ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

О. А. Золотухіна

Програмування C++

(частина 2)

Київ – 2021

Золотухіна О.А., Програмування С++. Навчальний посібник. Частина 2. К.: ДУТ, 2021. 52 с.

Посібник призначено для вивчення окремих тем дисципліни «Програмування С++». В ньому відображено навчальний матеріал, який вивчається у другому семестрі дисципліни для спеціальності 121, Інженерія програмного забезпечення. В посібнику відображено змістовні модулі «Обробка складних структур даних» та «Структурування програм» і розглянуто теми, пов’язані із використанням вказівників, рядків, структур, файлів, а також особливості структурування програм із застосуванням функцій.

Посібник містить теоретичний матеріал за темами, приклади програмного коду для розв’язання типових практичних задач, завдання для самостійного опрацювання та контрольні питання.

Посібник може бути використаний студентами спеціальності «Інженерія програмного забезпечення» освітньо-кваліфікаційного рівня «бакалавр» при вивченні дисципліни «Програмування С++», а також всіма, хто самостійно вивчає програмування.

ЗМІСТ

1. ОПИС НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

Навчальна дисципліна «Програмування С++» вивчається студентами освітньо-кваліфікаційного рівня «бакалавр» спеціальності 121, «Інженерія програмного забезпечення» (галузь знань 12, «Інформаційні технології»). Дисципліна вивчається в першому та другому семестрах на 1 курсі. Даний посібник призначено для використання в другому навчальному семестрі.

Мета дисципліни «Програмування С++» в другому семестрі – навчити студентів процесу якісної розробки програм мовою С++ із використанням імперативної парадигми програмування, розвинути навички складання програм обробки динамічних структур даних, рядків, структур, файлів, структурування програм за допомогою функцій, створення рекурсивних програм, а також засвоєння основ об’єктно-орієнтованого підходу в розробці програм С++.

Завдання: отримання студентом компетенцій для того, щоб розробляти структуровані програми мовою С++ із використанням типових схем та алгоритмів обробки складних даних з використанням імперативної та об’єктно-орієнтованої парадигм програмування.

В результаті вивчення даного курсу в другому семестрі студент повинен

знати:

– поняття вказівника, призначення та особливості використання вказівників в С++;

– особливості обробки динамічних масивів;

– особливості представлення та обробки рядків в С++;

– поняття структури (struct) в С++, особливості обробки структур, створення та обробка складних структур даних на основі struct;

– поняття файлу та потоку, класифікацію та особливості обробки файлів в С++;

– поняття функції в С++, особливості передачі параметрів в функції, параметри функції main;

– поняття рекурсії, особливості використання рекурсивних функцій;

– принципи проектування програм «з гори до низу» та «з низу до гори», особливості структурування програм, поняття «заглушки»;

– принципи розробки архітектури, модулів та компонент програмних систем;

– принципи і застосування на практиці структурного та об’єктно-орієнтованого програмування.

вміти:

– описувати вказівники та виконувати основні операції з вказівниками;

– використовувати вказівники для роботи з динамічними одновимірними, багатовимірними та зубчастими масивами;

– представляти та обробляти текстові дані за допомогою рядків в C++, використовувати стандартні бібліотеки C++ для роботи з рядками;

– зберігати та обробляти дані гетерогенної структури за допомогою struct, застосовувати struct для представлення та обробки списків, черг, стеків та інших складних структур даних;

– використовувати засоби C++ для обробки файлів різних типів та структури;

– описувати та використовувати користувальницькі функції з урахуванням вимог до структури та функціональності програми;

– використовувати параметри функції main;

– створювати рекурсивні функції та визначати складність їх роботи (глибину рекурсії);

– структурувати програми із використанням функцій, застосовуючи принципи проектування програм «з гори до низу» та «з низу до гори»;

– створювати програми із застосуванням «заглушок» у функціях;

– розробляти архітектури, модулі та компоненти програмних систем;

– реалізовувати фази та ітерації життєвого циклу програмних систем та інформаційних технологій на основі відповідних моделей і підходів розробки програмного забезпечення;

– писати бібліотеки і функції;

– оптимізувати програмний код з метою задоволення заданим критеріям;

– розбиратися в чужому програмному коді;

– розвивати існуючі рішення на C++.

В другому семестрі дисципліна включає змістовні модулі:

– «Обробка складних структур даних» (частина 2);

– «Структурування програм C++»;

– «Робота з файлами та потоками»;

– «Основи об’єктно-орієнтованого підходу в розробці програм C++».

Навчальний процес організовується шляхом надання лекційного матеріалу та проведення лабораторних занять. На лекціях закладаються основи розуміння студентами базових підходів та засобів C++ до обробки складних структур даних та основ структурування програм, на лабораторних/практичних заняттях студенти набувають навички із застосування засобів C++ для вирішення задач обробки складних структур даних та побудови складних програм. Для самостійної роботи студентам разом з рекомендованою літературою пропонується користуватися електронними версіями навчально-методичних матеріалів, що представлені в електронній бібліотеці університету, а також можливостями мережі Інтернет.

Поточний контроль здійснюється під час проведення лабораторних/практичних занять і має на меті перевірку рівня підготовленості студента до виконання конкретної роботи. Поточний тестовий контроль може проводитись на аудиторних заняттях або в форматі самостійної роботи. Семестровий контроль проводиться у формі екзамену в обсязі навчального матеріалу, визначеного навчальною програмою, і в терміни, встановлені навчальним планом.

Підсумкова оцінка з дисципліни «Програмування С++» виставляється з урахуванням балів, отриманих за виконання лабораторних/практичних робіт, поточної роботи в семестрі, а також балів, які студент отримав на екзамені.

При самостійній роботі студенти набувають навички самостійного освоєння засобів С++ для вирішення задач обробки різних структур даних, які не використані в навчальному процесі, та поглиблюють свої знання щодо структурування програм С++ за допомогою функцій.

2 ОБРОБКА СКЛАДНИХ СТРУКТУР ДАНИХ

2.1 Поняття вказівника, призначення та особливості використання вказівників в С++

Вказівник — це тип даних, який використовується для зберігання адрес змінних і об'єктів. Змінна типу вказівник зберігає значення адреса комірки пам'яті. Формат опису змінної-вказівника:

тип * ім'я_змінної-вказівника;

Приклади опису змінних-вказівників:

```
int *ptri; //вказівник на змінну цілого типу
char *ptrc; //вказівник на змінну символьного типу
float *ptrf; //вказівник на змінну дійсного типу.
```

Змінні різних типів займають різну кількість комірок пам'яті та по різному інтерпретуються, однак самі змінні типу вказівник мають однаковий розмір. Якщо оголошується декілька змінних-вказівників на один тип, то * ставиться перед кожним ім’ям. Наприклад, дана інструкція дозволяє оголосити три змінних-вказівника на комірки цілого типу:

```
int *ptr1, *ptr2, *prt3 ;
```

В наступному прикладі змінна ptr1 є вказівником, а змінна ptr2 – звичайною змінною цілого типу:

```
int *ptr1, ptr2;
```

Для отримання адреси комірки можна використовувати операцію &. Наприклад,

```
int v = 1;
int* ptr = &v; // ptr містить адресу змінної v
```

Для отримання значення, яке розташовано за деякою адресою, використовується операція непрямої адресації (розіменування вказівника) *.

