Programming Fundamentals (CS-302)

(Functions & Pointers

Dr. Ihsan Ullah

Lecturer
Department of Computer Science & IT
University of Balochistan

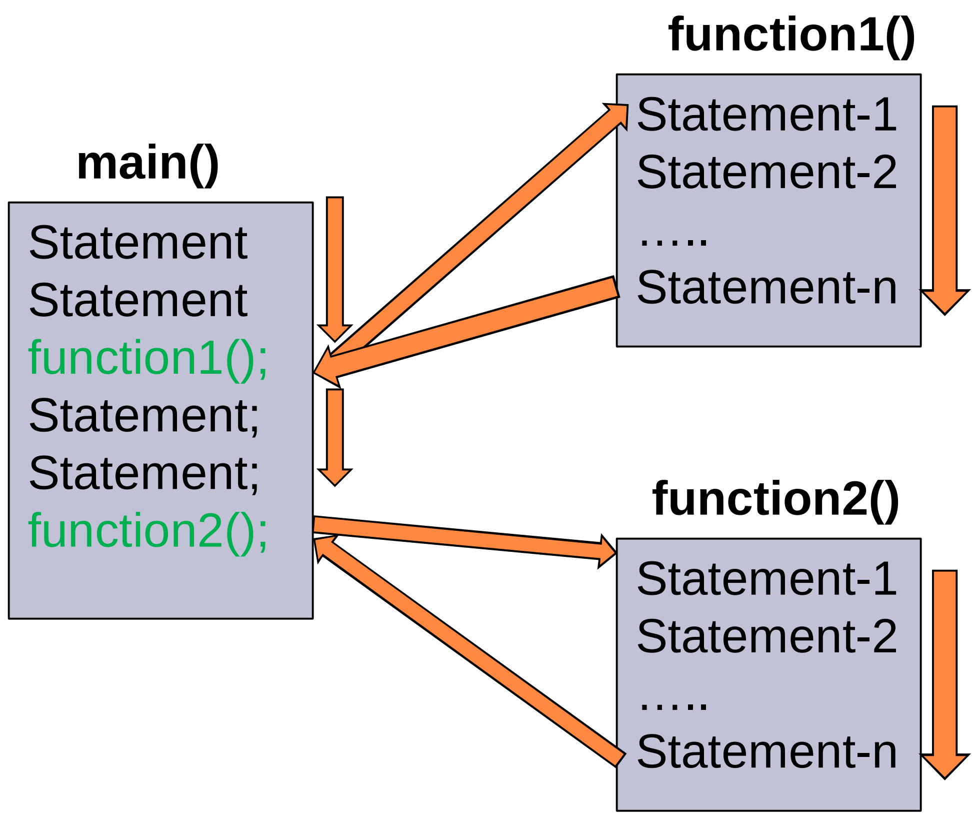
Outline

- Functions
  - Introduction
  - Function calling
  - Function types
    - Library function
    - User-defined functions
  - Function declaration
    - Arguments, return type
    - Passing by value
- Pointers
  - Passing by reference
  - Returning multiple values from a function

Functions: introduction

- A function is a block of statements that performs a specific task
- A program in C language may contain several functions
- Every C program contains at least one function
- Only one function of a C program must be main()
- There is no limit on the number of functions in a C program

Function calling



Function calling

- A function call transfers the control from the calling point to the function
- As a function finishes its task, control is transferred back to the subsequent statement from the point of calling
- Any function can call any other function
- Calling sequence may be different from the sequence functions are written in

Function types

- Library functions
  - Commonly required functions
  - Come with the compiler in the form of a library
  - Example: printf(), scanf()
- User-defined functions
  - Program-specific
  - Avoids redundancy (no need to repeat the same code again and again)
  - Divides the code in independent blocks
  - Ease of understanding and readability

A simple function

```
#include<stdio.h>
   void main (void)
     {
     line();
     printf("\t\t Hello");
     line();
     }

  void line()   //function header
  {
    printf("\n****************************\n");  //function body
  }
```

Function header

- Return-type function-name(arg1, arg2)
  - Return-type is the type of data a function returns. Default type is int
  - Function-name is a unique name given to the function
  - Arguments or parameters are enclosed in parenthesis and they are the data required by a function that are passed at the time of calling
  - Variables declared in one function are not available to other functions in a program

Function prototype

- Specification of a prototype enables the compiler to check for compile time errors such as number of parameters
- Syntax: Function header followed by a semicolon without the body (void line(void);)
- Prototype must be given before a call to the function is made
- For library functions, prototypes are defined in header files (stdio.h contains prototypes for printf() and scanf())

