МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет "Львівська політехніка"

ОСНОВИ РОБОТИ З ФУНКЦІЯМИ В С

ІНСТРУКЦІЯ

до лабораторної роботи №5 з курсу
"Основи програмування"
для базового напрямку "Програмна інженерія"

Затверджено
На засіданні кафедри
програмного забезпечення
Протокол № від

ЛЬВІВ – 2018

1. МЕТА РОБОТИ

Мета роботи – здобути практичні навики створення та застосування функцій у мові С .

2. ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1. Поняття функцій та їх роль у програмуванні

У процесі програмування реальних практичних задач дуже часто виникає ситуація коли на різних етапах розв'язання вихідної задачі доводиться неодноразово розв'язувати деяку часткову підзадачу. Причому алгоритм розв'язання цієї підзадачі передбачає виконання тої ж самої послідовності дій, але над різними наборами даних. Для підвищення ефективності програми та її простоти і наглядності, групу операторів, які реалізують таку послідовність дій, оформляють у вигляді самостійної програмної одиниці (підпрограми). У мові С такі підпрограми реалізуються у вигляді ***функцій***. Вони записуються в коді програми лише один раз, а у відповідних місцях програми (тобто там де потрібно розв'язати цю часткову підзадачу) забезпечується лише звертання до них (виклик функції). Така техніка розробки програм, по-перше, дозволяє уникнути повторення в програмі окремих фрагментів коду, а, по-друге, забезпечує повторне використання програмного коду, тобто, використання існуючих функцій як стандартних блоків для створення нових програм. Для цього кожна функція повинна розв'язувати одну чітко визначену задачу, а ім'я функції має наочно відображати цю задачу. Тоді сама програма на С, а точніше головна функція main(), буде просто містити виклики таких функцій у потрібному порядку, який забезпечує виконання алгоритму розв'язання вихідної задачі. Більше того, при належному розбитті (декомпозиції) вихідної складної задачі на послідовність простіших підзадач, розв'язання яких буде здійснюватися в окремих функціях, процес створення програми з таких стандартизованих функцій буде набагато ефективнішим, простішим і швидшим за рахунок можливості виконувати розробку окремих функцій різними програмістами і більш-менш незалежно один від одного. Такий підхід до розробки програм отримав назву *модульне програмування*, оскільки такі підпрограми можна роглядати як окремі самостійні модулі зі своїми вхідними та вихідними даними.

У мові С підпрограми реалізуються у вигляді функцій, які вводяться у програму за допомогою відповідного опису.

2.2 Опис функції

Синтаксис мови С передбачає такий формат опису функції:

```
[тип_результату] ім'я_функції(список_параметрів)
{
  тіло функції
}
```

Поле *тип_результату* задає тип значення, що повертається функцією. У ролі типу результату функції може використовуватися ключове слово ***void***, яке означає, що *функція не повертає* ніякого значення.

Поле *ім'я_функції* задає унікальний в межах програми ідентифікатор за допомогою якого можна звертатися до функції (викликати функцію). В мові С він трактується як особливий тип вказівника, який називається *вказівником на функцію*. Значенням цього вказівника є адреса точки входу в функцію. Зустрівши визначення функції, компілятор створює самостійну секцію коду програми, що на етапі компонування об'єднується з іншими функціями. Початкова адреса цієї секції і є значенням вказівника на функцію.

Поле *список_параметрів* представляє собою список так званих параметрів для позначення вхідних даних функції. Тому ці параметри часто називають *формальними параметрами*. З точки зору синтаксису мови С кожний формальний параметр описується аналогічно до звичайної змінної, тобто опис формального параметру включає задання типу

даних та ідентифікатора. Якщо функція потребує декілька формальних параметрів, то вони задаються у списку послідовно, розділяючись комою, але кожний параметр у цьому списку описується за наведеним вище правилом (тобто для кожного параметру треба вказати свій тип, навіть якщо всі вони мають одинаковий тип). У випадку, коли функція не має параметрів, список формальних параметрів або задається ключовим словом void, або є порожнім (але круглі дужки у цьому випадку залишаються).

