19CSE201 :Advanced Programming

Lecture 24
STLs in C++

By
Ritwik M
Assistant Professor(SrGr)
Dept. Of Computer Science & Engg

Based on material from multiple sources from the internet

Function Templates

- Syntax

```
template <class typename>
function_declaration;
```

- Example

```
template <class myType>
myType GetMax (myType varA, myType varB)
{ return (A>B?A:B); }
```

myType is a placeholder.
The compiler automatically
replaces it during runtime.

Function Templates - Example

```
template <class T>
T GetMax (T a, T b)
{
 return (a>b?a:b);
}

int main ()
{
 int i=5, j=6, k;
 long l=10, m=5, n;

 k=GetMax(i,j);
 n=GetMax(l,m);
 cout << k << endl; cout << n << endl;
 return 0;
}
```

Class Template

- A class can have members that use template parameters as types.

```
template <class T>
class mypair {
  T a, b;
  public:
    mypair (T first, T second)
      {
        a=first; b=second;
      }
    T getmax ();
};
```

```
template <class T>
T mypair<T>::getmax()
{
  T retval;
  retval = a>b? a : b;
  return retval;
}
```

```
int main ()
 {
   mypair <int> myobject (100, 75);
   cout << myobject.getmax();
   return 0;
 }
```

Standard Template Library (STL)

- STL is a collection of standard C++ template classes.
- It consists of generic methods and classes to work with different forms of data.
- STLs saves a lot of effort, reduces the redundancy of the code
  - leads to the increased optimization of the code blocks.
- usually used for building Data Structures

The Stack Data structure

- Stack follows the Last-In-First-Out (LIFO) fashion.
- The items are inserted at one end of the stack and an item is deleted from the same end of the stack
- Syntax
  - `stack <data_type> stack_name;`

Functions for a generic stack

- `stack.push(element)`
  - Inserts an element to the top of the stack.
- `stack.pop()`
  - deletes an element present at the top of the stack.
- `stack.empty()`
  - Checks whether the stack is empty or not.
- `stack.top()`
  - Returns the element present at the top of the stack.

Stack STL - Example

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

void display(stack <int> S)
{
    while (!S.empty())
    {
        cout << '\t' << S.top();
        S.pop();
    }
    cout << '\n';
}
```

```
int main () {
 stack <int> s;
 s.push(1);
 s.push(0);
 s.push(2);
 s.push(6);
 cout << "The stack is : ";
 display(s);
 cout << "The top element:\n" << s.top();
 cout << "After removing top element from
  the stack:\n";
  s.pop();
  display(s);
  return 0;
}
```

The Queue Data structure

- Stack follows the First-In-First-Out (FIFO) fashion.
- The items are inserted from the end and deleted from the front of the queue
- Syntax
  - `queue <data_type> queue_name;`

Functions for a generic Queue

- `queue.empty()`
  - Checks whether the queue is empty or not.
- `queue.push(element)`
  - Adds an element to the end of the queue.
- `queue.pop()`
- Deletes the first element of the queue.
- `queue.front()`
  - Returns an iterator element which points to the first element of the queue.
- `queue.back()`
  - Returns an iterator element which points to the last element of the queue.

Queue STL - Example

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

void display(queue <int> Q1)
{
    queue <int> Q = Q1;
    while (!Q.empty())
    {
        cout << '\t' << Q.front();
        Q.pop();
    }
    cout << '\n';
}
```

```
int main()
{
    int i=1;
    queue <int> qd;
    while (i<5)
    {
        qd.push(i);
        i++;
    }
    cout << "Queue:\n";
    display(qd);
    cout<<"Popping an element from queue..\n";
    qd.pop();
    display(qd);
    return 0;
}
```

Linked List

- Lists are sequence containers that allow non-contiguous memory allocation.
- List has slow traversal, but once a position has been found, insertion and deletion are quick.
- Here, we focus on the doubly linked list.
- For implementing a singly linked list, we use forward list.

