**Лекція № 10.** **Процедури і функції. Файли.**

**На період** карантину в дистанційній формі навчання на надані в кінці запитання потрібно надати письмові відповіді в текстовому файлі (бажано WORD), надіславши їх на адресу викладача **t.i.lumpova@gmail.com**.

**PTBD21<Номер лекції / практичної / лабораторної>[-<Номер завдання ЛР>][літера позначення типу роботи L – лекція, P –практична, R – лабораторна]<Прізвищеанглійською>**. Наприклад, **PTBD21L**buts.doc. Кожна відповідь оцінюється в 0,5 балів. Відповіді повинні бути не довгими і змістовними. Не копіюйте фрагментів з різних інформаційних джерел, подумайте і викладіть свою точку зору. При наявності відповідей-"близнюків" відповідь буде зараховуватися першому за часом надсилання.

**Строк виконання цієї роботи**

**Увага!** C++Shell не підтримує роботу з файлами.

Для роботи з файлами можна використати безкоштовний компілятор Dev-C++, який потрібно встановити на ваш комп’ютер. Скачати його можна за посиланням:

**https://sourceforge.net/projects/orwelldevcpp/**

Інструкція по роботі з компілятором надається за посиланням (але там забагато реклами та іншого сміття):

**https://studopedia.ru/18\_64672\_pokrokove-vikonannya-programi.html**

**Визначення функції**

**Функція** — це іменована логічно завершена сукупність оголошень і операторів, призначених для виконання певної задачі

Програма мовою С++ містить одну або декілька функцій, кожна з яких повинна бути оголошена та визначена до її першого використання. **Оголошення функці**ї (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. **Визначення функції** — це завдання способу виконання операцій.

Серед функцій програми повинна бути одна з ім’ям **main (головна функція)**, яка може знаходитися в будь-якому місці програми. Ця функція виконується завжди першою і закінчується останньою.

**Структура функції**

Усі функції мають однакову структуру визначення у вигляді:

**[тип результату] ім'я функції ([список формальних аргументів])**

**{ // *тіло функції***

**опис даних;**

**оператори;**

**[return] [вираз]; };**

**Тип результату** — будь-який базовий або раніше описаний тип значення (за винятком масиву і функції), 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.

Приклад.

**// Example program**

**#include <iostream>**

**#include <string>**

**using namespace std;**

**int add(int a, int b)**

**{ return a + b;}**

**int multiply(int c, int d);**

**int main()**

**{**

**cout << add(7, 8) << endl;**

**// всередині функції add(): a = 7, b = 8, значить a + b = 15**

**cout << multiply(4, 5) << endl;**

**// всередині функції multiply(): c = 4, d = 5, значить c \* d = 20**

**// Ми можемо передавати цілі вирази в якості аргументів**

**cout << add(2 + 3, 4 \* 5) << endl;**

**// всередині функції add(): a = 5, b = 20, значить a + b = 25**

**// Ми можемо передавати змінні в якості аргументів**

**int x = 4;**

**cout << add(x, x) << endl; // буде 4 + 4**

**cout << add(1, multiply(2, 3)) << endl; // буде 1 + (2 \* 3)**

**cout << add(1, add(2, 3)) << endl; // буде 1 + (2 + 3)**

**return 0;**

**}**

**int multiply(int c, int d)**

**{**

**return c \* d;**

**}**

**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 х, у;**

**або**

**static float р = 3.25;.**

***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.

**Огляд процесу виконання виклику функції**

1. Виділяється пам'ять для точки повернення й параметрів функції. Посилання на точку повернення з функції запам'ятовується.

2. Обчислюються значення аргументів для параметрів значень, посилання на пам'ять аргументів для параметрів посилань. Відбувається підстановка аргументів.

3. Виділяється пам'ять, відповідна до локальних імен змінних (окрім локальних статичних змінних).

4. Виконуються інструкції тіла функції до інструкції повернення.

5. Якщо підпрограма не є **void**-функцією, то значення, що повертається з її виклику, копіюється у пам'ять функції, яка містила виклик.

6. Функція, що містила виклик, продовжується з точки повернення.

