CC-211L

Object Oriented Programming

Laboratory 03

Introduction to C++ (III):
(Class Templates array /Stack/ Vectors)

Version: 1.0.0

Release Date: 26-01-2023

Department of Information Technology
University of the Punjab
Lahore, Pakistan

Contents:

Learning Objectives:

- Function Overloading
- C++ Arrays
- Initializing and declaring Arrays
- Multidimensional arrays
- C++ Vector class

Resources Required:

- Desktop Computer or Laptop
- Microsoft ® Visual Studio 2022

General Instructions:

- In this Lab, you are **NOT** allowed to discuss your solution with your colleagues, even not allowed to ask how is s/he doing, this may result in negative marking. You can **ONLY** discuss with your Teaching Assistants (TAs) or Lab Instructor.
- Your TAs will be available in the Lab for your help. Alternatively, you can send your queries via email to one of the followings.

Teachers:		
Course Instructor	Prof. Dr. Syed Waqar ul Qounain	swjaffry@pucit.edu.pk
Teacher Assistants	Syed M.Jafar Ali Naqvi	bitf21m032@pucit.edu.pk
	M. Usman Munir	bitf19m020@pucit.edu.pk
	Aqdas Shabir	bitf21m022@pucit.edu.pk
	Ibad Nazir	bitf21m024@pucit.edu.pk
	Umer Farooq	bitf21m010@pucit.edu.pk

Background and Overview:

Overview of Object-oriented programming:

Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data. In OOP, we have learned the following concepts:

- A **Class** is a user-defined data type which has its own data members and member functions.
- An **Object** is an identifiable entity with some characteristics and behavior. An Object is said to be an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (an object is created), memory is allocated.
- **Data members** are the data variables and **member functions** are the functions used to manipulate these variables and together these data members and member functions define the properties and behavior of the objects in a Class.
- **Encapsulation** is defined as wrapping up of data and information under a single unit. In Object-Oriented Programming, Encapsulation is defined as binding together the data and the functions that manipulate them.
- **Abstraction** means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.
- We can implement Abstraction in C++ using classes. The class helps us to group data members and member functions using available **access specifiers**. A Class can decide which data member will be visible to the outside world and which is not.

Object-oriented programming has several advantages over procedural programming:

- OOP is faster and easier to execute.
- OOP provides a clear structure for the programs.
- OOP helps to keep the C++ code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug.
- OOP makes it possible to create full reusable applications with less code and shorter development time.

Function Overloading in Classes:

Function overloading is a feature of object-oriented programming where two or more functions can have the same name but different parameters. When a function name is overloaded with different jobs it is called Function Overloading. In Function Overloading “Function” name should be the same and the arguments should be different. The number of parameters can be either same with different data types or different based on the problem.

Overview of C++ Arrays:

An **array** is a consecutive group of memory locations that share the same type. Each array has its own size. To refer to a particular location or element in an array, we specify the name of the array and the position number of the specific element in the array. Similarly, **Multidimensional arrays** with two dimensions are often used to represent tables of values consisting of information arranged in rows and columns. This means that the arrays which require two subscripts to identify a particular element are called two-dimensional array.

Overview of Class Vector:

C++ Standard Library class template vector is similar to array but also supports resizing. By default, all the elements of an integer vector object are set to 0. The value of an element of a vector can be accessed or modified using square brackets ([]). Standard class template vector is defined in header (line 5) and belongs to namespace std.

Activities:

Pre-Lab Activities:

Overview of constructor types and destructor:

Constructors are special class functions which performs initialization of every object. The Compiler calls the Constructor whenever an object is created. Constructors initialize values to object members after storage is allocated to the object. While defining a constructor you must remember that the **name of constructor** will be same as the **name of the class**, and constructors will never have a return type. Constructors can be defined either inside the class definition or outside class definition using class name and scope : : operator. Constructors are of three types:

- **Default Constructor** is the constructor which doesn't take any argument. It has no parameter.
- **Parametrized Constructor** are the constructors with parameter. Using this Constructor you can provide different values to data members of different objects, by passing the appropriate values as argument.
- **Copy Constructor** are special type of Constructors which takes an object as argument and is used to copy values of data members of one object into another object. We will study copy constructors in detail later.

