 = &x;      //p1 отримує адресу x
    p2 = p1;      //p2 присвоєно адресу x
    використавши p1
    printf("Адреса p1: %p, адреса p2: %p", p1, p2);
    //виводить адресу, яка зберігається в p1,p2
    return 0;
}
```

Арифметичні дії з вказівниками

Застосовуються тільки дві арифметичні операції: *додавання* до них *цілих чисел* і *віднімання*.

Приклад 2.2:

p1 — це вказівник на ціле, який містить значення 1004. Нехай ціле має довжину 2 байти. Після виразу

p1++;

вміст p1 стане 1006, а не 1005.

p1 = p1 + 9;

Порівняння вказівників

Можна порівнювати з двома вказівниками.

Наприклад 2.3

```
if (p < q)
    printf ("p вказує на меншу адресу в пам'яті ніж q\n");
```

Стек — це список побудований за принципом “першим прийшов — останнім вийшов”. Для створення стеку необхідно дві функції **push()** та **pop()**. Стек містить масив **stack**, в якому зберігається STACKSIZE елементів. Змінна **tos** містить адресу пам'яті вершини стеку і використовується для запобігання переповненню і вичерпування стеку.

```

#include <stdio.h>
#include <stdlib.h>

#define STACKSIZE 50

void push (int i);
int pop(void);

int *p1, *tos, stack[STACKSIZE];

int main(void)
{
    int value;
    p1 = stack;
        //присвоє p1 початок стеку
    tos = p1;                                //tos
містить вершину стеку
    do {
        printf("Enter a number (-1 to
quit, 0 to pop): ");
        scanf("%d", &value);
        if (value!=0) push (value);
        else printf ("this is it %d\n",
pop());
    } while(value!=-1);
    return 0;

```

```

}
void push(int i)
{
    p1++;
    if (p1==(tos + STACKSIZE)) {
        printf("stack overflow");
        exit(1);
    }
    *p1 = i;
}
int pop(void)
{
    if (p1==tos) {
        printf("stack underflow");
        exit(1);    }
    p1 --;
    return *(p1+1);
}

```

Черга — це лінійний список, побудований за принципом “першим прийшов — першим вийшов” (звичайна черга чекання біля каси тощо).

При реалізації черги в масиві **d** довжиною **M**, її перший елемент розміщується на початку **d**. Для фіксації значень індексів початку і кінця черги використовуються цілі змінні **a** і **b** з початковими значеннями **a=0**; **b=-1**. Вилучення елемента з непустиї черги здійснюється оператором $v = d[a++]$; додавання елемента в чергу — оператором **d[++b]=v**; . При роботі слід перевіряти, чи черга не стала порожньою: **a>b**; та чи ми не досягли кінця масиву: **b=M-1**

2.4 ДИНАМІЧНЕ ВИДІЛЕННЯ І ВКАЗІВНИКИ

Виділяють 4 області: *код програми, глобальні дані, стек і купа*.

Купа (heap) – це область вільної пам'яті, якою можна керувати через функції динамічного виділення **malloc()** і **free()**.

Функція **malloc()** виділяє пам'ять і повертає вказівник на початок виділеного блоку.

Функція **free()** повертає раніше виділену пам'ять в купу для повторного використання. Прототипи функцій **malloc()** і **free()** :

```
void *malloc(size_t кількість_байт);  
void free(void *p);
```

Використовують заготовочний файл **stdlib.h**.

Фрагмент коду виділяє 25 байт пам'яті:

```
char *p;  
p = (char *) malloc(25);
```

Виділення пам'яті і тестування на коректність вказівника:

```
int *p;  
  
if((p = (int *) malloc(100)) == NULL) {  
    printf("Out of memory.\n");  
    exit(1); }
```

2.5 ВКАЗІВНИКИ НА КОНСТАНТИ

Модифікатор **const** можна використовувати для захисту об'єкта від можливої його зміни функцією. Тобто, коли вказівник передається у функцію, функція може модифікувати оригінал змінної на яку вказує вказівник. Якщо вказівник визначений як `const`, то функція не зможе модифікувати об'єкт.

```
//це неправильно
void sp_to_dash(const char *str)
{
    while (*str) {
        if (*str == ' ') *str = '-';
        //неможливо виконати
        printf ("%c", *str);
        str ++;
    }
}
```

```
#include <stdio.h>
void sp_to_dash(const char
*str);
int main(void)
{
    sp_to_dash("this is a
test");
    return 0;
}
void sp_to_dash(const char
*str)
{
    while (*str) {
        if (*str == ' ') printf
("%c", '-');
        else printf ("%c", *str);
        str ++;
    }
}
```


2.6 ВКАЗІВНИКИ І МАСИВИ

```
char str[80], *p1;  
p1 = str;
```



Перший елемент масиву

```
str[4], або *(p1+4)
```

Обидва оператори повертають п'ятий елемент.

Використання масиву

```
int puts(const char *s)
{
    register int i;
    for(t=0; s[t];
    ++t) putchar(s[t]);
    return 1;
}
```

Використання вказівників

```
int puts(const char *s)
{
    while(*s) putchar(*s++);
    return 1;
}
```

2.7 ВКАЗІВНИКИ НА СИМВОЛЬНІ МАСИВИ

Версія із стандартної бібліотеки функції `strcmp()`, в якій використовуються вказівники

```
//Використання вказівників
int strcmp(const char *s1, const char *s2)
{
    while(*s1)
        if(*s1-*s2)
            return *s1-*s2;
        else {
            s1++;
            s2++;
        }
    return 0; //рівність
}
```

2.8 МАСИВИ ВКАЗІВНИКІВ

Для оголошення масиву цілочисельних вказівників з десяти елементів слід написати

```
int *x[10];
```

Для присвоєння адреси цілочисельної змінної Cpp третьому елементу масиву слід написати:

```
x[2] = &Cpp;
```

Для отримання значення цілочисельної змінної Cpp слід написати:

```
*x[2];
```

Приклад 2.4: функція, яка отримує масив

```
void display_array(int *q[])
{
    int t;

    for (t=0; t<10; t++)
        printf("%d", *q[t]);
}
```

2.9 ВКАЗІВНИКИ НА ВКАЗІВНИКИ

```
type** PointerToPoinetrName;
```

```
int nNum = 1;
int* pnNum = &nNum;
int ** ppnNum = &pnNum;
cout << nNum << "\n";
    // виводить 1, використовуючи змінну nNum
cout << *pnNum << "\n";
    /* виводить 1, використовуючи змінну pnNum */
cout << **ppnNum << "\n";
    /* виводить 1, використовуючи вказівник на
вказівник*/
```

```
int nNum = 1;
int **ppnNum = &&nNum;
```

Тема №3. СИМВОЛЬНИЙ ТИП ДАНИХ ТА РЯДКИ СИМВОЛІВ

Тематичний план

3.1 Загальні поняття щодо символний тип даних

3.2 Визначення рядків символів

3.1 ЗАГАЛЬНІ ПОНЯТТЯ ЩОДО СИМВОЛЬНИЙ ТИП ДАНИХ

До символів відносять:

- великі й малі латинські літери,
- великі й малі літери кирилиці,
- цифри,
- знаки арифметичних дій ('+', '-', '*', '/', '='),
- пробіл,
- розділові знаки ('.', ',', ';', ':', '!', '?', '—'),
- службові символи, що відповідають клавішам.

В С значення символічних констант записуються у одинарних лапках: *'3', 'f', '+', %'*.

Таблиця 3.1 – ASCII-кодів

Бінарне	Дес.	Шіст.	Графічне	Бінарне	Дес.	Шіст.	Графічне	Бінарне	Дес.	Шіст.	Графічне
0010 0000	32	20	пропуск (sp)	0100 0000	64	40	@	0110 0000	96	60	`
0010 0001	33	21	!	0100 0001	65	41	A	0110 0001	97	61	a
0010 0010	34	22	"	0100 0010	66	42	B	0110 0010	98	62	b
0010 0011	35	23	#	0100 0011	67	43	C	0110 0011	99	63	c
0010 0100	36	24	\$	0100 0100	68	44	D	0110 0100	100	64	d
0010 0101	37	25	%	0100 0101	69	45	E	0110 0101	101	65	e
0010 0110	38	26	&	0100 0110	70	46	F	0110 0110	102	66	f
0010 0111	39	27	'	0100 0111	71	47	G	0110 0111	103	67	g
0010 1000	40	28	(	0100 1000	72	48	H	0110 1000	104	68	h
0010 1001	41	29	)	0100 1001	73	49	I	0110 1001	105	69	i
0010 1010	42	2A	*	0100 1010	74	4A	J	0110 1010	106	6A	j
0010 1011	43	2B	+	0100 1011	75	4B	K	0110 1011	107	6B	k
0010 1100	44	2C	,	0100 1100	76	4C	L	0110 1100	108	6C	l
0010 1101	45	2D	-	0100 1101	77	4D	M	0110 1101	109	6D	m
0010 1110	46	2E	.	0100 1110	78	4E	N	0110 1110	110	6E	n
0010 1111	47	2F	/	0100 1111	79	4F	O	0110 1111	111	6F	o
0011 0000	48	30	0	0101 0000	80	50	P	0111 0000	112	70	p
0011 0001	49	31	1	0101 0001	81	51	Q	0111 0001	113	71	q
0011 0010	50	32	2	0101 0010	82	52	R	0111 0010	114	72	r

Ескейп-послідовність – це спеціальна комбінація символів, яка розпочинається зі зворотної скісної риси і записується в одинарних лапках.

Таблиця 3.2 – Деякі поширені у застосуванні ескейп-послідовності

Символьне подання	Опис
\n	символ переведення курсора на початок наступного рядка
\r	переведення каретки
\t	символ переведення курсора на наступну позицію табуляції (відповідає клавіші <Tab>)
\b	символ вилучення попереднього символу перед курсором (відповідає клавіші <BackSpace>)
\a	символ звукового сигналу системного динаміка
\\	символ \ (зворотна скісна риса)
\?	символ ? (знак запитання)
\'	символ ' (одинарні лапки)
\"	символ " (подвійні лапки)
\0	нуль-символ
\0xВісімкове число	символ, код якого зазначено у вісімковій системі числення
\0xШістнадцяткове число	символ, код якого зазначено у шістнадцятковій системі числення

char <ім'я символного рядка> [<кількість байтів >] ;

Приклад 3.1

```
char c, s, g;
```

```
char c=115;
```

Беззнакова модифікація *unsigned char*, дані даного мають значення у діапазоні 0 ... 255.

Приклад 3.2

```
unsigned char c;
```

Приклад 3.4: результати додавання

```
char c='A';  
char c1=c+5; // c1 = 'F'  
char c2=c+32; // c2 = 'a'  
char c3=c-10; // c2 = '7'
```

Таблиця 3.3 – Функції для роботи з символами

<i>Функція</i>	<i>Призначення</i>
tolower()	повертає символ у нижньому регістрі
toupper()	повертає символ у верхньому регістрі
isalnum()	латинських літер та цифр ('A' – 'Z', 'a' – 'z', '0' – '9')
isalpha()	латинських літер ('A' – 'Z', 'a' – 'z')
iscntrl()	керувальних символів (з кодами 0...31 та 127)
isdigit()	цифр ('0' – '9') isgraph() видимих символів, тобто не є відповідним до клавіш, тощо
islower()	латинських літер нижнього регістру ('a' – 'z')
isprint()	друкованих символів (isgraph() + пробіл)
isupper()	літер верхнього регістру ('A' – 'Z')
ispunct()	знаків пунктуації
isspace()	символів-роздільників

Алгоритми оброблення символьних змінних

- Щоб визначити код символу, треба значення цього символу присвоїти цілій змінній. І, навпаки, щоб дізнатися, який символ відповідає певному числу, слід це число присвоїти символу. *Наприклад*, після виконання команд

```
int x;  
char c='n';  
x=c;
```

- Визначити, чи є символ с цифрою, можна двома способами:
 - ❖ перевірити його належність до символьного проміжку від '0' до '9' за допомогою умови

```
if (c>='0' && c<='9') . . .
```

- ❖ застосувати функцію isdigit() для перевірки належності символу до множини цифр

```
if (isdigit (c)) . . .
```

➤ Дізнатися, чи є символ с великою латинською літерою, можна теж двома способами:

❖ перевірити його належність до символьного проміжку від 'A' до 'Z' за допомогою умови

```
if (c>='A' && c<='Z') . . .
```

❖ застосувати функцію isupper() для перевірки належності символу до множини великих латинських літер

```
if (isupper (c)) . . .
```

➤ Перевірка, чи є символ с латинською літерою

```
if (c>='A' && c<='Z' || c>='a' && c<='z') .
```

```
if (isalpha (c)) . . .
```

➤ Для перевірки, чи є символ с малою латинською літерою

```
if (c>='a' && c<='z') . . .
```

```
if (islower(c)) . . .
```

➤ Для перетворення малої латинської літери с на велику

```
c=toupper (c) ;
```

```
char c=c-32;
```

Приклад 3.5: ввести один символ та визначити його код в таблиці ASCII.

```
#include <iostream.h>
int main()
{ char c; int k;
  cout<<"Введіть символ: ";
  cin>>c; // Введення значення символу
  k=c;
  cout<<"Код введенного символу: "<<k<<endl;
  cin.get();
  return 0;
}
```

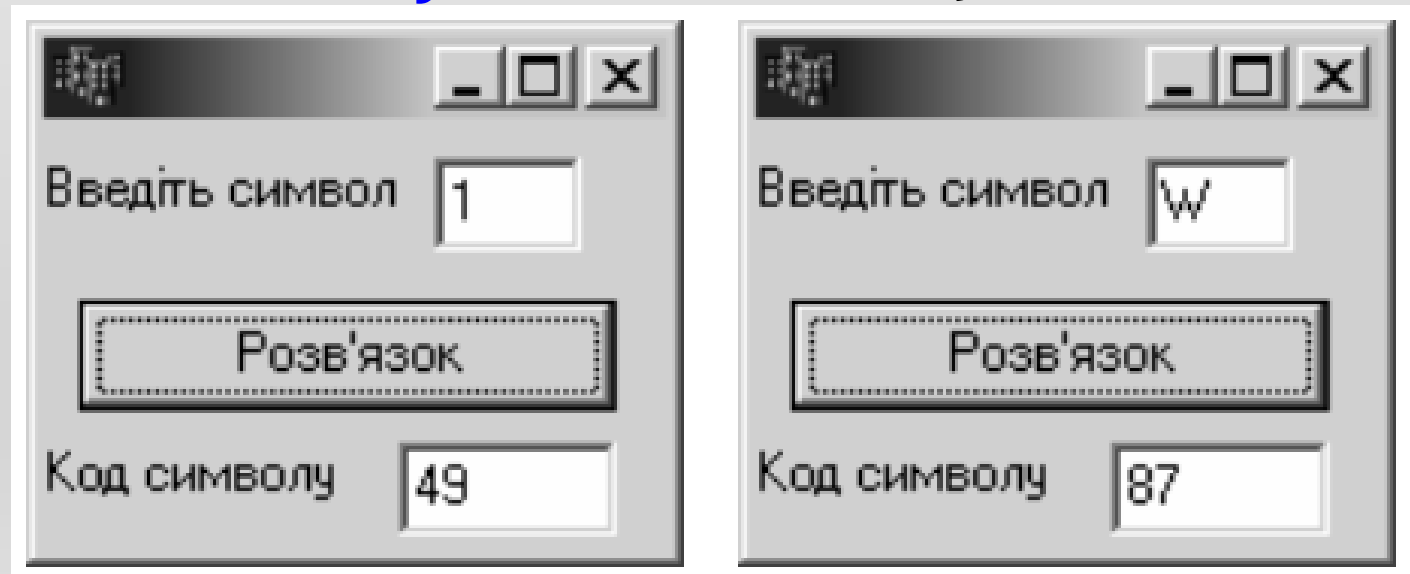


Рисунок 3.1

3.2 ВИЗНАЧЕННЯ РЯДКІВ СИМВОЛІВ

Масиви символів

Відмінною рисою символьного масиву є те, що в ньому насправді може бути менше символів, аніж зазначено при оголошенні.

Довжина динамічного рядка може задаватися змінною з визначеним заздалегідь значенням.

Довжина статичного рядка має задаватися лише константою.

Способи оголошення

```
char *s;           // Оголошення покажчика на перший символ рядка;  
                  // пам'ять під сам рядок не виділяється.  
  
char ss[15];       // Оголошення рядка ss з 14-ти символів;  
                  // пам'ять виділяється компілятором.  
  
const int n = 10;  
char st[n];        // Оголошення рядка st з n-1 (тобто 9-ти) символів;  
                  // пам'ять виділяється компілятором.  
  
int n = 10;  
char *str = new char[n]; // Оголошення рядка str з n-1 (тобто 9)  
                        // символів; пам'ять виділяється динамічно.
```


Приклад 3.6

```
char str[10] = "Vasia";
```

```
str[0]='V', str[1]='a', str[2]='s', str[3]='i', str[4]='a'
```

```
char str[] = "Vasia";    // Виділено й заповнено 6 байтів.
```

Символ у одинарних лапках, наприклад, 's' є **символьною константою**. Для зберігання такої константи компілятор C++ виділяє лише один байт пам'яті.

Символ у подвійних лапках, наприклад, "s" є **рядковою константою**, що окрім символу 's' містить символ '\0', який долучається компілятором.

Для введення й виведення рядків у консолі використовуються функції *scanf*-*printf* і *gets-puts*

```
const int n=10; char s[n];  
gets(s);  
puts(s);
```

У консолі рядки можна вводити і виводити за допомогою *cin* і *cout*, як будь-які змінні

```
const int n=10; char s[n];  
cin>>s;  
cout<<s;
```

При введенні рядків у консолі замість оператора *>>* більш оптимально використовувати метод *getline()* (два параметри: перший аргумент – рядок, який вводиться, а другий – кількість символів).

```
char s[4];  
cout<<"Введіть рядок: "<<endl;  
cin.getline(s, 4);
```

Поширеним засобом доступу до символів рядка є покажчики типу `char*`

```
char *st = "Комп'ютерна програма";
```

Рядок може вважатися за порожній у двох випадках:

- покажчик на рядок має нульове значення `NULL`
- покажчик вказує на масив, який складається з одного нульового символу

```
char *pc1 = 0;           // pc1 не адресує жодного масиву символів,  
const char *pc2 = "";    // pc2 адресує нульовий символ.
```

Функції стандартної бібліотеки для роботи з рядками

Таблиця 3.4 – Функції стандартної бібліотеки <string.h>

Функція	Призначення	Формат
strlen()	повертає довжину рядка (без урахування символу завершення рядка)	size_t strlen(char *s);
strcat()	долучає s2 до s1 і, як результат, повертає s1	char *strcat(char *s1, char *s2);
strncat()	долучає до рядка s1 перші n символів з рядка s2	char *strncat(char *s1, char *s2, size_t n);
strlwr()	перетворює всі латинські літери до нижнього регістру	char *strlwr(char *s);
strupr()	перетворює всі латинські літери до верхнього регістру	char *strupr(char *s);
strcmp()	порівнює рядки і повертає нульове значення, якщо рядки є однакові (s1=s2), від’ємне (s1s2). Порівняння відбувається посимвольно і в якості результату повертається різниця кодів перших неоднакових символів	int *strcmp(char *s1, char *s2);

Функція	Призначення	Формат
strncmp()	порівнює лише перші n символів рядка s1 з n символами рядка s2	int *strncmp (char *s1, char *s2, size_t n);
stricmp()	порівнює два рядки, не розрізняючи прописні й малі літери латиниці	int stricmp (char*s1, char *s2);
strnicmp()	порівнює лише перші n символів двох рядків, не розрізняючи великі й малі літери латиниці	int strnicmp (char *s1, char *s2, size_t maxlen);
strcpy()	копіює до s1 рядок s2, повертає s1, при цьому попереднє значення s1 втрачається	char *strcpy (char *s1, char *s2);
strncpy()	замінює перші n символів рядка s1 на перші n символів рядка s2	char *strncpy (char *s1, char *s2, size_t n);
strchr()	відшукує перше входження символу ch в рядок s і повертає покажчик на цей символ, тобто частину рядка s, розпочинаючи з символу ch і до кінця рядка. Якщо символу ch в рядку s немає, результат – NULL	char *strchr(char *s, int ch);

Функція	Призначення	Формат
strrchr()	відшукує останнє входження символу в рядку і повертає частину рядка s, розпочинаючи з останнього входження символу ch і до кінця рядка. Якщо символу ch в рядку s немає, результат – NULL	char *strrchr (char *s, int c);
strstr()	відшукує підрядок s2 в рядку s1, повертає частину рядка s1, розпочинаючи з першого спільного символу для обох рядків і до кінця s1	char *strstr(char *s1, char *s2);
strspn()	повертає довжину початкового сегмента рядка s1, символи якого є в рядку s2	size_t strcspn (char *s1, char *s2);
strcspn()	повертає індекс першого символу в рядку s1, який є спільним для обох рядків (нумерація індексів символів розпочинається з нуля)	size_t strcspn (char *s1, char *s2);
strset()	замінює усі символи рядка s на символ c	char *strset(char *s, char c);

Функція	Призначення	Формат
strnset()	замінює лише перші n символів рядка s на символ c	char *strnset (char *s, int ch, size_t n);
strpbrk()	відшукує місце першого входження в рядок s1 будь-якого символу рядка s2 і повертає частину рядка s1, розпочинаючи з цього символу і до кінця рядка	char *strpbrk (char *s1, const char *s2);
strrev()	реверс рядка s	char *strrev (char *s);
strtok()	повертає частину (лексему) рядка s1 від поточної позиції покажчика до розділового символу, зазначеного у рядку s2; зазвичай використовується для перетворювання рядка на послідовність лексем	char *strtok(char *s1, const char *s2);

Приклад 3.7

```
int k,n; char *s1="В Києві",*s2=" гарна погода",*s3;  
n=strlen(s1); // Обчислюється довжина рядка s1; n дорівнюватиме 18.  
k=strlen(s2); // Обчислюється довжина рядка s2; k=12.  
k=strcmp(s1,s2); // k=173; різниця між ASCII-кодами перших неоднакових  
// символів рядків s1 і s2 дорівнює 173, а оскільки 173>0,  
// можна стверджувати, що s1>s2.  
strcpy(s3,s2); // s3=" гарна погода" (рядку s3 присвоюється рядок s2).  
strncpy(s3,s1,2); //s3="В"; копіюються два символи з рядка s1 до s3.  
s3[2]='\0'; //Попереднє значення s3 втрачається, але, якщо не задати  
// нуль-символ, то рядок не буде обрізано і скопіюється  
// увесь рядок s1 до s3.  
strcat(s1,s2); // s1="В Києві гарна погода"; до рядка s1  
// долучається рядок s2.  
s3=strchr(s1, ' '); // Пошук пробілів у рядку s1; тепер s3=" Києві  
//гарна погода" – це рядок від першого пробілу до кінця  
// рядка s1, тобто без першого слова.
```



```
s3=strrchr(s1, ' '); // Пошук останнього пробілу в рядку s1;  
// тепер s3=" погода" – це останнє слово рядка s1.  
s3=strtok(s1, " "); // s3="В" – перше слово (лексема) рядка s1 до  
розділового  
// знака, який зазначено символом (пробілом) у другому  
// аргументі функції. Почергове здобуття усіх лексем  
// рядка можна організовувати у циклі  
n=strchr(s1, ' ')-s1+1; // n=3 – індекс першого пробілу в рядку s1,  
обчислений  
// як різниця покажчиків.  
k=strcspn(s1, " ")+1; // k=3 – індекс першого пробілу в рядку s1; оскільки  
// нумерація індексів символів розпочинається з 0, слід додати 1.  
s3=strstr(s1,s2); // s3 = "гарна погода"; пошук рядка s2 у рядку s1.  
s3=strrev(s2); // s3="адогоп анраг" – відбувається реверс рядка.  
s3=strupr("C++ Builder"); // s3="C++ BUILDER" – перетворення усіх  
// латинських літер рядка до верхнього регістру.  
s3=strupr(s3); // s3="c++ builder" – зворотне перетворення усіх  
// латинських літер рядка до нижнього регістру.
```

Приклад 3.8: програми з масивами символів

Створити рядок з усіх великих літер англійської абетки

```
#include <vcl.h>
#pragma hdrstop
#include <iostream.h>
#pragma argsused
int main(int argc, char* argv[])
{ char s[27]; // 26 латинських літер і '\0'
  char c;
  int i=0;
  for(c = 'A'; c <= 'Z'; c++) s[i++] = c;
  s[i] = '\0'; // Записування символу '\0' до
останнього символу рядка
  cout << "Англійська абетка: " << s;
  cin.get(); return 0;
}
```

Англійська абетка: ABCDEFGHIJKLMNOPQRSTUVWXYZ

Рисунок 3.2

Приклад 3.9 Ввести рядок і вивести всі його слова окремо без розділових знаків

1 спосіб

```
#include <iostream.h>
#include <string.h>
int main ()
{char s[80], *t;
  puts(" Введіть рядок, який треба
розбити на лексеми:");
  gets(s);
  puts("\n Лексеми: ");
  t= strtok(s, " .,;?!-");
  while (t != NULL)
  { puts(t);
    t = strtok(NULL, " .,;?!-");
  }
  cin.get();
  return 0;
}
```

Введіть рядок, який треба розбити на лексеми:
Hi, Tony! How are you? – I'm OK, thanks.
Лексеми:
Hi
Tony
How
are
you
I'm
OK
thanks

Рисунок 3.3

2 спосіб

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{int i, n=Edit1->Text.Length();
char *s=new char[n+2];
char *slovo=new char[n+1];
strcpy(s, Edit1->Text.c_str());
strcat(s," ");
n=strlen(s);
Memo1->Clear();
while(strchr(s,' ')) // Допоки у рядку є пробіли,
{for(i=0; i<n+2 && s[i]!=' '; i++); // відшукується позиція пробілу
    I
Strncpy(slovo, s, i); // копіюються символи до пробілу у змінну
    slovo
Slovo [i]='\0';
Memo1->Lines->Add(AnsiString(slovo)); s+=i+1; // Перехід до
    наступного слова
S+=i+1;
}
s-=n; // Повернення на початок рядка
delete[]s; delete[]slovo;
}
```

Рисунок 3.4

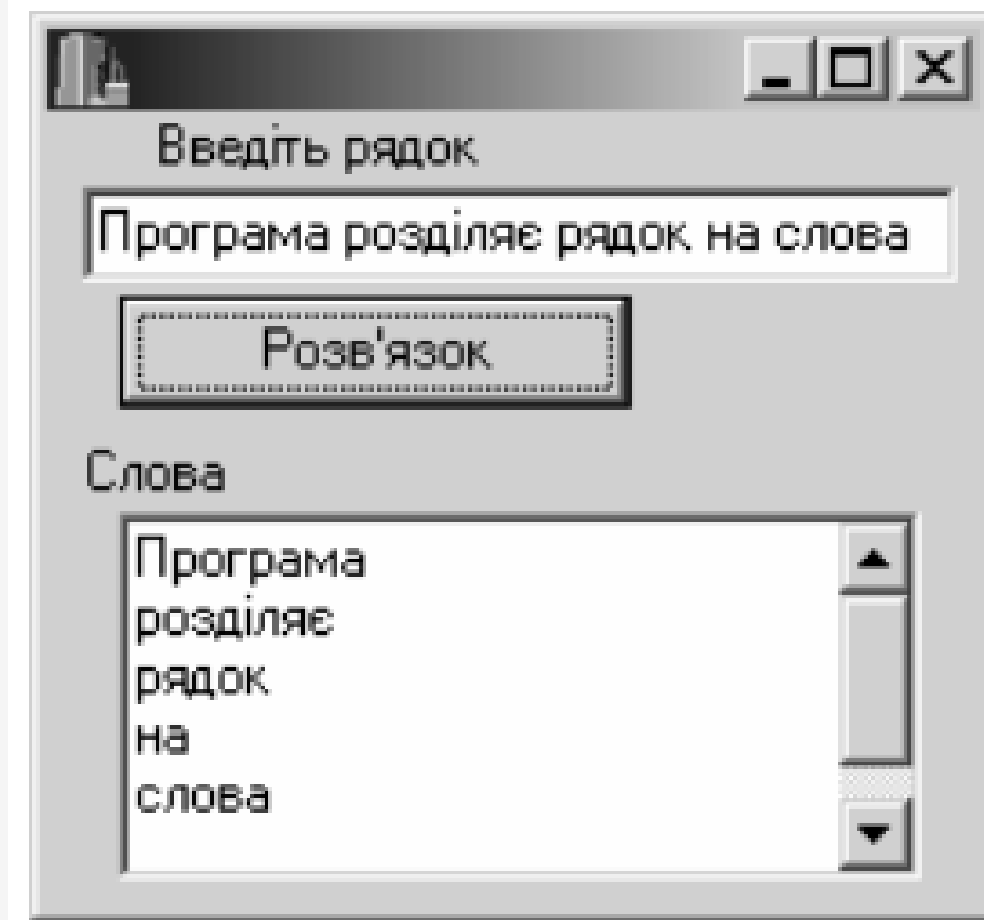


Рисунок 3.5

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{int n=Edit1->Text.Length(), i, j=0, kol=0;
  char *s=new char[n+2];
  char *slovo=new char[n+1];
  strcpy(s, Edit1->Text.c_str());
  strcat(s, " ");
  n=strlen(s);
  for(i=0;i<n-1;i++) if(!isspace(s[i])&&isspace(s[i+1])) kol++;
  char * *arr=new char *[kol];
  Memo1->Clear();
  while(strchr(s, ' '))
  {for(i=0; i<n+2 && s[i]!=' '; i++);
    strncpy(slovo, s, i);
    slovo[i]='\0';
    arr[j]=new char[strlen(slovo)+1];
    strcpy(arr[j], slovo);
    j++;
    s+=i+1;
  }
  s-=n;
  for(i=0; i<kol; i++) Memo1->Lines->Add(AnsiString(arr[i]));
  for(i=0; i<kol; i++) delete []arr[i];
  delete[]s; delete[]slovo; delete[]arr;
}
```

Рисунок 3.6

Приклад 3.10 Ввести рядок і перевірити, чи є він символьним поданням цілого числа

```
#include <stdio.h>
void main()
{ char st[20];
  printf("Введіть ціле число ");
  scanf("%s",&st);
  int i, n=strlen(st);
  // Перший символ може бути цифрою чи знаком '-'
  if((st[0]<'0' || st[0]>'9') && st[0]!='-')
  { printf("Рядок не є цілим числом");
    cin.get();
    return 0;
  }
  for(i=1; i<n; i++)
  if(st[i]<'0' || st[i]>'9') // Решта символів має бути
  цифрами
  { printf("Рядок не є цілим числом");
    cin.get();
    return 0;
  }
  printf("Рядок є цілим числом.\n");
  cin.get();
  return 0;
}
```

Введіть ціле число 23.5
Введений рядок не є цілим числом.
Введіть ціле число -56
Введений рядок є цілим числом.

Рисунок 3.7

Створювання функцій опрацювання символьних масивів

Передавання рядків у функцію є подібна до передавання масиву в якості параметра, при цьому треба зазначити адресу початку масиву.

```
float fun (int *a);  
float fun (int a[ ]);  
float fun (int a[25]);
```

```
#include <iostream.h>  
void show_string(char s[])  
{ cout << s << endl;  
}  
int main()  
{ show_string("Привіт, C++!");  
  char *st="Вчуся програмувати мовою  
C++");  
  show_string(st);  
  cin.get();  
}
```

```
#include <iostream.h>  
int length(char *s)  
{ int i;  
  // У циклі перебираються усі символи, доки не  
  зустрінеться '\0'.  
  for(i=0; s[i] != '\0'; i++);  
  return i; // Функція повертає довжину рядка,  
  яка є номером нуль-символу.  
}  
int main()  
{ char s1[] = "Вчимося програмувати мовою C++";  
  char *s2 = "Hello!";  
  cout<<"Рядок "<<s1<<" містить "<<length(s1)<<  
  " символів"<<endl;  
  cout<<"Рядок "<<s2<<" містить "<<length(s2)<<"  
  символів";  
  cin.get();  
}
```

Функції перетворювання рядка `char*` на число

```
int atoi(const char* str);  
long atol(const char* str);  
double atof(const char* str);
```

Функції цілочисельного перетворювання `atoi()` і `atol()`

[пробіли][знак]цифри

Функція перетворювання рядка в дійсне число `atof()`

*[пробіли][знак][цифри][.цифри][{ **d** | **D** | **e** | **E** } [знак]цифри]*

Основний недолік цих функцій полягає в тому, що вони ніяк не сигналізують про помилку, якщо така станеться в перебігу перетворювання рядка на число, наприклад через невідповідність його формату чи то з інших причин. Цю проблему розв'язують такі стандартні бібліотечні функції перетворювання рядків на числа:

```
long strtol(const char* str, char **endptr, int radix)
unsigned long strtoul(const char* str, char* *endptr,
                                int radix)
double strtod(const char* str, char* *endptr)
```

Дані функції мають відмінності від попередньої групи:

- через параметр `endptr` вони повертають покажчик на перший символ, який не може бути інтерпретований як частина числа;
- відбувається контроль переповнення і, якщо це сталося, функції сигналізують встановленням змінної `errno` у значення `ERANGE`, а також повертають, відповідно, `LONG_MAX/LONG_MIN`, `ULONG_MAX/ULONG_MIN` та `+/-HUGE_VAL` залежно від знака числа у переданому рядку;
- функція `strtol()` використовує інформацію про поточні встановлені через `setlocale` регіональні налагодження, а, отже, може коректно інтерпретувати числа з символом `'.'` як розділювачем цілої та дробової частини;
- для функцій `strtol()` і `strtoul()` можна задавати параметром `radix` основу системи числення, при цьому, якщо `radix` дорівнює `0`, основа визначається автоматично за першими символами числа

Приклад 3.11

```
char* end_ptr;  
long val = strtol(str, &end_ptr, 10);  
if(*end_ptr)  
    { . . . // Сигналізування про помилку в рядку  
    }  
if((val == LONG_MAX || val == LONG_MIN) && errno == ERANGE)  
    { . . . // Сигналізування про переповнення  
    }  
// Продовження штатної роботи
```

Рисунок 3.8

Перетворення числа на рядок

```
char* itoa(int value, char* string, int radix);  
char* ltoa(long value, char* string, int radix);  
char* ultoa(unsigned long value, char* string, intradix);
```

value — ціле число, яке треба перетворити на рядок;

string — буфером, до якого буде записано результат перетворювання;

radix — основа системи числення, в якій буде подано число.

```

#include <string.h>
#include <stdio.h>
#include <iostream.h>
int main()
{ char *s=new char [80];
  int n,i; double x,y,z;
  char c;
  gets(s);
  n=strlen(s);
  x=atof(s);
  for(i=1; i<n; i++)
  if(!isdigit(s[i])&& s[i]!='.')
  if(s[i]!='+'&& s[i]!='-'&& s[i]!='*'&& s[i]!='/')
  {cout<<"Error"; cin.get(); return 0;}
  else {c=s[i]; y=atof(&s[i+1]); break;}
  switch (c)
  {case '+': z=x+y; break;
   case '-': z=x-y; break;
   case '*': z=x*y; break;
   case '/': z=x/y; break;
  }
  cout<<x<<c<<y<<"="<<z<<endl;
  cin.get(); return 0;
}

```

Приклад 3.12 Ввести рядок – математичний вираз, що міститиме два числа і знак арифметичної дії поміж ними та обчислити значення цього виразу.

23,5+6

Error

23.5-7.6

23.5-7.6=15.9

Рисунок 3.9

Тема №4. МОДУЛЬНА ОРГАНІЗАЦІЯ ПРОГРАМ

Тематичний план

- 4.1 Основні правила організації функцій*
- 4.2 Визначення способів передавання параметрів до функцій*
- 4.3 Формування параметрів зі значенням за замовчуванням та функції зі змінною кількістю параметрів*
- 4.4 Основні поняття рекурсивних функцій*
- 4.5 Можливі перевантаження функцій*

4.1 ОСНОВНІ ПРАВИЛА ОРГАНІЗАЦІЇ ФУНКЦІЙ

Функція – це незалежна іменована частина програми, яка може багаторазово викликатися з інших частин програми, маніпулювати даними та повертати результати.

Основні різновиди функцій:

- стандартні убудовані функції, які є складовою частиною мови програмування (*sin(), pow() і тощо*)
- функції, які створюються безпосередньо користувачем для власних потреб.

Оголошення функції складається з прототипу (чи заголовок) і має форму

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

Приклад 4.1

float seredne (int a, int b);

Визначення функції

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

Приклад 4.2

```
float seredne (int a, int b)  
{ float sr;  
  sr=(a+b)/2.0;  
  return sr;  
}
```

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

Приклад 4.3

```
void max3(float x, float y, float z)
{ if (x>y && x>z) cout<<x;
  else
    if (y>x && y>z) cout<<y;
    else cout<<z;
}
```

Наприклад, функцію `seredne()` може бути викликано в один зі способів:

```
int a=5, x=2, y=-3, z, s1, s2;  
1) z = seredne(5, 4);           // z= 4.5  
2) s1 = seredne(a, 11);         // z= 8  
3) s2 = seredne(x, y);         // z= -0.5
```

```
float seredne(int a, int b)  
              ↑           ↑  
z = seredne( 5,      4 );
```

Функція може бути оголошена убудованою (inline):

```
inline int modul(int x)
{ if(x<0) return -i;
  else return i;
}
```

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.

5) Послідовність змінних, які є фактичними параметрами, має збігатися з послідовністю формальних параметрів.

6) У прототипі функції необов'язково задавати імена формальних параметрів, достатньо їх оголосити, наприклад у такий спосіб:

```
float seredne(int, int);
```

7) Якщо функція не отримує жодного параметра, слід у заголовку функції ставити порожні дужки чи записати у дужках void:

```
double func();  
double func(void);
```

8) Тип значення, яке повертає функція, може бути яким-завгодно, але не може бути масивом чи функцією.

```
int [10] func();
```

```
int * func();
```

9) 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*.

Глобальні змінні є відомими всім функціям програми, кожна з функцій може змінювати значення таких змінних.

```
int Sum(int A, int B)
{ // Локальну змінну x і параметри A та B видно лише у тілі функції Sum().
  int x = A + B;
  return x; }
void main()
{ int x = 2, y = 4; // x, y – локальні змінні для main()
  cout << Sum(x, y);
}
```


Приклад 4.4: програма з функціями

Віднайти периметр трикутника, заданого координатами вершин.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```
double segment(double x1, double y1, double x2, double y2)
{ return sqrt(pow(x2-x1,2)+pow(y2-y1,2)); }
void main()
{ double x1, y1, x2, y2, x3, y3, P;
  cout<<"Перша вершина:\n";
  cout<<"Введіть координату x: "; cin>>x1;
  cout<<"Введіть координату y: "; cin>>y1;
  cout<<"Друга вершина:\n";
  cout<<"Введіть координату x: "; cin>>x2;
  cout<<"Введіть координату y: "; cin>>y2;
  cout<<"Третя вершина:\n";
  cout<<"Введіть координату x: "; cin>>x3;
  cout<<"Введіть координату y: "; cin>>y3;
  P=segment(x1,y1,x2,y2)+segment(x2,y2,x3,y3)+
    segment(x1,y1,x3,y3);
  cout<<"Периметр дорівнює: "<<P<<endl;
}
```

Результати виконання програми:

Перша вершина:

Введіть координату x: 3

Введіть координату y: 5

Друга вершина:

Введіть координату x: 1

Введіть координату y: 7

Третя вершина:

Введіть координату x: 4

Введіть координату y: 2

Периметр дорівнює: 11.8217

Рисунок 4.2

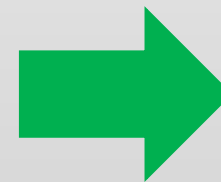
Рисунок 4.1

4.2 ВИЗНАЧЕННЯ СПОСОБІВ

ПЕРЕДАВАННЯ ПАРАМЕТРІВ ДО ФУНКЦІЙ

Якщо значення певних локальних змінних треба зберігати, при оголошенні 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;
}
```



n	m	p
0	0	0
1	1	0
2	2	0
n	m	p
3	0	0
4	1	0

Два способи передавання параметрів до функції:

- 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 покажчика і посилення.

Приклад 4.5: функція з передаванням параметрів за посиланням

```
void duplicate(int& a, int& b, int& c)
{ a*=2;  b*=2;  c*=2;
}
int main ()
{ int x=1, y=3, z=7;
  duplicate(x, y, z);
  cout << "x=" << x << ", y=" << y << ", z=" << z;
  return 0;
}
```

Результат:

x=2, y=6, z=14

Рисунок 4.3

```
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

Рисунок 4.4

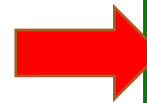
Якщо виникає потреба уникнути змінювання параметра всередині функції, використовується специфікатор **const**:

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

Приклад 4.6: функції з параметрами, які передаються за посиланням і за покажчиком

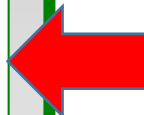
```
void swap_values1(float *a, float *b)  
{ float temp = *a;  
  *a = *b;  
  *b = temp;  
}
```

```
a=3; b=5;  
swap_values1(&a, &b);
```



```
void swap_values2(float &a, float &b)  
{ float temp = a;  
  a = b;  
  b = temp;  
}
```

```
a=3; b=5;  
swap_values2(a, b);
```



Якщо функція має повернути понад одне значення, можна передати їй за посиланням додаткові параметри, а у тілі функції їм буде присвоєно значення, які треба повернути.

```
void prevnext (int x, int& prev, int& next)
{ prev = x-1;  next = x+1; }
```

При використуванні масиву як параметра функції передається покажчик на його перший елемент, тобто масив завжди передається за адресою.

```
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, якщо вони невідомі на етапі компіляції, мають передаватися як параметри.

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

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

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


Приклад 4.7 Ввести масив і обчислити за допомогою функції добуток від'ємних елементів масиву.