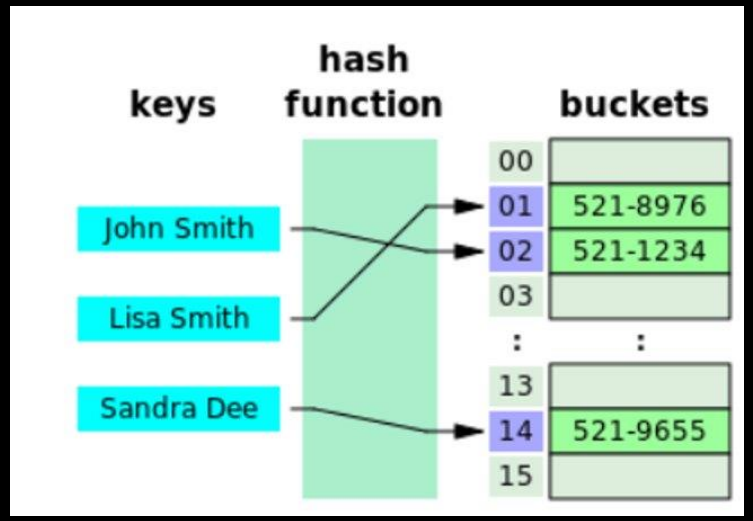
Linked List - Example

```
#include <iostream>
#include <list>
#include <iterator>
using namespace std;
void showlist(list <int> g)
{
 list <int> :: iterator it;
 for(it = g.begin(); it != g.end(); ++it)
 {   cout << '\t' << *it; }
 cout << '\n';
}
```

```
int main(){
      list <int> gqlist1, gqlist2;
      for (int i = 0; i < 10; ++i){
        gqlist1.push_back(i * 2);
        gqlist2.push_front(i * 3);
    }
    cout << "\nList 1 (gqlist1) is : ";
    showlist(gqlist1);
    cout << "\nList 2 (gqlist2) is : ";
    showlist(gqlist2);
 cout << "\ngqlist1.front() : " << gqlist1.front();
 cout << "\ngqlist1.back() : " << gqlist1.back();
    cout << "\ngqlist1.pop_front() : ";
    gqlist1.pop_front();
    showlist(gqlist1);
    cout << "\ngqlist2.pop_back() : ";
    gqlist2.pop_back();
    showlist(gqlist2);
    return 0;}
```

Hash Table

- Hash table (also, *hash map*) is a data structure that basically maps keys to values.
- A hash table uses a *hash function* to compute an *index* into an array of *buckets* or *slots*, from which the corresponding value can be found.
- Example



Hash STL -Example

```
#include<iostream>
#include<list>
using namespace std;
class Hash{
   int BUCKET; // No. of buckets
    list<int> *table;
public:
    Hash(int V);  // Constructor
     void insertItem(int x);
     void deleteItem(int key);
     int hashFunction(int x) {
        return (x % BUCKET);
    }
      void displayHash();
};
```

```
Hash::Hash(int b){
    this->BUCKET = b;
    table = new list<int>[BUCKET];
}

void Hash::insertItem(int key){
    int index = hashFunction(key);
    table[index].push_back(key);
}

void Hash::deleteItem(int key){
  // get the hash index of key
  int index = hashFunction(key);

 // find the key in (index)th list
  list <int> :: iterator i;
  for (i =table[index].begin();i!=table[index].end();i++){
    if (*i == key)
      break;
  }
 // if key is found in hash table, remove it
  if (i != table[index].end())
    table[index].erase(i);
} // example Continues in next slide
```

Hash STL – Example Cont.

```
// function to display hash table
void Hash::displayHash() {
  for (int i = 0; i < BUCKET; i++) {
    cout << i;
    for (auto x : table[i])
      cout << " --> " << x;
    cout << endl;
  }
}
```

```
int main()
{
  // array that contains keys to be mapped
  int a[] = {15, 11, 27, 8, 12};
  int n = sizeof(a)/sizeof(a[0]);

  // insert the keys into the hash table
  Hash h(7); // 7 is count of buckets in hash table
  for (int i = 0; i < n; i++)
    h.insertItem(a[i]);

  // delete 12 from hash table
  h.deleteItem(12);
  // display the Hash table
  h.displayHash();
  return 0;
}
```

Quick Summary

- Templates in C++
  - Function templates
  - Class templates
- Standard Template Library (STL)
- Stack using STLs
- Queue using STLs
- Linked List using STLs
- Hash Table using STLs
- Examples
- Exercises

Up Next

ADTs