Just like other member functions, constructors can also be overloaded. In fact, when you have both default and parameterized constructors defined in your class you are having Overloaded Constructors, one with no parameter and other with parameter.

Overview of Function Overloading:

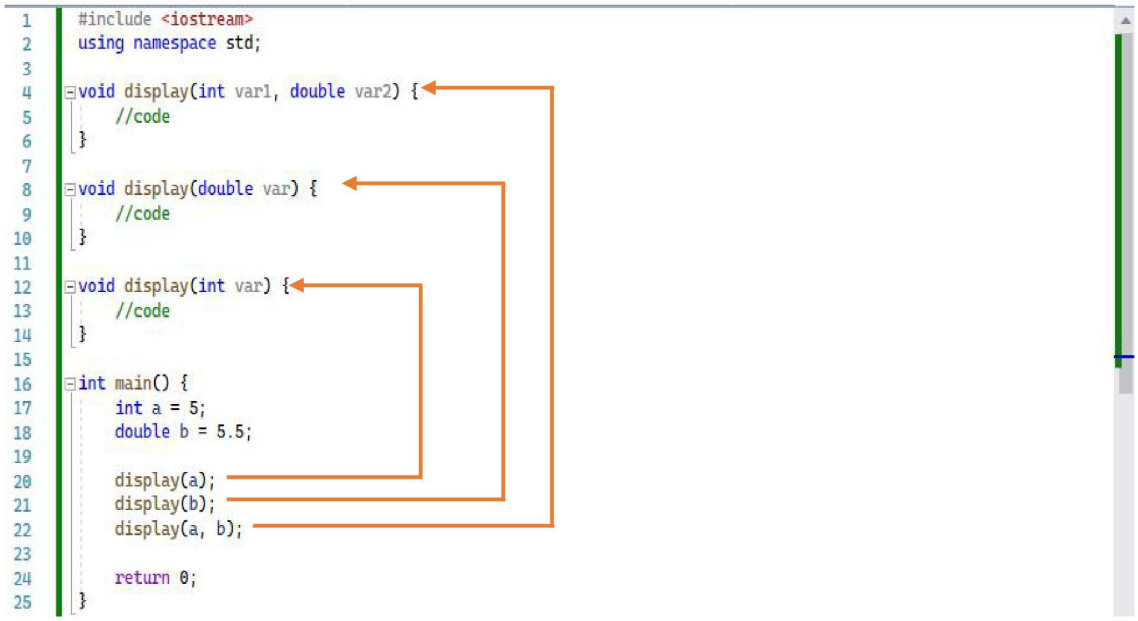
Function overloading is a feature of object-oriented programming where two or more functions can have the same name but different parameters. In Function Overloading "Function" name should be the same and the arguments should be different. There can be two possibilities, either the parameters count is the same with different type or parameters count is also different.

Example:

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

// function with 2 parameters
void display(int var1, double var2) {
    cout << "Integer number: " << var1;
    cout << " and double number: " << var2 << endl;
}

// function with double type single parameter
void display(double var) {
    cout << "Double number: " << var << endl;
}

// function with int type single parameter
void display(int var) {
    cout << "Integer number: " << var << endl;
}

int main() {
    int a = 5;
    double b = 5.5;

    display(a); // call function with int type parameter
    display(b); // call function with double type parameter
    display(a, b); // call function with 2 parameters

    return 0;
}
```

Fig. 01 (Function Overloading)

The calling to the overloaded function can be illustrated as:



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

void display(int var1, double var2) {
    //code
}

void display(double var) {
    //code
}

void display(int var) {
    //code
}

int main() {
    int a = 5;
    double b = 5.5;

    display(a);
    display(b);
    display(a, b);

    return 0;
}
```

Fig. 02 (Function Overloading)

Task 01: Introduction to C++ **[Estimated: 20 minutes / 15 marks]**

Write a program that lets the user to enter n integers into an array, and then display the count of all the even and odd numbers exist in that array. The program calls the following functions, which you have to implement.

