Programming Fundamentals- I

Rao Muhammad Umer
Lecturer,
Web: raoumer.github.io
Department of Computer Science & IT,
The University of Lahore.

Administrative Stuff

- Next week on **Thursday**, there will be a quiz on following topics:
- **Control structures (if, if-else, nested if-else, switch)**
- **Repetition structures (for, while, do-while)**
- **Arrays (1D, 2D)**

FUNCTIONS

// Addition of two integers without Function

```
#include<iostream>
int main()
{
int num1, num2, add_result;
cout << "Enter first number\n";
cin >> num1;
cout << "Enter second number\n";
cin >> num2;
add_result = num1 + num2;
cout << "Addition result is“ << add_result;
getchar();
return 0;
}
```

```
/* Addition of two integers using
    Function  */
#include <iostream>
int addnumbers(int x, int y);
using namespace std;
int main()
{
int num1, num2, add_result;
cout << "Enter first number\n";
cin >> num1;
cout << "Enter second number\n";
cin >> num2;
add_result= addnumbers(num1, num2);
cout << "Addition result is “ << add_result;
getchar();
return 0;
}
```

```
int addnumbers(int x, int y)
{
        int z;
        z = x +y;
        return z;
}
```

Functions

- The function is one of the most important things to understand in C++ programming.
- A function groups a number of program statements into a unit and gives it a name. This unit can then be **invoked** (called) from other parts of the program.
- A function is a **sub-unit** of a program which performs a specific task.
- Functions are **modules** in C++.
- Functions **take arguments**(constants values or variables) and may **return a value**.

Benefits of Functions

- Benefits of functions
  - **Divide and conquer**
    - Construct a program from smaller pieces or components
    - Modularize a program (These smaller pieces are called modules)
    - Each piece more manageable than the original program
    - Manageable program development
  - **Software reusability**
    - Use existing functions as building blocks for new programs
    - Abstraction - hide internal details (library functions)
  - **Avoid code repetition**

Benefits of Functions

- The most important reason to use functions is to aid in the conceptual organization of a program. **Dividing a program into functions is, one of the major principles of structured programming.**
- Another reason to use functions is **to reduce program size.**
- The function's code is **stored in only one place in memory,** even though the function is executed many times in the course of the program.

Functions

- A function **receives zero or more parameters**, performs a specific task, **and returns none or one value**.
- **A function is invoked by its name and parameters.**
  - No two functions have the same name in your C++ program.
  - The communication between the function and invoker is through the parameters and the return value.
- **A function is independent**:
  - It is “completely” self-contained.
  - It can be called at any places of your code.
- **Functions make programs reusable and readable.**

More Terminologies

Function Terminologies

- **Function prototype**
- **Function definition**
  - Function header
- **Calling a function**
- **Passing arguments**
- **Returning a value**

Function Prototype

```
/* Addition of two integers using
    Function  */
#include <iostream>
int addnumbers(int x, int y);
using namespace std;
int main()
{
int num1, num2, add_result;
cout << "Enter first number\n";
cin >> num1;
cout << "Enter second number\n";
cin >> num2;
add_result= addnumbers(num1, num2);
cout << "Addition result is “ << add_result;
getchar();
return 0;
}
```

```
int addnumbers(int x, int y)
{
        int z;
        z = x +y;
        return z;
}
```

Function Prototype

- A function prototype means **to declare the function** (as you declare a variable prior to use)
- 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.

Function Prototype

- Function prototype
  - Function name
  - Parameters – what the function takes in
  - Return type – data type function returns (default **`int`**)
  - Used to validate functions
  - **Prototype only needed if function definition comes after use in program**
  - The function with the prototype

```
int maximum( int, char, float );
```

    - Takes in 1 **`int`**, **`1 char`**, **`and 1 float`**
    - Returns an **`int`**

Function Definition

```
/* Addition of two integers using
    Function  */
#include <iostream>
int addnumbers(int x, int y);
using namespace std;
int main()
{
int num1, num2, add_result;
cout << "Enter first number\n";
cin >> num1;
cout << "Enter second number\n";
cin >> num2;
add_result= addnumbers(num1, num2);
cout << "Addition result is “ << add_result;
getchar();
return 0;
}
```

```
int addnumbers(int x, int y)
{
        int z;
        z = x +y;
        return z;
}
```