Операція * дозволяє звертатися до змінної не напряму, а через вказівник, який містить адресу цієї змінної. Ця операція є одномісною і має асоціативність зліва на право. Операцію не слід плутати з бінарною операцією множення. Наприклад,

```
int v;
int* ptr = &v; // ptr містить адресу змінної v
*ptr = 1; // в комірку, на яку вказує ptr, записується значення 1
```

При роботі з вказівниками можна використовувати операцію присвоювання – при цьому присвоюються адреси:

```
int *ptr1, *ptr2;
int v;
ptr1 = &v;
*ptr1 = 1;
ptr2 = ptr1; //обидва вказівника вказують на одну й ту ж комірку пам’яті
```

Завдання для самостійної роботи

Визначте значення змінної v після виконання вказаних операторів:

```
int *ptr1, *ptr2;
int v;
ptr1 = &v;
*ptr1 = 1;
ptr2 = ptr1;
*ptr2 = 10;
```

При роботі з вказівниками можна використовувати наступні операції:

– p + n, де p - вказівник, n - ціле позитивне число, результат - деякий вказівник, отриманий зміщенням p на n позицій вправо;

– p-n, де p - вказівник, n - ціле позитивне число, результат - деякий вказівник, отриманий зміщенням p на n позицій вліво;

– p-q, де p і q – вказівники на один і той же тип, результат - ціле число, що дорівнює кількості кроків, на яке потрібно змістити q вправо, щоб він досяг вказівника p, також цей результат можна називати «відстанню» між вказівниками, воно може бути і негативним, якщо елемент, на який спрямований вказівник q розташований правіше (тобто, далі), ніж елемент, на який спрямований вказівник p;

– p ++ (інкремент), p-- (декремент), де p – вказівник, p = p + 1, p = p-1 відповідно.

Інші операції заборонено!

Для виділення пам’яті під вказівники можна використовувати різні способи. Одним із них є застосування оператору new. Загальна форма оператору:

змінна-вказівник = new тип_змінної;

Наприклад:

```
int *a = new int; // Оголошення вказівника для змінної типу int
int *b = new float; // Оголошення вказівника для змінної типу float
```

Після застосування оператору new в динамічній пам’яті виділяється комірка відповідного типу. При використанні оператора new є можливість ініціалізації об'єкта:

змінна-вказівник = new тип_змінної (значення);

Наприклад:

```
int *a = new int(10); // Оголошення вказівника для змінної типу int з початковим значенням 10
```

Для вивільнення пам’яті, яка була виділена оператором new, використовується оператор delete. Формат використання оператора:

delete змінна-вказівник;

Наприклад:

delete a;

Контрольні питання

1. Що називається вказівником?
2. В чому відмінність вказівника від звичайної змінної?
3. Де розташовуються дані, на які вказують вказівники?
4. Які операції доступні над вказівниками?
5. Що є результатом додавання до вказівника числа? Віднімання числа?
6. Як виділити пам’ять під вказівник?
7. Як видалити пам’ять, на яку вказує вказівник?

2.2 Використання вказівників для роботи з динамічними масивами

При створенні будь-якого масиву в C ++ разом з ним створюється вказівник. Ім'я цього вказівника збігається з ім'ям масиву. Тип цього вказівника - «вказівник на базовий тип масиву». У цьому вказівнику зберігається адреса початкового елемента масиву. Щоб початок масиву не було втрачено, цей вказівник є константним, тобто його не можна направити на якийсь інший елемент масиву або записати туди адресу іншої змінної навіть відповідного типу, але можна скопіювати цю адресу в якийсь інший вказівник, який не є константним.

Для виділення пам’яті під масив за допомогою оператору new треба вказати розмір масиву:

змінна-вказівник = new тип_змінної [розмір_масиву];

Наприклад:

```
float *p = new float [10];
```

При виділенні пам’яті під масив одночасно ініціювати його не можна.

Приклад створення динамічного масиву та його заповнення числами від 1 до n:

```
#include <iostream>
using namespace std;
int main()
{
   int n; // розмір массиву
   cout << "Enter array size: ";
   cin >> n;

   int *a = new int[n]; // Виділення пам’яті під масив
   for (int i = 0; i < n; i++) {
      // Заповнення масиву і виведення значень його елементів
      a[i] = i+1;
      cout << "Value of " << i + 1 << " element is " << a[i] << endl;
   }
   delete [] a; // очищення пам'яті
   return 0;
}
```

З урахуванням того, що в масиві всі елементи розташовуються в пам'яті послідовно, почавши зі вказівника, спрямованого на початковий елемент, можна обійти всі елементи масиву, зміщуючи вказівник на кожному кроці вправо на мінімально можливу дистанцію (тобто, на сусідній праворуч елемент відповідного типу). Зсув вказівника може проводитися за допомогою операторів інкремента і декремента.

У наступному прикладі всі елементи масиву будуть виведені на екран без використання індексів (зверніть увагу, що при цьому параметри циклу

можуть бути будь-якими, головне, щоб цикл виконався потрібну кількість разів):

```
int a[] = {-15, 13, 21, -12, 5, 132};
int* p = a;
for (int i = 11; i <= 16; i++) {
        cout << *p << endl;
        p++;
}
```

Для звернення до елементів масиву також можна використовувати тотожність

$$a[i] == *(a + i),$$

де a – вказівник на масив будь-якого типу, i – допустимий індекс цього масиву. Таким чином, фрагменти 1 та 2 еквівалентні:

Фрагмент 1:

```
int a[] = {-15, 13, 21, -12, 5, 132};
for (int i = 0; i < 6; i++) {
        cout << * (a + i) << endl;
}
```

Фрагмент 2:

```
int a[] = {-15, 13, 21, -12, 5, 132};
for (int i = 0; i < 6; i++) {
        cout << a[i] << endl;
}
```

Вказівники можуть посилатися на інші вказівники. При цьому в комірках пам'яті, на які будуть посилатися перші вказівники, будуть міститися не значення, а адреси других вказівників. Число символів * при оголошенні вказівника показує порядок вказівника. Щоб отримати доступ до значення, на яке посилається вказівник його необхідно розіменовувати відповідну кількість разів. Логіка n-кратного розіменування полягає в тому, що програма послідовно перебирає адреси всіх вказівників аж до змінної, в якій міститься значення.

При роботі з двовимірними динамічними масивами спочатку об’являється вказівник другого порядку, який посилається на масив вказівників, а після цього необхідно виділити пам’ять під кожен одновимірний масив. В загальному випадку виділення пам’яті під двовимірний масив включає наступні оператори:

тип ***ім’я_масиву* = new *тип** [*кількість_рядків*];
for (int i = 0; i < *кількість_рядків*; i++)
ім’я_масиву [i] = new *тип* [*кількість стовпчиків в і-му рядку*];

Вивільнення пам'яті, що відведено під двовимірний динамічний масив, виконується в порядку, зворотному процесу виділення пам’яті:

for (int i = 0; i < *кількість_рядків*; i++)
delete [] *ім’я_масиву* [i];
delete [] *ім’я_масиву*;

Приклад виділення пам’яті під матрицю розміром 4x5, яка містить дійсні елементи:

```
//оголошення двовимірного динамічного масиву 4x5 елементів:
float **a = new float* [4]; // 4 рядки в масиві
   for (int i = 0; i < 4; i++)
      a[i] = new float [5]; // і 5 стовпчиків
```

//a – масив вказівників на виділену ділянку пам'яті під масив дійсних чисел типу float

Вивільнення пам'яті, що відведено під описаний двовимірний динамічний масив:

```
   for (int i = 0; i < 4; i++)
      delete [] a[i];
   delete [] a;
```

Використання динамічного способу виділення пам’яті дозволяє створювати зубчасті масиви – тобто в загальному випадку рядки двовимірного масиву не обов’язково повинні бути однакового розміру.

Контрольні питання

1. Який зв’язок між вказівниками та іменами масивів?
2. Опишіть виділення та вивільнення пам’яті під одновимірний масив.
3. Опишіть виділення та вивільнення пам’яті під двовимірний масив.
4. Що означає кількість * біля вказівника? Як отримати доступ до значень даних?
5. Що таке зубчастий масив? Як його створити?

2.3 Особливості представлення та обробки рядків в C++

2.3.1 Символьні масиви

Досі ми працювали лише з рядковими константами – послідовністю символів в подвійних лапках. Константні рядки нами використовувалися при виведенні на екран тієї чи іншої інформації:

```
cout << "Constant string";
```

Текст в лапках і є рядкова константа. Лапки використовуються для визначення початку і кінця рядкової константи, але не є її частиною. Досить часто необхідно не тільки друкувати якісь короткі повідомлення по ходу програми, а й працювати з певним текстом, зберігати його десь, звертатися до нього і редагувати, за потребою. До рядкової константи, розглянутої вище, ми не зможемо звернутися в програмі, наприклад для того, щоб перезаписати її, тому що ми не знаємо ані її імені, ані адреси в пам'яті.