Сукупність різних оголошень та операторів, які знаходяться між парою фігурних дужок в описі функції утворюють *тіло функції*.

Приклад: опис функції знаходження найбільшого серед двох заданих значень

```
int max(int a, int b)
{
      if(a>b)
            return a;
      else
            return b;
}
```

Тут, у ролі *тип_результату* виступає цілий тип int, `max` – є іменем функції, а *список_параметрів* має вигляд `int a, int b` , і він означає, що наша функція має два формальних параметри типу int. Оператор if утворює тіло функції.

2.3. Вихід з функції

Існують два способи завершення виконання функції і повернення у точку програми, з якої здійснено виклик. Перший - завершення виконання функції з використанням оператора повернення **return**, як показано у попередньому прикладі. Цей спосіб найчастіше застосовують тоді, коли функція має тип результату відмінний від void. Причому, як видно з попереднього прикладу, в тілі функції може бути декілька операторів return. Цю саму функцію max можна було б реалізувати і таким чином:

```
int max(int a, int b)
{
      int rez;

      if(a>b)    rez=a; else rez=b;

      return rez;
}
```

У даному випадку, функція спочатку обчислює результат у змінній `rez`, а потім повертає результат через оператор return.

Інший спосіб завершення виконання функції полягає у послідовному проходженні всіх операторів тіла функції до закриваючої фігурної дужки. Цей спосіб можливий лише для функцій з типом результату void, хоча як показує наступний приклад він не є єдино можливим способом виходу з функцій типу void.

```
void divide( float x, float y )
{
  float z;
  if (x == 0) return; /* не можна ділити на нуль*/
  z = y / x;
  printf("z=%lf",z);
}
```

Отже, якщо функція має тип результату відмінний від void, то єдиний спосіб виходу з неї – це оператор return (якщо звісно ми не хочемо отримати повідомлення компілятора про синтаксичну помилку).

2.4. Виклик функції

Виклик функції здійснюється в потрібному місці програми за її іменем (аналогічно як і для стандартних бібліотечних функцій) у такому загальному форматі:

ім'я_функції([список_аргументів])

Поле *список_аргументів* задає значення фактичних параметрів, які підставлються у відповідні формальні параметри. На відміну від списку формальних параметрів у списку аргументів не потрібно вказувати тип аргумента, достатньо записати лише ідентифікатор. В загальному випадку аргумент може задаватися виразом відповідного типу, тобто типу який співпадає з типом відповідного формального параметра в описі функції. Під час виклику функції у формальні параметри підставляються значення аргументів у тому порядку, в якому вони задані при виклику функції. Кількість формальних параметрів функції та аргументів повинні співпадати (за винятком випадку коли формальні параметри мають значення за замовчуванням). Якщо у функції відсутні формальні параметри, то при виклику такої функції повинен також бути відсутнім і список аргументів (але пара дужок () при виклику залишається).

Якщо функція має тип результату відмінний від void, то її виклик має бути записаний як частина певного виразу або оператора (найчастіше це є оператор присвоєння або оператор виведення). Так, наприклад, для описаної вище функції *max* правильними, як з синтаксичної, так і з семантичної точок зору, є такі можливі виклики:

```
z = max(x,y);
```
або
```
printf("%lf",max(x+y, 25));
```
або
```
if (max(a,b)>0) c=a;
```

У той же час виклики цієї ж функції вигляду `max(x,y);` хоча і є допустимими синтаксично, семантично (тобто за смислом) позбавлені сенсу, оскільки результат, який функція повертає у точку виклику не обробляється тим чи іншим чином. Тим більше, не можна функції присвоювати значення: `max(x,y)=20;`

Якщо функція має результат типу void, то єдиним можливим способом звертання до неї є виклик у вигляді окремого оператора, наприклад (для описаної вище функції *divide*):

```
divide(x,y);
```

Для цієї ж функції неправильними будуть такі виклики:

```
printf("%lf",divide(x,y));
```
або
```
z= divide(x,y);
```

