Programming Fundamentals (COMP1112)

FUNCTIONS

Function

- A function is named block of code that performs some action. A C++ program has at least one function, which is **main()**.
- 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 such that each function performs a specific task.

Function Declaration

- 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);

- Following are the function declarations −

int max(int num1, int num2); or

int max(int, int);

Function Definition

The general form of a C++ function definition is as follows −

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

Function Definition (cont..)

- A C++ function definition consists of a function header and a function body. Here are all the parts of a 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.
- **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.

Example

Following is the source code for a function called **max()**. This function takes two parameters num1 and num2 and return the biggest of both.

```
// function returning the max between two numbers
int max(int num1, int num2)
{   // local variable declaration
int result;
if (num1 > num2)
result = num1;
else      result = num2;
return result;
}
```

Function Call

- While creating a C++ function, you give a definition of what the function has to do. To use a function, you will have to call or invoke that function.
- When a program calls a function, program control is transferred to the called function. A called function performs defined task and when it’s 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. For example −

```
#include <iostream>
using namespace std;

// function declaration
int max(int num1, int num2);

int main ()
 {
// local variable declaration:
int a = 100;
int b = 200;
int ret;

// calling a function to get max value.
ret = max(a, b);

cout << "Max value is : " << ret << endl;
return 0;
}
```

```
// function returning the max between two
//numbers
int max(int num1, int num2)
{
// local variable declaration
int result;

if (num1 > num2)
result = num1;
else
result = num2;

return result;
}
```

Formal parameters vs Actual Parameters

- If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the **formal parameters** of the function.
- The formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.

```
/*C++ Program that defines a function which performs
arithmetic operation on two numbers on the basis of
the operator and operand entered by user and passed
to function as parameters.*/

void mathop (int, int, char);
int main()
{
 int a,b;
 char op;
      cout<<"Enter First Number:";
      cin>>a;
      cout<<"Enter Second Number:";
      cin>>b;
      cout<<"Enter Any Arithmetic Operator: ";
      cin>>op;
      mathop(a,b,op);
 return 0;
 }
```

```
void mathop(int a, int b, char op)
{
 switch (op)
  {
 case '+':
     cout<<a << op << b <<"+" << a+b;
     break;
 case '-':
    cout<<a << op << b <<"-" << a-b;
    break;
case '*':
    cout<<a << op << b<<"*" << a+b;
    break;
 case '/':
    cout<<a << op << b<<"/" << a/b;
    break;
 default:
    cout<<"Wrong Arithmetic Operator Entered";
    break;
  }
}
```

Function call types

- Call by value: 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. By default, C++ uses call by value to pass arguments
- Call by reference: copies the reference of an argument into the formal parameter. Inside the function, the reference is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

```
/*C++ program to demonstrate call by value vs call by
reference.*/

#include <iostream>
using namespace std;

void func(int &x, int y);

void func(int &x, int y) {
  x=x+1;
  y=y+1;
  cout<<"value of x is "<<x<<endl;
  cout<<"value of y is "<<y<<endl;
  return;
}
```

```
int main () {

  int a= 100;
  int b = 200;
  cout << "Before func call, value of a is " << a << endl;
  cout << "Before func call, value of b is " << b << endl;
  func(a, b);
  cout << "After func call, value of a is " << a << endl;
  cout << "After func call, value of b is " << b << endl;

  return 0;
}
```

Output:

```
Before func call, value of a is 100
Before func call, value of b is 200
value of x is 101
value of y is 201
After func call, value of a is 101
After func call, value of b is 200
```

Default values for parameters

- When you define a function, you can specify a default value for each of the last parameters. This value will be used if the corresponding argument is left blank when calling to the function.
- This is done by using the assignment operator and assigning values for the arguments in the function definition. If a value for that parameter is not passed when the function is called, the default given value is used, but if a value is specified, this default value is ignored and the passed value is used instead.

Default Parameters’ initialization

- The parameters initialized during function declaration are called default parameter's initialization.
- The default parameters allow user to call a function without giving required parameters.
- Example:

```
void Test ( int n );
int main()
{
Test();
Test (2);
Test (88);
return 0;
}
void Test(int n)
{
cout<<“n is  “<<n<<endl;
}
```

```
#include <iostream>
using namespace std;

int sum(int a, int b = 20) {
   int result;
   result = a + b;
   return (result);
}
```

```
int main () {
   // local variable declaration:
   int a = 100;
   int b = 200;
   int result;
    // calling a function to add the values.
   result = sum(a, b);
   cout << "Total value is :" << result << endl;
    // calling a function again as follows.
   result = sum(a);
   cout << "Total value is :" << result << endl;
    return 0;
}
```

Function overloading

- Function overloading (also *method overloading*) is a programming concept that allows programmers to define two or more functions with the same name and in the same scope.
- Each function has a unique signature (or header), which is derived from:
  - number of arguments
  - arguments' type
  - arguments' order

Passing array to function

- You can pass array as an argument to a function just like you pass variables as arguments. In order to pass array to the function you just need to mention the array name during function call.
- The array name itself is the address of first element of that array. For example if array name is arr then you can say that arr is equivalent to the &arr[0].
- Size of array does not pass with array name.

Passing array to function

```
/*C++ program that declares and initializes an
array of integer of size 5, pass it to function
which calculates the average and displays the
result in main*/

#include <iostream>
using namespace std;
double getAverage(int arr[], int size);
int main ()
{
int balance[5] = {1000, 2, 3, 17, 50};
double avg;
avg = getAverage( balance, 5 ) ;
cout << "Average value is: " << avg << endl;
return 0;
}
```

```
double getAverage(int arr[], int size)
{ //1
int i, sum = 0;
double avg;

for (i = 0; i < size; ++i)
  {
    sum += arr[i];
  }
avg = (double) sum/ size;
return avg;
}
```

Types of variables

- Global variable
  - Program level scope
  - Life is of program level
- Local variable
  - Scope is within the area in which declared
  - Life time starts when created
- Register variable
  - Use keyword “register”
  - Stored in register instead of RAM
  - register int a;
- Static variable
  - Initializes once at first execution
  - Life time is like of global variable
  - Scope is of local variable

Static variables

```
void func(){
static int n=0;
n++;
cout<<"value of n is "<<n<<endl;

}
int main(){
int i=0;
for(i=1;i<=5;i++)
func();
return 0;
}
```

Recursion

- A programming technique in which a function calls itself.
- Example: A program to calculate factorial of a given number

```
int fact( int n)
{
   if (n==0)
   return 1;
   else
   return n* fact(n-1);
}
int main(){
    int num;
    cout<<"Enter an integer:";
    cin>>num;
    cout<<"Factorial of "<< num<< "is "<<fact(num);
    return 0;
}
```

Memory allocation in recursion

When any function is called from main(), the memory is allocated to it on the stack. A recursive function calls itself, the memory for a called function is allocated on top of memory allocated to calling function and different copy of local variables is created for each function call. When the base case is reached, the function returns its value to the function by whom it is called and memory is de-allocated and the process continues.

References

- C++ How to Program
  By Deitel & Deitel
- The C++ Programming Language
  By Bjarne Stroustrup
- Object oriented programming using C++ by Tasleem Mustafa, Imran Saeed, Tariq Mehmood, Ahsan Raza
- https://www.tutorialspoint.com/cplusplus
- http://ecomputernotes.com/cpp/introduction-to-oop
- http://www.cplusplus.com/doc/tutorial
- https://www.guru99.com/c-loop-statement.html
- https://www.tutorialspoint.com/cplusplus/cpp_functions.htm