В C++ для зберігання рядків використовують символьні масиви. Можна представити символи такого масиву розташованими послідовно в сусідніх комірках пам'яті (одновимірний масив) – в кожній комірці зберігається один символ, який займає один байт. Один байт тому, що кожен елемент символьного масиву має тип char. Останнім символом кожного такого рядка є символ '\0' (нульовий символ). Наприклад:

char s [6] = { 'П', 'Д', '-', '1', '1', '\0'};

Сам текст складається з 5-ти символів. Якби в останній комірці перебувала наприклад точка, а не нульовий символ '\0' - для компілятора це вже не був би рядок, і працювати з таким набором символів треба було б, як зі звичайним масивом – записувати дані в кожну комірку окремо і виводити на екран посимвольно (за допомогою циклу):

```
#include <iostream>
using namespace std;
 int main()
{
 setlocale(LC_ALL, "rus");
 char s[5] = {'П', 'Д', '-', '1', '1'};
 for (int i = 0; i < 5; i++){
        cout << s[i];
 }
 cout << endl;
  return 0;
}
```

В С++ є куди більш зручний спосіб ініціалізації і звернення до символьних масивів – рядки. Для цього останнім символом такого масиву обов'язково повинен бути нульовий символ '\0'.

Для об’явлення та ініціалізації масиву рядковою константою використовується наступна схема: створюємо масив типу char, розмір в квадратних дужках вказувати не обов'язково (його підрахує компілятор), далі пишемо оператор = і в подвійних лапках необхідний текст:

char s[] = {"ПД-11"};

Прописувати нульовий символ не треба. Він присутній неявно і додається в кожну таку рядкову константу автоматично. Таким чином, при тому що ми бачимо 5 символів в рядку, розмір масиву буде 6, так як '\0' теж

символ і займає один байт пам'яті. Займе він останню комірку цього символьного масиву.

Для виведення рядка на екран, досить звернутися до нього за іменем:

```
cout << s << endl;
```

cout<< буде виводити на екран символ за символом, поки не зустріне в одній з комірок масиву символ кінця рядка '\0' і виведення перерветься. Таке звернення для звичайного символьного масиву (масиву без '\0') неприпустиме.

Якщо рядок необхідно ввести з клавіатури, то необхідно об’явити масив типу char із зазначенням його розміру, достатнього для зберігання символів, що вводять, включаючи '\0':

```
#include <iostream>
using namespace std;
int main()
{
	char str[10] = "";
	cout << "Input string (length <=9)";
	cin >> str;
	cout <<"Input str: "<< str << endl;
	return 0;
} endl;
```

Використовуючи порожні лапки при ініціалізації, ми присвоюємо кожному елементу масиву значення '\0'. Таким чином рядок буде очищено від "сміття" інших програм. Навіть якщо користувач введе текст, що містить меншу кількість символів, наступний за текстом буде символ '\0'. Це дозволяє уникнути небажаних помилок.

Існують певні нюанси при введенні рядків. Якщо для введення використовувати оператор >>, то в рядок буде вміщено текст від початку до

першого пробілу. Для введення рідка з пробілами треба використовувати спеціальні функції, наприклад, getline():

```
#include <iostream>
using namespace std;
int main()
{
    char str[10] = "";
    cout << "Input string (length <=9)";
    cin.getline(str, 10);
    cout <<"Input str: "<< str << endl;
    return 0;
} endl;
```

В дужках для функції вказано два аргументи – в який масив вводити символи (ім'я масиву str) і розмір цього масиву (10). cin.getline() зчитує в масив весь рядок з пробілами і табуляціями, поки не відбудеться натискання Enter або поки не буде перевищено розмір масиву. Символ нового рядка в масиві не збережеться, а заміниться на нульовий символ.

Контрольні питання

1. Який тип мають елементи рядка?
2. Чим відрізняється рядок від звичайного масиву символів?
3. Як ініціювати рядок константним значенням?
4. Що означає в рядках символ '\0'?
5. Як ініціюється рядок при присвоєнні значення ""?
6. Як ввести рядок, що містить пробільні символи (пробіл чи табуляцію)?

2.3.2 Функції роботи з рядками

У С ++ для роботи з рядками існує багато функцій. Стандартні функції для роботи з рядками розміщуються в бібліотеці cstring (або string.h - в

залежності від компілятору). В цьому пункті представлено найчастіше використані функції для роботи з рядками.

Завдання для самостійної роботи

Складіть програми із використанням наведених функцій, та визначте особливості і обмеження їх роботи.

strlen(им'я_рядка);

Повертає довжину рядка (без урахування нуль-термінального символу)

strcpy(s1,s2)

Побайтне копіювання символів рядка s2 в s1

Завдання: перевірити роботу функції при різних розмірах вхідних рядків (s1>s2, s2>s1, s1=s2)

strncpy(s1,s2, n)

Побайтне копіювання n символів рядка s2 в рядок s1. Результат роботи функції – вказівник на рядок s1

Завдання: перевірити роботу функції при критичних розмірах вхідних рядків та значеннях n

strcat(s1,s2)

Об'єднання рядків s1 та s2 (конкатенація, склеювання). Результат зберігається в s1

Завдання: перевірити роботу функції при критичних розмірах вхідних рядків

strncat(s1,s2,n)

Об'єднання рядка s1 та n символів рядка s2 (конкатенація, склеювання). Результат зберігається в s1

Завдання: перевірити роботу функції при критичних розмірах вхідних рядків та значеннях n

strcmp(s1,s2)

Порівнює рядок s1 з рядком s2 і повертає результат типу int: 0 -якщо рядки еквівалентні,> 0 - якщо s1 <s2, <0 - якщо s1> s2 З урахуванням регістру

Завдання: перевірити роботу функції при різних розмірах та значеннях вхідних рядків

strncmp(s1,s2,n)

Порівнює n символів рядка s1 з рядком s2 і повертає результат типу int: 0 -якщо рядки еквівалентні,> 0 - якщо s1 <s2, <0 - якщо s1> s2 з урахуванням регістру

Завдання: перевірити роботу функції при різних розмірах та значеннях вхідних рядків і значеннях n

stricmp(s1,s2) – працює аналогічно **strcmp(s1,s2),** але без урахування регістру

Завдання: перевірити роботу функції при різних розмірах та значеннях вхідних рядків

strnicmp(s1,s2,n) – працює аналогічно **strncmp(s1,s2,n),** але без урахування регістру

Завдання: перевірити роботу функції при різних розмірах та значеннях вхідних рядків і значеннях n

strchr(s,c)

Пошук першого входження символу з в рядку s. У разі вдалого пошуку повертає вказівник на місце першого входження символу c. Якщо символ не знайдено, то повертається NULL.

Завдання: перевірити роботу функції при різних значеннях вхідного рядка s та символу c

strstr(s1, s2)

Функція шукає перше входження рядка s2 в рядку s1. Повертає вказівник на перше входження рядка s2 в рядок s1, або порожній вказівник, якщо рядок s2 не є частиною рядка s1. У даному пошуку нуль-термінальний символ не враховується.

Завдання: перевірити роботу функції при різних значеннях вхідних рядків

strcspn(s1,s2)

Визначає довжину початкового сегменту рядка s1, що містить ті символи, які не входять в рядок s2

Завдання: перевірити роботу функції при різних значеннях вхідних рядків

strspn(s1,s2)

Повертає довжину початкового сегмента рядка s1, що містить тільки ті символи, які входять в рядок s2

Завдання: перевірити роботу функції при різних значеннях вхідних рядків

strprbk(s1,s2)

Повертає вказівник першого входження будь-якого символу рядка s2 в рядку s1

Завдання: перевірити роботу функції при різних значеннях вхідних рядків

2.3.3 Задачі на обробку рядків

Якщо умовами задачі не задано іншого, слід вважати роздільником між словами один чи декілька пробілів.