Змінні в локальній пам'яті функції не відповідають іменам у функції, що містила виклик, тобто ця пам'ять **недоступна** після того, як виклик закінчено. Вона **вважається звільненою**; її можна використовувати для наступного виклику цієї або іншої функції. Відбувається логічне звільнення, тобто зміст локальної пам'яті не змінюється, але стає недоступним.

**Автоматична пам'ять або програмний стек**

Ділянки пам'яті викликів функцій утворюються та звільняються в спеціальній області пам'яті процесу виконання програми – **автоматичній пам'яті**. Називається вона так тому, що за виконання викликів функцій пам'ять виділяється та звільняється без явних вказівок у програмі, написаній мовою високого рівня, тобто автоматично.

Під час виконання викликів функцій ділянки автоматичної пам'яті виділяються та звільняються за принципом "останньою зайнято – першою звільнено" ("*Last In –First Out*" (LIFO), тобто "останнім прийшов – першим пішов"). Тому автоматичну пам'ять програми також називають **програмним стеком**. коли послідовно виконуються два виклики функцій у тілі функції, то для другого виклику виділяється пам'ять, звільнена після першого.

Приклад.

**#include <iostream>**

**using namespace std;**

**int f(int x)**

**{ return ++x; }**

**int g(int x)**

**{ return x/=2; }**

**int main(){**

**int a=12;**

**cout << f(g(a)) << ' ';**

**cout << a << endl;**

**return 0; }**

З виклику функції **g** повертається **6** і присвоюється параметру **x** функції **f**. Пам'ять функції **g** звільняється. Тільки тепер виконуються інструкції функції **f**. Після їх закінчення в головну функцію повертається значення **7**. Пам'ять виклику функції **f** звільняється.

**Механізм передачі параметрів**

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

**double sqr (double); // прототип функції sqr()**

**main( ) // головна функція**

**{ cout << "Квадрат числа=" << sqr (10) << endl; }//виклик функції sqr()**

**double sqr (double p) //функція sqr()**

**{ return p\*p; } // повернення за значенням**

Функція завжди має бути визначена або оголошена до її виклику. В прикладі спочатку надається прототип функції як оголошення, а опис самої функції надається після **main**-функції, яка її викликає.

***При оголошенні, визначенні та виклику тієї самої функції типи та послідовність параметрів повинні співпадати.*** На імена параметрів обмежень на відповідність не існує, оскільки функцію можна викликати з різними аргументами, а в прототипах імена ігноруються компілятором (вони необхідні тільки для покращення читання програми). Тип значення, що повертає функція, та типи параметрів спільно визначають тип функції.

У найпростішому випадку при виклику функції потрібно вказати її ім’я, за яким у круглих дужках через кому – перелічити імена аргументів, що передаються. Виклик функції може здійснюватися у будь-якому місці програми, де за синтаксисом дозволяється вираз того типу, що формує функція. Якщо тип значення, що повертає функція не **void**, вона може входити до складу виразів або розташовуватись у правій частині оператора присвоювання.

***У мові С++ визначено декілька способів передачі параметрів і повернення результату обчислень функцій, серед них найбільш широке використання набули:***

1. виклик функції з передачею параметрів за допомогою формальних аргументів-значень;
2. виклик функції з передачею адрес за допомогою параметрів-покажчиків;
3. виклик функцій з використанням посилань, коли доступ до переданих параметрів забезпечується за допомогою альтернативного імені (синоніма);
4. виклик функцій з передачею даних за допомогою глобальних змінних;
5. виклик функцій з застосуванням параметрів, що задані за замовчуванням, при цьому можна використовувати або всі аргументи, або їх частину.

**Ми розглянемо тільки 1, 4, 5 випадки.**

**Виклик функції з передачею значень** полягає у тому, що у функцію передаються не самі аргументи, а їх копії. Ці копії можна змінювати всередині функції, і це ніяк не позначиться на значеннях аргументів, що за межами функції залишаться без зміни, наприклад:

void fun (int **р)** // функція fun()

**{++p;**

**cout << " p = " << p << endl; }**

**void main ( )** // головна функція

**{ int x = 10;**

**fun (x);** // виклик функції

**cout << "x = " << x << endl; }**

Результат роботи цього фрагмента програми: **р=11, х=10,** оскільки для виклику функції fun(x) до неї передається копія значення, що дорівнює **10**. Всередині функції значення копії змінної збільшується на **1**, тобто **(++р)**, і тому виводиться **р = = 11**, але за межами функції параметр **р** не змінюється. У цьому випадку функція не повертає ніякого значення.