```
double dob(double *a, int n)
{ int i, k=0; double p=1;
  for(i=0; i<n; i++)
    if(a[i]<0) { k++; p *= a[i]; }
  if(k>0) return p; else return 0; }

void __fastcall TForm1::Button1Click (TObject *Sender)
{ double P; int i, n = Mem1->Lines->Count;
  double *a = new double [n];
  for(i=0; i<n; i++) a[i]=StrToFloat(Mem1->Lines->Strings[i]);
  P = dob(a, n);
  if(P!=0) Edit1->Text=FloatToStr(P);
  else ShowMessage("Від'ємних нема");
}
```

Рисунок 4.5

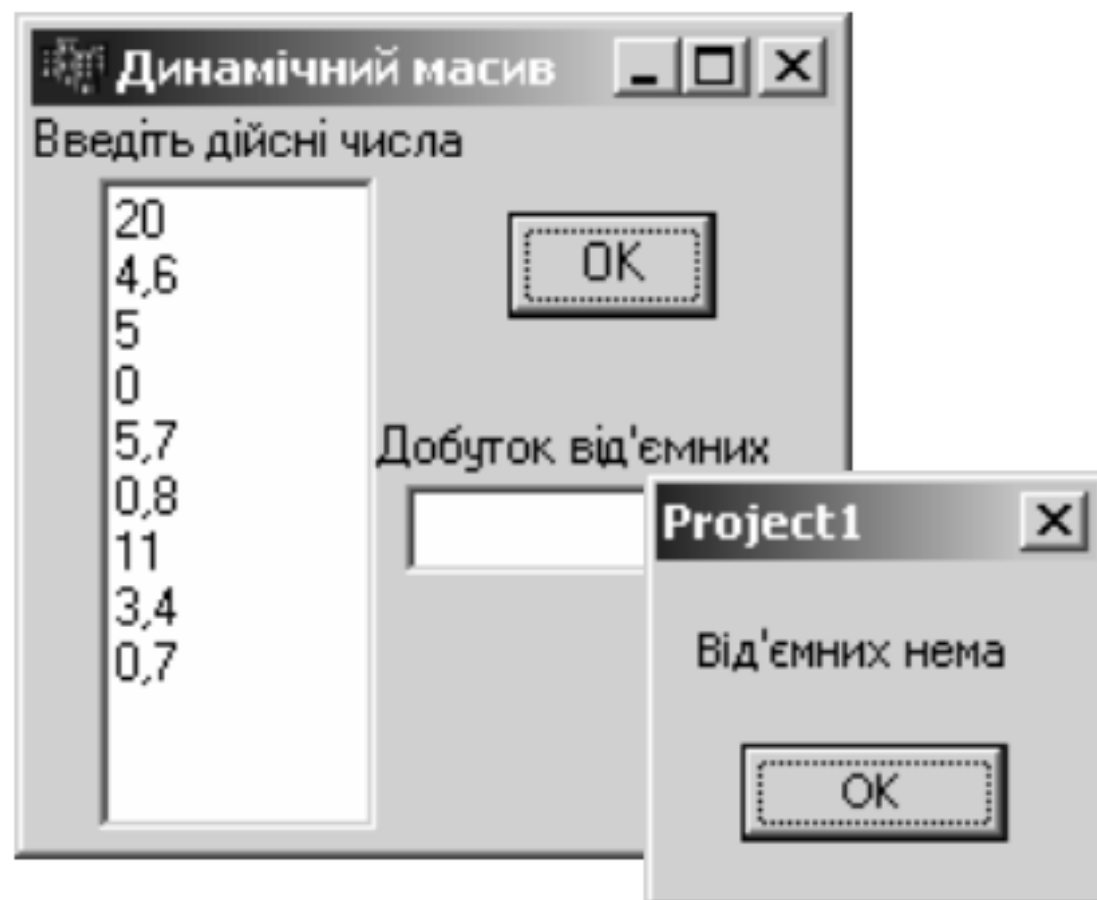
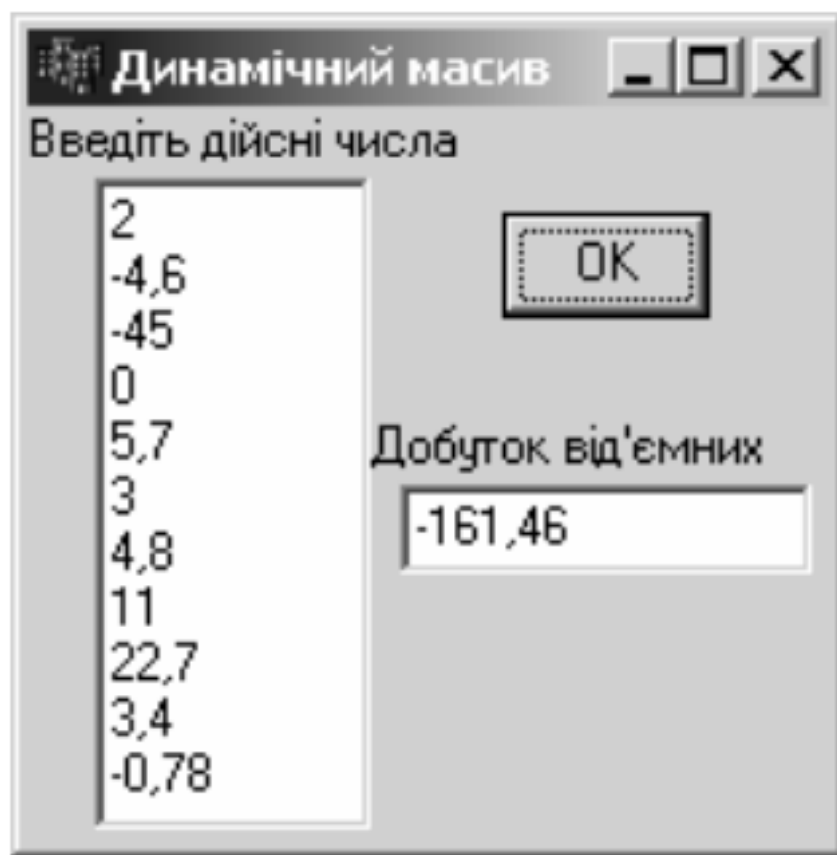


Рисунок 4.6

Приклад 4.8 Ввести рядок і перетворити його на верхній регістр.

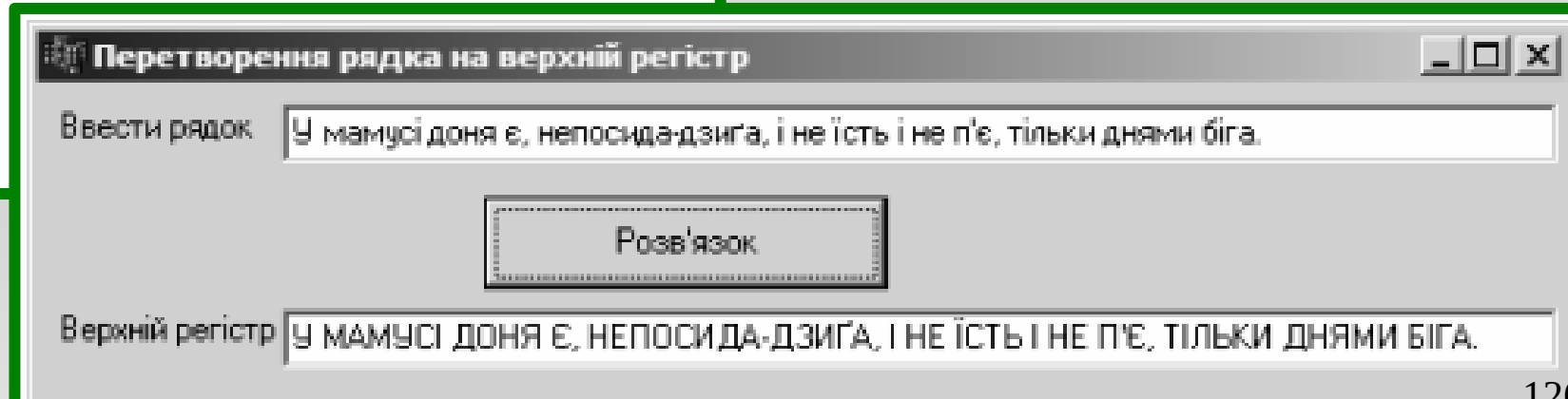
```
void toUp(char *c)
{ int n=strlen(c);

  for(int i=0; i<n; i++)
  { if(isalpha(c)) c = toupper(c);
    if(c >= 'a' && c <= 'я') c -= 32;
    if(c == 'ï' || c == 'е') c = c - 16;
    if(c == 'r') c = 'Г';
    if(c == 'i') c = 'I';
  } return c;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{ char s[50];
  strcpy(s, Edit1->Text.c_str());
  toUp(s);
  Edit2->Text=AnsiString(s);
}
```

Рисунок 4.8

Рисунок 4.7



4.3 ФОРМУВАННЯ ПАРАМЕТРІВ ЗІ ЗНАЧЕННЯМ ЗА ЗАМОВЧУВАННЯМ ТА ФУНКЦІЇ ЗІ ЗМІННОЮ КІЛЬКІСТЮ ПАРАМЕТРІВ

Параметри за замовчуванням 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.

Приклад 4.9: прототипи функцій з параметрами за замовчуванням.

```
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.
```

Приклад 4.10 Написати функцію, яка виконує ділення одного цілого числа на друге. Якщо друге число не зазначено, то виконується ділення на 2.

```
double divide(int a, int b=2)
{ return (double)a/b; }
void main()
{ cout << divide(12);      // 12:2
  cout << endl;
  cout << divide(20, 4);    // 20:4
  getch();
}
```

Рисунок 4.9

Результати виконання програми:

6

5

Рисунок 4.10

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.

Приклад 4.11: *стандартна функція зі змінним числом параметрів*

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

```
printf("Уведіть вихідні дані");      // Один параметр  
printf("Сума: %5.2f гривень", 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.

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);
```

```
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);
```

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.

Приклад 4.12 Ввести послідовність цілих чисел і обчислити за допомогою функції зі змінною кількістю параметрів середнє значення цієї послідовності. За завершення послідовності слід вважати значення, яке дорівнює -1 .

```
float sred_znach (int x, ...)
{ int i=0, j=0, sum=0;
  va_list uk_arg;
  va_start(uk_arg, x);      // Встановлення покажчика uk_arg на перший
                             // необов'язковий параметр

  if(x!=-1) sum=x;          // Перевірка наявності чисел у списку
  else return 0;
  j++;

  while((i=va_arg(uk_arg, int))!=-1) // Вибірка чергового параметра
    {sum+=i; j++;}           // й перевірка завершення списку
  va_end(uk_arg);           // Закриття списку параметрів
  return (float)sum/j;
}

void main()
{ float sr=sred_znach(2,3,4,-1);    // Виклик з чотирма параметрами
  cout << "sr=" << sr << endl;
  sr=sred_znach(5,6,7,8,9,-1);      // Виклик з шістьма параметрами
  cout << "sr=" << sr;
  getch();
}
```

Рисунок 4.11

4.4 ОСНОВНІ ПОНЯТТЯ РЕКУРСИВНИХ ФУНКЦІЙ

Рекурсивна функція – це функція, яка викликає сама себе.

Рекурсія буває *прямою* та *непрямою*.

Рекурсія дозволяє звести складне завдання до розв'язку його зменшених копій.

Приклад 4.13: числа Фібоначчі

$$F(n) = F(n-1) + F(n-2), \text{ де } F(0)=1, F(1)=1$$

Рекурсія повинна мати всередині себе умову завершення, за якою наступний крок рекурсії вже не виконуватиметься.

Послідовність рекурсивного виклику:

- у стеку резервується місце для формальних параметрів, куди записуються значення фактичних параметрів;
- при виклику функції до стека записується точка повертання – адреса тієї частини програми, де міститься виклик функції;
- на початку тіла функції у стеку резервується місце для локальних (автоматичних) змінних.

“Функція F виконує ... і викликає F, яка виконує ... і викликає F...”

Класичний приклад рекурсивної функції є обчислення факторіала

$$n! = n * (n-1) !$$

рекурентне співвідношення

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

повне рекурсивне визначення

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

рекурсивна функція

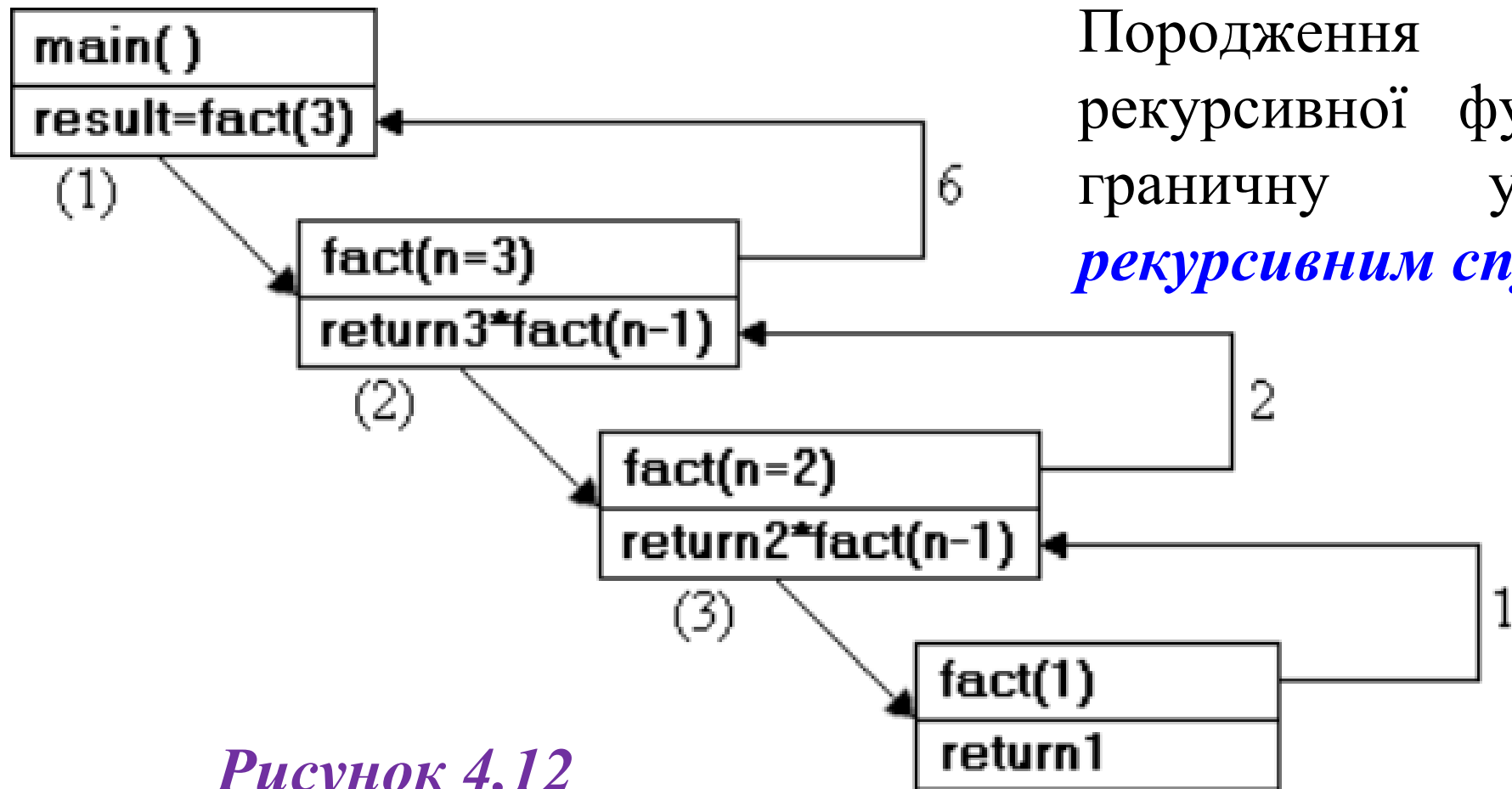


Рисунок 4.12

Породження усіх нових копій рекурсивної функції до виходу на граничну умову називається *рекурсивним спуском*.

Максимальна кількість копій рекурсивної функції, яке водночас може міститися у пам'яті комп'ютера, називається *глибиною рекурсії*.

Завершення роботи рекурсивних функцій, аж до самої першої, яка ініціювала рекурсивні виклики, називається *рекурсивним підйомом*.

Приклад 4.14: обчислення чисел Фібоначчі

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 і т.д

Ітераційна функція обчислення

```
long fib1(int n)
{ long a=0, b=1, c=n;
  for(int i=2; i<=n; i++)
  { c=a+b;   a=b;   b=c;   }
  return c;
}
```

Рекурсивна функція обчислення

```
long fib2(int n)
{ if (n==0 || n==1) return n;
  else return fib2(n-1)+fib2(n-2); }
```

```
long fib3(int n)
{ if (n>1) ? fib3(n-1)+fib3(n-2) : n; }
```

Приклад 4.15 Ввести ціле число і вивести всі цифри, які зображують це число.

```
void cnum (int n)
{ int a=10;
  if(n == 0) return;
  else { cnum(n/a);   cout << n%a<<endl;  }
}
void main()
{ int n;
  cout << "Увести ціле число: ";   cin >> n;
  cnum(n);
  getch(); return 0;
}
```

Рисунок 4.13

Результати виконання програми:

Увести ціле число: 1234

1

2

3

4

Рисунок 4.14

Приклад 4.16 Створити рекурсивну функцію піднесення числа до ступеня.

```
#include <iostream.h>
#include <conio.h>
double power(double x, int st)
{ if(st <= 0) return 1;
  else return(x*power(x,st-1));
}
void main()
{ double x; int st;
  cout<<"Увести число x= "; cin>>x;
  cout<<"Увести показник степеня st= "; cin>>st;
  cout<<"x ^ st= " << power(x, st);
  getch(); return 0;
}
```

Рисунок 4.15

Рисунок 4.16

Результати виконання програми:

Увести число x= 2.5

Увести показник степеня st= 2

x ^ st= 6.25

Приклад 4.17 Написати рекурсивну функцію, яка обчислює мінімальний елемент масиву.

```
int minimum(int *mas, int n, int k)
{ if(k == n-1) return k;
  int a = minimum(mas,n,k+1); // Виклик для підмасиву розміром на 1 менше.
  if(mas[a]<mas[k]) return a;   else return k;
}
void main()
{ int i, n, min;
  cout<<"Уведіть розмір масиву: ";   cin>>n;
  int *mas = new int[n];
  for(i=0;i<n;i++)
  { cout<<"Уведіть mas["<<i<<"]: ";   cin>>mas[i];  }
  cout<<endl;
  min = minimum(mas,n,0); // Виклик для всього масиву (k=0).
  cout<<"Мінімум: mas["<<min<<"] = "<<mas[min]<<endl;
  getch();
  return 0;
}
```

Рисунок 4.17

Приклад 4.18 Написати рекурсивну функцію, яка обчислює суму квадратних коренів натуральних чисел від 1 до n.

```
#include <iostream.h>
#include <conio.h>
#include <math.h>
double sumkor(int a)
{ if(a == 1) return 1;
  else return (sqrt(a) + sumkor(a-1)) ;
}
void main()
{ int n;
  cout<<"n= ";   cin>>n;
  cout<<sumkor(n) ;
  getch(); return 0;
}
```

Рисунок 4.18

Приклад 4.19 Ввести масив з 10-ти дійсних чисел і створити рекурсивну функцію, яка записує масив у зворотній послідовності.

```
void invertItem(double *a, int i, int j)
{ double t = a[i];  a[i] = a[j];  a[j] = t;
  i++; j--;
  if(i < j) invertItem(a, i, j)
}

void main()
{ double a[10];  int i;
  cout<<"Увести 10 дійсних чисел:"<<endl;
  for(i=0; i<10; i++)  cin>>a[i];
  invertItem(a, 0, 9);
  for(i=0; i<10; i++)  cout<<a[i]<<" ";
  getch(); return 0;
}
```

Рисунок 4.19

Результати виконання програми:

Увести 10 дійсних чисел:

0	1	2	3	4	5	6	7	8	9
9	8	7	6	5	4	3	2	1	0

Рисунок 4.20

Приклад 4.20 Ввести два числа й обчислити найбільший спільний дільник двох цілих чисел, застосовуючи рекурсивний алгоритм Евкліда.

```
int nod(int a, int b)
{ int c;
  // За умови  $b > a$  параметри змінюються місцями
  if(b > a) c = nod(b, a);
  else
    if(b <= 0) c = a;
    else c = nod(b, a%b); // Тут  $a \% b$  – залишок від ділення  $a$  на  $b$ 
  return c;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{ int a, b;
  a = StrToInt(Edit1->Text);
  b = StrToInt(Edit2->Text);
  Edit3->Text = IntToStr(nod(a, b));
}
```

Рисунок 4.21

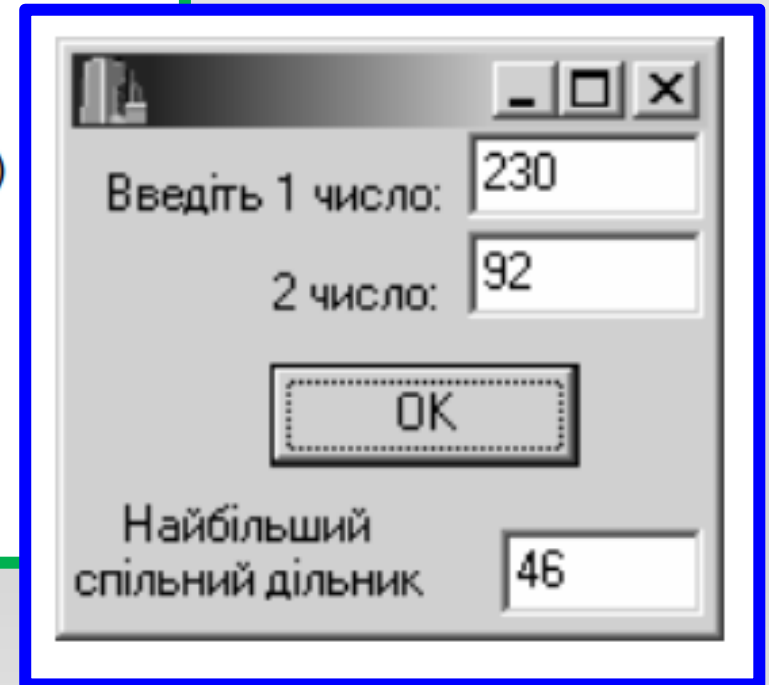


Рисунок 4.22

4.5 МОЖЛИВІ ПЕРЕВАНТАЖЕННЯ ФУНКЦІЙ

Використовування кількох функцій з однаковим ім'ям, але з різними типами параметрів називається **перевантаженням функцій**.

Компілятор визначає, яку саме функцію слід викликати, за типом фактичних параметрів, називається **дозволом перевантаження**.

Приклад 4.21: чотири варіанти функції для визначення максимального значення

```
int max(int, int); // Повертає найбільше з двох цілих
char* max(char*, char*); // Повертає рядок максимальної довжини
// Повертає найбільше значення з першого параметра і довжини рядка, переданого
// до функції в якості другого параметра
int max(int, char*);
// Повертає найбільше значення з першого параметра і довжини рядка, переданого
// до функції в якості першого параметра
int max(char*, int);
void f(int a, int b, char* c, char* d)
{   cout << max(a,b) << max(c,d) << max(a,c) << max(c,b); }
```

Рисунок 4.23

Виникнення неоднозначності:

- перетворення типу;
- використання параметрів-посилань;
- використання аргументів за замовчуванням.

Приклад 4.22: неоднозначність за перетворення типу

```
#include <iostream.h>
int f(int a){return a;}
int f(int a, int b=1){return a*b;}
void main ()
{ cout << f(10,2);    // Викликається f(int, int)
  cout << f(10);     } // Неоднозначність – що викликати: f(int, int) чи f(int)?
```

Рисунок 4.24

Якщо функції з однаковим ім'ям мають однакові параметри і типи результату, вважається, що друге визначення розглядатиметься як *редекларація* (повторне визначення).

Якщо параметри функцій з однаковим ім'ям збігаються і функцій відрізняються лише типом значення, яке повертається, це призведе до помилки.

Правила опису перевантажених функцій:

- перевантажені функції мають перебувати в одній області видимості, інакше станеться приховування аналогічне до однакових імен змінних у вкладених блоках;
- перевантажені функції можуть мати параметри за замовчуванням, при цьому значення одного й того самого параметра у різних функціях мають збігатися.

Функції не можуть бути перевантажені, якщо опис їхніх параметрів відрізняється лише специфікатором *const* чи використанням посилання, наприклад *int* та *const int* чи *int* та *int&*.

Приклад 4.23 Написати дві функції для обчислення суми елементів цілого й дійсного масивів відповідно. Використати ці функції для обчислення сум двох масивів.

```
int adder(int iarray[7])
{ int isum = 0;
  for(int i=0; i<7; i++)
    isum += iarray[i];
  return(isum);
}
float adder (float farray[7])
{ float fsum = 0;
  for(int i= 0; i < 7; i++)
    fsum += farray[i];
  return (fsum) ;
}
void main() {
  int iarray[7] = {5,1,6,20,15,0,12};
  float farray[7] = {3.3,5.2,0.05,1.49,3.12345,31.0,2.007};
  int isum; float fsum;
  isum = adder (iarray); // Виклик для масиву цілих чисел
  fsum = adder (farray); // Виклик для масиву дійсних чисел
  cout << "Сума масиву цілих чисел дорівнює " << isum << "\n";
  cout << "Сума масиву дійсних чисел дорівнює " << fsum << "\n";
  getch ();
}
```

Результати виконання програми:

Сума масиву цілих чисел дорівнює 59

Сума масиву дійсних чисел дорівнює 46.1705

Рисунок 4.26

Рисунок 4.25

Приклад 4.24 Написати три функції для обчислення периметра трикутника, чотирикутника й п'ятикутника відповідно.

```
// Визначення всіх потрібних версій функції
// Версія функції для трикутника
int Perimetr(int Len1, int Len2, int Len3)
{ return Len1 + Len2 + Len3; }
// Версія функції для чотирикутника
int Perimetr(int Len1, int Len2, int Len3, int Len4)
{ return Len1 + Len2 + Len3 + Len4; }
// Версія функції для п'ятикутника
int Perimetr(int Len1, int Len2, int Len3, int Len4, int Len5)
{ return Len1 + Len2 + Len3 + Len4 + Len5; }
int main()
{ int AB = 5, BC = 3, CA = 7;
  int CD = 8, DA = 11, DE = 2, EA = 9;
  cout << "Perimetr ABC = " << Perimetr(AB, BC, CA) << "\n";
  cout << "Perimetr ABCD = " << Perimetr(AB, BC, CD, DA) << "\n";
  cout << "Perimetr ABCDE = " << Perimetr(AB, BC, CD, DE, EA);
  getch();
}
```

Результати виконання програми.

```
Perimetr ABC = 15
Perimetr ABCD = 27
Perimetr ABCDE = 27
```

Рисунок 4.28

Рисунок 4.27

Тема №5. ОСНОВНІ ТИПИ КОРИСТУВАЧІВ

Тематичний план

5.1 Перейменовування типів (typedef)

5.2 Структури (struct)

5.3 Об'єднання (union)

5.3 Перерахування (enum)

5.4 Множини (Set)

5.1 ПЕРЕЙМЕНОВУВАННЯ ТИПІВ (TYPEDEF)

double A[5][7];

*A – двовимірний масив розмірністю
5 рядків та 7 стовпчиків*

typedef - службове слово для створення типів користувача на основі існуючих типів

typedef double matrica [5][7];

matrica A;

змінна A має тип matrica

`matrica B, C;` // *B і C – матриці 5×7*

`double sumr(matrica C);` /* *Заголовок функції, параметром якої є матриця 5×7* */

`matrica M[4];` // *Масив з 4-х матриць*

`double M[4][5][7];`

`typedef <тип> <нове ім'я> [<розмірність>];`

Приклад 5.1

```
typedef unsigned char byte; /* Перейменування типу “беззнаковий char” на  
byte */
```

```
typedef char MS[50]; /* Перейменування типу “рядок з 50-ти символів” на  
MS */
```

```
char s[20]; // Слово s – рядок з 20-ти символів,  
char mas[10][20]; // mas – масив з 10 слів  
typedef char slovo [20];  
slovo s; // Оголошення змінної s типу slovo  
slovo mas[10]; // і масиву з 10-ти слів
```

5.2 СТРУКТУРИ (STRUCT)

Структура – це складений тип даних, змінні (поля) якого можуть містити різноманітну й різнотипну інформацію.

```
struct  <ім'я_типу_структури>  
{  <тип1> <поле1>;  
    <тип2> <поле2>;  
    . . .  
    <типN> <полеN>;  
} <список_змінних>;
```

Рисунок 5.1

Приклад 5.2: оголошення структури з ім'ям *sportsman*

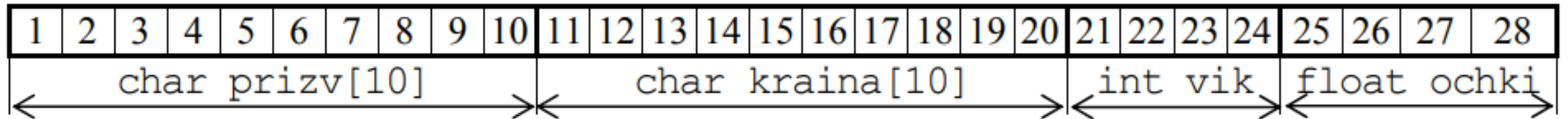
```
struct sportsman
```

```
{ char prizr[15]; char kraina[10]; // Прізвище та країна,  
  int vik; float ochki; // вік та кількість очок.
```

```
};
```

```
sportsman Sp; /* Змінна Sp типу sportsman може зберігати  
інформацію про одного спортсмена. */
```

```
sportsman Mas[15]; /* Масив Mas типу sportsman може містити  
інформацію про 15 спортсменів. */
```



Реальний обсяг пам'яті, яку займає структура, можна визначити за допомогою *sizeof()*.


```
sporsman Sp={"Бойко", "Україна", 20, 75.3};
```

```
struct POINT
```

```
{ int x, y;
```

```
} spot = { 20, 40 }; // Точка має координати x = 20, y = 40
```

```
struct complex
```

```
{ float real, im;
```

```
} compl[2][3]={{{1.4}, {5}, {-3.1}},{{2}, {-5.2}, {0}}};
```

<структурна змінна>.<ім'я поля>

```
strcpy(Sp.prizv, "Бойко");  
strcpy(Sp.kraina, "Україна");  
Sp.vik = 20;  
Sp.ochki = 75.3;
```

```
typedef struct  
{ AnsiString priz, gruppa;  
  int Bal_math, Bal_inform;  
} student; // Оголошення типу "студент"  
student W; // і змінної W цього типу.
```

```
sportsman *p=new sportsman; // Виділення пам'яті,  
p->ochki=95.3; // присвоєння очок полю ochki  
(*p).vik=23; // і віку полю vik.
```

```
struct
```

```
{ char prizm[15];  
  char kraina[10];  
  int vik; float ochki;  
}
```

```
Sp;
```

Оголошення структури з чотирма полями без створення іншого типу

```
struct date
```

```
{ int day, month, year; // День, місяць та рік  
};
```

```
struct person
```

```
{ char prizm[20], adresa[150]; // Прізвище, адреса,  
  char phone[10]; date birth_date; // телефон, дата народження,  
  date work_date; // дата вступу на роботу  
  float salary; // та зарплата.  
};
```

Структурні змінні одного типу можна переприсвоювати одну одній, при цьому виконується присвоювання усіх полів.

person man=P[0];

Поля структури можуть бути якого завгодно типу, у тому числі масивами.

У C++ структура при описуванні, окрім звичних полів, може містити елементи-функції. До функцій структури передаються як звичайні змінні.

5.3 ОБ'ЄДНАННЯ (UNION)

Об'єднання – формат даних, який може містити різні типи даних, але лише один тип водночас.

Обсяг пам'яті, яку займає об'єднання дорівнює найбільшому з розмірів його полів. Об'єднання застосовують для економії пам'яті у тих випадках, коли відомо, що понад одного поля водночас не потрібно.

```
union prim
{ int x;
  short y;
  double z;
};
```

```
struct
{ int type;
  union id
  { long id_num; char id_ch[20]; };
} price;
...
if (price.type == 1) cin >> price.id.id_num;
else cin >> price.id.id_ch;
```

Коли об'єднання є анонімним, то його елементи стають змінними, розташованими за однією адресою.

```
struct
{ int type;
  union
  { long id_num;
    char id_ch[20];
  };
} price;
...
if(price.type == 1) cin >> price.id_num;
else cin >> price.id_ch;
```

5.4 ПЕРЕРАХУВАННЯ (ENUM)

```
enum [<ім'я_типу>] {<список_констант>;
```

Ім'я типу задається в тому разі, якщо у програмі потрібно визначати змінні цього типу.

Змінним перераховного типу можна присвоювати кожне значення зі списку констант, зазначених при оголошуванні типу. Імена перераховних констант мають бути унікальними.

```
enum week { Sat=0, Sun=0, Mon, Tue, Wed, Thu, Fri } rob_den;
```

До перераховних змінних можна застосовувати арифметичні операції й операції відношення

```
enum days {Sun, Mon, Tue, Wed, Thu, Fri, Sat};  
days day1 = Mon, day2 = Thu;  
int diff = day2 - day1; cout << "Різниця у днях: " << diff << endl;  
if(day1 < day2)  
    cout << "day1 настане раніш, аніж day2 \n";
```

Значення перераховним константам можна встановлювати явно, ініціалізуючи їх при оголошенні

```
enum {first, second = 100, third};
```

Істотним недоліком типів перерахування є те, що вони не розпізнаються засобами введення-виведення C++. При виведенні такої змінної виводиться не її формальне значення, а її внутрішнє подання, тобто ціле число.

5.5 МНОЖИНИ (SET)

Множина – це група елементів, яка асоціюється з її ім'ям і з якою можна порівнювати інші величини, щоб визначати їхню належність до цієї множини.

```
Set <type, minval, maxval> змінні;
```

```
Set <char, 'A', 'Z'> s1;
```

Ініціалізація множини здійснюється за допомогою долучення елемента до множини операцією <<, а видалення елементів з множини – операцією >>.

```
s1 << 'A' << 'B' << 'C' << 'Z';
```

```
s1 >> 'B' >> 'C';
```

Для переглядання значень, записаних у множині, можна скористатися методом ***Contains(<значення>)***, який дає результат ***true***, якщо певне значення записано у множині, і ***false***, – якщо воно є відсутнє у множині.

```
AnsiString s=""; char c;
Set <char, 'A', 'Z'> s1;
s1 << 'A' << 'B' << 'Z';
for(c='A'; c<='Z'; c++)
    if(s1.Contains(c)) s = s + AnsiString(c)+" ";
Edit1->Text = s ; // s="ABZ"
```

```
AnsiString s = ""; char c;
Set <short int, 0, 200> f;
f << 1 << 5 << 19 << 119;
f << 114;
for(int i=1; i<=200; i++)
    if(f.Contains(i)) s += IntToStr(i) + " ";
Edit1->Text = s ; //s="1 5 19 114 119"
}
```

Існують ще метод очищення множини *Clear()* і метод *Empty()*, який визначає, чи не є множина порожньою.

Таблиця 5.1 – Методи роботи з множинами

Формат методу	Опис
Set& Clear();	Очищення заданої множини
bool Contains(T e1);	Перевірка наявності у множині елемента e1
bool Empty();	Перевірка того, чи не є множина порожньою

Таблиця 5.2 – Операції роботи з множинами

<i>Операція</i>	<i>Формат</i>	<i>Опис</i>
- -=	Set operator – (Set& rhs); Set operator -= (Set& rhs);	Із заданої множини вилучаються елементи множини rhs
* *=	Set operator * (Set& rhs); Set operator *= (Set& rhs);	Перетин множин, тобто спільні елементи заданої множини та rhs
+ +=	Set operator + (Set& rhs); Set operator += (Set& rhs);	До заданої множини долучаються елементи множини rhs
== !=	bool operator ==(Set& rhs); bool operator !=(Set& rhs);	Еквівалентність і нееквівалентність двох множин: заданої та rhs

Приклад 5.3

```
Set <short, 0, 200> f;  
f << 1 << 5 << 19 << 119; // f складається з 1, 5, 19, 119  
Set <short, 0, 200> f1;  
f1 << 2 << 45;           // f1 складається з 2, 45  
f = f.operator +(f1);      // або f+=f1; тепер f складається з 1,2,5,19,45,119  
Set <short, 0, 200> f2;  
f2 << 2 << 5 << 19 << 47; // f2 складається з 2, 5, 19, 47  
f = f.operator * (f2);     // Операція operator* або f*=f1;  
// f є перетином множин f та f2, в якому залишилися числа 2, 5 та 19
```

Рисунок 5.2

Тема №6. РОБОТА З ФАЙЛАМИ

Тематичний план

- 6.1 Загальне визначення поняття файлів*
- 6.2 Робота з текстовими файлами на мові програмування C/C++*
- 6.3 Робота з текстовими файловими потоками на мові програмування C/C++*
- 6.4 Виконання послідовного та довільного записування та зчитування інформації у файлах*
- 6.5 Забезпечення опрацювання текстових файлів за допомогою дескрипторів*
- 6.6 Основи роботи з бінарними файлами*
- 6.7 Основи роботи з бінарними файловими потоками*
- 6.8 Виконання опрацювання бінарних файлів із застосуванням дескрипторів*

6.1 ЗАГАЛЬНЕ ВИЗНАЧЕННЯ ПОНЯТТЯ ФАЙЛІВ

Файли – це іменовані області пам'яті на зовнішньому носії, призначені для довготривалого зберігання інформації.

Файли мають **імена** й є організовані в ієрархічну деревовидну структуру з **каталогів** (тек) і простих файлів.

Відмінність файлів від масивів:

- вони можуть змінювати свій розмір;
- звертання до елементів файлів здійснюється не за допомогою операції індексації [], а за допомогою спеціальних системних викликів та функцій;
- доступ до елементів файла відбувається з так званої позиції зчитування-записування, яка автоматично просувається при операціях зчитування-записування, тобто файл є видимим послідовно.

Відкриття файла – це вибір книжки чи блокнота за заголовком на його обкладинці й відкриття обкладинки.

Типи файлів, що опрацьовуються в C/C++: ***текстові*** та ***бінарні***.

Текстові файли призначено для зберігання текстів, тобто сукупності символьних рядків змінної довжини. Кожен рядок завершується керувальною послідовністю '\n', а розділювачами слів та чисел у рядку є пробіли й символи табуляції.

Бінарні файли зберігають дані в тому самому форматі, в якому вони були оголошені, і їхній вигляд є такий самий, як і в пам'яті комп'ютера.

6.2 РОБОТА З ТЕКСТОВИМИ ФАЙЛАМИ НА МОВІ ПРОГРАМУВАННЯ C/C++

Інформація про файл заноситься до змінної типу *FILE**.

Тип *FILE* означено у бібліотеці *<stdio.h>*.