- makeData() – accept an array with its size and fill it with the data entered by the user.
- display() – accept an array with its size and display its contents on the screen.
- getElement() – accept an array with its size and third parameter index, the function should return the contents exist on particular index of the array.
- countEvens() – accept an array with its size and calculate and return total number of evens exist in that array.
- countOdds() – accept an array with its size and calculate and return total number of evens exist in that array.
- bool removeFirst(): accepts array, with size and key. The function searches the array a for the item key. If key is found, its first occurrence is removed; all the elements above that position are shifted down, size is decremented, and true is returned to indicate a successful removal. If key is not found, the array is left unchanged and false is returned.
- dropSmallest() –accept a array with its size and return a pointer to a newly created array that contains all the elements of the passed array except the smallest one.

Task 02: Constructor Overloading **[Estimated: 30 minutes / 20 marks]**

Consider a class named **Job** that has a deadline as a data member and relevant getter/setter method(s). Assume you must schedule two earliest jobs on the basis of their deadlines. That is, if there are three jobs in the system with deadlines (deadline1, deadline2, and deadline3, respectively) then the system should report the top two earliest jobs (with the smallest *deadline* value). You might need to find the *deadline* with the smallest and second smallest value. Make sure to implement the

main functionality. Create 10 objects of class and initialize them with hardcoded values using parameterized constructor. Display the jobs with deadlines and find and report the two earliest jobs

Expected output (console):

Fig. 03 (Pre-Lab Task)

Task 3: Function overloading [Estimated 20 minutes / 15 marks]

Write a C++ program that asks a user to perform one of the 8 options:

- multiplication for integers
- multiplication for the doubles,
- division for integers,
- division for doubles
- multiplication of integer and double
- division of integer and double and it's vice versa and so on.

After this, assume that user has to input two numbers. Based on the option selected, User is now free to enter numbers from any type(multiplication, division), hence, it can be both integers, one integer one double, both doubles and so on. After this, call the respective overloaded function and display result. You must create the all the possible overloaded functions for the **multiplication () and division ()** functionality This cycle is repeated every time, until the user presses -1 to exit.

In-Lab Activities:

Introduction to C++ Arrays:

An array is a contiguous group of memory locations that all have the same type. To refer to a particular location or element in the array, we specify the name of the array and the position number of the element in the array. You refer to any one of these elements by giving the array name followed by the element's position number in square brackets ([]). The position number is more formally called a subscript or index (this number specifies the number of elements from the beginning of the array. The first element has subscript 0 (zero) and is sometimes called the zeroth element. Thus, the elements of array c are c [0] (pronounced "c subzero"), c [1], c [2] and so on. The highest subscript in array c is 11, which is 1 less than the number of elements in the array of size 12. array names follow the same conventions as other variable names.

Declaring Arrays:

To specify the type of the elements and the number of elements required by an array use a declaration of the form.

array<*type, arraySize*> *arrayName*;

The notation indicates that array is a class template. The compiler reserves the appropriate amount of memory based on the type of the elements and the arraySize. (Recall that a declaration which reserves memory is more specifically known as a definition.) The arraySize must be an unsigned integer. For example, to tell the compiler to reserve 12 elements for integer array c, use the declaration,

array<int,7> arr1;

Initializing Arrays:

The arrays are initialized using the loop or the initializer list. The concept is well explained in the Examples.

