Standard Template Library

Standard Template Library

- **Standard Template Library (STL):** a library provide by C++, containing templates for frequently used data structures and algorithms
- Ready made data structures are generic & can be used with any data type.
- It makes development faster
- STL data structures are growable & resizable

Standard Template Library

- STL Contains
  - **Containers:** classes that store data and impose some organization on it
  - **Iterators:** like pointers; provides mechanisms for accessing elements in a container. Iterator support forward, reverse and Bidirectional traversal
  - **Algorithms:** Set of functions that process data stored in containers

Containers

- **Sequential containers:** organize and access data sequentially, as in an array. Both side access of data.
  - **vector :** provide fast random access
  - **list :** supports fast insertion and deletion
  - **dequeue:** double ended queue support both side insertion & deletion

Containers

- **Sequential container Adapters:** Adapters use sequential containers internally but data access is only from one end(side)
  - **stack:** Last In First Out manner
  - **queue:** First In First Out manner
  - **Priority_queue:** priority managed queue

Containers

- **Associative Containers:** use keys to allow data elements to be quickly accessed. **It stores key value pairs.**
  - **map:** support duplicate values but unique key
  - **set:** collection of unique elements
  - **multimap:** same key can appear multiple time
  - **multiset :** duplicate elements can be inserted

Creating Object of Containers

- To create a list of int, write

  list<int> mylist;

- To create a vector of string objects, write

  vector<string> myvector;

- Requires the <vector> and <list> header files
- Practical Example

Iterators

- Generalization of pointers, used to access information in containers
- Types of Iterators
  - forward (uses ++)
  - bidirectional (uses ++ and --)
  - random-access
  - input (can be used with cin and istream objects)
  - output (can be used with cout and ostream objects)

Containers and Iterators

- Each container class defines an iterator type, used to access its contents
- The type of an iterator is determined by the type of the container:

  list<int>::iterator x;
  list<string>::iterator y;
  x is an iterator for a container of type list<int>

- Each container class defines functions that return iterators:

  begin(): returns iterator to item at start

  end(): returns iterator denoting end of container

Containers and Iterators

- Iterators support pointer-like operations: if iter is an iterator:
  - ***iter** is the item it points to: this dereferences the iterator
  - **iter++** advances to the next item in the container
  - **iter--** backs up in the container
- The end() iterator points to past the end: it should never be dereferenced

Traversing a Container

- **Creating vector:**

```
vector<int> v;
for  (int k=1; k<= 5; k++)
v.push_back(k*k);
```

- **Traverse vector using iterators:**

```
vector<int>::iterator iter = v.begin();
while (iter != v.end())
  { cout << *iter << " ";
    iter++
  }
```

Thank You

Lazy programmers !
I am providing you ready made functionality!!!
…… STL