LAB 12: Functions

Objective(s):
To Understand about:

Apply user defined functions in C++

1. Default arguments.
2. Using reference variable as parameters
3. Overloading Functions

CLOs: CLO1, CLO4

Default Arguments:
Default arguments are passed to parameters automatically if no argument is provided in the function call.
It's possible to assign default arguments to function parameters. A default argument is passed to the parameter when the actual argument is left out of the function call. The default arguments are usually listed in the function prototype. Here is an example:

```
void showArea(double = 20.0, double = 10.0);
```

Default arguments are literal values or constants with an = operator in front of them, appearing after the data types listed in a function prototype. Since parameter names are optional in function prototypes, the example prototype could also be declared as

```
void showArea(double length = 20.0, double width = 10.0);
```

It's possible to assign default arguments to function parameters. A default argument is passed to the parameter when the actual argument is left out of the function call. The default arguments are usually listed in the function prototype. Here is an example:
void showArea(double = 20.0, double = 10.0);
Default arguments are literal values or constants with an = operator in front of them, appearing after the data types listed in a function prototype. Since parameter names are optional in function prototypes, the example prototype could also be declared as
void showArea(double length = 20.0, double width = 10.0);
In both example prototypes, the function showArea has two double parameters. The first is assigned the default argument 20.0 and the second is assigned the default argument 10.0. Here is the definition of the function:

```
void showArea(double length, double width)
{
double area = length * width;
cout << "The area is " << area << endl;
}
```

The default argument for length is 20.0 and the default argument for width is 10.0. Because both parameters have default arguments, they may optionally be omitted in the

function call, as shown here:
showArea();
In this function call, both default arguments will be passed to the parameters. The parameter
length will take the value 20.0 and width will take the value 10.0. The output of the function will be
The area is 200
The default arguments are only used when the actual arguments are omitted from the function
call. In the call below, the first argument is specified, but the second is omitted:
showArea(12.0);
The value 12.0 will be passed to length, while the default value 10.0 will be passed to width. The output of the function will be
The area is 120
Of course, all the default arguments may be overridden. In the function call below, arguments are supplied for both parameters:
showArea(12.0, 5.5);
The output of the function call above will be
The area is 66

Example: Default Argument

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

// defining the default arguments
void display(char = '*', int = 3);

int main() {
   int count = 5;

   cout << "No argument passed: ";
   // *, 3 will be parameters
   display();

   cout << "First argument passed: ";
    // #, 3 will be parameters
   display('#');

   cout << "Both arguments passed: ";
   // $, 5 will be parameters
   display('$', count);

   return 0;
}

void display(char c, int count) {
    for(int i = 1; i <= count; ++i)
    {
        cout << c;
    }
```

Output:

```
No argument passed: ***
First argument passed: ###
Both arguments passed: $$$$$
```

Things to Remember:

Once we provide a default value for a parameter, all subsequent parameters must also have default values. For example,

```
// Invalid
void add(int a, int b = 3, int c, int d);

// Invalid
void add(int a, int b = 3, int c, int d = 4);

// Valid
void add(int a, int c, int b = 3, int d = 4);
```

If we are defining the default arguments in the function definition instead of the function prototype, then the function must be defined before the function call.

```
// Invalid code

int main() {
    // function call
    display();
}

void display(char c = '*', int count = 5) {
    // code
}
```

In C++ we can pass arguments into a function in different ways. These different ways are

- Call by Value
- Call by Reference
- Call by Address

Sometimes the call by address is referred to as call by reference, but they are different in C++. In call by address, we use pointer variables to send the exact memory address, but in call by reference we pass the reference variable (alias of that variable). This feature is not present in C, there we have to pass the pointer to get that effect. In this section we will see what are the advantages of call by reference over call by value, and where to use them

Call by Value:
In call by value, the actual value that is passed as argument is not changed after performing some operation on it. When call by value is used, it creates a copy of that variable into the stack section in memory. When the value is changed, it changes the value of that copy, the actual value remains the same.

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

void my_function(int x) {
  x = 50;
  cout << "Value of x from my_function: " << x << endl;
}

main() {
  int x = 10;
  my_function(x);
  cout << "Value of x from main function: " << x;
}
```
Output:
```
Value of x from my_function: 50
Value of x from main function: 10
```

Call by Reference
In call by reference the actual value that is passed as argument is changed after performing some operation on it. When call by reference is used, it creates a copy of the reference of that variable into the stack section in memory. It uses a reference to get the value. So, when the value is changed using the reference it changes the value of the actual variable.