```
#include <stdio.h>
```

```
FILE *f;
```

```
FILE *fopen(const char *filename, const char *mode);
```

Таблиця 6.1 – Специфікатори режиму відкривання файлів

Параметр	Опис
r	відкрити файл лише для зчитування даних
r+	відкрити існуючий файл для зчитування й записування даних
a	відкрити чи створити файл для записування даних у кінець файла
a+	відкрити чи створити файл для зчитування чи то записування даних у кінець файла
w	створити файл для записування даних
w+	створити файл для зчитування і записування даних

```
f = fopen("Prim.txt", "wt");
```

```
if(f == NULL) {ShowMessage("Файл не відкрито"); return;}
```

Припинити роботу з файлом

```
fclose(FILE *f).
```

Приклад 6.1: відкриття текстового файлу для зчитування

```
FILE *f;  
// Перевіряємо, чи повертає ця функція нульовий покажчик.  
if ( (f=fopen ("t.txt", "rt") ) ==NULL)  
    { ShowMessage ("Неможливо відкрити файл");    return; }  
.  
.  
.  
// Сюди вставляються команди зчитування з файла.  
fclose (f);    // Закриття файла.
```

Рисунок 6.1

Зчитування рядка з файла здійснюється функцією fgets():

```
char *fgets(char *s, int m, FILE *stream);
```

Перевірка кінця файла здійснюється функцією feof():

```
int feof (FILE *stream);
```

Приклад 6.2

```
char s[50]; Memo1->Clear();  
do { fgets(s, 50, f);  
    Memo1->Lines->Add(s);  
} while (!feof(f));
```

```
if (s[strlen(s)-1]=='\n')    s[strlen(s)-1]= '\0';
```

```
while (fgets(s, 50, f))  
{ if (s[strlen(s)-1]=='\n')    s[strlen(s)-1] = '\0';  
  Memo1->Lines->Add(s); }
```

Рисунок 6.2

Зчитування форматованих даних можна також здійснювати за допомогою функції fscanf():

```
int fscanf(FILE *stream, const char *format[, address.]);
```

Приклад 6.3

```
float r;  
Memol->Clear();  
while(fscanf(f, "%e", &r) > 0)  
    Memol->Lines->Add(FloatToStr(r));
```

Рисунок 6.3

Таблиця 6.2 – Специфікатори параметра *format*

<i>Символ</i>	<i>Значення, що вводиться</i>	<i>Тип аргументу</i>
i, I	десятькове, вісімкове чи шістнадцятькове ціле число	int, long
d, D	десятькове ціле число	int, long
e, E	дійсне число з рухомою крапкою	float
f, F	дійсне число з фіксованою крапкою	float
s	рядок символів	char s[]
c	СИМВОЛ	char

Записування даних до текстового файла

```
int fputs(const char *s, FILE *stream);
```

```
int fprintf(FILE *stream, const char *format [, argument]);
```

Приклад 6.4

```
f=fopen("a.txt","at+");  
if(f==0)  
    { ShowMessage("Не вдається створити файл "); return;}  
char s[40];  
// Копіюємо рядок з Edit1 до s, перетворюючи його на C-рядок.  
strcpy(s,Edit1->Text.c_str());  
// Долучаємо до рядка символ <Enter>, інакше у файлі  
strcat(s, "\n"); // все буде записано одним рядком.  
fputs(s,f);      // Записуємо рядок s до файла.  
fclose(f);
```

```
char s[20];  
strcpy(s, "Іванов");  
int year=1985;  
fprintf(F, "ХАРАКТЕРИСТИКА\nспівробітник %s, %i р.н.\n",  
        &s, year);
```

ХАРАКТЕРИСТИКА

співробітник Іванов, 1985 р.н.

Рисунок 6.4

Визначення поточної позиції файла

long int ftell(FILE *stream);

Зміна поточної позиції файла

int fseek(FILE *stream, long offset, int whence);

Таблиця 6.3 – Параметр whence

Константа	whence	Точка відліку
SEEK_SET	0	початок файла
SEEK_CUR	1	поточна позиція
SEEK_END	2	кінець файла

Приклад 6.5

```
fseek(f, 0, SEEK_SET);
```

```
fseek(f, 0, 0);
```

Визначення сумарного обсягу пам'яті у байтах, який займає файл

```
fseek(f, 0, SEEK_END);  
int d = ftell(f);
```

Послідовність роботи з файлом при записуванні даних

- Відкрити чи то створити файл `fname.txt` для записування (або для зчитування й записування) у кінець файла:

```
f=fopen("fname.txt", "at+");
```

- Записати дані до файла `f` однією з команд:

```
fputs(s, f); // Записування C-рядка s
```

```
//Форматоване записування C-рядка nazva, дійсного числа price і цілого числа kol
```

```
fprintf(f, "%s %6.2f %i\n", nazva, price, kol);
```

- Закрити файл:

```
fclose(f);
```

Послідовність роботи при зчитуванні даних з файла

- Відкрити файл для зчитування

`f=fopen("fname.txt", "rt+");`

- Здійснити зчитування даних з файла однією з команд:

***`fgets(s, 80, f);`** // Зчитування C-рядка s довжиною у 80 символів*

// Форматоване зчитування за адресами відповідних змінних

`fscanf(f, "%s %f %i\n", &nazva, &price, &kol);`

- Для зчитування послідовно всіх даних з файла треба організувати цикл з умовою, яка перевіряє досягання кінця файла, використовуючи чи то функцію **`feof(f)`**, яка є ідентична до true при досяганні кінця файла, чи функцію зчитування даних, яка дає нульовий результат за досягання кінця файла. Можна використовувати інформацію про сумарний розмір файла і контролювати досягання кінця файла функцією **`ftell(f)`**.
- Закрити файл:

`fclose(f);`

Введення даних до файла з числового масиву

Перетворення елементів масиву на C-рядки, які записати до файла за допомогою функції `fputs()`.

```
char s[80];  
for(int i=0; i<10; i++)  
{ itoa(a[i],s,10);      // Перетворення елемента на C-рядок,  
  strcat(s, "\n");      // долучення символу переведення на новий рядок  
  fputs(s, f);          // і записування елемента до файла.  
}
```

```
for(int i=0; i<10; i++)  
{ itoa(a[i], s, 10);    // Перетворюємо елемент на C-рядок,  
  strcat(s, " ");       // додаємо пробіл після чергового числа  
  fputs(s, f);          // і записуємо число з пробілом до файла.  
}  
fputs("\n", f);
```

```
char s[80]="", s1[10];  
for(int i=0; i<10; i++) // До рядка долучається черговий елемент  
    { itoa(a[i], s1, 10); strcat(s, s1); } // у символній формі.  
strcat(s, "\n");  
fputs(s, f);
```

Запис елементу до файла за допомогою функції форматованого введення даних `fprintf()`:

```
for(int i=0; i<10; i++)    fprintf(ft, "%6.2f\n", a[i]);
```

Матрицю до файла можна записати у наступні способи

- *Кожен елемент матриці перетворити до C-рядка і записати до файла окремо, після завершення кожного рядка матриці приєднати символ :*

```
for(int i=0; i<10; i++)
{ for(int j=0; j<10; j++)
  { itoa(a[i][j],s,10); // Перетворення числа на C-рядок,
    strcat(s,"\t");      // долучення табуляції після чергового числа
    fputs(s,f);    }      // і записування числа з пробілом до файла.
  fputs("\n",f); //Записування символу переведення курсора у кінець рядка.
}
```

- *Сформувати C-рядок з елементів кожного рядка матриці й записати його до файла:*

```
char s[80]="";  
for(int i=0; i<10; i++)  
{ for(int j=0; j<10; j++)  
  { itoa(a[i][j],s,10); // Перетворення елемента на C-рядок,  
    strcat(s, "\t");    // долучення табуляції після чергового числа  
  }  
  strcat(s, "\n"); // і символу переведення позиції на кінець рядка.  
  fputs(s, f);    // Записування рядка матриці до файла.  
}
```

- *Кожен елемент матриці записати до файла у числовій формі за допомогою функції fprintf():*

```
for(int i=0; i<10; i++) // Записування елемента  
{ for(int j=0; j<10; j++) fprintf(ft,"%6.2f\n",a[i][j]);  
  fputs("\n", f);      // і символу переведення курсора у кінець рядка.  
}
```

Приклад 6.5 Заповнити матрицю 5×5 випадковими числами і записати цю матрицю до текстового файла.

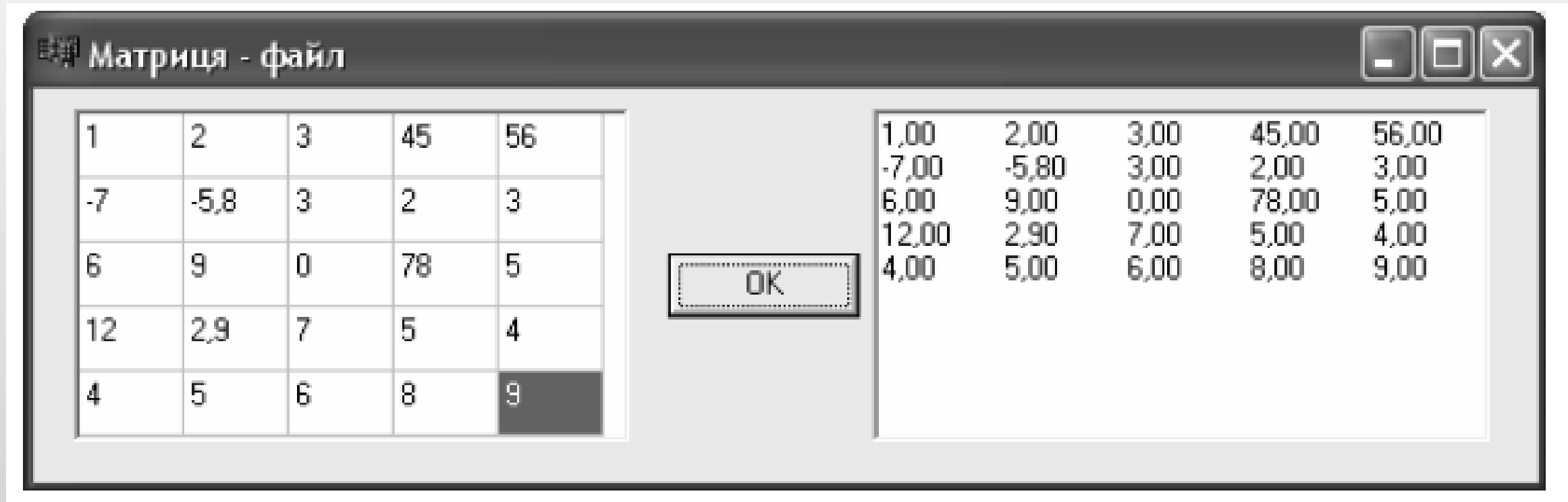


Рисунок 6.5


```

#include <stdio.h>
float a[5][5];
FILE* f;
void __fastcall TForm1::Button1Click(TObject *Sender)
{ int i,j,c; char s[100];
  randomize();
  for(i=0; i<5; i++)
  for(j=0; j<5; j++)
  { StringGrid1->Cells[j][i]=((1000-random(2000))/10.);
    a[i][j]=StrToFloat(StringGrid1->Cells[j][i]);
  }
  if((f=fopen("my_text.txt", "wt"))==0)
  { ShowMessage("Файл не відкрито "); return; }
  for(int i=0; i<5; i++)
  { for(int j=0; j<5; j++)
    { // Перетворення елемента спочатку на AnsiString, потім на C-рядок.
      strcpy(s, FloatToStrF(a[i][j], ffFixed,6,2).c_str());
      strcat(s, "\t"); // Долучення табуляції після чергового елемента,
      fputs(s, f); // записування числа з пробілом до файла
    }
    fputs("\n", f); // і символу переведення курсора у кінець рядка.
  }
  fclose(f);
  Mem1->Lines->LoadFromFile("my_text.txt");
}

```

Рисунок 6.6

```
char s[80];  
char *D=" "; // Пробіл – розділювач поміж числами у рядку.  
char *p=new char [80]; // Рядок для числа у символьній формі.  
fgets(s,80,f); // Зчитування рядка,  
p=strtok(s,D); // і виокремлення першого числа.  
while (p) // Допоки рядок не є порожній,  
{ a[i]=atof(p); // перетворюється дійсне число з символьної форми,  
  i++; // збільшується індекс елемента масиву  
  p=strtok(NULL,D); // і виокремлюється наступне число.  
}
```

Рисунок 6.7

6.3 РОБОТА З ТЕКСТОВИМИ ФАЙЛОВИМИ ПОТОКАМИ НА МОВІ ПРОГРАМУВАННЯ C/C++

`ifstream`



вхідні файлові потоки (для зчитування)

`ofstream`



вихідні файлові потоки (для записування)

`fstream`



двонаправлені файлові потоки (для зчитування та записування)

`<fstream.h>`

// Створення (відкриття) вихідного потоку (записування)

```
ofstream outfile("Test.dat");
```

// Створення (відкриття) вхідного потоку (зчитування)

```
ifstream fin ("Test.dat");
```

// Створення (відкриття) введення-виведення (записування і зчитування)

```
fstream f_in_out("Test.dat");
```

Рисунок 6.8

Операції *“помістити в потік”* (<<) і *“взяти з потоку”* (>>)

Перевага операцій, які працюють з текстовими файлами, порівняно з розглянутими у попередніх розділах функціями, є простота використання і автоматичне розпізнавання типів даних.

```
int  i=1, j=25, il, jl; double a=25e6, al;
char  s[40], sl[40];
strcpy(s, "ІВАНОВ");
ofstream fout ("Test.dat"); // Створення файла як вихідного потоку
if(!fout) {ShowMessage("Файл не вдається створити"); return;}
fout << i << ' ' << j << ' ' << a << ' ' << s <<endl;
fout.close();               // Закриття файла
ifstream fin("Test.dat");   // Відкриття файла як вхідного потоку
if(!fin) {ShowMessage("Файл не вдається відкрити"); return;}
fin >> il >> jl >> al >> sl;
fin.close();                // Закриття файла
```

Рисунок 6.9

Таблиця 6.4 – Режими відкривання файлів

Режим	Значення режиму
<code>ios::in</code>	Відкрити файл для зчитування
<code>ios::out</code>	Відкрити файл для записування
<code>ios::trunc</code>	Вилучити дані з файла
<code>ios::nocreate</code>	Якщо файл не існує, відкривання файла не відбудеться
<code>ios::ate</code>	Після відкриття перемістити позицію зчитування-записування на кінець файла
<code>ios::app</code>	Відкрити для записування даних в кінець файла
<code>ios::binary</code>	Відкрити файл, вважаючи його за бінарний (не текстовий)

Приклад, щоб відкрити бінарний файл з ім'ям `Test.dat` як для зчитування, так і для записування, слід записати функцію

```
fstream filename("Test.dat",ios::in|ios::out|ios::binary);
```


Таблиця 6.5 – Відповідність режимів відкривання файлів в C++ і

Режим C++	Режим C	Значення режиму
<code>ios::in</code>	"r"	Відкрити файл для зчитування. Позиція встановиться на початок файла
<code>ios::out</code> <code>ios::out ios::trunc</code>	"w"	Відкрити файл для записування. Позиція встановиться на початок файла. Якщо файл існує, записування здійснюватиметься поверх даних, тобто попередньо відбудеться урізання довжини файла до нуля
<code>ios::out ios::app</code>		
<code>ios::out ios::app</code>	"a"	Відкрити файл для записування даних в кінець файла
<code>ios::in ios::out</code>	"r+"	Відкрити файл для зчитування-записування даних, починаючи з довільної позиції, тобто позиція може змінюватись
<code>ios::in ios::out ios::trunc</code>	"w+"	Відкрити файл для зчитування-записування, урізавши довжину файла до нуля для існуючого файла

Щоб перевірити правильність відкриття файла, можна скористатися умовним оператором

```
if(!filename)  
    { ShowMessage("Неможливо відкрити файл!"); return; }
```

Завершення роботи з файлом здійснюється за допомогою функції

```
filename.close();
```

6.4 ВИКОНАННЯ ПОСЛІДОВНОГО ТА ДОВІЛЬНОГО ЗАПИСУВАННЯ ТА ЗЧИТУВАННЯ ІНФОРМАЦІЇ У ФАЙЛАХ

Послідовне записування до файла і зчитування з файла

Для записування певної інформації до файла використовується операція “помістити в потік”

```
filename << блок1 << блок2 << ... << блокN;
```

Для зчитування з файла певної інформації використовується операція “взяти з потоку”

```
filename >> блок1 >> блок2 >> ... >> блокN;
```


Для зчитування цілого

```
getline (char*, int, char='\n')
```

Приклад 6.6 Увести інформацію про студентів: прізвище, найменування групи, рік народження. Переглянути весь список і вивести дані про студентів, яким на даний момент є понад 18 років.

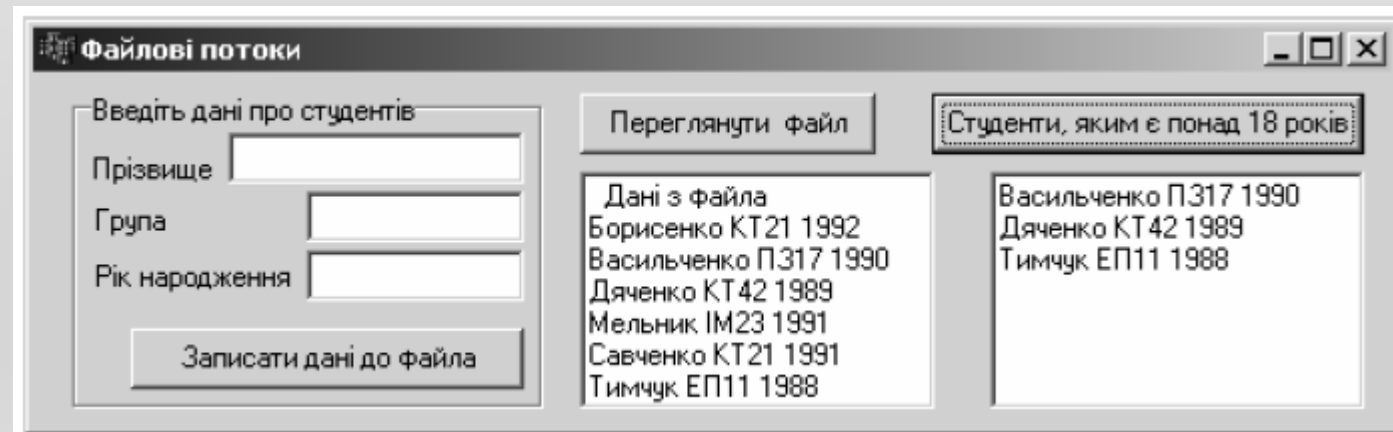


Рисунок 6.10

```
#include <fstream.h>
#include <DateUtils.hpp> // Для функції
                          визначення року YearOf()
// Кнопка "Записати дані до файла".
void __fastcall
TForm1::Button1Click(TObject *Sender)
{ char PIB[50], gr[10]; int god;
  // Відкриття файла для записування даних у
  кінець файла
  ofstream output("dan.txt", ios::app);
  if(!output) // Перевірка правильності
  відкриття файла для записування
  { ShowMessage("Файл не створено або не
  відкрито!"); return; }
  strcpy(PIB,Edit1->Text.c_str());
  strcpy(gr,Edit2->Text.c_str());
  god=StrToInt(Edit3->Text);
  output<<PIB<<' '<< gr<<' '<<god<<endl; //
  Записування до файла
  output.close(); // Закриття файла
  Edit1->Clear(); Edit2->Clear(); Edit3-
  >Clear();
  Edit1->SetFocus();
}
```

```

// Кнопка “Переглянути файл”
void __fastcall TForm1::Button2Click(TObject *Sender)
{ char *s=new char[80];
Memo1->Lines->Add("\t Дані з файла");
ifstream f("dan.txt"); // Відкриття файла для зчитування
if(!f) { ShowMessage("Немає файла"); return;}
while(f.getline(s,80)) Memo1->Lines->Add(AnsiString(s));
f.close(); // Закриття файла
}

// Кнопка “Студенти, яким є понад 18 років”
void __fastcall TForm1::Button3Click(TObject *Sender)
{ char PIB[50], ff[10]; int god;
Memo2->Clear();
fstream f("dan.txt",ios::in); // Відкриття файла для зчитування
if(!f) { ShowMessage("Немає файла"); return;}
while(f >> PIB >> gr >> god)
if(YearOf(Date())-god > 18)
Memo2->Lines->Add(String(PIB)+" "+String(gr)+" "+IntToStr(god));
f.close();
}

```

Довільне записування до файла і зчитування з файла

Довільне записування і зчитування даних означає, що записування даних до файла і зчитування цих даних здійснюватимуться з довільної зазначеної позиції файла.

Позиціонування виконується за допомогою методу *seekp()* класу *ifstream* при записуванні даних до файла, і за допомогою методу *seekg()* класу *ofstream* — при зчитуванні даних з файла.

Константні значення позицій:

ios::beg — початок потоку (встановлюється за замовчуванням);

ios::end — кінець потоку;

ios::cur — поточна позиція потоку.

Приклад 6.7:

f.seekg(n); – позиціонування на n-й байт від початку файла;

f.seekg(n, ios::cur); – позиціонування на n-й байт уперед від поточної позиції;

f.seekp(k, ios::end); – позиціонування на k-тий байт від кінця файла;

f.seekp(0, ios::end); – позиціонування на кінець файла.

Методи *tellg()* та *tellp()* повертають поточну позицію файла для потоків введення та виведення.

Для організації записування використовується маніпулятор *setw(num)*, який задає довжину поля виведення з num позицій. Для його використання слід долучити заголовний файл *<iomanip.h>*.

Приклад 6.8

```
fout<<setw(15)<<name<<setw(20)<<surname<<setw(7)<<year<<endl;
```

Приклад 6.9. Записати до файла дані про студентів: прізвище, найменування групи, рік народження. Запрограмувати можливість змінювати у файлі той рядок (запис), номер якого зазначить користувач у текстовому полі (зауважимо, що рядки нумеруються з 0). Надати можливість переглядання файла, розпочинаючи з зазначеного запису, – номер запису теж задає користувач.

Введіть дані про студентів

Прізвище

Група

Рік народження

Дані з файла

Борисенко	ІМ23	1990
Васильченко	ПЗ12	1992
Горбанюк	КТ43	1993
Горобець	КТ10	1989
Картатий	ІМ32	1992

Введіть номер запису файла (нумерація з 0)

Дані з файла із запису № 2

Горбанюк КТ43 1993

Дорошенко КТ11 1991

Картатий ІМ32 1992

Введіть номер запису (нумерація з 0), який треба замінити

Рисунок 6.11

```

#include <iomanip.h>
#include <fstream.h>
// Кнопка "Записати дані до файла"
void __fastcall TForm1::Button1Click(TObject *Sender)
{ char PIB[50], gr[10]; int god;
// Відкриття файла для записування даних у кінець
// файла
ofstream output("dan.txt", ios::app);
if(!output) // Перевірка правильності відкриття файла
для записування
{ ShowMessage("Файл не створено або не відкрито!");
return; }
strcpy(PIB, Edit1->Text.c_str());
strcpy(gr, Edit2->Text.c_str());
god=StrToInt(Edit3->Text);
// Записування до файла з використанням маніпулятора
setw
output<<setw(15)<<PIB<<setw(8)<<gr<<setw(7)<<god<<endl;
output.close(); // Закриття файла
Edit1->Clear(); Edit2->Clear(); Edit3->Clear();
Edit1->SetFocus();
}
// Кнопка "Переглянути файл"
void __fastcall TForm1::Button2Click(TObject *Sender)
{ char *s=new char[80];
Memo1->Clear();

```

```

Memo1->Lines->Add("\t Дані з файла");
ifstream f("dan.txt"); // Відкриття файла для
зчитування
if(!f) { ShowMessage("Немає файла"); return;}
while(f.getline(s,80)) Memo1->Lines-
>Add(AnsiString(s));
f.close(); // Закриття файла
}
// Кнопка "Замінити рядок файла із зазначеним номером"
void __fastcall TForm1::Button3Click(TObject *Sender)
{ char PIB[50], gr[10]; int god;
ofstream output("dan.txt",ios::in|ios::out);
if(!output)
{ ShowMessage("Файл не створено або не відкрито!");
return; }
int k=StrToInt(Edit4->Text);
// Уведення нових значень k-го рядка.
strcpy(PIB, Edit1->Text.c_str());
strcpy(gr, Edit2->Text.c_str());
god=StrToInt(Edit3->Text);
output.seekp(k*32); // Позиціонування у k-тий рядок
// файла
// Записування з використанням маніпулятора setw
output<<setw(15)<<PIB<<setw(8)<<gr<<setw(7)<<god<<endl;
output.close();
}

```

// Кнопка “Переглянути файл, розпочинаючи із зазначеного запису”

```
void __fastcall TForm1::Button4Click(TObject *Sender)
{ int k=StrToInt(Edit5->Text); // Введення номеру запису
  char PIB[50], gr[10]; int god;
  Memo2->Clear();
  Memo2->Lines->Add(" Дані з файла із запису №
  "+IntToStr(k));
  ifstream fin("dan.txt");
  if(!fin) { ShowMessage("Немає файла"); return;}
  fin.seekg(k*32); // Позиціонування у k-тий рядок файла
  while(fin>>PIB>>gr>>god) // Читаються дані з файла
  Memo2->Lines->Add(String(PIB)+" "+String(gr)+" "+
  IntToStr(god));
  fin.close();
}
```


6.5 ЗАБЕЗПЕЧЕННЯ ОПРАЦЮВАННЯ ТЕКСТОВИХ ФАЙЛІВ ЗА ДОПОМОГОЮ ДЕСКРИПТОРІВ

Дескриптор (Handle) – унікальний ідентифікатор цілого типу.

Функції опрацювання файлів через дескриптори

int FileCreate (AnsiString FileName); Створює файл з ім'ям FileName і повертає дескриптор створеного файла, який можна в подальшому використовувати для роботи з цим файлом. Якщо файл не вдалося створити, функція повертає значення -1

int FileOpen (AnsiString FileName, unsigned Mode); Відкриває файл з ім'ям FileName у режимі Mode і повертає дескриптор відкритого файла

void FileClose(int Handle); Закриває файл з дескриптором Handle

int FileRead(int Handle, void *Buffer, unsigned Count); Читає з файла з дескриптором Handle данні з Buffer розміром Count байтів і повертає значення прочитаних байтів; у разі помилки повертає –1

int FileWrite(int Handle, void *Buffer, unsigned Count); Записує до файла з дескриптором Handle данні з Buffer розміром Count байтів і повертає значення записаних байтів; у разі помилки повертає –1

int FileSeek(int Handle, int Offset, int Origin); Переміщує поточну позицію файла на Offset байтів відносно позиції визначеної параметром Origin

bool DeleteFile (AnsiString FileName); Видаляє файл з ім'ям FileName. Якщо файл не можливо видалити чи він не існує, функція повертає значення false, інакше – true

bool RenameFile (AnsiString OldName, AnsiString NewName); Перейменовує файл, тобто змінює ім'я OldName на NewName. Якщо перейменувати файл не вдалося, функція повертає значення false, інакше – true

Приклад 6.10

```
int f = FileCreate("info.txt");
```

```
int FileOpen(const AnsiString FileName, unsigned Mode);
```

fmOpenRead – відкрити лише для зчитування з файла;

fmOpenWrite – відкрити лише для записування у файл;

fmOpenReadWrite – відкрити і для зчитування і для записування (режим редагування файла);

fmCreate – за умови наявності файла, відкрити його для записування, інакше – створити новий файл. На відміну від інших констант, які означено в модулі *<SysUtils>*, цю константу означено у *<classes.h>*.

```
int f = FileOpen(fmOpenWrite, "info.txt");  
FileClose(f);
```

Переміщення поточної позиції файла

```
int FileSeek(int Handle, int Offset, int Origin);
```

Значення параметра Origin	Спосіб відліку
0	від початку файла
1	від поточної позиції
2	від кінця файла

```
FileSeek (f, 0, 0) ;
```

```
FileSeek (f, 1, 1) ;
```

Приклад 6.11 Створити файл з інформацією про чисельність населення областей України: назва області, обласний центр, кількість мешканців області. Вивести області, в яких кількість жителів перевищує 1 500 000 людей.

Опрацювання текстового файла через дескриптор

Введіть данні по області

Назва області

Обласний центр

Кількість жителів

Визначити розмір файла

Назва області	Обласний центр	Кільк. жителів
Дніпропетровська	Дніпропетровськ	3500000
Волинська	Луцьк	250000
Одеська	Одеса	2400000
Закарпатська	Ужгород	560000
Черкаська	Черкаси	650000

Вивести області, в яких мешкає більше 1 500 000 жителів

Дніпропетровська Дніпропетровськ 3500000

Одеська Одеса 2400000

Рисунок 6.12

```

int f; // Дескриптор оголошено глобально
__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner) // Конструктор форми
{ SG1->Cells[0][0]="Назва області";
SG1->Cells[1][0]="Обласний центр";
SG1->Cells[2][0]="Кільк.жителів";
SG1->ColWidths[0]=80;
SG1->ColWidths[1]=70;
SG1->ColWidths[2]=70;
} // Тут SG1 – нове значення властивості
Name компонента StringGrid1
void __fastcall TForm1::FormCreate(TObject
*Sender)
{ if( FileExists("obl.txt") )// Перевірка
наявності файла
f = FileOpen("obl.txt",fmOpenWrite);// і
відкриття його.
else // Якщо файл є відсутній,
f = FileCreate("obl.txt");// створити його.
FileClose(f);
}

```

```

// Кнопка “Записати до файла”.
void __fastcall TForm1::Button1Click(TObject
*Sender)
{ AnsiString st;
if(Edit1->Text==" " || Edit2->Text==" " || Edit3-
>Text==" ")
{ ShowMessage("Не всі данні введено!");
Edit1->SetFocus();
return;
}
st=Edit1->Text+" "+Edit2->Text+" "+
StrToInt(Edit3->Text)+ "\r\n";
// Відкриття файла для зчитування і
записування.
f=FileOpen("obl.txt",fmOpenWrite);
if(f != -1) // Якщо помилка при відкритті файла
не було,
{ FileSeek(f,0,2); // позиція переміститься на
кінець файла.
FileWrite(f,st.c_str(),st.Length());
FileClose(f);
}
}

```

```

else // У разі помилки доступу до файла, виведеться
повідомлення.
{ ShowMessage("Помилка доступу до файла"); return; }
Edit1->Clear(); Edit2->Clear(); Edit3->Clear();
Edit1->SetFocus();
}
// Для зчитування рядків з файла створено окрему функцію
GetLine(),
// яка читає рядки символів від поточної позиції до символу
"новий рядок"
// і повертає значення кількість прочитаних символів.
int GetLine(int f, AnsiString *st)
{ unsigned char buf[256]; // Оголошення рядка (буфера),
unsigned char *p = buf; // покажчика на рядок,
int n; // цілих змінних для кількості прочитаних байтів
int len = 0; // і довжини рядка.
// У циклі читаються данні типу char по одному байту, тобто
посимвольно.
while(n != 0) // Допоки є що читати,
{ if(*p == '\r') // перевіряється чи є зчитаний символ
"кінцем" рядка.
{ n = FileRead(f, p, 1); // Прочитати символ '\n'.
break;
}
len++;
p++;
n = FileRead(f, p, 1);
}

```

```

*p = '\0'; // Записування у рядок символу
кінця рядка
if(len != 0) st->printf("%s", buf);
return len;
}
// Кнопка "Переглянути файл"
void __fastcall
TForm1::Button2Click(TObject *Sender)
{ AnsiString st;
bool d=true;
f=FileOpen("obl.txt", fmOpenReadWrite);
if(f == -1)
{ ShowMessage("Файл не відкрито"); return;
}
while(GetLine(f,&st) != 0)
{ if(d)
{ SG1->Rows[SG1->Row]->DelimitedText=st;
d=false; }
else
{ SG1->RowCount++;
SG1->Row=SG1->RowCount-1;
SG1->Rows[SG1->Row]->DelimitedText=st;
}
}
FileClose(f);
}

```

```
// Кнопка "Визначити розмір файла"
void __fastcall TForm1::Button3Click(TObject *Sender)
{ f=FileOpen("obl.txt",fmOpenReadWrite);
if(f==-1)// У разі помилки доступу до файла, вивести відповідне
повідомлення
{ ShowMessage("Файл не відкрито"); return; }
int k=FileSeek(f,0,2); // Переведення позиції на кінець файла
Edit4->Text=IntToStr(k);
FileClose(f);
}
// Вивести області, в яких мешкає більше 1 500 000 жителів
// Кнопка "Перший спосіб розв'язання"
void __fastcall TForm1::Button4Click(TObject *Sender)
{ f=FileOpen("obl.txt", fmOpenReadWrite);
if(f == -1)
{ ShowMessage("Файл не відкрито"); return; }
Memo1->Clear();
char *buf, *stw, *p; AnsiString st, bt;
while(GetLine(f,&st) != 0)
{ stw=st.c_str();
bt=AnsiString(stw);
p=strpbrk(stw,"0123456789"); /* strpbrk() відшукує місце першого
входження в рядок stw будь-якого символу з другого рядка і повертає
частину
рядка s1, починаючи з цього символу і до кінця рядка, тобто в p
запишеться
кінцівка рядка зі значенням кількості мешканців. */
if((StrToInt(p)>=1500000)) Memo1->Lines->Add(bt);
}
FileClose(f);
}
```

```
// Для другого способу створимо функцію
GetString() зчитування рядка st по словах,
// яка повертає кількість прочитаних символів,
// а параметр flag дозволяє визначити кінець
рядка.
int GetString(int f, AnsiString *st, int &flag)
{ unsigned char buf[256]; // Оголошення рядка
(буфера),
unsigned char *p = buf; // покажчика на рядок,
int n; // цілих змінних для кількості
прочитаних байтів
int len = 0; // і довжини рядка.
do n = FileRead(f, p, 1); while((n != 0) && (*p
== ' '));
while((n != 0) && (*p != ' '))
{ if(*p == '\r')
{ n=FileRead(f, p, 1); // Прочитати символ '\n'
flag=1;
break;
}
len++; p++;
n=FileRead(f, p, 1);
}
*p = '\0';
if(len != 0) st->printf("%s", buf);
return len;
}
```

```

// Кнопка “Другий спосіб розв’язання”
void __fastcall TForm1::Button5Click(TObject *Sender)
{ Memo1->Clear();
  AnsiString st, buf, strz="";
  int ls, flag=0;
  f=FileOpen("obl.txt", fmOpenReadWrite);
  if(f==-1) { ShowMessage("Файл не відкрито"); return; }
  do
  { ls=GetString(f,&st,flag);
    if(ls!=0)
    { buf=st;
      for(int i=st.Length();i<=15;i++) st=st+" ";
      strz += st+" ";
    }
    bool t=false;
    for(int i=1;i<=buf.Length();i++)
    if(!(buf[i]>='0'&& buf[i]<='9'))
    { t=true; break; }
    if(!t&&(StrToInt(buf)>=1500000))
    Memo1->Lines->Add(strz);
    if(flag==1)
    { strz="";
      flag=0;
    }
  } while(ls!=0);
  FileClose(f);
}

```


6.6 ОСНОВИ РОБОТИ З БІНАРНИМИ ФАЙЛАМИ

У бінарному файлі число, на відміну від текстового, зберігається у внутрішньому його поданні.

Для відкривання бінарного файла використовується та сама команда ***fopen()***, лише у другому параметрі (режимі відкривання файла) замість літери “***t***” треба записати літеру “***b***”.

Записування і зчитування у двійкових файлах найчастіш здійснюються за допомогою відповідно функцій ***fwrite()*** та ***fread()***.

Приклад 6.12

Відкриття бінарного файлу для зчитування

```
f = fopen("tmp.dat", "rb");
```

Прототип функції зчитування *fread()*

```
size_t fread  
{ char *buffer, // Масив для зчитування даних,  
size_t elemSize, // розмір одного елемента,  
size_t numElems, // кількість елементів для зчитування  
FILE *f // і покажчик потоку  
};
```

Приклад 6.13

```
FILE *f;  
double buff[100];  
size_t res;  
f = fopen ("tmp.dat", "rb"); // Відкриття файла  
if(f == 0)  
    { ShowMessage ("Не можу відкрити файл для  
зчитування");  
        exit (1); // Завершення роботи з кодом 1  
    }  
// Зчитування 100 дійсних чисел з файла  
res = fread(buff, sizeof(double), 100, f); // res дорівнює  
реальній кількості прочитаних чисел
```

Функція бінарного записування до файла ***fwrite()*** є аналогічна до функції зчитування ***fread()***.

```
size_t fwrite  
{ char *buffer, // Масив записуваних даних,  
  size_t elemSize, // розмір одного елемента,  
  size_t numElems, // кількість записуваних елементів  
  FILE *f // і покажчик потоку.  
};
```

Приклад 6.14

```
FILE *f;  
double buff[100];  
size_t res;  
... // Введення елементів масиву buff[100]  
f = fopen("tmp.dat", "wb"); // Створення бінарного файла  
if(f == 0)  
    { ShowMessage ("Не можу відкрити файл для запису");  
      exit(1); // Завершення роботи програми з кодом 1  
    }  
// Записування 100 дійсних чисел до файла  
res = fwrite(buff, sizeof(double), 100, f); // У разі успішного  
записування res = 10
```

До обох функцій передається покажчик на дані, які виводяться чи вводяться, параметр *elemSize* задає розмір передаваних даних в байтах, а параметр *numElems* визначає кількість таких даних.

У бінарних файлах можна визначати позицію зчитування-записування і переміщувати її у довільне місце файла командами відповідно *ftell()* та *fseek()*.

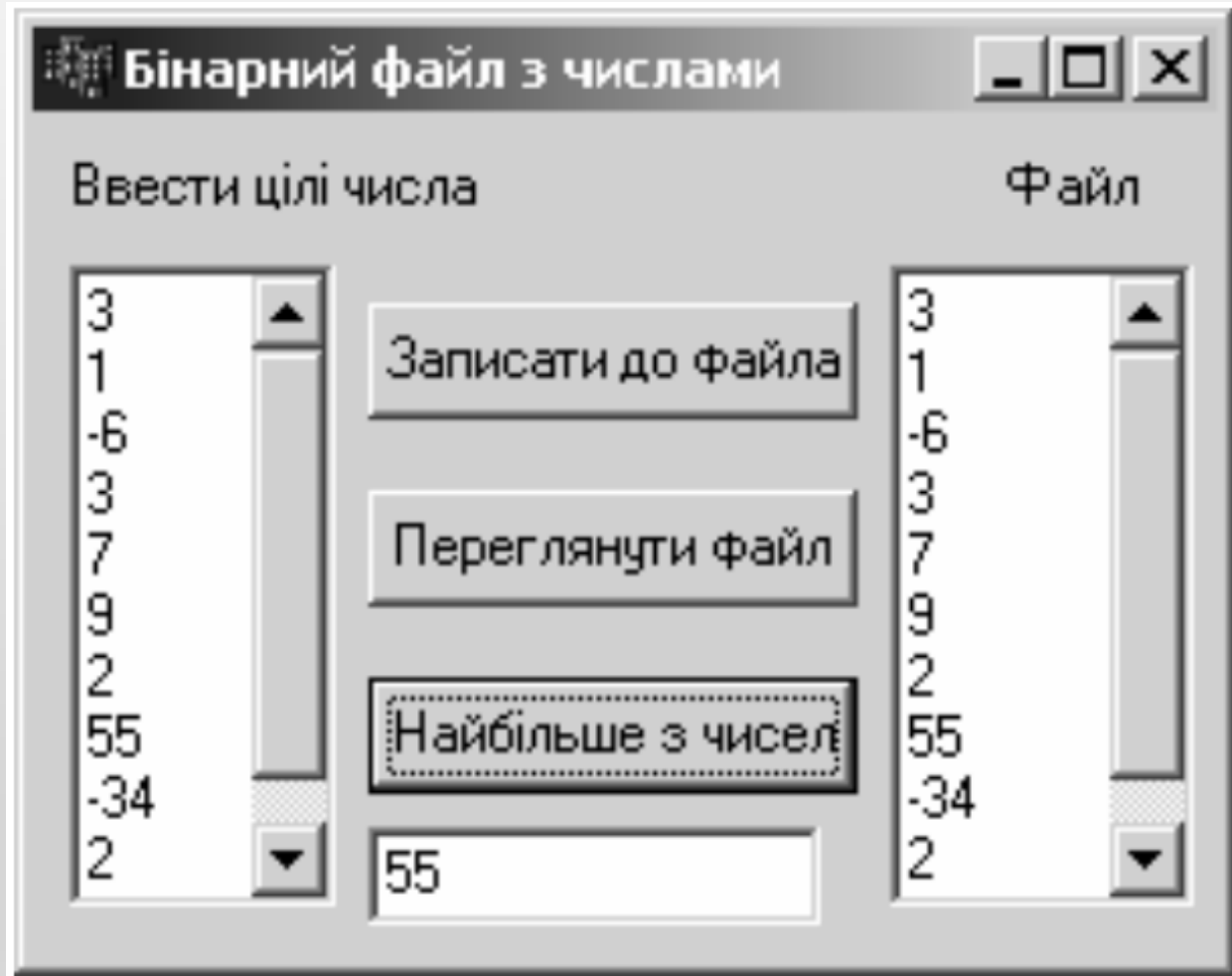
```
fseek(F, sizeof(double)*(i-1), 0);  
fread(&a, sizeof(double), 1, F);
```

Приклад 6.15

```
fread(&z, sizeof(double), 1, F);  
z *= 10;  
fseek(F, -sizeof(double), 1);  
fwrite(&z, sizeof(double), 1, F);
```

```
long int pos = ftell(F); // Запам'ятовування позиці.  
fread(&z, sizeof(double), 1, F); // Зчитування числа з файла до  
змінної  
z *= 10; fseek(F, pos, 0); // Відновлення позиці  
fwrite(&z, sizeof(double), 1, F); // Записування числа зі змінної з  
до файл
```

Приклад 6.16 Створити бінарний файл, який містить цілі числа, і віднайти серед них найбільше число.