Example:

```cpp
#include <iostream>
#include<array>
using namespace std;
int main() {

    //Initializing using Initializer List
    array<int, 5> arr1{ 1, 2, 3, 4, 5 }; //initializes a array of size 5 having intial values 1,2,3,4,5

    array<int, 5> arr2;
    //Initializing using the loop
    for (int i{ 0 }; i < arr2.size(); ++i) {
        arr2[i] = 0;
    }
    cout << "Elements initialized using Initializer list: ";
    for (int i{ 0 }; i < arr1.size(); ++i) {
        cout << arr1[i] << " ";
    }
    cout << "\nElements initialized using the loop: ";
    for (int i{ 0 }; i < arr2.size(); ++i) {
        cout << arr2[i] << " ";
    }
    return 0;
}
```

Fig. 04 (Initializing Arrays)

Output:

Fig. 05 (Initializing Arrays)

Traverse array using range-based for loop:

Range-based for statement allows you to iterate through an array without using a counter, thus avoiding the possibility of "stepping outside" the array and eliminating the need for you to implement your own bounds checking. The example uses the range-based for to display an array's contents and to modify value of each element of array to "0".

Example

```
#include <iostream>
#include<array>
using namespace std;
int main() {

    array<int, 5> arr1{ 1, 2, 3, 4, 5 };
    cout << "Array Elements: ";
    for (int n : arr1)
    {
        cout << n << " "; //Display array "arr1"
    }
    for (int& n : arr1)
    {
        n = 0; //Modify each element to "0"
    }
    cout << "\nAfter Modification: ";
    for (int n : arr1)
    {
        cout << n << " "; //Display array after modification
    }
    return 0;
}
```

Fig. 06 (Range based loop)

Output:

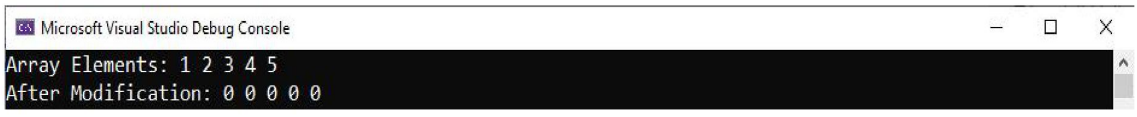


Fig. 07 (Range based loop)

Multi-dimensional Arrays:

Arrays that require two subscripts to identify a particular element are called two-dimensional arrays or 2-D arrays. Arrays with two or more dimensions are known as multidimensional arrays. Multidimensional arrays are an extension of 2-D matrices and use additional subscripts for indexing. A 3-D array, for example, uses three subscripts. The first two are just like a matrix, but the third dimension represents pages or sheets of elements.

The example below demonstrates initializing two-dimensional arrays in declarations. An array of arrays with two rows and two columns.

Example:

```cpp
#include <iostream>
#include<array>
using namespace std;
const size_t rows{ 2 };
const size_t cols{ 2 };
int main() {
    //Initializing two 2D Arrays
    array<array<int, cols>, rows> arr1{ 1, 2, 3, 4};
    array<array<int, cols>, rows> arr2{ 1, 2, 3 };

    cout << "Elements of arr1:\n";
    for (auto row : arr1)
    {
        for (auto n : row)
        {
            cout << n << " ";
        }
        cout << "\n";
    }
    cout << "Elements of arr2:\n";
    for (auto row : arr2)
    {
        for (auto n : row)
        {
            cout << n << " ";
        }
        cout << "\n";
    }
    return 0;
}
```

Fig. 08 (Multidimensional Arrays)

Output:

```
Microsoft Visual Studio Debug Console
Elements of arr1:
1 2
3 4
Elements of arr2:
1 2
3 0
```

Fig. 09 (Multidimensional Arrays)

Searching and Sorting Arrays:

Sorting data—placing it into ascending or descending order—is one of the most important computing applications. The std::sort() function generally takes two parameters, the first one being the point of the array/vector from where the sorting needs to begin and the second parameter being the length up to which we want the array/vector to get sorted. The third parameter is optional and can be used in cases such as if we want to sort the elements lexicographically.

