МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЧЕРНІГІВСЬКИЙ КОЛЕГІУМ»
імені Т.Г. ШЕВЧЕНКА

МОВА ПРОГРАМУВАННЯ С++

МЕТОДИЧНІ ВКАЗІВКИ
ДО ПРАКТИЧНИХ ЗАНЯТЬ, ЛАБОРАТОРНИХ РОБІТ І САМОСТІЙНОЇ РОБОТИ З ДИСЦИПЛІНИ «ПРОГРАМУВАННЯ»

Чернігів НУЧК 2024

УДК 004.43
І73

Укладачі:
Бакалов Валерій Григорович, кандидат технічних наук, доцент кафедри хімії, технологій та фармації Національного університету «Чернігівський колегіум» імені Т.Г. Шевченка;
Горошко Юрий Васильович, доктор педагогичних наук, професор, завідувач кафедри інформатики і обчислювальної техніки Національного університету «Чернігівський колегіум» імені Т.Г. Шевченка.

Бакалов В.Г., Горошко Ю.В.

І73 Мова програмування С++. Методичні вказівки до практичних занять, лабораторних робіт і самостійної роботи з дисципліни “Програмування” / укладачі: Бакалов В.Г., Горошко Ю.В., Чернігів: НУЧК, 2024, 62 с.

Затверджено вченою радою природничо-математичного факультету Національного університету «Чернігівський колегіум» імені Т.Г. Шевченка, протокол №3 від 28.10.2024 р.

Рецензенти:
кандидат фізико-математичних наук, доцент кафедри математики Національного університету «Чернігівський колегіум» імені Т.Г. Шевченка, доцент **Болюнов Олексій Олександрович**
кандидат фізико-математичних наук, доцент кафедри інформаційних технологій та програмної інженерії Національного університету «Чернігівська політехніка», доцент **Акименко Андрій Миколайович**

Методичні рекомендації складено для здобувачів освіти, які навчаються за освітньо-професійною програмою Комп’ютерні науки і Середня освіта (інформатика) першого (бакалаврського) рівня вищої освіти. Основною метою методичних вказівок є формування у студентів знань базових конструкцій мови програмування С++ для опису алгоритмів розв’язуваних задач. Практичні заняття та лабораторні роботи запропоновані відповідно до змісту ОП Комп’ютерні науки і Середня освіта (інформатика), що дозволить забезпечити організацію ефективної індивідуальної роботи студентів.

ЗМІСТ

С.

ВСТУП 4

Тема 1 Алфавіт мови С++. Типи даних С++. Вирази та оператори. Оператори: вибору - if, switch; циклу - for, while, do-while, continue 5

Практичне заняття № 1. 13

Лабораторна робота №1 13

Тема 2 Функції. Рекурсивні функції. Покажчики та посилання..... 17

Практичне заняття № 2. 24

Лабораторна робота №2 24

Тема 3 Масиви. Функції введення-виводу. Рядки та операції з ними. Функції класу string. Побітові операції 27

Практичне заняття № 3. 38

Лабораторна робота №3 38

Тема 4 Сортування масивів 41

Практичне заняття № 4 48

Лабораторна робота №4 48

Тема 5 Бібліотека STL. Вектори, ітератори, списки. Функція сортування..... 49

Практичне заняття № 5. 53

Лабораторна робота №5 53

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 55

Додаток А. Програма сортування методом «бульбашки» 56

Додаток Б. Програма сортування методом вибору..... 58

Додаток В. Програма сортування методом вставки..... 59

Додаток Г. Програма сортування методом злиття..... 60

Додаток Д. Програма сортування методом Шелла..... 62

ВСТУП

Мова програмування С++ є найбільш поширеною мовою програмування упродовж кількох останніх десятиріч. На базі цієї мови “виросло” багато сучасних мов програмування. Цьому сприяли такі її властивості, як лаконічність, потужність, гнучкість, мобільність, можливість доступу до комірок пам’яті а також до бітів. Крім того, програми написані цією мовою виконуються швидше, ніж програми, написані іншими мовами.

Писати програми на мові С++ можна в інтегрованому середовищі розробки Visual Studia, Qt та CodeBlocks. Visual Studia працює під Windows, Qt та CodeBlocks працюють як під Windows так під Linux. Слід відмітити, що Qt є крос-платформенним фреймворком і має багато модулів для розробки програмного забезпечення, що полегшує розробку програм.

У вказівках наводяться короткі теоретичні відомості про основні складові мови і приклади їх застосування. Для кращого сприйняття команди мови С++ і програми виділені синім коліром.

Це дозволяє добре засвоїти студентами синтаксис мови програмування С++.

На практичних заняттях потрібно вивчити інформацію з коротких теоретичних відомостей методичних вказівок і бажано познайомитись з інформацією з інформаційних посилань. Після ознайомлення з теоретичними відомостями по темі заняття студенти набирають, відлагожують і запускають програми, які наведені у методичних вказівках. Для кращого сприйняття студентами теме викладач надає завдання на корекцію програм.

На лабораторних роботах викладач дає кожному студенту варіант завдання по темі заняття. По завданню студент повинен скласти програму його вирішення. Якщо студент не встигає виконати завдання на занятті, то виконує його самостійно у вільній від занять час.

Тема 1

Алфавіт мови С++. Типи даних С++. Вирази та оператори.
Оператори: вибору - if, switch; циклу - for, while, do-while, continue

Короткі теоретичні відомості
Алфавіт мови С++ включає:

- великі (A-Z) і малі (a-z) літери латинського алфавіту та символ підкреслення (_);
- арабські цифри від 0 до 9;
- знаки арифметичних дій +, -, *, /, %, ++, --;
- знаки побітових операцій <>, &, |, ~, ^;
- знаки відношень <=, ==, !=, >, >=;
- знаки логічних операцій &&, ||, !;
- розділові знаки , ; : пропуск;
- спеціальні знаки ., =, ->, ?, \, $, #, ', ";
- символи дужок (,), [,], {, }.

Ідентифікатором, тобто ім'ям програмного об'єкта, називається будь-яка послідовність літер латинського алфавіту, цифр і символу підкреслення, за умови, що першою стоїть літера або символ підкреслення, а не цифра.

Існує два різновиди ідентифікаторів:

- стандартні, наприклад імена всіх вбудованих у мову функцій;
- користувальницькі.

Характерно, що мова С++ чуттєва до регістру літер, тому компілятор розпізнає великі і малі літери латинського алфавіту як різні символи.

Ідентифікатори можуть мати будь-яку довжину, але значимими є не більше 31 символу (в деяких компіляторах це обмеження не більше 8 символів).

Ідентифікатори не повинні збігатися з ключовими словами мови С++.

Ключові слова мови С++ поділяються на такі основні групи:

- специфікатори типів — char, int, long, typedef, short, float, double, enum, struct, union, signed, unsigned, void;
- кваліфікатори типів — const і volatile;
- класи пам'яті — auto, extern, register, static;
- для побудови операторів — for, while, do, if, else, switch, case, continue, goto, break, return, default, sizeof.

Щоб ідентифікатор можна було використовувати в програмі, він повинен бути попередньо описаний, при цьому для нього резервується деяка область пам'яті, розмір якої залежить від конкретного типу ідентифікатора (наприклад символьний, цілий і т.д.).

Типи даних C++ наведені в таблиці.

Тип	Назва	Розмір, байт	Діапазон	Приклади можливих значень	Типи чисел
char	символьний (знаковий)	1	від -128 до 127	'a', '\n', '4'	цілі
unsigned char	беззнаковий символьний	1	від 0 до 255	3, 127	
short	короткий цілий	2	від -32768 до 32767	-74, 30001	
unsigned short	беззнаковий короткий	2	від 0 до 65535	0, 11, 45431	
int	цілий (знаковий)	4	від -2147483648 до 2147483647	-567, 127005	
unsigned int	беззнаковий цілий	4	від 0 до 4294967295	5, 6543	
long	цілий (знаковий)	4	від -2147483648 до 2147483647	-5678, 9814	
float	дійсний одинарної точності	4	від 1.210^{-38} до $3.4{\cdot}10^{38}$	78.3 -0.25, 0.005, $3.7{\cdot}10^{-8}$, $4{\cdot}10^{12}$	дійсні
double	дійсний подвійної точності	8	від $2.2{\cdot}10^{-308}$ до $1.8{\cdot}10^{308}$		
long double	довгий дійсний	10	від $3.4{\cdot}10^{-4932}$ до $3.4{\cdot}10^{4932}$		
bool	логічний	1	true або false	false(0) true(>=1)	
enum	перерахований	2 або 4			
void	порожній без значення				

Вирази та оператори

Для записування виразів використовують відповідні операції. В C++ існують унарні, бінарні та тернарні операції, залежно від кількості операндів. Більшість операцій є бінарними, тобто мають два операнди, один з яких розміщується перед знаком операції, а другий – після. Наприклад, операція додавання “+” має два операнди: x та y і обчислює їхню суму. Унарні операції - це ті, які мають лише один операнд. Наприклад, вираз x++ означає застосування до операнда x операції унарного інкремента (збільшення на 1) (опис і приклади будуть трошки нижче). Тернарну умовну операцію (?:), яка має три операнди, буде розглянуто пізніше. Вирази можуть бути арифметичними, логічними та порівняння.

Арифметичні вирази

Арифметичні вирази − це сукупність числових констант, змінних і функцій, пов’язаних знаками арифметичних операцій і круглих дужок.

До базових арифметичних операціям можна віднести операції складання (+), віднімання (-), множення (*), поділу (/), взяття за модулем (%), тобто обчислюється залишок від поділу одного операнда на інший.

Для ефективного використання значення, що повертається операціями, призначений оператор присвоєння (=) та його модифікації: складання з присвоєнням (+=), віднімання

з присвоєнням (-=), множення з присвоєнням (*=), поділ з присвоєнням (/=), взяття за модулем з присвоєнням (%=) та цілий ряд інших.

Нижче наведено лістинг програми з операціями які розглядалися.

```
#include <iostream>  /* Заголовоний файл з класами, функціями
                        для і включає стандартні бібліотеки введення і виведення */
using namespace std;  //Простір імен std для використання
                      // виведення (cout) та введення (cin)
int main()
{ int x=0, y=7, c= 17;
  char z='\n';            // перехід на новий рядок
  x=y;                    // x=7
  cout << x << z;
  x=2*b+c;                // x=31
  cout << x << endl;      // друк значення x перехід на новий рядок
  cout << x%2 << z;       // 31%2=1
  x+=y;                   // x=x+b=31+7=38
  cout << x << endl;
  x-=11;                  // 31-11=27
  cout << x << endl;
  x%=4;                   // 20%4=3
  cout << x << z;
  return 0;
}
```

Розрахунок покаже такі значення 7 31 1 38 27 3 виведених в стовпчик.

Оператори інкременту та декременту.

Ефективний засіб збільшення та зменшення значення операнда на одиницю - унарні оператори інкременту (++) та декременту (--).

По відношенню до операнду цей вид операторів може бути префіксним та постфіксним. Префіксний оператор застосовується до операнда перед його використанням, постфіксний оператор застосовується до операнда після його використання.

Приклад. Якщо int *x*=2, *y*=3;

Запишемо *x=y++*;

Тоді значення *y* спочатку присвоюється *x* (*x*=3), а тільки потім *y* збільшиться на одиницю (*y*=4).

Якщо записати *x=++y*; то спочатку *y* збільшиться на одиницю (*y*=4) і тільки потім присвоюється *x* (*x*=4).

Логічні операції.

У мові С++ існують наступні логічні операції: логічне множення кон`юнкція (&& програмісти її називають "І"), логічне додавання диз`юнкція (|| - "АБО"), логічне заперечення (!- "НІ"). Значенням логічного виразу є хибність (false або нуль) чи істина (true або одиниця).

Таблиця істинності логічних операцій.

Операнд 1	Операнд 2	Кон'юнкція &&	Диз'юнкція ||	Заперечення ! (операнда 1)
хибність	хибність	хибність	хибність	істина
хибність	істина	хибність	істина	істина
істина	хибність	хибність	істина	хибність
істина	істина	істина	істина	хибність

Приклад.

(x > y) && (a > b)

Цей вираз істинний, якщо істинні як вираз (x > y), так і вираз (a > b). Якщо (x > y) хибність, то весь вираз має бути помилковим і використання значення виразу (a > b) ігнорується.

(x > y) || (a > b)

Цей вираз істинний, якщо істинний або вираз (x > y), або вираз (a > b), або якщо істинні обидва вирази. Якщо (x > y) істинний, весь вираз істинний, так що в цьому випадку не обчислюється істинність виразу (a > b).

Унарна операція ! ("НІ") змінює на протилежне початкове логічне значення. Якщо (x > y) хибність, то !(x > y) істина.

Оператори порівняння

Мова С++ має шість операторів порівняння, які наведені в таблиці.

Оператор	Символ	Приклад	Операція
Більше	>	x > y	true, якщо x більше y, в іншому випадку - false
Менше	<	x < y	true, якщо x менше y, в іншому випадку - false
Більше або дорівнює	>=	x >= y	true, якщо x більше або дорівнює y, в іншому випадку - false
Менше або дорівнює	<=	x <= y	true, якщо x менше або дорівнює y, в іншому випадку - false
Дорівнює	==	x == y	true, якщо x дорівнює y, в іншому випадку - false
Не дорівнює	!=	x != y	true, якщо x не дорівнює y, в іншому випадку - false

Оператори вибору if, if-else, switch case**.**

До операторів вибору відносять оператор умовного переходу if та оператор-перемикач switch. Оператор умовного переходу if використовується для розгалуження процесу обчислень на два напрямки і має такий формат запису:

```
if (вираз) оператор 1
    else оператор 2;
```

де вираз - логічне значення (true - «істина» або false - «неправда»).

Реалізується оператор *if* таким чином: спочатку обчислюється вираз і, якщо значення виразу не дорівнює нулю («істина»), виконується оператор 1, в протилежному випадку - оператор 2 і далі управління передається оператору, що є наступним за

умовним оператором if. Слід відмітити, що оператор має ще скорочену форму в якій відсутнє else. В такому випадку в залежності від виразу («істина») виконується оператор 1 і далі виконується наступний оператор, в протилежному випадку оператор 1 не виконується.

Приклад використання вибору if

```
#include <iostream>
using namespace std;
int main()
{    int x,y;
     cin>>x;
     if(x>2) {  y=x*x;   cout << " y=" << y << endl;  } // x>2 – істина друк y=x*x
     else
     {if(x<2) { y=x+10;  cout << " y=" << y << endl; }// x<2 – істина друк y=x+10
     else  { y=x*30;  cout << " y=" << y << endl; } // залишилось x=2 – друк y=x*30
     }
     return 0;
}
```

Щоб зменшити написання коду оператора вибору if в мові програмування С++ використовують тернарний оператор який має такий вид

```
#include <iostream> // тернарний оператор
using namespace std;
int main()
{    int x,y;
     cin >> x;
     (x>2)?(cout << " y=" << x*x) // ? – заміняє if, якщо істина то друкується y=x*x
     :(x<2)?              //: - заміняє else і знову йде вибір ? - x<2
     (cout << " y=" << x+10) // якщо x<2 – істина, то друкується y=x+10
     :(cout << " y=" << x*30); // : - залишилось по вибору x=2 і тоді друкується y=x*30
     return 0;
}
```

Оператор-перемикач switch називають оператором множинного розгалуження. Він використовується для вибору не одного а багатьох варіантів рішення і має таку форму запису

```
#include <iostream>
using namespace std;
int main()
{    int a,y;
     cin >> a>>y;  //Введення значення a, y
     switch (a)
     {case 1,3: cout << y+10 << endl; break;
     case 2: cout << y*y << endl; break;
     case 4,5: cout << y/2 << endl; break;
```

```
    default:  cout << " Error  from 1-5" << endl;
    return 0;
}
```

Параметр вибору (a) може приймати ціле або символьне значення.