Function Header

```
/* Addition of two integers using
    Function  */
#include <iostream>
int addnumbers(int x, int y);
using namespace std;
int main()
{
int num1, num2, add_result;
cout << "Enter first number\n";
cin >> num1;
cout << "Enter second number\n";
cin >> num2;
add_result= addnumbers(num1, num2);
cout << "Addition result is “ << add_result;
getchar();
return 0;
}
```

```
int addnumbers(int x, int y)
{
        int z;
        z = x +y;
        return z;
}
```

/* The first line of the function definition (as shown in blue color above) is called Function Header. */

Function Definition

- Function definition format

  return-value-type function-name(parameter-list)

  {

  declarations and statements

  }

  - Function-name: any valid identifier
  - Return-value-type: data type of the result (default **`int`**)
    - **`void`** – indicates that the function returns nothing
  - Parameter-list: comma separated list, declares parameters
    - A type must be listed explicitly for each parameter unless, the parameter is of type **`int`**
    - The word int preceding the function name indicates that this particular function has a return value of type int.

Function Definition

- Function definition format (continued)

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

  - Declarations and statements: **function body (block)**
    - Variables can be declared inside blocks
    - Functions can not be defined inside other functions

Invoking a Function

- **Invoking a function means calling a function to use it.**
    - Provide function name and arguments (data)
    - Function performs operations or manipulations
    - Function returns results
- **Function call analogy:**
    - Boss asks worker to complete task
        - Worker gets information, does task, returns result
        - Information hiding: boss does not know details

Invoking a Function

- The function is *called (or invoked, or executed) to do its job*:

- Format for calling functions

```
VariableToStoreReturnValue = FunctionName( argument );
```

  - If multiple arguments, use comma-separated list

  – Arguments may be constants, variables, or expressions

Arguments

- **An *argument is the input to the function;*** *it is placed* ***inside the parentheses*** *following the function* name.
- The function then ***processes*** the argument and returns a value; this is the output from the function.
- ***An argument is a piece of data*** *(an int value, for example) passed from a program to the function.*
- Arguments allow a function to operate with different values.

Returning a Value

- To return a value from a C++ function you must explicitly return it with a return statement.
- Syntax:
- **return x; // value of x will be returned**
- **return <expression>; // value of the expression will be returned. e.g. return x+10**

The expression can be any valid C expression that resolves to the type defined in the function header.

Returning a Value

- Returning control
  - If nothing returned
    - **`return ;`**
    - **or, until reaches right brace**
  - If something returned
    - **`return`** *expression* **;**

Some Examples

- Function Prototype Examples

```
double squared (double number);
void print_report (int);
int get_menu_choice (void);
```

- Function Definition Examples

```
double squared (double number)
{
        return (number * number);
}
void print_report (int report_number)
{
        if (report_nmber == 1)
                cout << “Printer Report 1”;
        else
                 cout << “Not printing Report 1”;
}
```

Some Examples

- **int get_menu_choice (void);**

  void parameter list means it takes no parameters

- **void print_report (int);**

  return type void means it returns nothing

An example function

```
#include <iostream>
int maximum (int, int);
int main()
{
     int i= 4;
     int j= 5;
     int k;
     k= maximum (i,j);  /* Call maximum function */
     printf ("%d is the largest from %d and %d\n",k,i,j);
     getchar();
     return 0;
}

int maximum (int a, int b)
/* Return the largest integer */
{
     if (a > b)
          return a;  /* Return means "I am the result of the function"*/
     return b;       /* exit the function with this result */
}
```

Prototype the function

Call the function

function header

The function itself

Syntax

- Function Prototype:

```
return_type  function_name (type1  name1, type2
   name2, ..., typen  namen);
```

- Function Definition:

```
return_type function_name (type1 name1, type2 name2,
                                          ...,typen namen)
{
   ....statements...
}
```

Notes about functions

- A function can take any number of arguments **mixed** in any way.
- At this stage, we are dealing with a function that can return **at most one argument**.
- We can declare variables within a function just like we can within main()-these variables will **“reset”**, when we return from the function

Where do functions go in the program

- It is common to make main() the first function in your code.
- A usual order is: **Prototypes THEN main THEN other functions.**
- Prototypes must come before functions are used.

void functions

- A function doesn't necessarily have to take or return arguments. We prototype such a function using `void`.

```
void print_hello (void);
```