Example:

```cpp
#include <iostream>
#include<array>
#include<algorithm>
using namespace std;
int main() {
    array<int,10> arr1{ 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };
    //Sorting Array in Ascending Order
    sort(arr1.begin(), arr1.end());

    cout << "Array sorting in acsending order: ";
    for (int i: arr1)
        cout << i << " ";

    //Sorting Array in Descending Order
    sort(arr1.begin(), arr1.end(),greater<int>());

    cout << "\nArray sorting in decsending order: ";
    for (int i : arr1)
        cout << i << " ";
    return 0;
}
```

Fig. 10 (Sorting)

Output:

```
Microsoft Visual Studio Debug Console
Array sorting in acsending order: 0 1 2 3 4 5 6 7 8 9
Array sorting in decsending order: 9 8 7 6 5 4 3 2 1 0
```

Fig. 11 (Sorting)

The process of finding a particular element of an array is called searching. Binary search is a widely used searching algorithm that requires the array to be sorted before search is applied. It works by comparing the middle item of the array with our target, if it matches, it returns true otherwise if the middle term is greater than the target, the search is performed in the left sub-array. If the middle term is less than the target, the search is performed in the right sub-array.

The prototype for binary search is:

binary_search(startaddress, endaddress, valuetofind);

Example:

```cpp
#include <iostream>
#include<array>
#include<algorithm>
using namespace std;
void search(array<int,10>arr, int element)
{
    if (binary_search(arr.begin(), arr.end(), element))
        cout << element <<" found in the array\n";
    else
        cout << element <<" not found in the array\n";
}
int main() {
    array<int,10> arr1{ 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };
    //Sorting Array in Ascending Order
    sort(arr1.begin(), arr1.end());

    //Finding Element "2" in array
    search(arr1, 2);

    //Finding Element "2" in array
    search(arr1, 12);
    return 0;
}
```

Fig. 12 (Searching)

Output:

```
Microsoft Visual Studio Debug Console
2 found in the array
12 not found in the array
```

Fig. 13 (Searching)

Task 01: (Fire the Salesmen) **Estimated[15 Marks/30 Minutes]**

Create a Class Salesman.

Attributes:

- Name
- Salary
- Sales_made (it is number of sales this salesman made in a month)

Methods

Make all necessary methods.

Main function:

In main function, user will tell the number of salesman. Create that many objects, take input from user for them and put them all in a vector named “salesmen”. Create an other function to which you will pass this vector named “FireSalesmen”. In this function, first you will sort the salesmen vector with respect to their sales_made attribute in descending order . After that remove the salemen that have made sales less that 10. You are not allowed to create an other container of any kind for this purpose.

Display the Remaining Salesmen in Main func.

Task 02: Arrays: [Estimated: 15 Marks/30 minutes]

Create a class Number.

Attributes:

- Numbers (STL array of type numbers).size 10.
- Size (size of array, value 10)

Methods:

- Create all required methods for this class.
- Create a method to input numbers in array.
- Create a methods to display all numbers in array.
- Create a methods to delete a number in array. Place -1 at the place of the number you want to delete in array.
- Create a method to get copy of array returned.
- Create a method to update an element in array. Take input of index number and new value from user and replace the already placed value with the new one.

Task 03: Srting Reverser [Estimated: 20 Marks/ 20 minutes]

Create a class Reverser. User can provide a string to it and it will return its reversed form using class methods.

- **Attributes:**
- Str (string)
- Str_Stack (stack STL)
- Helper_str (string)

Methods:

- Reverse_string_using_stack
- Reverse_string_using_string

The above methods should return a new string which will be the reversed form of str string in the class. Make setters and getters for str string only.

Post- Lab Activities:

Vector Class:

Vectors are part of the C++ Standard Template Library. To use vectors, we need to include the vector header file in our program as #include <vector>.

Vector Declaration :

Once we include the header file, here's how we can declare a vector in C++:

```
std::vector<T> vector_name;
```

The type parameter <T> specifies the type of the vector. It can be any primitive data type such as int, char, float, etc.

For example, **vector<int> num**; Here, num is the name of the vector. Notice that we have not specified the size of the vector during the declaration. This is because the size of a vector can grow dynamically so it is not necessary to define it.

Vector Initialization:

There are different ways to initialize a vector in C++.

1) Providing values directly to the vector: e.g. :
    a. vector<int> vector1 = {1, 2, 3, 4, 5}
    b. vector<int> vector2 {1, 2, 3, 4, 5};

Here, we are initializing the vector by providing values directly to the vector. Now, both vector1 and vector2 are initialized with values 1, 2, 3, 4, 5.