При виконанні цього оператора спочатку обчислюється значення виразу (a), потім це значення порівнюється (послідовно зверху донизу) зі значеннями константних виразів після case. У випадку збігу значень (a) і одного з цих константних виразів case виконується оператор який знаходиться після нього. Якщо на прикінці знаходиться оператор break то далі виконується оператори після switch. Відповідно, якщо нема break, то будуть послідовно виконуватись оператори case до появи break. Якщо значення виразу (a) не збігається з жодним із значень константних виразів після case, то виконується оператори після default і здійснюється вихід з оператора switch. Коли в операторі switch відсутній оператор default (він необов'язковий) і значення (a) не збігається з жодним із значень константних виразів, то відбувається вихід з оператора switch.

Оператори циклу for, while, do-while.

В програмуванні дуже часто потрібно щоб одна і таж послідовність команд виконувалась декілька разів. Для цього у мові C++ застосовується поняття циклічного процесу або циклу. На кожному кроці циклу виконанується фрагмент команд який незначно відрізняється від попереднього його виконання. Цикл може виконуватися впродовж заздалегідь заданого числа кроків, а може завершуватися при виконанні деякої умови.

У мові C++ існує три види управляючих операторів циклу:

- цикл for;
- цикл while з передумовою;
- цикл do … while з постумовою.

Оператор for.

Синтаксис

for (вираз 1; вираз 2; вираз 3) тіло циклу;

У виразі 1 присвоюється початкове значення управляючій змінній циклу. Ця змінна є лічильником, який керує роботою циклу. Вираз 2 є умовним виразом в якому перевіряється значення змінної циклу і якщо умовний вираз – істина, то цикл продовжується, якщо – ні (хибність) то закінчується виконання циклу. Вираз 3 змінює значення змінної циклу для кожної ітерації (наприклад кожен раз додає одиницю). Слід відмітити, що всі вирази (1-3) можуть мати декілька управляючих змінних циклу або не мати деяких.

Приклад використання оператору циклу for

```
#include <iostream>
using namespace std;
int main()
{    int sum=0;
     for(int i=1; i<=10; i++)
```

```
        { sum +=i;
        cout << "  i= " << i << endl; // друк поточного значення змінної циклу
        cout << "  sum =" << sum << endl; // друк поточної суми
        }
        cout << "sum = " << sum << endl; // друк суми цифр до 10
    return 0;
    }
```

Відповідь sum =55.

Оператор while з передумовою

Синтаксис

while (вираз) тіло циклу;

Тіло циклу виконується до тих пір, поки значення виразу буде – істина (true). Про зміну значення управляючої циклу повинен потурбуватись програміст або у виразі, або у тілі циклу.

Приклад використання оператору циклі while при розрахунку суми цифр до 10.

```
#include <iostream>
using namespace std;
int main()
{    int i,j;
     i=0; sum=0;
     while (i++<10)  // лічильник циклу змінюється у виразі
     { //i++;             // можна лічильник змінювати у тілі
     sum+=i;
     cout << "sum= "<< sum << endl; // друк поточної суми
     }
     cout << "sum= "<< sum << endl; // друк кінцевої суми
return 0;
}
```

Відповідь sum =55.

Оператор do … while з постумовою

Синтаксис

do тіло циклу while (вираз);

Тіло циклу виконується до тих пір, поки вираз має значення – істина (true). На відміну від циклу while, в якому перевірка умови закінчення циклу робиться до виконання тіла циклу, в циклі do така перевірка має місце після виконання тіла циклу. Таким чином тіло циклу do буде виконано хоч би один раз, навіть якщо вираз має значення хибне (false) із самого початку.

Приклад використання оператору циклі do… while при розрахунку суми цифр до 10.

```
#include <iostream>
using namespace std;
int main()
{    int i,sum;
     i=1; sum=0;
```

```
    do
    {   i++;
    sum+=i;
    }
    while(i<10); // можна лічильник змінювати також у виразі
    cout << "  sum=" << sum << endl;
    return 0;
}
```

Відповідь sum =55.

Оператори переривання виконання break, goto, continue.

Для завчасного переривання повторювань операторів циклу будь-якого типу в тілі циклу можна застосовувати оператор break. Він передає керування на наступний оператор. Усередині вкладених операторів, оператор break завершує лише оператори do-while, for, switch чи while.

Щоб передати керування за межі вкладеної структури, слід використовувати оператори return (вихід з поточної функції) та goto (безумовний перехід). Для переходу до наступної ітерації циклу можна застосовувати оператор continue. Цей оператор, подібно до оператора break, використовується лише всередині операторів циклу, але, на відміну від останнього, виконування програми після оператора continue продовжується не з оператора, який іде за перерваним оператором, а з початку перерваного оператора, тобто оператор continue є тотожний переходові до наступної чергової ітерації. Оператор continue, так само як і оператор break, перериває найглибинний з циклів.

До операторів переривання виконання відносять оператор безумовного переходу goto та оператори виходу з циклу.

Синтаксис оператора безумовного переходу goto має вигляд:

```
goto метка;
```

Слід відмітити, що мітка це ім’я, після якого ставляться дві крапки. Виконання програми переходить на оператори де стоїть мітка.

Слід зауважити, що використання оператора безумовного переходу goto вважається негарним стилем програмування оскільки він ускладнює розуміння логіки програми.

Програма використання continue.

```
#include <iostream>
using namespace std;
int main()
{    for(int a = 1, b = 0; a < 10; b += a, a++)
     { if (b % 2) continue; // визначення парних сум a і b
          cout << " a= " << a<< "  b= " << b<< endl;
     }
     return 0;
}
```

Результати

a= 1 b= 0

a= 4 b= 6
a= 5 b= 10
a= 8 b= 28
a= 9 b= 36

Таж сама програма з використання break.

```
#include <iostream>
using namespace std;
int main()
{    for(int a = 1, b = 0; a < 10; b += a, a++)
        { if (b % 2) break; // виходить із циклу при першому виконанні
            cout << " a= " << a<< "  b= " << b<< endl;
        }
    return 0;
}
```

Результати в рядок a= 1 b= 0

Практичне заняття № 1.

Після ознайомлення з теоретичними відомостями по темі заняття студенти набирають, відлагожують і запускають програми, які наведені у методичних вказівках. Для кращого сприйняття студентами теме викладач надає завдання на корекцію програм.

Лабораторна робота №1

При виконанні лабораторної роботи студент самостійно складає програму на вірішення задачі, номер варіанту для кожного розділу вказує викладач.

Арифметичні операції

Необхідно скласти програму і отримати результати.

1) x = (7 + 6) % 5 / 2;
2) x = -3 * 4 % -6 / 5;
3) int x = 2, y = 1, z = 3; x *=8 / 2 + y++ + ++z;
4) m = 2; x = 2; x *= y = z = 4/m;
5) int x = 2, y, z; а) x *= 3 + 2; б) x += y = z = 4; в) x = y == z; г) x = x == (y = z);
6) z = 1; x = 2; y = 3; z = 3 * x++ - --y;
7) Дано двозначне число. Необхідно виділити в ньому одиниці і десятки.
8) Дано тризначне число. Визначити кількість сотень, десятків і одиниць в ньому. Написати число, отримане при прочитанні початкового числа з права на ліво.
9) Дано двозначне число. Знайти число, отримане переставленням першої і останньої цифр.
10) Дано тризначне число. У ньому закреслили першу зліва цифру і приписали її в кінець вихідного числа. Знайти число, що вийшло.
11) Дано тризначне число. У ньому закреслили останню цифру і приписали її в початок вихідного числа. Знайти число, що вийшло.

12) Дано тризначне число. З ним необхідно виконати наступні дії: а) відняти останню цифру; б) результат розділити на 10; в) до результату зліва приписати останню цифру вихідного числа.

13) Дано тризначне число. Знайти число, отримане шляхом перестановки першої та другої цифри вихідного числа.

Використання логічних операцій та операторів

Необхідно скласти програму і отримати результат.

1) x = 2, y = 1, z = 0; а) x = x && y || z; б) x || ! y && z; в) x = y = 1; z = x++ - 1; г) z += -x++ + ++y;

2) Визначити значення логічного виразу, якщо A = 1, B = 0, C = 0: а) A || B && !C; б) !A && !B; в) !(A && B) || B; г) A && !B || C; д) A && (!B || C); е) A && (!(B || C)).

3) Визначити значення логічного виразу, якщо x = 0, y = 1, z = 0:
а) x та не (z або y) або не z; б) (не x або не y) та (x або y); в) x та y або x та z або не z.

4) Визначити значення логічного величини при всіх можливих значеннях логічних величин x і y (побудувати таблицю істинності):
а) !x && !y; б) y || !(x && !y); в) F = !(x || !y) || !y.

5) Записати логічний вираз, який має значення true, тільки при виконанні зазначених умов: а) x > 2 та y > 2; б) x > 1 або y > -2; в) невірно, що x > 2;
г) невірно, що x > 0 та x < 5; д) 10 < x <= 20; е) x < 5 або x > 10; ж) 0 < y <= y та x > 5.

6) Записати логічний вираз, який має значення true, якщо: а) кожне з чисел x та y більше 100; б) тільки одне з x та y парне; в) хоча б одне з x та y більше 0; г) кожне з чисел x, y, z кратне 3; д) тільки одне з x, y, z менше 0; е) x кратне 2 або 3; ж) x не кратне 3 та закінчується на 0.

Використання умовних операцій

Необхідно скласти програму і отримати результат.

1) Нехай дано значення змінної a. Написати умовний оператор, який обчислює наступні змінні f:

а) $f = \begin{cases} a^2, \text{якщо} -2 \le a \le 2 \\ 4 \quad \text{в інших випадках} \end{cases}$

б) $f = \begin{cases} a^2 + 4a + 5, \text{ якщо } a \le 2 \\ 1/(a\,2 + 4a + 5), \text{в інших випадках} \end{cases}$

в) $f = \begin{cases} 0, \text{ якщо } a \le 0 \\ a, \text{якщо } 0 \le a \le 1 \\ a^4, \text{в інших випадках} \end{cases}$

2) Нехай дані значення змінних a, b, c. Написати умовний оператор, який виконує наступні дії:

– якщо a <= b <= c, то всі числа записати їх квадратами;

– якщо a > b > c, то кожне замінити максимальним з трьох;

– в інших випадках – змінити знак кожного числа

3) Нехай дані значення змінних x, y. Написати умовний оператор, який виконує наступні дії:

– якщо x < 0 та y < 0, то кожне замінити його модулем;

– якщо негативне тільки одне, то обидва значення збільшити на 0,5;

– якщо обидва значення не негативні і жодне з них не належить відрізку [0,5 ; 2,0], то обидва значення зменшити в 10 разів;

– в інших випадках – x, y залишити без зміни.

4) Написати програму обчислення площі кільця. Програма повинна передбачати введення даних, перевіряти правильність вихідних даних (повідомляти про помилку) а також розраховувати площу кільця.

5) Написати програму вирішення квадратного рівняння. Програма повинна перевіряти правильність вихідних даних і повідомляти про це.

6) Написати програму, яка виводить приклад на множення двох однозначних чисел, запитує відповідь користувача, перевіряє його і виводить повідомлення «Правильно!» або «Ви помилилися» і правильний результат.

7) Написати програму, яка запитує у користувача номер місяця і потім виводить відповідну назву часу року. Якщо користувач введе неприпустиме число, програма повинна вивести повідомлення «Помилка даних».

8) Написати програму, яка обчислює дату наступного дня (задається день, місяць, рік. Врахувати кількість днів різних місяців, а також чи рік високосний).

9) Програма перекладу літерних оцінок в цифрові.

10) Програма найпростішого калькулятору (введення даних і знаку операції – складання, віднімання, множення, ділення).

Використання операторів циклу

1) Обчислити суму квадратів натурального ряду чисел від 1 до n, використовуючи оператор циклу for.

2) Розглянути приклад з попередньої роботи. Написати програму визначення числа n, при якому сума чисел S попереднього ряду не перевищить величину K, яку введено з клавіатури.

3) Написати програму, яка виводить таблицю квадратів перших п'яти цілих позитивних чисел.

4) Написати програму, яка перетворює введене користувачем десяткове число в двійкове.

5) Написати програму, яка визначає максимальне число з введеної з клавіатури послідовності позитивних чисел (довжина послідовності не обмежена, а нуль є завершенням введення).

6) Написати програму, яка перевіряє, чи є введене користувачем ціле число простим (простим називається число, яке ділиться тільки на одиницю і на саме себе).

7) Написати програму, яка обчислює найбільший спільний дільник двох цілих чисел.

8) Роздрукувати літери латинського алфавіту відповідно до наведеного нижче виду:

A B C D E
B C D E F
C D E F G

D E F G H
E F G H I

9) Написати програму, яка обчислює суму чисел від 1 до введеного числа. Кожне наступне число зростає по відношенню до попереднього на 3.

10) Написати програму, яка із довільних 5 зростаючих чисел буде обирати найближче до введеного з клавіатури шостого числа.

Тема 2

Функції. Рекурсивні функції. Покажчики та посилання.

Короткі теоретичні відомості

Функції - це іменована сукупність операторів, призначених для виконання певного завдання.

Часто програми переривають виконання одних функцій заради виконання інших. Іноді, коли програма виконує код, вона може зіткнутися з викликом функції. Виклик функції – це вираз, який вказує процесору перервати виконання поточної функції і приступити до виконання іншої функції. Процесор "залишає закладку" в поточній точці виконання, а потім виконує функцію, що викликається. Коли виконання функції, що викликається завершено, процесор повертається до закладки і відновлює виконання перерваної функції.

Програма мовою С++ містить одну або декілька функцій, кожна з яких повинна бути оголошена та визначена до її першого використання. Оголошення функції (прототип, заголовок) задає ім'я функції, тип значення, що повертає функція (якщо воно є), а також імена та типи аргументів, які можуть передаватися як у функцію, так і з неї.

Усі функції мають однакову структуру визначення у вигляді:

```
[тип результату] ім'я функції  ([список формальних аргументів])
{// тіло функції
      опис даних;
      оператори;
[return]  [вираз];
}
```

Слід відмітити, що тип результату - будь-який базовий (наприклад int, char) або раніше описаний тип значення, що повертається функцією (необов'язковий параметр). За відсутності цього параметра тип результату за замовчуванням буде цілий (int). Він також може бути описаний ключовим словом (void), тоді функція не повертає ніякого значення. Якщо результат повертається функцією, то в тілі функції є необхідним оператор return вираз, де вираз формує значення, що співпадає з типом результату.

Ім'я функції - ідентифікатор функції, за яким завжди знаходиться пара круглих дужок«**( )**», де записуються формальні аргументи. Фактично ім'я функції - це особливий вид покажчика на функцію, його значенням є адреса початку входу у функцію.

Список формальних аргументів - визначає кількість, тип і порядок проходження переданих у функцію вхідних аргументів, які розділяються комою. У випадку, коли параметри відсутні, дужки залишаються порожніми або містять ключове слово (void). Формальні параметри функції локалізовані в ній і недоступні для будь-яких інших функцій.

Список формальних аргументів має такий вигляд:

([const] тип 1 [параметр 1], [const] тип 2 [параметр 2], …).

