**Лекція 5. Використання функцій**

**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.

Слід нагадати, що 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.

Усі функції мають однакову структуру визначення у вигляді:

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

**{ // *тело функции***

**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.

**Узагальнимо.**

*Визначення (definition) функції* включає наступні елементи (згідно затвердженому ANSI-стандарту мови С) :

1. вказівку про можливість використання функції поза даним файлом (за допомогою одного з службових слів static або extern; останнє вживається за замовчуванням);
2. тип результату, який повертає функція; відсутність результату позначається службовим словом void;
3. ідентифікатор (ім’я) функції;
4. кількість та типи *формальних параметрів* в круглих дужках; в разі їх відсутності вживається службове слово void;
5. важливо: всі параметри функції передаються **за значенням**
6. тіло функції, яке взяте в фігурні дужки; складається з визначення внутрішніх (локальних) змінних функції та інструкцій, що мають виконуватись під час роботи функції. Якщо функція повертає результат, то серед цих інструкції обов’язково має бути інструкція виду: return <вираз>;

причому тип виразу повинен збігатись з типом результату функції.

***Пригадаємо класи пам’яті об’єктів мов С/C++***

|  |  |  |  |
| --- | --- | --- | --- |
| Класи пам'яті зовнішніх *глобальних* об'єктів (функцій та змінних) | | Класи пам'яті внутрішніх ***локальних*** об'єктів (лише змінні) | |
| **extern**  (за замовчу-ванням) | Доступ можливий з усіх файлів програми | **auto**  (за замовчу-ванням) | Доступ можливий лише у блоці, де визначений об'єкт |
| **static** | Доступ можливий лише у файлі, де визначений об'єкт | **register** | Компілятор намагатиметься розміщувати об'єкт на регіст-рах процесора |

**Важливі зауваження про класи пам'яті**

1. Визначення змінної означає виділення під неї пам'яті, декларація змінної – це представлення її компілятору.
2. Всі змінні, визначені поза межами функцій, є глобальними (клас пам'яті extern за замовчуванням). Вони доступні у всіх функціях програми.
3. Всі змінні, визначені у функціях, а також параметри функцій, – локальні (клас пам'яті auto за замовчуванням). Доступні лише в тому блоці, де визначені.
4. Якщо біля глобальної змінної вказується службове слово extern, то це не визначення, а декларація змінної, яка буде визначена далі.
5. Локальна змінна з тим самим ідентифікатором “перекриває” глобальну змінну. Те саме стосується і локальних змінних різних рівнів.
6. Службове слово register біля локальної змінної – це лише побажання до системи розмістити її на регістрах процесора, щоб прискорити звертання до неї – компілятор в праві проігнорувати це прохання.

**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. Прототип - це можливість зробити функцію видимою для компілятора. В одному програмному файлі допустимо кілька проьлтипів однієї функції і лише одне її визначення.

Приклад.

**int strlen (char \*); // декларація функції**

**int strlen (char \*s) // визначення функції**

**{**

**char \*p = s;**

**while (\*p != '\0') p++;**

**return p-s;**

**}**

**Огляд процесу виконання виклику**

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;**

**system("pause"); return 0; }**

З виклику функції **g** повертається **6** і присвоюється параметру **f.x**. 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** 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 no значенню**

**… }**

Функція завжди має бути визначена або оголошена до її виклику. ***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 слід 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:***

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

**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)** //----------- функция fun()

**{++p;**

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

**void main ( )** //----------- главная функция

**{ int x = 10;**

**fun (x);** //----------- вызов функции

**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 функції з використанням параметра-посилання** здійснює передачу до функції не самої змінної, а тільки посилання на неї. У цьому випадку забезпечується доступ до переданого параметра за допомогою визначення його альтернативного імені, бо посилання є синонімом імені параметра. Тоді всі дії, що відбуваються над посиланням, є діями над самою змінною. Такий спосіб передачі параметрів і повернення результату передбачає запис у списку фактичних параметрів імені змінної, а у списку формальних — параметрів-посилань. Наприклад:

//-------------- **використання параметра-посилання**

**void fun (int &p)** //---------- функция fun()

**{ ++p;**

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

**void main** **( )**

**{ int x** **= 10;**

**fun (x);** //---------- вызов функции fun()

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

Одержимо результат попереднього фрагмента, тобто **р = 11 і х=11.**

При застосуванні вказаних вище параметрів у функцію передаються не значення змінних, а їхні адреси, тому можливо міняти значення цих змінних усередині функції і передавати за її межі (в інші функції). Коли необхідно, щоб деякі параметри не змінювали свої значення всередині функції, їх слід оголосити як параметри-константи, використовуючи модифікатор **const**.