1. Дано рядок. Розрахувати кількість знаків пунктуації в рядку. Перелік знаків пунктуації визначити самостійно (не менше шести).
2. Дано рядок. Визначити, чого в ній більше – літер латинського алфавіту чи цифр.
3. Дано рядок. Замінити в ньому пробіли на коми.
4. Дано рядок. Визначити, скільки в ньому великих літер латинського алфавіту.
5. Дано рядок. Вивести номери символів, які не є літераи латинського алфавіту.
6. Дано рядок. Видалити з нього всі цифри.
7. Дано рядок. Видалити з неї усі символи, які не є цифрами.
8. Дано два рядки. Визначити, чи співпадають вони.
9. Дано рядок, який містить одне слово. Визначити, чи є вона паліндромом.
10. Дано рядок. Визначити кількість слів в рядку.
11. Дано рядок. Замінити кожен знак + на два знаки ++.
12. Дано рядок. Видалити з неї всі однакові символи, які ідуть підряд. Наприклад: вхідний рядок ‘aaabbbcdefgggabc’, результат ‘cdefabc’.
13. Дано рядок. Підрахувати кількість слів, які мають задану довжину N.
14. Дано рядок. Видалити з нього всі пробіли.
15. Дано рядок. Дописати після кожної коми пробіл.
16. Дано рядок. Видалити з неї символи, які стоять на парних позиціях.

17. Дано рядок. Продублювати в ньому кожен непарний символ.
18. Дано рядок. Визначити ккількість слів, які починаються із заданого символу.
19. Видалити з рядка частину, обмежену дужками разом із дужками.
20. Дано рядок, в якому міститься одна крапка з комою (;). Підрахувати кількість символів до неї та після неї.
21. Дано рядок, в якому слова розділено одним чи декількома пробілами. Перетворити рядок, залишивши між словами лише по одному пробілу.
22. Заданий набір слів, між якими ставиться крапка з комою (;). Визначити, скільки в ньому слів, що закінчуються буквою а.
23. Заданий рядок. Вивести слова, які містять хоча б одну букву «к».
24. Заданий рядок. Вивести слова, які починаються і закінчуються одним і тим же символом.
25. У записці слова зашифровані – кожне з них записано навпаки. Розшифрувати повідомлення.
26. Заданий текст. Скласти новий текст, що містить тільки останні символи всіх слів.
27. Заданий рядок. Вивести слова, які не містять символ тире.
28. Заданий рядок. Визначити, скільки слів мають таку ж саму довжину, що й перше слово.
29. Заданий рядок. Вивести слова, які містять приставку «над».
30. Заданий рядок, який містить текст англійською мовою. Обчислити кількість слів, які мають закінчення «ing».
31. Заданий рядок. Визначити, чи є в ньому хоча б одне слово довжиною N.
32. В заданому рядку замінити кожне входження цифри 1 на слово «один».

33. Заданий рядок. Сформувати новий рядок з першого та останнього слів вхідного рядка. Якщо вхідний рядок містить тільки одне слово, то у вихідному рядку слід продублювали його.
34. Заданий рядок. Сформувати з нього два нових рядка: у перший помістити символи першого рядка, які є літерами англійського алфавіту, в другий усі цифри.
35. Заданий рядок. Визначити, скільки в ньому слів, довжина яких парна.
36. Заданий рядок. Вивести перше зліва слово, яке має довжину N.
37. Заданий рядок. Вивести слово максимальної довжини.
38. Заданий рядок. Вивести слово мінімальної довжини.
39. Заданий рядок. Створити новий рядок, в який записати всі символи вхідного рядка, які не є літерами латинського алфавіту.
40. Заданий рядок. Замінити перше слово в рядку на символи *.
41. Заданий рядок. Замінити всі голосні літери на символ *.
42. Заданий рядок. Визначити позиції початку та кінця другого слова в рядку.
43. Заданий рядок. створити новий рядок, переписавши туди перші 3 слова вхідного рядка.
44. Заданий рядок. Роздільниками між словами вважаються наступні знаки пунктуації (крапка; крапка з комою та інші). Обчислити кількість таких слів.

2.4 Поняття структури (struct) в C++, особливості обробки структур, складні структури даних на основі struct

До цього ми використовували стандартні (вбудовані) типи даних для змінних в програмах (int, float, char, bool). Для обробки даних, які мають складну структуру, в C++ використовуються структури struct.

Структури використовуються для представлення об’єктів, які мають певний логічний зв’язок між елементами. Структура в загальному випадку –

це сукупність різнотипних елементів, яким присвоюється одне ім’я (воно може бути відсутнім), що займає одну ділянку пам’яті. Елементи, що складають структуру, називаються полями. Формат опису структури:

struct *ім’я_типу*

{

об’явлення змінних, що входять в структуру

} *список_змінних*;

При об’явленні структури можна вказувати *ім’я_типу*, за допомогою якого можна потім в програмі створювати структури. *список_змінних* визначає змінні, які створюються при об’явленні структури. Хоча б один із зазначених елементів обов’язково необхідно вказати при об’явленні структури.

Після того, як структура визначена, можна створювати змінні типу структури - об'єкти структури. Наприклад:

```
struct student

{ char pip [25];          // ПІП студента
  float avg;               // середній бал
}  s1, s2;
```

Змінні s1 і s2 можна оголосити окремим оператором, наприклад:

```
struct student sl, s2;
```

Ініціювання полів структури слід здійснювати або при її описі, або в тілі програми. При описі структури ініціювання полів виглядає, наприклад, так:

```
sl = { "Петренко П. П.", 3.5};
```

Для звернення до полів структури використовується точка, яку необхідно ставити після імені змінної:

ім’я_змінної.ім’я_поля

Слід зазначити, що елементом структури може бути також структура.

Якщо поле структури описано, як вказівник, то для звернення до його значення замість точки використовується ->.

Змінні типу структура можна використовувати як елементи інших, більш складних структур даних, наприклад, масивів, списків, черг тощо. Динамічний список представляє деяку кількість компонентів (вузлів), що містять безпосередньо інформаційну частину (число, рядок або більш складні типи даних), а також посилання на наступний компонент (можливо, що вузол містить 2 посилання: на наступний і на попередній, в такому випадку, список називається двозв'язним).

Для опису елементу списку використовується структура наступного вигляду:

```
struct typeID {
        інформаційні_поля;
        typeID * next; //посилання на наступний елемент списку
};
```

next – ім'я поля структури, яке використовується для зберігання адреси наступного елементу.

Перший елемент списку зазвичай називають головою (head). Для створення пустого списку необхідно присвоїти голові списку значення NULL. Операції з компонентами списку залежать від того, яку організацію має структура даних, яка представлена списком. Загальні операції: додавання елементу в голову/в кінець/в середину списку, видалення елементу з голови/з кінця/з середини списку, перегляд всіх елементів або окремого елементу списку.

Контрольні питання

1. Для яких даних використовуються структури?
2. Який формат має опис структури?
3. Яким чином відбувається доступ до полів структури?

4. В яких випадках використовується точка, а в яких знак -> при роботі зі структурою?
5. Чи можна використовувати в якості полів інші структури? Дане типу самої структури? Вказівник на тип самої структури?
6. Яким чином треба звертатися до полів в масиві структур? До елементу масиву, який є полем структури?
7. Як описати елемент однозв’язного списку за допомогою структур?
8. Які операції використовуються зі списковими структурами?

2.5 Поняття файлу та потоку, класифікація та особливості обробки файлів в C++

2.5.1 Загальна інформація про файли та потоки в C++

Файл – це конкретний пристрій введення/виведення. **Потік** – це узагальнений пристрій введення/виведення, який підтримує єдиний інтерфейс, що не залежить від того, до якого конкретного пристрою здійснюється доступ.

Всі потоки поводяться схожим чином. Оскільки вони не залежать від фізичних пристроїв, то та ж функція, яка виконує запис в дисковий файл, може ту ж операцію виконувати і на іншому пристрої, наприклад, на консолі.

Види потоків:

- текстові;
- двійкові (бінарні).

Текстовий потік – це послідовність символів. (розмір символу 1 байт).