У списку формальних аргументів для кожного параметра треба вказати його тип (не можна групувати параметри одного типу, вказавши їх тип один раз).

Тіло функції може складатися з описів змінних і операторів. Змінні, що використовуються при виконанні функції, можуть бути глобальні і локальні. Змінні, що описані (визначені) за межами функції, називають глобальними. За допомогою глобальних параметрів можна передавати дані у функцію, не включаючи ці змінні до складу формальних параметрів. У тілі функції їх можна змінювати і потім отримані значення передавати в інші функції.

Змінні, що описані у тілі функції, називаються локальними або автоматичними. Вони існують тільки під час роботи функції, а після реалізації функції система видаляє локальні змінні і звільняє пам'ять. Тобто, між викликами функції вміст локальних змінних знищується, тому ініціювання локальних змінних треба робити щоразу під час виклику функції. За необхідності збереження цих значень, їх треба описати як статичні за допомогою службового слова static, наприклад:

```
static int  a, b;
```

Статична змінна схожа на глобальну, але діє тільки у тій функції, в якій вона оголошена.

На початку програми можна не описувати всю функцію, а записати тільки прототип. Запис прототипу може містити тільки перелік типів формальних параметрів без імен, а наприкінці прототипу завжди ставиться символ (;), тоді як у описі (визначенні) функції цей символ після заголовка не присутній.

Механізм передачі параметрів є основним засобом обміну інформацією між функцією, що викликається, та функцією, яка викликає. Параметри, котрі зазначаються у заголовку опису функції, як відомо, називаються формальними, а параметри, які записані у операторах виклику функції - фактичними.

При виконанні функції, всі її параметри створюються як локальні змінні, а значення кожного з аргументів копіюється в відповідний параметр (локальну змінну). Цей процес називається передачею за значенням. Таким чином будь-які зміни з аргументом, зроблені всередині функції, не відображаються на ньому.

Однак механізм передачі масивів у функцію інший. Це буде розглянуто в подальшому. На даному етапі необхідно пам'ятати, що на відміну від передачі у функцію параметра за значенням, будь-які зміни з масивом, зроблені всередині функції, відобразяться на масиві, який переданий в функцію як аргумент.

Приклад використання функції з передачею значення

```
#include <iostream>
using namespace std;
void fu(int a)
{  a++;
   cout << " Function  a= " << a << endl;
}
int main()
{  int a=1;
```

```
    fu(a);
    cout << " main a=" << a << endl;
    return 0;
}
```

Результат

Function a= 2

 main a=1

Як видно із програми main йде виклик функції і їй передається значення a=1. У самій функції створюється локальна змінна a (це дублікат a) і вона змінюється на одиницю. Це значення дійсне тільки у функції і не передається у основну програму main, де значення a не змінилось і залишилось a=1.

Приклад використання функції прототипа

```
#include <iostream>
using namespace std;
void fu(); // прототип функції
int main()
{  for(int i=0;i<6;i++)
    fu(); // Буде друкуватись в циклі Print fu()
    return 0;
}
void fu() // функція нічого не повертає
    {  cout << "  Print fu()" << endl;
    }
```

Прототип (сигнатура) функції fu() наведена раніше головної функції main, а сама функція розташована після main. В циклі 6 разів буде визиватись функція fu() і кожен раз буде друкувати Print fu().

Приклад функції, що вбудовується (ознака - перед описом функції вказують inline). Така функція використовується тоді, коли вона дуже проста і краще (по часу виконання програми) її на стадії компіляції вбудувати в тіло програми

```
#include <iostream>
using namespace std;
inline int Fu(int x, int y)  // Функція, що вбудовується
{  return x+y;}
int main()
{  int x,y,z,z1;
    x=5; y=10;
    z=Fu(x,y);
    z1=Fu(x*x,y-2);
    cout << "  z= " << z <<   "  z1= " << z1 <<  endl;
return 0;
}
```

Результат z=15 z1=33.

Перевантаження дозволяє мати кілька однойменних функцій, що виконують схожі операції над аргументами. Всі функції мають однакове ім'я, но різний тип результату і список формальних аргументів. В залежності від того який тип формальних аргументів і скільки їх у функції, що викликає, буде відповідний перехід до потрібної функції.

```
#include <iostream>
using namespace std;
int Fu(int a, int b, int c)
      {    return a+b+c;  }
           int Fu(int a, int b)
      {    return a+b; }
           double Fu(double a, double b)
      {    return a+b; }
      char Fu(char a, char b)
      {    return a+b-60; }
int main()
{     cout << Fu(3,6) << endl<< endl;
      cout << Fu(3,6,9) << endl<< endl;
      cout << Fu('A','B') << endl<< endl;
      cout << Fu(3.5,6.8) << endl<< endl;
 return 0;
}
```

Результат в рядок 9 18 G 10.3.

Аргументи зі значеннями за замовчуванням Наявність аргументів за замовчуванням дає можливість розширити множину підходящих функцій. Приклад виклику функції з параметрами по замовчанню

```
#include <iostream>
using namespace std;
void Fu(int x=20,int n=6, int a=3)   // Параметри по замовчанню
{  for(int i=0;i<n;i++)
   cout << char(x+40+i-a) << endl;
}
int main()
{  Fu(28,4);   // 20 передається x
   return 0;
}
```

Результат в рядок A B C D.

Слід відмітити, що у функцію передаються значення 28 і 4, а значення a=3 береться із самої функції. При значенні лічильника i=0 будемо мати char (x+40+i-a) =char(28+40+0-3) = char(65). Згідно таблиці кодування ASCII значення 65 це буква A. Значення лічильника зростає і відповідно будуть друкуватися наступні букви.

Шаблони функцій робляться для того, щоб одна і таж функція використовувалась для різних типів аргументів. Наприклад шаблонна функція для швидкого сортування може виконувати сортування як з цілими значеннями аргументів так і з char

аргументами. Оголошення й визначення шаблона функції завжди починається із ключового слова template. Приклад програми з шаблоном.

```
#include <iostream>
using namespace std;
template <typename T1,typename T2>   //template <class T1,class T2>
T1 Fu(T1 a,T2 b)
{    return a-b; }
int main()
{    cout << Fu(7,3.2) << endl<< endl;
     cout << Fu(12.3,67.2) << endl<< endl;
     cout << Fu('z',40) << endl<< endl;
     cout << Fu(100,'A') << endl<< endl;
return 0;
}
```

При першому зверненні перший аргумент (7) int і відповідно функція поверне значення типу T1 яке буде int. У другому зверненні до функції перший аргумент дійсне (12.3) float і функція поверне значення float. У третьому – char і функція поверне значення char. У четвертому – повертатися буде int значення.

Сама функція виконує віднімання від значення першого аргументу значення другого. При четвертому звертанні від 100 буде відніматися 65 (символ A в кодуванні ASCII має значення 65) і отримаємо 35.

Результат виконання програми в рядок буде 3 -54.9 R 35

Рекурсивні функції

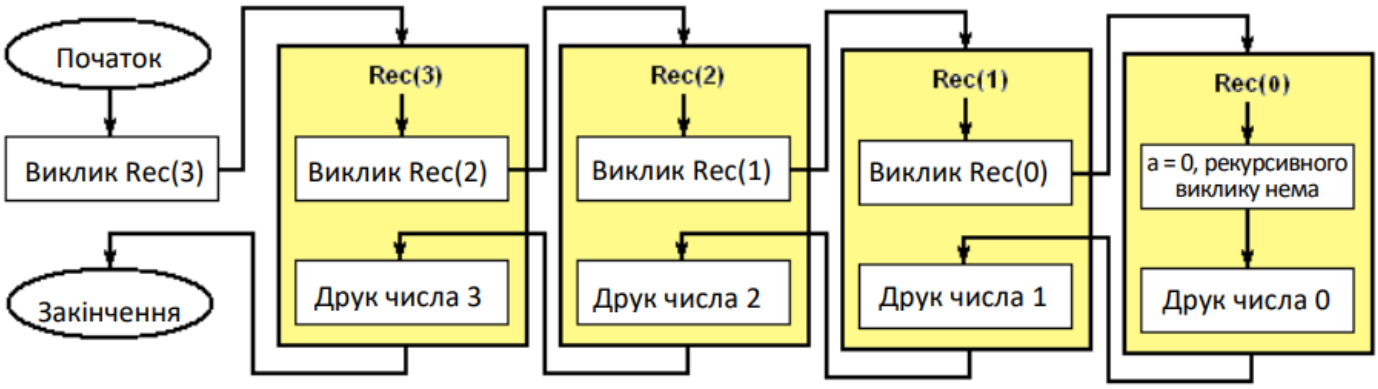
Мова **C++** підтримує можливість звернення функції самої до себе - рекурсію. Розрізняють пряму і непряму рекурсії. Функція називається прямо рекурсивною, якщо містить у своєму тілі виклик самої себе. Якщо ж функція викликає іншу функцію, що у свою чергу викликає першу, то така функція називається непрямо рекурсивною. Рекурсивні функції найчастіше використовуються для компактної реалізації рекурсивних алгоритмів.

Класичні приклади використання рекурсії - реалізація операції піднесення до ступеню і обчислення факторіала числа. Зазначимо, що ці приклади популярні тільки через їхню зручність для пояснення поняття рекурсії, однак вони не дають виграшу в програмній реалізації порівняно з ітераційним способом розв’язання цих задач.

Розглянемо суть на прикладі рекурсивної функції Rec.

```
void Rec( int a)
{    if (a > 0)      // Умова виходу (гранична умова)
     Rec(a - 1); // Шаг рекурсії
     cout << a << endl;
}
```

Розглянемо, що станеться, якщо в основній програмі поставити виклик Rec(3). Нижче представлена блок-схема, що показує послідовність виконання операторів



Функція Rec викликається з параметром a = 3. В ній міститься виклик функції Rec з параметром a = 2. Попередній виклик ще не завершився, тому створюється ще одна функція, і до закінчення її роботи перша свою роботу не закінчує. Процес виклику закінчується, коли параметр a > 0. У цей момент одночасно виконуються 4 екземпляри функції. Кількість одночасно виконуваних функцій називають глибиною рекурсії. Четверта викликана функція (Rec(0)) надрукує число 0 і закінчить свою роботу. Після цього управління повертається до функції, яка її викликала (Rec(1)) і друкується число 1. І так далі доки не завершяться всі функції. Результатом вихідного виклику буде друк чотирьох чисел: 0, 1, 2, 3.

Розглянемо рекурсивну функцію обчислення факторіала. Для того щоб одержати значення факторіала числа n!, необхідно помножити на n факторіал числа (n-1)!. Ураховуючи, що 0! = 1 та 1! = 1, наведемо приклад цієї функції:

```
int fact(int n)
{ return (n-1) ? n*(fact(n-1):1; }
```

Покажчики та посилання.

При виконанні ініціалізації змінної, їй автоматично присвоюється вільна адреса в пам'яті, і, будь-яке значення, яке ми присвоюємо змінній, зберігається за цією адресою в пам'яті. Наприклад:

```
int a;
```

Цій змінній виділяється частина оперативної пам'яті. В якості прикладу припустимо, що змінній априсвоюється комірка пам'яті під номером 0x6dfeec (0x спереду вказує, що це комірка пам'яті в шістнадцятирічній системи числення). Кожен раз, коли програма зустрічає змінну a в виразі, то вона розуміє, що для того, щоб отримати значення - їй потрібно зазирнути в комірку пам'яті під номером 0x6dfeec.

Хороша новина - нам не потрібно турбуватися про те, які конкретно адреси в пам'яті виділені для певних змінних. Ми просто посилаємося на змінну через присвоєний їй ідентифікатор, а компілятор конвертує цей ідентифікатор у відповідну адресу в пам'яті. Однак цей підхід має деякі обмеження.

Оператор адреси & дозволяє дізнатися, яку адресу в пам'яті присвоєно певній змінній. Розглянемо приклад програми.

```
#include <iostream>
#include <windows.h>
using namespace std;
```

```
int main()
{  SetConsoleCP(1251);
   SetConsoleOutputCP(1251);
   int a=40000;
   cout << "  Значення a=" << a << "  Адрес &a=" << &a<< endl;
return 0;
}
```

Результат буде наступний

Значення a=40000 Адрес &a=0x6dfeec

Оператор розіменування *.

```
#include <iostream>
using namespace std;
int main()
{  int a=75;
   cout << "&a= " << &a << " *&a= " << *&a << endl;
return 0;
}
```

Результат

&a= 0x6dfed0 *&a= 75

Примітка: Хоча оператор розіменування виглядає так само, як і оператор множення, відрізнити їх можна по тому, що оператор розіменування - унарний, а оператор множення - бінарний.

Ми розглянули оператор адреси і розіменування і тепер можна розглянути покажчик.

Покажчик - це змінна, значенням якої є адреса комірки в пам'яті. Покажчики оголошуються так само, як і звичайні змінні, тільки із зірочкою між типом даних і ідентифікатором. Оскільки покажчики містять тільки адреси, то при присвоюванні значення покажчику - це значення повинно бути адресую. Для отримання адреси змінної використовується оператор адреси.

Розглянемо приклад покажчика і присвоювання йому значення.

```
include <iostream>
using namespace std;
int main()
{  int a=75;
   cout << "&a= " << &a << " *&a= " << *&a << endl;
   int *pa; // pa це покажчик
   pa=&a; // значення покажчика pa рівно адресу перемінної a
   cout <<  " pa=" <<pa << " *pa= " << *pa << endl;
return 0;
}
```

Примітка: Ця зірочка не є оператором розіменування. Вона всього лише частина синтаксису оголошення покажчика.

Результат

&a= 0x6dfee8 *&a= 75

pa=0x6dfee8 *pa=7

Як видно із результатів і змінна і покажчик мають однакові адреси і значення.

Виникає питання чи доцільно використовувати покажчик, якщо є сама змінна. Слід відмітити, що покажчики корисні в наступних випадках:

1 Масиви (будуть розглядатися в наступному занятті) реалізовані за допомогою покажчиків. Покажчики можуть використовуватися для ітерації по масиву.

2 Вони є єдиним способом динамічного виділення пам'яті в С++ (Цей спосіб буде розглядатися пізніше). Це, безумовно, найбільш поширений варіант використання покажчиків.

3 Вони можуть використовуватися для передачі великої кількості даних в функцію без копіювання цих даних.

4 Вони використовуються для досягнення поліморфізму при роботі зі спадкуванням (розглядається при роботі з класами).

5 Вони можуть використовуватися для представлення однієї структури/класу в іншій структурі/класі, формуючи, таким чином, "ланцюжки".

Практичне заняття № 2.

Після ознайомлення з теоретичними відомостями по темі заняття студенти набирають, відлагожують і запускають програми, які наведені у методичних вказівках. Для кращого сприйняття студентами теме викладач надає завдання на корекцію програм.

Лабораторна робота №2

При виконанні лабораторної роботи студент самостійно складає програму на вірішення задачі, номер варіанту для кожного розділу вказує викладач.

Використання функцій

1) Написати програму, яка використовує функцію для обчислення об'єму конуса.

2) Написати програму з використанням функції, яка порівнює два числа і виводить знак порівняння.

3) Написати програму з використанням функції, яка досліджує можливість вирішення квадратного рівняння і видає:

2 – якщо коренів 2;

1 – якщо корінь 1;

0 – якщо дискримінант менше 0;

–1 – якщо помилка вихідних даних (a = 0).

4) Написати програму з використанням функції для визначення периметру трикутника, який заданий координатами вершин.