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

void my_function(int &x) {
  x = 50;
  cout << "Value of x from my_function: " << x << endl;
}

main() {
  int x = 10;
  my_function(x);
  cout << "Value of x from main function: " << x;
}
```

Output:

```
Value of x from my_function: 50
Value of x from main function: 50
```

Where to use Call by reference?

The call by reference is mainly used when we want to change the value of the passed argument into the invoker function.

One function can return only one value. When we need more than one value from a function, we can pass them as an output argument in this manner.

Function Overloading:

With function overloading, multiple functions can have the same name with different parameters:

Example:

```
int myFunction(int x)
float myFunction(float x)
double myFunction(double x, double y)
```

Consider the following example, which have two functions that add numbers of different type:

Example:

```
int plusFuncInt(int x, int y) {
  return x + y;
}

double plusFuncDouble(double x, double y) {
  return x + y;
}

int main() {
  int myNum1 = plusFuncInt(8, 5);
  double myNum2 = plusFuncDouble(4.3, 6.26);
  cout << "Int: " << myNum1 << "\n";
  cout << "Double: " << myNum2;
  return 0;
}
```

Instead of defining two functions that should do the same thing, it is better to overload one. In the example below, we overload the plusFunc function to work for both int and double:

Example

```
int plusFunc(int x, int y) {
  return x + y;
}

double plusFunc(double x, double y) {
  return x + y;
}
int main() {
  int myNum1 = plusFunc(8, 5);
  double myNum2 = plusFunc(4.3, 6.26);
  cout << "Int: " << myNum1 << "\n";
  cout << "Double: " << myNum2;
  return 0;
}
```

Note: Multiple functions can have the same name as long as the number and/or type of parameters are different.

Lab Tasks:

Task 1:

Write a program that computes and displays the charges for a patient's hospital stay. First, the program should ask if the patient was admitted as an in-patient or an outpatient. If the patient was an in-patient, the following data should be entered:

- The number of days spent in the hospital
- The daily rate
- Hospital medication charges
- Charges for hospital services (lab tests, etc.)

The program should ask for the following data if the patient was an out-patient:

- Charges for hospital services (lab tests, etc.)
- Hospital medication charges

The program should use two overloaded functions to calculate the total charges. One of the functions should accept arguments for the in-patient data, while the other function accepts arguments for out-patient information. Both functions should return the total charges.
Input Validation: Do not accept negative numbers for any data.

Task2: Population

In a population, the birth rate is the percentage increase of the population due to births, and the death rate is the percentage decrease of the population due to deaths. Write a program that displays the size of a population for any number of years. The program should ask for the following data:

- The starting size of a population
- The annual birth rate
- The annual death rate
- The number of years to display

Write a function that calculates the size of the population for a year. The formula is
$N = P + BP - DP$
where N is the new population size, P is the previous population size, B is the birth rate, and D is the death rate.

Task 3: Paint Job Estimator

A painting company has determined that for every 110 square feet of wall space, one gallon of paint and eight hours of labor will be required. The company charges $25.00 per hour for labor. Write a modular program that allows the user to enter the number of rooms that are to be painted and the price of the paint per gallon. It should also ask for the square feet of wall space in each room. It should then display the following data:

- The number of gallons of paint required
- The hours of labor required
- The cost of the paint
- The labor charges
- The total cost of the paint job

Input validation: Do not accept a value less than 1 for the number of rooms. Do not accept a value less than $10.00 for the price of paint. Do not accept a negative value for square footage of wall space.

Task 4:

Write function which accept an array as argument and increment array's elements by 5, then write a function to show array elements

Note: input array in main function