Department of Networks

First Year

Problem Solving and Programming 1

Functions in C++

- A function is a group of statements that perform a particular task.
- You may define your own functions in C++.
- Using functions can have many advantages, including the following:
    - You can reuse the code within a function.
    - You can easily test individual functions.
    - If it's necessary to make any code modifications, you can make modifications within a single function, without altering the program structure.
    - You can use the same function for different inputs.
- Every valid C++ program has at least one function - the main() function.

The Return Type

- The main function takes the following general form:

```
int main()
{
      // some code
  return 0;
}
```

- A function's return type is declared before its name.
- In the example above, the return type is **`int`**, which indicates that the function returns an integer value.
- Occasionally, a function will perform the desired operations without returning a value.
- Such functions are defined with the keyword **`void`**.
- **`void`** is a basic data type that defines a valueless state.

Defining a Function in C++

- Define a C++ function using the following syntax:

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

- **`return_type`**: Data type of the value returned by the function.
- **`function_name`**: Name of the function.
- **`parameter`**: 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.
- **`body of the function`**: A collection of statements defining what the function does.
- Parameters are optional; that is, you can have a function with no parameters.
- As an example, let's define a function that does not return a value, and just prints a line of text to the screen.

```
void printSomething()
{
    cout << "Hi there!";
}
```

- Our function, entitled **`printSomething(),`** returns void, and has no parameters.

- Now, we can use our function in **main**().

```
int main()
{
   printSomething();
   return 0;
}
```

- To call a function, you simply need to pass the required parameters along with the function name.
- You must declare a function prior to calling it.
- Example:

```
#include <iostream>
using namespace std;
void printSomething() {
    cout << "Hi there!";
}
int main() {
   printSomething();
return 0;  }
```

- Putting the declaration after the **main**() function results in an error.
- A function declaration, or function prototype, tells the compiler about a function name and how to call the function.
- The actual body of the function can be defined separately.
- Function declaration is required when you define a function in one source file and you call that function in another file.
- In such case, you should declare the function at the top of the file calling the function.

```
#include <iostream>
using namespace std;
//Function declaration
void printSomething();
int main() {
   printSomething();
   return 0;
}
```

```
//Function definition
void printSomething() {
   cout << "Hi there!";
}
```

Function Parameters

- For a function to use arguments, it must declare formal parameters, which are variables that accept the argument's values.
- Argument: a piece of data that is passed into a function or a program.
- Example:

```
void printSomething(int x)
{
    cout << x;
}
```

- This defines a function that takes one integer parameter and prints its value.
- Formal parameters behave within the function similarly to other local variables.

- They are created upon entering the function and are destroyed upon exiting the function.
- Once parameters have been defined, you can pass the corresponding arguments when the function is called.
- **Example**

```
#include <iostream>
using namespace std;
void printSomething(int x) {
   cout << x;
}
int main() {
   printSomething(2025);
}
```

// Outputs 2025

- The value 2025 is passed to the function as an argument and is assigned to the formal parameter of the function: x.
- Making changes to the parameter within the function does not alter the argument.
- You can pass different arguments to the same function.

- For example:

```
int timesTwo(int x) {
return x*2;
}
```

- The function defined above takes one integer parameter and returns its value, multiplied by 2.
- We can now use that function with different arguments.

```
int main() {
cout <<timesTwo(8); // Outputs 16
cout <<timesTwo(5);  // Outputs 10
cout <<timesTwo(50); // Outputs 100
}
```

Functions with Multiple Parameters

- You can define as many parameters as you want for your functions, by separating them with commas.
- Let's create a simple function that returns the sum of two parameters.

```
int addNumbers(int x, int y) {
    // code goes here }
```

- As defined, the **addNumbers** function takes two parameters of type **int**, and returns **int**.
- **Data type** and **name** should be defined for each parameter.
- Now let's calculate the sum of the two parameters and return the result:

```
int addNumbers(int x, int y) {
    int result = x + y;
  return result;
}
```

- Now we can call the function.

```
int addNumbers(int x, int y) {
   int result = x + y;
return result;
}
int main() {
   cout << addNumbers(50, 25); // Outputs 75
}
```

- You can also assign the returned value to a variable.

```
int main() {
```

```
int x = addNumbers(35, 5);
cout << x;
// Outputs 40
}
```

- You can add as many parameters to a single function as you want.
- **Example**:

```
int addNumbers(int x, int y, int z, int a)
{
    int result = x + y + z + a;
return result;
}
```

- If you have multiple parameters, remember to separate them with commas, both when declaring them and when passing the arguments.
- **Example:**

```
float circle_area(int radius)
{
   float result = radius*radius*3.14;
```

```
return result;
}
int main()
{
   int R;
   cin>>R;
   cout<<"area = "<<circle_area(R);
}
float sphere_volume(int radius)
{
float res = (float)(4/3)*radius*radius*radius*3.14;
return res;
}
int main()
{
   int R;
   cin>>R;
   cout<<"Volume = "<<sphere_volume (R);
}
```

HW on Functions:

Write C++ programs for each of the following:

1- Write a C++ function called **Power**() that receives two integer numbers X and Y and returns X^Y. **Example:** Power(2,3), the result will be 8.

2- Write a C++ Function called **Factorial**() that calculates the factorial of any integer number. Factorial of any integer number X = 1*2*3*…X: **Example:** Factorial(4), the result will be 24.

3- Write a C++ function called **cylinder_area**() to calculate the area of any cylinder. The function receives the Height ***h*** and the Radius ***r*** of the cylinder and returns the area of the cylinder: Area of cylinder = $2\pi rh+2\pi r^2$.

4- Write a C++ function called **box_volume**() to calculate the **volume** of any box. The function receives the length ***L***, the height *H* and width *W* of the box and calculates the volume according to this formula: Volume = *L*H*W* and returns the result.

5- Write a C++ function called **print_fators**() that receives an integer number prints the **factors** of that number.

6- Write a C++ function called **ToCelsius**() that converts temperature from **Fahrenheit** to **Celsius**. Note: *C*=(*F*-32)*(5/9). The function should receive a float value and return a float result.

7- Write a C++ function called **ToUpper**() that converts a small letter to a capital letter.

8- Write a C++ function called **ToDecimal**() that receives a binary number and returns the corresponding decimal number.

9- Write a C++ function called **month_name**() that receives an integer number that represents the month number and returns the month name.

10- Write a C++ function called **Is_triangle_valid**() that receives 3 angles of a triangle and checks if the triangle valid or not. Note: a triangle is a valid triangle when the total of its angles equals 180. The function should return *true* if the triangle is valid and return *false* otherwise.