У стандарті С вважається, що текстовий потік організований у вигляді рядків, кожен з яких закінчується символом нового рядка. Однак в кінці останнього рядка цей символ не є обов'язковим. У текстовому потоці на вимогу базової середовища можуть відбуватися певні перетворення символів. Наприклад, символ нового рядка може бути замінений парою символів - повернення каретки і переведення рядка. Тому може і не бути однозначної

відповідності між символами, які пишуться (читаються), і тими, які зберігаються в зовнішньому пристрої. Крім того, кількість тих символів, які пишуться (читаються), і тих, які зберігаються в зовнішньому пристрої, може також не збігатися завдяки можливим перетворенням.

Двійковий (бінарний) потік – це послідовність байтів, яка взаємно однозначно відповідає байтам на зовнішньому пристрої, причому ніякого перетворення символів не відбувається. Крім того, кількість тих байтів, які пишуться (читаються), і тих, які зберігаються на зовнішніх пристроях, однакова. Однак в кінці двійкового потоку може додаватися декілька нульових байтів (це визначається додатком). Такі нульові байти, наприклад, можуть використовуватися для заповнення вільного місця в блоці пам'яті незначної інформацією, щоб вона в точності заповнила сектор на диску.

2.5.2 Відмінність файлів та потоків в C++

У мові С (С++) файлом може бути все що завгодно, починаючи з дискового файлу і закінчуючи терміналом або принтером. Потік пов'язують з певним файлом, виконуючи ***операцію відкриття***. Як тільки файл відкритий, можна проводити обмін інформацією між ним і програмою.

Не всі файли мають однакові можливості. Наприклад, до дискового файлу прямий доступ можливий, в той час як до деяких принтерів – ні. Таким чином, ***всі потоки однакові, а файли – ні***.

Якщо файл може підтримувати ***запити на місце розташування*** (вказівник поточної позиції), то при відкритті такого файлу вказівник поточної позиції у файлі встановлюється в початок. При читанні з файлу кожного символу (або запису в нього) вказівник поточної позиції збільшується, забезпечуючи тим самим просування по файлу.

Файл від'єднується від певного потоку (тобто розривається зв'язок між файлом і потоком) за допомогою ***операції закриття***. При закритті файлу, відкритого з метою виведення, вміст (якщо воно є) пов'язаного з ним потоку записується на зовнішній пристрій. Цей процес, який зазвичай називають

дозапис потоку, гарантує, що ніяка інформація випадково не залишиться в буфері диску. Якщо програма завершує роботу нормально, тобто або main () повертає керування операційній системі, або викликається exit (), то всі файли закриваються автоматично. У разі аварійного завершення роботи програми, наприклад, в разі краху або завершення шляхом виклику abort (), файли не закриваються.

2.5.3 Структура FILE. Операції з файлами бібліотеки stdio.h

У кожного потоку, пов'язаного з файлом, є керуюча структура, яка містить інформацію про файл; вона має тип **FILE**. Введення або виведення від кожного пристрою автоматично перетворюється системою введення/виведення в потік.

Вказівник файлу –це вказівник на структуру типу FILE. Він вказує на структуру, яка містить різні відомості про файл, наприклад, його ім'я, статус і покажчик поточної позиції в початок файлу. По суті, вказівник файлу визначає конкретний файл і використовується відповідним потоком при виконанні функцій введення / виводу. Щоб виконувати в файлах операції читання і запису, програми повинні використовувати вказівники відповідних файлів.

В таблиці 2.1 наведені основні операції в C (C++) для роботи з файлами, які описані за допомогою структури FILE.

Таблиця 2.1 – основні операції для роботи з файлами

fopen()	Відкриває файл
fclose()	Закриває файл
putc()	Записує символ в файл
fputc()	То ж, що і putc (), тільки з файлами
getc()	Читає символ з файлу
fgetc()	То ж, що і getc (), тільки з файлами
fgets()	Читає рядок з файлу

fputs()	Записує рядок в файл
fseek()	Встановлює вказівник поточної позиції на певний байт файлу
ftell()	Повертає поточне значення вказівника поточної позиції у файлі
fprintf()	Для файлу той же, що printf () для консолі
fscanf()	Для файлу той же, що scanf () для консолі
feof()	Повертає значення true (істина), якщо досягнуто кінець файлу
ferror()	Повертає значення true, якщо сталася помилка
rewind()	Встановлює вказівник поточної позиції в початок файлу
remove()	Стирає файл
fflush()	Дозапис потоку в файл

Порядок роботи з файлом, що описаний структурою FILE:

1. Об’явлення змінної-вказівника файлу:

```
FILE *ім’я_змінної;
```

2. Відкриття файлу для виконання операцій. Спосіб відкриття залежить від майбутніх операцій.
3. Закриття файлу.

Відкриття файлу.

Функція fopen () відкриває потік і пов'язує з цим потоком певний файл. Потім вона повертає вказівник цього файлу. Найчастіше під файлом мається на увазі дисковий файл.

Прототип функції fopen ():

```
FILE *fopen(const char *ім’я_файлу, const char *режим);
```

Імя_файлу – це вказівник на рядок символів, що представляє собою допустиме ім'я файлу, в яке також може входити специфікація шляху до

цього файлу. Рядок, на який вказує режим, визначає, яким чином файл буде відкритий.

У таблиці 2.2 показано, які значення рядка ***режим*** є допустимими. Рядки, подібні "r + b" можуть бути представлені і у вигляді "rb +".

Таблиця 2.2 – Режими відкриття файлу

R	Відкрити текстовий файл для читання
W	Створити текстовий файл для запису
A	Додати в кінець текстового файлу
Rb	Відкрити двійковий файл для читання
Wb	Створити двійковий файл для запису
Ab	Додати в кінець файлу
r+	Відкрити текстовий файл для читання / запису
w+	Створити текстовий файл для запису / читання
a+	Додати в кінець текстового файлу або створити текстовий файл для читання / запису
r+b	Відкрити двійковий файл для читання / запису
w+b	Створити двійковий файл для читання / запису
a+b	Додати в кінець файлу або створити бінарний файл для читання / запису

Функція fopen () повертає вказівник файлу. Якщо при відкритті файлу відбувається помилка, то fopen () повертає порожній (null) вказівник.

Приклад відкриття файлу:

```
FILE *fp;
fp = fopen("test", "w");
```

Або

```
FILE *fp;
```

```
if ((fp = fopen("test","w"))==NULL) {
  cout<<"File open error\n";
  exit(1);
}
```

В наведених прикладах файл з іменем test відкривається для запису (параметр "w" функції fopen). Змінна fp у випадку успішного відкриття файлу буде вказувати на перший байт файлу test.

Закриття файлу.

Функція fclose () закриває потік, який був відкритий за допомогою виклику fopen (). Функція fclose () записує в файл всі дані, які ще залишалися в дисковому буфері, і проводить, так би мовити, офіційне закриття файлу на рівні операційної системи. Відмова при закритті потоку тягне всілякі неприємності, включаючи з втрати даних, зіпсованих файлів і можливих періодичних помилок в програмі. Функція fclose () також звільняє блок управління файлом, пов'язаний з цим потоком, даючи можливість використовувати цей блок знову. Так як кількість одночасно відкритих файлів обмежена, то, можливо, доведеться закривати один файл, перш ніж відкривати інший.

Прототип функції fclose ()

```
int fclose(FILE *вказівник_на_файл);
```

Повернення нуля означає успішну операцію закриття. У разі ж помилки повертається EOF. Щоб точно дізнатися, в чому причина цієї помилки, можна використовувати стандартну функцію ferror (). Зазвичай відмова при виконанні fclose () відбувається тільки тоді, коли з якихось причин зникає доступ до диску (наприклад, передчасне видалення) або на диску не залишилося вільного місця.

2.3 Файлове введення/виведення з використанням потоків

Для роботи з файлами через потоки використовується бібліотека потокового введення-виведення **fstream**:

- ofstream визначає потік введення;
- ifstream визначає потік виведення.

Послідовність операцій з потоками така ж сама, як і при роботі зі структурою FILE.

1. Об’явлення змінної, пов’язаної з потоком відповідного типу. Тип потоку залежить від майбутніх операцій.
2. Відкриття потоку для виконання операцій (зв’язування конкретного файлу з визначеним потоком).
3. Закриття потоку (розривання зв’язку файлу з потоком).