2.5. Прототип функції

Виклик функції означає використання цієї функції. Тут можна провести аналогію зі змінними: оголошення змінної ще не означає, що вона використовується, лише виконання певних операцій над змінною (наприклад, операції присвоєння) приводить до її реального використання. У той же час ми знаємо, що використання змінної без її оголошення породжує помилку. Описана ситуація залишається в силі й у випадку із функціями: заборонено використовувати функцію, яка не була оголошеною. Таке "попереднє" оголошення функції прийнято називати ***прототипом функції***. Для визначення прототипу функції потрібно задати тип результату функції, її ім'я та список типів формальних параметрів у такому форматі:

[тип_результату] ім'я_функції(список_типів_параметрів);

Наприклад, прототипи розглянутих раніше функцій (*max* та *divide*) мають такий вигляд:

```
int max(int, int);
void divide(float, float);
```

Зауважимо, що прототип функції співпадає із заголовком в описі функції за виключенням того, що в прототипі можуть бути відсутні імена формальних параметрів.

Дозволено також вказувати в прототипі імена параметрів, відмінні від тих, що використовуються в описі функції, наприклад оголосити так:

```
int max(int ma, int mb);
```

```
void divide(float da, float db);
```

Прототип функції виконує два завдання. Перше полягає в ідентифікації типу значення, яке поверне функція, так, щоб компілятор міг згенерувати коректний код для типу даних, що повертаються. Друге завдання - у визначенні типу та кількості аргументів, які використовуються функцією. Використовуючи прототип, компілятор може виконати, ретельний контроль числа аргументів і відповідність їхніх типів у викликах функції та її описі. Більше того, реалізація другого завдання прототипу функції передбачає можливість автоматичного перетворення типів фактичних аргументів до типу, що зазначений у прототипі функції. Таке перетворення типів відбувається за так званими *правилами приведення типів С*, суть яких полягає в наступному: якщо типи формального параметру і аргумента функції не співпадають (але вони відносяться до одного з вбудованих типів мови С), то відбувається автоматичне приведення значення “нижчого” типу до значення “вищого” типу згідно ієрархії основних типів, яка наведена у табл.1 (в порядку від “вищого” типу до “нижчого”).

Таблиця 1. Ієрархія приведення вбудованих типів мови С

Типи даних
long double
double
float
unsigned long int (синонім unsigned long)
long int (синонім long)
unsigned int (синонім unsigned)
int
unsigned short int (синонім unsigned short)
short int (синонім short)
unsigned char
char

Якщо аргумент (фактичний параметр) функції має “нижчий” тип ніж тип відповідного формального параметра, то таке перетворення типу відбувається без втрати значення аргумента. У протилежному випадку, високою є імовірність втрати значення аргумента, наприклад, якщо відбувається перетворення числа типу float у число типу int. Таку ситуацію наочно демонструють результати виконання наступної програми.

```
#include<stdio.h>
#include<conio.h>

int square(int);  // прототип функції

void main()
{
      printf("%d",square(4));   // хочемо отримати квадрат від 4
      printf("%d",square(4.7)); // хочемо отримати квадрат від 4.7
      _getch();
}

int square (int x)   // опис функції
{
      return x*x;
}
```

Тут, в обох випадках отримаємо значення 16, незважаючи на те, що в другому виклику функції `square` аргумент має значення типу double. Зауважимо, також, що прототипи функцій зазвичай розташовують на початку тексту програми відразу після директив препроцесора #include.

2.6. Способи передачі агрументів

Мова С має два способи передачі аргументів: передача аргументів за значенням і передача аргументів за посиланням(адресою).

2.6.1. Виклик функції з передачею значення

При цьому способі виклику функції значення аргументу копіюється у відповідний формальний параметр функції. Тому зміни значення цього параметра всередині функції не впливають на значення змінних, які використовувалися для виклику. В усіх розглянутих раніше прикладах використовувався сам цей спосіб передачі аргументів. Саме через те, що у стек записуються лише копії значень аргументів функції (стек-це спеціальна ділянка оперативної пам'яті через яку відбувається передача аргументів у функції), у ролі аргументів можуть виступати як змінні та константи, так і вирази, які повертають значення потрібного типу (тобто типу формального параметра.