```
#include <stdio.h>
FILE* f;
// Кнопка "Записати до файла".
void __fastcall TForm1::Button1Click(TObject
*Sender)
{ int z, i, N=Memo1->Lines->Count;
  if((f=fopen("num.dat", "wb"))==0) // Відкриття
  файла для створення
  { ShowMessage("Не вдається створити файл");
  return;}
  for(i=0; i<N; i++)
  { z=StrToInt(Memo1->Lines-
>Strings[i]); //Читання числа z з Memo1
    fwrite(&z, sizeof(int),1,f); // Записування
    числа z до файла
  }
  fclose(f); }
```

Рисунок 6.13


```

// Кнопка “Переглянути файл”
void __fastcall TForm1::Button2Click(TObject
*Sender)
{ int z; Memo2->Clear();
  if((f=fopen("num.dat", "rb"))==0) // Відкриття
файла для зчитування
  { ShowMessage("Не вдається відкрити файл");
return; }
  while(fread(&z, sizeof(int), 1, f)) //
Зчитування числа
  Memo2->Lines->Add(IntToStr(z));
  fclose(f);
}

```

```

// Кнопка “Найбільше з чисел”
void __fastcall TForm1::Button3Click(TObject
*Sender)
{ int z, max;
  if((f=fopen("num.dat", "rb"))==0) // Відкриття
файла для зчитування
  { ShowMessage("Не вдається відкрити файл");
return; }
  fread(&z, sizeof(int), 1, f); // Зчитування
першого числа
  max= z; // і запам'ятовування його в max.
  while(fread(&z, sizeof(int), 1, f)) //
Зчитування числа.
  if(max<z) max=z; //Якщо число більше за max,
воно запам'ятовується в max.
  fclose(f);
  Edit1->Text=IntToStr(max);
}

```

6.7 ОСНОВИ РОБОТИ З БІНАРНИМИ ФАЙЛОВИМИ ПОТОКАМИ

Робота з бінарним форматом файла – *ios::binary*.

Дані до бінарних файлів записуються за допомогою методу *write()* класу *ofstream*, а зчитуються – за допомогою методу *read()* класу *ifstream*

```
fout.write((char*)&A, sizeof(A));  
fin.read((char*)&A, sizeof(A));
```

У бінарному файлі також існує можливість довільного доступу до файла за допомогою методів *seekp()* та *seekg()*. При повторному відкритті файла з використанням тієї ж самої файлової змінної потік слід очищувати за допомогою методу *clear()*.

Приклад 6.17 Створити файл з інформацією про успішність студентів: прізвище, ім'я, оцінки з інформатики, математики та фізики. Вивести на екран середній бал кожного студента. Виявити, чи є серед студентів збіг прізвищ (однофамільці).

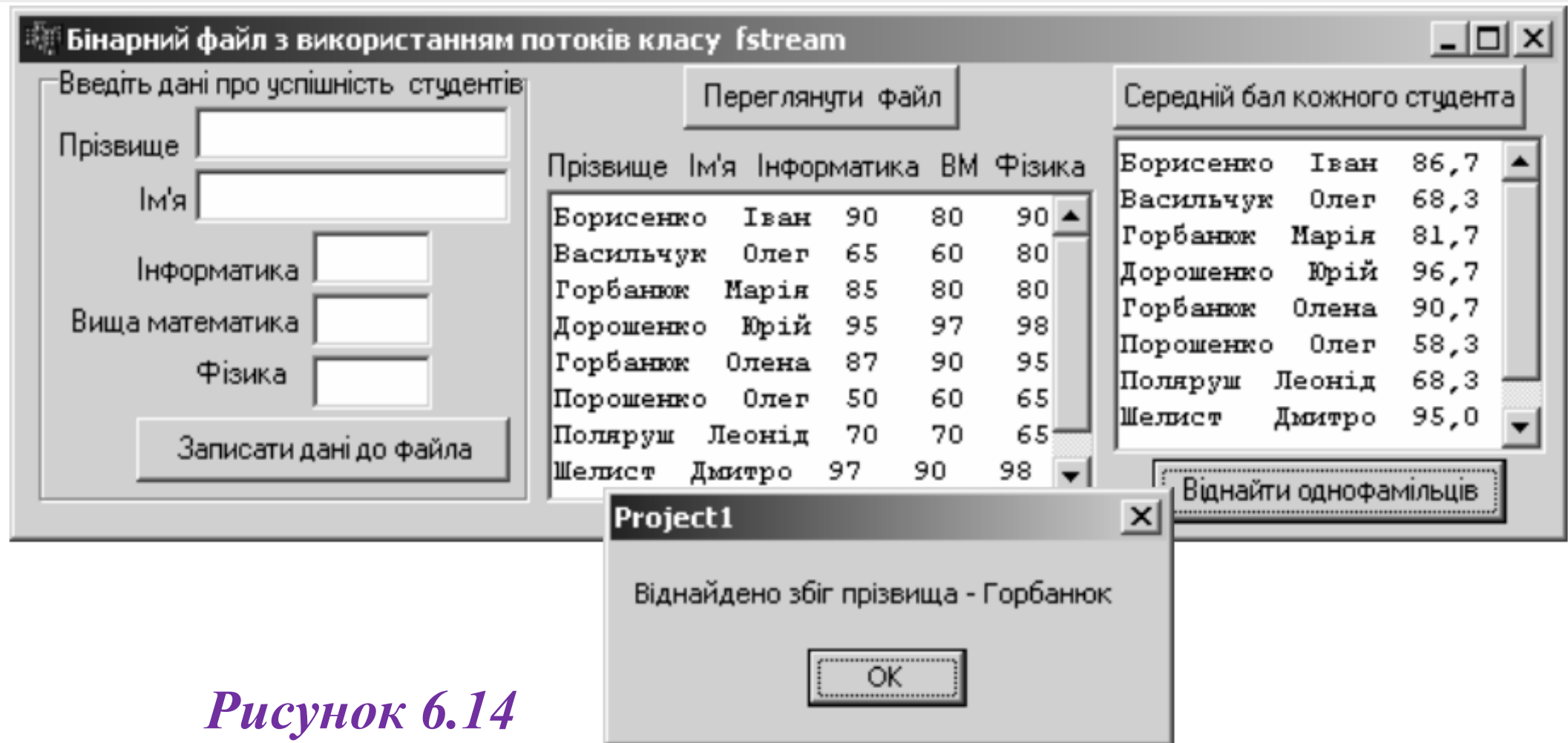


Рисунок 6.14

```
#include <fstream.h>
fstream f;
char *filename=new char[30];
struct student
{ char name[30], surname[20];
  int inf, mat, fiz; };
student A;
void __fastcall TForm1::FormCreate(TObject *Sender)
{ AnsiString fn=InputBox("Відкриття файла",
  "Введіть ім'я файла", "stud.dat");
  if(fn=="") { ShowMessage("Ви не ввели дані"); return; }
  strcpy(filename,fn.c_str());
  // Створення чи відкриття файла
  f.open(filename,ios::out|ios::app|ios::binary);
  f.clear(); // Очищення потоку
  if(!f) { ShowMessage("Помилка відкриття файла"); return; }
  f.close();
}
```

```
// Кнопка “Записати дані до файла”
void __fastcall TForm1::Button1Click(TObject *Sender)
{ f.open(filename,ios::out|ios::app|ios::binary);
  f.clear();
  if(!f) { ShowMessage("Помилка відкриття файла "); return; }
  f.seekp(0,ios::end); // Позиціонування на кінець файла
  strcpy(A.name, Edit1->Text.c_str());
  strcpy(A.surname, Edit2->Text.c_str());
  A.inf=StrToInt(Edit3->Text);
  A.mat=StrToInt(Edit4->Text);
  A.fiz=StrToInt(Edit5->Text);
  f.write((char*)&A, sizeof(A)); // Записування даних до файла
  f.close();
}
```

```
// Кнопка “Переглянути файл”
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    Memo1->Clear();
    f.open(filename, ios::out|ios::in|ios::binary);
    f.clear();
    if(!f)
    {
        ShowMessage("Помилка відкриття файла "); return; }
    while(f.read((char*)&A, sizeof(A))) // Зчитування з файла
        Memo1->Lines->Add(AnsiString(A.name)+" "+
            AnsiString(A.surname)+" "+ IntToStr(A.inf)+" "+
            IntToStr(A.mat)+" "+IntToStr(A.fiz));
    f.close();
}
```

```
// Кнопка “Середній бал кожного студента”
void __fastcall TForm1::Button3Click(TObject *Sender)
{ f.open(filename,ios::out|ios::in|ios::binary);
  f.clear();
  Memo2->Clear();
  if(!f) { ShowMessage("Помилка відкриття файла"); return; }
  double sr;
  while(f.read((char*)&A, sizeof(A)))
  { sr=(A.inf + A.mat + A.fiz)/3.; // Обчислення середнього
    балу
    Memo2->Lines->Add(AnsiString(A.name) + " " +
      AnsiString(A.surname)+" "+FormatFloat("0.0",sr));
  }
  f.close();
}
```

```

// Кнопка "Віднайти однофамільців"
void __fastcall TForm1::Button4Click(TObject *Sender)
{ f.open(filename,ios::out|ios::in|ios::binary);
  f.clear();
  if(!f) { ShowMessage("Помилка відкриття файла"); return; }
  student A1; long int k=0;
  while(f.read((char*)&A, sizeof(A)))
  { while(f.read((char*)&A1, sizeof(A1)))
    if(strcmp(A.name,A1.name)==0)
    { ShowMessage("Віднайдено збіг прізвища - " +
      AnsiString(A.name));
      return;
    }
    f.clear();
    k++;
  }
  // Позиціонування на структуру файла з номером k
  f.seekg(k*sizeof(A),ios::beg);
  }
  ShowMessage("Збігу прізвищ немає");
  f.close();
}

```


6.8 ВИКОНАННЯ ОПРАЦЮВАННЯ БІНАРНИХ ФАЙЛІВ ІЗ ЗАСТОСУВАННЯМ ДЕСКРИПТОРІВ

Порядок дій для записування даних у файл

```
int f; //Оголошення дескриптора  
f=fopen("a.dat",fmOpenWrite); //Відкриття файла для  
записування  
int kb=FileSeek(f, 0, 2); //Встановлення позиції на кінець файла  
//і одночасне визначення розміру файла  
int k=25;  
//Записування до файла значення k цілого типу розміром  
sizeof(int)  
FileWrite(f, &k, sizeof(int));  
FileClose(f); //Закриття файла
```

Відкриття файла і зчитування даних

```
f=FileOpen("a.dat", fmOpenRead);  
FileRead(f, (int*)&k, sizeof(int));  
FileClose(f);
```

Відкриття файла для зчитування-записування

```
f=FileOpen("a.dat",fmOpenReadWrite);  
if(f != -1)  
/* Далі можна опрацьовувати файл: читати чи  
записувати данні, встановивши потрібну позицію  
за допомогою функції FileSeek(). */
```

Приклад 6.18 Створити файл з інформацією про авіарейси: номер рейса, маршрут руху, час відправлення (за шаблоном ЧЧ:ММ), час прибуття в пункт призначення (за шаблоном ЧЧ:ММ), при чому з клавіатури мають вводитися: номер рейса, маршрут руху, час відправлення і тривалість польоту, а час прибуття в пункт призначення слід обчислити. Передбачити можливість вилучення даних про нічні рейси (з 00:00 по 5:00). Відсортувати данні за часом відправлення літаків.

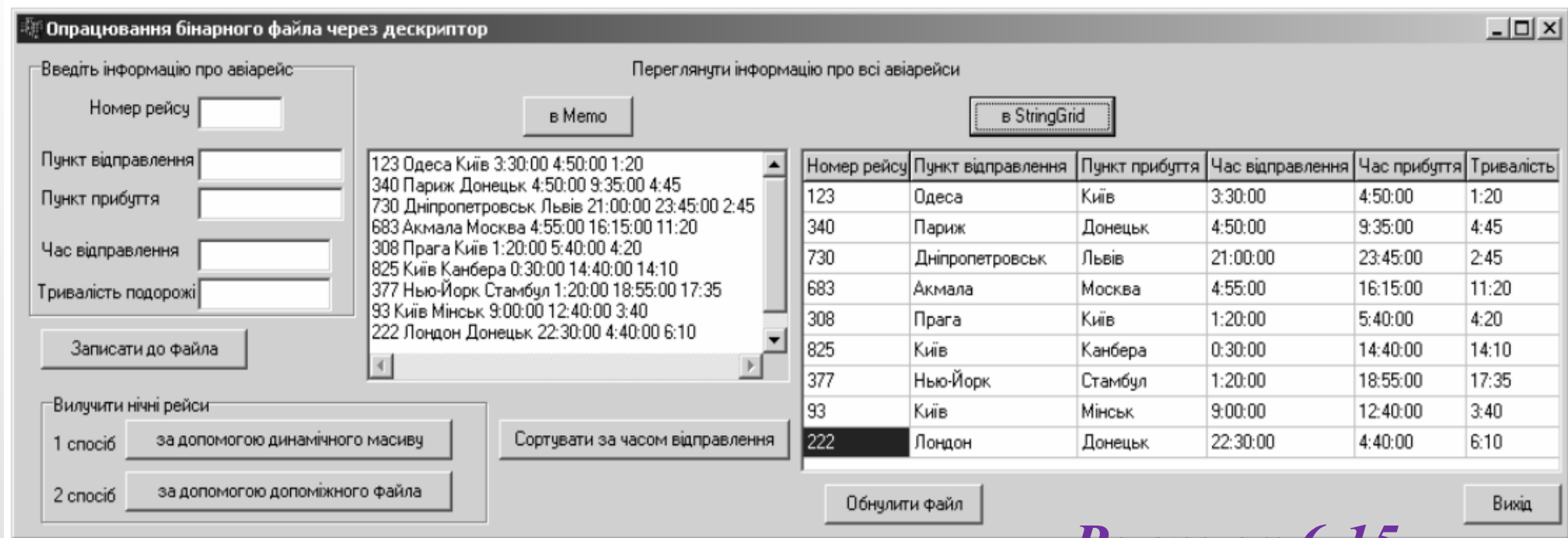


Рисунок 6.15

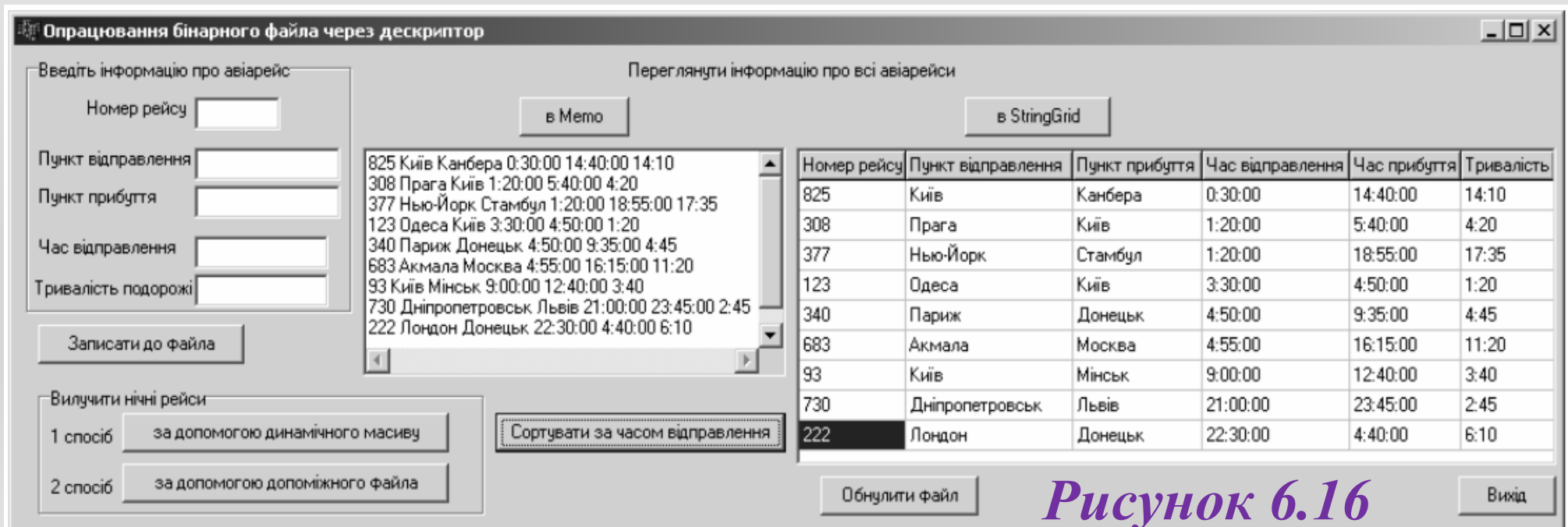


Рисунок 6.16

Опрацювання бінарного файла через дескриптор

Введіть інформацію про авіарейс

Номер рейсу

Пункт відправлення

Пункт прибуття

Час відправлення

Тривалість подорожі

Записати до файла

Вилучити нічні рейси

1 спосіб

2 спосіб

Сортувати за часом відправлення

Переглянути інформацію про всі авіарейси

Номер рейсу	Пункт відправлення	Пункт прибуття	Час відправлення	Час прибуття	Тривалість
93	Київ	Мінськ	9:00:00	12:40:00	3:40
730	Дніпропетровськ	Львів	21:00:00	23:45:00	2:45
222	Лондон	Донецьк	22:30:00	4:40:00	6:10

Обнулити файл

Вихід

Рисунок 6.17

```
int MessageDlg( const AnsiString Msg, TMsgDlgType
DlgType, TMsgDlgButtons Buttons, int HelpCtx);
```

DlgType задає характер повідомлення

mtWarning – зауваження , mtError – помилка , mtInformation – інформаційне повідомлення , mtConfirmation – запит підтвердження , mtCustom – без рисунка;

Buttons задає кількість і вигляд кнопок. Можливі значення цього параметра: ***mbYesNoCancel***, ***mbYesNoAllCancel***, ***mbOKCancel***, ***mbAbortRetryIgnore***, ***mbAbortIgnore***.

HelpCtx визначає екран контекстної довідки, що буде з'являтися за натискання клавіші . Якщо така довідка не планується, слід задати значення 0.

Значення параметра Buttons

mbOK
mbCancel
mbYes
mbNo
mbAbort
mbRetry
mbIgnore
mbAll
mbNoToAll
mbYesToAll

Відповідне значення, яке повертається

mrOk
mrCancel
mrYes
mrNo
mrAbort
mrRetry
mrIgnore
mrAll
mrNoToAll
mrYesToAll

Діалогове повідомлення, створене у програмі командою

```
int btn = MessageDlg("Ви дійсно хочете вилучити всю  
інформацію з файла?", mtConfirmation, mbOKCancel, 0);
```

Рисунок 6.18


```

// Глобальні оголошення
struct avio_struct
{ char num[5], from[20], to[20], all_time[6];
  TDateTime start_time, last_time;
  AnsiString info( )
{ return AnsiString(num) + " " + (AnsiString)from + " " +
  (AnsiString)to + " " + start_time.TimeString( ) + " " +
  last_time.TimeString( )+" "+(AnsiString)all_time;
} };
int size = sizeof(avio_struct); // Розмір структури
avio_struct w; // Змінна типу структури
int f; // Дескриптор
char s[]="a.dat"; // Ім'я файла
__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner) // Конструктор форми
{ if (!FileExists(s)) // Якщо такого файла не існувало,
  f=FileCreate(s); // він створюється.
  FileClose(f);
  SG1->Cells[0][0]="Номер рейсу";
  SG1->Cells[1][0]="Пункт відправлення";
  SG1->Cells[2][0]="Пункт прибуття";
  SG1->Cells[3][0]="Час відправлення";
  SG1->Cells[4][0]="Час прибуття";
  SG1->Cells[5][0]="Тривалість";
}

```

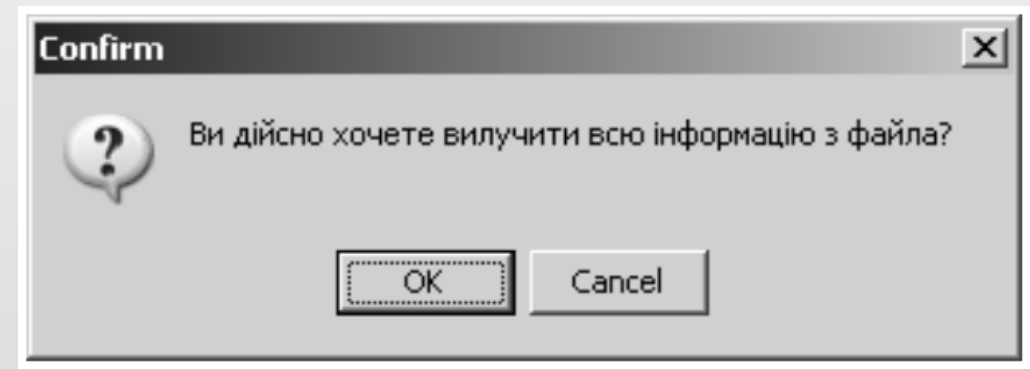


Рисунок 6.19


```

// Кнопка "Записати до файла"
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    AnsiString s1,s2;
    strcpy(w.num,Edit1->Text.c_str( ));
    strcpy(w.from,Edit2->Text.c_str( ));
    strcpy(w.to,Edit3->Text.c_str( ));
    w.start_time=Edit4->Text;
    strcpy(w.all_time,Edit5->Text.c_str( ));
    s1=(AnsiString)w.all_time;
    int n=StrToInt(s1.SubString(1,s1.Pos(':')-1));
    if (n>=24) n=n%24;
    // Створення шаблону ЧЧ:ММ.
    s2=IntToStr(n)+":"+s1.SubString(s1.Pos(':')+1,2);
    w.last_time=w.start_time+s2;
    // Відкриття файла для зчитування-записування.
    f=fopen(s,fmOpenReadWrite);
    if(f != -1) // Перевірка коректності відкриття файла.
    {
        FileSeek(f,0,2); // Позиціонування на кінець файла
        FileWrite(f,&w,size); // і записування даних.
        FileClose(f);
    }
    else //Якщо сталася помилка при відкритті, виведеться відповідне
    {
        ShowMessage("Помилка доступу до файла"); // повідомлення.
    }
    return;
}

Edit1->Clear( ); Edit2->Clear( ); Edit3->Clear( );
Edit4->Clear( ); Edit5->Clear( ); Edit1->SetFocus( );
}

```

```

// Переглядання інформації про всі авіарейси в Мемо
void __fastcall TForm1::Button2Click(TObject *Sender)
{ Memo1->Clear();
  f=FileOpen(s, fmOpenRead);
  while(FileRead(f, &w, size)) Memo1->Lines->Add(w.info( ));
  FileClose(f);
}

// Переглядання інформації про всі авіарейси в StringGrid
void __fastcall TForm1::Button3Click(TObject *Sender)
{ SG1->RowCount=2;
  AnsiString st;
  bool d=true;
  f=FileOpen(s, fmOpenRead);
  if (f==-1)
  { ShowMessage("Файл не відкрито");
    return;
  }
  while (FileRead(f, &w, size))
  { st = w.info( );
    if (d)
    { SG1->Rows[SG1->Row]->DelimitedText=st;
      d = false;
    }
    else
    { SG1->RowCount++;
      SG1->Row=SG1->RowCount-1;
      SG1->Rows[SG1->Row]->DelimitedText=st;
    } }
    FileClose(f);
  }
}

```

```

// Кнопка "Сортувати за часом відправлення"
void __fastcall TForm1::Button4Click(TObject *Sender)
{ f=FileOpen(s,fmOpenReadWrite);
  int fkol=FileSeek(f,0,2); // Визначення розміру файла в
байтах
  int kol=fkol/size; // Обчислення кількості записів у файлі
  avio_struct temp; // Допоміжна змінна типу структури для
сортування
  // Виділення динамічної пам'яті під масив структур
  avio_struct *av=new avio_struct[kol];
  FileSeek(f,0,0); // Позиціонування на початок файла
  while (FileRead(f,&w,size))
  *(av+i) = w; // Збереження запису як елемента масиву
  FileClose(f);
  for(int i=0;i<kol-1;i++)
  for(int j=i+1;j<kol;j++)
  if(av[j].start_time<av[i].start_time) // Сортування даних
  { temp=av[i];
    av[i]=av[j];
    av[j]=temp; }
  f=FileOpen(s,fmOpenReadWrite); // Відкриття файла для
записування
  for (int i=0;i<kol;i++)
  { FileWrite(f,&av[i],size); // Записування даних до
обнуленого файла
  delete av; // Звільнення динамічного масиву
  }
  FileClose(f);
  Button2Click(NULL); // Переглядання даних після видалення в
Мемо
  Button3Click(NULL); // і в StringGrid.
}

```

```

// Вилучення нічних рейсів за допомогою динамічного масиву.
void __fastcall TForm1::Button5Click(TObject *Sender)
{ TDateTime d1=TTime("00:00"), d2=TTime("05:00");
  Memo1->Clear( );
  f=FileOpen(s,fmOpenReadWrite);
  int fkol=FileSeek(f,0,2); // Розмір файла в байтах
  int kol=fkol/size; // Кількість записів у файлі
  // Виділення динамічної пам'яті під масив структур
відповідного розміру
  avio_struct *av = new avio_struct[kol];
  int kolzap=0;
  FileSeek(f,0,0);
  while(FileRead(f,&w,size))
  { if(w.start_time>=d1 && w.start_time<=d2)
    continue;
    *(av+kolzap) = w; // Збереження запису в елементі масиву
    kolzap++;
  }
  FileClose(f); DeleteFile(s); // Закриття і видалення файла
  f=FileCreate(s); // і створення заново.
  FileClose(f);
  f=FileOpen(s,fmOpenWrite); // Відкриття файла для записуван
  for(int i=0;i<kolzap;i++)
  FileWrite(f,&av[i],size); // Записування даних до файла
  FileClose(f);
  for(int i=0;i<kolzap;i++)
  delete av; // Звільнення динамічної пам'яті від масиву.
  Button2Click(NULL); // Переглядання даних після видалення в
Мемо
  Button3Click(NULL); // і в StringGrid.
}

```

```

// Вилучення нічних рейсів за допомогою допоміжного
файла
void __fastcall TForm1::Button6Click(TObject *Sender)
{ int tmp=FileCreate("tmp.dat"); // Створення
тимчасового файла
FileClose(tmp);
f=FileOpen(s, fmOpenRead); // Відкриття початкового
файла для зчитування,
tmp=FileOpen("tmp.dat", fmOpenWrite); // а тимчасового –
для записування.
FileSeek(f, 0, 0);
while (FileRead(f, &w, size))
{ if (!(w.start_time >= TTime("00:00") &&
w.start_time <= TTime("05:00")))
FileWrite(tmp, &w, size); // Записування даних до
тимчасового файла
}
FileClose(f); FileClose(tmp); // Закриття обох файлів,
DeleteFile(s); // видалення початкового файла
RenameFile("tmp.dat", s); // і перейменування
тимчасового файла.
Button2Click(NULL); // Переглядання даних після
видалення в Мемо
Button3Click(NULL); // і в StringGrid.
}

```

```

// Кнопка “Обнулити файл”
void __fastcall
TForm1::Button7Click(TObject
*Sender)
{ int btn=MessageDlg("Ви дійсно
хочете видалити усю інформацію з
файла?", mtConfirmation,
mbOKCancel, 0);
if (btn==1)
{ f=FileCreate(s); // Обнулення
файла
FileClose(f); // і закриття його.
ShowMessage("Файл пустий!");
}
}
// Кнопка “Вихід”
void __fastcall
TForm1::Button8Click(TObject
*Sender)
{ Close();
}

```

Тема №7. ОСНОВИ РОБОТИ З ДИНАМІЧНИМИ СТРУКТУРАМИ ДАНИХ

Тематичний план

- 7.1 Загальне поняття списку*
- 7.2 Визначення стеку та черги в динамічних структурах*
- 7.3 Бінарні дерева*
- 7.4 Загальне поняття автоматів та робота з ними*

7.1 ЗАГАЛЬНЕ ПОНЯТТЯ СПИСКУ

Динамічна структура даних
— це набір однотипних елементів, розміщуваних у пам'яті динамічно, тобто у перебігу виконання програми за допомогою операції *new()* чи то функції *malloc()*.

*Динамічна
структура даних*

*Динамічний
масив*

Списки

Стеки

Черги

Бінарні дерева

Рисунок 7.1

Списки використовуються замість масивів для зберігання й опрацювання однотипних даних, кількість яких заздалегідь є невідома й може змінюватися у перебігу роботи.

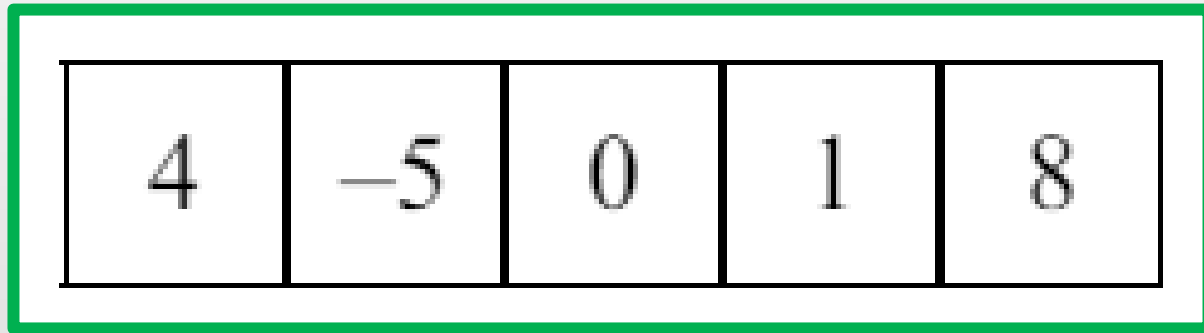


Рисунок 7.2 Одновимірний масив з 5-ти елементів

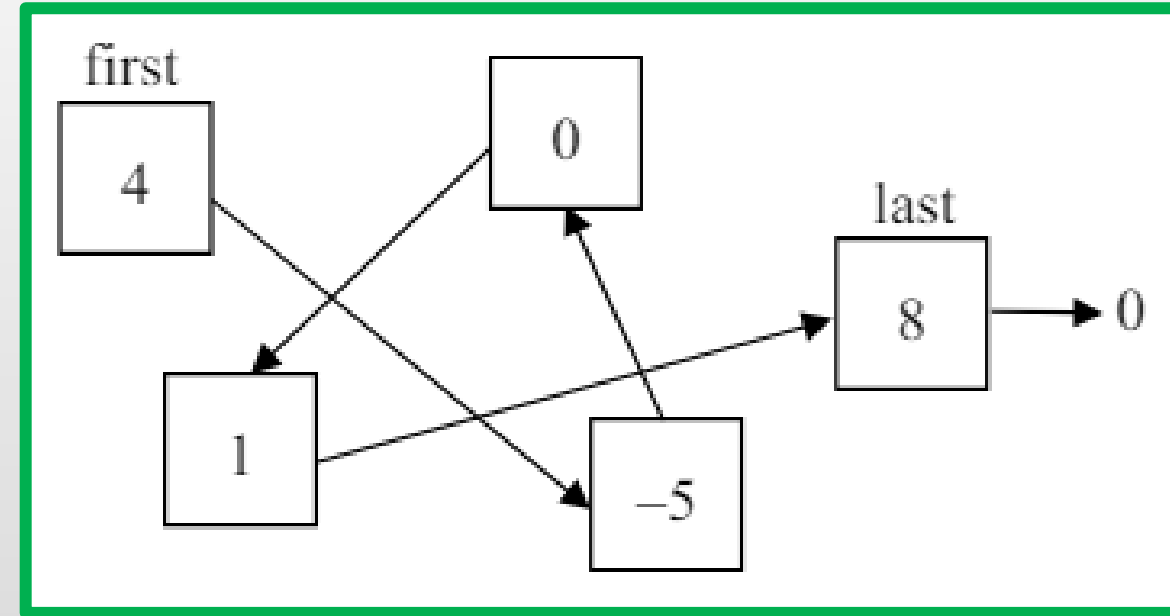


Рисунок 7.3 Елементи списку

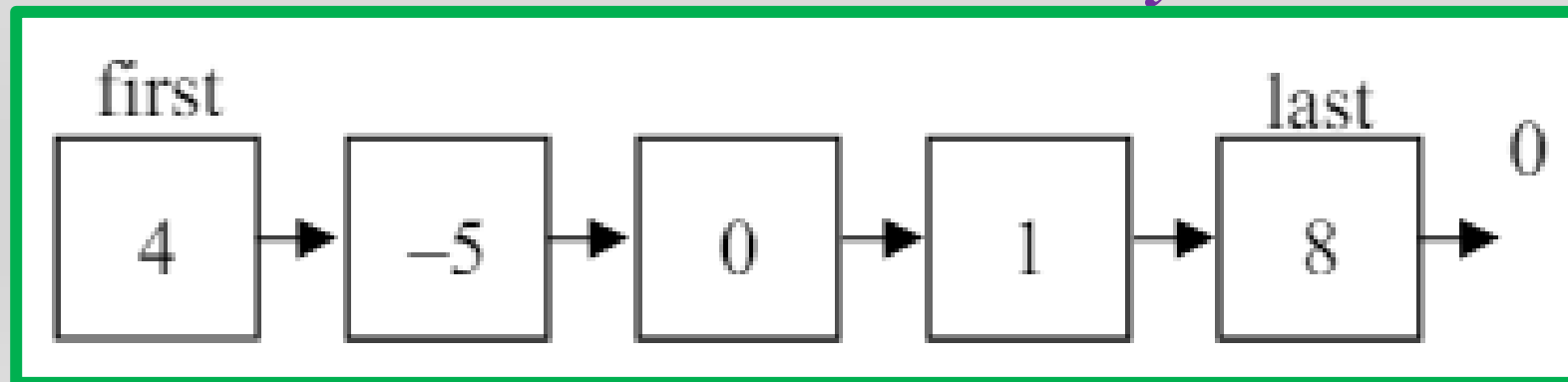


Рисунок 7.4 Список

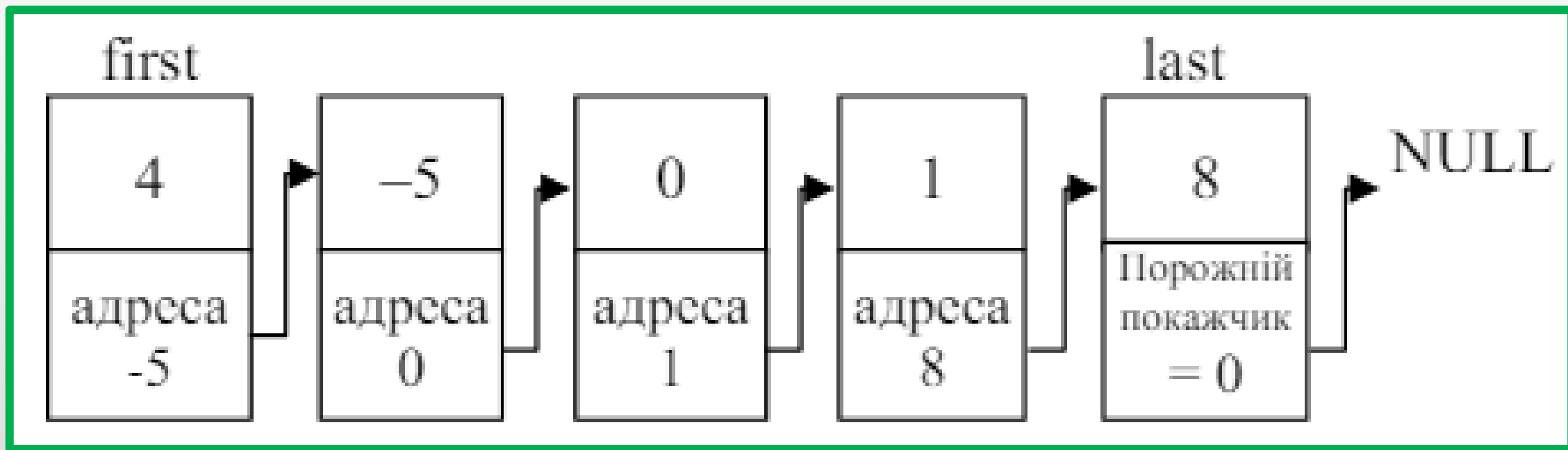


Рисунок 7.5 Список

У верхній частині кожного елемента зображено його поле даних (число), а в нижній – поле next (адреса наступного елемента в пам'яті)

Оголошення типу “елемент списку”

```
struct <елемент списку>
{ <оголошення полів даних>
  <елемент списку>* <показчик на наступний елемент>;
};
```

```
struct Element
{ int d;           // ціле число
  Element *next;   // показчик на наступний елемент
};
```

```
Element *first=0;
```

Знаючи покажчик на перший елемент, до другого елемента можна звернутися в такий спосіб: *first->next*, а до третього: *first->next->next* і т. д. Числове значення першого елемента: *first->d*.

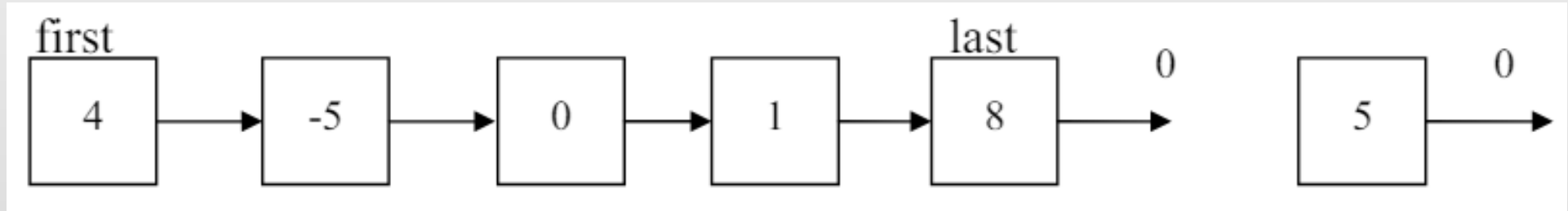
Для початкового створення списку слід створити перший елемент:

- виділити під нього місце у пам'яті: *first=new Element;*
- занести в нього числове значення: *first->d=4;*
- обнулити покажчик на наступний елемент: *first->next=NULL;*
- покажчику на останній елемент last присвоїти адресу створеного тільки що першого елемента, оскільки інших елементів поки немає: *last=first;*

Для долучення нового елемента c в кінець списку слід виконати такі дії:

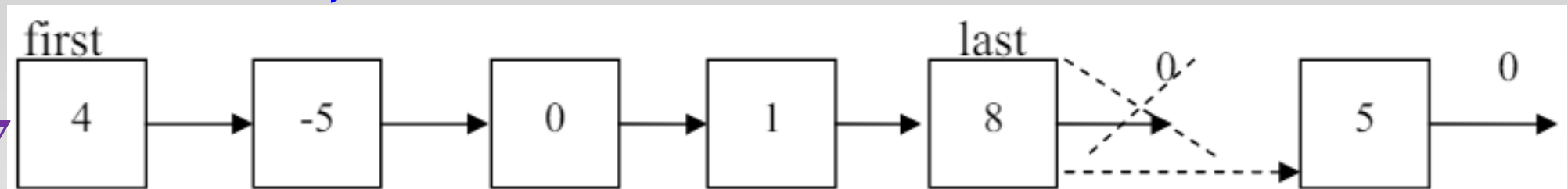
- виділити місце в пам'яті під новий елемент: $c = \text{new Element};$
- занести до нього поля даних (приміром ціле значення 5): $c \rightarrow d = 5;$
- покажчик на наступний елемент є пустий, а, оскільки наступних елементів покищо немає, йому слід присвоїти значення NULL (0): $c \rightarrow \text{next} = \text{NULL};$

Рисунок 7.6



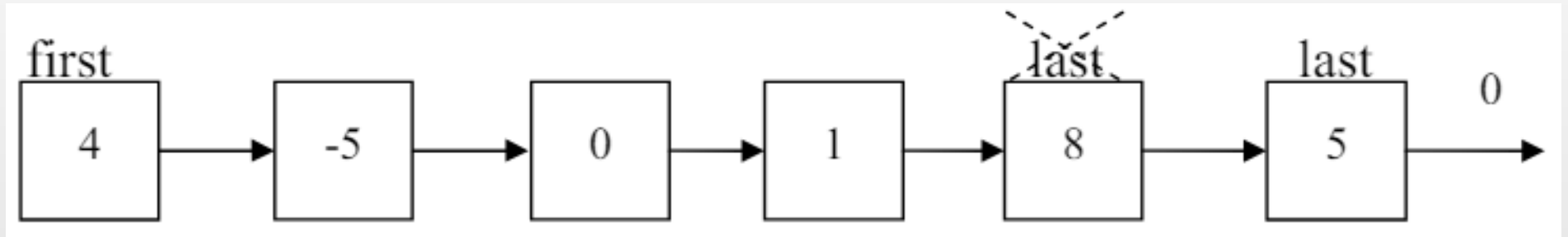
- створений елемент слід долучити до списку після останнього елемента, на який указує last. Тобто елемент c повинен стати наступним після last. Для цього адресу нового елемента слід записати у поле next останнього елемента: $\text{last} \rightarrow \text{next} = c;$

Рисунок 7.7



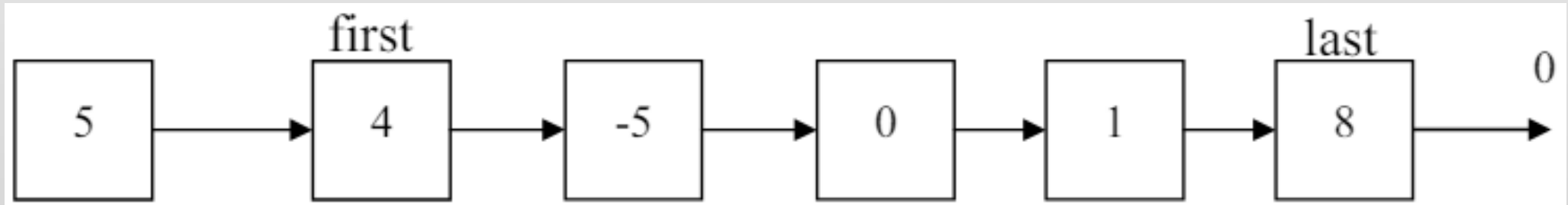
- присвоїти покажчикові на останній елемент last покажчик на новий елемент c: ***last=c;***

Рисунок 7.8



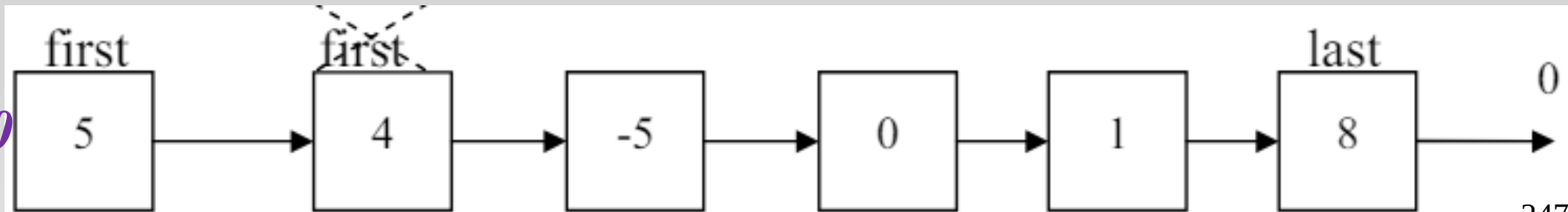
- ❖ новий елемент долучити до списку, тобто в його поле next записати адресу першого елемента: ***c->next=first;***

Рисунок 7.9



- ❖ покажчикові first присвоїти покажчик на новий елемент: ***first=c;***

Рисунок 7.10



Пройдення за списком виконується переважно у циклі **while**. Допоки не дісталися потрібного елемента чи то допоки список не завершено, переходимо до наступного елемента за його адресою.