Відкриття потоку:

```
F.open(«file», mode);
```

F – змінна, описана як ofstream, file – повне ім'я файлу на диску, mode – режим роботи з файлом. Зверніть увагу на те, що при вказанні повного імені файлу на диску потрібно ставити подвійний слеш замість одинарного.

Режими відкриття файлових потоків наведено в таблиці 2.3.

Таблиця 2.3 – Режими відкриття файлових потоків

ios :: in	відкрити файл в режимі читання даних; режим є режимом за замовчуванням для потоків ifstream
ios :: out	відкрити файл в режимі запису даних (при цьому інформація про існуючий файлі знищується); режим є режимом за замовчуванням для потоків ofstream;
ios :: app	відкрити файл в режимі запису даних в кінець файлу
ios :: ate	пересунутися в кінець вже відкритого файлу
ios :: trunc	очистити файл, це ж відбувається в режимі ios :: out
ios :: nocreate	не виконувати операцію відкриття файлу, якщо він не існує
ios :: noreplace	не відкривати існуючий файл
ios::binary	відкрити файл, як бінарний

Параметр mode може бути відсутнім, в цьому випадку файл відкривається в режимі за замовчуванням для даного потоку.

Після вдалого відкриття файлу (в будь-якому режимі) в змінній F буде зберігатися true, в іншому випадку false. Це дозволить перевірити коректність операції відкриття файлу.

Операції з потоком

Після відкриття файлу в режимі запису, в нього можна писати точно так же, як і на екран, тільки замість стандартного пристрою виведення **cout** необхідно вказати ім'я відкритого файлу.

Після відкриття файлу в режимі читання, з нього можна читати так же, як і з клавіатури, вказуючи замість стандартного пристрою виведення cin ім'я відкритого файлу.

Для визначення того, закінчився потік чи ні, використовується функція **F.eof(),** яка повертає логічне значення: true або false, залежно від того чи досягнуто кінець файлу.

Закриття потоку

Закриття потоку здійснюється за допомогою оператора:

```
F.close ();
```

2.4 Приклад роботи з текстовим файлом

Програма створює текстовий файл, записує в нього кілька чисел і потім виводить числа з файлу через крапку з комою.

```
#include<iostream>
#include<fstream> //бібліотека для роботи с файлом

using namespace std;

void main()
{
```

```
ifstream in; //вхідний потік
ofstream out;//вихідний потік
out.open("test.txt"); // відкриття файлу test.txt як вихідного потоку
int x = 10;
for (int i = 0; i < 5; i++)
{
	out << x << "    "; //запис числа x у вихідний потік, числа записуються через пробіл
	x += 5;
}
out.close();// закриття вихідного потоку
in.open("test.txt"); // відкриття файлу test.txt як вхідного потоку
while (!in.eof()) //обробляємо вхідний потік, доки він не скінчиться
{
	in >> x; //зчитуємо значення з вхідного потоку
	cout << x << "; ";
}

in.close();// закриваємо вхідний потік
system("pause");
}
```

2.5 Робота з бінарними файлами

Для того, щоб правильно читати дані з файлів, що зберігають дані в бінарному вигляді, потрібно знати загальну структуру зберігання всередині файлу.

Коли ми говоримо про бінарні дані, **тип char виступає в ролі типу byte**.

Щоб ми могли записати якесь значення в бінарному представленні, нам потрібно для початку вивести це бінарне представлення, а щоб записалася правильна кількість байт, потрібно явно вказувати цю кількість.

2.5.1 Приклад роботи з бінарним файлом

Бінарний файл 1.txt містить ціле (int) та дійсне (double) число. Вивести вміст файлу на екран.

```
#include <fstream>      //бібліотека для роботи с файлом
#include <iostream>

using namespace std;

int main() {
  const char* FName = "C:\\1.txt";      //Шлях до файлу на диску

  int x = 0;                            //Змінні, які виступають буфером для читання з файлу
  double y = 0;

   /*Початок роботи з файлом*/
  ifstream in(FName,ios::binary); // відкриваємо файл для читання як бінарний
     in.read((char*)&x, sizeof(x));          //перенос байтів із файлу в буферну змінну x
     in.read((char*)&y, sizeof(y));          //перенос байтів із файлу в буферну змінну y
  in.close();
  /*Кінець роботи с файлом*/

   cout << x << '\n' << y << '\n';//виведення вмісту буферних змінних
   cin.get();
}
```

Для визначення кількості зчитаних/записаних байт можна використовувати конструкцію (char*)&x, sizeof(x), де x – буферна змінна.

Завдання для самостійної роботи

Використовуючи опис структури файлу bmp, створити чорний квадрат заданого розміру. Опис структури файлу:

Описи структури елементів файлу можна знайти в Інтернеті.

Контрольні питання

1. Дайте поняття файлу та потоку. Чим вони відрізняються?
2. Які типи файлів існують? Чим вони відрізняються за способом доступа? За типом та структурою елементу?
3. Вкажіть операції з текстовими файлами.
4. Вкажіть операції із файлами прямого доступу.
5. Наведіть приклади функцій для роботи з текстовими файлами.
6. Наведіть приклади функцій для роботи з бінарними файлами.

3 СТРУКТУРУВАННЯ ПРОГРАМ

3.1 Поняття функції в С++, особливості передачі параметрів в функції, параметри функції main

Підпрограма є важливим елементом структурного програмування. Спочатку підпрограми з'явилися як засіб оптимізації програм за обсягом займаної пам'яті – вони дозволили не повторювати в програмі ідентичні блоки коду, а описувати їх одноразово і викликати в міру необхідності. До теперішнього часу дана функція підпрограм стала допоміжної, головне їх призначення – ***структуризація програми з метою зручності її розуміння і супроводу.***

Виділення набору дій в підпрограму і виклик її в міру необхідності дозволяє логічно виділити цілісну підзадачу, що має типове рішення. Така дія має ще одну (крім економії пам'яті) перевагу перед повторенням однотипних дій. Будь-яка зміна (виправлення помилки, оптимізація, розширення функціональності), зроблена в підпрограмі, автоматично відбивається на всіх її виклики, в той час як при дублюванні кожну зміну необхідно вносити в кожне входження змінюваного коду. Навіть в тих випадках, коли в підпрограму виділяється одноразовий набір дій, це виправдано, оскільки дозволяє скоротити розміри цілісних блоків коду, що складають програму, тобто зробити програму більш зрозумілою і доступною для огляду.

Підпрограми в С++ реалізуються у вигляді функцій. Опис функції має такий узагальнений вигляд:

```
тип_даних        ім'я_функції( список_формальних_параметрів )
{
        оператори_тіла_функції;
}
```

Список формальних параметрів у загальному випадку складається зі списку вхідних і вихідних даних. Тобто, якщо потрібно передавати функції якісь дані, то всередині круглих дужок оголошуються параметри функції, які відокремлюються один від одного комами.

Якщо функція в явному вигляді не повертає нічого, то це описується службовим словом void. void - це тип даних, який не може зберігати будь-які дані. void ніяк по-іншому не використовується і потрібен тільки для того, щоб компілятор міг визначити тип функції:

```
void   ім'я_функції( список_формальних_параметрів)
{
        оператори_тіла_функції;
}
```

Якщо функція повертає якийсь результат, то це визначається типом, описаним в заголовку функції перед іменем функції. При цьому всередині тіла функції обов'язково повинен бути оператор return:

```
тип   ім'я_функції(список_формальних_параметрів)
{
        оператори_тіла_функції;
        return значення_що_повертається;
}
```

Виклик функції здійснюється наступним чином:

```
ім'я_функції ( список_фактичних_параметрів)
```

Способи об'явлення функцій в С++:

1. В одному файлі з main:

– перед main;

– після main.

2. В окремому файлі:

– файл *.cpp;

– файл *.cpp, *h.

Якщо функція об’являється перед main, то вона описується звичайним способом – заголовок+тіло. Якщо після main, то перед main необхідно вказати прототип функції, а її реалізацію навести після функції main. Прототипом називають заголовок функції, який завершується крапкою з комою. Прототип може не містити імен параметрів, а тільки опис їх типів та способів передачі. Порядок та типи параметрів в прототипі та подальшому описі функції повинні бути однаковими.

