Lecture (06)
Function

By:

Dr. Ahmed ElShafee

Introduction

- A function is a group of statements that together perform a task. Every C++ program has at least one function, which is **main()**, and all the most trivial programs can define additional functions
- You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division usually is so each function performs a specific task.
- A function **declaration** tells the compiler about a function's name, return type, and parameters. A function **definition** provides the actual body of the function.

Defining a Function:

-
```
return_type function_name( parameter list )
 }
body of the function
 {
```

- **Return Type**: A function may return a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword **void**.
- **Function Name:** This is the actual name of the function. The function name and the parameter list together constitute the function signature.


```
return_type function_name( parameter list )
 }
body of the function
 {
```


- **Parameters:** A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- **Function Body:** The function body contains a collection of statements that define what the function does.

•

```
int max(int num1, int num2)
 }
// local variable declaration
int result;
if (num1 > num2)
result = num1;
else
result = num2;
return result;
 {
```


C/C++ Built in functions

Char getchar(void)

Void printf(char *)

Int strcmp(char *, char*)

….

….

….

Function Declarations:

- A function **declaration** tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.
- A function declaration has the following parts:

```
return_type function_name( parameter list);
```

- For the previous example

```
int max(int num1, int num2);
```

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


Calling a Function:

- When a program calls a function, program control is transferred to the called function. A called function performs defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns program control back to the main program.
- To call a function, you simply need to pass the required parameters along with function name, and if function returns a value, then you can store returned value.

Example 01

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int max(int num1, int num2);
int _tmain(int argc, _TCHAR* argv[])
{
	int a,b;
	int ret;
	cout<<"Enter 1st value:";
	cin>>a;
	cout<<"Enter 2nd value:";
	cin>>b;
	// calling a function to get max value.
	ret = max(a, b);
	cout << "Max value is : " << ret << endl;
	cout<<"press any key to continue";
	fflush(stdin);
	cin.get();
	return 0;
}
```

```
int max(int num1, int num2)
{
// local variable declaration
	int result;
	if (num1 > num2)
	result = num1;
	else
	result = num2;
	return result;
}
```


Function Arguments:

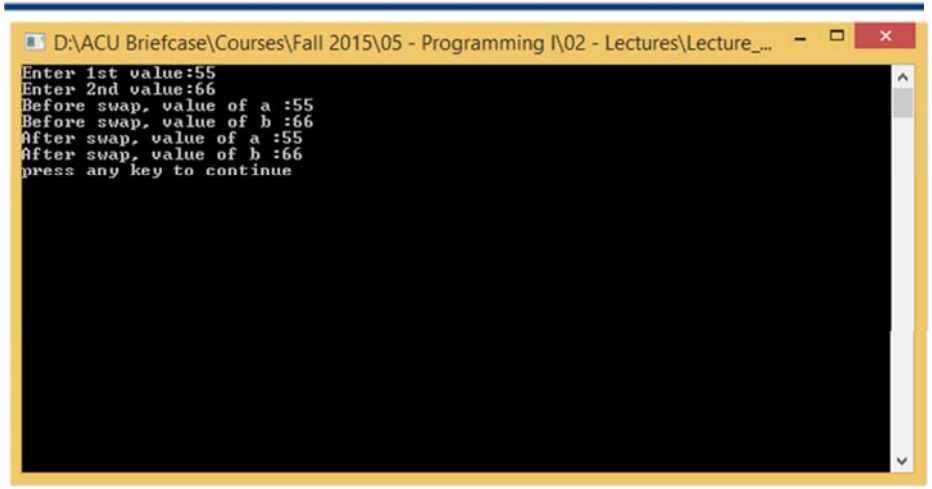
- While calling a function, there are two ways that arguments can be passed to a function:

Call type	Description
Call by value	This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument
Call be pointer	This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument

Example 02 : Call by value

