University of Engineering and Technology,
Peshawar, Pakistan

CSE 102: Computer Programming

Lecture 05

Functions

By;
Dr. Muhammad Athar Javed Sethi

What is a function?

- The function is one of the most basic things to understand in C++ programming.
- A function is a sub-unit of a program which performs a specific task.
- We have already (without knowing it) seen functions from the C++ library cout( ), cin( ) etc.
- We need to learn to write our own functions.

Types of Functions

- Built In Functions
- User Define Functions

Built In Functions

- Already been defined as part of the language.
- Can be used in any program.
- Built in functions are provided for general use.

User Define Functions

- These functions are created by users.
- These functions are written as part of program to perfome specific task.
- These functions are written for specific use.
- User define functions has three parts, i.e.
  - Function Declaration (prototype)
  - Function Definition
  - Function Usage/Calling

Declaring C++ Functions (Prototype)

- All functions should be declared at the top of the program (before they are used).

```
int factorial(int);
int max(int, int);
```

- The declaration tells the compiler how the function should be used, what it should take and what it should return. (following information provided to compiler)
  - Name of the function.
  - Type of data returned by the function.
  - The number and types of arguments or parameters used in the function.

Declaring C++ Functions (Prototype) *(cont)*

- Semi colon is used at the end of function declaration.
- Function declaration is similar to variable declaration.
- Rules for naming functions are same as those for naming variables.

Declaring C++ Functions (Prototype) *(cont)*

- Syntax;

type function_name (arguments);

- Examples;
  - void display (void);
  - int sum (int, int);
  - float temp (void);
  - void print (int,float,char);

Defining Functions

- Actual code of the function is called function definition.
- Set of instructions written to perfome a specific task.
- Function definition is always outside the main( ) function.
- It can be written before or after main function( ).
- It can also be written in a separate file.
  - Included in program using **#include** directive.

Defining Functions *(cont)*

- Function definition consists of two parts;
  - Declarator
  - Body of Function

Defining Functions *(cont)*

- Function definition consists of two parts;
  - Declarator
    - It is heading of the function definition.
    - Same as the function declaration but it is not terminated by the semicolon (;).
  - Body of Function
    - Set of statements enclosed in braces after the declarator are called the body of the function.

Defining Functions *(cont)*

- General format (syntax) of the function definition is;

```
type function_name ([list of parameters])
{
   set of instructions
}
```

Declarator

Body of Functions

Function Calling

- Executing the statements of a function to perform a specific task is called calling of the function.
- Function is called by referencing its name.
- The parameters of the function are given in parentheses after the name of the function.
- When a function is called the control shifts to the function definition and the statements (body of function) is executed.
- After executing the statements in the body of the function, the control returns to the calling function and the next statement that comes immediately after the function call statement is executed.

Passing Arguments to Functions

- If a function needs data to perform a specific task, this data is provided through arguments of the function.
- Arguments are placed in parentheses.
- Arguments are either constant or variables.
- They are written in same sequence in which they are defined in the function declaration.
- Data type of an argument in the function call must also match the corresponding data types in the function declaration.

Example OF Function

```
#include <iostream>
using namespace std;
void sum(int,int);
int main()
{
        sum(10,15);
        cout << “Ok”
        return 0;
}
```

```
void sum(int x, int y)
{
   int s;
   s=x+y;
   cout<< “sum=” << s << endl;
}
```

Above program simply adds two integer numbers. The number whose sum is to be calculated are passed to the function arguments

Returning Data From Functions

- Function can return only one value (except string and array).
- Type of data that a user defined function returns is declared in function declaration.
- By using “return” keyword function returns data or value to the calling function.
- Syntax;
  - return expression;

Passing Arrays as Arguments to a Function

- When array is passed to a function only starting address of the array is passed.
- C++ does not make a separate copy of the array (pass by reference).
- It only assigns the starting address of the same memory area of the array to the array name used in the declarator of the function.
- Function can change the contents of the array by directly accessing the memory cells.

Passing Arrays as Arguments to a Function *(cont)*

- Function Declaration (prototype) of a function with array Argument
  - void max(float[ ], int[ ]);
  - void temp(float[ ] [ ], int [ ][ ]);
- Function Definition with Array Arguments

```
void max(float stud[ ], int marks[ ])
{

Body of Functions

}
```

```
void temp(float faculty[ ] [ ],
int dept[ ][ ])
{

Body of Functions

}
```

Passing Arrays as Arguments to a Function *(cont)*

- Calling Function with Array Arguments
  - When a function is called uses array as argument, only the name of the array without subscript is passed.
  - Only the memory address of the array is passed to the function.
  - Data access is from the same memory location.