Передача інформації у функцію здійснюється наступними способами:

– за значенням;

– через вказівник;

– через посилання.

При передачі за значенням передається копія параметру, зміна значення параметру ніяк не впливає на значення змінних у блоці, в якому функція була викликана. Синтаксис параметру-значення:

тип ім’я_параметру

Таким способом можна передавати як змінні, так і константні значення. Приклад функції, яка приймає цілочисельний параметр:

```
void f1( int a)
{
        ...
        a=10;
        …
}
```

Приклади викликів функції f1:

```
int x=3;
```

```
f1(x);
…
f1(5);
```

В жодному з випадків будь-які дії з параметром всередині функції не відобразяться на даних, які було передано у функцію.

При передачі через вказівник, у функцію передається адреса змінної або іншого об’єкту. В цьому випадку окрема пам’ять для параметру не виділяється, що корисно, наприклад, при обробці великих обсягів даних. Будь-які зміни значення параметру призводять до зміни об’єкту в блоці виклику, оскільки вони використовують спільну пам'ять. Синтаксис параметру-вказівника:

*тип * ім’я_параметру*

Через вказівники можна передавати лише адреси змінних, константні значення таким способом передавати не можна. При зверненні до значення, що зберігається за адресою відповідного параметру, в тілі функції необхідно використовувати оператор розіменування.

Приклад функції, яка приймає цілочисельний параметр-вказівник:

```
void f2( int * a)
{
        ...
        *a=10;
        …
}
```

Приклади викликів функції f2:

```
int x=3;
```

f2(&x); //зміна a всередині функції призведе до відповідної зміни x, оскільки і a, і x вказують на одну й ту ж комірку пам’яті

```
…
f2(5);//такий виклик є синтаксично невірним!!!
…
int *p = new int (0);
f2(p);
```

При передачі через посилання & створюється нове посилання на об’єкт, що виступає параметром функції. Синтаксис параметру-посилання:

тип & ім’я_параметру

Так само, як і у випадку з вказівниками, такий параметр можна змінювати всередині функції і всі зміни будуть насправді відбуватися з об’єктом, який є фактичним параметром. Однак, на відміну від використання вказівника, параметр, переданий за посиланням, можна використовувати як звичайну змінну, без застосування операції розіменування.

Приклад функції, яка приймає цілочисельний параметр-посилання:

```
void f3( int & a)
{
        ...
        a=10;
        …
}
```

Приклади викликів функції f3:

```
int x=3;
f2(x); //зміна a всередині функції призведе до відповідної зміни x
…
f2(5);//такий виклик є синтаксично невірним, оскільки неможливо оперувати адресою константи
```

Коли в якості аргументу функції використовується масив, то передається тільки адреса масиву, а не копія всього масиву. При виконанні функції з ім'ям масиву в функцію передається вказівник на перший елемент масиву. Параметр повинен мати тип, сумісний з вказівником. Існує три способи об’явлення параметра, призначеного для отримання вказівника на масив:

тип ім’я_масиву[розмір]

тип ім’я_масиву[]

*тип *ім’я_масиву*

Усі три методи оголошення параметра призводять до однакового результату – створення вказівника. Якщо масив не змінюється всередині функції, то перед типом масиву вказується слово *const*.

Змінна, оголошена всередині тіла функції, називається локальною змінною. Всі змінні, які описані в заголовку функції та в межах її тіла, є локальними та доступні лише в цій функції. Поза межами функції вони недоступні. Локальні змінні зникають з області видимості при виході з функції. Змінні, що описані у блоці, зовнішньому до блоку виклику та блоку опису функції, доступні так само, як і звичайні змінні тіла функції. Однак такий канал обміну інформацією дуже важко контролювати, тому використання подібних змінних у сучасних підходах до програмування не вітається.

У С ++ локальні змінні можна оголошувати як статичні (static). При цьому, хоча змінна є видимою тільки в тілі функції, однак для всіх примірників функції існує тільки одна копія змінної.

Завдання для самостійної роботи

Описати функції для вирішення наступних завдань:

- дано 2 цілих числа, знайти їх найбільший спільний дільник;
- дано 2 цілих числа, знайти їх найменше спільне кратне;
- дано 2 дійсних числа, визначити більше з них по модулю;

– дано 2 дійсних числа, визначити число, в якому дрібна частина менше;

– дано одновимірний масив дійсних чисел, обчислити суму елементів, що не дорівнюють нулю;

– дані координати точки в двовимірному просторі, визначити відстань точки до початку координат;

– дано одновимірний масив дійсних чисел, знайти номер першого за рахунком елемента, який дорівнює заданому числу X;

– дано одновимірний масив дійсних чисел, знайти суму і добуток його елементів;

– дано одновимірний масив дійсних чисел, знайти добуток його від'ємних елементів;

– дано одновимірний масив дійсних чисел, знайти суму його елементів в діапазоні [a; b];

– дано рядок, порахувати кількість слів рядка, що не містять заданий символ;

– дано рядок, порахувати кількість слів рядка, що закінчуються на заданий символ;

– дано рядок, визначити позицію, в якій починається перше слово рядка;

– дано рядок, порахувати кількість слів рядка, що починаються на заданий символ;

– дано рядок, порахувати кількість слів рядка, що містять заданий символ.

Контрольні питання

1. Що таке функція? Для чого призначені функції в програмі?
2. Що оформлюється у вигляді функції?
3. Який синтаксис має функція, що не повертає значення в явному вигляді?

4. Який синтаксис має функція, яка повертає в явному вигляді якесь значення?
5. Яким чином відбувається виклик функції?
6. Опишіть особливості передачі параметрів за значенням. Що можна передавати в якості параметра-значення?
7. Опишіть особливості передачі параметрів за вказівником. Що можна передавати в якості параметра-вказівника?
8. Опишіть особливості передачі параметрів за посиланням. Що можна передавати в якості параметра-посилання?
9. Як передати у функцію масив?
10. Яка видимість змінних відносно функції?

3.2 Поняття рекурсії, особливості використання рекурсивних функцій

Визначення функцій не можуть бути вкладеними, тобто не можна всередині тіла однієї функції визначити тіло іншої. Однак, можна викликати одну функцію з іншої. У тому числі функція може викликати сама себе.

Розглянемо функцію обчислення факторіала цілого числа. Її можна реалізувати двома способами. Перший спосіб використовує ітерацію:

```
int fact(int n)
{
    int result = 1;
    for (int i = 1; i <= n; i++)
        result = result * i;
    return result;
}
```

Другий спосіб (рекурсія):

```
int fact(int n)
{
  if (n = = 1)     // факторіал 1 дорівнює 1
```

```
        return 1;
    else                    // факторіал числа n дорівнює факторіалу n-1 помноженому на n
        return n * fact(n -1);
}
```

Рекурсивна функція - це функція, яка викликає саму себе (пряма рекурсія). Непряма рекурсія - коли дві або більше функцій викликають одна одну. Коли функція викликає себе, в стеку створюється копія значень її параметрів, після чого управління передається першому оператору функції. При повторному виклику процес повторюється. Базис рекурсії складають наступні положення:

– функція містить пряму або непряму рекурсію (безпосередній виклик самої себе, або виклик через іншу функцію);

– описано хоча б один нерекурсивний спосіб досягнення результату – це необхідно для того, щоб рекурсія могла завершитися;

– параметри рекурсії на кожному виклику змінюються таким чином, щоб осягнути нерекурсивного опису – це також є умовою того, що рекурсія не буде нескінченою.

Якщо хоча б одна з умов базису не виконується, то рекурсія не є коректною.

Завдання для самостійної роботи

Описати рекурсивні функції для вирішення наступних завдань:

– функція, яка дозволяє обчислити вираз 1*2*3*...*n + 2*3*4*...*(n-1) + 3*4*...*(n-2) + ...

