UNIT 5: FUNCTIONS

Dr. Nguyen Thi Thu Huong
Department of Computer Science
School of Information and Communication Technology
Hanoi University of Science and Technology

FUNCTIONS IN C

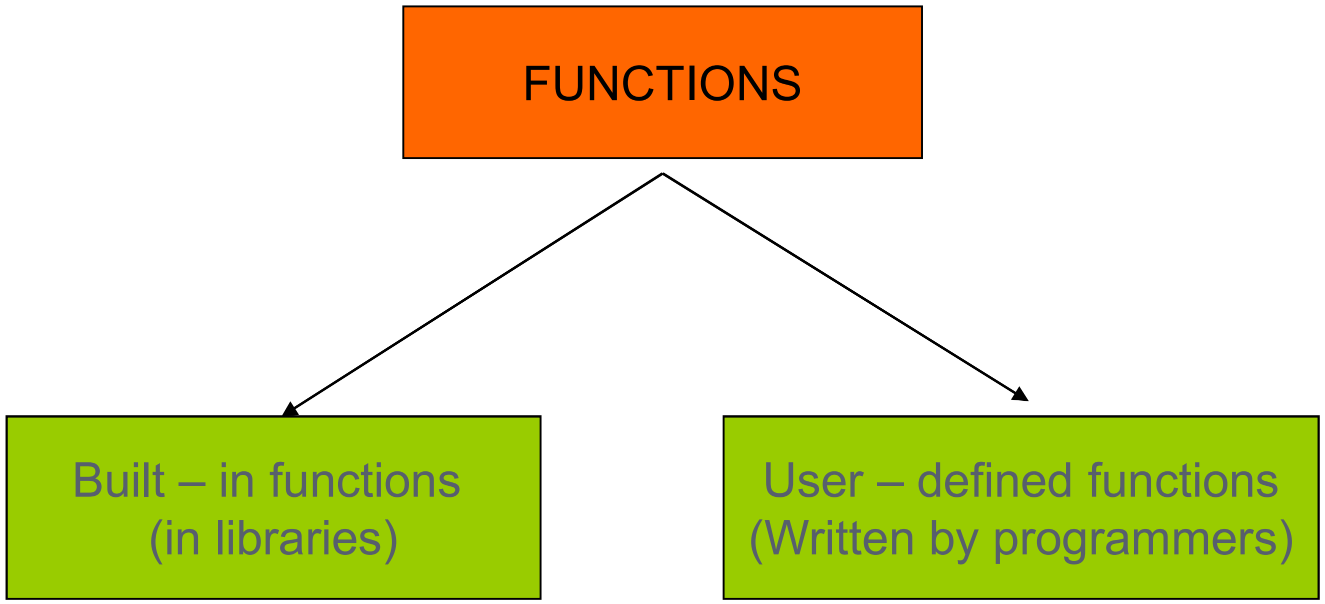
- Introduction
- Program Modules in C
- Functions
- Function Definitions
- Function Prototypes
- Calling Functions: Call by Value and Call by Reference
- Scope Rules

Introduction: Divide and conquer

- Construct a program from smaller pieces or components
- Avoid rewriting same logic/code again and again in a program.
- Each piece more manageable than the original program
- Improve understandability of very large C programs.

PROGRAM MODULES IN C

- Programs written by combining user-defined functions with library functions


FUNCTIONS
Built – in functions (in libraries)
User – defined functions (Written by programmers)

Where to write function definition?

- The function definition itself can act as an implicit function declaration.
- All identifiers in C need to be declared before they are used.That's why function definitions were put before main. If the order was reversed the compiler would not recognize the function.
- To correct this a prototype could be added before main.

FUNCTION DEFINITION BEFORE THE MAIN

```
#include<stdio.h>
int square(int x){
    int y;
    y =  x * x;
    return y;
}
main(){
    int i;
   for (i=0; i<= 10; i++)
   // function call
    printf("%d  ", square(i));
    getch();
}
```

FUNCTION DEFINITION WITH PROTOTYPE

```
#include<stdio.h>
// function prototype, also called function declaration
float square ( float x );
main( )
{   int i;
  for (i=0; i<= 10; i++)
// function call
printf("%d  ", square(i));
   getch();
}
 float square ( float x )   // function definition
{
   float y ;
  y = x * x ;
   return ( p ) ;
}
```