Примітка: Нехай (x1, y1), (x2, y2), (x3, y3) – координати вершин трикутника. Щоб обчислити його периметр, потрібні довжини сторін (відрізків із кінцями у вершинах)

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$
$$\sqrt{(x_3 - x_1)^2 + (y_3 - y_1)^2}$$

$$\sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2}$$

Вони обчислюються, по суті, однаково, лише з різними парами точок. Тому використання функціє є доцільним.

5) Написати функцію, що перевіряє, чи можна утворити трикутник із трьох відрізків із заданими довжинами.

6) Написати функцію, що повертає максимальне значення двох її дійсних параметрів.

7) Написати програму, яка використовує функцію для обчислення об'єму циліндра.

8) Написати програму, яка використовує функцію для обчислення об'єму урізаного конусу.

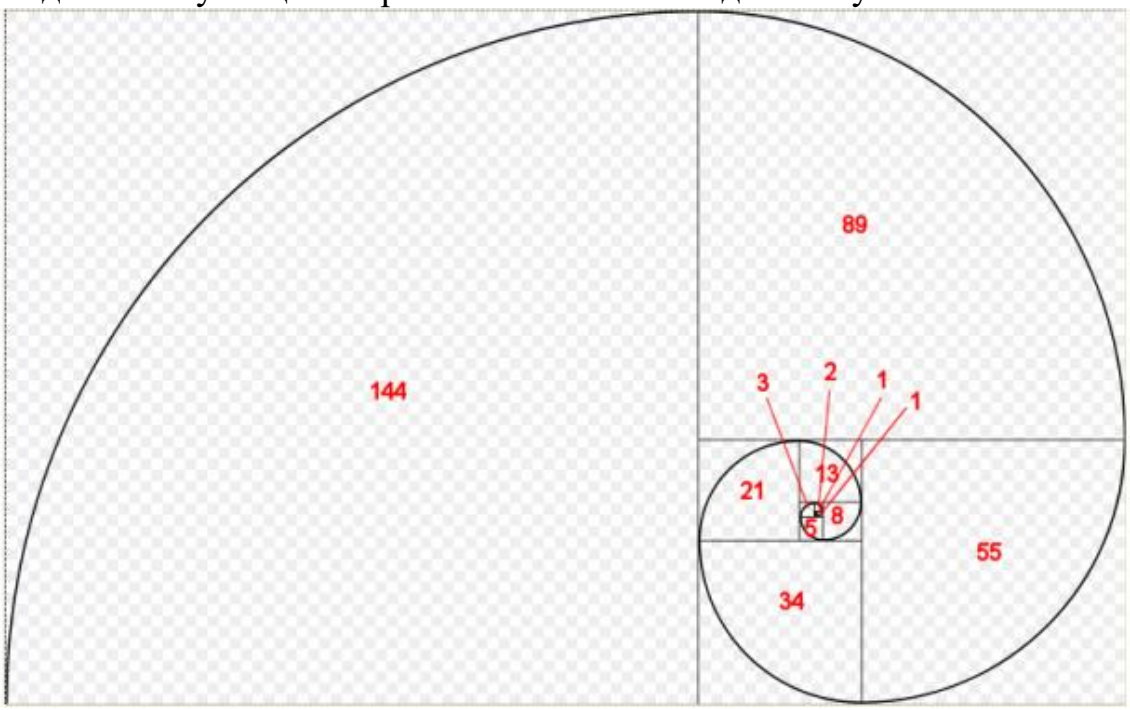
9) Написати програму, яка використовує функцію для визначення кількості днів до кінця місяця.

10) Написати програму, яка використовує функцію для визначення кількості однакових цифр у заданому числі.

Використання рекурсивних функцій і покажчиків

1) Написати функцію, яка знаходить факторіали чисел від 0 до 8.

2) Написати рекурсивну функцію для визначення чисел Фібоначчі. Одним з найбільш відомих математичних рекурсивних алгоритмів є послідовність Фібоначчі. Послідовність Фібоначчі можна побачити навіть в природі: розгалуження дерев, спіраль мушлі, плоди ананасу тощо. Спіраль Фібоначчі виглядає наступним чином:



Кожне з чисел Фібоначчі — це довжина горизонтальної сторони квадрата, в якій знаходиться дане число. Математично числа Фібоначчі визначаються наступним чином:

f(n) = 0, якщо n = 0

1, якщо n = 1

f(n-1) + f(n-2), якщо n > 1.

3) Напишіть рекурсивну функцію, яка приймає ціле число в якості вхідних даних і повертає суму всіх чисел цього значення (наприклад, 482 = 4 + 8 + 2 = 14). Протестуйте вашу програму, використовуючи число 83569 (результатом повинно бути 31).

4) Напишіть програму, яка просить користувача ввести ціле число, а потім використовує рекурсивну функцію для виводу бінарного представлення цього числа.

5) Програма знаходження числа комбінацій із n елементів по m за формулою

$$C_n^m = \frac{n!}{m!(n-m)!}.$$

6) Написати програму знаходження за введеним радіусом кулі її об'єм та площу поверхні. Використовуючи адресації змінних та покажчиків для повернення із функції значення двох змінних.

7) Написати програму в якій введено натуральне число N. Треба визначити чи є воно точним ступенем двійки. Якщо да то вивести YES інакше NO. Використовувати введення в ступінь не можна.

8) Дано натуральне число N. Обчисліть суму його цифр. При вирішенні цього завдання не можна використовувати рядки, списки, масиви (та й цикли, зрозуміло).

9) Дано натуральне число N. Виведіть усі його цифри за одним, у зворотному порядку, розділяючи їх пробілами або новими рядками. При вирішенні цього завдання не можна використовувати рядки, списки, масиви (та й цикли, зрозуміло). Дозволена лише рекурсія та цілочислова арифметика.

10) Розробити програму обчислення найбільшого спільного дільника (дано два числа) з використанням рекурсивної функції.

Тема 3

Масиви. Функції введення-виводу. Рядки та операції з ними. Функції класу string**. Побітові операції**

Короткі теоретичні відомості

Масиви - це впорядкований набір об'єктів, у яких однаковий тип, наприклад, набір цілих чисел або символів, які розрізняються за порядковим номером. У масиві можна зберігати якусь кількість однотипних об'єктів без необхідності введення окремого імені змінної для кожного з них. Опис масивів у програмі відрізняється від опису простої змінної наявністю після імені квадратних дужок «[ ]», в яких задається кількість елементів масиву (розмірність). Слід нагадати, що у мові C++ нумерація елементів масиву починається з 0.

Масив може бути одновимірним або багатовимірним. Кількість вимірювань і довжина кожного з вимірювань визначає загальну довжину масиву.

Масив, як і будь-який інший об'єкт в мовах C/C++, має бути оголошений перед тим, як він буде використаний: тип ідентифікатор [довжина 1]...[довжина N]. В залежності від того який масив одномірний, двомірний і буде вказуватись довжина кожного вимірювання.

Оголошення одномірного масиву може бути наступним:

```
int a[5];
int a[5]= {3, 9, 12, 7, 4};
int a[ ]= {3, 9, 12, 7, 4};
```

Як видно кількість елементів масиву відома і її змінити неможливо.

Багатовимірні масиви, можна оголошувати, розглядаючи їх як масив масивів наступним:

```
int a[3][4];
int a[3][4]= {3, 9, 12, 7, 1, 5, 14, 18, 2, 6, 8, 10, 13, 15, 17, 19,  23, 25, 27, 26};
int a[ ][ ]={{3, 9, 12, 7},{1, 5, 14, 18},{2, 6, 8, 10},{13, 15, 17, 19},{23, 25, 27, 26}}
```

За замовчуванням, якщо в оголошеному масиві задають тільки декілька перших елементів, то його інші елементи будуть нулями. Так, у випадку, коли

```
float mas[10]= {2.2,34.56};
```

, останні вісім елементів масиву одержать значення 0.

Індекс масиву – це цілочисельний вираз, значення якого може бути в діапазоні від 0 до значення, рівного довжині вимірювання, зменшеній на 1.

Якщо при виведенні на друк вказати ім'я масиву, то буде надруковане значення комірки з якої починається масив. Таким чином ім'я масиву є фактично покажчиком на масив.

Приклад програми в якій видно, що ім'я масиву є покажчиком і допускає арифметику з цим покадчиком (наприклад arr+i).

```
#include <iostream>
#include <windows.h>
using namespace std;
```

```
int main()
{     SetConsoleCP(1251);
      SetConsoleOutputCP(1251);
      const int SIZE=4;
      int arr[]{45,23,87,2};
      for(int i=0;i<SIZE;i++)
      {     cout << " arr="<< arr[i] << endl;    }
      int *p1=arr;
      cout << "  p1=" <<p1 <<"  *p1=" << *p1 << endl;
      cout << "  arr=" << arr << endl << endl;
      for(int i=0;i<SIZE;i++)
      {   cout << " Адрес p1="<< (p1+i) << "   Значення =" << *(p1+i) << endl;    }
      cout <<"SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS"<< endl;
      for(int i=0;i<SIZE;i++)
      {  cout << " Адрес arr="<< (arr+i) << "   Значення =" << *(arr+i) << endl;    }
      cout << " Адрес 4 arr="<< (arr+4) << "  Значення 4="<<*(arr+4)<< endl;
return 0;
}
```

Результати

```
 arr=45
 arr=23
 arr=87
 arr=2
 p1=0x6dfebc  *p1=45
 arr=0x6dfebc
 Адрес p1=0x6dfebc   Значення =45
 Адрес p1=0x6dfec0   Значення =23
 Адрес p1=0x6dfec4   Значення =87
 Адрес p1=0x6dfec8   Значення =2
 SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
 Адрес arr=0x6dfebc   Значення =45
 Адрес arr=0x6dfec0   Значення =23
 Адрес arr=0x6dfec4   Значення =87
 Адрес arr=0x6dfec8   Значення =2
 Адрес 4 arr=0x6dfecc  Значення 4=7208636
```

Як видно із результатів масив починається з комірки пам’яті arr=0x6dfebc і додавання поточного значення і (arr+i) збільшує адресу пам’яті комірки на 4 байти arr=0x6dfec0.

Нижче наведено приклад програми яка використовує функції з масивами. Масиви передаються у функцію по покажчику. У функціях не створюється дублікат масиву, а функція напряму працює з комірками пам’яті, де розташовані елементи масиву.

```
#include <iostream> // Передача масиву в функцію
```

```
using namespace std;// Передача масиву в функцію по покажчику
void Prin(int x[6], int n)  //int *x=&a[0]
{     x[0]=11;
      for(int i=0;i<n;i++) cout << x[i] << "\t";
      cout << endl;
}
void Prin2(int x[], int n) // int *x=&a[0]
{     for(int i=0;i<n;i++) cout << x[i] << "\t";
      cout << endl;
}
void Prin3(int *ptr, int n) // int *x=&a[0]
{     for(int i=0;i<n;i++) cout << ptr[i] << "\t";
      cout << endl;
}
int main()
{     int a[6];  //1
      for(int i=0;i<6;i++)
      {  a[i]=rand()%10; //формування елементів масиву випадковими значеннями
      cout << a[i] << "\t";
}
      cout << "\n";
      Prin(a,6);
      Prin2(a,6);
      Prin3(a,6);
      cout << "\n\n";
      for(int i=0;i<6;i++)  cout << a[i] << "\t";
return 0;
}
```

Результати

```
1     7     4     0     9     4
11    7     4     0     9     4
11    7     4     0     9     4
11    7     4     0     9     4
11    7     4     0     9     4
```

Як видно із результатів, в першій функції нульовій комірки присвоюється значення x[0] = 11 замість 1. Саме це значення передається в основну програму. Таким чином в першій функції передається посилання на масив a і змінюючи значення масиву у функції ми змінюємо значення в основній програмі. Слід відмітити, що вказуючи у функції масив без квадратних дужок означає, що передається покажчику масиву.

Функції введення-виводу

Для відображення даних в консоль використовується оператор cout, який служить для виведення простого тексту, а також значень змінних у консоль.

Оператор cin – здійснює введення даних із консолі. Як тільки програма зустрічає даний оператор, вона зупиняється і чекає на реакцію користувача, поки користувач не введе дані і не натисне "Введення" (Enter). Тільки після цього продовжиться виконання.

Синтаксис оператора введення: cin » ім'я змінної;. Ім'я змінної вказує на змінну, в яку потрібно помістити дані, введені з клавіатури: Наприклад:

cin >> a >> b;

Таким чином зчитують значення двох змінних a, b, які вводяться з клавіатури (в рядок два числа через пробіл, або в стовбець через Enter). Вони будуть занесені в комірки пам'яті, які виділені для ідентифікаторів a, b.

Введення-виведення даних у мові C++ реалізовано з використанням концепції потоків. Під потоком розуміють процес введення-виведення даних (як послідовності символів) у файл.

Периферійні пристрої введення-виведення клавіатура і монітор розглядають як текстові файли. Під час виконання будь-якої програми автоматично підключають стандартні потоки для введення даних з клавіатури та виведення на монітор. При бажанні потоки можна перенаправити на інший пристрій, наприклад, файл жорсткого диску чи флеш-накопичувача.

Вказівки введення-виведення даних для стандартних пристроїв (клавіатури й монітора) описано у бібліотеці iostream (Input/Output Stream — потік введення / виведення), яку необхідно підключати майже в кожній програмі. Цю бібліотеку включено у стандартну бібліотеку C++.

В C++ використовуються спеціальні типи даних — потоки:

- ifstream — для зчитування;
- ofstream — для запису;
- fstream — для зчитування і запису.

При роботі с текстовими файлами потрібно від'єднувати бібліотеку fstream.

Алгоритм зчитування з (текстового) файлу:

- Описати змінну типу ifstream.
- Відкрити файл з допомогою функції open.
- Зчитати інформацію з файлу.
- Закрити файл.

Алгоритм запису у (текстовий) файл:

- Описати змінну типу ofstream.
- Відкрити файл з допомогою функції open.
- Вивести інформацію у файл.
- Закрити файл.

Відкриття файлу можна здійснити за допомогою функції fopen(), яка повертає покажчик на структуру типу FILE, який можна використовувати для подальших операцій з файлом.

FILE * fp (*назва файлу, тип*);

Тут тип визначає доступу до файлу при його відкритті:

- "r" — для читання (файл повинен існувати);

- "w" — для запису (якщо файл існує, його вміст буде втрачено);
- "a" — для запису в кінець (файл буде створено, якщо його немає);
- "r+" — для читання і запису (файл повинен існувати);
- "w+" — порожній файл для читання і запису;
- "a+" — для читання і доповнення (файл буде створено, якщо його немає).

Параметр режим можна не вказувати. У цьому випадку файл буде відкрито в режимі як встановлено для даного потоку. Після вдалого відкриття файлу у файловій змінній буде збережено значення true, після невдалого — false. Це дає можливість перевірити коректність відкриття файлу.

Розглянемо три еквівалентні за результатом приклади здійснення кроків 1 і 2 для потоку f:

- ofstream f; f.open("D:\\univer\\file.txt", ios::out);
- ofstream f; f.open("D:\\univer\\file.txt");
- ofstream f("D:\\unever\\file.txt", ios::out);

Примітка. Зверніть увагу на потребу писати додаткову косу риску в записі шляху до файлу.

Після відкриття файлу в режимі запису в нього можна виводити дані так само, як і на екран. Але, замість стандартного пристрою виведення cout необхідно вказати файлову змінну. Наприклад, fp << a; або fp << b << c << d;

Закриття потоку здійснюють оператором close. Наприклад, fp.close();

Зчитування інформації з текстового файлу задають аналогічно запису, але з використанням типу ifstream (замість ofstream) і >> замість << — див. наступний приклад.