**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. Слід 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;**             // глобальные параметры

**sum ( );**                //----------- прототип функции

**main ( )**                //----------- главная функция

{ **cin >> а >>b;**

**sum();**                //----------- вызов sum()

**cout << с << endl; }**

**sum( )**                  //----------- функция 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 умолчанию передаётся один аргумент — z

**funct2 (10.2);** //пo умолчанию передаются два аргумента — у, z

**funct3 ( );**        // по умолчанию передаются все аргументы

}

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 **functl (13.5, 75);** на ек­рані буде виведено:

**х = 13.5**        **у = 75       z = 80,** 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”;.**

**Масиви як параметри функцій**

Аргументами (параметрами) функцій можуть бути не тільки змінні, але й масиви. Можна використовувати як масиви фіксованого розміру, так і невизначеного (масиви змінної довжини). При застосуванні масивів фіксованої довжини в заголовку функції в списку формальних аргументів указується тип масиву і його розмір, наприклад:

**void sort (int mas[30]);**

Якщо описується функція з масивом змінної довжини, то в заголовку вказується тип масиву невизначеного розміру і обов’язково ще один параметр, за допомогою якого задається розмірність масиву, наприклад:

**void sort (int mas[ ],int n);**

Всі масиви у функції передаються за адресою (як покажчики), тому у випадку зміни масивів у функції ці зміни зберігаються при поверненні у викликаючу функцію.

***Приклад9.3.***У масива хm1(10),m2(15),m3(12)визначити мінімальний елемент та його індекс.

**/\* определение минимальных значений  массива, использование глобальной переменной\*/**

**#include <iostream.h>**

**#include <conio.h>**

**int ind=0; //--- глобальная переменная**

**vvod(float mas[ ],int n) //функция ввода элементов мас. vvod()**

**{ for (int і = 0; і <n ;i++)**

**{ cout<<"Введите" <<і <<" элемент ";**

**сіn>>mas [і]; }**

**cout<<endl; }**

**float fmin(float mas[ ],int n) // ф-ция определения мин. Эл-та**

**{ ind = 0; float min =mas[0];**

**for (int і = 1; і < n; i++)**

**if (mas[i] <min) { min = mas[i]; ind = i;}**

**return min; }**

**main()//---------------главная функция**

**{float ml[5];**

**cout« "\*\*\*\*\*Введите массив m1\n";**

**vvod(ml,5); //вызов функции vvod()— ввод массиваm1[]**

**cout<<"min1 = "<<fmin (m1,5); //вызов функцииfmin()**

**cout<<" ind = "<<ind<<endl;**

**getch(); }**

Результати обчислень:

**\*\*\*\*\*Введите массив    m1**

**Введите 0 элемент 5**

**Введите 1 элемент 7**

**Введите 2 элемент 3**

**Введите 3 элемент 9**

**Введите 4 элемент 1**

**min1= 1     ind= 4**

Програма, крім **головної функції main()**,має також функцію **vvod()** введення елементів деякого формального масиву та функцію **fmin()**визначення мінімального елемента цього масиву. У головній функції здійснюється виклик функцій для розв’язання необхідних обчислень кожного конкретного масиву, для передачі параметрів використовується глобальна змінна.

**Параметри функцій - багатовимірні масиви**.

У цьому випадку використовуються масиви як фіксованої розмірності, так і невизначеної довжини. У заголовку функції під час роботи з багатовимірним масивом фіксованого розміру, наприклад матриці mat(7,10),вказуються розмірності масиву:

**void fun1(intmat[7][10]);**

Якщо застосовується багатовимірний масив невизначеної довжини, то невизначеним може бути тільки один вимір розмірності, наприклад:

**void fun2 (int mat[][10], int rows, int cob);**

***Приклад.***Для заданої матриці зробити обчислення середнього значення кожного її стовпця з використанням функції введення розмірності матриці, функції введення матриці і функції одержання середнього значення її стовпців.