Prototype

Main Program Here

```
void print_hello (void)
// this function prints hello
{
   printf ("Hello\n");
}
```

Function takes and returns void (no arguments)

void functions

Another prototype

```
void odd_or_even (int);
```

```
void odd_or_even (int num)
/* this function prints odd or even
appropriately */
{
    if ((num % 2) == 0) {
        printf ("Even\n");
        return;
    }
    printf ("Odd\n");
}
```

Function which takes one
int arguments and returns none

void functions

Another prototype

```
void odd_or_even (int);

 main program here
 void odd_or_even (int num)
 /* this function prints odd or even
 appropriately */
 {
      if ((num % 2) == 0) {
           printf ("Even\n");
           return;
      }
      printf ("Odd\n");
 }
```

Function which takes one int arguments and returns non

Different Function Components in one Look

Component	*Purpose*	*Example*
Declaration (prototype)	Specifies function name, argument types, and return value. Alerts compiler (and programmer) that a function is coming up later.	`void func();`
Call	Causes the function to be executed.	`func();`

Definition	The function itself. Contains the lines of code that constitute the function.	`void func()`
		`{`
		`// lines of code`
		`}`

```
/* Addition of two integers using
    Function  */
#include <iostream>
int addnumbers(int x, int y);
using namespace std;
int main()
{
int num1, num2, add_result;
cout << "Enter first number\n";
cin >> num1;
cout << "Enter second number\n";
cin >> num2;
add_result= addnumbers(num1, num2);
cout << "Addition result is “ << add_result;
getchar();
return 0;
}
```

```
int addnumbers(int x, int y)
{
        int z;
        z = x +y;
        return z;
}
```

Eliminating the Prototype

```
/* Addition of two integers using
   Function  */
#include <iostream>
int addnumbers(int x, int y)
{
        int z;
        z = x +y;
        return z;
}

using namespace std;
int main()
{
int num1, num2, add_result;
cout << "Enter first number\n";
cin >> num1;
cout << "Enter second number\n";
cin >> num2;
add_result= addnumbers(num1, num2);
cout << "Addition result is “ << add_result;
getchar();
return 0;
}
```

Returning a value using if else statement

A single value is returned, however, multiple return statements can be used within a single function using **“if-then-else”** statement.

```
// Returning a value and if else
#include<iostream>
int myfunc (int x, int y);
using namespace std;
int main()
{
int num1, num2, smaller;
cout << "Enter first number\n";
cin >> num1;
cout << "Enter second
number\n";
cin >> num2;
smaller = myfunc (num1, num2);
cout << smaller << " is smaller\n";
getchar();
return 0;
}
```

```
int myfunc (int x, int y)
{
if(x < y )
    return x;
else
    return y;
}
```

```
// Calculate factorial without function
#include<iostream>
using namespace std;
int main()
{
int i, fact, num1;
cout << "Enter a positive integer\n";
cin >> num1;
fact = 1;
for (i= num1; i>=1; i--)
    {
        fact = fact * i;
    }
cout << "Factorial of” << num1 << “is” << fact << endl;
getchar();
return 0;
}
```

```
/* Calculate factorial using
Functions */
#include<iostream>
int cal_fac (int x);
int main()
{
int i, fact, num1;
cout << "Enter a positive
integre\n";
cin >> num1;
fact = cal_fac (num1);
cout << "Factorial of" << num1
<< "is" << fact;
getchar();
return 0;
}
```

```
int cal_fac (int x)
{
int i, fact;
fact = 1;
if (x == 0)
      fact = 1;
else
 {
    for (i= x; i>=1; i--)
       {
            fact = fact * i;
       }
  }
return fact;
}
```

```
/* Program that will pass temperature in
centigrade to a function that converts it
into Fahrenheit and returns the value to
the main function */
#include <iostream>
float ctof (float); //declaration
int main()
{
float centi, farhen;
cout << "Enter temperature in
centigrade\n";
cin >> centi;
farhen = ctof (centi);
cout << centi << “Centigrade=“ << fahren
<< “Faherenheit”;
getchar();
return 0;
}
```

```
/* converts centigrade to
Fahrenheit*/

float ctof (float cgrade)
{
float fgrade;
fgrade = (9/5) * cgrade + 32.0;
return fgrade;
}
```