При цьому способі для звертання до функції достатньо написати її ім’я, а в дужках — значення або перелік фактичних аргументів. Фактичні аргументи повинні бути записані в тій же послідовності, що і формальні, і мати відповідний тип (крім аргументів за замовчуванням і перевантажених функцій).

Якщо формальними аргументами функції є параметри-значення і в ній не використовуються глобальні змінні, функція може передати у викликаючу її програму лише одне значення, що записується в операторі **return**. Це значення передається в місце виклику функції. Достроковий вихід з функції можна також організувати з використанням оператора return.

**Виклик функцій з передачею даних за допомогою глобальних змінних**

Використовувати глобальні змінні для передачі даних між функціями дуже легко, оскільки вони видимі в усіх функціях, де описані локальні змінні з тими ж іменами. Але такий спосіб не є поширеним, тому що ускладнює налагодження програми та перешкоджає розташуванню функції у бібліотеці загального використання. Потрібно прагнути, щоб функції були максимально незалежними, а їхній інтерфейс повністю визначався прототипом функції. Наведемо приклад використання глобальних змінних:

#include **< iostream.h >**

**int a, b, с;**             // глобальні параметри

**sum ( );**                //прототип функції

**main ( )**                //головна функція

{ **cin >> а >>b;**

**sum();**                // виклик sum()

**cout << с << endl; }**

**sum( )**                  // функція sum()

**{ с** = **а** **+ b;** **}**

**Передача даних за замовчуванням.**

При написанні функції всім аргументам або декільком з них присвоюються початкові значення і задовольняються такі вимоги: коли якому-небудь аргументу присвоєно значення за замовчуванням, то всі аргументи, що розташовані за ним (тобто записані праворуч), повинні мати значення за замовчуванням. Таким чином, список параметрів поділяється на дві частини: параметри, що не мають значення за замовчуванням, і параметри, що мають такі значення.

У випадку виклику функції для параметрів, що не мають значень за замовчуванням, обов’язково повинен бути фактичний аргумент, а для параметрів, що мають значення за замовчуванням, фактичні аргументи можна опускати, коли ці зна­чення не треба змінювати.

Якщо деякий параметр має значення за замовчуванням та для нього відсутній фактичний аргумент, то і для всіх наступних (тобто записаних пізніше) параметрів фактичні аргументи повинні бути відсутні, тобто їхні значення передаються до функції за замовчуванням, наприклад:

**void functl (float х, int у, int z = 80)**

**{ cout << "x = " << x << " у = " << у << "z = " << z** **<< endl;}**

**void funct2 (float x, int у = 25, int z = 100)**

**{ cout << "x = " << x << "y = " << у << "z = " << z << endl; }**

**void funct3 (float x = 3.5, int у = 40, int z = 200)**

**{ cout << "x = " << x << "y = " << у << "z = " << z << endl; }**

**main ( ) {**

**functl (5.1, 10);**//пo замовчуванню передається один аргумент — z

**funct2 (10.2);** // пo замовчуванню передаються два аргумента — у, z

**funct3 ( );**     // пo замовчуванню передаються всі аргументи

}

На екрані буде виведено:

**х = 5.1       у = 10      z = 80**

**х = 10.2    у = 25      z = 100**

**х = 3.5      у = 40      z = 200**

**Аргумент за замовчуванням — це той аргумент, значення якого задане при описі заголовка функції, а при її виклику його можна не вказувати.**

Якщо замість параметра, заданого за замовчуванням при звертанні до функції, записується інше значення фактичного параметра, то значення за замовчуванням перекривається заданим фактичним значенням. Так, наприклад, в останньому програм­ному фрагменті при виклику функції **funct2(13.5, 75);** на екрані буде виведено:

**х = 13.5**        **у = 75       z = 100,** тобто **z** — прийнято за замовчуванням.

Якщо виникає необхідність збереження значень локальних змінних між викликами функції, то вони повинні бути оголошені як статичні, тобто з описом **static**, наприклад: **static char st[ ] = “Приклад”;.**

**Поняття файлу**

Під час розв’язання задач на комп’ютері часто виникає необхідність у використанні даних, які записані на зовнішніх носіях інформації (дисках) і оформлені у вигляді файлів даних. Незалежно від того, які дані містять файли (числа, символи, рядки, масиви, структури тощо), в мові С++ вони трактуються як потоки даних **(stream),** які являють собою послідовність байтів, що зчитуються або записуються.

