

## Лекція №9

# Опис та використання функцій

## Навчальні питання

1. Створення й використання функцій
2. Аргументи функції
3. Рекурсивні функції



# Література

1. Керниган Б., Ритчи Д. Язык программирования Си ДПер. с англ., 3-е изд., испр. — СПб.: "Невский Диалект", 2001. - 352 с: ил.
2. Язык программирования C++. Лекция и упражнения. Учебник: Перевод с англ./Стивен Прата – СПб.: ООО «ДиаСофтЮП», 2005.-1104с.
3. Теренс Чан Системное программирование на C++ для UNIX: пер. с англ. – К.: Издательская группа ВHV, 1997. – 592с.
4. Уэйк М., Прата С., Мартин Д. Язык Си Руководство для начинающих: Пер. с англ.. – М: Мир, 1988.-512с.,ил.

# 1. Створення й використання функцій



# Типові етапи рішення задач (підзадачі)

	Етапи	Опис
1.	Постановка задачі та її аналіз	<ol style="list-style-type: none"> <li>1. Вхідні дані, отримані результати, вигляд подачі даних</li> <li>2. Умови отримання розв'язку задачі</li> <li>3. Результати, які вважаються вірними</li> </ol>
2.	Формалізація задачі, вибір методу розв'язання - математичне моделювання задачі	<ol style="list-style-type: none"> <li>1. Запис задачі за допомогою формул, графіків, рівнянь, нерівностей, таблиць тощо</li> <li>2. Математична модель задачі - зв'язок вихідних даних з вхідними даними за допомогою математичних співвідношень</li> <li>3. Методи розв'язку задачі</li> </ol>
3.	Складання алгоритму розв'язку задачі	<ol style="list-style-type: none"> <li>1. Алгоритм визначається обраним методом</li> <li>2. При складанні алгоритму необхідно враховувати властивості методу</li> </ol>
4.	Складання програми	Написання програми на мові програмування
5.	Тестування і відлагодження програми	<ol style="list-style-type: none"> <li>1. Перевірка правильності роботи програми за допомогою тестів</li> <li>2. виправлення наявних помилок</li> </ol> <p>Тест - спеціально підібрані вхідні дані та результати, які мають бути отримані в результаті обробки програмою цих даних</p>
6.	Остаточне виконання програми, аналіз результатів	<ol style="list-style-type: none"> <li>1. Виконання програми</li> <li>2. Проведення аналізу результатів</li> <li>3. Можлива зміна підходу до розв'язання задачі та повернення до першого етапу</li> </ol>

# Поняття функції

- модуль, що містить деяку послідовність операцій

*Процес розробки та реалізації* – це побудова операцій, що вирішують конкретну задачу

## *Розроблена функція*

- Окрема абстрактна операція
- Для використання необхідно знати інтерфейс функції
  - ✓ вхідні дані
  - ✓ результати виконання

# Приклади функцій

- Функції мови Сі

*printf()*, *scanf()*, *getchar()*, *putchar()*

- Створена функція

*main()*

# Використання функції забезпечує

- Позбавлення необхідності повторного програмування коду, що повторяється в програмі
- Застосування функції у різних програмах
- Полегшення читання програми, внесення змін і коректування помилок



# Відношення програміста до функцій -“Чорний ящик”

!!! Вміст (код) “чорного ящика” при використанні програміста не цікавить

*Програміст має знати* інтерфейс функції

- ✓ формат вхідної інформації
- ✓ формат повертаємих результатів
- Інтерфейс описується в прототипі функції

# Вхідні та вихідні дані функції

## *Вхідні дані*

- ✓ аргументи
- ✓ глобальні структури даних, що використовуються функцією

## *Вихідні дані*

- ✓ значення, які функція повертає
- ✓ глобальні дані, що модифікуються

# Код функції

- послідовність операторів над аргументами або змінними функції відповідного типу даних
- Змінні що містяться у функції - локальні змінні