Приклад.

```
/* приклад виклику функції з передачею значення */
#include<stdio.h>
#include<conio.h>

int square(int);  // прототип функції

void main()
{
      int p=5;
      printf("p=%d p*p=%d p=%d",p, square(p),p);
      _getch();
}

int square (int x)    // опис функції
{     x*=x; return x; }
```

У цьому прикладі значення аргументу для функції *square()* (тобто 5) скопіювалося у параметр `x`. Коли відбулося присвоєння `x*=x`, змінилося значення лише формального параметру `x`. Змінна `p`, яка використовувалася у ролі аргумента при виклику *square()*, і надалі має значення 5, тому на екран виведеться такий рядок: *p=5 p*p=25 p=5*.

2.6.2. Звертання за адресами параметрів

При цьому способі виклику функції передається не копія значення аргумента, а його адреса. Це означає, що всередині функції, яка викликається, ми маємо доступ до оригінала значення аргумента (через його адресу). Тому будь-яка модифікація відповідного формального параметра всередині функції означає й одночасну зміну аргумента.Очевидно, що відповідний формальний параметр функції повинен бути вказівником , базовий тип якої збігається з типом змінної. Таку організацію взаємозв'язку параметрів, коли у функцію передаються адреси змінних називають звертанням через посилання (call by reference). Щоб переставити місцями значення двох змінних , функція повинна оперувати з їх адресами, тобто формальні параметри функції повинні бути вказівниками на змінні, значення яких пересилаються.

Якщо ми запишемо функцію обміну таким чином:

```
void Swap(int a, int b)
{
int d = a;
a=b;
b=d;
}
```

То при виклику функції, щоб поміняти місцями значення двох змінних:

```
int alpha=26, beta = 18;
Swap(alpha, beta);
```

Значення `alpha` і `beta` не зміняться. Хоча в тілі функції `Swap` формальні параметри `a` і `b` обмінюються значеннями, це не впливає на значення фактичних параметрів.

Для того, щоб переставити місцями значення двох змінних , функція повинна оперувати з їх адресами таким чином:

```
void Swap(int *pa, int *pb)
{
int d = *pa;
*pa=*pb;
*pb=d;
}
```

```
 /* приклад програми, яка переставляє у зворотному порядку елементи
 масиву з індексами від 4 до 15  */
#include<stdio.h>
#include<conio.h>


void Swap(int *p1, int *p2);

int main(void)
{
int ar[]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18};
int n1,n2,k;
int *p, *q;
k=sizeof(ar)/sizeof(int);
n1=4;
n2=15;
if (n1>n2)
Swap(&n1,&n2);
for(p=ar+n1, q=ar+n2;p<q;p++,q--)
Swap(p,q);
printf("\n Результат переставлення елементів %d - %d\n\t",n1+1,n2+1);
for(p=ar; p<ar+k;p++)
printf("%d\t",*p);
return 0;
}

void Swap(int *pa, int *pb)
{
int d = *pa;
*pa=*pb;
*pb=d;
}
```

Результат роботи програми:

Перший раз викликаємо функцію `Swap` для впорядкування значень `n1 i n2`, передаючи у функцію адреси цих змінних. Далі її циклічно використовуємо для переставлення місцями двох відповідних елементів масиву. При цьому параметрами функції є вказівники на елементи, які переставляються.

2.7. Локальні та глобальні змінні

Усі змінні, оголошені всередині довільної функції є ***локальними*** змінними цієї функції. Це означає, що їх значення зберігаються тільки на час виконання цієї функції. Інколи кажуть, що локальні змінні мають *область видимості* обмежену тілом функції, в якій вони оголошені.

Усі формальні параметри функції розглядаються також як локальні змінні цієї функції. Усі змінні, оголошені поза межами всіх функцій з яких складається програма, називаються ***глобальними***. Доступ до таких змінних можливий у будь-якій функції. Іншими словами, область видимості глобальних змінних обмежена лише файлом у якому знаходиться вихідний код програми.