PROTOTYPE

- The prototype gives basic structural information:
  - what the function will return,
  - what the function will be called
  - what arguments the function can be passed

Function calls

- Invoking functions
  - Provide function name and arguments (data)
  - Function performs operations or manipulations
  - Function returns results

FUNCTION DEFINITION

```
return-value-type  function-name (parameter-list )
  {
    declarations and statements
  }
```

- Function-name: any valid identifier
- Return-value-type: data type of the result
  - **void** - function returns nothing
- Parameter-list: comma separated list, declares parameters (default **int**)

Function Definitions

- Declarations and statements: function body (block)
  - Variables can be declared inside blocks (can be nested)
  - Function can not be defined inside another function
- Returning control
  - If nothing returned
    - **return;**
    - or, until reaches right brace
  - If something returned
    - **return** *expression*;

FUNCTION PROTOTYPES

- Function prototype
  - Function name
  - Parameters - what the function takes in
  - Return type - data type function returns
  - Used to validate functions
  - Prototype only needed if function definition comes after use in program

    int maximum(int x, int y, int z);

    - Takes in 3 **ints**
    - Returns an **int**

Calling Functions: Call by Value and Call by Reference

- Used when invoking functions
- Call by value
  - Copy of argument passed to function
  - Changes in function do not effect original
  - Use when function does not need to modify argument
    - Avoids accidental changes
- Call by reference
  - Passes original argument
  - Changes in function effect original
  - Only used with trusted functions
- For now, we focus on call by value

Scope rules

- Scope rules tell us if an entity (*i.e.*, variable, parameter or function) is accessible at certain places.
- Places where an entity can be accessed is referred to the **scope** of that entity.

SCOPE RULES

- File scope
  - Identifier defined outside functions, known in all functions
  - Used for *global variables, function definitions, function prototypes*
- Function scope
  - Can only be referenced inside a function body

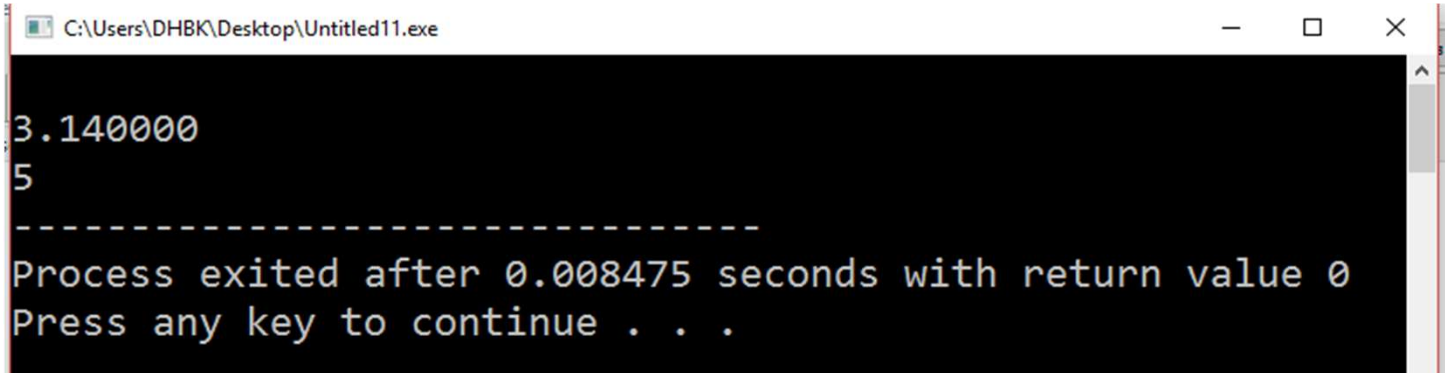
Scope Rules

- Block scope
  - Identifier declared inside a block
    - Block scope begins at definition, ends at right brace
  - Used for *variables, function parameters (local variables of function*)
  - Outer blocks "hidden" from inner blocks if there is a variable with the same name in the inner block

BLOCK SCOPE

- Execution blocks, delimited with {}, define scopes.

```
{       int t=5;
        {
        float t=3.14;
        printf("%f",t);
        }
        printf("%d",t);
}
```



```
3.140000
5
--------------------------------
Process exited after 0.008475 seconds with return value 0
Press any key to continue . . .
```

- The variable's scope is limited to the {} block that contains its declaration.