# Синтаксис опису функції



Рис. 9.1

*тип\_поверт\_значення ім'я\_функції ([параметри])*

*{  
тіло функції  
}*

- ▣ **ім'я функції** - правильний ідентифікатор
- ▣ **тіло функції**
  - ✓ визначення й описи змінних
  - ✓ оператори
- ▣ **параметри** – аргументи функції
- ▣ **тип\_поверт\_значення** - в мові C є не обов'язковим

## Приклад

*/\* Обрахування квадрату  $x$ , опис функції в програмі \*/*

*float qvdr (float x)*

*{*

*return x\*x;*

*}*

# Передача фактичного аргументу у функцію

- Вимагає, щоб визначення функції включило в себе формальний аргумент того ж типу, що й фактичний

## *Опис*

```
void function1(int); //опис прототипу функції, поміщаємо до файлу опису  
void function2(int *); //опис прототипу функції, поміщаємо до файлу опису
```

```
int x=10; //в тілі головної функції
```

## *Передача значення змінної x*

```
function1(x); //в тілі головної функції
```

## *Передача адреси змінної x*

```
function2(&x); //в тілі головної функції
```

# Завершення виконання функції

- Виконано весь код тіла функції
- Зустрівся оператор *return*

# Функція повертає

- ✓ значення
  - ✓ покажчик на масив
  - ✓ покажчик на функцію
- 
- Тип *void* - функція не має повертаємого значення
  - Тіло функції при поверненні значення *x* завершується оператором

*return (x);*

!!! Функція не може повертати масив або функцію



# Оператор return

- 1) Завершує виконання коду функції
  - 2) Повертає результат роботи функції в основну програму
  - 3) Передає управління в основній програмі, наступному операторові який знаходиться за викликом функції
- !!! Завершення коду відбувається навіть у тому випадку, якщо оператор **return** є не останнім оператором у функції*

```
main( )
{
    int a=10;
    d=abs(a);
    return; /* повертаєме значення відсутнє */
}
/* Функція, що обчислює абсолютну величину числа*/
abs(int x) {
    if(x < 0) return(-x);
    else return(x);
    printf("Робота завершена!\n");
}
```

# Оголошення функцій

- Оголошенню функції можуть передувати специфікатори класу пам'яті *extern* або *static*
  - ▣ *static* обмежує видимість функції - невидима поза утримуючим її файлом
  - Якщо в описі не зазначений клас пам'яті - за замовчуванням *extern*
  - В середині функцій можливий виклик будь-якої іншої функцій
- !!! Неможливо **оголосити функцію** в середині тіла іншої функції
- !!! Якщо в програмі є звертання до функції - опис прототипу функції має розміщатися раніше її визначення і звернення

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

```
/* прототипу функції func1(), func2() рекомендується розмістити до файлу  
заголовку, приклад File10.h, а файл включити #include "File10.h" */
```

```
float func2(float , float );
```

```
void func1(char *);
```

```
/* головна програма */
```

```
main() {
```

```
    int a=4, b=5;
```

```
    func1("a*b= ");
```

```
    printf(" %f", func2(a, b));
```

```
    return 0;
```

```
}
```

```
/* визначення функцій func1() */
```

```
void func1(char *sh) {
```

```
    printf("%s", sh);
```

```
}
```

```
float func2(float x, float y) {
```

```
    return x*y;
```

```
}
```

# Приклад – рекомендуєма структура

```
/*Головна програма в Union10.c, до проекту необхідно включити всі перераховані файли */  
#include<stdio.h>  
#include "File10.h "  
main()  
{  
    float a=4, b=5;  
    func1("a*b= ");  
    printf(" %d", func2(a, b));  
    return 0;  
}  
/* File10.c - файл визначення функцій func1(), func2() */  
void func1(char *sh)  
{  
    printf("%s", sh);  
}  
float func2(float x, float y)  
{  
    return x*y;  
}  
/* File10.h - файл оголошення прототипу функцій func1(), func2() */  
void func1(char *);  
float func2(float, float);
```

