Preview

- STL: Stack
  - push, pop, empty, constructor, size top
- STL: Queue
  - Push (enqueue), pop(dequeue), empty
- STL: Set
  - Set Member Functions

1

STL: Stack

- **LIFO stack**
- Stacks are a type of container adaptors, specifically designed to operate in a LIFO context (last-in first-out), where elements are inserted and extracted only from the end of the container

2

STL: Stack Member Functions

Member functions

(constructor)	Construct stack (public member function)
empty	Test whether container is empty (public member function)
size	Return size (public member function)
top	Access next element (public member function)
push	Add element (public member function)
pop	Remove element (public member function)

3

```cpp
// stack1.cpp
// STL: stack member functions
// constructor, push(), pop(), empty() size()
#include <iostream>
#include <stack>
#include <unistd.h>
using namespace std;

int main ()
{
  stack<int> mystack; // create am integer stack

  for (int i=0; i<5; ++i)
  {
      cout <<"Now inserting "  << i <<" in the stack " <<endl;
      sleep(1);
      mystack.push(i);
  }
  cout <<"size of mystack is " <<mystack.size()<<endl;

  cout << "Popping out elements from the top of the stack..";
  while (!mystack.empty())
  {
     cout << " Now popping " << mystack.top() << " from the stack.." <<endl;
     sleep(1);
     mystack.pop();
  }
  cout<<endl;
  cout <<"Now size of mystack become " <<mystack.size() <<endl;
  cout << endl;

  return 0;
}
```


4

STL Stack with Structured Data

```cpp
// stack2.cpp
// stack STL with strutured data type Student
#include <iostream>
#include <stack>
using namespace std;
struct Student {
          char LastName[20];     //last name
          char FirstName[20];    // First Name
          int IDNumber;          // Student ID
          Student();             // Constructor
          Student(int);          // it is Null constructor for  declare
          bool operator == (const Student);
          bool operator > (const Student);
          bool operator < (const Student);
          friend ostream& operator << (ostream &stream, const Student &student);
};
// Student Constructor
Student::Student()
{
          cout << "What is the student's Last Name: ";
          cin >> LastName;
          cout << "What is the student's First Name: ";
          cin >> FirstName;
          cout << "What is the student's ID#: ";
          cin >> IDNumber;
}
Student::Student(int x)
{;}
```


5

```cpp
// == Operator Overloader: checks if the numbers are equal
bool Student::operator == (Student x)
{
          return (x.IDNumber == IDNumber);
}
// > Operator Overloader: checks if the number is greater than
bool Student::operator > (Student x)
{
          return (IDNumber > x.IDNumber);
}
// < Operator Overloader: checks if the number is less than
bool Student::operator < (Student x)
{
          return (IDNumber < x.IDNumber);
}
// << Operator Overloader: displays the structures information
ostream& operator << (ostream &stream, const Student &student)
{
          stream << "ID#: " << student.IDNumber <<" - " << student.LastName << ", " << student.FirstName
<< endl;
          return(stream);
}
int main ()
{
  // vector for Student structure
  stack <Student> StdStack;
  for (int i= 1; i <4; i++)
  {
          Student *ptr;
          ptr = new Student;
          StdStack.push(*ptr);
  }
  cout << "Popping out elements...Now..."<<endl;
  while (!StdStack.empty())
  {
     cout << " " << StdStack.top();
     StdStack.pop();
  }
  cout<<endl;
  cout <<"Now size of student stack become " <<StdStack.size()<<endl;
  cout << endl;
  return 0;
}
```

6

STL: Queue

- **FIFO queue**
- **queue**s are a type of container adaptors, specifically designed to operate in a FIFO context (first-in first-out), where elements are inserted into one end of the container and extracted from the other.