```
Element *c;  
c=first;  
while(c!=0) // Допоки список не завершено,  
{ ... // виконання дії з c->d  
c=c->nex; // й переходження до наступного елемента списку.  
}
```

За вилучення першого елемента списку його адреса запам'ятовується в допоміжному покажчику **c**, а покажчик **first** переміщується на другий елемент.

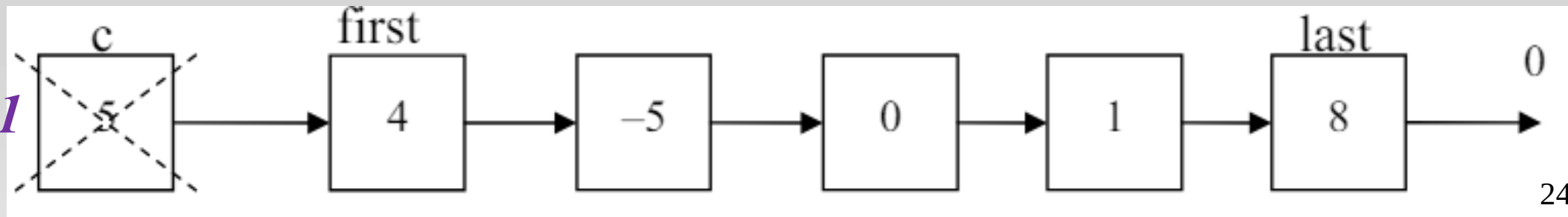


Рисунок 7.11

7.2 ВИЗНАЧЕННЯ СТЕКУ ТА ЧЕРГИ В ДИНАМІЧНИХ СТРУКТУРАХ

Стеки, черги - представляють собою списки, для яких дозволені тільки операції вставки та видалення першого та/або останнього елементу.

Стек - це впорядкований набір елементів, у якому додавання нових та видалення існуючих елементів допустимо тільки з одного кінця, який називається **вершиною стеку**.

LIFO (*Last In - First Out*)

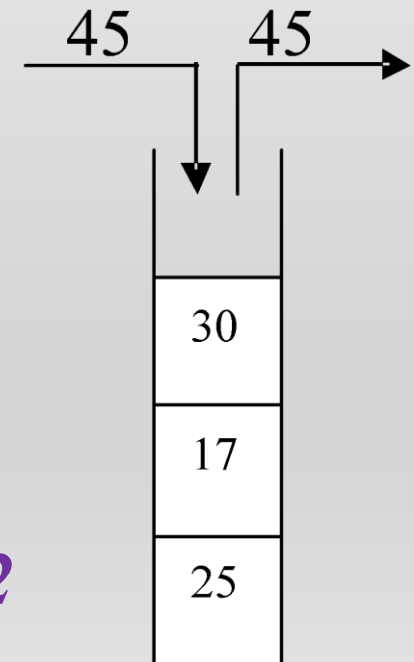


Рисунок 7.12

```
struct Node
{
    int      data; \\ область даних
    Node     *next, *prev;\\ посилання на сусідні вузли
};
typedef Node *PNode; \\ тип «вказівник на вузол»
```

```
struct Stack
{
    PNode Head, Tail;
}
```

Додавання елемента на вершину стеку

```
void Push ( Stack &S, int i )
{
    PNode NewNode;
    NewNode = new Node;
    NewNode->data = i; \\ створити новий вузол та заповнити даними
    NewNode->next = S.Head;
    NewNode->prev = NULL;
    if ( S.Head )
        S.Head->prev = NewNode;
    S.Head = NewNode; \\ додати новий вузол у початок списку
    if (! S.Tail )
        S.Tail = S.Head;
}
```


Отримання верхнього елемента з вершини стеку

```
int Pop ( Stack &S )
{
    PNode TopNode = S.Head;
    int i;
    if ( !TopNode ) return 0;\\ якщо стек пустий, повернути нуль
    i = TopNode->data;
    S.Head = TopNode->next;
    if ( S.Head )
        S.Head->prev = NULL;\\ переставити посилання
    else
        S.Tail = NULL;
    delete TopNode; \\ звільнити пам'ять
    return i;
}
```

Приклад 7.1

Задача: вводится символьний рядок, у який записано вираз з дужками трьох типів: [], {} і (). Визначити, чи правильно розставлені дужки (не звертаючи уваги на інші символи). Приклади:

[()]{ } [[({})] }

Спрощена задача: те ж саме, але з одним видом дужок.

Рішення: лічильник вкладеності дужок. Послідовність правильна, якщо в кінці лічильник дорівнює нулю і при прохіді жодного разу не ставав негативним.

(	(	)	)	(	)
1	2	1	0	1	0

(	(	)	)	)	(
1	2	1	0	-1	0

(	(	)	)	(
1	2	1	0	1



Чи можна вирішити вихідну задачу так само, але з трьома лічильниками?

	[	(	{	)	]	}
(:	0		1		0	
[:	0	1				0
{ :	0			1		0

Рішення задачі з дужками

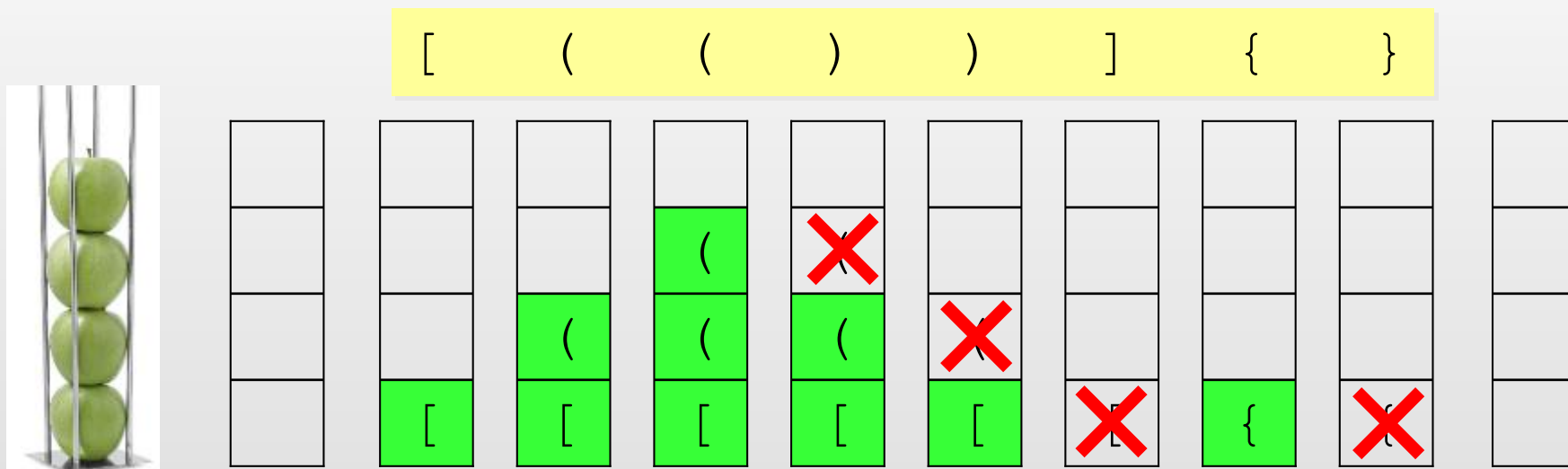


Рисунок 7.13

Алгоритм:

- на початку стек пустий;
- у циклі переглядаємо всі символи рядка по порядку;
- якщо черговий символ – дужка, яка відкривається, заносимо її на вершину стеку;
- якщо символ – дужка, яка закривається, перевіряємо вершину стеку: там повинна бути відповідна дужка, яка відкривається (якщо це не так, то помилка);

Реалізація стеку (масив)

Структура-стек:

```
const MAXSIZE = 100;
struct Stack {
    char data[MAXSIZE]; // стек на 100 символів
    int  size;           // число елементів
};
```

Додавання елементу:

```
int Push ( Stack &S, char x )
{
    if ( S.size == MAXSIZE ) return 0;
    S.data[S.size] = x;
    S.size ++;
    return 1;
}
```

ПОМИЛКА:
переповнення
стеку

додати елемент

немає помилки

Реалізація стеку (масив)

Зняття елементу з вершини:

```
char Pop ( Stack &S )  
{  
    if ( S.size == 0 ) return char(255);  
    S.size --;  
    return S.data[S.size];  
}
```

ПОМИЛКА:
стек пустий

Порожньо чи ні?

```
int isEmpty ( Stack &S )  
{  
    if ( S.size == 0 )  
        return 1;  
    else return 0;  
}
```

```
int isEmpty ( Stack &S )  
{  
    return (S.size == 0);  
}
```

Програма

```
void main()
{
    char br1[3] = { '(', '[', '{' };
    char br2[3] = { ')', ']', '}' };
    char s[80], upper;
    int i, k, error = 0;
    Stack S;
    S.size = 0;
    printf("Введіть вираз з дужками > ");
    gets ( s );
    ... // тут буде основний цикл обробки
    if ( ! error && (S.size == 0) )
        printf("\nВираз правильний\n");
    else printf("\nВираз неправильний\n");
}
```

дужки, які
відкриваються

дужки, які
закриваються

те, що зняли зі стеку

признак помилки

Обробка рядка (основний цикл)

```
for ( i = 0; i < strlen(s); i++ )  
{  
    for ( k = 0; k < 3; k++ )  
    {  
        if ( s[i] == br1[k] )    // якщо дужка, яка відкривається  
        {  
            Push ( S, s[i] ); // заштовхнути в стек  
            break;  
        }  
        if ( s[i] == br2[k] ) // якщо дужка, яка закривається  
        {  
            upper = Pop ( S ); // зняти верхній елемент  
            if ( upper != br1[k] ) error = 1;  
            break;  
        }  
    }  
    if ( error ) break;  
}
```

цикл за всіма символами
рядка S

цикл за всіма видами дужок

помилка: стек пустий
або не та дужка

була помилка: далі немає сенсу
перевіряти

Реалізація стеку (список)

Структура вузла:

```
struct Node {  
    char data;  
    Node *next;  
};  
typedef Node *PNode;
```

Додавання елементу:

```
void Push (PNode &Head, char x)  
{  
    PNode NewNode = new Node;  
    NewNode->data = x;  
    NewNode->next = Head;  
    Head = NewNode;  
}
```


Реалізація стеку (список)

Зняття елементу з вершини:

```
char Pop (PNode &Head) {  
    char x;  
    PNode q = Head;  
    if ( Head == NULL ) return char(255);  
    x = Head->data;  
    Head = Head->next;  
    delete q;  
    return x;  
}
```

стек пустий

Зміни в основній програмі:

```
Stack S; PNode S = NULL;  
S.size = 0;  
...  
if ( ! error && (S.size == 0) )  
    printf("\nВираз правильний\n");  
else printf("\nВираз неправильний \n");
```

(S == NULL)

Обчислення арифметичних виразів

Як обчислювати автоматично:

$(a + b) / (c + d - 1)$

Інфіксний запис

(знак операції між операндами)



необхідні дужки!

Префіксний запис (знак операції до операндів)

$/ \ a \ + \ b \ c \ + \ d \ - \ 1$

польська нотація,
[Jan Łukasiewicz](#) (1920)



дужки не потрібні, можна однозначно
обчислити!

Постфіксний запис (знак операції після операндів)

$a \ + \ b$

$c \ + \ d \ - \ 1$

$/$

зворотна польська нотація,
[F. L. Bauer](#) and [E. W. Dijkstra](#)

Обчислення виразів

Постфіксна форма:

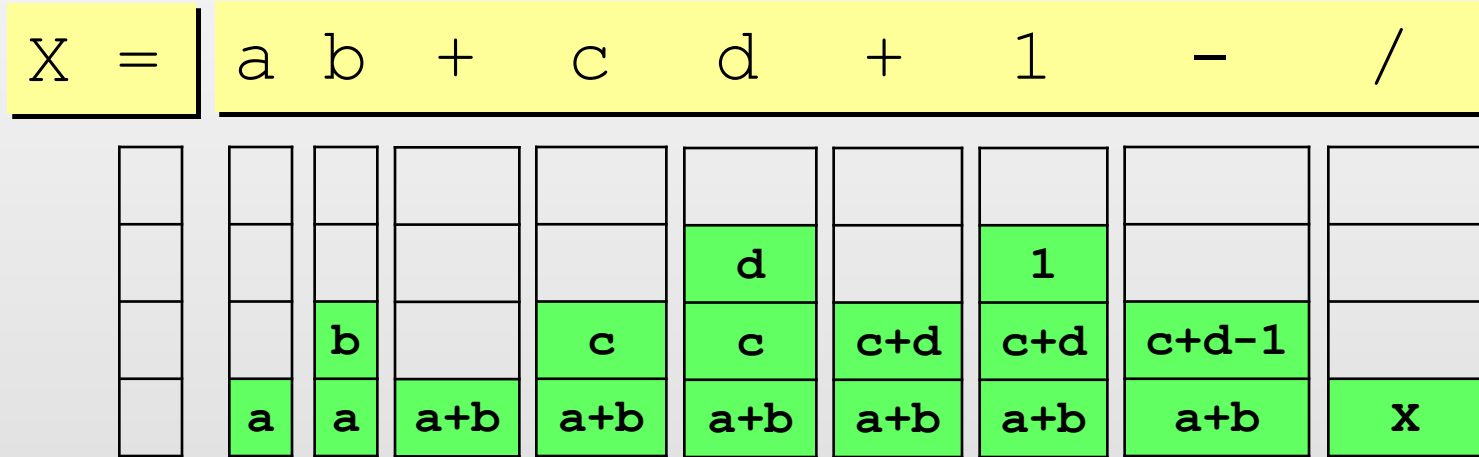


Рисунок 7.14

Алгоритм:

- взяти черговий елемент;
- якщо це не знак операції, додати його у стек;
- якщо це знак операції, то
 - взяти зі стеку два операнда;
 - виконати операцію та записати результат у стек;
- перейти до кроку 1.

Системний стек (*Windows – 1 Мб*)

Використовується для

- розміщення локальних змінних;
- зберігання адрес повернення (за якими переходить програма після виконання функції або процедури);
- передачі параметрів у функції та процедури;
- тимчасового зберігання даних (у програмах на мові *Ассемблер*).

Переповнення стеку (*stack overflow*):

- занадто багато локальних змінних (вихід – використовувати динамічні масиви);
- дуже багато рекурсивних викликів функцій та процедур (вихід – переробити алгоритм так, щоб зменшити глибину рекурсії або відмовитися від неї взагалі).

Черга

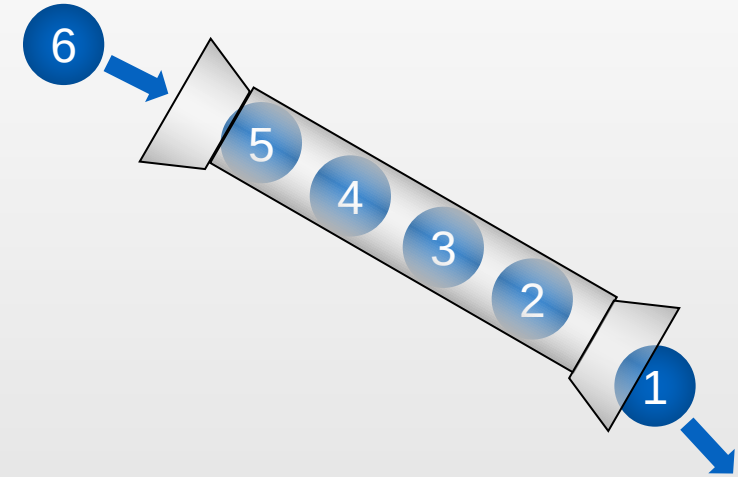


Рисунок 7.15

Черга – це лінійна структура даних, у якій додавання елементів можливо тільки з одного кінця (кінця черги), а видалення елементів – тільки з іншого кінця (початку черги).

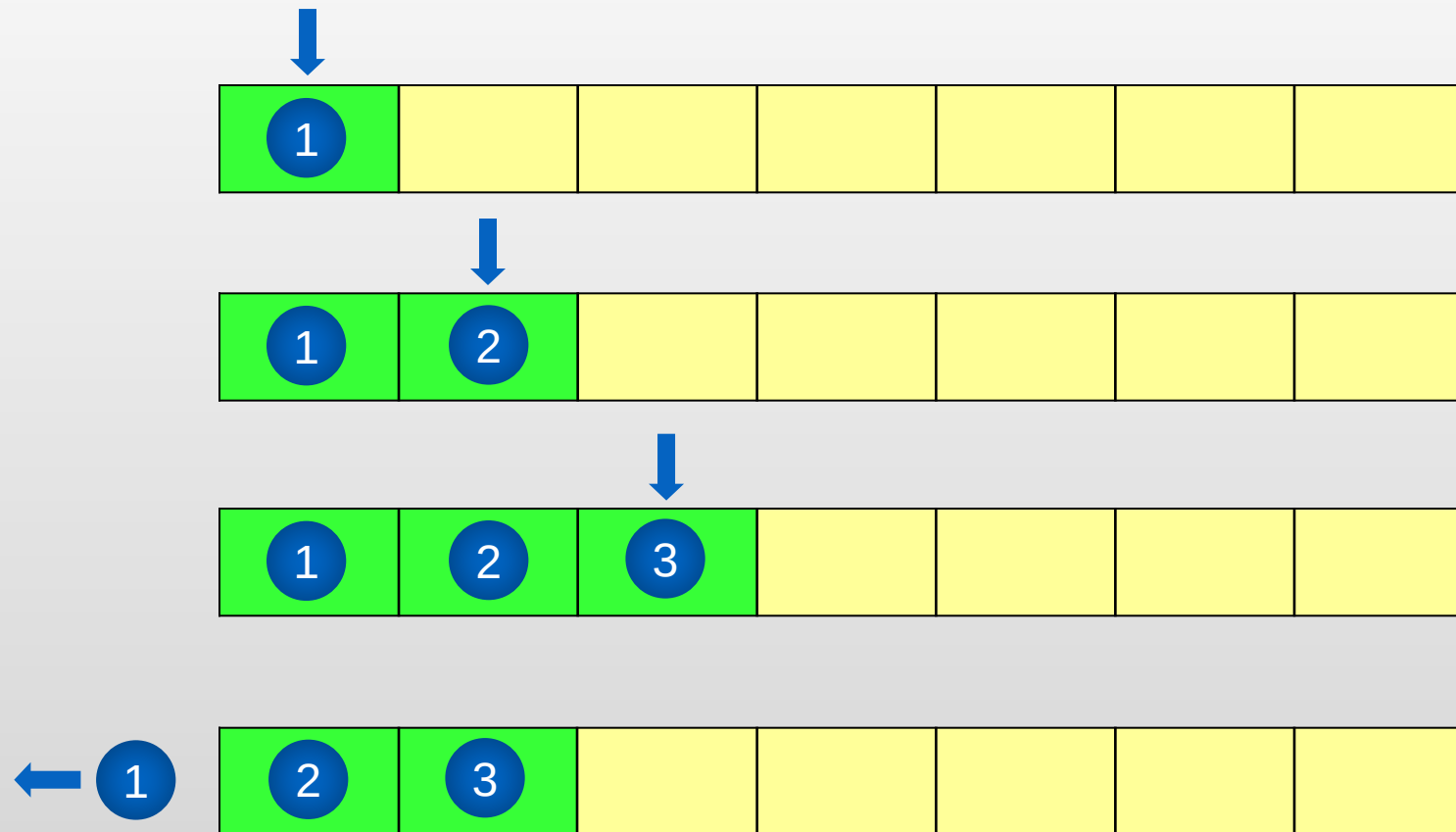
FIFO = First In – First Out

«Хто першим увійшов, той першим вийшов».

Операції з чергою:

- додати елемент у кінець черги (*PushTail* = заштовхнути у кінець);
- видалити елемент з початку черги (*Pop*).

Реалізація черги (масив)



самий простий спосіб



- 1) треба заздалегідь виділити масив;
- 2) при вибірці з черги треба здвигати всі елементи.

Рисунок 7.16

Реалізація черги (кільцевий масив)

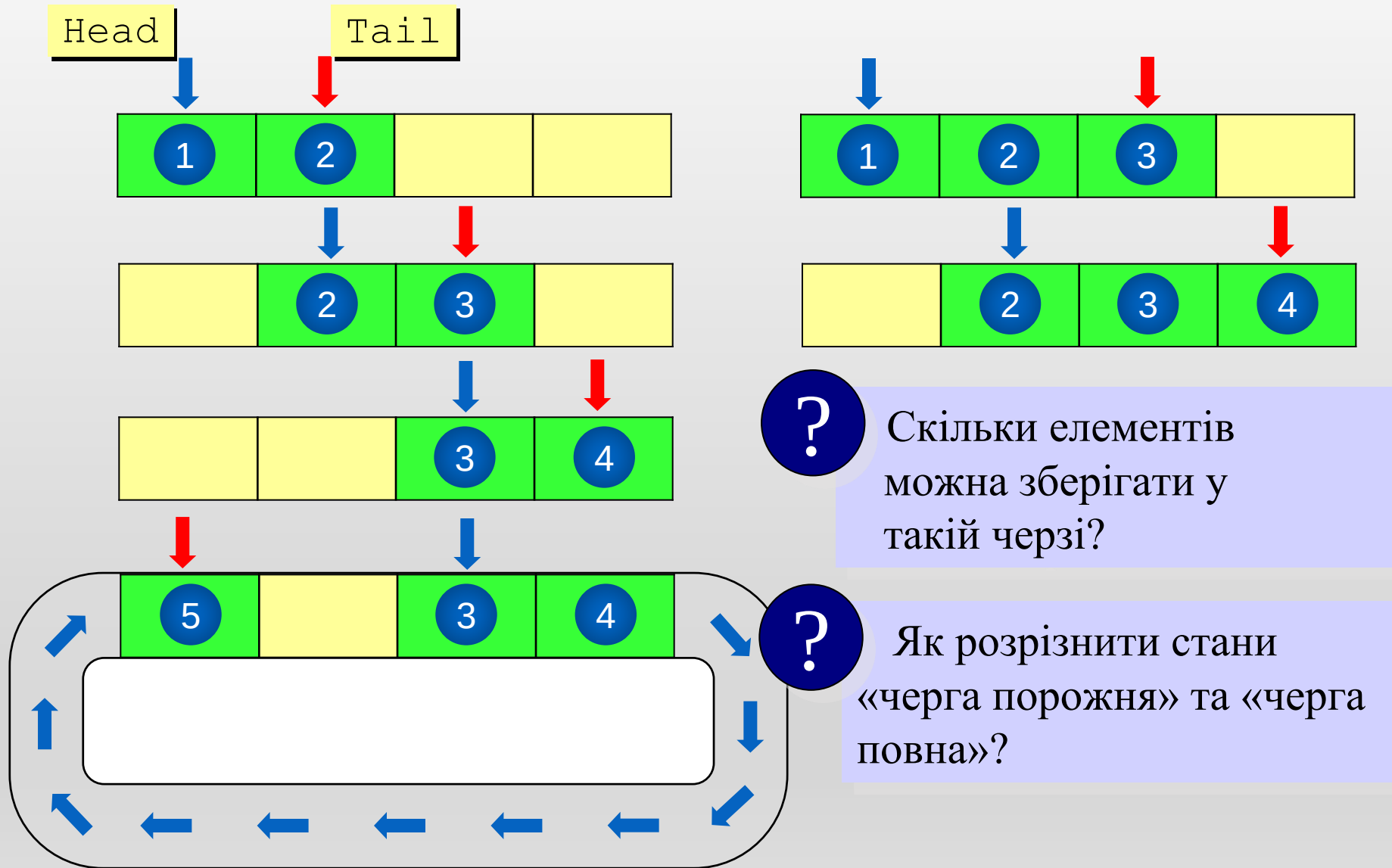
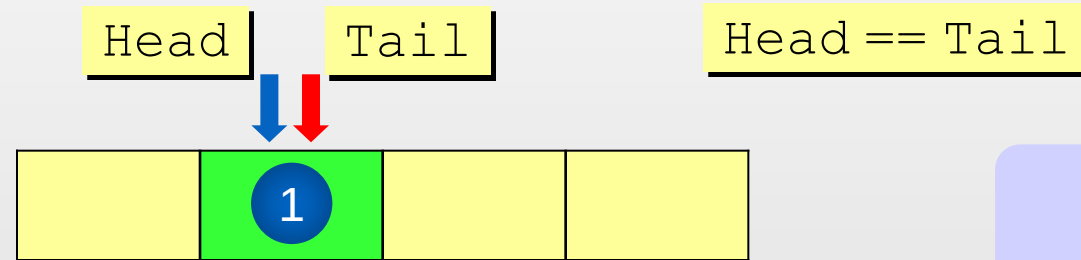


Рисунок 7.17

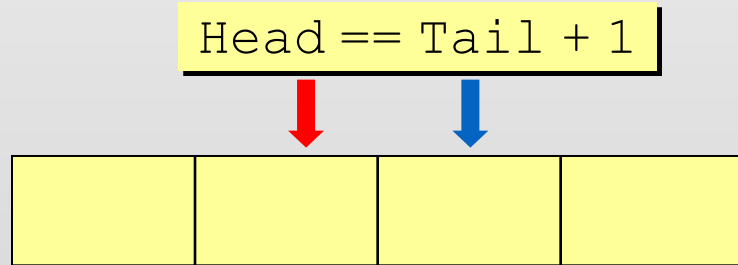
Реалізація черги (кільцевий масив)

У черзі 1 елемент:

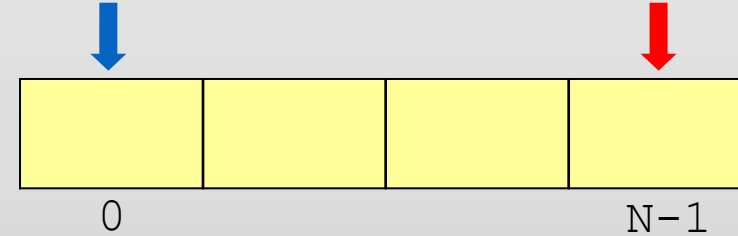


розмір
масиву

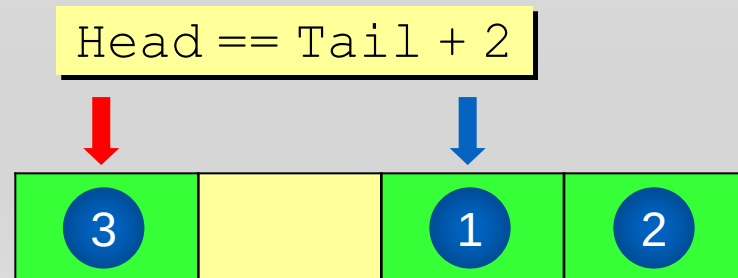
Черга порожня:



$$\text{Head} == (\text{Tail} + 1) \% N$$



Черга повна:



$$\text{Head} == (\text{Tail} + 2) \% N$$

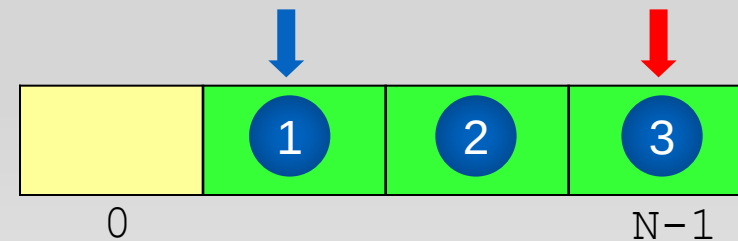


Рисунок 7.18

Реалізація черги (кільцевий масив)

Структура даних:

```
const MAXSIZE = 100;  
struct Queue {  
    int data[MAXSIZE];  
    int head, tail;  
};
```

Додавання у чергу:

```
int PushTail ( Queue &Q, int x )  
{  
    if ( Q.head == (Q.tail+2) % MAXSIZE )  
        return 0;  
    Q.tail = (Q.tail + 1) % MAXSIZE;  
    Q.data[Q.tail] = x;  
    return 1;  
}
```

замкнути у
кільце

черга повна,
не додати

вдало додали

Реалізація черги (кільцевий масив)

Вибірка з черги:

```
int Pop ( Queue &Q )  
{  
    int temp;  
    if ( Q.head == (Q.tail + 1) % MAXSIZE )  
        return 32767;  
    temp = Q.data[Q.head];  
    Q.head = (Q.head + 1) % MAXSIZE;  
    return temp;  
}
```

черга порожня

взяти перший
елемент

видалити його з
черги

Реалізація черги (списки)

Структура вузла:

```
struct Node {  
    int  data;  
    Node *next;  
};  
typedef Node *PNode;
```

Тип даних «черга»:

```
struct Queue {  
    PNode Head, Tail;  
};
```

Реалізація черги (списки)

Додавання елементу:

```
void PushTail ( Queue &Q, int x )  
{  
    PNode NewNode;  
    NewNode = new Node;  
    NewNode->data = x;  
    NewNode->next = NULL;  
    if ( Q.Tail )  
        Q.Tail->next = NewNode;  
    Q.Tail = NewNode;  
    if ( Q.Head == NULL )  
        Q.Head = Q.Tail;  
}
```

створюємо
новий вузол

якщо у списку вже щось
було, додаємо у кінець

якщо у списку нічого
не було, ...

Реалізація черги (списки)

Вибірка елементу:

```
int Pop ( Queue &Q )  
{  
    PNode top = Q.Head;  
    int x;  
    if ( top == NULL )  
        return 32767;  
    x = top->data;  
    Q.Head = top->next;  
    if ( Q.Head == NULL )  
        Q.Tail = NULL;  
    delete top;  
    return x;  
}
```

якщо список
порожній, ...

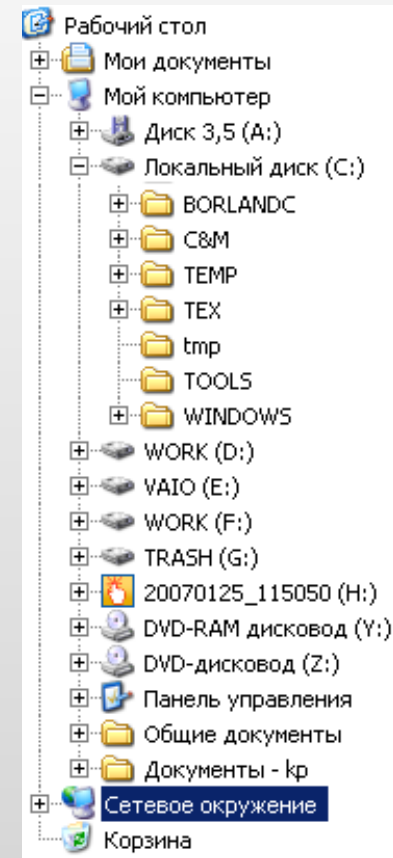
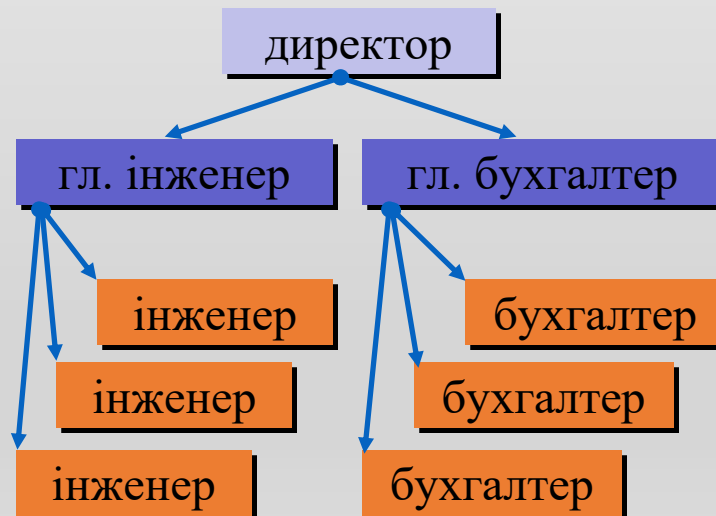
запам'ятали
перший елемент

якщо у списку нічого
не залишилось, ...

звільнити
пам'ять

7.3 БІНАРНІ ДЕРЕВА

Дерева



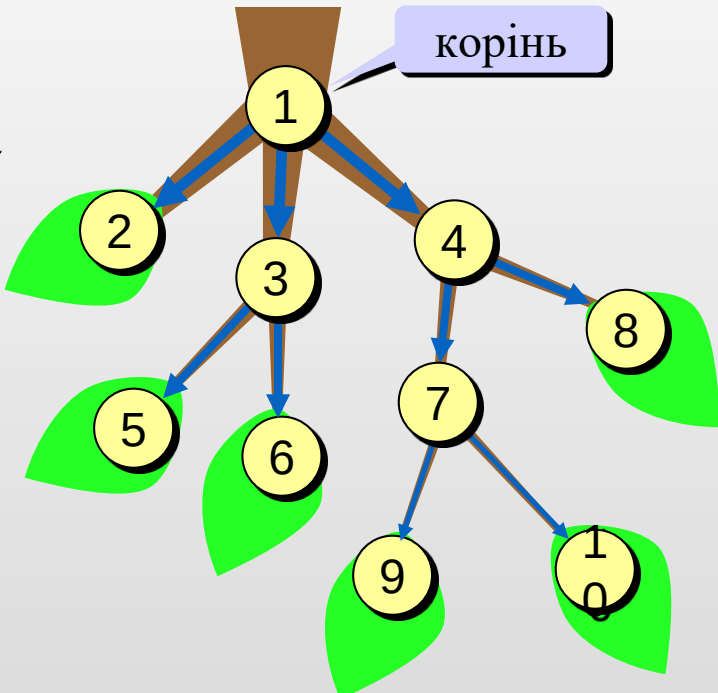
Що спільного в усіх прикладах?

Рисунок 7.19

Дерево — це структура даних, яка складається з вузлів та з'єднуючих їх напрямлених ребер (дуг), причому у кожен вузол (окрім кореневого) веде рівно одна дуга.

Корінь — це початковий вузол дерева.

Лист — це вузол, з якого не виходить жодної дуги.



Які структури – не дерева?

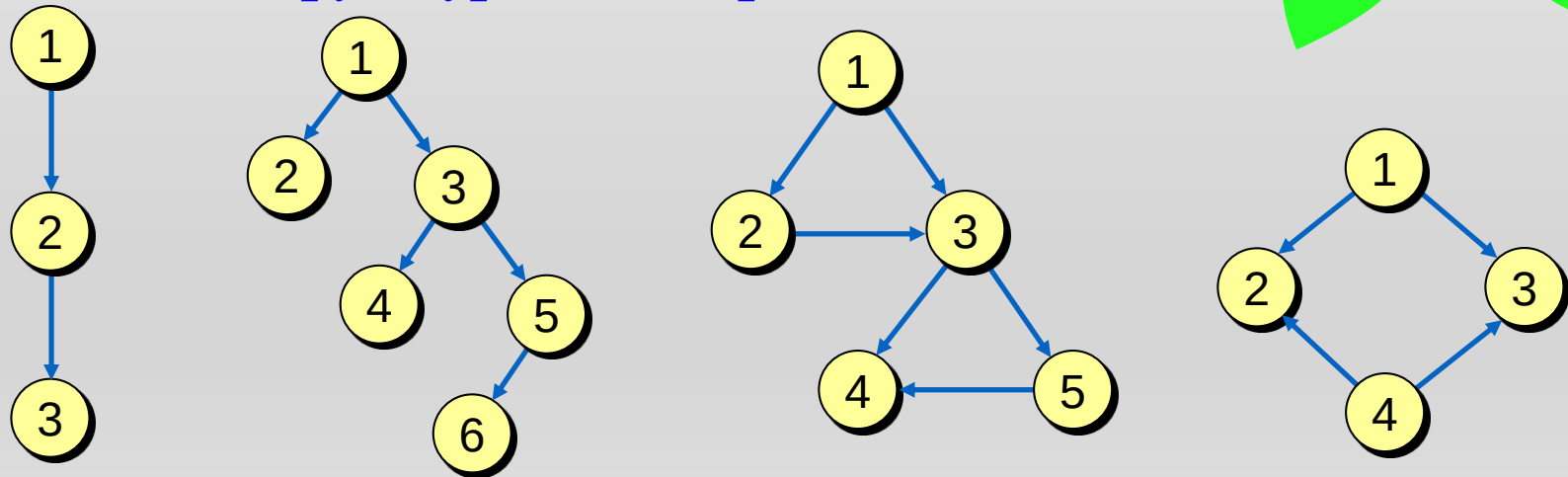


Рисунок 7.20



За допомогою дерев зображуються відносини підпорядкованості (ієрархія, «старший – молодший», «батько – дитина»).

Предок вузла x – це вузол, з якого існує шлях за стрілками у вузол x .

Нащадок вузла x – це вузол, у який існує шлях за стрілками з вузла x .

Батько вузла x – це вузол, з якого існує дуга безпосередньо у вузол x .

Син вузла x – це вузол, у який існує дуга безпосередньо з вузла x .

Брат вузла x (*sibling*) – це вузол, у якого той же батько, що й у вузла x .

Висота дерева – це найбільша відстань від кореня до листа (кількість дуг).

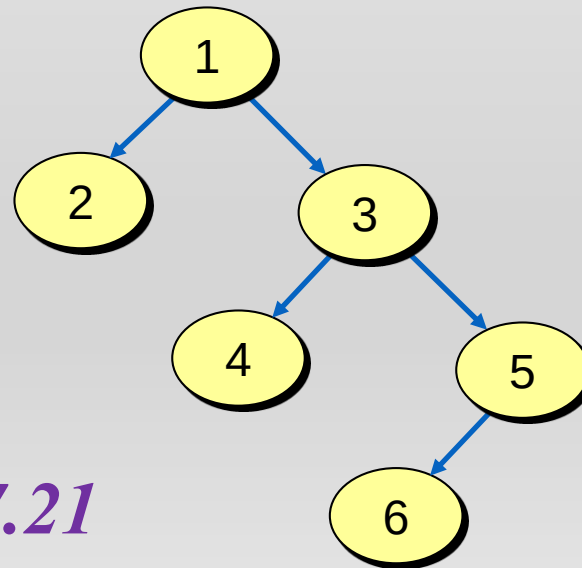


Рисунок 7.21

Дерево – рекурсивна структура даних

Рекурсивне визначення:

1. Порожня структура – це дерево.
2. Дерево – це корінь та декілька зв'язаних з ним дерев.

Двійкове (бінарне) дерево – це дерево, у якому кожен вузол має не більше двох синів.

1. Порожня структура – це двійкове дерево.
2. Двійкове дерево – це корінь та два зв'язаних з ним двійкових дерева (ліве та праве піддерева).

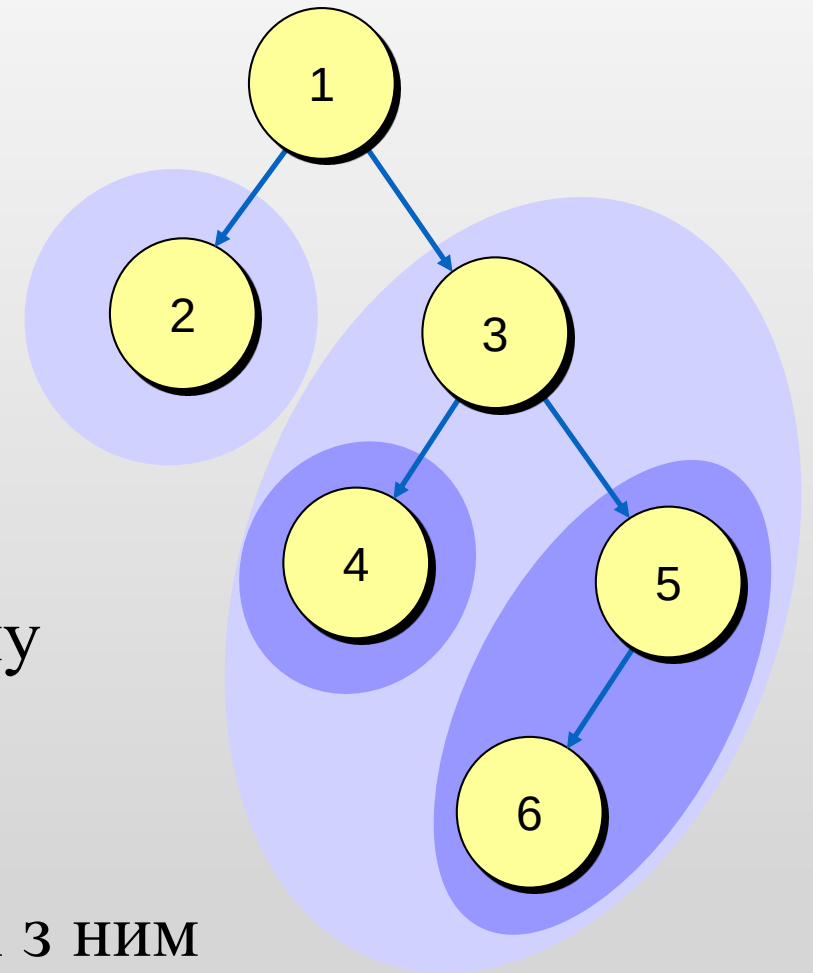


Рисунок 7.22

Двійкові дерева

Застосування:

- пошук даних у спеціально побудованих деревах (бази даних);
- сортування даних;
- обчислення арифметичних виразів;
- кодування (метод Хаффмана).

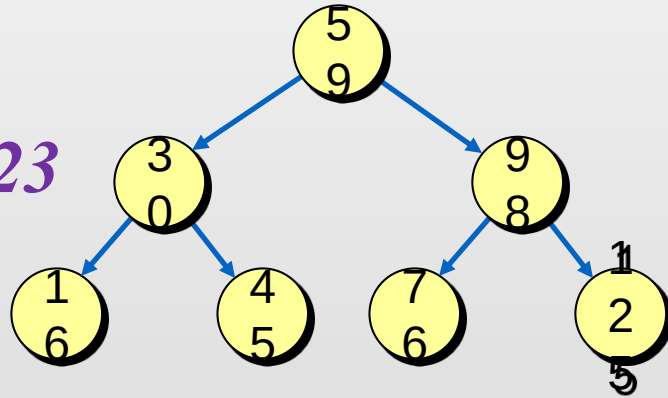
Структура вузла:

```
struct Node {  
    int data;           // корисні дані  
    Node *left, *right; // посилання на лівого  
                        // та правого синів  
};  
typedef Node *PNode;
```

Двійкові дерева пошуку

Ключ – це характеристика вузла, за якою виконується пошук (частіше за все – одне з полів структури).

Рисунок 7.23



Яка закономірність?

Зліва від кожного вузла знаходяться вузли з меншими ключами, а справа – з більшими.

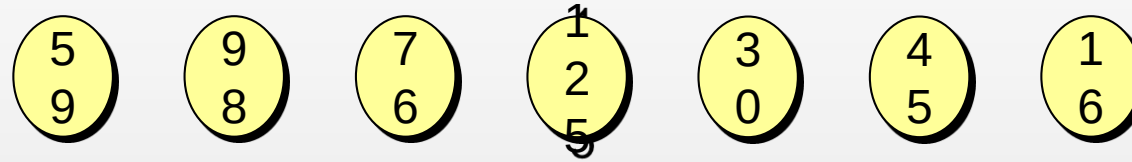
Як шукати ключ, який дорівнює x :

- якщо дерево порожнє, ключ не знайдено;
- якщо ключ вузла дорівнює x , то стоп.
- якщо ключ вузла менше x , то шукати x у правому піддереві;
- якщо ключ вузла більше x , то шукати x у лівому піддереві.



Зведення задачі до такої ж задачі меншої розмірності – це ...?

Пошук у масиві (N елементів):



При кожному порівнянні відкидається 1 елемент.

Число порівнянь – N .

Пошук по дереву (N елементів):

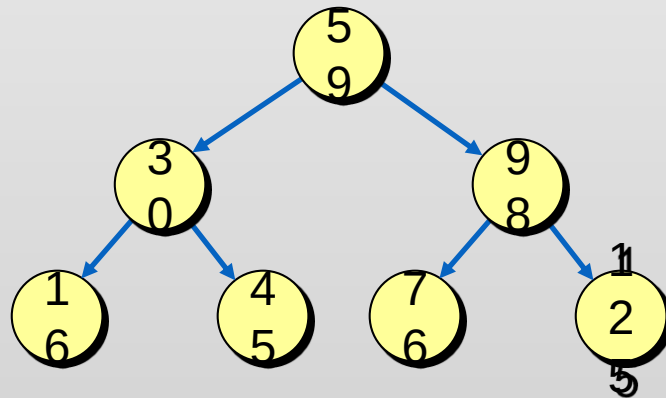


Рисунок 7.24

При кожному порівнянні відкидається половина елементів, які залишилися.

Число порівнянь $\sim \log_2 N$.



швидкий пошук



1) треба заздалегідь побудувати дерево;

2) бажано, щоб дерево було мінімальної висоти.

Реалізація алгоритму пошуку

```
//-----  
// Функція Search – пошук по дереву  
// Вхід: Tree – адреса кореня,  
//       x – що шукаємо  
// Вихід: адреса вузла або NULL (не знайшли)  
//-----  
PNode Search (PNode Tree, int x)  
{  
    if ( ! Tree ) return NULL;  
    if ( x == Tree->data )  
        return Tree;  
    if ( x < Tree->data )  
        return Search(Tree->left, x);  
    else  
        return Search(Tree->right, x);  
}
```

дерево порожнє:
ключ не знайшли...

знайшли,
повертаємо
адресу кореня

шукати у
лівому
піддереві

шукати у
правому
піддереві

Як побудувати дерево пошуку?

```
//-----  
// Функція AddToTree - додати елемент до дерева  
// Вхід: Tree - адреса кореня,  
//       x   - що додаємо  
//-----
```

```
void AddToTree (PNode &Tree, int x)
```

```
{
```

```
if ( ! Tree ) {
```

```
    Tree = new Node;
```

```
    Tree->data = x;
```

```
    Tree->left = NULL;
```

```
    Tree->right = NULL;
```

```
    return;
```

```
}
```

```
if ( x < Tree->data )
```

```
    AddToTree ( Tree->left, x );
```

```
else AddToTree ( Tree->right, x );
```

```
}
```

адреса кореня може
змінитися

дерево порожнє: створюємо
новий вузол (корінь)

додаємо до лівого або
правого піддерева



Мінімальна висота не гарантується!

Обхід дерева – це перерахування всіх вузлів у певному порядку.

Обхід ЛКП («лівий – корінь – правий»):



Обхід ПКЛ («правий – корінь – лівий»):



Обхід КЛП («корінь – лівий – правий»):



Обхід ЛПК («лівий – правий – корінь»):

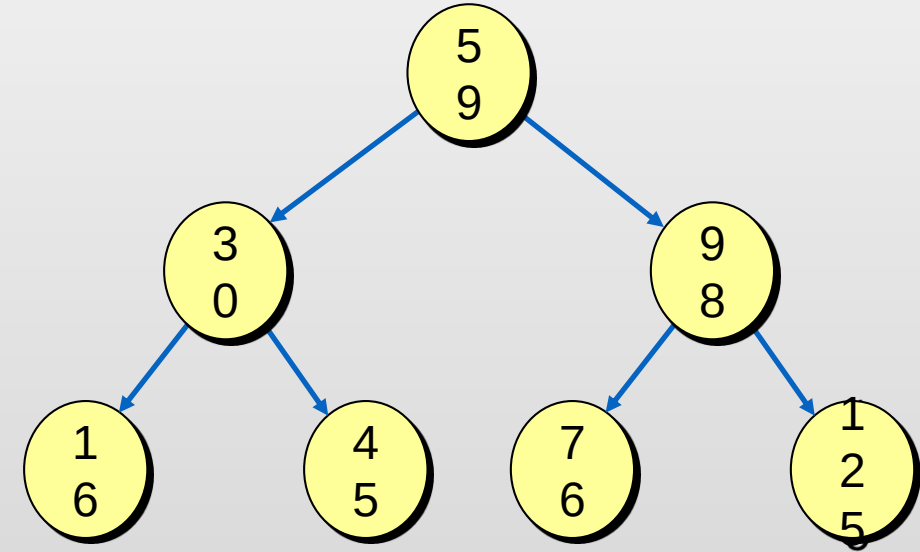


Рисунок 7.25

Обхід дерева – реалізація

```
//-----  
// Функція LKP – обхід дерева у порядку ЛКП  
//           (лівий – корінь – правий)  
// Вхід: Tree – адреса кореня  
//-----  
void LKP ( PNode Tree )  
{  
    if ( ! Tree ) return;  
    LKP ( Tree->left );  
    printf ( "%d ", Tree->data );  
    LKP ( Tree->right );  
}
```

обхід цієї гілки
завершено

обхід лівого піддерева

виведення даних кореня

обхід правого піддерева

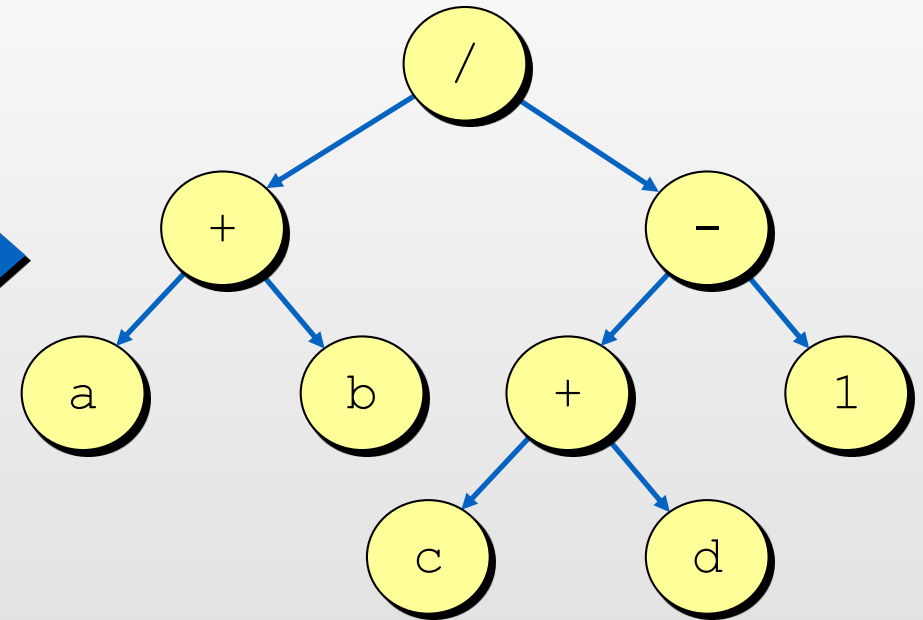
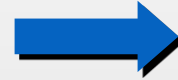


Для рекурсивної структури зручно
застосовувати рекурсивну обробку!

Розбір арифметичних виразів

Як обчислити автоматично:

$(a + b) / (c + d - 1)$



Інфіксий запис, обхід ЛКП

(знак операції між операндами)

$a + b / c + d - 1$



необхідні дужки!

Рисунок 7.25

Префіксий запис, КЛП (знак операції до операндів)

$/ + a b - + c d 1$

польська нотація,
[Jan Łukasiewicz](#) (1920)



дужки не потрібні, можна однозначно обчислити!

Постфіксий запис, ЛПК (знак операції після операндів)

$a b + c d + 1 - /$

зворотна польська нотація,
[F. L. Bauer](#) and [E. W. Dijkstra](#)

Обчислення виразів

Постфіксна форма:

X = a b + c d + 1 - /

					d		1		
		b		c	c	c+d	c+d	c+d-1	
	a	a	a+b	a+b	a+b	a+b	a+b	a+b	x

Алгоритм:

- взяти черговий елемент;
- якщо це не знак операції, додати його у стек;
- якщо це знак операції, то
 - взяти зі стеку два операнда;
 - виконати операцію та записати результат у стек;
- перейти до кроку 1.

Задача: у символному рядку записано правильний арифметичний вираз, який може містити тільки однозначні числа та знаки операцій $+-*\backslash$. Побудувати дерево синтаксичного розбору та форму запису виразу

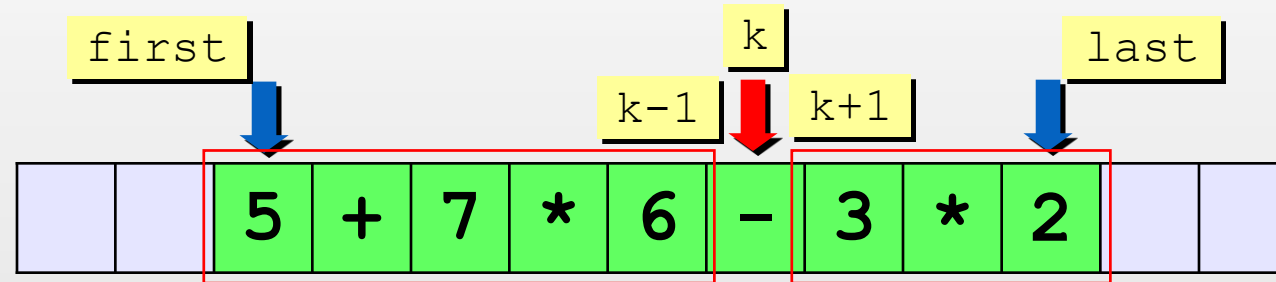
Алгоритм:

- ввести рядок;
- побудувати дерево;
- обчислити вираз за деревом.

Обмеження:

- помилки не обробляємо;
- багатозначні числа не дозволені;
- дробові числа не дозволені;
- дужки не дозволені.

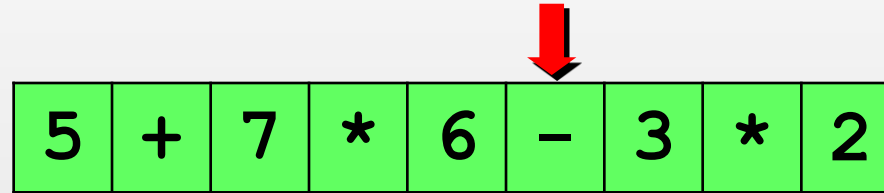
Побудова дерева



Алгоритм:

- якщо `first=last` (залишився один символ – число), то створити новий вузол та записати у нього цей елемент; інакше...
- серед елементів від `first` до `last` включно знайти останню операцію (елемент з номером `k`);
- створити новий вузол (корінь) та записати у нього знак операції;
- рекурсивно застосувати цей алгоритм два рази:
 - побудувати ліве піддерево, розібрав вираз із елементів масиву з номерами від `first` до `k-1`;
 - побудувати праве піддерево, розібрав вираз із елементів масиву з номерами від `k+1` до `last`.

Як знайти останню операцію?



Порядок виконання операцій

- множення та ділення;
- додавання та віднімання.

Пріоритет (старшинство) — число, яке визначає послідовність виконання операцій: раніше виконуються операції з більшим пріоритетом:

- множення та ділення (пріоритет 2);
- додавання та віднімання (пріоритет 1).



Треба шукати останню операцію з найменшим пріоритетом!

Пріоритет операції

```
//-----  
// Функція Priority - пріоритет операції  
// Вхід:  символ операції  
// Вихід: пріоритет або 100, якщо не операція  
//-----  
int Priority ( char c )  
{  
    switch ( c ) {  
        case '+': case '-':  
            return 1;  
        case '*': case '/':  
            return 2;  
    }  
    return 100;  
}
```

додавання та
віднімання:
пріоритет 1

множення та
ділення:
пріоритет 2

це взагалі не
операція

Номер останньої операції

```
//-----  
// Функція LastOperation - номер останньої операції  
// Вхід:  рядок, номери першого і останнього  
//        символів частини, яка розглядається  
// Вихід: номер символу - останньої операції  
//-----  
int LastOperation ( char Expr[], int first, int last )  
{  
    int MinPrt, i, k, prt;  
    MinPrt = 100;  
    for( i=first; i <= last; i++ ) {  
        prt = Priority ( Expr[i] );  
        if ( prt <= MinPrt ) {  
            MinPrt = prt;  
            k = i;  
        }  
    }  
    return k;  
}
```

перевіряємо всі
СИМВОЛИ

знайшли операцію з
мінімальним
пріоритетом

повернути номер
символу

Побудова дерева

Структура вузла

```
struct Node {  
    char data;  
    Node *left, *right;  
};  
typedef Node *PNode;
```

Створення вузла для числа (без нащадків)

```
PNode NumberNode ( char c )  
{  
    PNode Tree = new Node;  
    Tree->data = c;  
    Tree->left = NULL;  
    Tree->right = NULL;  
    return Tree;  
}
```

один символ, число

повертаємо адреса
створеного вузла


```
//-----  
// Функція MakeTree - побудова дерева  
// Вхід:  рядок, номери першого і останнього  
//        символів частини, які розглядаються  
// Вихід: адреса побудованого дерева  
//-----  
PNode MakeTree ( char Expr[], int first, int last )  
{  
    PNode Tree;  
    int k;  
    if ( first == last )  
        return NumberNode ( Expr[first] );  
    k = LastOperation ( Expr, first, last );  
    Tree = new Node;  
    Tree->data  = Expr[k];  
    Tree->left  = MakeTree ( Expr, first, k-1 );  
    Tree->right = MakeTree ( Expr, k+1, last );  
    return Tree;  
}
```

залишилось
тільки число

новий вузол:
операція

Обчислення виразу за деревом

```
//-----  
// Функція CalcTree - обчислення за деревом  
// Вхід:  адреса дерева  
// Вихід: значення виразу  
//-----  
int CalcTree (PNode Tree)  
{  
    int num1, num2;  
    if ( ! Tree->left ) return Tree->data - '0';  
    num1 = CalcTree( Tree->left);  
    num2 = CalcTree(Tree->right);  
    switch ( Tree->data ) {  
        case '+': return  num1+num2;  
        case '-': return  num1-num2;  
        case '*': return  num1*num2;  
        case '/': return  num1/num2;  
    }  
    return 32767;  
}
```

повернути число,
якщо це лист

обчислюємо
операнди
(піддерева)

виконуємо
операцію

некоректна
операція

Основна програма

```
//-----  
// Основна програма: введення та обчислення  
// вираз за допомогою дерева  
//-----  
void main()  
{  
    char s[80];  
    PNode Tree;  
    printf ( "Введіть вираз > " );  
    gets(s);  
    Tree = MakeTree ( s, 0, strlen(s)-1 );  
    printf ( "= %d \n", CalcTree ( Tree ) );  
    getch();  
}
```

Дерево гри

Приклад 7.2

Задача. Перед двома гравцями лежать дві купки камінців, у першій з яких 3, а у другій – 2 камінця. У кожного гравця необмежено багато камінців.

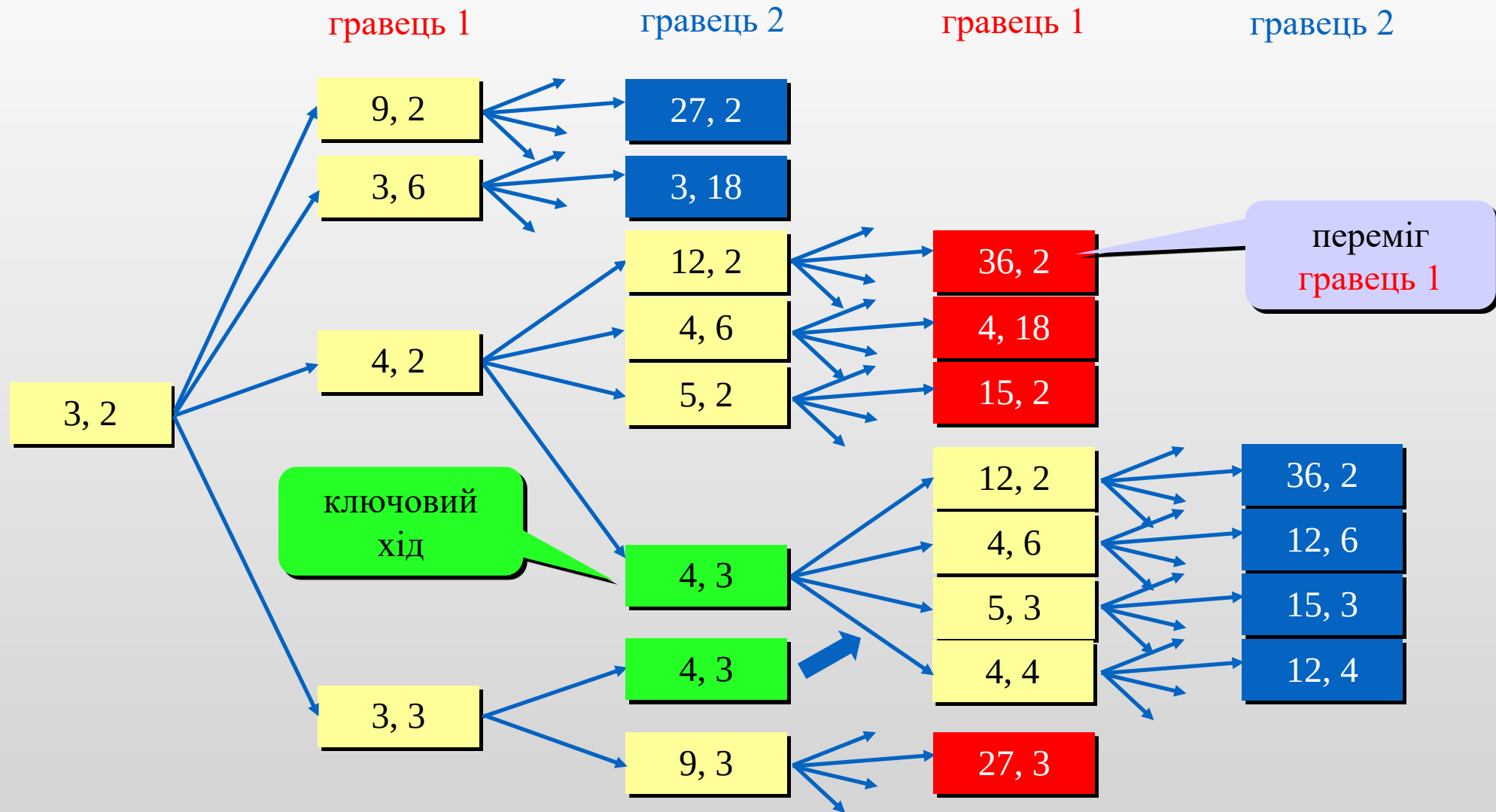
Гравці ходять по черзі. Хід полягає у тому, що гравець або **збільшує в 3 рази** число камінців у якійсь купі, або **додає 1 камінець** у якусь купу.

Перемагає гравець, після ходу якого загальне число камінців у двох купах стає **не менше 16**.

Хто перемагає при безпомилковій грі – гравець, який робить перший хід, або гравець, який робить другий хід? Як повинен ходити гравець, який переможе?



Рисунок 7.26



При правильній грі перемагає гравець 2!

Рисунок 7.27

7.4 ЗАГАЛЬНЕ ПОНЯТТЯ АВТОМАТІВ ТА РОБОТА З НИМИ

Автомат – це абстрактний об’єкт, призначений для виконання певної роботи.

Автомат отримує для опрацювання вхідну послідовність даних, а як результат видає вихідну послідовність даних.

Означення скінченного автомата було наведено 1955 р. Джорджем Х. Мілі й 1956 р. Едвардом Ф. Муром.

$$M = (S, X, Y, F, s_0)$$

S – непорожня скінчена множина станів, елементи якої ($s_1, s_2, s_3, \dots$);

X – непорожня скінчена множина вхідного алфавіту автомату;

Y – непорожня скінчена множина вихідного алфавіту автомата;

F – відображення $F: S \times X \rightarrow S \times Y$;

s_0 – початковий стан автомата.

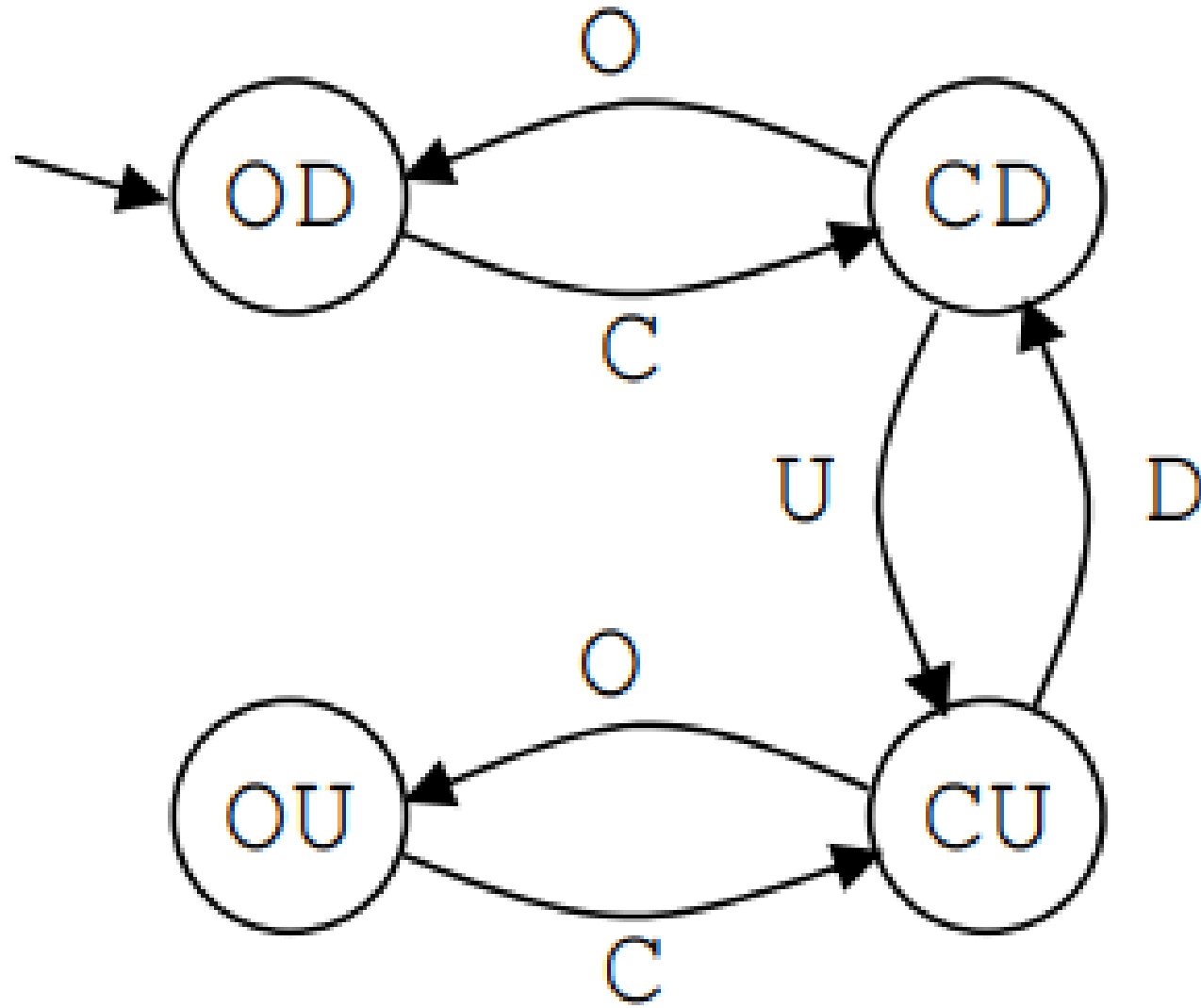
Алфавіт автомата — це скінчена множина певних символів.

Наприклад, множина $\{1, 2\}$ — це алфавіт, який складається з одиниці та двійки, множина $\{A, B, \dots, Z\}$ — це алфавіт великих латинських літер.

Вхідним алфавітом є скінчена множина допустимих вхідних символів, з якої формуються зчитуванні автоматом рядки.

Вихідний алфавіт — це скінчена множина допустимих вихідних символів, з якої формуються рядки-результати.

Приклад 7.3



Команди:

O – відчинити двері (від англ. open – відчинити);

C – зачинити двері (від англ. close – зачинити);

D – спуститися на перший поверх (від англ. down – донизу);

U – піднятися до другого поверху (від англ. up – догори).

Стани:

CD – перший поверх, двері зачинено;

OD – перший поверх, двері відчинено;

CU – другий поверх, двері зачинено;

OU – другий поверх, двері відчинено;

Рисунок 7.28 Граф переходу автомата у вигляді ліфту у двоповерховому будинку

Таблиця 7.1 – Правила виконання «Стан-Команда»

Стан	Команди			
	O	C	U	D
CD	OD	—	CU	—
OD	—	CD	—	—
CU	OU	—	—	CD
OU	—	CU	—	—

Автомат називають **синхронним**, якщо він перетворює один елемент вхідного на один елемент вихідного алфавіту.

Програми, що реалізують синхронні автомати, можуть використовувати оператори циклу **for** для опрацювання послідовності елементів та вкладені в нього оператори **switch** та **if** для аналізу й перетворювання елементів.

Приклад 7.4 Автомат задано алфавітом $X = Y = \{0, 1\}$ і трьома станами – A, B, C:

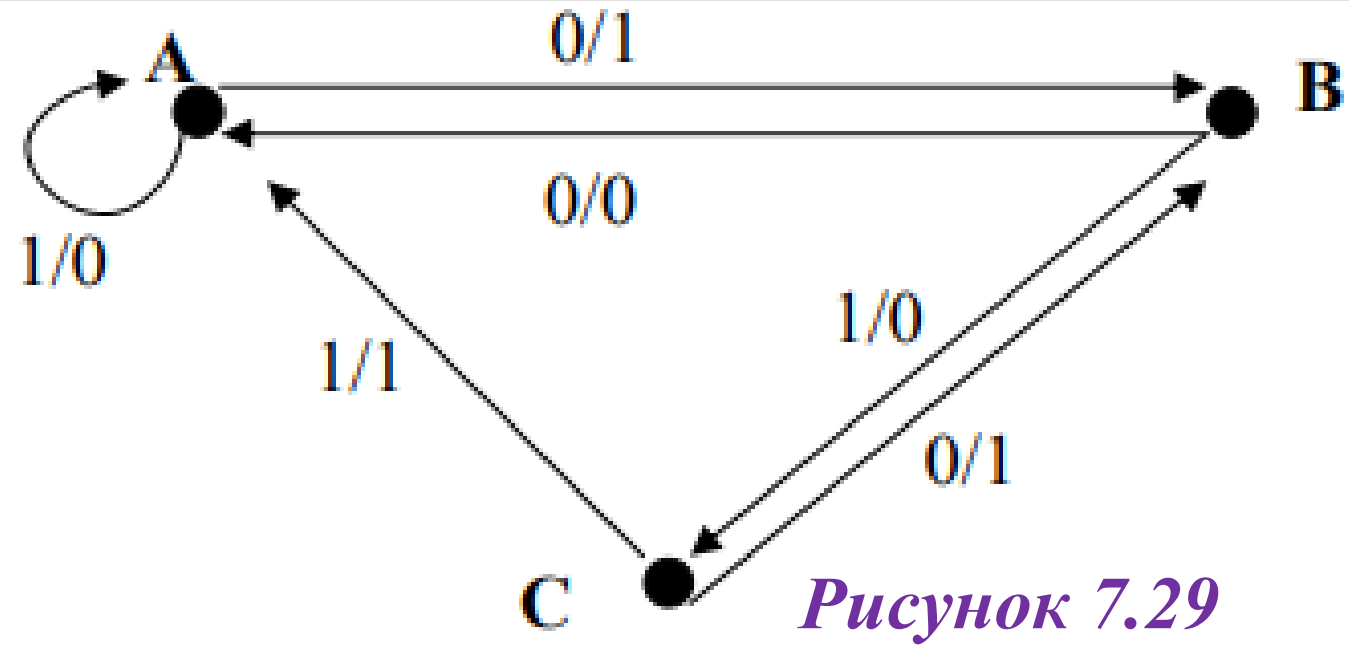


Рисунок 7.29

Стан	Вхідний елемент	
	0	1
A	В, 1	А, 0
В	А, 0	С, 0
С	В, 1	А, 1

Рисунок 7.30

```

AnsiString Automat1(AnsiString S)
{ // Для визначення станів автомата використовується перераховний тип даних
  enum conditions{A, B, C};
  conditions sit=A; //Початковий стан автомата
  for(int i=1;i<=S.Length();i++)
  switch (sit)
  {case A:{if (S[i]=='0') {S[i]='1'; sit=B;}else S[i]='0';break;}
  case B:{if (S[i]=='0') sit=A; else sit=C; S[i]='0';break;}
  case C:{if (S[i]=='0') sit=B; else sit=A; S[i]='1';break;}
  }
  return S;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{ AnsiString S=Edit1->Text;
  Edit2->Text=Automat1(S); }

```

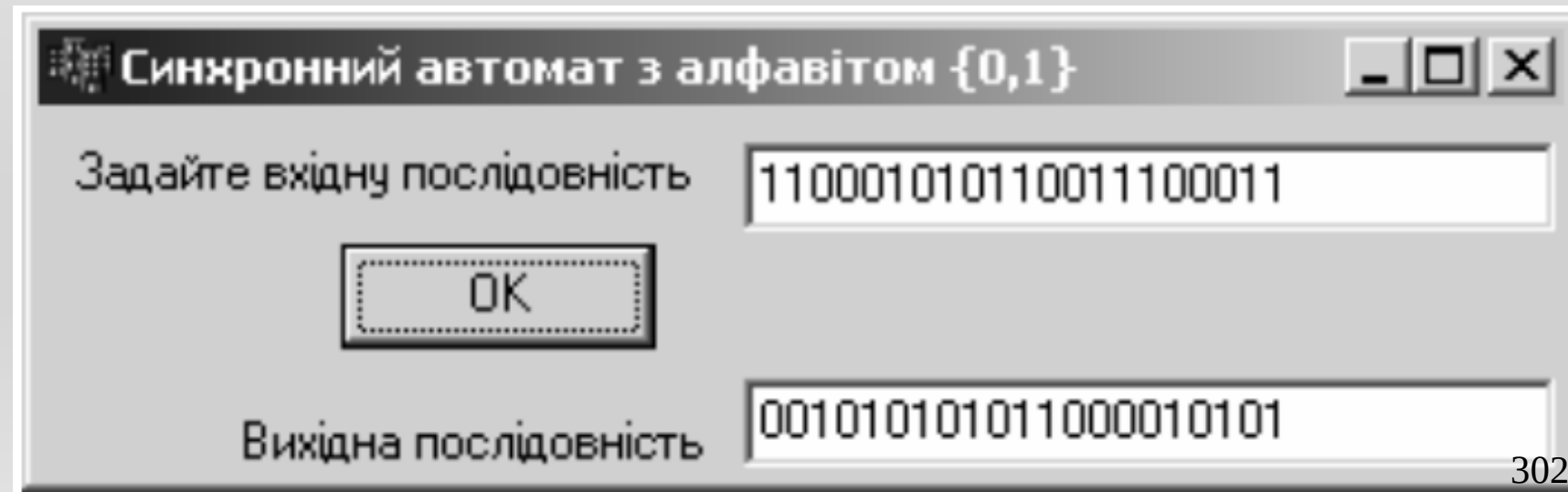


Рисунок 7.31

Автомат називається *асинхронним*, якщо вхідна послідовність може бути перетворена ним на послідовність іншої довжини, тобто один символ може бути перетворено у довільну кількість символів.

Автомат A називають *оборотним*, якщо існує інший автомат B , який перетворює довільну вихідну послідовність автомата A на вхідну послідовність автомата A .

Автомат неодмінно має бути оборотним, якщо його використовують для шифрування.

Приклад 7.5 Задано автомат алфавітом $X=Y=\{0, 1, 2\}$ і двома станами A та B:

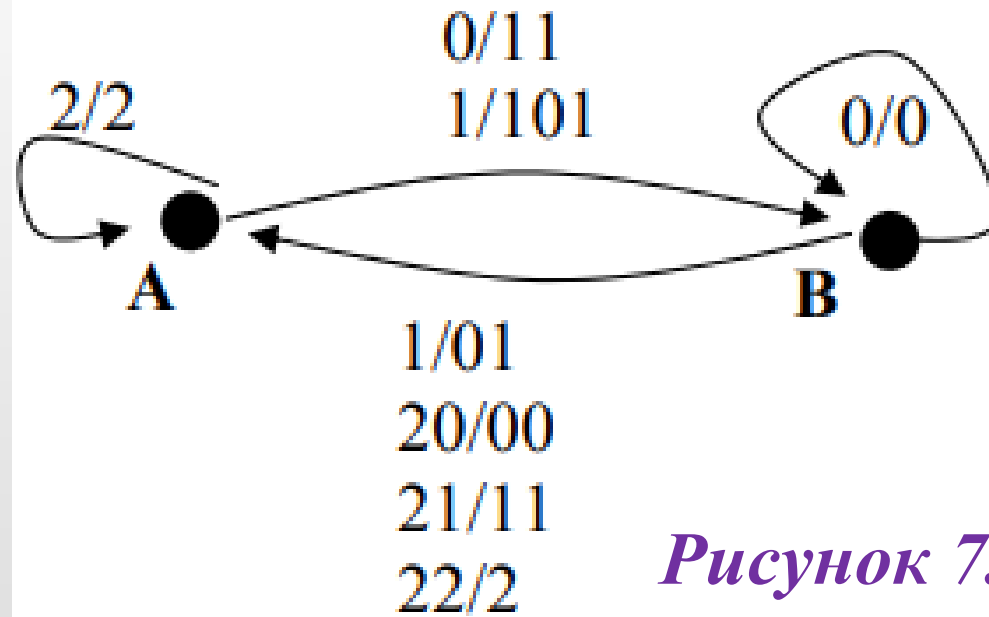


Рисунок 7.32

Асинхронний автомат з алфавітом {0,1,2}

Задайте вхідну послідовність

✓ OK

Вихідна послідовність

Рисунок 7.33