```
#include "stdafx.h"
#include <iostream>
using namespace std;
void swap(int x, int y);
int _tmain(int argc, _TCHAR* argv[])
{
	int a,b;
	int ret;
	cout<<"Enter 1st value:";
	cin>>a;
	cout<<"Enter 2nd value:";
	cin>>b;
	cout << "Before swap, value of a :" << a << endl;
	cout << "Before swap, value of b :" << b << endl;
	swap(a, b);
	cout << "After swap, value of a :" << a << endl;
	cout << "After swap, value of b :" << b << endl;
	cout<<"press any key to continue";
	fflush(stdin);
	cin.get();
	return 0;
}
```

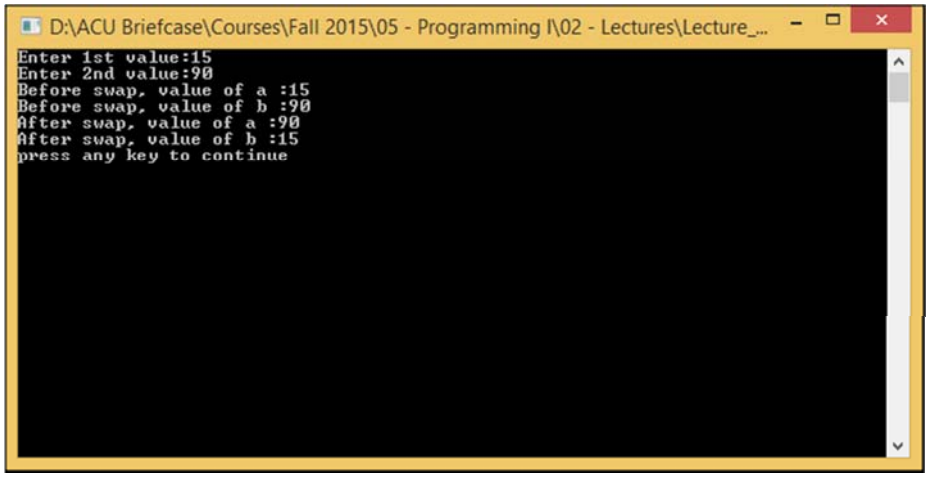
```
void swap(int x, int y)
{
	int temp;
	temp = x; /* save the value of x */
	x = y; /* put y into x */
	y = temp; /* put x into y */
	return;
}
```



Example 03 : Call by pointer

```
#include "stdafx.h"
#include <iostream>
using namespace std;
void swap(int *x, int *y);
int _tmain(int argc, _TCHAR* argv[])
{
	int a,b;
	int ret;
	cout<<"Enter 1st value:";
	cin>>a;
	cout<<"Enter 2nd value:";
	cin>>b;
	cout << "Before swap, value of a :" << a << endl;
	cout << "Before swap, value of b :" << b << endl;
	swap(&a, &b);
	cout << "After swap, value of a :" << a << endl;
	cout << "After swap, value of b :" << b << endl;
	cout<<"press any key to continue";
	fflush(stdin);
	cin.get();
	return 0;
}
```

```
void swap(int *x, int *y)
{
	int temp;
	temp = *x; /* save the value of x */
	*x = *y; /* put y into x */
	*y = temp; /* put x into y */
	return;
}
```

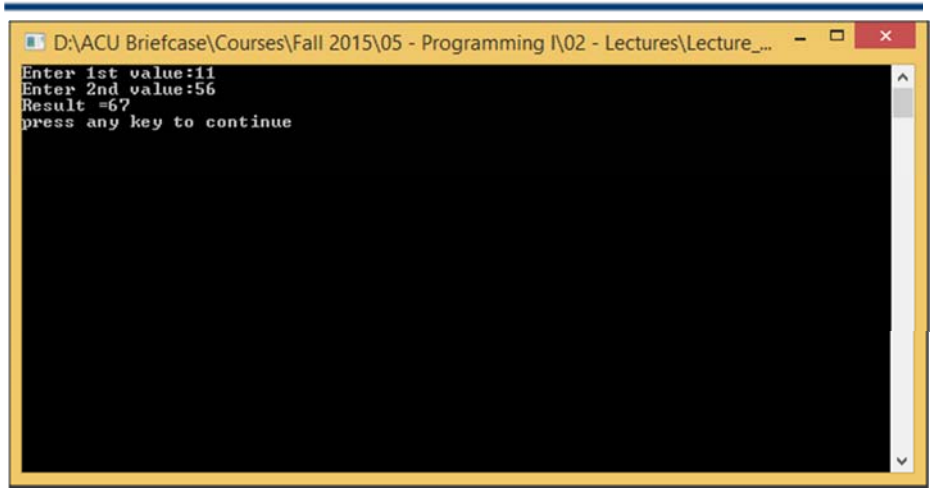



Default Values for Parameters:

- When you define a function, you can specify a default value for for each of the last parameters.
- This value will be used if the corresponding argument is left blank when calling to the function.

Example 05

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int sum(int x=0, int y=0);
int _tmain(int argc, _TCHAR* argv[])
{
    int a,b;
    int ret;
    cout<<"Enter 1st value:";
    cin>>a;
    cout<<"Enter 2nd value:";
    cin>>b;
    ret=sum();
    cout << "Result =" << ret << endl;
    cout<<"press any key to continue";
    fflush(stdin);
    cin.get();
    return 0;
}
int sum(int x, int y)
{
    int z=y+x;
    return y;
}
```

