**Практична робота №8. Написання програм з використанням функцій з передачею параметрів за різними механізмами**

**Завдання.**

Написати на мові С++ програму введення даних для розрахунку формул з передачею параметрів за різними механізмами.

1. Ознайомитися з теоретичною частиною.
2. Продовжуємо працювати з програмою, складеною на ПР№7. В головній програмі додати ще 2 позиції в подання меню (10 та 11) та зробити функції, в яких надається код обчислення формул в пп. 2.1 та 2.2. Порядок виклику функцій надається в пп. 2.1 та 2.2, відповідно.

2.1. Функція розрахунку Р, отримує значення х, у, а, b, яким в заголовку встановлюють замовчувані значення. Зробити 5 викликів функції, де не надається жодного параметра, надаються значення одного, двох, трьох та всіх параметрів.

, де *х*=0,25; *у*=1,31; *а*=3,5; *b*=0,9

2.2. Розрахунок Y виконується в циклі (по х) в case шляхом виклику відповідної функції, для якої параметри ; ;  визначаються як глобальні змінні, а значення х передається параметром. Формула розрахунку:

 де ; ; ; ; .

1. Вхідні дані ввести, а результати вивести, використовуючи потокове введення-виведення даних. В першому рядку кожної програми записати

*// ОПІ-41 Група № Прізвище Номер ЛР*

1. Результати надсилати на електронну адресу викладача [**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com)у вигляді cpp-файлу з іменем у форматі

**<Номер групи><Номер лабораторної><Прізвище англійською>**

Наприклад, МІВТ-МНТ-ЕТ-41-07Ivanov.cpp.

Іншим рішенням є надсилання поштою посилання на текст програми за URL адреси, яку надає C++Shell, вказавши в темі листа, номер групи прізвище студента та номер ПР.

В темі листа вказати, номер групи, прізвище студента та номер ПР як "ПР№5".

**Строк відсилки ЛР для МІВТ/ МНТ/ЕТ -41 08.11.2022**

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, В темі листа вказати, номер групи, прізвище студента та номер ПР як "ПР№7 В темі листа вказати, номер групи, прізвище студента, номер ПР та фразу "Запитання".

**Теоретичні відомості.**

**Покажчики** — це змінні, котрі містять адресу пам’яті, розподіленої для об’єкта відповідного типу. При оголошенні змінної-покажчика слід вказати тип даних, адресу яких буде містити змінна, та ім’я покажчика з символом «\*».

Загальний формат опису покажчика має вигляд:

**тип \*ім’я;**

де **тип** — тип значень, на який вказує покажчик;  
**ім’я** — ім’я змінної-покажчика;  
«\*» — операція над типом, що читається «покажчик на тип».

Наприклад:

**int \*рn** – покажчик на ціле значення;  
**float \*pf1, \*pf2;** — два покажчики на дійсні значення.

Покажчики не прив’язують дані до якого-небудь визначеного імені змінної і можуть містити адреси будь-якого неіменованого значення. Існує адресна константа **NULL**, що означає порожню адресу.

Мова C++ налічує лише дві операції, які стосуються адрес змінних, а саме:

**«&»** — **операція взяття адреси** («адреса значення»);

**«\*»** — **операція розіменування** («значення за адресою»).

Операція взяття адреси **«&»** застосовується разом зі змінною і повертає адресу цієї змінної. Операція розіменування «\*» використовується разом з покажчиками і бере значення, на яке вказує змінна-покажчик, розташована безпосередньо після символу «\*».

Оголошення покажчиків можна здійснити одним з таких способів:

**<тип> \*ptr;**  
**<тип> \*ptr = <змінна-покажчик>;**  
**<тип> \*ptr = &<ім’я змінної>;.**

Наприклад:  
**int \*ptx, b; float у;** — оголошені змінна-покажчик **ptx** та змінні **b і у**;

**float \*sp = &у;** — покажчику **sp** присвоюється адреса змінної **у**;

**float \*р = sp;** — покажчику **р** присвоюється значення (адреса значення), яке міститься в змінній **sp**, тобто адреса змінної **у**.

