MIT World Peace University

Object Oriented Programming with Java and C++
Second Year B. Tech, Semester 1

Implementation of STL in C++

Practical Report

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A2, PA 20

November 6, 2022

Contents

1 Aim and Objectives

- To understand the user of Standard Template Library in C++
- To get familiar with list containers and iterators.

2 Problem Statement

A shop maintains the inventory of items. It stores information of items like ItemCode, ItemName, Quantity and Cost of it in a list of STL. Whenever Customer wants to buy an item, sales person inputs the ItemCode and or ItemName and the system searches in a file and displays whether it is available or not otherwise an appropriate message is displayed. If it is, then the system displays the item details and request for the quantity of items required. If the requested quantity of items are available, the total cost of items is displayed; otherwise the message is displayed as required items not in stock. After purchasing an item, system updates the list. Design a system using a class called Items with suitable data members and member functions. Implement a Menu Driven C++ program using STL concepts.

3 Theory

3.1 Concept of Standard Template Library

The Standard Template Library (STL) is a set of C++ template classes to provide common programming data structures and functions such as lists, stacks, arrays, etc. It is a library of container classes, algorithms, and iterators. It is a generalized library and so, its components are parameterized. It has a lot of useful things that we can use in our own code, without worrying to write lengthy implementations of basic data structure concepts.

STL has 4 components:

1. Algorithms : *They act on containers and provide means for various operations for the contents of the containers.*
2. Containers : *Containers or container classes store objects and data.*
3. Functions :*The STL includes classes that overload the function call operator. Instances of such classes are called function objects or functors.*
4. Iterators : *As the name suggests, iterators are used for working upon a sequence of values. They are the major feature that allows generality in STL.*

3.2 How is STL different from the C++ Standard Library?

The **STL** was written by Alexander Stepanov in the days long before C++ was standardised. The STL was already widely used as a library for C++, giving programmers access to containers, iterators and algorithms. When the standardisation happened, the language committee designed parts of the C++ Standard Library (which is part of the language standard) to very closely match the STL.

Over the years, many people — including prominent book authors, and various websites — have continued to refer to the C++ Standard Library as **The STL** despite the fact that the two entities are separate and that there are some differences. These differences are even more pronounced in the upcoming new C++ standard, which includes various features and significantly alters some classes.

So A lot of the functions and containers were written before the formation of various important libraries that we now use in C++. It is those libraries that are called the "STL". C++ standard libaries are ones written after it using those libraries in part.

3.3 Concept of Containers, Ierators and Algorithms

3.3.1 Containers

In C++, there are generally 3 kinds of STL containers:

- Sequential Containers : *In C++, sequential containers allow us to store elements that can be accessed in sequential order. Internally, sequential containers are implemented as arrays or linked lists data structures.*

  Types of Sequential Containers

  - Array
  - Vector
  - Deque
  - List
  - Forward List

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

int main() {
  // initialize a vector of int type
  vector<int> numbers = {1, 100, 10, 70, 100};

  // print the vector
  cout << "Numbers are: ";
  for(auto &num: numbers) {
    cout << num << ", ";
  }

  return 0;
}

//Output
Numbers are: 1, 100, 10, 70, 100,
```

- Associative Containers: *In C++, associative containers allow us to store elements in sorted order. The order doesn't depend upon when the element is inserted. Internally, they are implemented as binary tree data structures.* Types of associative Containers.

  - Set
  - Map
  - Multiset
  - Multimap

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

      int main() {

        // initialize a set of int type
        set<int> numbers = {1, 100, 10, 70, 100};

        // print the set
        cout << "Numbers are: ";
        for(auto &num: numbers) {
          cout << num << ", ";
        }

        return 0;
      }
      // Output:
      Numbers are: 1, 10, 70, 100,

```

- Unordered Associative Containers: *In C++, STL Unordered Associative Containers provide the unsorted versions of the associative container. Internally, unordered associative containers are implemented as hash table data structures.* Types of Unordered Associated Containers
  - Unordered Set
  - Unordered Map
  - Unordered Multiset
  - Unordered Multimap

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

int main() {

  // initialize an unordered_set of int type
  unordered_set<int> numbers = {1, 100, 10, 70, 100};