7

STL: Queue Member Functions

Member functions

(constructor)	Construct queue (public member function)
empty	Test whether container is empty (public member function)
size	Return size (public member function)
front	Access next element (public member function)
back	Access last element (public member function)
push	Insert element (public member function)
pop	Delete next element (public member function)

8

```cpp
// queue1.cpp
// STL: queue member functions
// push(), pop(), empty(), front(), size()
#include <iostream>
#include <queue>
using namespace std;

int main ()
{
  queue<int> myqueue;
  int myint;
  cout << "Please enter some integers (enter 0 to end):\n";
  do {
        cin >> myint;
        if (myint != 0)
           myqueue.push (myint);
  } while (myint);
  cout << "current size of myqueue is " <<myqueue.size() <<endl;

  cout << "myqueue contains: ";
  while (!myqueue.empty())
  {
    cout << " " << myqueue.front();
    myqueue.pop();
  }
  cout <<endl;
  return 0;
}
```


9

STL: Queue with Structured Data

```cpp
// queue2.cpp
// queue STL with strutured data type Student
#include <iostream>
#include <queue>
using namespace std;
struct Student {
          char LastName[20];     //last name
          char FirstName[20];    // First Name
          int IDNumber;          // Student ID
          Student();             // Constructor
          Student(int);          // it is Null constructor for  declare
          bool operator == (const Student);
          bool operator > (const Student);
          bool operator < (const Student);
          friend ostream& operator << (ostream &stream, const Student &student);
};
// Student Constructor
Student::Student()
{
          cout << "What is the student's Last Name: ";
          cin >> LastName;
          cout << "What is the student's First Name: ";
          cin >> FirstName;
          cout << "What is the student's ID#: ";
          cin >> IDNumber;
}
Student::Student(int x)
{;}
```


10

```cpp
// == Operator Overloader: checks if the numbers are equal
bool Student::operator == (Student x)
{
          return (x.IDNumber == IDNumber);
}
// > Operator Overloader: checks if the number is greater than
bool Student::operator > (Student x)
{
          return (IDNumber > x.IDNumber);
}
// < Operator Overloader: checks if the number is less than
bool Student::operator < (Student x)
{
          return (IDNumber < x.IDNumber);
}
// << Operator Overloader: displays the structures information
ostream& operator << (ostream &stream, const Student &student)
{
          stream << "ID#: " << student.IDNumber <<" - " << student.LastName << ", " << student.FirstName
<< endl;
          return(stream);
}
int main ()
{
  // vector for Student structure
  queue <Student> StdQueue;
  for (int i= 1; i <4; i++)
  {
            Student *ptr;
            ptr = new Student;
            StdQueue.push(*ptr);
  }
  cout << "Popping out elements from the Queue...Now..."<<endl;
  while (!StdQueue.empty())
  {
     cout << " " << StdQueue.front();
     StdQueue.pop();
  }
  cout<<endl;
  cout <<"Now size of student stack become " <<StdQueue.size()<<endl;
  cout << endl;
  return 0;
}
```

11

STL: Set

Set

- Sets are a kind of associative containers that stores unique elements, and in which the elements themselves are the *keys*.
- Associative containers are containers especially designed to be efficient accessing its elements by their key (unlike sequence containers, which are more efficient accessing elements by their relative or absolute position).

12

STL: Set

- Internally, the elements in a set are always sorted from lower to higher
- Sets are typically implemented as *binary search trees*
- Main characteristics of set:
  - Unique element values: no two elements in the set can compare equal to each other.
  - The element value is the key itself.

13

STL: Set Member Functions

Member functions

(constructor)	Construct set (public member function)
(destructor)	Set destructor (public member function)
operator=	Copy container content (public member function)

Iterators:

begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)

Capacity:

empty	Test whether container is empty (public member function)
size	Return container size (public member function)
max_size	Return maximum size (public member function)

14

STL: Set Member Functions

Modifiers:

insert	Insert element (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)

Operations:

find	Get iterator to element (public member function)
count	Count elements with a specific key (public member function)
lower_bound	Return iterator to lower bound (public member function)
upper_bound	Return iterator to upper bound (public member function)

15

STL: Set member Functions

```
// set1.cpp
// STL: Set Member Functions
// begin(), insert(), end()
#include <iostream>
#include <set>
using namespace std;

int main ()
{
  set<int> myset;
  set<int>::iterator it;
  int myint;

  cout << "Please enter some integers (enter 0 to end):\n";
  // each element is inserted as a sorted order
  do {
    cin >> myint;
    myset.insert (myint);
  } while (myint);

   for (it=myset.begin(); it!=myset.end(); ++it)
     cout << " " << *it;
  cout << endl;

  return 0;
}
```