При оголошенні покажчиків символ «\*» може знаходитися перед ім’ям покажчика або відразу після оголошення типу покажчика і поширювати свою дію тільки на одну змінну-покажчик, перед якою він записаний:

**long \*pt;   long\*Uk;   int \*ki, x, h;** — оголошення описів.

За потреби для опису покажчика на комірку довільного типу замість ідентифікатора типу записується слово **void**, а саме:

**void \*р, \*pt;** — опис двох покажчиків на довільний тип даних.

**Перед використанням покажчика у програмі його обов’язково необхідно ініціювати**, іншими словами, необхідно присвоїти адресу якого-небудь даного, інакше можуть бути непередбачені результати.

Для одержання доступу до значення змінної, адреса якої зберігається в покажчику, досить у відповідному операторі програми записати ім’я покажчика з символом «\*» — здійснити операцію розіменування.

Розглянемо фрагмент програми з поясненнями:

**int \*р, \*р1;** — оголошені два покажчики на комірку пам’яті типу **int**;

**int х = 12, у = 5, m[7];** — оголошені змінні **х**, **у** і масив **m**, змінні ініційовані;

**р = &у;**     // р (&у); — покажчику **р** присвоєна адреса змінної **у**.

**cout << “Адреса р ” << р << “Значення за цією адресою = ” << \*р;**

Виведеться адреса комірки пам’яті, де записана змінна **у** і значення цієї змінної (**тобто 5**).

Використовуючи запис **х = \*р;**, одержимо **х = 5** тому, що **\*р = у = 5;**.

Змінити величину параметра **у** можна так:

**у = 10;**         // \*р= 10;  
**\*р = \*р+5;**    //у +=5;.

Остання операція означає збільшення значення змінної цілого типу на **5**, тобто **у= 15**.

При ініціюванні покажчиків їм можна присвоювати або адресу об’єкта (змінної), або адресу конкретного місця пам’яті (масиву), або число 0 (нуль), а саме:

**int \*pt = (char \*) 0x00147;** — присвоюється адреса комірки;

**int \*arrpt = new int [10];** — визначається початкова адреса розміщення динамічного масиву;

**char \*р = 0;** — здійснюється ініціювання нулем.

Оскільки покажчики — це спеціальні змінні, то в операціях з іншими покажчиками вони можуть використовуватися без символу «\*», тобто без розкриття посилання, наприклад:

**float \*pt1, \*pt2, х=15, m[5];**

**pt1 = &x;**

**pt2 = pt1;**

**pt1 = m**;         //pt1 = &m[0];

де **m** — ім’я масиву, що розглядається як спеціальний покажчик-константа.

***Приклад***

**#include <iostream.h>**

**#include <Windows.h>**

**using namespace std;**

**int main ( )**

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

**int x = 10;**

**int \*px (&x); // int \*px = &x;**

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

**cout << "\*px =" << \*px << endl;**

**x \*= 2; //x=x\*2;**

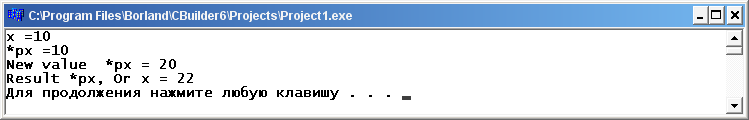
**cout << "New value \*px = " << \*px << endl;\***

**px += 2; // \*px=\*px + 2;**

**cout << "Result \*px, Or x = " << x << endl;**

**system("pause"); //затримка екрану**

**}**

Результат виконання програми:  


Для змінної-покажчика існує своя адреса і тому будуть доцільними записи:

**int \*pt1, \*pt2;**

**pt1 = (int\*) &pt2;** — покажчику **pt1** присвоюється адреса пам’ятi де розташована змінна **pt2**.

**Обмеження на застосування операції взяття адреси**:

* не можна визначати адресу літеральної константи (оскільки для неї не виділяється комірка пам’яті), тобто такий запис, як **vp = &345;** — неприпустимий;
* не можна визначати адресу результату арифметичного виразу, тобто запис **vp = &(x + y);** теж неприпустимий.

**Дозволені операції для змінних-покажчиків:**

* операція розіменування «\*»;
* операція взяття адреси «&»;
* операція присвоювання «=»;
* операції інкремент «++» і декремент « –-»;
* операції додавання «+» і віднімання «-»;
* операції відношення (порівняння) покажчиків однакового типу: «==», «!=», «<», «<=», «>», «>=».

**Функції**

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

**static int**

**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.

**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**.

В C++ передача за посиланням може здійснюватися двома способами:

* використовуючи безпосередньо посилання;
* за допомогою вказівників.