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{    int a,b;
     ifstream fp; //Оголошення потоку зчитування з файлу fp
     ofstream fo; //Оголошення потоку запису у файл fo
     fp.open("input.txt");  //Прив'язування назви файлу до fp
     fo.open("output.txt"); //Прив'язування назви файлу до fo
     fp >> a >> b;         //Зчитування з файлу
     fo << a*b << endl;    //Запис у файл
     cout << a << " * " << b << " = " << a*b;
     fp.close();
     fo.close();
return 0;
}
```

Із програми видно, що спочатку підключається бібліотека #include <fstream>, а потім оголошуються покажчика fp, fo. Вказані покажчики зв'язуються з відповідними файлами input.txt і output.txt. Далі в програмі зчитуються дані a, b із файлу input.txt, а у файл output.txt записуються добуток a*b. В кінці обидва файли fp та fo закриваються.

Якщо відомо лише тип, але не кількість величин у файлі, потрібно перед кожним зчитуванням перевіряти, чи досягнуто кінець файлу за допомогою булевої функції fp.eof(), де fp — назва потоку. Функція повертає true або false відповідно до того, досягнуто кінець файлу чи ні.

Рядки та операції з ними.

Символами вважаються: великі й малі латинські літери, великі й малі літери кирилиці, цифри, знаки арифметичних дій ('+', '–', '*', '/', '='), пробіл, розділові знаки ('.', ',', ';', ':', '!', '?', '–'), службові символи, що відповідають клавішам , <Enter>, <Esc>, <Tab> тощо. В C++ значення символьних констант записуються у одинарних лапках: '7', 'g', '-'.

Крім того в ASCII-кодів є так звана escape послідовність символів, яка не друкується і починається зі зворотної скісної риси і записується в одинарних лапках. Кожна з наведених нижче комбінацій символів вважається за один символ.

\n символ переведення курсора на початок наступного рядка

\r переведення каретки

\t символ переведення курсора на наступну позицію табуляції (відповідає клавіші),

\b символ вилучення попереднього символу перед курсором (відповідає клавіші),

\a символ звукового сигналу системного динаміка

\\ символ \ (зворотна скісна риса)

\? символ ? (знак запитання)

\' символ ' (одинарні лапки)

\" символ " (подвійні лапки)

\0 нуль-символ.

Тип символьних змінних у C++ називається char. Наприклад, при оголошенні

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

У мови C++ рядки розглядалися як символьні масиви, і вся робота з ними ґрунтувалася на використанні цих масивів. Розроблена бібліотека функцій string.h містить потужні засоби для роботи з рядковими масивами. Рядок являє собою масив символів, який закінчується нуль-символом. Нагадаємо, що нуль-символ має код, що дорівнює 0, і запис у вигляді керуючої послідовності '\0'. За розташуванням нуль-символу визначається фактична довжина рядка (нуль символ входить в довжину рядка).

Для опису рядка використовуються звичайні засоби опису масивів, наприклад: char str[25];.

Відмінною рисою символьного масиву є те, що в ньому насправді може бути менше символів, аніж зазначено при оголошенні. Окрім того, з цими масивами можна виконувати певні специфічні дії, які не можна здійснювати з числовими масивами (наприклад перевіряти наявність у масиві літери чи послідовності літер, копіювати масив як одне ціле, порівнювати масиви за алфавітом, дописувати один масив наприкінці іншого тощо).

Індексування такого масиву, як і будь-якого іншого, починається з нуля.

Символьні послідовності, розділені тільки пропусками, розглядаються як один рядок, тобто запис:

“Всі студенти
групи присутні на занятті”
ідентичний до рядку: “ Всі студенти групи присутні на занятті”.

Адреса першого символу рядка може використовуватися по різному:

- якщо рядок застосовується при ініціюванні масиву типу char, адреса його першого елемента стає синонімом імені масиву. Наприклад, ідентичними є такі описи масиву:

```
char s [ ] = "Група";
char s [6] = "Група";
char s [6] = {'Г','р','у','п','а','\0'};
```

- якщо рядок використовується для ініціювання покажчика типу char*, адреса першого символу рядка буде початковим значенням покажчика, наприклад:

```
char *pst = "Група";
```

Тут описується змінна-покажчик pst, яка одержує початкове значення, що дорівнює адресі першого елемента (символу **‘Г‘**);

- якщо рядок використовується у виразі, що застосовує покажчик, то компілятор підставляє у вираз рядка адресу його першого символу, наприклад:

```
char *pst;
pst = "Група програмістів";.
```

Тут pst одержує адресу символу «Г» (тобто першого символу рядка).

Слід звернути увагу на те, що при описі символьного масиву його ім’я — не змінна, а покажчик-константа на початок рядка, тому її не можна використовувати в деяких операціях адресної арифметики.

Функції класу string

Для роботи з символьними рядками є бібліотека функцій string.h

Функція	Пояснення
strlen(ім’я рядка)	Визначає довжину вказаного рядка, без врахування нуль-символу
Копіювання рядків	
strcpy(s1,s2)	Виконує побайтове копіювання символів із рядка s2 в рядок s1
strncpy(s1,s2, n)	Виконує побайтове копіювання n символів із рядка s2 в рядок s1і повертає рядок s1
Конкатенація рядків	
strcat(s1,s2)	Об’єднує рядок s2 з рядком s1. Результат зберігає в s1
strncat(s1,s2,n)	Об’єднує n символів рядка s2 з рядком s1. Результат зберігає в s1
Порівняння рядків	
strcmp(s1,s2)	Порівнює рядок s1 з рядком s2 і повертає результат типу int: 0 – коли рядки еквівалентні, >0 – коли s1<s2, <0 – коли s1>s2 з урахуванням регістру
strncmp(s1,s2,n)	Порівнює n символів рядка s1 з рядком s2 і повертає результат типу int: 0 – коли рядки еквівалентні, >0 – коли s1<s2, <0 – коли s1>s2 з урахуванням регістру
stricmp(s1,s2)	Порівнює рядок s1 з рядком s2 і повертає результат типу int: 0 – коли рядки еквівалентні, >0 – коли s1<s2, <0 – коли s1>s2 без урахування регістру
strnicmp(s1,s2,n)	Порівнює n символів рядка s1 з рядком s2 і повертає результат типу int: 0 – коли рядки еквівалентні, >0 – коли s1<s2, <0 – коли s1>s2 без урахуванням регістру
Обробка символів	

Функція	Пояснення
isalnum(c)	Повертає значення true, коли с є буквою або цифрою, і false в інших випадках
isalpha(c)	Повертає значення true, коли с є буквою і false в інших випадках
isdigit(c)	Повертає значення true, коли с є цифрою і false в інших випадках
islower(c)	Повертає значення true, коли с є буквою нижнього регістру і false в інших випадках
isupper(c)	Повертає значення true, коли с є буквою верхнього регістру і false в інших випадках
isspace(c)	Повертає значення true, коли с є пробілом і false в інших випадках
toupper(c)	Якщо символ с є символом нижнього регістру, то функція перетворює символ с у верхньому регістрі, інакше символ повертається без змін
Функція пошуку	
strchr(s,c)	Пошук першого входження символу с в рядок s. У випадку вдалого пошуку повертає покажчик на місце першого входження символу с. Коли символ не знайдено, то повертається нуль.
strcspn(s1,s2)	Визначає довжину початкового сегмента рядка s1, містить ті символи, які не входять в рядок s2
strspn(s1,s2)	Визначає довжину початкового сегмента рядка s1, містить ті символи, які входять в рядок s2
strprbk(s1,s2)	Повертає покажчик першого входження будь якого символу рядка s2 в рядок s1
Функція перетворення	
atof(s1)	Перетворює рядок s1 в тип double
atol(s1)	Перетворює рядок s1 в тип long int
Функція стандартної бібліотеки введення/виведення <stdio>	
getchar(c)	Зчитує символ с із стандартного потоку введення, повертає символ у форматі int
gets(s)	Зчитує потік символів зі стандартного пристрою введення в рядок s доти, доки не буде натиснута клавіша ENTER

Приклад використання функції strcpy(s1,s2)

```
#include <iostream>
#include <string.h>
using namespace std;
int main()
{    char s1[6]="Hello";
     char s2[]="Warning";
     strcpy(s1,s2);
     cout << " s1= " << s1 << endl;
     cout << " s2= " << s2 << endl;
return 0;
}
```

Результат

s1= Warning

s2= Warning

Як видно із результату, незважаючи на те що s2 має 8 символів, а рядок s1 всього 6 символів копіювання виконано. Таким чином збільшено кількість символів у рядку s1 до 8.

Побітові операції

На практиці досить часто доводиться відстежувати стан різних програмних об'єктів з допомогою прапорів. Для цього використовуються булевські змінні. Однак якщо дуже багато ознак, зручніше використовувати окремі біти змінних. Це дозволяє заощаджувати пам'ять.

Для того щоб вільно адресуватися до окремих біт, використовуються так звані порозрядні логічні операції, які представлені в таблиці

1	&	логічне І (множення)
2	|	логічне АБО (додавання)
3	^	що виключає АБО
4	~	логічне НЕ (інверсія)

Результат застосування оператора 1 «логічне І» множення

A	B	C=A&B
0	0	0
0	1	0
1	0	0
1	1	1

Результат застосування оператора 2 «логічне АБО» складання

A	B	C=A|B
0	0	0
0	1	1
1	0	1
1	1	1

Результат застосування оператора 3 «Виключає АБО» - XOR

A	B	C=A^B
0	0	0
0	1	1
1	0	1
1	1	0

Результат застосування оператора 4 «логічне НЕ»

A	C=~A
0	1
1	0

Операції зсуву вліво та вправо

Для здійснення зсуву послідовності біт ліворуч і праворуч застосовують відповідно операції << і >>. Операнд праворуч від знаку операції вказує, на яку величину мають бути зрушені біти. Задаючи тим самим кількість біт, «виведених» із змінних, і число нульових біт, заповнюють змінну з іншого боку.

Слід враховувати, що з використанні правого зсуву, якщо найстарший біт дорівнює одиниці (ознака негативного числа у змінної зі специфікатором signet), деякі компілятори можуть «ввести» нулі зліва. Для цього рекомендується перетворювати операнд операції на беззнаковий тип (unsigned)

Приклад побітових операцій &- і | - або ^ -xor негативне або ~ - не

```
#include <iostream>
using namespace std;
int main()
{    int a=5;     int b=7;
     printf("%i\n",a&b); //101  111  -> 101  ->5
     printf("%i\n",a|b); //101  111  -> 111  ->7
     printf("%i\n",a^b); //101  111  -> 010  ->2
     printf("%i\n",~a);  // -6
return 0;
}
```

Результат в один рядок 5 7 2 -6.

Значення 5 в двійковій системі це 101, а 7 – 111. Всі бітова операції прості і виконуються в одну дію, окрім ~ - не. Вона складна і виконується наступним чином. Спочатку 101 інвертується 11111010, далі віднімається 1 і отримаємо 11111001, потім число інвертується крім старшого біту 10000110. Отримаємо -6.

Приклад який дозволяє у введеному невід'ємному цілому 16 розрядному числі за допомогою бітових операцій визначити кількість одиниць у двійковому поданні введеного числа.

```
#include <iostream>
#define BIT_COUNT 16 // Константа
#include <bitset>
using namespace std;
int main()
{    unsigned short num;
     int i;
     unsigned short s=0;
     unsigned short mask=1; //Маска 0000000000000001
     cout <<" Enter the number";     cin >>num;
     cout <<" num=" << num << endl;
     bitset<16> y(num);// Виведення числа num в двійковому обчисленні
     cout << "bin num =" << y << '\n';
     for(i=1;i<=BIT_COUNT;i++)
```

```
      { if((num&mask)>0) // побітове і - коли буде  >0 то це буде одиниця
            s++;
            mask=mask<<1; // побітовий зсув на одиницю вліво
      }
      cout <<"Number s= "<<s<< endl;
return 0;
}
```

Результат

```
    Enter the number  23
     num=23
    bin num =0000000000010111
    Number s = 4
```

Приклад стиснення інформації за рахунок використання бітових операцій

```
#include <iostream>
#include <bitset>
using namespace std;
int main()
{ char micx;      int tmp;
   cout <<" Enter: 1 - male; 2 - female: ";
   cin >> tmp;     cout <<endl;
   micx=micx|tmp;
   printf("micx=%i\n",micx);
   cout <<" Please enter: 1 - You are under 18 years old; 2 - You are over 18 years old:";
    cin >> tmp;    cout <<endl;
    micx=micx|(tmp<<2);
    printf("micx=%i\n",micx);
    cout <<"Enter:1 - you draw badly; 2 - you draw well; 4 - you are an artist:";
    cin >> tmp;    cout <<endl;
    micx=micx|(tmp<<5);
    printf("micx=%i\n",micx);
    for(int i=15;i>=0;i--)         { printf("%i",(micx>>i)&1);        }
    cout <<endl;
    bitset<16> y(micx);  // Виведення числа micx у двійковому обчисленні
    cout << "bin micx =" << y << '\n';
return 0;
}
```

Результат