  // print the set
  cout << "Numbers are: ";
  for(auto &num: numbers) {
    cout << num << ", ";
  }

  return 0;
}
// Output
Numbers are: 70, 10, 100, 1,

```

3.3.2 Iterators

Iterators are one of the four pillars of the Standard Template Library or STL in C++. An iterator is used to point to the memory address of the STL container classes. For better understanding, you can

relate them with a pointer, to some extent. Iterators act as a bridge that connects algorithms to STL containers and allows the modifications of the data present inside the container. They allow you to iterate over the container, access and assign the values, and run different operators over them, to get the desired result.

Applications of Iterators:

1. The primary objective of an iterator is to access the STL container elements and perform certain operations on them.
2. The internal structure of a container does not matter, since the iterators provide common usage for all of them.
3. Iterator algorithms are not dependent on the container type.
4. An iterator can be used to iterate over the container elements. It can also provide access to those elements to modify their values.
5. Iterators follow a generic approach for STL container classes. This way, the programmers dont need to learn about different iterators for different containers.

Example to Demonstrate use of Iterators

```
#include<iostream>
#include<iterator> // for iterators
#include<vector> // for vectors
using namespace std;
int main()
{
  vector<int> ar = { 1, 2, 3, 4, 5 };

  // Declaring iterator to a vector
  vector<int>::iterator ptr;

  // Displaying vector elements using begin() and end()
  cout << "The vector elements are : ";
  for (ptr = ar.begin(); ptr < ar.end(); ptr++)
    cout << *ptr << " ";

  return 0;
}
//Output
The vector elements are : 1 2 3 4 5
```

3.3.3 Algorithms

STL provide different types of algorithms that can be implemented upon any of the container with the help of iterators. Thus now we don't have to define complex algorithm instead we just use the built in functions provided by the algorithm library in STL.

Algorithm functions provided by algorithm library works on the iterators, not on the containers. Thus one algorithm function can be used on any type of container. Use of algorithms from STL saves time, effort, code and are very reliable.

There are many types of algorithms already implemented reliably in the STL. Here we will use a simple Non Modifying Algorithm as an example.

```cpp
#include <iostream>
#include <algorithm>
using namespace std;
int main ()
{
  int values[] = {5,1,6,9,10,1,12,5,5,5,1,8,9,7,46};
  int count_5 = count(values, values+15, 5);
  cout<<"The number of times '5' appears in array= "<<count_5;
  return 0;
}
// Output
The number of times 5 appears in an array= 4
```

4 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers : g++ and gcc on linux for C++

5 Input

For C++

1. Basic menu to add new elements, or purchase an item.
2. The Details of the item to add.
3. The quantity and code of the product that you wanna purchase.

6 Output

For C++

1. A list of all the elements in the Database.
2. Their codes and Quantities
3. Appropriate messages after each action is performed.

7 Code

7.1 C++ Implementation of Problem A

```cpp
// Shop has item code, item name, quantity, and the cost.
// Input would be some item name, where we would first add things to the shop
    inventory
// Another option would be to check for the item in the database which is a list.
// If the item is found in the shop then you are supposed to ask them the number
    of items.
// if its available then you sell it, or else you tell them that that may items
    arent in stock.
```

```

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

// struct so we can put it in a single linked list.
struct Items
{
    int item_code;
    string item_name;
    int item_quantity;
    int item_cost;
    Items(int a, string b, int q, int c) : item_code(a), item_name(b),
    item_quantity(q), item_cost(c)
    {
    }
} currentItem(0, "", 0, 0), itemToAdd(0, "", 0, 0);

// linked list items so we can put objects of structures in it easily.
list<Items> items = {
    Items(101, "Burger", 10, 150),
    Items(102, "Fries", 10, 129),
    Items(103, "Ice Cream", 10, 40),
    Items(104, "Coke", 10, 50),
};

bool itemFound = false;

// inserts items in the list
void insertItems()
{
    cout << "Enter the details of the Item that you wanna enter to the database"
    << endl;
    cout << "Enter the Code of the new Item: ";
    cin >> itemToAdd.item_code;

    cout << "Enter the Name of the new Item: ";
    cin >> itemToAdd.item_name;

    cout << "Enter the Quantity of the new Item: ";
    cin >> itemToAdd.item_quantity;

    cout << "Enter the Cost of the new Item: ";
    cin >> itemToAdd.item_cost;

    items.push_back(itemToAdd);
    cout << "Item added successfully" << endl;
}

// returns true or false depending on whether the item was found
bool searchItem(int itemCode)
{
    int i = 0;
    for (list<Items>::iterator it = items.begin(); it != items.end(); it++, i++)
    {
        struct Items temp = *it;
        if (temp.item_code == itemCode)
        {
            currentItem = temp;
```