За замовчуванням у кожній програмі С++ можна користуватися такими стандартними потоками: стандартного введення **(сіn),** стандартного виведення **(cout)** та виведення помилок **(сеrr).** Щоб користуватися файлами, потоки повинні бути створені і закріплені за цими файлами.

**Використання файлів даних у програмі передбачає виконання таких операцій**:

* створення потоку обміну даними між файлом і пам’яттю комп’ютера;
* зв’язування цього потоку з конкретним ім’ям файлу на диску і відкриття файлу;
* запис даних у файл або читання їх з файлу;
* закриття файлу.

Для реалізації цих операцій існують спеціальні класи (структури даних разом з функціями обробки цих даних), які містять конструктори створення необхідних потоків:

* **ifstream** — для створення потоку читання даних;
* **ofstream** — для створення потоку запису даних у файл;
* **fstream** — використовується як для запису даних у файл, так і їх читання.

**Конструктори для роботи з файлами**

**Конструктори** — це спеціальні функції, які мають таке саме ім’я, що й ім’я класу. Вони записуються як з параметрами, так і без параметрів. Конструктори з параметрами одночасно створюють відповідний потік, зв’язують його з файлом на диску, відкривають файл для роботи і мають такі форми запису:

**ofstream іп (“іф”, ios::out); або ofstream (“іф”);**

**ifstream іп (“іф”, ios::in); або ifstream (” “);**

**fstream іп (“іф”, ios::in | ios::out);**

де **іп** — ім’я потоку, який створюється для роботи з файлом;

**іф** — константа або змінна типу **char[ ],** її значення — ім’я файлу на диску.

Перший з конструкторів використовується для запису даних у файл, другий — для читання даних з файлу, а третій — як для запису, так і для читання даних, наприклад:

**ofstream fout( "myfile.dat", ios::out);**

Цей запис створює потік з ім’ям **fout**, зв’язує його з файлом на диску, який має ім’я **myfile.dat** і відкриває цей файл для запису даних. Файл **myfile.dat** буде створено у тому ж каталозі, що і програма. Якщо треба створити файл у другому місці, то для запису його імені треба вказати шлях, наприклад, **a:\\pvp\\myfile.dat**. Тепер цей файл буде записано на диску **а:** в каталозі **pvp**.

Зверніть увагу на те, що для запису шляху треба використовувати подвійні зворотні косі риски.

Можна також для роботи з файлами застосувати конструктори без параметрів:

**ofstream іп;**

**ifstream іп;**

**fstream іп;**

де **іп** — ім’я відповідного потоку, тоді для зв’язування потоку з ім’ям файлу на диску і відкриття його для роботи треба додатково використовувати функцію-член відповідного класу, тобто:

**іn.ореn(“іф”, ios :: режим | ios :: режим);**

(в цьому запису  **|** означає "або", тобто можливість завдання декількох режимів).

Наприклад, відкриття файлу для запису до нього даних матиме вигляд:

**ofstream fout;**

**fout.open ( "a:\\pvp\\my file.dat", ios::out);**

Конструктори як з параметрами, так і без них, виконують однакову роботу, тому яким з них надати перевагу — справа користувача.

***Приклад***  Створити файл на диску і записати до нього масив чисел. Прочитати цей файл і вивести його компоненти на екран.