// вычисление среднего значения столбцов матрицы

#include <iostream.h>

#include <conio.h>

**const int mincol = 1;**

**const int maxcol = 4;**

**const int minrow = 2;**

**const int maxrow = 4;**

//------функция getnum()для ввода количества строк и столбцов

**int getnum (const char \*elemtype, int low, int high)**

**{int n;**

**do**

**{cout<<**"Vvedite kolichestvo" <<**elemtype**<< "n ot["**<<low**<<"] to[" **<<high**<< "] : "**;**

**cin >>n; }**

**while (n <low ||n > high); // OR**

**return n; }**

//--------------функция inmatr() ввода элементов матрицы

**void inmatr (float matr[][maxcol], int rows, int cols)**

**{ for (int і = 0; і < rows; і++)**

**{ cout <<**"Vvedite " **<<і** << " stroky matrix " <<endl;

**for (int j= 0;j<cols; j++)**

**cin >>matr [i][j];}**

cout <<endl; }

//функция srcols() определения средних значений столбцов

**void srcols (float matr [ ] [maxcol], int rows, int cols)**

**{ float sum, sr;**

**for(int j= 0;j<cols;j++)**

**{ sum= 0.0;**

**for (int і= 0;і<rows; i++) sum+=matr [i][j];**

**sr = sum/rows;**

cout.precision(3);

**cout <<**"Srednee znachenie stolbca "<<j<<" = "<<**sr**<< endl; **}**

**}**

**void main( )** //-------------------главная функция

**{ float matr [maxrow][maxcol]; int rows, cols;**

**rows= getnum ("strok", minrow, maxrow);** //ввод к-ва строк и столбцов

**cols = getnum ("stolbcov", mincol, maxcol);**

**inmatr (matr, rows, cols);** //------ввод матрицы

cout<<endl;

**srcols(matr,rows,cols);** //-вычисление ср.значения столбцов матрицы

getch();

**}**

Результати обчислень:

**Vvedite kolichestvo strok n ot [2] do[30] : 3**

**Vvedite kolichestvo stolbcov n ot [1] do [20] : 4**

**Vvedite 0 stroky matrix**

**5.6 8.9 1.25 6.78**

**Vvedite 1 stroky matrix**

**0.25 7.6 1.3 5.29**

**Vvedite 2 stroky matrix**

**3.6 10.5 0.6 11.67**

**Srednee znachenie stolbca0 = 3.15**

**Srednee znachenie stolbca1=9**

**Srednee znachenie stolbca2 = 1.05**

**Srednee znachenie stolbca 3 = 7.91**

Програма, крім **головної функції main()**,має також функцію **vvod()**введення елементів деякого формального масиву та функцію **fmin()**визначення мінімального елемента цього масиву. У головній функції здійснюється виклик функцій для розв’язання необхідних обчислень кожного конкретного масиву, для передачі параметрів використовується глобальна змінна.

**Передача масиву у функцію за допомогою покажчиків**

У цьому випадку функція повинна мати опис покажчика на той тип даних, елементами якого є масив, а у викликаючій функції достатньо вказати лише ім’я масиву (як відомо, у мові С++ ім’я масиву є адресою цього масиву).

За необхідності передачі багатовимірних масивів за допомогою покажчиків треба враховувати те, що в пам’яті такі масиви зберігаються як одновимірні. Наприклад, при використанні, матриці **a[n,m]** у пам’яті послідовно розміщуються n рядків по  **m  елементів** у кожному. Тому для роботи окремо з кожним рядком необхідно сформувати масив покажчиків на ці рядки, тобто:

**int а[n][m], \*b[n];  
for(і= 0;і<kstrok;і++)  
b[i] = &a[i][0];**

Тоді кожен елемент масиву b[і] буде зберігати адресу і-го рядка матриці а[n,m], тому до кожного елемента і-го рядка j-ro стовпця можна звертатись як b[i][j] або \*(\*(b+i)+j).

Використовуючи масив покажчиків\*b[n], можна передавати як параметр функції подвійний покажчик(\*\*b), а у викликаючій функції достатньо вказати лише ім’я матриці.