```
  Enter: 1 - male; 2 - female: 1
  micx=1
  Please enter: 1 - You are under 18 years old; 2 - You are over 18 years old: 2
  micx=9
  Enter:1 - you draw badly; 2 - you draw well; 4 - you are an artist: 4
   char micx=I
```

micx=73
01001001
bin micx =01001001

Перші два байти числа micx 01 – введено 1, третій і четвертий байт 10 – введено 2, з п’ятого по сьомий 100 – введено число 4. Таким чином у одному байті зберігається інформація трьох питань.

Практичне заняття № 3.

Після ознайомлення з теоретичними відомостями по темі заняття студенти набирають, відлагожують і запускають програми, які наведені у методичних вказівках. Для кращого сприйняття студентами теме викладач надає завдання на корекцію програм.

Лабораторна робота №3

При виконанні лабораторної роботи студент самостійно складає програму на вірішення задачі, номер варіанту для кожного розділу вказує викладач.

Масиви

1) Написати програму яка знаходить в масиві число, яке є найближче до введеного числа. Наприклад є масив: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100. Введено число 37. Найближчим є число 40.
2) Написати програму яка знаходить в масиві число, яке є найближчим менше до введеного числа. Наприклад є масив: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100. Введено число 37. Найближчим меншим до введеного є число 30.
3) Ввести масив з 10-ти цілих чисел. Створити новий масив, елементи якого обчислити, поділивши кожен з елементів початкового масиву на суму його елементів з непарними індексами.
4) Ввести масив до 11-ти дійсних чисел і з’ясувати, чи є він упорядкованим за зростанням.
5) Розробити програму для введення до 15-ти дійсних чисел і впорядкувати їх за зростанням.
6) У послідовності цілих чисел непарні елементи (за значенням) замінити на одиниці, а парні елементи – на нулі.
7) В одновимірному масиві дійсних чисел, який складається з парної кількості елементів, переставити місцями елементи, які стоять поряд (1 та 2, 3 та 4 тощо).
8) Розробити проект для введення матриці дійсних чисел розмірністю 3×5 і віднайти максимальний елемент матриці та його індекси.
9) Розробити проект для введення матриці цілих чисел розмірністю 5×7 та обчислення елементів вектора сум від’ємних елементів стовпчиків матриці.
10) Ввести матрицю розмірністю 4×4 дійсних чисел і замінити елементи головної діагоналі на мінімальні елементи відповідних рядків.

Введення–виведення

1) Заповнити матрицю 5×5 випадковими числами і записати цю матрицю до текстового файлу.

2) Заповнити матрицю 3×5 числами з текстового файлу і обчислити суму елементів матриці.

3) Створити програму запису у файл всіх чисел Фібоначчі, які не перевищують натуральне число n.

4) Дано файл, компоненти якого є цілими числами. Скласти програму для обчислення: а) кількості парних чисел серед компонент; б) кількості квадратів непарних чисел серед компонент.

5) Дано файл, компоненти якого є цілими числами. Скласти програму для обчислення: а) різниці між найбільшим парним і найменшим непарним числами з компонент; б) кількості компонент у найдовшій зростаючій послідовності компонент файлу.

6) Скласти програму запису до файлу всіх непарних чисел Фібоначчі, які не перевищують натуральне число n.

7) Скласти програму запису до файлу всіх чисел Фібоначчі, що є простими числами та не перевищують натуральне число n.

8) Дано файл F, компоненти якого є цілими числами. Побудувати файл G, який містив би всі компоненти файлу F: а) що є парними числами.

9) Дано файл F, компоненти якого є цілими числами. Побудувати файл G, який містив би всі компоненти файлу F: а) що діляться на 3 і на 5.

10) Дано файл F, компоненти якого є цілими числами. Побудувати файл G, який містив би всі компоненти файлу F: а) що є точними квадратами.

Рядки

1) Скласти програму перевірки входження заданого символу до заданого рядка

2) Вивести на друк лише великі латинські літери, що входять до заданого рядка.

3) В заданому рядку всі символи '0' замінити на '1', а символи '1' – на '0'.

4) Створити програму перетворення рядка, замінивши в ньому кожну крапку трьома крапками.

5) Скласти програму побудови інверсії заданого рядка. Інверсією рядка називається рядок, записаний тими самими символами у зворотному порядку.

6) Скласти програму підрахунку загальної кількості входжень символів '+', '-', '*' у рядок.

7) Скласти програму, яка визначає позицію першого (останнього) входження заданого символу до заданого рядка.

8) Рядок містить арифметичний вираз, у якому використовуються круглі дужки. Перевірити, чи правильно в ньому стоять дужки.

9) Скласти програму, що повертає кількість різних символів у рядку.

10) Знайти символ, який входить до рядка S найбільшу кількість разів.

Побітові операції

1) Скласти програму яка стискає інформацію чотирьох питань з двома відповідями в кожному питанні до одного байту.

2) Скласти програму яка стискає інформацію чотирьох питань з двома відповідями в кожному питанні до 4 бітів.

3) Скласти програму яка визначає у довільному введеному невід'ємному цілому 16 розрядному числі кількість одиниць у двійковому поданні введеного числа, а також виводить позицію одиниць.

4) Скласти програму яка виводить на консоль всі двійкові числа одного байту як у зростаючому порядку так і у зворотньому.

5) Скласти програму знаходження в масиві одного числа. Всі числа крім одного входять в масив парне число разів. Знайти число, яке входить в масив непарне число разів*.

6) Відомо, що у масиві всі числа крім двох входять парне число раз. Скласти програму знаходження в масиві два числа, які входять в масив непарне число разів. Примітка можна використовувати тільки один масив**.

Тема 4

Сортування масивів.

Короткі теоретичні відомості

Сортування масиву— один з найбільш розповсюджених процесів обробки даних. Завдяки йому здійснюється розміщення об'єктів у визначеному порядку, наприклад, чисел за зростанням або за спаданням їх значень, прізвищ у алфавітному порядку тощо.

Найчастіше задачі сортування вирішуються в інформаційних пошукових системах, бо пошук у впорядкованих масивах проводиться набагато швидше, ніж у невпорядкованих.

Існують різні методи сортування, серед них — обмінне сортування (метод «бульбашки»), сортування вибором, сортування вставками, швидке сортування, сортування розділення і злиття, сортування Шелла тощо. Ці методи відрізняються швидкістю отримання результату, складністю і універсальністю.

Розглянемо деякі методи сортування: обміном, вибором та вставками, сортування розділення та злиттям, сортування Шелл. Названі методи легко описуються у формі чітких алгоритмів і передбачають нескладну програмну реалізацію.

Сортування вибором.

Сортування елементів масиву у зростаючому порядку за методом вибору можна описати наступним чином:

1. Серед всіх 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.

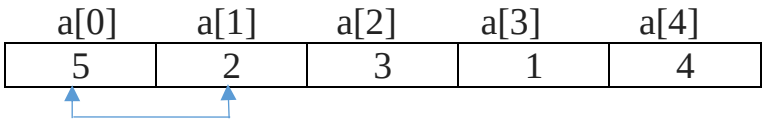
Аналіз описаних вище дій показує, що для програмної реалізації цього методу сортування буде потрібно два цикли for.

У зовнішньому циклі повинен змінюватися номер елементу, куди буде заноситися черговий мінімальний елемент. Номери цих елементів мають змінюватися від першого до передостаннього. Цей цикл буде визначати кількість проходів по масиву.

Внутрішній цикл повинен забезпечити послідовне порівняння елементу, зафіксованого першим циклом, з усіма елементами, які слідують в масиві за ним.

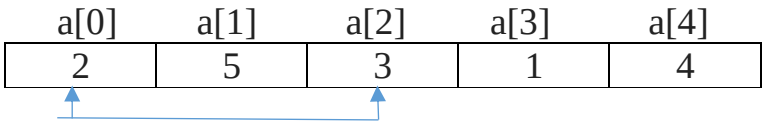
У тілі внутрішнього циклу відбувається порівняння елементів, індекси яких задаються параметрами зовнішнього і внутрішнього циклу. Якщо в результаті порівняння знаходиться елемент, що менший ніж зафіксований, то порівнювані елементи міняються місцями.

Розглянемо приклад сортування вибором графічно на п'яти елементах.



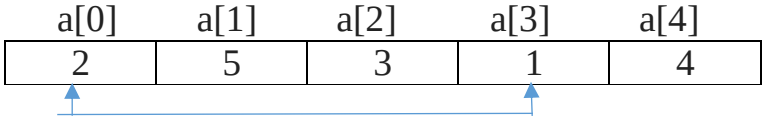
2 < 5 – да тоді потрібно обміняти a[0] і a[1]

Після обміну масив буде мати наступний вигляд



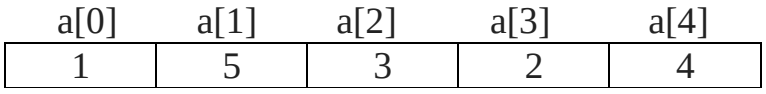
3 < 2 - ні тоді обмін не потрібен

Далі



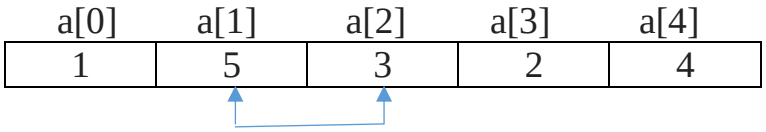
1 < 2 - да тоді потрібно обміняти a[0] і a[3]

Після обміну масив буде мати наступний вигляд

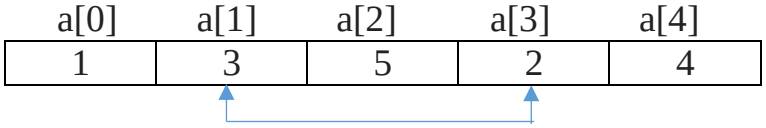


Далі порівнюють a[4]=4 < a[0]=1 – ні і обмін не потрібен. Тому масив після першого проходу буде мати саме такий вигляд.

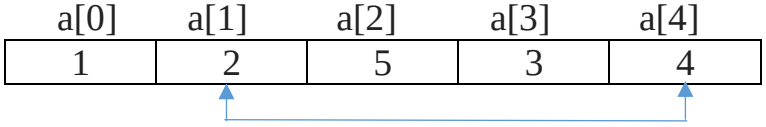
Другий перегляд масиву вже потрібно починати с другого елементу і шукати мінімальне значення с другого по п’ятий елемент. Результати другого проходу представлені на рисунку.



3 < 5 - да - тоді потрібно обміняти a[1] і a[2]

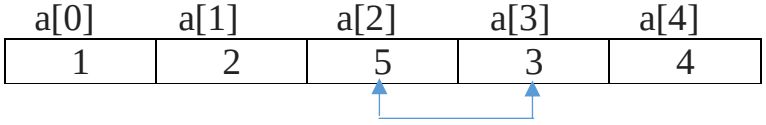


2 < 3 - да - тоді потрібно обміняти a[1] і a[3]

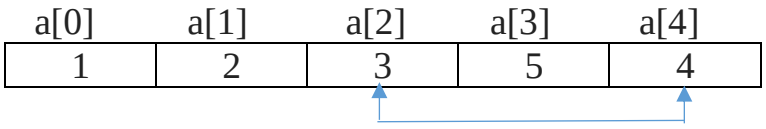


4 < 2 - ні - тоді обмін не потрібен

Під час третього перегляду масиву в ролі змінюваного елементу виступає елемент A[2]=5, і з ним будуть послідовно порівнюватися елементи A[3] і A[4]. Як наслідок, елементи A[2] і A[3] поміняються значеннями.

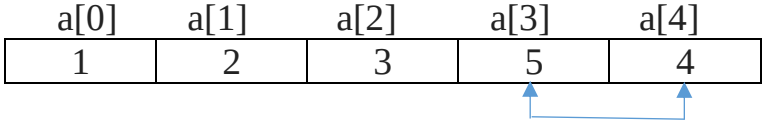


3 < 5 – так - тоді потрібно обміняти a[2] і a[3]



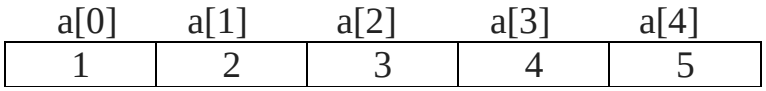
4 < 3 - ні - тоді обмін не потрібен

При останньому, четвертому проході елемент A[3] поміняється значенням з єдиним, наступним за ним, елементом A[4].



4 < 5 – так - тоді потрібно обміняти a[3] і a[4]

Відповідно отримаємо відсортований масив виду



Функція, яка буде реалізовувати вказаний метод сортування вибором буде мати вид

```
void sortCh(int arr[], int size)
{ for(int i=0; i<size-1; i++)
    { for(int j=i+1; j<size; j++)
        { if(arr[j]<arr[i])
            {int b =arr[i];   arr[i]=arr[j];   arr[j]= b; }
        }
    }
}
```

Сортування методом «бульбашки» (обмін)

У методі «бульбашки» кожен елемент масиву, починаючи з першого, порівнюється з наступним, і якщо він більше наступного, то елементи міняються місцями. В результаті після першого проходу, при сортуванні на зростання, найбільший елемент виявиться на останньому місці.

У наступному проході все повторюється заново і найбільший серед решти елементів виявляється на передостанньому місці.

Кожен прохід по масиву від його початку упорядковує щонайменше один елемент у хвості масиву, тому кожного наступного проходу доводиться розглядати менше пар елементів ніж на попередньому проході, бо останні елементи вже впорядковані.

Інколи початкове розташування елементів може бути таким, що масив може бути відсортований за невелику кількість проходів. Ознакою впорядкованості масиву є відсутність обмінів. Тому для скорочення кількості проходів по масиву, слід після закінчення кожного з них перевіряти, чи був зроблений хоч один обмін значень елементів.

Програма має два цикли. Зовнішній цикл виконується поки по завершенні чергового проходу по масиву не виявиться, що перестановок елементів не було. У цьому випадку змінна per збереже значення true, і це означатиме, що масив відсортований. У внутрішньому циклі послідовно змінюється індекс елементів, що порівнюються. Тому

для організації проходу по масиву використовується цикл for. Максимальне значення параметра цього циклу у кожному проході зменшується на 1.

Функція, яка виконує сортування методом бульбашок буде

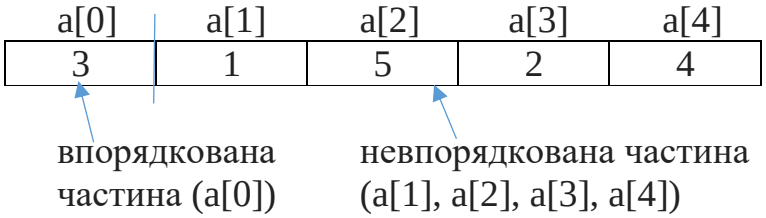
```
void sortBu(int arr[], int size)
{ int zik=size; bool per;
   do
   { zik-=1;  per=true;
     for(int i=0; i<zik; i++)
     { if arr[i] > arr[i+1]
       { int a=arr[i];  arr[i]=arr[i+1];  arr[i+1]-a;  per=falsel; }
       }
   } while(!per);
}
```

Сортування вставкою

Уявимо, що у довільний момент масив має дві частини впорядковану частину a[0], a[1],…, a[i-1], яка розташована на початку масиву і невпорядковану частину a[i], a[i+1],…, a[n]. Потрібно вставити перший елемент із невпорядкованої частини у впорядковану. Послідовні кроки сортування полягають у тому, що перший елемент з невпорядкованої частини порівнюється з останнім елементом впорядкованої послідовності. Якщо виявляється, що порядок розташування цих елементів не відповідає вимогам сортування, то елемент з невпорядкованої частини вилучається і переноситься у впорядковану частину. Місце для цього елементу звільняється шляхом зсуву впорядкованих елементів вправо на місце витягнутого елемента. Зсув впорядкованих елементів на одну позицію вправо триває, поки не буде знайдено місце для елемента, вилученого з невпорядкованої послідовності.

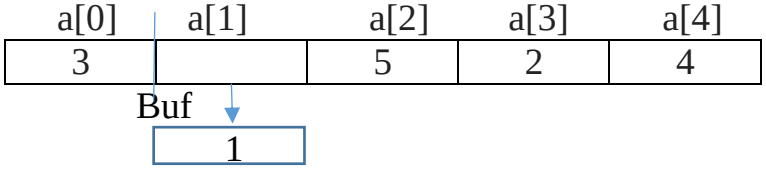
Щоб уявить, як проводиться сортування вставкою розглянемо схему дій на рисунку. На початку сортування впорядкована частина складається всього з одного, першого елемента, а всі інші елементи розташовуються в другій частині масиву.

Початкове розміщення елементів масиву

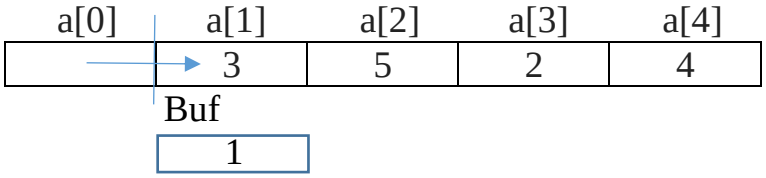


Перевіряємо a[1] < a[0], порядок порушено.

Переносимо a[1] у буфер

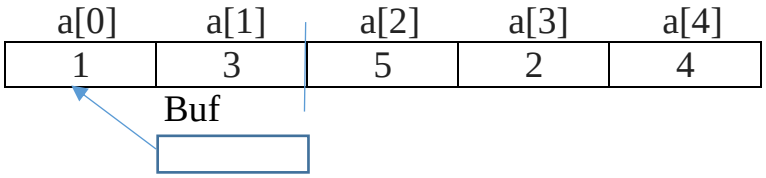


Переміщуємо a[0] на місце a[1]



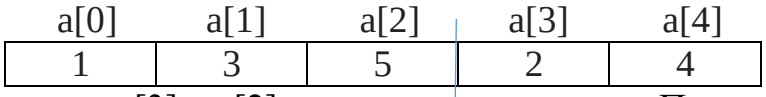
Перевіряємо, чи підходить місце, що звільнилося, для елемента з буферу.

Місце підходить, тому переміщуємо елемент із буферу у позицію a[0].

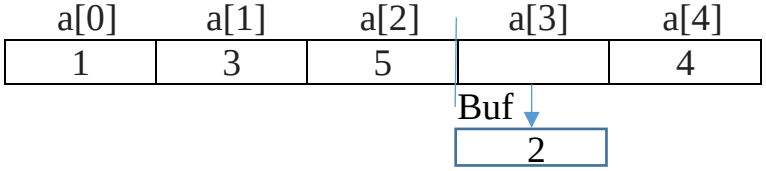


Тепер два впорядкованих елемента ліворуч від a[2].

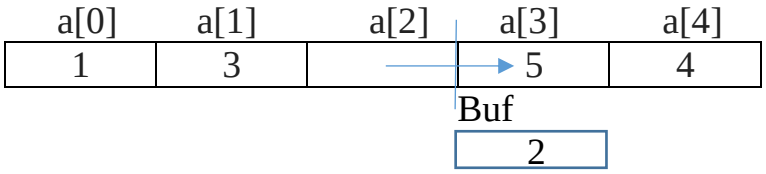
Далі перевіряємо a[2] < a[1], ні нічого міняти непотрібно і межа впорядкованості зміщується праворуч.



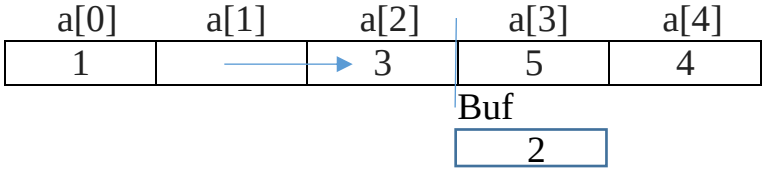
a[3] < a[2], да порядок порушено. Переносимо a[3] в буфер



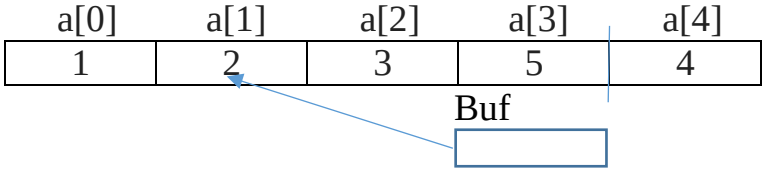
Переміщуємо a[2] вправо на a[3]



Перевіряємо, чи підходить місце, що звільнилося, для елементу з буфера. Місце не підходить, тому переміщуємо a[1] у позицію a[2].



Перевіряємо, чи підходить місце, що звільнилося, для елементу з буфера. Місце підходить, тому переміщуємо значення із буфера у позицію a[1].



Тепер ліворуч вже чотири елементу впорядковані.

На останньому кроці через буфер буде обмін a[3], і a[4].

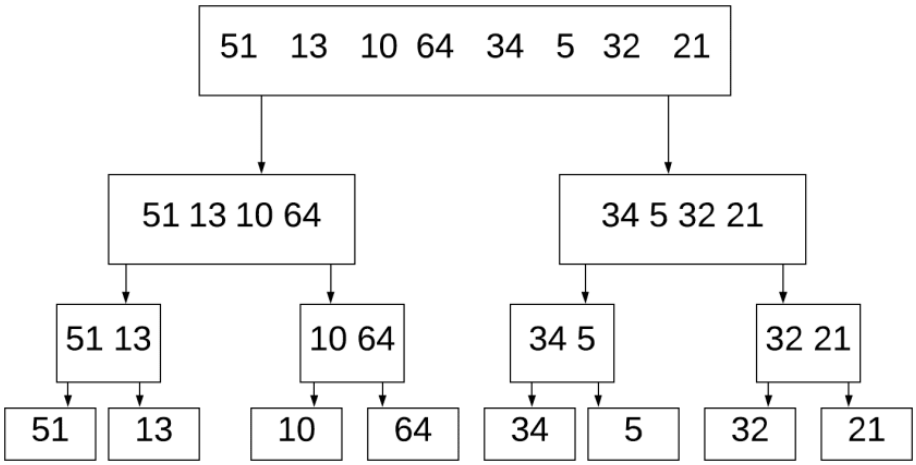
Функція, яка реалізує сортування вставкою буде мати наступний вигляд