У мові С поняття локальності змінних є дещо глибшим і розширеним у порівнянні з іншими мовами програмування (наприклад, з мовою Паскаль). Локальні змінні можна оголошувати у довільному місці функції в межах так званого *програмного блоку* під яким розуміють довільну послідовність оголошень та операторів, яка знаходиться між парою фігурних дужок {}. Тіло довільної функції в С також розглядається як програмний блок. У цьому випадку змінна є локальною по відношенню до блоку в якому вона оголошена і її область видимості обмежена цим блоком. Більше того, в С дозволяється використовувати однакові ідентифікатори для позначення змінних в різних блоках. Це можливо завдяки вбудованому механізму перекриття імен мови С. Наступний приклад програми демонструє ці можливості.

```
#include <stdio.h>
#include <conio.h>

void a (void);    // прототип функції
int x=1;          // глобальна змінна

void main()
{
 int  x = 5;    // локальна змінна функції main
 printf( "x=%d\n " , x );
 {
      int x=7; // локальна змінна внутрішнього блоку main
      printf( "x=%d\n " , x );
 }  _
 printf( "x=%d \n" , x );
 a();
 _getch();
}

void a(void)     // опис функції a_
{
 printf( "x=%d \n" , x++ );
}
```

Результат виконання цієї програми зображено на рис.1.

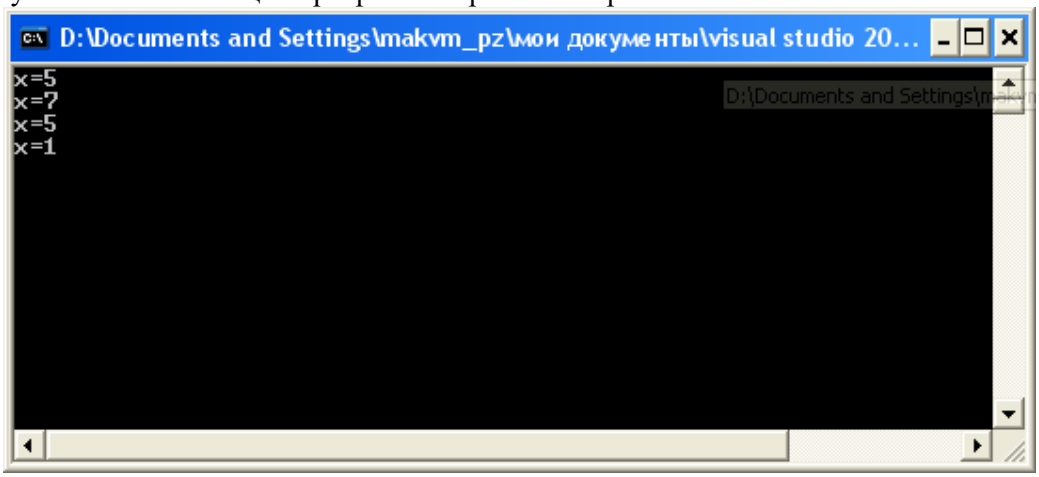

Рис.1. Результат виконання програми, яка демонструє механізм перекриття імен мови С

3. КОНТРОЛЬНІ ЗАПИТАННЯ

1. Яке призначення функцій у програмуванні?
2. Який загальний вигляд опису функції у мові С?
3. Що таке формальні параметри функції? Для чого вони призначені?
4. Як здійснюється вихід з функції?
5. Як здійснюється виклик функції?
6. Що таке аргументи функції? Яка різниця між фактичними та формальними параметрами функції?
7. Що таке прототип функції? Для чого він призначений?.
8. У чому полягають правила приведення типів у мові С?
9. Які способи передачі аргументів у функцію Ви знаєте?
10. Чому не можна задавати вираз у ролі аргумента коли передача аргумента у функцію відбувається за посиланням?
11. Що таке локальні змінні? А глобальні? Наведіть приклад.
12. У чому полягає суть механізму перекриття імен мови С?