# Питання



## 2. Аргументи функції



# Аргумент

- ▣ **Формальний аргумент** - змінна у визначенні (заголовку) функції
- ▣ **Фактичний аргумент** - конкретне значення, привласнене цій змінній при визові функції з програми
  - ✓ константа
  - ✓ змінна
  - ✓ складний вираз
- ▣ Незалежно від типу фактичного аргументу він спочатку обчислюється, а потім його величина передається функції

*/\*Головна програма Union10.c, до проекту необхідно включити всі перераховані файли \*/*

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

```
#include "File10.h"
```

```
main()
```

```
{
```

```
float a=4, b=5;
```

```
func1("a*b= ");
```

```
printf(" %d", func2(a, b)); /*фактичні аргументи a,b*/
```

```
return 0;
```

```
}
```

*/\* File10.c - файл визначення функцій func1(), func2() \*/*

```
void func1(char *sh) /* формальні аргументи *sh */
```

```
{
```

```
printf("%s", sh);
```

```
}
```

```
float func2(float x, float y) /* формальні аргументи x, y*/
```

```
{
```

```
return x*y;
```

```
}
```

*/\* File10.h - файл оголошення прототипу функцій func1(), func2() \*/*

```
void func1(char *); /*прототип – визначає тип формального аргументу *sh */
```

```
float func2(float, float);
```



# Робота компілятора при роботі з функцією

- 1) виділення ділянки пам'яті для формальних параметрів
- 2) обчислення значення фактичних параметри при виклику функції
- 3) запис значення фактичних параметрів в ділянки пам'яті, що виділені для формальних параметрів
- 4) виконання коду у тілі функції з використанням значень отриманих та внутрішніх об'єкт-параметрів
- 5) передача результатів в точку виклику функції

!!! Функція не має впливу на фактичні параметри головної

# Специфікації аргументів

## *Явні аргументи*

- Список специфікації аргументів, кількість і типи яких фіксовані та відомі в момент компіляції

## *Змінна кількість аргументів*

- Має один або кілька обов'язкових аргументів
- Інші аргументи не визначені на момент компіляції

# Передача аргументів за значенням

- Усі параметри, за винятком параметрів типу покажчик передаються за значенням

## *При виклику*

- ✓ у функцію передаються значення змінних
- ✓ функція не в змозі змінити значень в головній програмі, яка здійснює виклик

## Приклад передачі аргументу за значенням

*/\*Вивід значення та адреси аргументу функції в основній програмі та в тілі функції\*/*

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

```
void main( ) {
```

```
    int a=10;
```

```
    printf("before test :      a=%d, &a=%u \n", a, &a );
```

```
    test(a);
```

```
    printf("after test :      a=%d, &a=%u \n", a, &a );
```

```
    return;
```

```
}
```

```
/* Test1.c */
```

```
void test(int a) {
```

```
    a=15;
```

```
    printf(" in function test1: a=%d, &a=%u \n", a, &a );
```

```
}
```

```
/* Test1.h */
```

```
void test(int );
```

***Результат***

```
before test :      a=10, &a=55990
```

```
in function test1: a=15, &a=55994
```

```
after test :      a=10, &a=55990
```

# Особливість передачі аргументу за значенням

## При виклику функції

- ✓ відводиться пам'ять для локальних копій параметрів → призводить до збільшення об'єму задіяної пам'яті
- ✓ копіюється в неї фактичний параметр → витрачається час на копіювання