```

AnsiString Assinhr1(AnsiString S)
{ enum conditions {A, B}; //
Перераховний тип даних
  conditions sit=A;
  AnsiString T=""; // Рядок, в якому
  формуватиметься результат.
  while(!S.IsEmpty( )) // Допоки рядок
  не є порожній,
  { switch (sit) // перевіряються усі
  стани автомата
  { case A:
  { if(S[1]=='0'){T=T+"11"; sit=B;}
  else
  if(S[1]=='1'){T=T+"101"; sit=B;}
  else T=T+"2";
  S.Delete(1,1); // Видалення
  опрацьованого символу.
  break;
  } // end case A
  case B:
  { if(S[1]=='0') T=T+"0";
  else

```

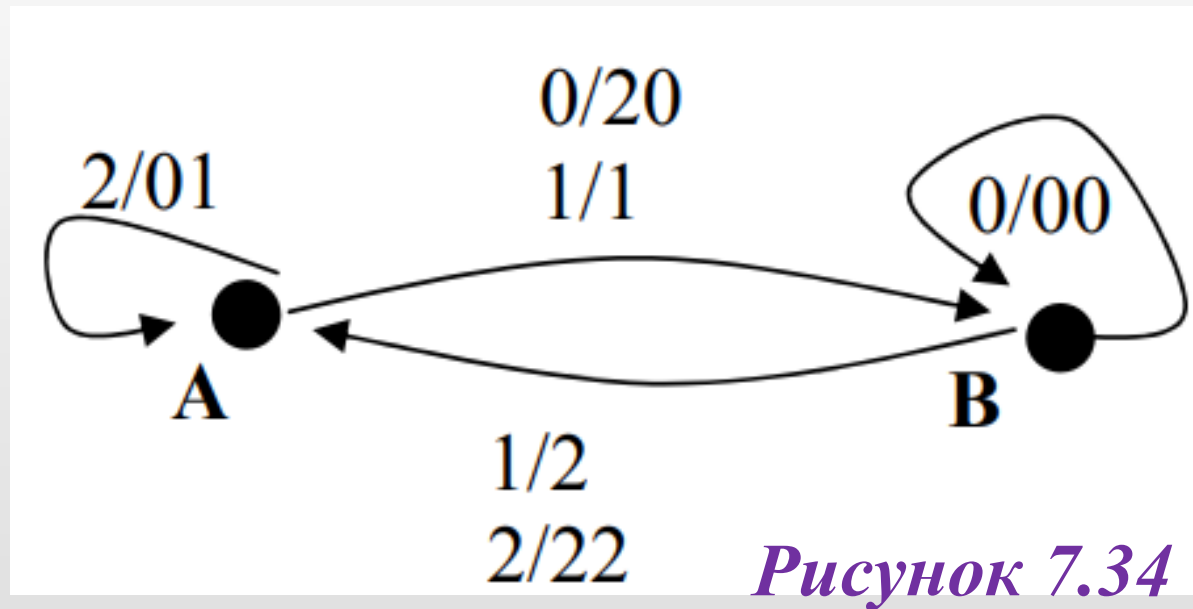
```

{if(S[1]=='1'){T=T+"01"; sit=A;}
else
if(S.Length()>1)
{ String sub=S.SubString(1,2);
if(sub=="20") T=T+"00"; else
if(sub=="21")T=T+"11"; else
if(sub=="22") T=T+"2";
sit=A;
} }
if((S[1]=='2')&&(S.Length()>=1))S.Delete(1,2);
else S.Delete(1,1);
break;
} // end case B
} // switch
} // while
return T;
}

void __fastcall TForm1::BitBtn1Click(TObject
*Sender)
{ AnsiString s=Edit1->Text;
  Edit2->Text=Assinhr1(s);
}

```

Приклад 7.6 Задано автомат алфавітом $X=Y=\{0, 1, 2\}$ і двома станами – A та B:



Асинхронний автомат з алфавітом {0,1,2}

Задайте вхідну послідовність

Вихідна послідовність

Рисунок 7.35

```

AnsiString Assinhr2 (AnsiString S)
{ enum conditions{A,B};
  conditions sit=A;
  AnsiString T=""; // Рядок-результат
  for (int i=1;i<=S.Length();i++)
  switch (sit)
  { case A:
    { if (S[i]=='1'){T=T+"1"; sit=B;}
    else
    if (S[i]=='0'){T=T+"20"; sit=B;}
    else T=T+"01";
    break;
  } // end case A
  case B:
  { if(S[i]=='0') T=T+"00";
  else
  if(S[i]=='1'){T=T+"2"; sit=A;}
  else

```

```

    { T=T+"22"; sit=A;}
    break;
  } // end case B
} // switch
return T;
}
//-----
void __fastcall
TForm1::BitBtn1Click(TObject *Sender)
{ String s=Edit1->Text;
  Edit2->Text= Assinhr2(s);
}
//-----
void __fastcall
TForm1::Edit1KeyPress(TObject *Sender,
char &Key)
{ if(Key<'0' || Key>'2') Key=0;
}

```


Приклад 7.7 Нехай задано два асинхронні автомати K та M з алфавітами $X=Y=\{0, 1, 2\}$ і двома станами A та B . Створити новий автомат N як композицію автоматів K та M . Порядок використання автоматів у композиції (зліва-направо) задано послідовністю елементів алфавіту $\{K, M\}$.

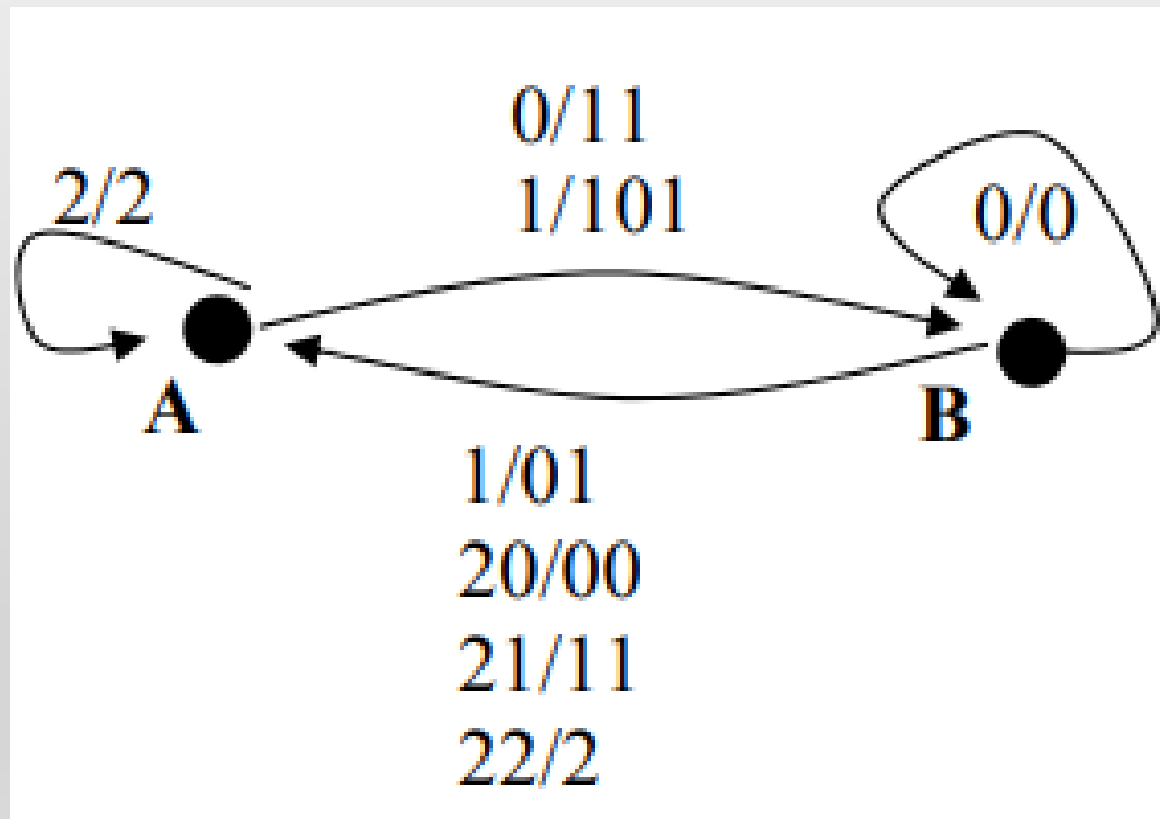


Рисунок 7.36

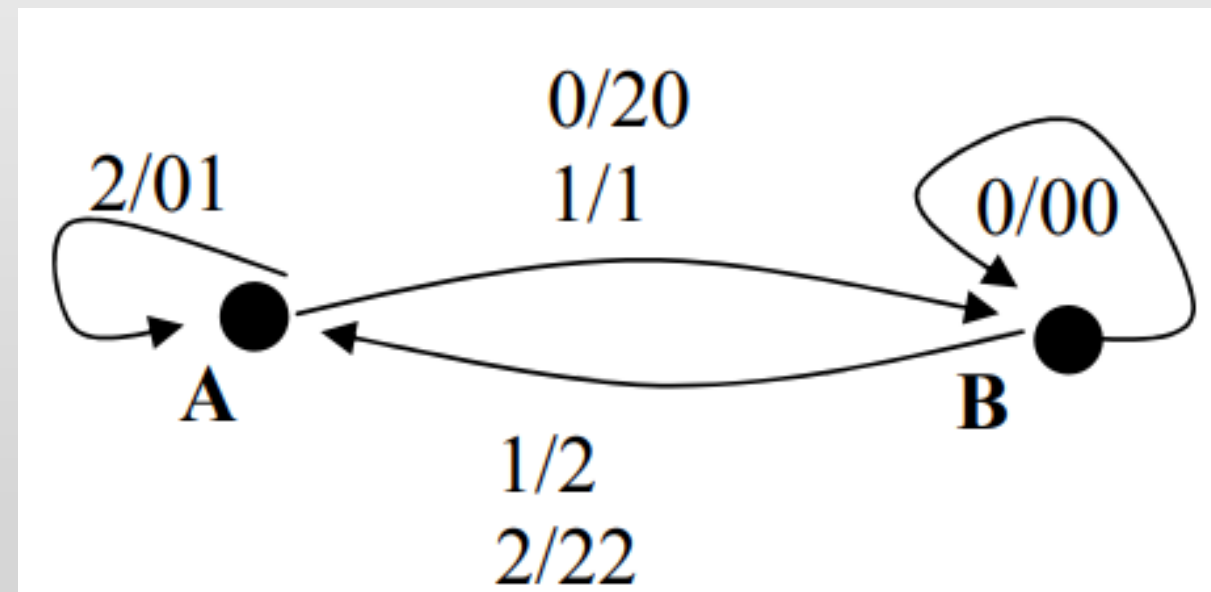


Рисунок 7.37

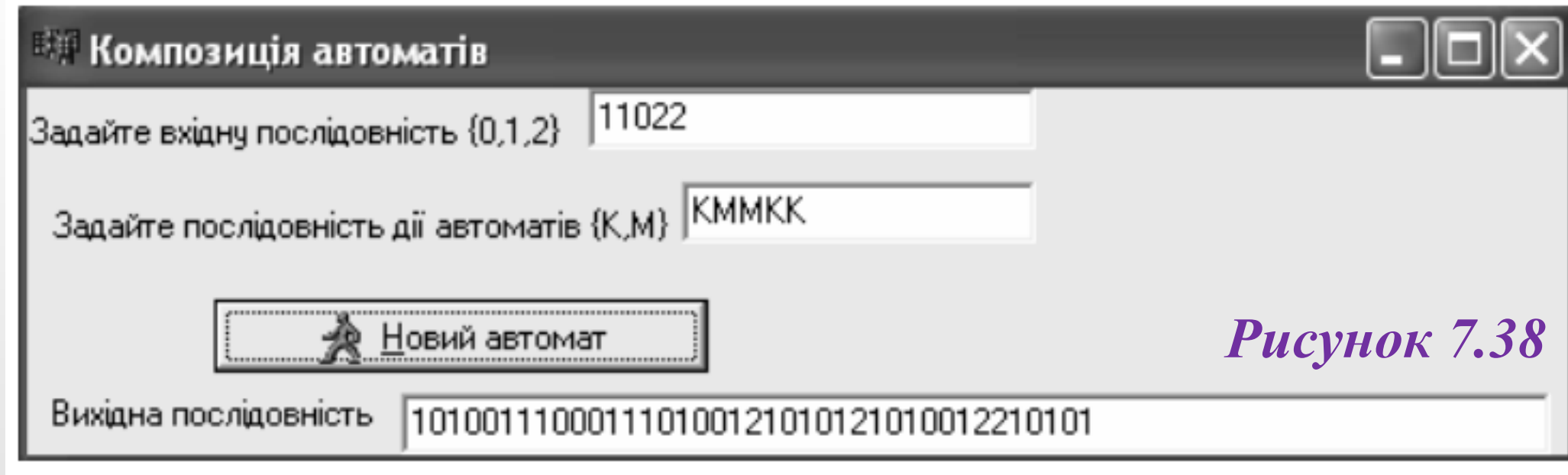


Рисунок 7.38

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    AnsiString S=Edit1->Text;
    AnsiString T=Edit2->Text;
    while(T!="")
    {
        if(T[1]=='K') S= Assinhr1(S);else S= Assinhr2(S);
        T.Delete(1,1);
    }
    Edit3->Text=S;
}
```

Тема №8. РОЗШИРЕНІ МОЖЛИВОСТІ РОБОТИ З КОНСОЛЬНИМ ВІКНОМ

Тематичний план

8.1 Огляд можливостей роботи з консольним вікном

8.1 ОГЛЯД МОЖЛИВОСТЕЙ РОБОТИ З КОНСОЛЬНИМ ВІКНОМ

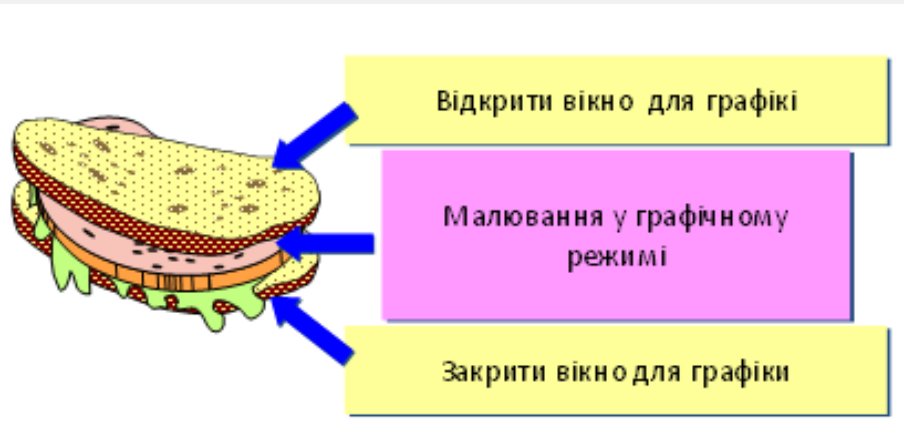
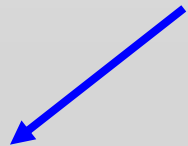
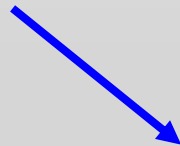


Рисунок 8.1

Режими роботи



Текстовий



Графічний

Borland C ++ (Turbo C) – середовище програмування (IDE), розроблена фірмою Borland для створення програм на мовах програмування Сі та С ++(90-ті роки).

```
#include <graphics.h> // підключити графічні функції  
#include <conio.h>
```

```
void main()  
{  
    int d = VGA, m = VGAHI; //визначити тип драйвера екрана режим  
    VGAHI: 640x480, 16 кольорів  
  
    initgraph ( &d, &m, "c:\\borlandc\\bgi" ); //включити графічний  
    режим  
    getch();  
    closegraph(); //виключити графічний режим  
}
```

Win32 API (WinAPI) – це набір функцій (*API – Application Programming Interface*), що працюють під управлінням ОС Windows.

Вони містяться в бібліотеці **windows.h**.

Функції API в текстовому режимі

BOOL SetConsoleTextAttribute(HANDLE**,**WORD**);**

HANDLE – дескриптор консолі

Дескриптор (описувач) об'єкта - службова структура даних, що представляє собою беззнаковий цілий номер і служить для ідентифікації різних об'єктів.

Отримання дескриптора вікна

```
HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
```

STD_OUTPUT_HANDLE - дескриптор стандартного пристрою вводу / виводу stdout

Біти слова WORD:

I_b R_b G_b B_b I_f R_f G_f B_f

I_b и I_f - Інтенсивність фону та тексту

R_b, G_b, B_b - колір фону

R_f, G_f, B_f - колір тексту

0 до 255

Приклад 8.1

```
SetConsoleTextAttribute(hStdOut, 2);  
Printf("Green\n");
```

2 -> 00000010

$$G_f = 1$$

Управління кольором через прапори

```
SetConsoleTextAttribute(hStdOut, FOREGROUND_RED | FOREGROUND_GREEN  
| FOREGROUND_INTENSITY);  
printf("Yellow\n");
```

00001110

I_b R_b G_b B_b I_f R_f G_f B_f

$$I_f = 1$$

$$R_f = 1$$

$$G_f = 1$$

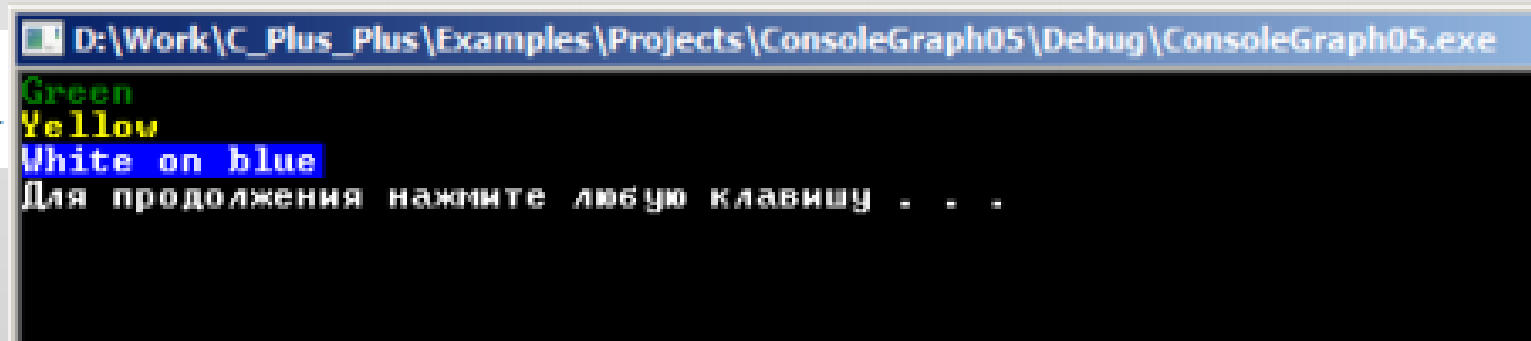


Рисунок 8.2

```
SetConsoleTextAttribute(hStdOut, BACKGROUND_BLUE | BACKGROUND_INTENSITY |  
FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE | FOREGROUND_INTENSITY);  
printf("White on blue\n");
```



$$I_b R_b = \{00, 01, 10, 11\}$$

За вертикаллю

$$G_b B_b = \{00, 01, 10, 11\}$$

За горизонталлю

$$I_b \quad R_b \quad G_b \quad B_b \quad I_f \quad R_f \quad G_f \quad B_f$$

Рисунок 8.3


```
SetConsoleTitle("Заголовок консольного окна");
```

```
system("title Заголовок консольного окна");
```

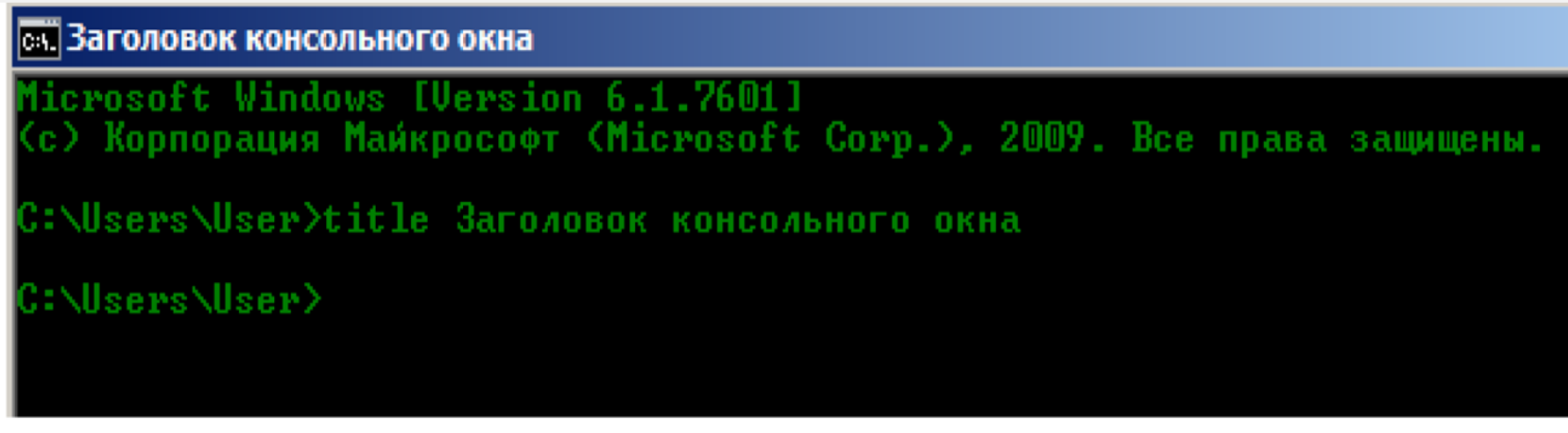


Рисунок 8.4

```
COORD cursorPos;
```

```
cursorPos.X = 5;
```

```
cursorPos.Y = 2;
```

```
SetConsoleCursorPosition(hStdOut, cursorPos);
```

cursorPos – структура з 2 полями X,Y з координати текстового курсору

Приклад 8.2 Відображення двічі «текстового» прямокутника різними кольорами.

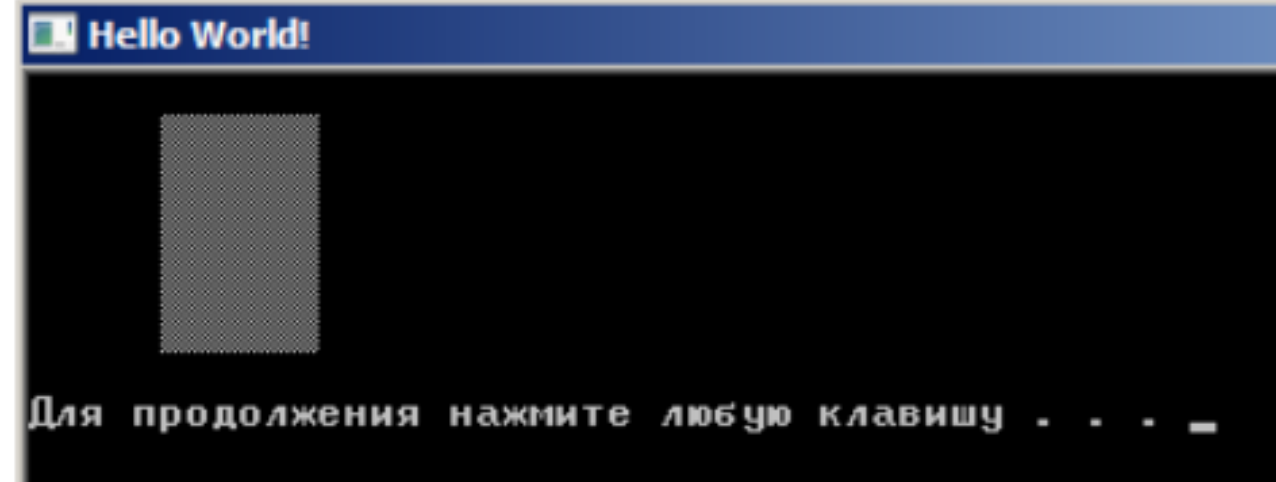
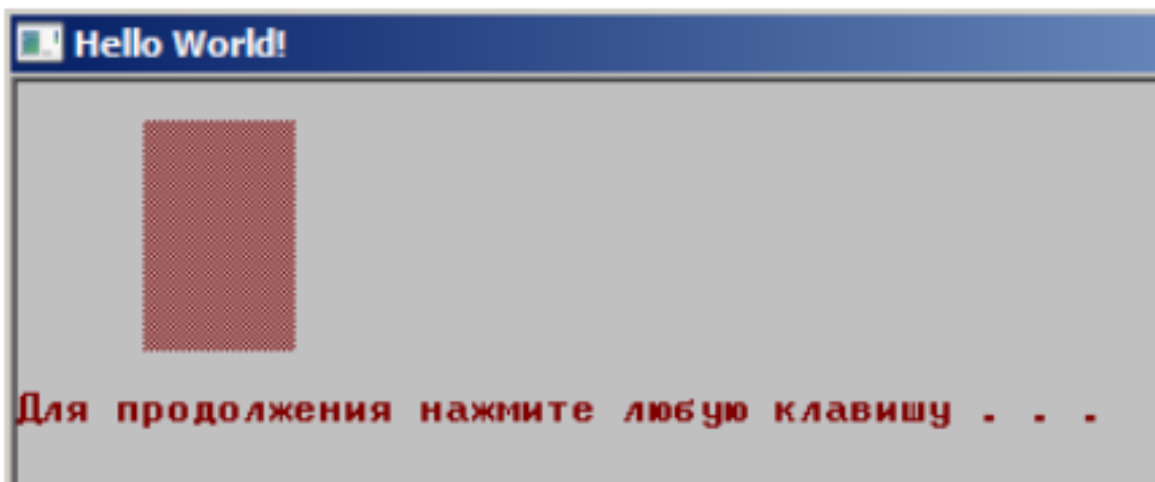


Рисунок 8.5

```

#include <windows.h>
CONSOLE_SCREEN_BUFFER_INFO csbiInfo;
void drawBox(HANDLE hStdOut, COORD Pos);
void main()
{
    SetConsoleTitle("Hello World!");    //заголовок вікна
    HANDLE hStdOut =
        GetStdHandle(STD_OUTPUT_HANDLE);
    GetConsoleScreenBufferInfo(hStdOut, &csbiInfo);
    WORD wOldColorAttrs;
    wOldColorAttrs = csbiInfo.wAttributes; // старі атрибути
    SetConsoleTextAttribute(hStdOut, BACKGROUND_RED |
        BACKGROUND_GREEN | BACKGROUND_BLUE
        | FOREGROUND_RED);
    system("CLS"); // заливка консольного вікна сірим
    кольором
    COORD cursorPos;
    cursorPos.X = 5; // координати лівого верхнього кута
    cursorPos.Y = 1; // координати лівого верхнього кута
    drawBox(hStdOut, cursorPos);
    cursorPos.X = 0;
    cursorPos.Y = 8;
    SetConsoleCursorPosition(hStdOut, cursorPos);
    system("pause");
}

```

```

// Повертаємо вихідні кольори
SetConsoleTextAttribute(hStdOut, wOldColorAttrs);
system("CLS");    //очищуємо/перифарбовуємо все
вікно
cursorPos.X = 5;
    cursorPos.Y = 1;
drawBox(hStdOut, cursorPos); // Знову рисуємо
прямокутник
cursorPos.X = 0;
    cursorPos.Y = 8; // Нове положення текстового
курсору
SetConsoleCursorPosition(hStdOut, cursorPos);
system("pause");
}

void drawBox(HANDLE hStdOut, COORD Pos)
{
    char c='+';
    for (int i = 0; i < 6; i++) {
        SetConsoleCursorPosition(hStdOut, Pos);
        for (int j = 0; j < 6; j++) printf("%c",c);
        Pos.Y += 1;
    }
}

```

BOOL WINAPI FillConsoleOutputCharacter(HANDLE hConsoleOutput, // ідентифікатор вікна консолі
TCHAR cCharacter, // символ, який буде друкуватися
DWORD nLength, // кількість символьних ячеек, в які символ повинен бути надрукований
COORD dwWriteCoord, // COORD структура, яка задає координати першої ячейки
LPDWORD lpNumberOfCharsWritten // Вказівник на змінну, яка отримує кількість символів, що дійсно надруковані)

Приклад 8.3 Виведення підряд 200 символів "X", починаючи з 10 - го осередка 5 - го рядка.

```
DWORD cWrittenChars;  
FillConsoleOutputCharacter(wHnd,(TCHAR)'X', 200,{10,5},&cWrittenChars);
```

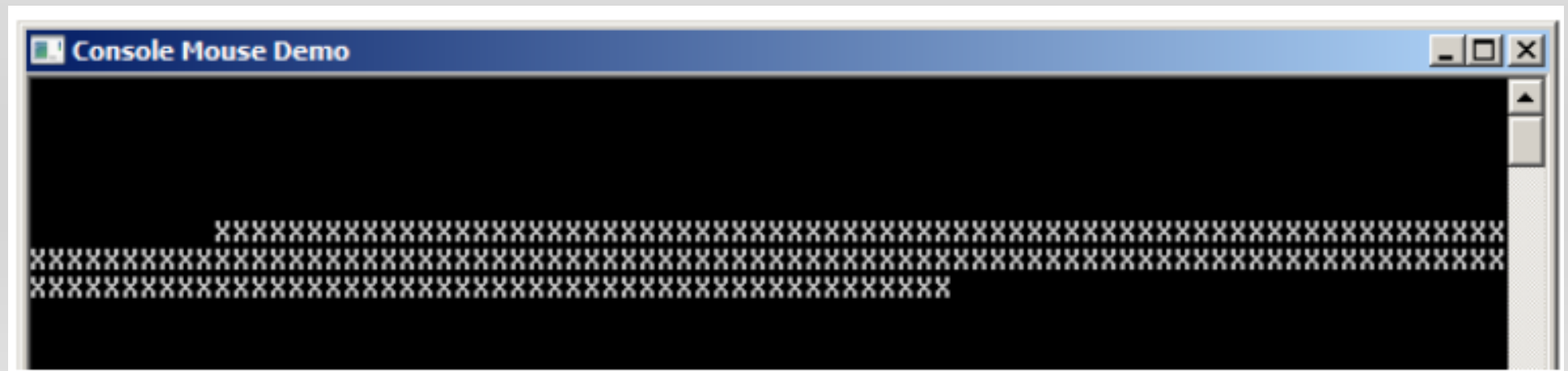


Рисунок 8.6

BOOL WINAPI FillConsoleOutputAttribute(HANDLE hConsoleOutput, // ідентифікатор вікна консолі
WORD wAttribute, // атрибут, що заповнює
DWORD nLength, // кількість ячеек, в які повинен бути заповнені атрибутом
COORD dwWriteCoord, // координати першої ячейки
LPDWORD lpNumberOfAttrsWritten // Вказівник на змінну, яка отримує кількість символних ячеек, чиї атрибути дійсно встановлені)

Приклад 8.4 Встановити бірюзовий колір ячеек, починаючи з 5 - го по 7 - мий рядки і ще 20 осередків 8 - го рядка.

```
DWORD cWrittenChars;  
FillConsoleOutputCharacter(wHnd,(TCHAR)'X',200, {10,5},  
&cWrittenChars);  
WORD attr = BACKGROUND_BLUE |  
BACKGROUND_GREEN |BACKGROUND_INTENSITY;  
FillConsoleOutputAttribute(wHnd,attr,260,{0,5},&cWrittenC  
hars);
```

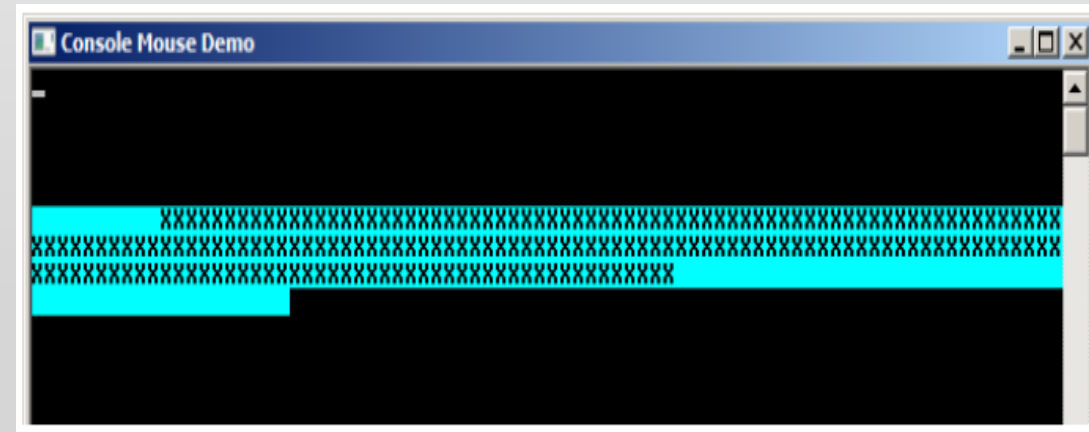


Рисунок 8.7

Функції API в графічному режимі

HDC hdc = **GetDC**(GetConsoleWindow()); Отримання контексту вікна

ReleaseDC(**HWND**, hdc); Звільнення дескриптора контексту вікна

HWND hwnd = GetConsoleWindow(); **HWND** – ідентифікатор вікна

Обробка помилок

```
HWND hwnd = GetConsoleWindow();
HDC hdc;
if (hwnd!=NULL)
{
    hdc=GetWindowDC(hwnd);
    if(hdc==0)
        printf("Error DC Window");
}
else
    printf("Error Find Window" );
```

Приклад 8.5. Отрисовка прямоуготника в консольному вікні.

```
#include <windows.h>
void main()
{
    HDC hdc = GetDC(GetConsoleWindow());
    Rectangle(hdc, 100, 60, 180, 160);
    system("pause");
}
```



Рисунок 8.8

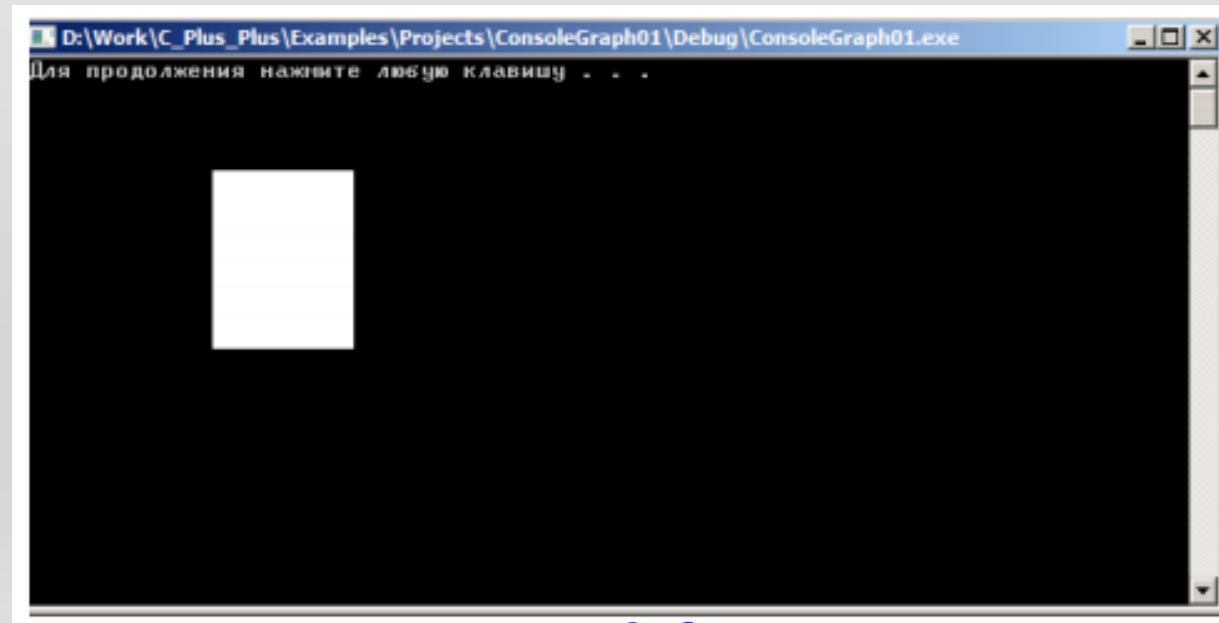


Рисунок 8.9

Отримання розмірів консольного вікна

```
#include <stdio.h>
#define _WIN32_WINNT 0x0500
#include <windows.h>
int main()
{HWND hWnd=GetConsoleWindow();
RECT rc;
GetClientRect(hWnd, &rc);
printf(" X=%i Y=%i",rc.right, rc.bottom);
getchar();
return EXIT_SUCCESS;}
```

Рисунок 8.10

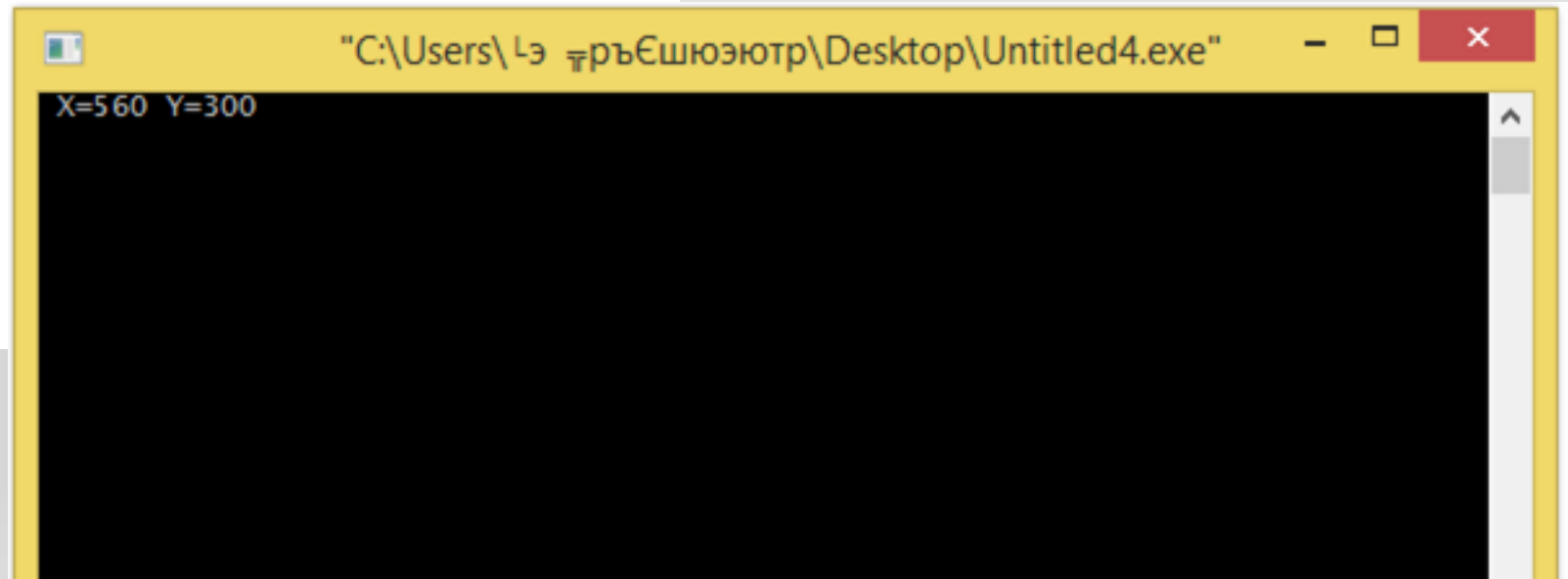


Рисунок 8.11

HPEN Pen = CreatePen(PS_SOLID, 3, RGB(255,0,0)); Стил ь лінії (перо)

PS_SOLID – суцільна лінія

3 – товщина лінії

RGB(255,0,0) – колір лінії (макрос)

PS_DOT, PS_DASH, PS_DASHDOT

SelectObject(HDC,HPEN) – завантаження пера у контекст

MoveToEx(HDC,X,Y,NULL); переносить поточну позицію курсору в точку (X,Y)

LineTo(HDC,X,Y) малює відрізок прямої від поточного положення
покажчика до точки (X, Y) і переміщає покажчик в цю точку

Приклад 8.6

```
#include <windows.h>
void main()
{
// отримуємо дескриптор контексту консольного вікна
HDC hdc = GetDC(GetConsoleWindow());
Rectangle(hdc,100,100,200,200); // Рисуємо квадрат
HPEN Pen = CreatePen(PS_SOLID,3,RGB(255,0,0)); // створюємо перо
SelectObject(hdc, Pen); // завантажуюємо створене перо в контекст
// переміщуємо покажчик у центр квадрата
MoveToEx(hdc,150,150,NULL);
//рисуємо лінію від поточного положення покажчика
LineTo(hdc, 250, 50);
//рисуємо лінію з кінця попереднього відрізка
LineTo(hdc, 200, 200);
//переміщуємо вказівник у ліву верхню вершину квадрата
MoveToEx(hdc, 100, 100, NULL);
//рисуємо лінію від поточного положення курсору до точки (250,50)
LineTo(hdc, 250, 50);
system("pause");
DeleteObject(Pen);
ReleaseDC(NULL, hdc);
}
```