Review About Functions

- A function can take any number of arguments mixed in any way.
- A function can return at most one argument.
- We can declare variables within a function just like we can within `main()` – these variables will be deleted when we return from the function

Where Do Functions Come In The Program

- Generally speaking it doesn't matter too much.
- main() is a function just like any other (you could even call it from other functions if you wanted).
- It is common to make main() the first function in your code.
- Functions must be entirely separate from each other.
- Prototypes must come before functions are used.
- A usual order is:
  - Prototypes THEN main THEN other functions.

What Are These Prototype Things?

- A prototype tells your C++ program what to expect from a function – what arguments it takes (if any) and what it returns (if any)
- Prototypes should go before main()
- A function MUST return the variable type we say that it does in the prototype.

What is scope?

- The *scope* of a variable is where it can be used in a program.
- Normally variables are LOCAL in *scope* – this means they can only be used in the function where they are declared
  - main is a function.
  - User define function.
- We can also declare GLOBAL variables.
- If we declare a variable outside a function it can be used in any function beneath where it is declared.

Example: Print Stars

```
#include <iostream>
void print_stars(int);
using namespace std;

int main()
{
    int i;
    for (i= 0; i < 5; i++)
        print_stars(5);
    return 0;
}

void print_stars (int n)
{
    int i;
    for (i= 0; i < n; i++)
        cout << "*";
    cout<<"\n";
}
```

Variables here are LOCAL variables

Drawbacks of Global Variables

- Since they are accessible to all functions, it is difficult to track changes in their values.

- They occupy a large amount of memory permanently during program execution and the data accessing speed of program becomes slow.

Example: Draw Backs of Global Variable

```
#include <iostream>
void print_stars(int);
int i;   /* Declare global i */

int main()
{
    for (i= 0; i < 5; i++)
        print_stars(5);
    return 0;
}

void print_stars (int n)
{
    for (i= 0; i < n; i++)
        cout<< "*";
    cout<<"\n";
}
```

Summary

This program only prints ONE row of five stars, i..e.

*****

Static Variables

- Special Variables that are declared inside a function by using the key word “Static”.

- Types
  - Static Local Variables.
  - Static Global Variables.

- Like Local Variables, these can only be accessed in the function in which they are declared but the remain in existence for the life time of the program.

- Another difference between local variables and static local variables is that the initialization in the static variables take place only once when function is called for the first time.

Reference Parameters

- There are two ways to pass arguments to a function, These are;
  - Arguments passed by value.
  - Arguments passed by reference.

Reference Parameters

- Arguments passed by value.
  - When an argument is passed by value to a function, a new variable of the data type of the argument is created and data is copied into it.
  - The function access the value in the newly created variable and the data in the original variable in the calling function is not changed.

Reference Parameters

- Arguments passed by Reference
  - Data can also be passed to a function by reference of a variable name that contains data.
  - The reference provides the second name or alias for a variable name.
  - When a variable is passed by reference to a function, no new copy of the variable is created, only the address of the variable is passed to the function.
  - Any change in the reference variable also changes the value in the original variable.
  - The reference parameters are indicated by an ampersand**(&)** sign after the data type of both in function prototype and in the function definition.

Example: Reference Arguments

```
#include <iostream>
void exch (int &, int &)
main()
{
int a,b;
cout<<“Assign value to variable A?”;
cin>>a;
cout<<“Assign value to variable B?”;
cin>>b;
exch(a,b);
Cout<<“values after exchange”<<endl;
cout<<“Value of A=” << a<< endl;
cout<<“Value of B=”<<b<<endl;
}
void exch (int& x, int& y)
{
int t;
t=*x;
*x=*y;
*y=t;
}
```

Summary: Program exchanges the value of two variables by using reference arguments.

Function Overloading

- Declaring more than one function with the same name but with different set of arguments and return data types is called Function Overloading.
- Example
  - int sum(int, int); //prototype
  - int sum(int, int,int);   //prototype
  - double sum(double,double,double); //prototype