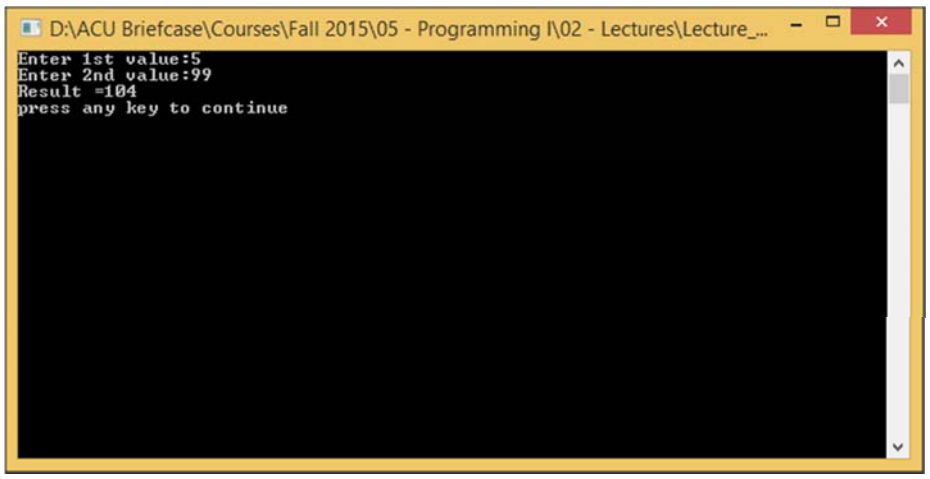



Local variables and global variables

- Local variable: Is a variable defined inside function/loop/condition/ or any type of blocks {},
- Local variable has a focus inside its block only, and is undefined outside its block.
- If you call local variable outside its block, program give compilation error
- You may define a variable having the same name out side a block.

Example 06

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int sum(int , int );
int _tmain(int argc, _TCHAR* argv[])
{
	int a,b;
	int ret;
	cout<<"Enter 1st value:";
	cin>>a;
	cout<<"Enter 2nd value:";
	cin>>b;
	ret=sum(a, b);
	cout << "Result =" << ret << endl;
	cout<<"press any key to continue";
	fflush(stdin);
	cin.get();
	return 0;
}
int sum(int a, int b)
{
	int ret=a+b;
	return ret;
}
```

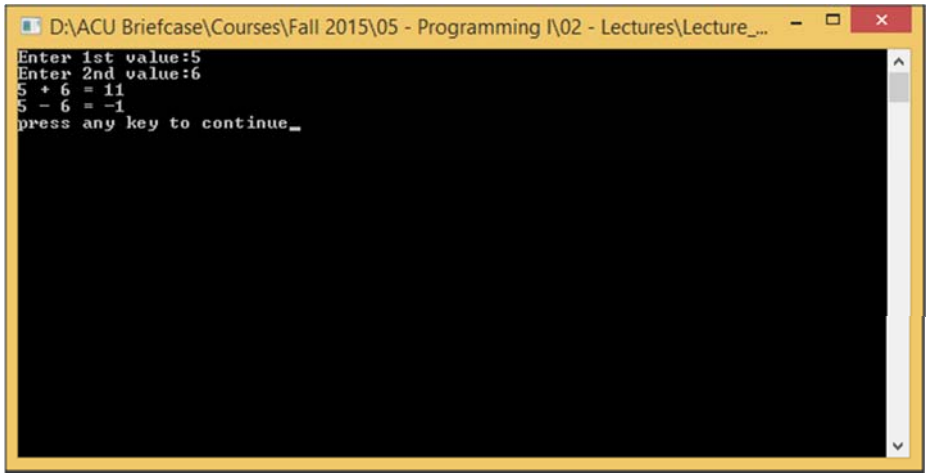


- Global variable is a variable define in outer block and can be accessed by any inner blocks

Example 07

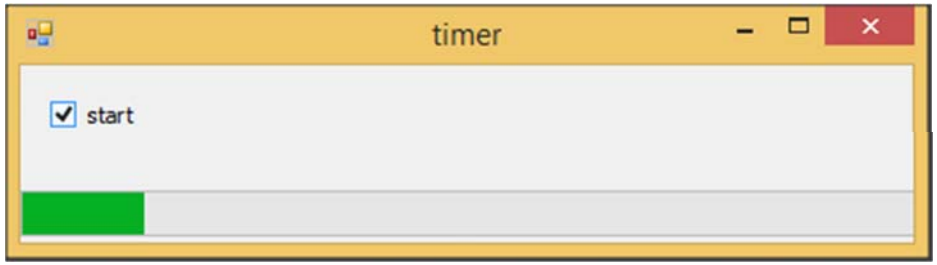
```
#include "stdafx.h"
#include <iostream>
using namespace std;
int a,b,ret;
int add(void);
int subtract(void);
int _tmain(int argc, _TCHAR* argv[])
{
	cout<<"Enter 1st value:";
	cin>>a;
	cout<<"Enter 2nd value:";
	cin>>b;
	ret=add();
	cout <<a<<" + "<<b <<" = "<< ret << endl;
	ret=subtract();
	cout <<a<<" - "<<b <<" = "<< ret << endl;
	cout<<"press any key to continue";
	fflush(stdin);
	cin.get();
	return 0;
}
```