Function with no arguments & no return value

```
# include<stdio.h>
void line(void); //prototype
    void main (void)
      {
      line();
      printf("\t\t Hello");
      line();
      }

   void line()   //function header
   {
     printf("\n***************************\n");  //function body
   }
```

Function with arguments but no return value

```
#include<stdio.h>
void sum(int, int);
   void main (void) {
       int a,b;
       printf("enter first number: ");
       scanf("%d",&a);
       printf("enter second number: ");
       scanf("%d",&b);
       sum(a,b);
      }

  void sum(int a, int b) {
    printf("sum = %d",a+b);
  }
```

Function with arguments & return value

```
#include<stdio.h>
int fact(int);
void main (void) {
    int num, res;
    printf("enter a number: ");
    scanf("%d",&num);
    res=fact(num);
    printf("Factorial of %d is %d",num,res);
    }
    int fact(int n) {
        int a,fact=1;
        for(a=n;a>1;a--)
            fact*=a;
        return fact;
     }
```

Call by value

- All previous examples use call by value
- In call by value, the value of a variable is passed into the function
- Any operation performed on the value of that variable is not reflected in the calling block

Call by value check

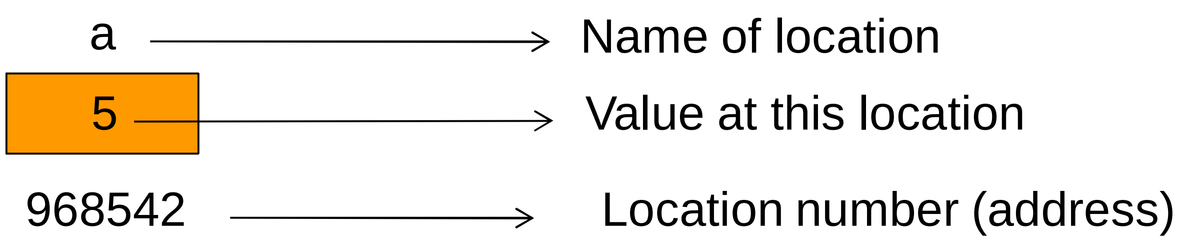
```
#include<stdio.h>
void sum(int, int);
 void main (void)
   {
   int a=1, b=1;
   sum(a , b);
   printf("a and b in main function are %d & %d \n", a , b);
   }

   void sum(int a, int b){
       a++; b++;
       printf("\na and b in sum function are %d & %d", a, b);
   }
```

Call by reference

- Instead of a value, the address of a variable is passed into the function
- Through the passed address, a function can directly operate on the original variable

Address versus content

- Example int a = 5;
- A location in memory is reserved which is identified by a



- printf(“%d”,a) will display 5
- printf(“%d”,&a) will display the address (968542)

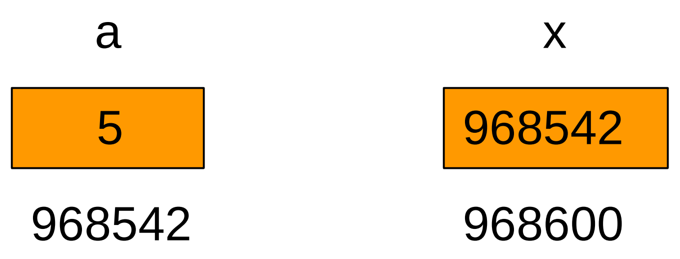
Pointers

- A pointer stores the address of a variable

  int a=5;

  int *x;

  x=&a;



- printf("%d",*a); displays 5
- See examples pointer.c and callbyreference.c

Returning more than one value

- A function can return only one value
- Call by reference enables to return more than one value from a function
- call by reference is made possible through using pointers
- To do so, addresses of variables are passed from the calling function and the called function directly modifies values at passed addresses
- See example retrnMultiVals.c

Recursion

- A function which makes a call to itself
- A process in which one of the instructions are to repeat the process
- Divides a complex problem into identical simple cases
- A recursive function must have at least one exit condition otherwise the function will continue calling itself repeatedly until the runtime stack overflows
- See Example recrus.c

Summary

- Functions are used to avoid repetitions and divide the code into separate blocks
- Function prototype and declaration specify the blueprint of a function
- Function definition specifies the body of the function
- Pointers hold addresses of other variables
- Pointers can be used to pass addresses of variables into a function and return more than one value
- Recursion is the calling of a function by itself