Синтаксис передачі з використанням посилань має на увазі застосування як аргумент посилання на тип об'єкта. Наприклад, функція

**double** Glue ( **long**& x, **int**& y ) ;

одержує два посилання на змінні типу **long** і **int**. При передачі у функцію параметра за посиланням компілятор автоматично передає у функцію адресу змінної, зазначеної в якості аргументу. Ставити знак амперсанта перед аргументом у виклику функції не потрібно. Наприклад, для попередньої функції виклик з передачею параметрів за посиланням виглядає в такий спосіб:

c = Glue (a, b) ;

Приклад прототипу функції при передачі параметрів через вказівник:

**void** Setnumber ( **int**\*, **long**\* ) ;

Тоді виклик функції має наступний вигляд:

Setnumber (&n,&a ) ;

Приклад функції, яка приймає в якості параметра дві змінні, та міняє їх місцями(параметри передаються з використанням посилань):

void swap(int &x, int &y)

{

int temp=x;

x=y;

y=temp;

}

int \_tmain()

{

int a=3,b=4;

swap(a,b);

cout<<"a="<<a<<"b="<<b<<'\n';

system("pause");

return 0;

}

Приклад функції, яка приймає в якості параметра дві змінні, та міняє їх місцями(параметри передаються з використанням вказівників):

void swap (int \*x, int \*y)

{

int temp=\*x;

\*x=\*y;

\*y=temp;

}

int \_tmain()

{

int a=3,b=4;

swap (&a,&b);

cout<<"a="<<a<<"b="<<b<<'\n';

system("pause");

return 0;

}

Якщо передати параметри за значенням, зміни не будуть збережені, бо в функцію буде передано копії змінних, а не їх адреси.

Крім того, функції можуть повертати не тільки значення деякої змінної, але й вказівник або посилання. Наприклад, функції, прототип яких:

**іnt\*** Count ( **int** ) ;

**іnt&** Increase ( ) ;

повертають вказівник і посилання відповідно на цілу змінну типу **int**. Слід мати на увазі, що повернення посилання або вказівника з функції може привести до проблем, якщо змінна, на яку робиться посилання, вийшла з області видимості. Наприклад,

**int**& func () { **int** x ; **return** x ; }

У цьому випадку спроба повернути посилання на локальну змінну х приведе до помилки, яка, на жаль, з'ясується тільки в ході виконання програми.

Ефективність передачі адреси об'єкта замість самої змінної відчутна й у швидкості роботи, особливо, якщо використовуються великі об'єкти, зокрема масиви.

Якщо потрібно у функцію передати досить великий об'єкт, однак його модифікація не передбачається, на практиці використовується передача константного вказівника. Даний тип виклику припускає використання ключового слова **const**, наприклад, функція

**int**\* **const** Fname ( **int**\* **const** Number )

приймає й повертає вказівник на константний об'єкт типу **int**. Будь-яка спроба модифікувати такий об'єкт у межах тіла викликуваної функції викличе повідомлення компілятора про помилку. Розглянемо приклад, що ілюструє використання константних вказівників.

**#include** <iostream.h>

**int**\* **const** call ( **int**\* **const** ) *// прототип функції, яка приймає* *й*

*// повертає константний вказівник типу int*

**void** main ()

{

**int** X = 13 ; *// оголошення змінної х її ініціалізація*

**int**\* px = &X ; *// оголошення вказівника на змінну х його ініціалізація*

call ( px ) ; *// виклик функції*

}

**int**\* **const** call ( **int**\* **const** x ) *// заголовок функції*

{

cout << \*х ; *// виводить на екран значення змінної, на яку вказує вказівник х*

// \*х++ ; *// не можна модифікувати об'єкт!*

**return** x ;

}

Замість наведеного вище синтаксису константного вказівника як альтернативи при передачі параметрів можна використовувати константні посилання, що мають той же зміст, що й константні вказівники.

**#include** <iostream.h>

**const** **int**& call ( **const** **int**& x ) *// заголовок функції, яка приймає* *й*

{*// повертає константне посилання типу int*

cout << x ; *// виводить на екран значення змінної, на яку посилається посилання*

// x++ ; *// не можна модифікувати об'єкт!*

**return** x ;

}

**void** main ()

{

**int y = 13 ; // оголошення змінної х її ініціалізація**

**int& rx = y ; // оголошення посилання на змінну х її ініціалізація**

**cout<<"\nПовертаєме значення="<<call ( rx ) ; // виклик функції**

**system("pause");**

}