16

```
// set2.cpp
// STL: set member functions
// begin(), end(), insert(), erase(), find()
#include <iostream>
#include <set>
using namespace std;

int main ()
{
  set<int> myset;
  set<int>::iterator it;

  // insert some values:
  for (int i=1; i<10; i++)
          myset.insert(i*10);  // 10 20 30 40 50 60 70 80 90
  it=myset.begin();
  it++;         // "it" points now to 20
  myset.erase (it);
  for (it=myset.begin(); it!=myset.end(); ++it)
          cout << " " << *it;
  cout << endl;
  myset.erase (40);  // erase 40
  for (it=myset.begin(); it!=myset.end(); ++it)
          cout << " " << *it;
  cout << endl;
  it=myset.find (60);
  myset.erase ( it, myset.end() );  //erase between 60 to the end
  for (it=myset.begin(); it!=myset.end(); ++it)
          cout << " " << *it;
  cout << endl;

  return 0;
}
```


17

```
// set21.cpp
// STL: Set Member Functions
// begin(), insert(), end(), erase
#include <iostream>
#include <set>
using namespace std;

int main ()
{
        set<int> myset;
        set<int>::iterator it;
        int myint;

        cout << "Please enter some integers (enter 0 to end):\n";
        // each element is inserted as a sorted order
        do {
                cin >> myint;
                myset.insert (myint);
         } while (myint);
        //print
        for (it=myset.begin(); it!=myset.end(); ++it)
                cout << " " << *it;
        cout << endl;

        it=myset.begin();
        it++;
        it++; // now it point to third element
        myset.erase ( it, myset.end() ); //remove element between third to last

        for (it=myset.begin(); it!=myset.end(); ++it)
                cout << " " << *it;
        cout << endl;
  return 0;
}
```

18

```cpp
// set3.cpp
// STL: set member functions
// constructor begin(), end(), swap()
#include <iostream>
#include <set>
using namespace std;

int main ()
{
  int myints[]={12,75,10,32,20,25};

  set<int> first (myints,myints+3);     // 10,12,75
  set<int> second (myints+3,myints+6);  // 20,25,32
  set<int>::iterator it;

  cout <<"Before Swap"<<endl;
  cout << "first contains: ";
  for (it=first.begin(); it!=first.end(); ++it)
          cout << " " << *it;
  cout << endl;

  cout << "second contains:";
  for (it=second.begin(); it!=second.end(); ++it)
          cout << " " << *it;
  cout << endl;

  first.swap(second);  // swap first and second set
  cout <<"After Swap"<<endl;
  cout << "first contains: ";
  for (it=first.begin(); it!=first.end(); ++it)
          cout << " " << *it;
  cout << endl;

  cout << "second contains:";
  for (it=second.begin(); it!=second.end(); ++it)
          cout << " " << *it;
  cout << endl;

  return 0;
}
```

19

```cpp
// set4.cpp
// STL set member function: count()
// Searches the container for an element with a key of x and
// returns the number of times the element appears in the container
// Because set containers do not allow for duplicate keys, this means that
// the function actually returns 1 if the element is found, and zero otherwise.
#include <iostream>
#include <set>
using namespace std;

int main ()
{
  set<int> myset;
  int i;

  // set some initial values:
  for (i=1; i<5; i++)
      myset.insert(i*3);     // set: 3 6 9 12

  for (i=0;i<10; i++)
  {
    cout << i;
    if (myset.count(i)>0)
      cout << " is an element of myset.\n";
    else
      cout << " is not an element of myset.\n";
  }
  return 0;
}
```


20

```cpp
// set5.cpp
// STL: set member functions lower_bound/upper_bound
// Returns an iterator pointing to the first element in the container
// which does not compare less than x or greater than x
#include <iostream>
#include <set>
using namespace std;
int main ()
{
  set<int> myset;
  set<int>::iterator itlow,itup, it;

  for (int i=1; i<10; i++)
      myset.insert(i*10); // 10 20 30 40 50 60 70 80 90
  cout << " initial myset contains: ";
  for (it=myset.begin(); it!=myset.end(); ++it)
          cout << " " << *it;
  cout << endl;

  itlow=myset.lower_bound (30);        // set iterator itlow to 30 as lower bound
  itup=myset.upper_bound (60);         // set iterator itup to 60 as upper bound
 // remove the block between low and up become 10 20 70 80 90
  myset.erase(itlow,itup);
  cout << " Now myset contains: ";
  for (it=myset.begin(); it!=myset.end(); ++it)
          cout << " " << *it;
  cout << endl;

  return 0;
}
```


21