– функція, яка дозволяє обчислити вираз $\cfrac{x}{x+\cfrac{2}{x+\cfrac{4}{\ldots}}}$

$$x+\frac{n}{x}$$

– функція, яка дозволяє обчислити вираз $\cfrac{1}{1+\cfrac{1}{3+\cfrac{1}{5+\cfrac{1}{\dots}}}}$

$$(n-2)+\frac{1}{n}$$

– функція, яка дозволяє обчислити вираз $\cfrac{-1}{x+\cfrac{3}{x+\cfrac{-5}{\dots}}}$

$$x+\frac{(-1)^n n}{x}$$

– функція, яка дозволяє обчислити вираз n! – (n-1)! + (n-2)! – (n-3)! ...

– функція, яка дозволяє обчислити вираз $\cfrac{x}{x+\cfrac{x^3}{x+\cfrac{x^5}{\dots}}}$, (|x|<1)

$$x+\frac{x^n}{n}$$

– функція, яка дозволяє визначити, чи є рядок правильним записом виразу <вираз> :: = <символ> | <символ> * <вираз> * <символ> | <символ> ! <вираз> ! <символ>, де <символ> :: = a | b

Контрольні питання

1. Дайте визначення рекурсії.
2. Що таке пряма рекурсія? Наведіть приклади.
3. Що таке непряма рекурсія? Наведіть приклади.
4. Що є складовими базису рекурсії? Для чого використовується кожне з положень?

3.3 Принципи структурного програмування, особливості структурування програм, поняття «заглушки»

Становлення і розвиток структурного програмування пов'язане з ім'ям Едсгер Дейкстри. Основні принципи структурного програмування.

Принцип 1. Слід відмовитися від використання оператора безумовного переходу goto.

Принцип 2. Будь-яка програма будується з трьох базових керуючих конструкцій: послідовність, розгалуження, цикл.

Послідовність – одноразове виконання операцій в тому порядку, в якому вони записані в тексті програми.

Галуження – одноразове виконання однієї з двох або більше операцій, в залежності від виконання заданої умови.

Цикл – багаторазове виконання однієї і тієї ж операції до тих пір, поки виконується задана умова (умова продовження циклу).

Принцип 3. У програмі базові керуючі конструкції можуть бути вкладені одна в одну довільним чином. Ніяких інших засобів управління послідовністю виконання операцій не передбачається.

Принцип 4. Фрагменти програми, що повторюються, можна оформити у вигляді підпрограм (функцій). Таким же чином (у вигляді підпрограм) можна оформити логічно цілісні фрагменти програми, навіть якщо вони не повторюються. У цьому випадку в тексті основної програми, замість поміщеного в підпрограму фрагмента, вставляється інструкція «Виклик підпрограми». При виконанні такої інструкції працює викликана підпрограма. Після цього триває виконання основної програми, починаючи з інструкції, наступної за командою «Виклик підпрограми».

Принцип 5. Кожну логічно завершену групу інструкцій слід оформити як блок. Блоки є основою структурного програмування.

Блок – це логічно згрупована частина вихідного коду, наприклад, набір інструкцій, записаних підряд в вихідному коді програми. Поняття блок означає, що до блоку інструкцій слід звертатися як до єдиної інструкції. Блоки служать для обмеження області видимості змінних і функцій. Блоки можуть бути порожніми або вкладеними один в іншій. Межі блоку строго визначені. Наприклад, в if-інструкції блок обмежений фігурними дужками {...}.

Принцип 6. Всі перераховані конструкції повинні мати один вхід і один вихід. Довільні керуючі конструкції (такі, як в страві спагеті) можуть мати довільну кількість входів і виходів. Обмеживши себе керуючими конструкціями з одним входом і одним виходом, ми отримуємо можливість побудови довільних алгоритмів будь-якої складності за допомогою простих і надійних механізмів.

Принцип 7. Розробка програми ведеться покроково, методом «з гори донизу» (top-down method).

Спочатку пишеться текст основної програми, в якому, замість кожного зв'язкового логічного фрагмента тексту, вставляється виклик підпрограми, яка буде виконувати цей фрагмент. Замість справжніх підпрограм в програму вставляються фіктивні частини – заглушки, які, кажучи спрощено, нічого не роблять.

Якщо говорити точніше, заглушка задовольняє вимогам інтерфейсу замінного фрагмента (модуля), але не виконує його функцій або виконує їх частково. Потім заглушки замінюються або доопрацьовуються до справжніх повнофункціональних фрагментів (модулів) відповідно до плану програмування. На кожній стадії процесу реалізації вже створена програма повинна правильно працювати по відношенню до більш низького рівня. Отримана програма перевіряється та налагоджується.

Після того, як програміст переконається, що підпрограми викликаються в правильній послідовності (тобто загальна структура програми вірна), підпрограми-заглушки послідовно замінюються на реально працюючі, причому розробка кожної підпрограми ведеться тим же методом, що і основний програми. Розробка закінчується тоді, коли не залишиться жодної заглушки.

Така послідовність гарантує, що на кожному етапі розробки програміст одночасно має справу з доступним для огляду і зрозумілим йому безліччю фрагментів, і може бути впевнений, що загальна структура всіх вищих рівнів програми вірна.

При супроводі та внесення змін до програми з'ясовується, в які саме процедури потрібно внести зміни. Вони вносяться, не зачіпаючи частини програми, які безпосередньо не пов'язані з ними. Це дозволяє гарантувати, що при внесенні змін і виправлення помилок не вийде з ладу якась частина програми, яка перебуває в даний момент поза зоною уваги.

Слід також врахувати, що деякі автори зазначають, що принципи структурного програмування в рівній мірі можуть застосовуватися при розробці програм як «з гори донизу», так і «знизу догори».

Дотримання принципів структурного програмування робить тексти програм, навіть досить великих, такими, що нормально читаються. Серйозно полегшується розуміння програм, з'являється можливість розробки програм в нормальному промисловому режимі, коли програму може без особливих труднощів зрозуміти не тільки її автор, а й інші програмісти. Це дозволяє розробляти досить великі для того часу програмні комплекси силами колективів розробників, і супроводжувати ці комплекси протягом багатьох років, навіть в умовах неминучих змін в складі персоналу.

1. Структурне програмування дозволяє значно скоротити число варіантів побудови програми по одній і тій же специфікації, що значно знижує складність програми і, що ще важливіше, полегшує розуміння її іншими розробниками.

2. У структурованих програмах логічно пов'язані оператори перебувають візуально ближче, а слабо пов'язані – далі, що дозволяє обходитися без блок-схем та інших графічних форм зображення алгоритмів (по суті, сама програма є власною блок-схемою).

3. Сильно спрощується процес тестування і налагодження структурованих програм.

Поліпшення читабельності структурних програм пояснюється тим, що відсутність оператора goto дозволяє читати програму від верху до низу без розривів, викликаних передачами управління. У підсумку можна відразу

(одним поглядом) виявити умови, необхідні для модифікації того чи іншого фрагмента програми.

Контрольні питання

1. В чому полягають основні принципи структурного програмування?
2. Чому оператор goto під забороною в структурних програмах? Що таке стиль «спагеті»?
3. Які базові оператори використовуються в структурних програмах?
4. Дайте поняття блоку. Наведіть його ключові характеристики та надайте приклади.
5. В чому полягає принцип розробки «з гори до низу»?
6. Що таке «заглушка» в програмі? Для чого вони використовуються? Наведіть приклад заглушки.

СПИСОК ЛІТЕРАТУРИ

1. C++ Crash Course: A Fast-Paced Introduction./ Lospinoso Josh. ISBN 1593278885. - 2019.- 792с.
2. International Standard ISO/IEC 14882:2014(E) – Programming Language C++ , ISBN-13: 978- 0321563842: [Електронний ресурс]. – Режим доступу: https://isocpp.org/std/the-standard.
3. C/C++ language and standard libraries reference: [Електронний ресурс]. – Режим доступу: https://msdn.microsoft.com/en-us/library/hh875057.aspx
4. Jeremy A. Hansen. The Rook’s Guide to C++. Rook’s Guide Press. URL https://rooksguide.files.wordpress.com/2013/12/rooks-guide-isbn-version.pdf.
5. Програмування на С++ в прикладах і задачах : Навч. посіб. / О. Васильєв. – Київ : Видавництво Ліра-К, 2017. – 382 с.