```
            return true;
        }
    }
    return false;
}

// just find the element at the element currentItemIndex, and replace it with the currentItem struct object.
void updateItems(int quantity, int currentItemCode)
{
    for (list<Items>::iterator it = items.begin(); it != items.end(); it++)
    {
        struct Items temp = *it;
        if (temp.item_code == currentItemCode)
        {
            currentItem.item_quantity -= quantity;
            items.erase(it);
            items.push_back(currentItem);
        }
    }
}
// display the things, and check if the selected item code by the user exists in the thing by calling searchItem. Then input the item quantity, check for it, and update the currentItem object.
// then call the updateItem function.
bool displayAndPurchaseItems()
{
    int selectedItemCode = -1;
    int selectedQuantity = 0;
    struct Items tempItem(0, "", 0, 0);
    cout << "The Items that you can buy are: " << endl;

    for (list<Items>::iterator it = items.begin(); it != items.end(); it++)
    {
        tempItem = *it;
        cout << tempItem.item_code << " " << tempItem.item_name << " " << tempItem.item_quantity << " " << tempItem.item_cost << endl;
    }
c:
    try
    {
        cout << "Please enter the code of the item that you wanna buy" << endl;
        cin >> selectedItemCode;
        if (!searchItem(selectedItemCode))
        {
            throw selectedItemCode;
        }
    l:
        cout << "Enter the Quantity of the Item that you wanna buy. "
             << endl;
        cout << "Max quantity is: " << currentItem.item_quantity << endl;
        cin >> selectedQuantity;
        try
        {
            if (selectedQuantity > currentItem.item_quantity)
            {
                throw selectedQuantity;
            }
            else
```

```
118             {
119                 cout << "Thank you for Purchasing!" << endl;
120                 updateItems(selectedQuantity, selectedItemCode);
121             }
122         }
123         catch (int something)
124         {
125             cout << "Quantity you entered is too much! Try again!" << endl;
126             goto l;
127         }
128         return true;
129     }
130     catch (int something)
131     {
132         cout << "The code of the item that you entered doesnt exist. Try again. "
    << endl;
133         goto c;
134     }
135     return true;
136 }
137 
138 int main()
139 {
140     int choice = 0;
141     struct Items;
142     cout << "Welcome to McRonalds" << endl;
143 
144     do
145     {
146         cout << endl
147              << "What do you wanna do?\n\
148         1. Add new Items\n\
149         2. Purchase Item\n\
150         3. Quit\n"
151              << endl;
152         cin >> choice;
153         switch (choice)
154         {
155         case 1:
156             insertItems();
157             break;
158         case 2:
159             if (!displayAndPurchaseItems())
160             {
161                 cout << "Item couldnt be purchased!" << endl;
162             }
163             else
164             {
165                 cout << "Item purchased and Updated Successfully" << endl;
166             }
167             break;
168         case 3:
169             cout << "Thanks for Visiting our store!" << endl;
170             break;
171         default:
172             cout << "Sorry, we cant do that in this store" << endl;
173             break;
174         }
175 
```

```
176         itemFound = false;
177     } while (choice != 3);
178     return 0;
179 }
```

Listing 1: Main.Cpp

7.1.1 C++ Input and Output

```
1  Welcome to McRonalds
2
3  What do you wanna do?
4          1. Add new Items
5          2. Purchase Item
6          3. Quit
7
8  1
9  Enter the details of the Item that you wanna enter to the database
10 Enter the Code of the new Item: 106
11 Enter the Name of the new Item: Water
12 Enter the Quantity of the new Item: 100
13 Enter the Cost of the new Item: 30
14 Item added successfully
15
16 What do you wanna do?
17         1. Add new Items
18         2. Purchase Item
19         3. Quit
20
21 1
22 Enter the details of the Item that you wanna enter to the database
23 Enter the Code of the new Item: 107
24 Enter the Name of the new Item: Lemonade
25 Enter the Quantity of the new Item: 25
26 Enter the Cost of the new Item: 40
27 Item added successfully
28
29 What do you wanna do?
30         1. Add new Items
31         2. Purchase Item
32         3. Quit
33
34 2
35 The Items that you can buy are:
36 101 Burger 10 150
37 102 Fries 10 129
38 103 Ice Cream 10 40
39 104 Coke 10 50
40 106 Water 100 30
41 107 Lemonade 25 40
42 Please enter the code of the item that you wanna buy
43 10
44 The code of the item that you entered doesnt exist. Try again.
45 Please enter the code of the item that you wanna buy
46 103
47 Enter the Quantity of the Item that you wanna buy.
48 Max quantity is: 10
49 24
50 Quantity you entered is too much! Try again!
```