## При виході з функції - пам'ять звільняється

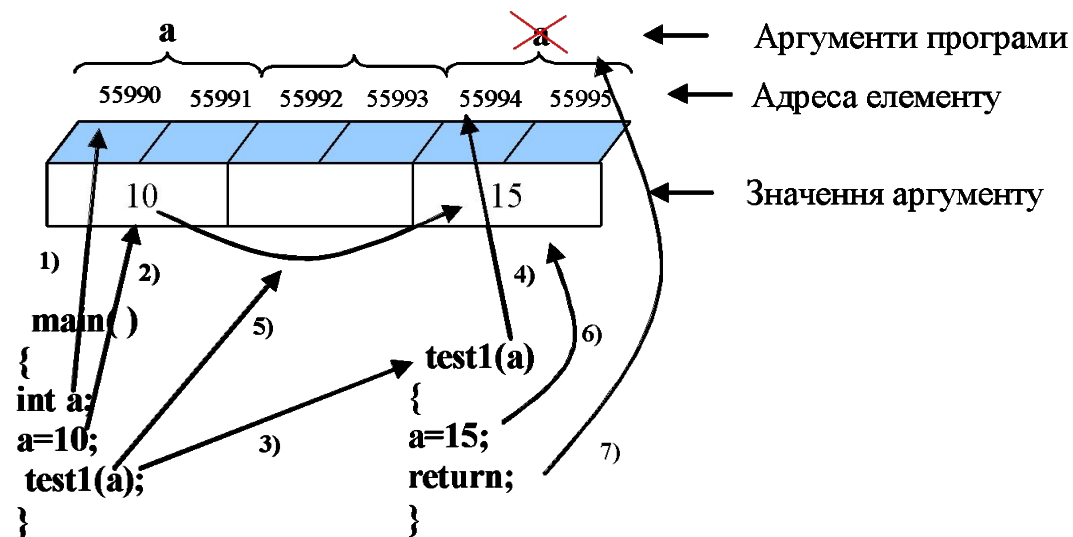


Рис. 9.2

# Передача аргументів за покажчиком

- Додатково об'єми пам'яті під формальні змінні не створюються → не витрачається додатковий час для копіювання даних в пам'яті

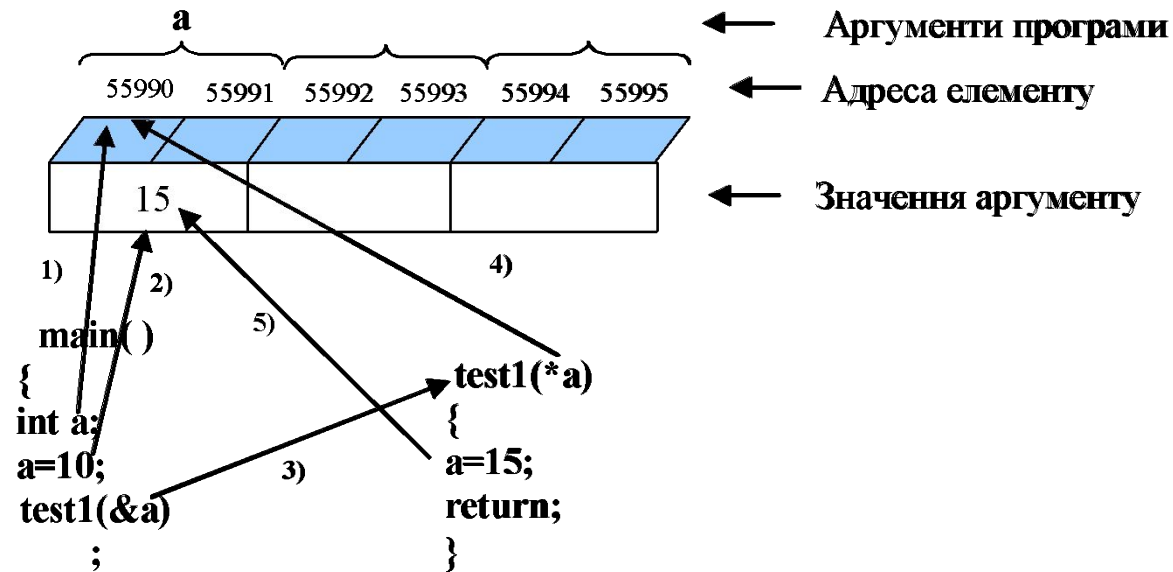


Рис. 9.3

## Приклад передачі аргументу за показником

*/\* Вивід значення та адреси аргументу функції в основній програмі та в тілі функції \*/*