4. ЛАБОРАТОРНЕ ЗАВДАННЯ

1. Ознайомитися з теоретичним матеріалом викладеним вище в даній інструкції і виконати приклади програм.
2. Одержати індивідуальне завдання.
3. Розробити алгоритм розв'язання індивідуального завдання і подати його у вигляді блок-схеми.
4. Скласти програму на мові С у відповідності з розробленим алгоритмом.
5. Виконати обчислення по програмі.
6. Підготувати та здати звіт про виконання лабораторної роботи.

5. СПИСОК ЛІТЕРАТУРИ

1. Керниган Б., Ритчи Д. Язык программирования С. - М. - Финансы и статистика. - 1992. – 272 с.
2. Уэйт М., Прата С., Мартин Д. Язык С. Руководство для начинающих. - М. - Мир. - 1988. – 512 с.
3. К. Джамса. Учимся программировать на языке С++. М.: Мир, 1997. – 320 с
4. Герберт Шилдт. Полный справочник по С++. М. – С.-П.-К., Вильямс. – 2003. – 800 с.
5. Демидович Е. М. Основы алгоритмизации и программирования. Язык Си. (Учебное пособие). – Санкт-Петербург: "БХВ Петербург". – 2006. – 439 с.

6. ІНДИВІДУАЛЬНІ ЗАВДАННЯ

Протабулювати, задану згідно варіанту функцію, на проміжку $[a,b]$ з кроком h двома способами.

Для обчислення значення, заданої згідно варіанту функції y в точці $x \in [a,b]$:

1) першим способом: оголосити і реалізувати функцію мовою С, формальним параметром якої є x, а результатом функції є значення y;
2) другим способом: оголосити і реалізувати функцію мовою С, першим формальними параметром функції є x, другим формальним параметром є аргумент, куди буде повернено результат обчислення за формулою. Функція повертає:

$$\begin{cases} -1, & якщо \quad y<0; \\ 0, & якщо \quad y=0; \\ 1, & якщо \quad y>0. \end{cases}$$

Табуляцію оформити у вигляді окремої функції. Результати обчислень подати у вигляді таблиці.

Всі функції розмістити в заголовному файлі.

1. $f=\sin^2(x/2), a=0, b=\pi/2$;
2. $f=x^{15}\sqrt{1+3x^8}, a=0, b=1$;
3. $f=\cos^2(4x), a=-\pi/2, b=0$;
4. $f=\arcsin\sqrt{\dfrac{x}{1+x}}, a=0, b=3$;
5. $f=\dfrac{1}{\sqrt[3]{(x+1)^2}}, a=0, b=7$
6. $f=\sin x\sin 2x\sin 3x, a=0, b=\pi/2$;
7. $f=(1+\sin 2x)^2, a=0, b=\pi/4$;
8. $f=\dfrac{1}{\cos^2(x/3)}, a=0, b=\pi$;
9. $f=e^x\cos^2 x, a=0, b=\pi$;
10. $f=sh^4 x, a=0, b=\ln 2$;
11. $f=x\sqrt[3]{1-x}, a=1, b=9$;
12. $f=\dfrac{1}{x\sqrt{x^2-1}}, a=-3, b=-2$;
13. $f=\sqrt{1-\cos 2x}, a=0, b=100\pi$;
14. $f=\sqrt[3]{(2x-1)}, a=0, b=4.5$;
15. $f=x^2\cos x, a=0, b=2\pi$;
16. $f=\sqrt{1+2x}, a=0, b=4$;
17. $f=xe^{-x}, a=0, b=\ln 2$;
18. $f=x\sin x, a=0, b=\pi$;
19. $f=x\arctan(x), a=0, b=\sqrt{3}$;
20. $f=x^2\sqrt{4-x^2}, a=0, b=2$;
21. $f=x/(x^2+3), a=1, b=3$;
22. $f=x/\sqrt{(3x+2)}, a=0, b=2$;
23. $f=(3x^2+x-2)/(3x+1), a=0, b=1$;
24. $f=\cos^4 x, a=0, b=\pi$;
25. $f=\dfrac{\arcsin\sqrt{x}}{\sqrt{x(1-x)}}, a=0, b=1$;
26. $f=\dfrac{x\sin x}{1+\cos^2 x}, a=0, b=\pi$;
27. $f=(x\ln x)^2, a=1, b=e$.