```
Enter the Quantity of the Item that you wanna buy.
Max quantity is: 10
2
Thank you for Purchasing!
Item purchased and Updated Successfully

What do you wanna do?
        1. Add new Items
        2. Purchase Item
        3. Quit

2
The Items that you can buy are:
101 Burger 10 150
102 Fries 10 129
104 Coke 10 50
106 Water 100 30
107 Lemonade 25 40
103 Ice Cream 8 40
Please enter the code of the item that you wanna buy
106
Enter the Quantity of the Item that you wanna buy.
Max quantity is: 100
3
Thank you for Purchasing!
Item purchased and Updated Successfully

What do you wanna do?
        1. Add new Items
        2. Purchase Item
        3. Quit

2
The Items that you can buy are:
101 Burger 10 150
102 Fries 10 129
104 Coke 10 50
107 Lemonade 25 40
103 Ice Cream 8 40
106 Water 97 30
Please enter the code of the item that you wanna buy
101
Enter the Quantity of the Item that you wanna buy.
Max quantity is: 10
1
Thank you for Purchasing!
Item purchased and Updated Successfully

What do you wanna do?
        1. Add new Items
        2. Purchase Item
        3. Quit

3
Thanks for Visiting our store!
```

Listing 2: Output for Problem 1

8 Conclusion

Thus, the purpose of the STL libraries in C++ was understood, and implemented successfully. Containers like lists and iterators for them were also used and understood.

9 FAQs

1. **What are class templates? How are they created? What is the need for class templates?**
   Templates are powerful features of C++ which allows us to write generic programs. There are two ways we can implement templates:

   - Function Templates
   - Class Templates

   Similar to function templates, we can use class templates to create a single class to work with different data types. There are few, but important reasons to use class templates:

   - Class templates come in handy as they can make our code shorter and more manageable.
   - If you have various functions that you wanna implement on a set of data, but you arent sure which data type will be given as input, then you can use class templates.
   - Example: If you creating a calculator, a class template to include basic functions like addition, subtraction etc would be perfect as you can get an integer or a floating point value as your input for your calculator.

2. **Create a template for bubble sort functions.**

```
template <class T>
T bubbleSort(T arr[], int n)
{
  int i, j;
  for (i = 0; i < n - 1; i++)
    for (j = 0; j < n - i - 1; j++)
      if (arr[j] > arr[j + 1])
        swap(arr[j], arr[j + 1]);
}


```

3. **Explain with example, how Function Templates are implemented?**

```
#include <iostream>
using namespace std;
template<class T> T add(T &a,T &b)
{
  T result = a+b;
  return result;
}
int main()
{
  int i =2;
  int j =3;
  float m = 2.3;
  float n = 1.2;
  cout<<"Addition of i and j is :"<<add(i,j);
  cout<<'\n';
  cout<<"Addition of m and n is :"<<add(m,n);
  return 0;
```

```
}

```

4. **Explain with example how can a class template be created.**

```
// Class template
template <class T>
class Number {
  private:
  // Variable of type Tf
  T num;

  public:
  Number(T n) : num(n) {}   // constructor

  T getNum() {
    return num;
  }
};

int main() {

  // create object with int type
  Number<int> numberInt(7);

  // create object with double type
  Number<double> numberDouble(7.7);

  cout << "int Number = " << numberInt.getNum() << endl;
  cout << "double Number = " << numberDouble.getNum() << endl;

  return 0;
}
// Output
int Number = 7
double Number = 7.7

```

5. **Explain Generic functions and Generic class.**

- Generic functions use the concept of a function template. Generic functions define a set of operations that can be applied to the various types of data.
- The type of the data that the function will operate on depends on the type of the data passed as a parameter.
- For example, Quick sorting algorithm is implemented using a generic function, it can be implemented to an array of integers or array of floats.
- A Generic function is created by using the keyword template. The template defines what function will do.
- Just like a class is a collection of a bunch of functions, that can be inherited and instantiated at once, generic functions are similar in that manner, except in a generic class, you could use a generic data type, and this would be applicable and useful for implementing each function in the Class. So each function in the class can be called and can manipulate variables of the genereic data type for which the class is defined.