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

```
void main() {
```

```
    int a=10;
```

```
    printf("before test :      a=%d, &a=%u \n", a, &a );
```

```
    test(&a);
```

```
    printf("after test :      a=%d, &a=%u \n", a, &a );
```

```
    return;
```

```
}
```

```
/* Test1.c */
```

```
void test(int *a) {
```

```
    *a=15;
```

```
    printf(" in function test1: a=%d, &a=%u \n", *a, a );
```

```
}
```

```
/* Test1.h */
```

```
void test(int *);
```

**Результат**

```
before test :      a=10, &a=55990
```

```
in function test1: a=15, &a=55990
```

```
after test :      a=15, &a=55990
```

# Функції із змінним числом аргументів

- для передачі у функції деякого числа фіксованих та невизначеного числа додаткових аргументів

## *Опис*

*тип ім'я\_функції(список аргументів, ...)*

## *Список аргументів включає*

- ✓ скінченне число обов'язкових параметрів - список не може бути порожнім
- ✓ невизначене число аргументів - ставиться три крапки



# Робота зі змінним числом аргументів

У *stdarg.h* визначений тип списку *va\_list* і три функції  
*va\_start()*, *va\_arg()*, *va\_end()*

- ❖ Функція *va\_start()* - починає роботу зі списком і встановлює покажчик *ap* на перший аргумент списку аргументів з невизначеним числом

*Синтаксис*  $\text{void va\_start(va\_list ap, lastfix)}$

*lastfix* – ім'я останнього фіксованого параметру

- ❖ Функція *va\_arg()* - повертає значення наступного аргументу зі списку

*Синтаксис*  $\text{void va\_arg(va\_list ap, type)}$

- Перед викликом *va\_arg()* значення *ap* повинне бути встановлене викликом *va\_start()* або *va\_arg()*
- Кожний виклик *va\_arg()* переводить покажчик на наступний аргумент

- ❖ Функція *va\_end()* - завершує роботу зі списком, звільняючи пам'ять

*Синтаксис*  $\text{void va\_end(va\_list ap)}$

```
//Функція сумує аргументи, признак кінця списку аргументів - 0
#include <stdio.h>
#include <stdarg.h>
#include "Test3.h"
int main(void)
{
    sum("Сума 1+2+3+4 рівна %d\n", 1,2,3,4,0);
    return 0;
}
/* Файл Test3.c */
void sum(char *msg, ...)
{
    int total = 0;
    va_list ap;
    int arg;
    va_start(ap, msg);
    while ((arg = va_arg(ap, int)) != 0)    total += arg;
    printf(msg, total);
    va_end(ap);
}
/* Файл Test3.h */
void sum(char * , ...);
```

# Питання



# 3. Рекурсивні функції



# Визначення

- ▣ *Рекурсія* – це спосіб організації обчислювального процесу, при якому функція звертається сама до себе
- ▣ *Рекурсивна функція* - якщо під час її виконання можливий повторний її виклик безпосередньо (прямий виклик) або шляхом виклику іншої функції, в якій міститься звертання до неї (непрямий виклик)

# Пряма рекурсія

- рекурсія, при якій всередині тіла деякої функції міститься виклик тієї ж функції

## *При виклику*

- ✓ в стеку створюється копія значень її параметрів, як і при виклику звичайної функції
- ✓ після чого управління передається першому оператору функції
- При повторному виклику цей процес повторюється

```
void funk(int i)
```

```
{  
  /* ... */  
  funk (i);  
  /* ... */  
}
```

# Непряма рекурсія

- рекурсія, що здійснює рекурсивний виклик функції шляхом ланцюга викликів інших функцій
- При цьому всі функції ланцюга, що здійснюють рекурсію, вважаються також рекурсивними