```
int add(void)
{
	int ret=a+b;
	return ret;
}
int subtract(void)
{
	int ret=a-b;
	return ret;
}
```

Example 08

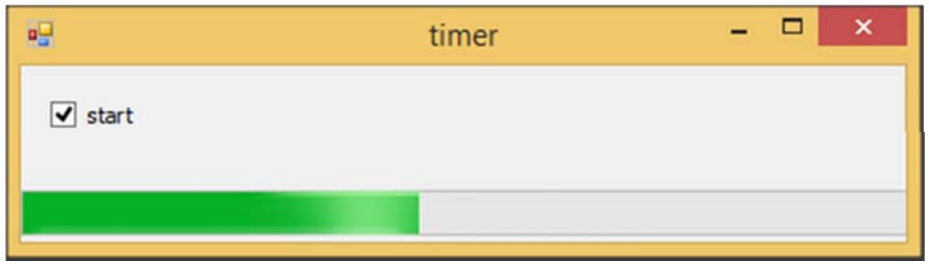
Count to 100 and restart




```
private: System::Void checkBox1_CheckedChanged(System::Object^  sender, System::EventArgs^  e) {
            if(checkBox1->Checked)
            {
                timer1->Enabled=true;
            }
            else
            {
                timer1->Enabled=false;
            }
        }
private: System::Void timer1_Tick(System::Object^  sender, System::EventArgs^  e) {
            progressBar1->Value=(progressBar1->Value+1)%100;
        }
};
```


Example 09

Count to 100 and reverse




```
private: System::Void checkBox1_CheckedChanged(System::Object^  sender, System::EventArgs^  e)
            if(checkBox1->Checked)
            {
                timer1->Enabled=true;
            }
            else
            {
                 timer1->Enabled=false;
            }
        }
private: System::Void timer1_Tick(System::Object^  sender, System::EventArgs^  e) {
            if(direction==0)
            {
                progressBar1->Value=(progressBar1->Value+1);
                 if(  progressBar1->Value==100) direction=1;
            }
            else if(direction ==1)
            {
                 progressBar1->Value=(progressBar1->Value-1);
                   if(  progressBar1->Value==0) direction=0;
            }

        }
```

Static Variable

- Adding modifier <static> while declaring a variable make it independent on its block.
- Static variable declared be a block but is not destroyed after ending the block.
- It remains in program memory.

Example 10

```
#include "stdafx.h"
#include <iostream>
using namespace std;
double cashInHand(double cash);
int _tmain(int argc, _TCHAR* argv[])
{
	double cash;
	while(1)
	{
		cout<<"Enter the current cash :";
		cin>>cash;
		cout<<"the current cash = "<<cash<<endl;
		if(cash==0) break;
		cout<<"the total cash in hand = "<<cashInHand(cash)<<endl;
		cout<<"****************************************"<<endl;
	}
	cout<<"the total cash in hand = "<<cashInHand(cash)<<endl;
	cout<<"press any key to exit,...";
	fflush(stdin);
	cin.get();
	return 0;
}

double cashInHand(double cash)
{
	static double cashInHand=0;
	cashInHand=cashInHand+cash;
	return cashInHand;
}
```