***Приклад.*** Для заданої матриці переставити місцями парні та непарні строки з використанням функції введення матриці, функції виведення матриці і функції перестановки.

//Обмін місцями парних і непарних рядків матриці

#include <iostream.h>

**void inmatr(int\*\*,int,int);** //прототип функції введення матриці

**void outmatr(int\*\*,int,int);** //прототип функції виведення матриці

**void obmen(int\*\*,int);** // прототип функції обміну рядків матриці

**void main()**

**{int const k=12;**

**int kstrok, kstolbcov,i,j;**

**int a[k][k];**

**int \*b [k]; {**

cout<<endl;

cout<<"Vvedite kolicestvo strok matrici "<<endl;

**cin>>kstrok;**

cout<<"Vvedite kolicestvo stolbcov matrici "<<endl;

**cin>>kstolbcov;**

//формування масиву покажчиків на кожен рядок матриці

**for(i = 0; i< kstrok; i++) b [i] = &a[i][0];**

cout<<"Vvedite matricu " <<endl;

**inmatr (b, kstrok, kstolbcov);**// введення матриці

**outmatr(b, kstrok, kstolbcov);**

**obmen (b, kstrok);** // обмін рядків матриці

cout<<"Rezult matrica " <<endl;

**outmatr(b, kstrok, kstolbcov);//** виведення матриці

system("pause"); **} }**

**void inmatr (int\*\*x, int kstrok, int kstolbcov)** // функція введення матриці

**{for (int i = 0; i < kstrok; i++)**

**for (int j= 0;j < kstolbcov; j++)**

**cin>>x[i][j]; }**

**void obmen (int\*\*x, int kstrok)** //----- функція обміну рядків матриці

**{int \*p;**

**for (int i= 0;i< kstrok-1; i+=2)**

**{p= x[i];**

**x[i]=x[i+1];**

**x[i+1]=p;}**

**}**

//- функція виведення рядків матриці

**void outmatr (int\*\*x, int kstrok, int kstolbcov)**

**{for (int i= 0;i<kstrok;i++)**

**{cout<<endl;**

**for (int j= 0; j < kstolbcov; j++)**

**cout <<x[i][j] <<" "; }**

**cout<<endl;**

**system("pause");**

**}**

Важливі зауваження

1. Передача параметру посиланням

int incr (int& n)

{

n++;

return n;

}

............................

int s = 5;

cout << incr (s);

Тут посилання як параметр функції буде створюватися та ініціалізуватися при кожному виклику функції, а при завершенні виклику функції це буде руйнуватися, тобто кожного разу буде створюватися нове посилання. Тут параметр, що передається, є псевдонімом фактичного параметра (аргументу). Тут не відбувається копіювання і функція має прямий доступ до аргументу.

1. Функція може повертати посилання, оскільки посилання не є окремою сутністю (це псевдонім чогось іншого), важливо, щоб об’єкт, на який вона посилається існував після завершення. Функція в якості результату може повертати посилання, але це не повинно бути посилання на локальні змінні.

***Контрольні запитання для самоперевірки***.

1. Чи обов’язково вказувати типи параметрів функції в прототипі? Чи обов’язково вказувати імена параметрів?
2. Опишіть призначення функцій та особливості їх оголошення.
3. Охарактеризуйте особливості передачі параметрів функції значенням, масивів; замовчувані значення параметрів.
4. Дайте пояснення, що таке «Області видимості функцій».
5. Охарактеризуйте особливості повернення та використання значення функцій.
6. Визначення і оголошення глобальних об’єктів і функцій.
7. Тривалість життя об’єктів: автоматичне, статичне і динамічне.
8. Визначте типи формальних параметрів.

**Для самостійного вивчення** *(2 години)*: Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

**Рекомендована література**

1. Ковалюк Т. В. Алгоритмізація та програмування: Підручник. — Львів: «Магнолія 2006», 2013. — 400 с., ил.
2. Павловская Т.А. С/С++. Программирование на языке высокого уровня. СПб.: Питер, 2003. – 461 с. URL: <http://www.ph4s.ru/bookprogramir_1.html>
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..

5. Дейтел Х., Дейтел П. Основы программирования на С++. – М.: Бином, 1999. – 1024 с. URL: <http://ijevanlib.ysu.am/wp-content/uploads/2018/03/deytel.pdf>