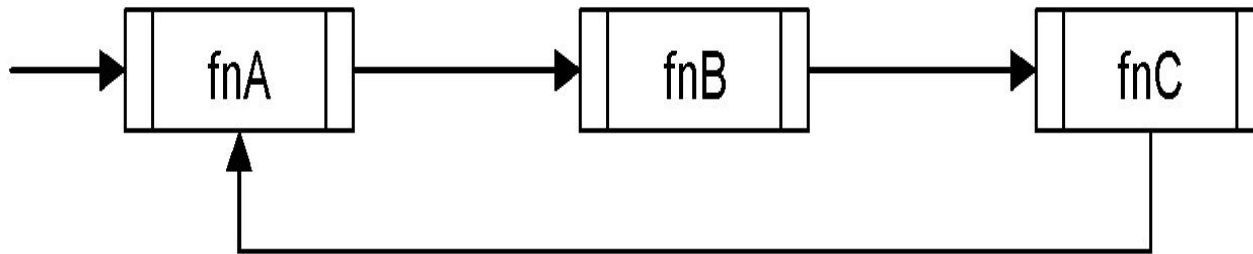


Рис. 9.4

# Приклад непрямої рекурсії

```
void fnA(int i);  
void fnB(int i);  
void fnC(int i);  
void fnA(int i) {  
    /* ... */  
    fnB (i);  
    /* ... */  
}  
void fnB(int i) {  
    /* ... */  
    fnC(i);  
    /* ... */  
}  
void fnC(int i) {  
    /* ... */  
    fnA(i);  
    /* ... */  
}
```



# Приклад рекурсивно обчислює $n!$

/\*зручно скористатися рекурсивним виразом  $n!=n*(n-1)!$  при  $n=6$ ,  $n!=?*$ \*/

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

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

```
double fact(int n) {
```

```
    if (n<=1) return 1;
```

```
    return (fact(n-1)*n);
```

```
}
```

```
void main()
```

```
{
```

```
    int n;
```

```
    double value;
```

```
    clrscr();
```

```
    printf("N=");
```

```
    scanf("%d",&n);
```

```
    value=fact(n);
```

```
    printf("%d! = %.50g",n,value);
```

```
    getch();
```

```
}
```

У файлі прототипу опис `double fact(int)` 41

# Кроки при розробці рекурсивних функцій

- 1) *Рекурсивне занурення функції* в саму себе - поки вибраний параметр не досягне граничного значення
  - Важлива вимога, щоб функція не створила нескінченну послідовність викликів самої себе
  - *Глибина рекурсії* - кількість кроків
- 2) *Рекурсивний вихід* - поки вибраний параметр не досягне початкового значення
  - Забезпечує отримання проміжних і кінцевих результатів

# Показчики на функції

- використовується для передачі функцій як параметрів іншим функціям
- За означенням показчик на функцію містить адресу першого байта або слова виконуваного коду функції

/\* показчик на функцію, що приймає два параметри типу float і повертає значення типу float \*/

**float (\*func) (float a, float b);**

# Особливість використання покажчика на функцію

- Для використання покажчика на функцію потрібно спочатку присвоїти йому значення адреси пам'яті, де розташована функція

## Приклад - помилка використання

```
#include<stdio.h>
#include<conio.h>
void main(void)
{
    void (*efct)(char *s); /* змінній-покажчику виділена ОП, але efct не містить
        значення адреси ОП для функції */
    efct("Error"); /* груба помилка – спроба працювати з неініціалізованим
        покажчиком*/
}
```

!!! Не путайте з описом *void \* efct(char \*s);*

```
#include<stdio.h>
#include<conio.h>
void print(char *s)
{
    puts(s);
}

main()
{
    void (*efct)(char *s);
    efct=&print;          /* efct=print */
    (*efct)("Function Print!"); /* efct("Function Print!"); */
}
```

**Результат**

Function Print!

# Питання