```
void sortIn(int arr[], int size)
{ for(int i=1; i<size; i++)
    {if(arr[i] < arr[i-1]) //Порівнюємо елементи на межі сортування
        { int buf=arr[i]; // Порядок порушено, переносимо i-й елемент у буфер
        int j=i; // Починаємо зсув   j – індекс вільного місця
        do { arr[j]=arr[j-1]; // Зсув праворуч
            j--;
        // Поки ліворуч число більше, або дійшли початку масиву
            } while (j!=0 && buf < arr[j-1]);
        arr[j]=buf;  // Вставляємо елемент, що мав номер i на нове місце з номером j
    }
  }
  }
```

Сортування злиттям

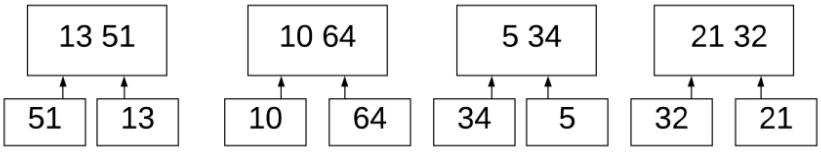
Merge Sort – це алгоритм сортування. Його ще називають «Поділяй і володарюй». Алгоритм працює наступним чином:

- Поділяємо послідовність із n чисел на 2 половини
- Рекурсивно сортуємо дві половинки
- Об'єднуємо дві відсортовані половини в одну відсортовану послідовність

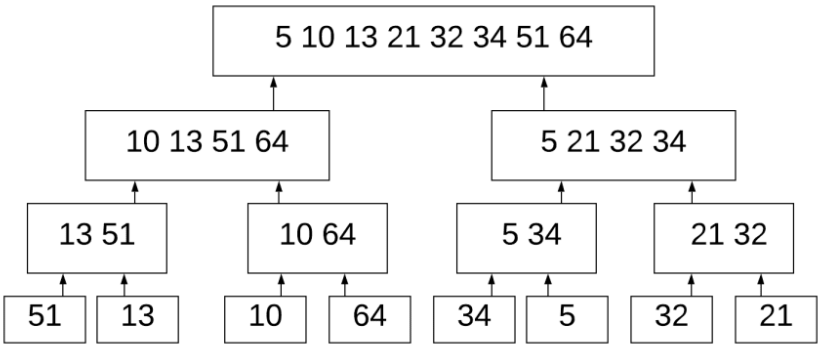


На цьому зображенні ми розбиваємо 8 чисел на окремі цифри. Тепер можемо розпочати процес сортування.

Порівнює 51 і 13. Оскільки 13 менше, поміщаємо її в ліву частину. Так само робимо для (10, 64), (34, 5), (32, 21).



Потім об'єднуємо (13, 51) з (10, 64). Ми знаємо, що 13 є найменшим у першому списку, а 10 – найменшим у правому списку. 10 менше 13, тому нам не потрібно порівнювати 13 із 64. Ми порівнюємо та об'єднуємо два відсортовані списки.



Сортування злиттям є прикладом алгоритму «поділяй і володарюй». Слід відмітити, що цей спосіб сортування є більш швидким, чим розглянуті раніше.

Функція, яка реалізує сортування «поділяй і володарюй» включає в себе ще одну функцію і буде мати наступний вигляд

```
void merge(int merged[], int lenD, int L[], int lenL, int R[], int lenR);
void mergeSort(int data[], int lenD)
{    if(lenD>1)
     {    int middle = lenD/2;
          int rem = lenD-middle;
          int* L = new int[middle];
          int* R = new int[rem];
          for(int i=0;i<lenD;i++)
               {  if(i<middle) { L[i] = data[i]; }
                    else{ R[i-middle] = data[i]; }
               }
               mergeSort(L,middle);
               mergeSort(R,rem);
               merge(data, lenD, L, middle, R, rem);
     }
}
 void merge(int merged[], int lenD, int L[], int lenL, int R[], int lenR)
{    int i = 0;
     int j = 0;
     while(i<lenL||j<lenR)
          { if (i<lenL & j<lenR)
               {  if(L[i]<=R[j]){   merged[i+j] = L[i];   i++;   }
                    else{   merged[i+j] = R[j];    j++;   }
                }
               else if(i<lenL){  merged[i+j] = L[i];  i++; }
                         else if(j<lenR){  merged[i+j] = R[j];  j++; }
          }
}
```

Сортування Шелла

Сортування вставками виконується над двома сусідніми елементами, у зв'язку з чим елемент може пересуватися вздовж масиву лише на одне місце за один раз. Наприклад, якщо елемент з найменшим значенням ключа виявляється в кінці масиву, потрібно зробити N кроків, щоб помістити його в належне місце. Сортування методом Шелла є найпростішим розширенням методу вставок, швидкодія якого вища за рахунок забезпечення можливості обміну місцями елементів, які знаходяться далеко один від одного. Саме в цьому і полягає суть сортування методом Шелла.

У методі Шелла пропонується використовувати послідовність кроків 1 4 13 40 121 364 1093 3280 9841. Як видно із послідовності кожен раз вона зменшується в 3 рази і округляється в меншу сторону.

Практичне заняття № 4.

Після ознайомлення з теоретичними відомостями по темі заняття студенти набирають, відлагожують і запускають програми, які наведені у методичних вказівках. Для кращого сприйняття студентами теме викладач надає завдання на корекцію програм.

Лабораторна робота №4

При виконанні лабораторної роботи студент самостійно складає програму на вірішення задачі сортування.

Сортування.

Студенти повинні виконати дві роботи.

1 Кожен студент повинен розробити програму сортування масиву із 8 елементів 51, 13, 10, 64, 34, 5, 32, 21 використовуючи функції, які наведені в коротких теоретичних відомостях. Метод сортування вказує викладач.

2 В додатках наведені програми сортування масиву (100000 елементів) різними методами. Потрібно набрати програму відповідного методу сортування (вказує викладач) її налагодити і отримати час, який витрачений на сортування масиву.

Тема 5

Бібліотека STL. Вектори, ітератори, списки. Функція сортування.

Короткі теоретичні відомості

Бібліотека (Standard Template Library) – це бібліотека контейнерних класів, яка включає вектори, списки, черги і стеки, а також цілий ряд алгоритмів загального призначення. Ціль включення STL в стандарт мови – позбавлення розробника писати рутинні коди.

Ядро бібліотеки складає три групи шаблонних класів: контейнери, алгоритми і ітератори.

STL представляє два виду контейнерів: послідовні і асоціативні.

Послідовні контейнері призначені для забезпечення послідовного або довільного доступу до своїх елементів.

Асоціативні контейнери отримують доступ до своїх елементів по ключу.

Всі контейнерні класи бібліотеки STL визначені в просторі імен std.

Послідовні контейнері це вид контейнерів, в якому введено відношення порядку для сукупності збережених об'єктів. Для елементів контейнера визначені поняття перший, наступний, попередній і наступний. До цього виду контейнерів належать: вектори, масиви, списки, деки і строки типу string.

Вектори – це контейнерний клас, в якому доступ до його елементів здійснюється по індексу (як в масивах).

При напису коду потрібно підключити бібліотеку векторів #include <vector>

Вектор забезпечує контроль кількості елементів.

Фактично вектор це динамічний масив (дуже прокачений).

Вектори

Клас vector (або просто *«вектор»*) — це динамічний масив, здатний збільшуватися в міру необхідності для утримання всіх своїх елементів. Клас vector забезпечує довільний доступ до своїх елементів через оператор індексації [], а також підтримує додавання і видалення елементів.

Розглянемо приклади застосування векторів.

У першому прикладі вектор описується, додаються елементи вектора в кінець (push_back), визначається довжина вектора (size()), створюється додатковий елемент вектора як у масиві, показується, що за межами вектора не можна призначити значення елементу вектора ().

```
int main()
{    SetConsoleCP(1251);
     SetConsoleOutputCP(1251);
     vector<int> vec_maj;
     vec_maj.push_back(10);
     vec_maj.push_back(20);
     vec_maj.push_back(30);
```

```
    vec_maj.push_back(40);
    vec_maj[0]=12;
    cout <<" Кіл-сть елементів =" << vec_maj.size() << endl;
    for(int i=0;i<vec_maj.size();i++)  { cout <<vec_maj[i] << endl;   }
    vec_maj[20]=10002; // Присвоювання 20 елементу значення, але він за межами вектора
    cout << "vec_maj[20]=" << vec_maj[20] << endl;
                    // кількість елементів незмінна - всього 4
    cout <<" Кіл-сть елементів =" <<   vec_maj.size() << endl;
return 0;
}
```

У другому прикладі використовується метод at() доступу до елементу з перевіркою виходу за межі, конструктори ініціалізації елементів масиву, метод capacity() визначення довжини вектора з резервними комірками і показується чим він відрізняється від size().

```
int main()
{   SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    vector<int> vec_maj;
    vec_maj.push_back(10);  // Додає елементи вектора в кінець
    vec_maj.push_back(20);
    vec_maj.push_back(30);
    vec_maj.push_back(40);
    // Вказує на елемент 2. at() не виходить за межи вектору. Якщо виходить -  ПОМИЛКА
    cout <<vec_maj.at(2) << endl;
    vec_maj.pop_back(); // Видаляє останній елемент вектора
    cout <<" Кіл-сть елементів =" << vec_maj.size() << endl;
    cout <<" Кіл-сть елементів =" << vec_maj.size() << endl;
    vec_maj.clear(); // Видаляє всі елементи вектору
    vector<int> vec_maj={2,34,52,67}; // Конструктор вектора –ініціалізація
    for(int i=0;i<vec_maj.size();i++)  { cout <<vec_maj[i] << endl;}
    vector<int> vec_maj(6,25); // Конструктор вектора – ініціалізація 5 і 6 будуть -25
    for(int i=0;i<vec_maj.size();i++) { cout <<vec_maj[i] << endl;   }
    // capacity() відрізняється size() тим, що показує додаткові комірки, які поки вільні
  cout << "Скільки ще елементів " << vec_maj.capacity() << endl; // Скільки ще елементів є
    vec_maj.reserve(20); // Резервує місце
    cout << "Скільки ще елементів " << vec_maj.capacity() << endl;
return 0;
}
```

У третьому прикладі показуються методи: резервування комірок (reserve(20)), зменшення пам’яті за рахунок комірок які не використовуються (shrink_to_fit()), перевірка на наявність елементів вектора (empty()), змінює розмір вектора і вставляє нулі (resize(10)).