// Створення файлу та запис до нього масиву

**#include <iostream>**

**#include <fstream>**

**#include <conio.h>**

**#include <Windows.h>**

**using namespace std;**

**int main()**

**{ int i, mas[5];**

**system("color F0");**

**// запис елементів масиву до файлу:**

**ofstream fout("massiv.txt"); /\* створення потоку fout та відкриття файлу з іменем massiv. txt для запису \*/**

**if (! fout) cout <<"Cannot open file\n";**

**for (i = 0; i <5; i++)**

**{**

**cout << " Enter "<< i << " element\n";**

**cin >> mas[i]; // введення елементу масиву з клавіатури**

**fout << mas[i] << " "; //запис елементу до файлу**

**}**

**fout.close();**

**//---------- читання компонентів файлу та виведення на екран**

**ifstream fin("massiv.txt"); /\* створення потоку fin для читання файлу\*/**

**if (!fin) cout << " Cannot open file fo reading\n";**

**else {**

**cout <<"REZULTAT \n";**

**for (i = 0; i <5; i++)**

**{**

**fin >> mas[i];**

**// читання поточного елементу масиву з файлу**

**cout << "mas[" << i << "]=" << mas[i] << " ";**

**}**

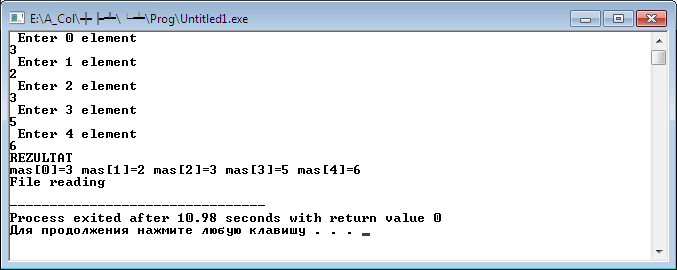
**cout << "\nFile reading\n";};**

**fin.close();**

**return 0;**

**}**

Результат розв’язання прикладу має вигляд:



*Для самостійного вивчення (10 годин)*: Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

*Контрольні запитання для самоперевірки*.

1. Які змінні називаються локальними?
2. Розкажіть про порядок дій, які виконуються під час виклику функції.
3. Які параметри називаються формальними, а які фактичними?
4. Чи можна у викликаємій функції змінювати параметри при **виклику функції з передачею значень? Чи вплине така зміна значень параметрів на дії в головній функції, якщо ці параметри потім там використовуються?**
5. **Які правила встановленні для формування списку параметрів, коли** передача даних виконується за замовчуванням?

*Контрольні запитання для надання письмових відповідей*.

1. .Що потрібно зробити, щоб зберегти значення в середині функції між її викликами? Наведіть приклад у вигляді фрагменту коду.
2. Розгляньте останній приклад , виберіть всі мовні конструкції щодо оброблення файлів, які ми детально не розглядали в лекції.

*Література*

1. Ковалюк Т. В. Алгоритмізація та програмування: Підручник. — Львів: «Магнолія 2006», 2013. — 400 с., ил.
2. Єжова Л. Ф. Алгоритмізація і програмування процедур обробки інформації: Навч.-метод. посіб. для самост. вивч. дисц. К.: КНЕУ, 2000.
3. Вступ до програмування мовою С++. Організація обчислень: навч. посіб. / Ю. А. Бєлов, Т. О. Карнаух, Ю. В. Коваль, А. Б. Ставовський. – К. : Видавничо-поліграфічний центр "Київський університет", 2012. – 175 с. с.: іл. ISBN (укр.) . URL: <http://csc.knu.ua/uk/library/books/belov-24.pdf>
4. Джейс Либерти Освой самостоятельно С++ за 21 день: 3-е изд. пер. с англ.: Уч. пос. – М.: Издательский дом „Вильямс”, 2001. – 816 с.: ил..

Робота в класі

#include <iostream>

using namespace std;

/\*int f(int x)

{ return ++x; }\*/

int f(int x,int y=0)

{ static int c=0;

c=c+x\*y;

return c; }

int f1(int x,int y=10)

{ static int c=0;

c=c+x\*y;

return c; }

int g(int x)

{ return x/=2; }

int f2(int a)

{ a++;

return a; }

int main(){

int a=12;

cout << f2(a) << ' '<< a<< endl;

cout << f(g(a)) << ' '<<f(a,a)<< ' '<<f(a,a)<<endl;

cout << f1(a,1)<< ' '<<f1(a)<<endl;

return 0; }