2) Providing the size of the vector. E,g:
    a. vector<int> vector3(5, 12);

Here, 5 is the size of the vector and 12 is the value. This code creates an int vector with size 5 and initializes the vector with the value of 12. So, the vector is equivalent to vector<int> vector3 = {12, 12, 12, 12, 12};

Basic Operations:

Add Element: To add a single element into a vector, we use the **push_back()** function. It inserts an element into the end of the vector**.**

Delete Element: To delete a single element from a vector, we use the **pop_back()** function.

Access Element: In C++, we use the index number to access the vector elements. Here, we use the **at()** function to access the element from the specified index**.**

Example:

The following example demonstrates the use of vector by taking input integers from user, adding them to vector and removing the odd integer.

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

int main() {
    int num;
    vector<int>v;
    cout << "Enter numbers: ";
    for (int i = 0; i < 10; i++)
    {
        cin >> num;
        v.push_back(num);//Adding element

        if (num % 2 != 0)//Checking if the number is odd at ith position
            v.pop_back();//Removing element
    }
    cout << "\n";
    for (auto itr : v)
        cout << itr << " ";
    return 0;
}
```

Fig. 15 (Vector Class)

Output:

Fig. 16 (Vector Class)

Task 01: [Vector] **[Estimated 20 marks/30 minutes]**

a) Use a vector of type integer to write the following function:

- **Input():** Get input from the user and return the filled vector
- **Display(vector):** display the vector onto the console.
- **getMin(vector):** displays the minimum from the vector.
- **getMax(vector):** displays the maximum from the vector.
- **countElements(vector):** displays the count of the elements in vector
- **sortElements(vector, order):** sort the elements in ascending or descending order.

b) Use a vector of type string to write following functions:

- Input(): Get input from the user and return the filled vector
- Display(vector): display the vector onto the console.
- ChangeCaps(vector): capitalize the first character of each element of a given string vector and return the new vector. E.g. vector contains “red” “green“ “black” “white” “Pink” elements, the new vector has “Red” “Green“ “Black” “White” “Pink”

- isNumberString(vector): return the new vector having elements which contains the numbers. E.g. vector contains “red12” “green“ “black2” “2white” “Pink” elements, the new vector has “red12” “Black2” “2White”.

Submissions:

- For In-Lab Activity:
  - Save the files on your PC.
  - TA's will evaluate the tasks offline.
- For Pre-Lab & Post-Lab Activity:
  - Submit the .cpp file on Google Classroom and name it to your roll no.

Evaluations Metric:

- All the lab tasks will be evaluated offline by TA's
- **Division of Pre-Lab marks:** **[50 marks]**
  - Task 01: Introduction to C++ [15 marks]
  - Task 02: Constructor Overloading [20 marks]
  - Task 03: Function Overloading [15 marks]
- **Division of In-Lab marks:** **[55 marks]**
  - Task 01: The Sieve of Eratosthenes [10 marks]
  - Task 02: Arrays [15 marks]
  - Task 03: Rolling Dice – 2D array [30 marks]
- **Division of Post-Lab marks:** **[20 marks]**
  - Task 01: Vector [20 marks]

References and Additional Material:

- STL Array C++
  https://www.geeksforgeeks.org/stdarray-in-cpp/

- STL Vector C++
  https://www.geeksforgeeks.org/vector-in-cpp-stl/

- Function Overloading
  https://www.geeksforgeeks.org/function-overloading-c/

Lab Time Activity Simulation Log:

- Slot – 01 – 00:00 – 00:15: Class Settlement
- Slot – 02 – 00:15 – 00:40: In-Lab Task
- Slot – 03 – 00:40 – 01:20: In-Lab Task
- Slot – 04 – 01:20 – 02:20: In-Lab Task
- Slot – 05 – 02:20 – 02:45: Evaluation of Lab Tasks
- Slot – 06 – 02:45 – 03:00: Discussion on Post-Lab Task