```
int main()
```

```
{    SetConsoleCP(1251);
     SetConsoleOutputCP(1251);
     vector<int> vec_maj;
     vec_maj.push_back(10);
     vec_maj.push_back(20);
     vec_maj.push_back(30);
     vec_maj.push_back(40);
     cout << "Скільки ще є елементів" << vec_maj.capacity() << endl;
     vec_maj.;
     cout << "Скільки ще є елементів" << vec_maj.capacity() << endl;
     vec_maj.shrink_to_fit(); // Зменшує пам’ять за рахунок яка не використовується
     cout << "Скільки ще є елементів" << vec_maj.capacity() << endl;
     cout << vec_maj.empty() << endl; // перевірка чи є елементи true або false
     cout << "_____" << endl;
  vec_maj.resize(10); // Конструктор - Змінює розмір вектора на потрібну значення і вставляє 0
   vec_maj.resize(11,76); // Конструктор - Додає до існуючого вектора 11 елемент, який буде 76
     for(int i=0;i<vec_maj.size();i++)    { cout <<vec_maj[i] << endl; }
return 0;
}
```

Ітератори

Дуже важливим інструментом роботи з векторами, списками та іншими контейнерами є метод ітараторів. Він полягає в наступному. Елементам вектора або іншому контейнеру в бібліотеці STL відповідає певне значення ітератора (номер комірки). Ітератор спрощує роботу з векторами або іншими контейнерами (наприклад списками). При використанні векторів ітератор не є критичним, тому що можна як у масивах звернутися до будь-якої комірки вектора не використовуючи ітератор.

Перед застосуванням ітератора при роботі з векторами потрібно підключити відповідну бібліотеку #include <iterator>. Далі потрібно вказати тип ітератора і його назву vector<int>::iterator iter;. Встановлюється початкове значення ітератора iter = vector1.begin();. В даному випадку в початок вектора. Далі ітератор виступає фактично покажчиком на відповідну комірку вектора (*iter). В прикладі ітератор використовується як покажчик для послідовного виклику значень комірок вектора і отримання суми всіх його значень.

```
#include <iostream>
#include <vector>
#include <iterator>
using namespace std;
int main()
{ vector<int> vector1;
  vector<int>::iterator iter;
  for (int i=0; i < 10; i++)
          { vector1.push_back(i+5); }
```

```
  for(int i=0;i<vector1.size();i++)
      cout << vector1[i] << endl;
  cout << "_____________" << endl;
  int total = 0;
  iter = vector1.begin();
  cout << "vector1[0]="<< *iter << endl;
  while (iter != vector1.end()) { total += *(iter++); }
  cout << "summa= " << total << endl;
return 0;
}
```

Результати розрахунку.

```
5
6
7
8
9
10
11
12
13
14

_____________
vector1[0]=5
summa= 95
```

Вектор є аналогом динамічного масиву C++, а контейнер array являє собою статичний масив у якого відомий тип елементів а також відома кількість елементів.

Список

list (або просто *«список»*) — це двонаправлений список, кожен елемент якого містить 2 покажчики: один вказує на наступний елемент списку, а інший — на попередній елемент списку (дивись рисунок). list надає доступ тільки з початку або з кінця списку — довільний доступ заборонено. Якщо потрібно знайти значення десь в середині, то потрібно почати з одного кінця і перебирати кожен елемент списку до тих пір, поки не буде знайдено необхідний елемент. Перевагою двонаправленого списку є те, що додавання елементів відбувається дуже швидко, якщо відомо куди потрібно додавати. Зазвичай для перебору елементів двонаправленого списку використовуються ітератори.

Функція сортування

Розглянемо приклад використання ітераторів для контейнерів array і алгоритм сортування. Видно, що контейнер array має ім’я s і включає 10 цілих значень. Сортування здійснюється по зростанню з початку (s.begin()) до кінця (s.end()). Для друку використовується ітератор (ostream_iterator). Друге сортування проводиться по спаданню з 5 елементу (s.begin()+5) і до кінця.

```
#include <algorithm>  // Для підключення sort
#include <array>   // В бібліотеці STL
#include <iostream>
#include <iterator>
using namespace std;
int main()
{   array<int, 10> s{5, 7, 4, 2, 8, 6, 1, 9, 0, 3};
    sort(s.begin(), s.end()); // Сортує по зростанню с початку до кінця
    copy(s.begin(), s.end(), ostream_iterator<int>(cout," ")); // Друкує
    cout << endl;
    cout << "_____________" << endl;
    sort(s.begin()+5, s.end(), greater<int>()); // Сортує у зворотньому порядку з 5 елементу
    copy(s.begin(), s.end(), ostream_iterator<int>(cout," "));
    cout << endl;
    return(0);
}
```

Результати розрахунку

0 1 2 3 4 5 6 7 8 9

_____

0 1 2 3 4 9 8 7 6 5

Практичне заняття № 5.

Після ознайомлення з теоретичними відомостями по темі заняття студенти набирають, відлагожують і запускають програми, які наведені у методичних вказівках. Для кращого сприйняття студентами теме викладач надає завдання на корекцію програм.

Лабораторна робота №5

При виконанні лабораторної роботи студент самостійно складає програму на вірішення задачі, номер варіанту для кожного розділу вказує викладач.

Бібліотека STL

1) Сортування вектора по зростанню {19, 23, 27, 5, 3, 2, 1, 12, 27, 15}
2) Сортування вектора по спаданню {19, 23, 27, 5, 3, 2, 1, 12, 27, 15}
3) Сортування вектора - перша половина по зростанню, а друга по спаданню {19, 23, 27, 5, 3, 2, 1, 12}
4) Написати програму додавання елементів вектору в кінець і визначення кількості елементів у вектору
5) Написати програму додавання елементів вектору в кінець і використовуючи ітератор встановити нове значення у друго елемента
6) Написати програму конструювання 10 елементів вектора зі значенням 4
7) Написати програму додавання 10 елементів списку {19, 23, 27, 5, 3, 2, 1, 12, 27, 15}

8) Написати програму визначення мінімального значення із 10 елементів списку {19, 23, 27, 5, 3, 2, 1, 12, 27, 15}
9) Написати програму визначення додавання мінімального та максимального значення із 10 елементів списку {19, 23, 27, 5, 3, 2, 1, 12, 27, 15}
10) Написати програму визначення в списку кількості елементів, які мають відемне значення. Список створити самостійно

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 С++. Основи програмування. Теорія та практика: підручник / [О.Г. Трофименко, Ю.В. Прокоп, І.Г. Швайко, Л.М. Буката та ін.] ; за ред.О.Г.Трофименко. – Одеса: Фенікс, 2010. – 544 с.

1 Основи програмування на С/С++ в прикладах. Частина 1: навч.-метод. посібник / Соболь М.О., Любченко Н.Ю, Паржин Ю.В., Пугачов Р.В. – Харків : НТУ "ХПІ", 2021. – 113 с

2 Коротенко Г.М., Коротенко Л.М. Методичні вказівки до виконання лабораторних робіт з дисципліни «Обчислювальна техника та програмування». Мова програмування С++ / Дніпро: НТУ «ДП», 2021 – 179 с.

3 Ткачук В.М. Програмування на С++. Лабораторний практикум. Іваво-Франківськ : Вид-во Прикарпатського національного університету імені Василя Стефаника, 2011, - 160 с.

4 Грицюк Ю., Рак Т. Програмування мовою С++: навчальний посібник. Львів, Вид-во ЛДУ БЖД, 2011, - 292 с.

5 Галкін О.В., Верес М.М. Мова програмування С++: конспект лекцій. К.: ДП «Вид.дім Персонал», 2017, - 260 с

6 http://cpp.dp.ua/

7 https://acode.com.ua/urok-33-rozmir-typiv-danyh/

ДОДАТОК А

Програма сортування методом «бульбашки»

```
#include <fstream>
#include <iostream>
#include <string>
#include <stdlib.h>
#include <ctime>
using namespace std;
template <class Item>
void sort_puzrk(Item a[], int r)
{ int d=0;  int l=0;  int v;  bool b;
  for(int i=0;i<r-1;i++)  {   b=true;
                              for(int j=0;j<r-i-1;j++)
                                  { if(a[j]>a[j+1]) { v=a[j]; a[j]=a[j+1]; a[j+1]=v; b=false; }
                                    d++;
                                  }
                             if (b) goto label1;
                            }
  label1: cout << " d=" << d << endl;
}
int main()
{   unsigned int start_time =  clock();
    cout << " start_time =" << float(start_time)/CLOCKS_PER_SEC << endl;
    FILE *f;
    f = fopen("1.txt","w");
    ofstream fout ("1.txt");
    int r=100000;
    int arr[r];
    srand(time(NULL));
    for(int i=0;i<r;i++)  { arr[i]=(rand()%r);  }
    int k=0;
    for(int i=0;i<r;i++)
    { if (k<20) {fout << arr[i] << "\t"; k++;}
            else {fout << endl; k=0; i=i-1;}
    }
    sort_puzrk(arr,r);
    fout << endl;
    for(int i=0;i<r;i++)
    { if (k<20) {fout << arr[i] << "\t"; k++;}
            else {fout << endl; k=0; i=i-1;}
    }
    fclose(f);
    unsigned int end_time = clock();
```

```
    cout << " end_time= " << float(end_time)/CLOCKS_PER_SEC << endl;
    cout << " Time = " << float(end_time - start_time)/CLOCKS_PER_SEC  << endl;
    return 0;
}
```

ДОДОТОК Б

Програма сортування методом вибору

```
#include <fstream>
#include <iostream>
#include <string>
#include <stdlib.h>
#include <ctime>
using namespace std;
template <class Item>
void wubor(Item a[],int r)
{ for(int i=0;i<r-1;i++)
   { for(int j=i+1;j<r;j++)
      { if(a[j]<a[i]) {int tmp=a[i]; a[i]=a[j]; a[j]=tmp;}
      }
   }
}
int main()
{   unsigned int start_time =  clock();
    cout << " start_time =" << float(start_time)/CLOCKS_PER_SEC << endl;
    FILE *f;
    f = fopen("1.txt","w");
    ofstream fout ("1.txt");
    int r=100000;
    int arr[r];
    srand(time(NULL));
    for(int i=0;i<r;i++)
                          { arr[i]=(rand()%r); }
    int k=0;
    for(int i=0;i<r;i++)
    { if (k<20) {fout << arr[i] << "\t"; k++;}
            else {fout << endl; k=0; i=i-1;}
    }
    wubor(arr,r);
    fout << endl;
    for(int i=0;i<r;i++)
    { if (k<20) {fout << arr[i] << "\t"; k++;}
            else {fout << endl; k=0; i=i-1;}
    }
    fclose(f);
    unsigned int end_time = clock();
    cout << " end_time= " << float(end_time)/CLOCKS_PER_SEC << endl;
    cout << " Time = " << float(end_time - start_time)/CLOCKS_PER_SEC  << endl;
    return 0;
}
```

ДОДАТОК В

Програма сортування методом вставки

```
#include <fstream>
#include <iostream>
#include <string>
#include <stdlib.h>
#include <ctime>
using namespace std;
template <class Item>
void wstawka(Item a[],int r)
{ for(int i=1;i<r;i++)
            { if(a[i]<a[i-1]){  int buf=a[i]; int j=i;
                                do {a[j]=a[j-1]; j=j-1;}
                                while(j!=0&&buf<a[j-1]);
                               a[j]=buf;
                             }
            }
}
int main()
{   unsigned int start_time =  clock();
   cout << " start_time =" << float(start_time)/CLOCKS_PER_SEC << endl;
   FILE *f;
   f = fopen("1.txt","w");
   ofstream fout ("1.txt");
   int r=100000;
   int arr[r];
   srand(time(NULL));
   for(int i=0;i<r;i++)    { arr[i]=(rand()%r);    }
   int k=0;
   for(int i=0;i<r;i++) { if (k<20) {fout << arr[i] << "\t"; k++;}
                                else {fout << endl; k=0; i=i-1;}
                        }
   wstawka(arr,r);
   fout << endl;
   for(int i=0;i<r;i++)
   { if (k<20) {fout << arr[i] << "\t"; k++;}
            else {fout << endl; k=0; i=i-1;}
   }
   fclose(f);
   unsigned int end_time = clock();
   cout << " end_time= " << float(end_time)/CLOCKS_PER_SEC << endl;
   cout << " Time = " << float(end_time - start_time)/CLOCKS_PER_SEC  << endl;
   return 0;
}
```

ДОДАТОК Г

Програма сортування методом злиття

```
#include <fstream>
#include <iostream>
#include <string>
#include <stdlib.h>
#include <ctime>
using namespace std;
void merge(int merged[], int lenD, int L[], int lenL, int R[], int lenR);
void mergeSort(int data[], int lenD)
{  if(lenD>1){    int middle = lenD/2;
                  int rem = lenD-middle;
                  int* L = new int[middle];
                  int* R = new int[rem];
                  for(int i=0;i<lenD;i++)
                    {  if(i<middle){   L[i] = data[i];   }
                               else{  R[i-middle] = data[i];   }
                    }
                 mergeSort(L,middle);
                mergeSort(R,rem);
                merge(data, lenD, L, middle, R, rem);
              }
}
void merge(int merged[], int lenD, int L[], int lenL, int R[], int lenR)
{int i = 0;
 int j = 0;
 while(i<lenL||j<lenR){
   if (i<lenL & j<lenR){
                          if(L[i]<=R[j]){   merged[i+j] = L[i];   i++;  }
                                   else{    merged[i+j] = R[j];    j++;  }
                        }
                        else if(i<lenL){   merged[i+j] = L[i];   i++;  }
                                 else if(j<lenR){  merged[i+j] = R[j];   j++;  }
                       }
}
int main()
{   unsigned int start_time =  clock();
   cout << " start_time =" << float(start_time)/CLOCKS_PER_SEC << endl;
   FILE *f;
   f = fopen("1.txt","w");
   ofstream fout ("1.txt");
   int r=100000;
   int arr[r];
```

```
    srand(time(NULL));
    for(int i=0;i<r;i++)    { arr[i]=(rand()%r);     }
    int k=0;
    for(int i=0;i<r;i++)
                { if (k<20) {fout << arr[i] << "\t"; k++;}
                        else {fout << endl; k=0; i=i-1;}
                 }
     mergeSort(arr,r);
    fout << endl;
    for(int i=0;i<r;i++)  { if (k<20) {fout << arr[i] << "\t"; k++;}
                                    else {fout << endl; k=0; i=i-1;}
                            }
    fclose(f);
    unsigned int end_time = clock();
    cout << " end_time= " << float(end_time)/CLOCKS_PER_SEC << endl;
    cout << " Time= " << float(end_time - start_time)/CLOCKS_PER_SEC  << endl;
    return 0;
}
```

ДОДАТОК Д

Програма сортування методом Шелла

```
#include <fstream>
#include <iostream>
#include <string>
#include <stdlib.h>
#include <ctime>
using namespace std;
template <class Item>
void shell_1(Item a[],int r)
{ int h; int d=0;
  for(h=1;h<=(r-1)/9;h=3*h+1);
     for( ;h>0;h/=3)
        for(int i=h;i<=r;i++)  { int j=i;Item v=a[i];
                                  while(j>=h && v<a[j-h])
                                 {a[j]=a[j-h];j-=h;d++; }
                                 a[j]=v;
                               }
  cout << "d=" << d << endl;
}
int main()
{   unsigned int start_time =  clock();
    cout << " start_time =" << float(start_time)/CLOCKS_PER_SEC << endl;
    FILE *f;
    f = fopen("1.txt","w");
    ofstream fout ("1.txt");
    int r=100000;
    int arr[r];
    srand(time(NULL));
    for(int i=0;i<r;i++)    { arr[i]=(rand()%r);   }
    int k=0;
    for(int i=0;i<r;i++)
    { if (k<20) {fout << arr[i] << "\t"; k++;}
            else {fout << endl; k=0; i=i-1;}
    }
    shell_1(arr,r);
    fout << endl;
    for(int i=0;i<r;i++)
    { if (k<20) {fout << arr[i] << "\t"; k++;}
            else {fout << endl; k=0; i=i-1;}    }
    fclose(f);
    unsigned int end_time = clock();
    cout << " end_time=" << float(end_time)/CLOCKS_PER_SEC << endl;
    cout << " Время =" << float(end_time - start_time)/CLOCKS_PER_SEC  << endl;
    return 0;
}
```