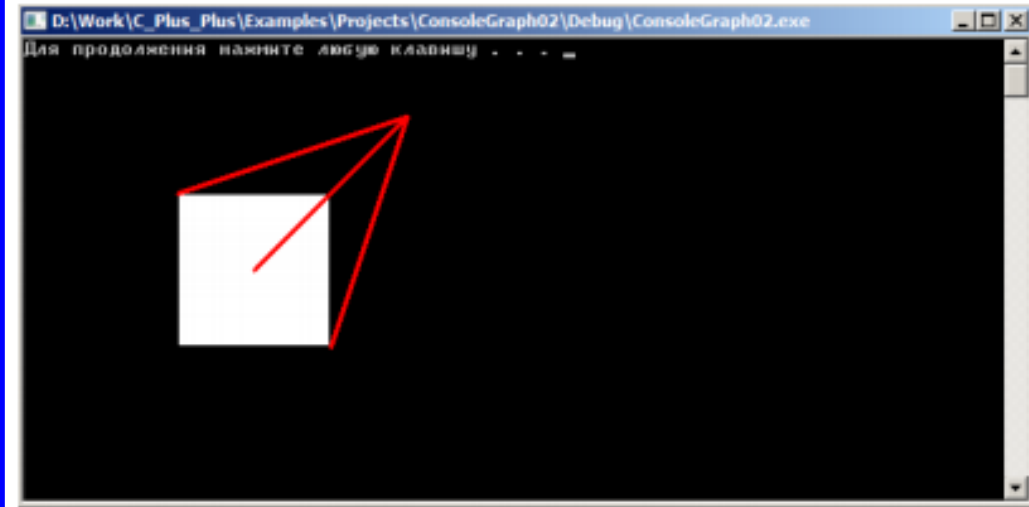


Рисунок 8.12

Встановлення кольору пікселя в точці екрану з координатами (X, Y)

SetPixel(HDC dc, int X, int Y, **COLORREF** Color);

COLORREF – структура, який створює макрос RGB

Функція повертає колір, який був встановлений в дійсності

Приклад 8.7 Рисунок синусоїди

```
#include <windows.h>
#include <math.h>
const double Pi = 3.141592;
void main()
{
    HDC hdc = GetDC(GetConsoleWindow());
    double x, y;
    COLORREF penColor = RGB(255, 255, 0);
    // Рисуємо графік синусоїди за точками
    for(x = 0; x < 100; x++){
        y = sin(Pi*x/25);
        SetPixel(hdc, 100+int(x), 100+int(50*y), penColor);
    }
    // створюємо перо для рисування
    HPEN Pen =
    CreatePen(PS_SOLID, 3, RGB(255, 0, 0));
    SelectObject(hdc, Pen);
    // рисуємо синусоїду за відрізками
    int
    xScale=10, yScale=50; // масштаб по осям X и Y
```

```
int xO=300, yO=100;    // положення початку
координат
// діапазон зміни абсциси
float xBeg = -8.0f, xEnd = 8.0f;
// вказівник у початкову точку кривої
MoveToEx(hdc, xO+xScale*xBeg,
yO+yScale*sin(xBeg), NULL);
// рисуємо відрізки від точки до точки
for (x=xBeg; x<=xEnd; x+=0.01f)
    LineTo(hdc, xO+xScale*x, yO+yScale*sin(x));
system("pause");
DeleteObject(Pen);
ReleaseDC(NULL, hdc);
}
```

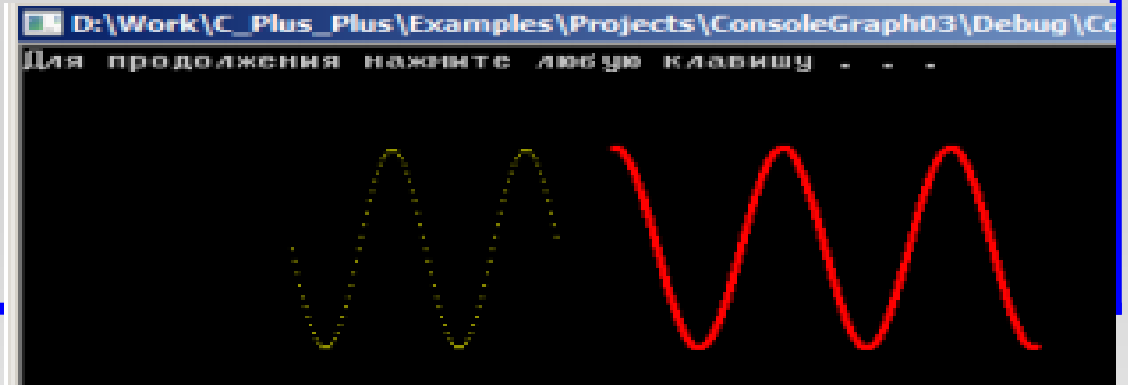
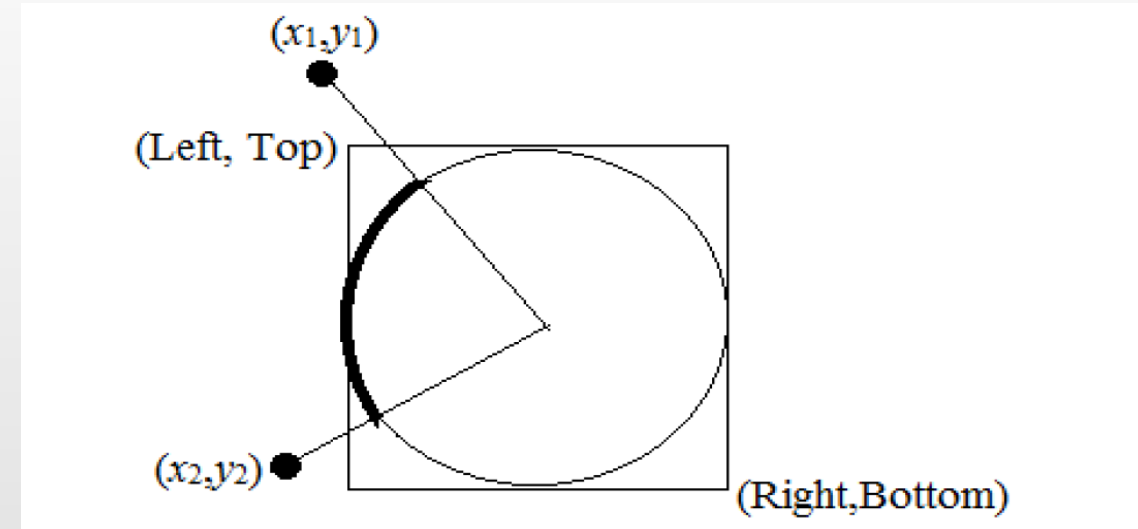


Рисунок 8.13

```
BOOL Arc (hDC, int left, int top, int right, int bottom, int x1, int y1, int x2, int y2);
```

```
Arc (hdc, 220, 100, 280, 160, 0, 0, 0, 0);
```



аналог

Рисунок 8.14

```
BOOL ArcTo (hDC, int left, int top, int right, int bottom, int x1, int y1, int x2, int y2);
```

Спочатку від поточного покажчика до початку дуги функція малює відрізок, потім малює дугу і встановлює поточний покажчик в її кінцеву точку.

```
BOOL AngleArc (HDC hdc, int X, int Y, DWORD R, FLOAT startAngle, FLOAT sweepAngle);
```

Рисую дугу окружності і відрізок від поточної позиції до початку цієї дуги.

```
BOOL Polyline (hdc, POINT *pt, int cPoints);
```

Рисує ламану, що проходить через точки, координати яких знаходяться в масиві pt.

cPoints функції Polyline передається кількість точок масиву (≥ 2).

Приклад 8.8 Рисунок трикутника

```
POINT ptr[4]={ {350, 90}, {400, 140}, {300, 140}, {350, 90} };  
Polyline(hdc, ptr, 4);
```

аналог

```
BOOL PolylineTo (hdc, POINT *pt, int cPoints);
```

Ламана починає рисуватися від поточного положення покажчика до першої точки масиву pt, і далі проходить через інші точки цього масиву. Функція переводить покажчик поточної позиції в кінцеву точку ламаної.

```
BOOL PolyPolyline (HDC hdc, const POINT *lppt, const DWORD
*lpdwPolyPoints, DWORD cCount); Рисує відразу кілька ламаних
```

`const POINT *lppt` - масив координат вершин всіх шматків ламаної

`lpdwPolyPoints` – це масив кількостей вершин в кожному шматку

```
BOOL TextOut (HDC hdc, int xStart, int yStart, LPCTSTR lpString,
int countChars);
```

`xStart` та `yStart` - визначають точку прив'язки текстового рядка `lpString`

`countChars` - містить кількість символів, яке слід надрукувати.

Приклад 8.9

```
char str[] = "Три ломаных"; TextOutA(hdc,
360, 40, str, strlen(str));
```

SetBkColor SetTextColor *зміна коліру тексту і коліру фону*

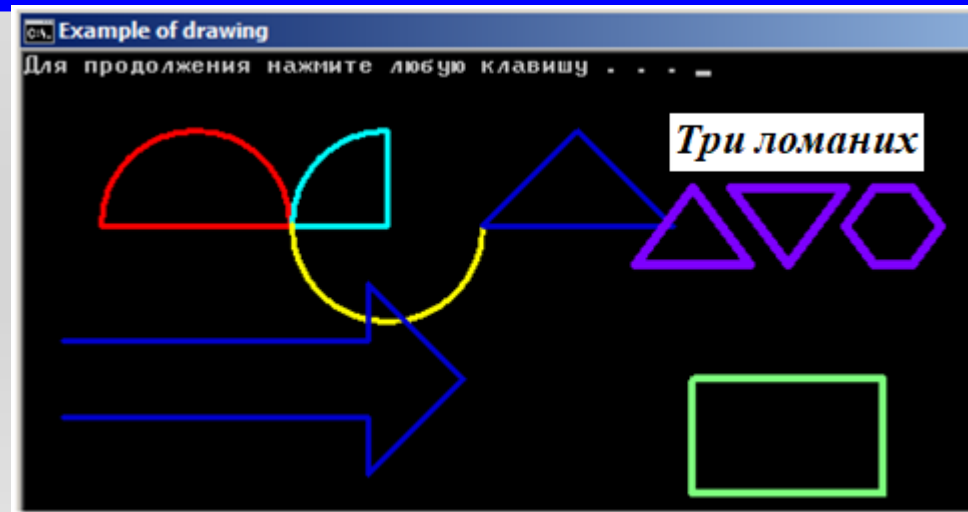
Приклад 8.10

```
COLORREF bk = SetBkColor(hdc, RGB(255, 255, 0));  
if (bk == CLR_INVALID)  
    Printf("Color error\n");  
SetTextColor(hdc, RGB(255, 0, 0));  
TextOutA(hdc, 260, 20, «Червоний текст на  
жовтому фоні", 28);
```


Приклад 8.11 Демонстрація функцій малювання кривих.

```
#include <windows.h>
BOOL Line(HDC hdc, int x1,int y1,int x2,int y2);
int main() {
SetConsoleTitle("Example of drawing");
HDC hdc=GetDC(GetConsoleWindow());
HPEN Pen = CreatePen(PS_SOLID, 3, RGB(255, 0, 0)); // перо 1
SelectObject(hdc, Pen); // поміщаємо перо 1 у контекст
Arc(hdc, 40,40,140,140,200,90,0,90); //Дуга півкола
Line(hdc,40,90,140,90); // діаметр півкола
HPEN Pen2 = CreatePen(PS_SOLID,3,RGB(0,255,255)); // перо 2
SelectObject(hdc, Pen2); // поміщаємо перо 2 в контекст
LineTo(hdc,190,90); // горизонтальний відрізок
```

Рисунок 8.15



```
// вертикальний радіус і дуга чверті кола
ArcTo(hdc,140,40,240,140,190,0,0,90);
HPEN
Pen3=CreatePen(PS_SOLID,3,RGB(255,255,0)); //
перо 3
SelectObject(hdc, Pen3); // поміщаємо перо
3 в контекст
AngleArc(hdc, 190,90,50,180,180); // жовте
півколо
HPEN Pen4=CreatePen(PS_SOLID,3,RGB(0,0,200));
// перо 4
SelectObject(hdc, Pen4); // поміщаємо перо
4 в контекст
POINT ptr[3]={{340,90},{290,40},{240,90}};
PolylineTo(hdc, ptr, 3); // рисуємо
трикутник
POINT
tArrow[7]={{20,150},{180,150},{180,120},{230,
170},{180,220},{180,190},{20,190}};
Polyline(hdc,tArrow,7); // рисуємо стрілку
```

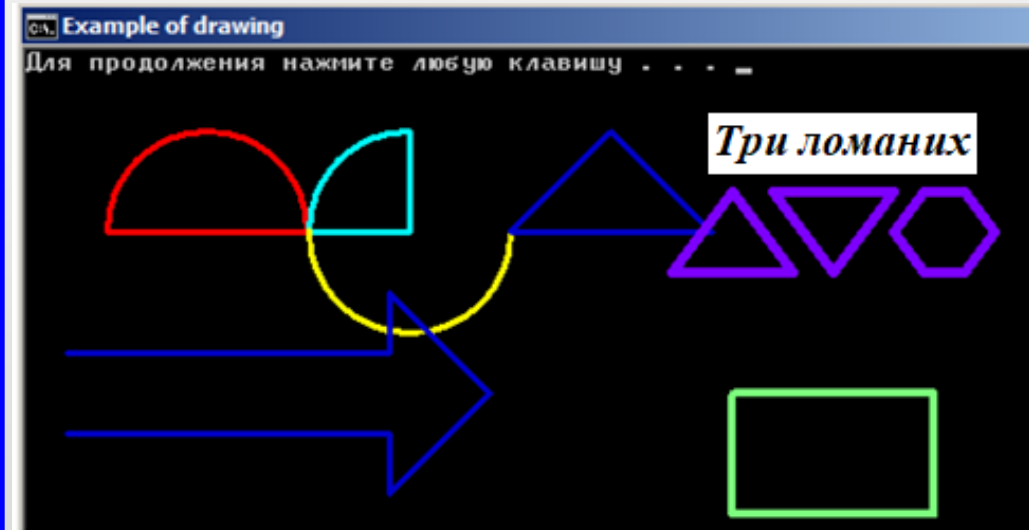


Рисунок 8.16

```

char str[] = "Три ломаних";
TextOutA(hdc, 360, 40, str, strlen(str)); // виводимо
текст
SelectObject(hdc, CreatePen(PS_SOLID, 5, RGB(125, 0, 255)
));
POINT
ptFigs[15]={{350, 70}, {320, 110}, {380, 110}, {350, 70},
{370, 70}, {400, 110}, {430, 70}, {370, 70}, {445, 70}, {430, 9
0}, {445, 110}, {465, 110}, {480, 90}, {465, 70}, {445, 70}};
DWORD PolyPoints[3]={4, 4, 7}; // рисуємо
відразу
PolyPolyline(hdc, ptFigs, PolyPoints, 3); // три
ломаних
SelectObject(hdc, CreatePen(PS_SOLID, 4, RGB(125, 255, 12
5)));
POINT ptRect[5]=
{{350, 170}, {450, 170}, {450, 230}, {350, 230}, {350, 170}};
Polyline(hdc, ptRect, 5); // рисуємо прямокутник
system("pause");
return 0;}
BOOL Line(HDC hdc, int x1, int y1, int x2, int y2) {
MoveToEx(hdc, x1, y1, NULL);
return LineTo(hdc, x2, y2); }

```

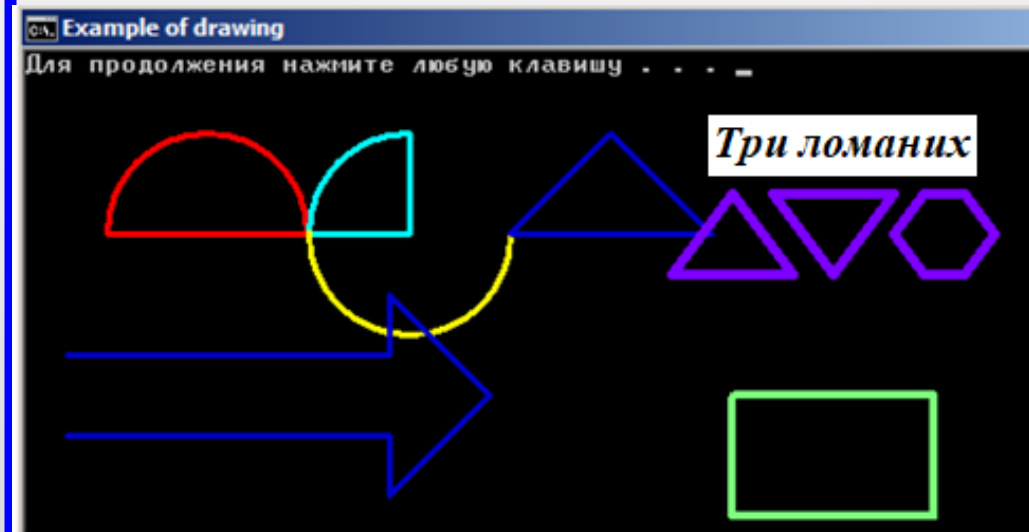


Рисунок 8.17

Тема №9. НАЛАГОДЖЕННЯ ПРОГРАМ

Тематичний план

- 9.1 Визначення помилок компіляції*
- 9.2 Попередження і підказки*
- 9.3 Компонування*
- 9.4 Помилки етапу виконання*

9.1 ВИЗНАЧЕННЯ ПОМИЛОК КОМПІЛЯЦІЇ

Помилки компіляції (помилки першого рівня) пов’язані з помилковим записом команд, тобто це орфографічні й синтаксичні помилки.

Таблиця 9.1 – Помилки компіляції

Форма повідомлення про помилку	Характер помилки
[C++ Error] Unit1.cpp(N): E2379 Statement missing;	Пропущено крапку з комою в N-му чи у попередньому (N-1)-му рядку
[C++ Error] Unit1.cpp(N): E2379 Statement missing;	Неправильний запис імені функції чи не долучено заголовний файл, у якому оголошено цю функцію
[C++ Error] Unit1.cpp(N): E2235 Member function must be called or its address taken	Пропущено дужки у виклику функції, яка міститься в N-му рядку
[C++ Error] Unit1.cpp(N): E2316 '_fastcall TForm::Button1Click (TObject*)'is not a member of 'TForm1'	Неправильно створено заголовок відгуку на подію Button1Click
[C++ Error] Unit1.cpp(N): E2089 Identifier 'Button2Click' cannot have a type qualifier	Неможливо створити відгук на подію Button2Click, оскільки не всі фігурні дужки закрито у попередній функції

Продовження таблиці 9.1 –

Форма повідомлення про помилку	Характер помилки
[C++ Error] Unit1.cpp(N): E2451 Undefined symbol 'x'	Змінну x не було оголошено до використання чи то ім'я змінної написано з помилкою
[C++ Error] Unit1.cpp(N): E2238 Multiple declaration for 'x'	Багаторазове оголошення змінної x в одній області видимості
[C++ Error] Unit1.cpp(N): E2134 Compound statement missing }	Кількість фігурних дужок, що відкриваються й закриваються, не збігається, а саме не вистачає закритої фігурної дужки у тексті програми, який розміщено вище
[C++ Error] Unit1.cpp(N): E2121 Function call missing)	Пропущено (чи помилково поставлено) круглу дужку, яка закриває список параметрів при викликанні функції
[C++ Error] Unit1.cpp(N): E2227 Extra parameter in call to pow(double, double)	Кількість параметрів при викликанні функції перевищує їхню кількість за синтаксисом
[C++ Error] Unit1.cpp(N): E2193 Too few parameters in call to 'pow(double, double)'	Кількість параметрів є менша, ніж цього вимагає синтаксис функції

Продовження таблиці 9.1 –

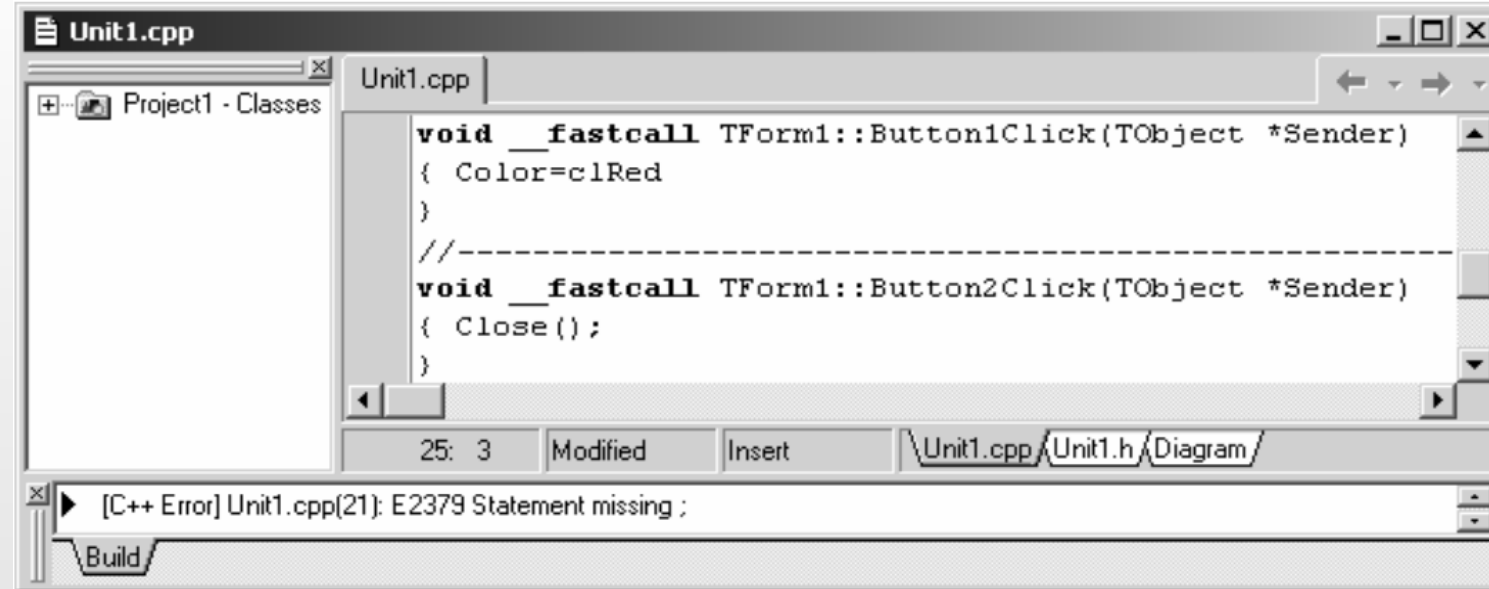
<i>Форма повідомлення про помилку</i>	<i>Характер помилки</i>
[C++ Error] Unit1.cpp(N): E2030 Misplaced break	Присутність break у цьому місці програми не має сенсу
[C++ Error] Unit1.cpp(N): E2054 Misplaced else	Неможливо встановити до якого if належить else
[C++ Error] Unit1.cpp(N): E2034 Cannot convert 'int' to 'char'	Суперечка типів: не можна присвоїти змінній типу char значення типу int
[C++ Error] Unit1.cpp(N): E2040 Declaration terminated incorrectly	Оголошення припинено некоректно. Помилка внаслідок неправильного розташування операторних дужок чи то дійсно некоректного оголошення
[C++ Error] Unit2.cpp(N): E2356 Type mismatch in redeclaration of 'sum(int, int)'	Невідповідність типу прототипу функції та її визначення (реалізації)
[C++ Error] Unit2.cpp(N): E2308 do statement must have while	Помилка виникає внаслідок неправильного розташування фігурних дужок в операторі do-while
[C++ Error] Unit2.cpp(N) E2308 Unterminated string or character constant	Рядок не завершено, тобто в кінці рядка пропущено лапки (")

Продовження таблиці 9.1 –

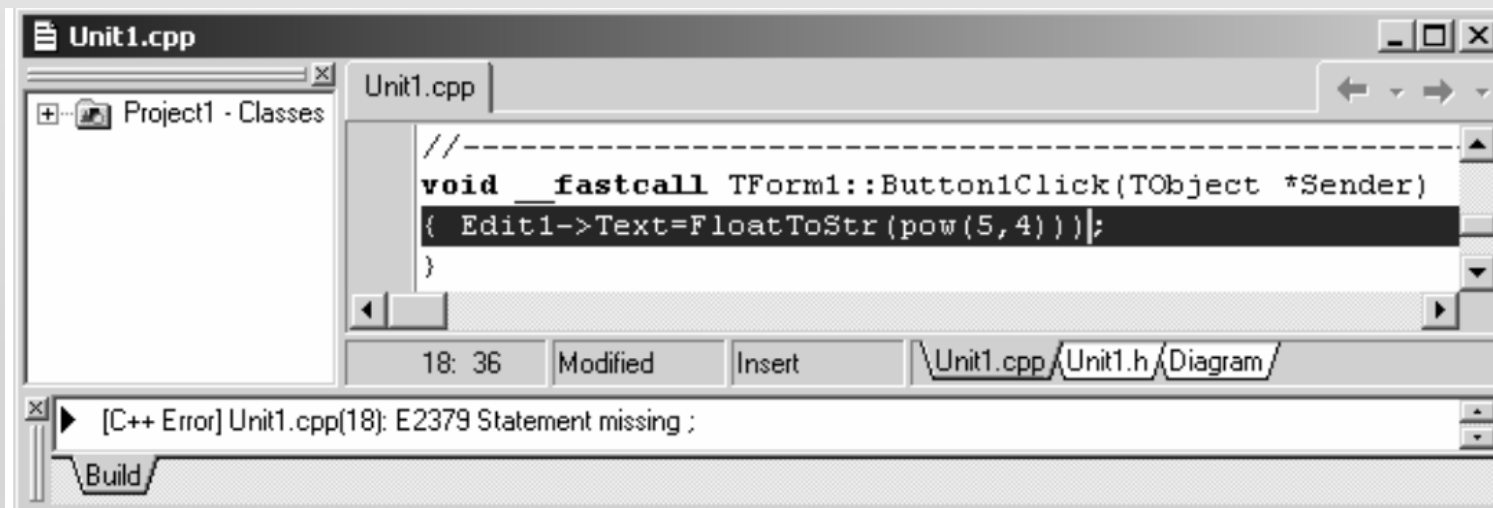
<i>Форма повідомлення про помилку</i>	<i>Характер помилки</i>
[C++ Error] Unit1.cpp(N): E2376 If statement missing (	Пропущено одну чи обидві дужки в умові оператора if чи умову написано з синтаксичними помилкам
[C++ Error] Unit1.cpp(N): E2378 For statement missing ;	Відсутня крапка з комою як розділювач складових частин оператора for
[C++ Error] Unit1.cpp(N): E2060 Illegal use of floating point	Неприпустима операція над дійсним числом

Приклад 9.1

```
[C++ Error] Unit1.cpp (N) : E2379 Statement missing;
```

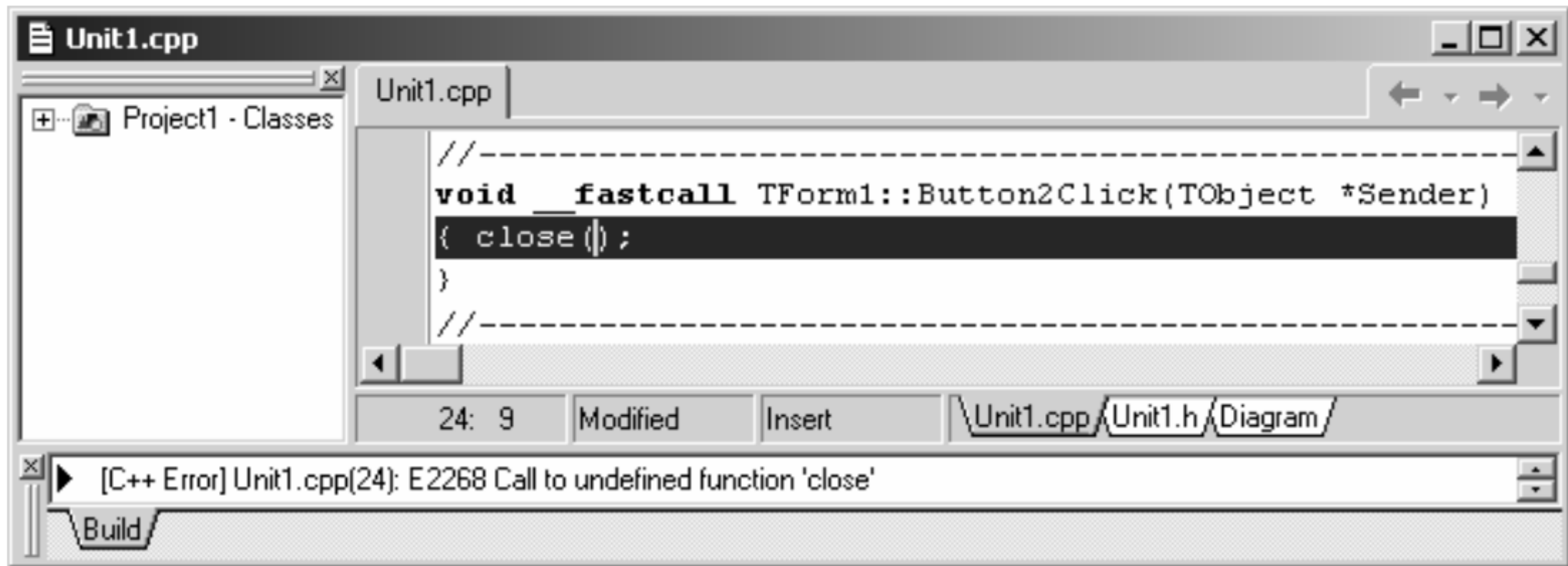
Помилка E2379: пропущено крапку з комою



Помилка E2379: зайва дужка у математичному виразі

Рисунок 9.1

[C++ Error] Unit1.cpp(N): E2268 Call to undefined function 'close'



Помилка E2268: ім'я функції написано неправильно

Рисунок 9.2

9.2 ПОПЕРЕДЖЕННЯ І ПІДКАЗКИ

[C++ Warning] Unit1.cpp(N): W8060 Possibly incorrect assignment

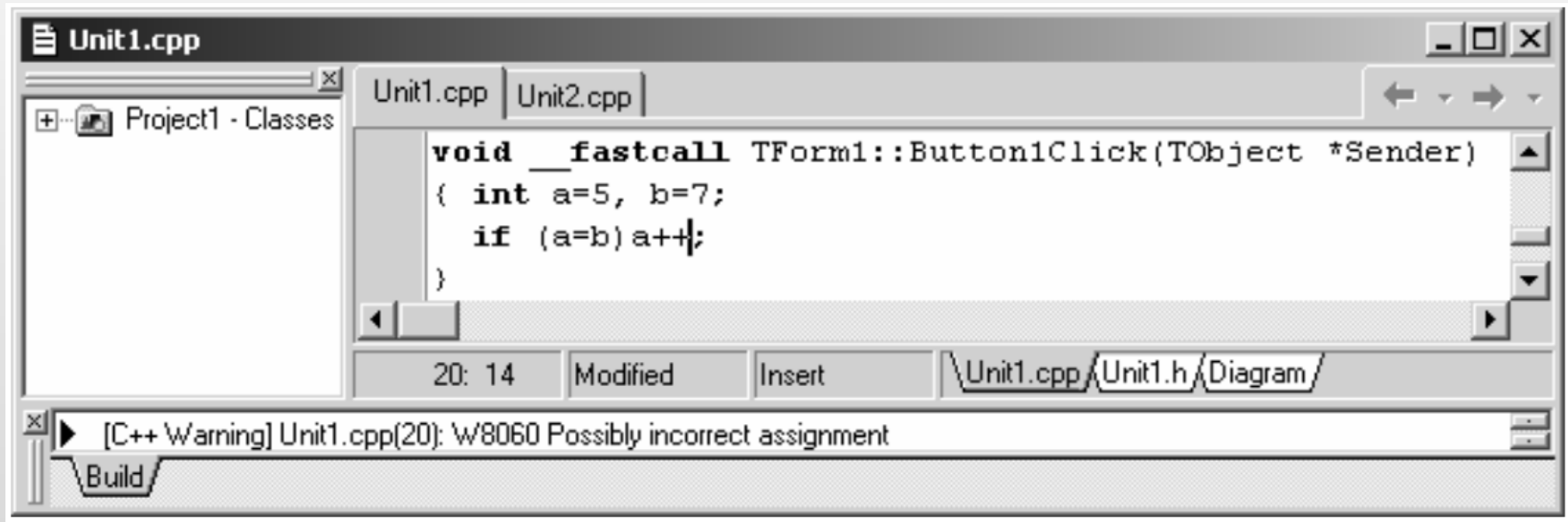


Рисунок 9.3

[C++ Warning] Unit1.cpp(N) W8004 'x' is assigned a value that is never used

[C++ Warning] Unit1.cpp(N) W8004 Possible use of x before definition

9.3 КОМПОНУВАННЯ

Причиною помилок при компонуванні є недоступність файлів бібліотек чи інших раніше відкомпільованих модулів.

Приклад 9.2: *[Linker Error] Unresolved external 'sum(int, int)' referenced from ...* (далі йде повне ім'я файла)

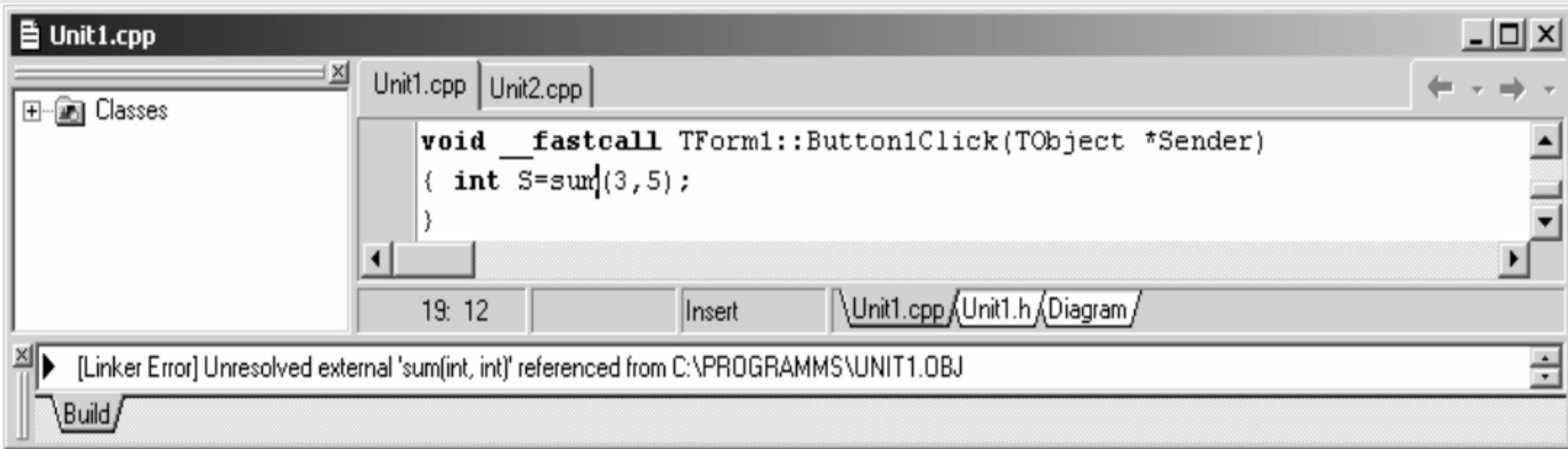


Рисунок 9.4

9.4 ПОМИЛКИ ЕТАПУ ВИКОНАННЯ

Помилки другого рівня (помилки етапу виконання – run time errors) або *виняткові ситуації (exceptions)* зазвичай пов'язані з некоректними початковими значеннями, помилками обраного алгоритму чи неправильною програмною реалізацією алгоритму.

Причини помилок етапу виконання:

- неправильне визначення вихідних даних;
- логічні помилки;
- накопичування похибок результатів обчислень.

1) У програмі міститься код

```
void __fastcall TForm1::Button1Click  
                                (TObject *Sender)  
{ float x=5.7, y;  
  Edit1->Text=x/y;  
}
```



Рисунок 9.5

2) Віднайти й виправити помилки у програмі з кодом

```
void __fastcall TForm1::Button1Click  
                                (TObject *Sender)  
{ int x=5;  
  while (x<15)  
    x-=2;  
  Edit1->Text=IntToStr(x);  
}
```



Рисунок 9.6

Приклад 9.3 Обчислити силу струму за формулою $I = U / R$, де U – напруга джерела (В), R – опір (Ом).

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    float u, r, i;
    u=StrToFloat(Edit1->Text);
    r=StrToFloat(Edit2->Text);
    i=u/r;
    Edit3->Text=FloatToStr(i);
}
```



Рисунок 9.7

```
u =StrToFloat (Edit1->Text);
```

Переповнення (нескінченний цикл) *stack overflow*.

```
void f(int n)
{ f(n+1);
}
void __fastcall TForm1::Button1Click(TObject *Sender)
{ f(5);
}
```



Рисунок 9.8

```
void f(int n)
{ if (n >= 10) return;
  f(n+1); }
```


Надто велике/мале число чи то неприпустиме значення для використання у якості аргументу функції спричиняють помилки:

```
pow: domain error  
log: SING error.
```

```
x=pow(-5, 0.3);
```



Рисунок 9.9

```
float x=0;  
Edit1->Text=FloatToStr(log(x));
```

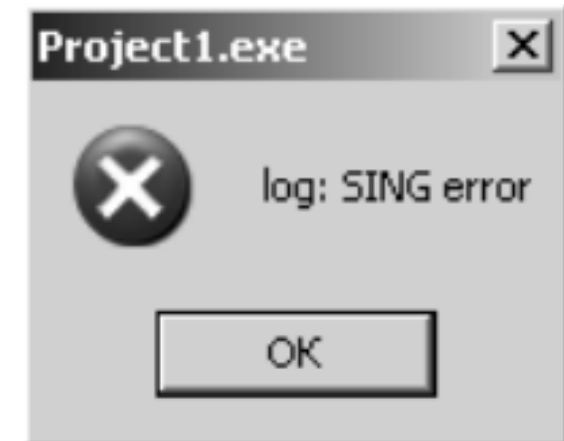


Рисунок 9.10

Ділення на нуль: *Division by zero.*

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    float x=0;
    for(int i=0; i<5; i++)
        x+=1./i;
    Edit1->Text=FloatToStr(x);
}
```



Рисунок 9.11

```
if (i != 0) x+=1./i;
```

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. ДСТУ ISO 5807:2016 Оброблення інформації. Символи та угоди щодо документації стосовно даних, програм та системних блок-схем, схем мережевих програм та схем системних ресурсів (ISO 5807:1985, IDT) [Електронний ресурс]. - Режим доступу: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=67202. - Назва з екрана.
2. Жуковський С.С. Об'єктно-орієнтоване програмування мовою С++: навч.-метод. посіб. для студ. напряму 6.040302 Інформатика / С.С. Жуковський, Т.А. Вакалюк. – Житомир: Вид-во ЖДУ, 2016. – 100 с.
3. Карпенко Н.В. Розробка програм на мові С у сучасних середовищах: навч.-метод. посіб. / Н.В. Карпенко. – Дніпро : ЛПРА, 2016. – 144 с.
4. Яворський Н. Б. Лабораторний практикум з дисципліни “Алгоритмізація та програмування”: навч. посіб. / Н. Б. Яворський, У. Б. Марікуца, М. І. Андрійчук, та ін. – Львів: Видавництво Львівської політехніки, 2018. – 191 с.
5. Алгоритмізація та програмування: Практикум [Електронний ресурс]: навч. посіб. для здобувачів ступеня бакалавра за спеціальністю 122 “Комп’ютерні науки” / Л. І. Кублій; КПІ ім. Ігоря Сікорського. - Київ: КПІ ім. Ігоря Сікорського, 2019. - 209 с.

6. С++. Алгоритмізація та програмування: підручник / О.Г. Трофименко, Ю.В. Прокоп, Н.І. Логінова, О.В. Задерейко. - 2-ге вид. перероб. і допов. – Одеса : Фенікс, 2019. – 477 с.
7. Комп'ютерні технології. Посібник для студентів спеціальності 172 «Телекомунікації та радіотехніка» факультету радіофізики електроніки та комп'ютерних систем / М.В. Кононов. – Київ: ФРЕКС Київського національного університету імені Тараса Шевченка, 2019. – 281 с.
8. Алгоритмізація та програмування: Практикум [Електронний ресурс]: навч. посіб. для здобувачів ступеня бакалавра за спеціальністю 122 “Комп'ютерні науки” / Л. І. Кублій; КПІ ім. Ігоря Сікорського. - Київ: КПІ ім. Ігоря Сікорського, 2019. - 209 с.
9. Обчислювальна техніка та програмування: Алгоритми та їх реалізація. Комп'ютерний практикум [Електронний ресурс]: навч. посіб. для студ. спеціальності 141 Електроенергетика, електротехніка та електромеханіка та 184 Гірництво / КПІ ім. Ігоря Сікорського; уклад. : Д. В. Філянін, В. П. Опришко, В. О. Броницький та ін. – Київ : КПІ ім. Ігоря Сікорського, 2022. – 95 с.
10. Основи алгоритмізації та програмування: навч. посіб. / Л.М. Бандоріна, Т.О. Климкович, К.О. Удачина. – Дніпро: УДУНТ, 2022. – 158 с.