C++ 11 Standard Template Library

This documentation is for reference purpose only and is for those who have attended the classroom sessions at Thinking Machines.

- **During your classroom session appropriate theory needs to be written against each example.**
- **You are required to bring this book daily for your classroom sessions.**
- **Some examples won't compile. They have been written to explain some rules.**
- **If you try to understand the examples without attending theory sessions then may god help you.**

Note : To compile all examples use -std=c++11 option as shown below

g++ SourceCodeFileName.Exetension -std=c++11 -o ExecutableFileName.Extension

The string class

string1.cpp

```
#include<iostream>
using namespace std;
int main()
{
string g;
g="Good";
cout<<g<<endl;
string m;
m="Boys";
string k=g+" "+m;
cout<<m<<endl;
cout<<k<<endl;
cout<<(g==k)<<endl;
cout<<(g!=k)<<endl;
cout<<(g<k)<<endl;
string z[10];
for(int i=0;i<=9;i++)
{
cout<<"Enter a string (without spaces) ";
cin>>z[i];
}
for(int e=0;e<=8;e++)
{
for(int f=e+1;f<=9;f++)
{
if(z[f]<z[e])
{
g=z[e];
z[e]=z[f];
z[f]=g;
}
}
}
cout<<"Sorted list"<<endl;
for(int e=0;e<=9;e++)
{
cout<<z[e]<<endl;
}
return 0;
```

```
}
```

string2.cpp

```
#include<iostream>
using namespace std;
int main()
{
string g;
g="God is great";
cout<<"First character : "<<g.front()<<endl;
cout<<"Last character : "<<g.back()<<endl;
cout<<"By position : "<<g.at(4)<<endl;
g+="!!";
cout<<g<<endl;
g.append(" Ujjai");
cout<<g<<endl;
g.push_back('n');
cout<<g<<endl;
g.append(10,'A');
cout<<g<<endl;
g.assign("Ujjain is the city of gods");
cout<<g<<endl;
g.assign("God is great",6);
cout<<g<<endl;
g.assign(10,'A');
cout<<g<<endl;
g="I live in Ujjain";
cout<<g<<endl;
string i;
i.assign(g,2,4);
cout<<i<<endl;
g.insert(0,"God is great.");
cout<<g<<endl;
string m="Cool & ";
g.insert(7,m);
cout<<g<<endl;
m="Sameer is a person";
m.insert(12,g,7,5);
cout<<m<<endl;
m.insert(16,"Fool",3);
cout<<m<<endl;
m="Mohan is a bad person";
m.insert(5,10,'-');
cout<<m<<endl;
m.pop_back();
m.pop_back();
m.pop_back();
cout<<m<<endl;
m.erase();
```

```
cout<<m<<endl;
cout<<m.empty()<<endl;
m="Ujjain is the city of gods";
m.erase(0,10);
cout<<m<<endl;
m.erase(8);
cout<<m<<endl;
m="I live in Ahmedabad.Ahmedabad is the city of gods";
string f="Ahmedabad";
string k="Ujjain";
m.replace(10,f.length(),k);
cout<<m<<endl;
m.replace(17,f.length(),k);
cout<<m<<endl;
m.replace(17,6,10,'_');
cout<<m<<endl;
g="Cool";
m.swap(g);
cout<<m<<endl;
cout<<g<<endl;
return 0;
}
```

---

string3.cpp

```
#include<string.h>
#include<iostream>
using namespace std;
int main()
{
string g;
g="God is great";
char a[101];
strcpy(a,g.c_str());
cout<<a<<endl;
g="I live in Ujjain. Ujjain is the city of GODS";
cout<<g<<endl;
cout<<g.length()<<","<<g.size()<<endl;
g.clear();
cout<<g.length()<<","<<g.size()<<","<<g.empty()<<endl;
g="I live in Indore, Indore is the city of GODS";
string search="Indore";
size_t x1;
x1=g.find(search);
cout<<search<<" from index 0 found at index "<<x1<<endl;
size_t x2;
x2=g.find(search,x1+1);
cout<<search<<" from index "<<x1+1<<" found at index "<<x2<<endl;
```

```
size_t x3;
x3=g.find(search,x2+1);
cout<<search<<" from index "<<x2+1<<" found at index "<<x3<<endl;
cout<<"size_t in unsigned int and largest possible value is 2^32-1 which is "<<string::npos<<endl;
x1=g.find('I');
cout<<"I from index 0 found at index "<<x1<<endl;
x2=g.find('I',x1+1);
cout<<"I from index "<<x1+1<<" found at index "<<x2<<endl;
x3=g.find('U',x2+1);
cout<<"I from index "<<x2+1<<" found at index "<<x3<<endl;
g="I live in Udaipur, Udaipur is the city of lakes";
search="Udaipur";
string replaceWith="Ujjain";
x1=0;
while((x1=g.find(search,x1))!=string::npos)
{
g.replace(x1,search.length(),replaceWith);
}
search="lakes";
replaceWith="GODS";
x1=g.find("lakes");
g.replace(x1,search.length(),replaceWith);
cout<<g<<endl;
x1=g.rfind("Ujjain");
cout<<x1<<endl;
x2=g.rfind("Ujjain",x1-1);
cout<<x2<<endl;
x3=g.rfind("Ujjain",x2-1);
cout<<x3<<endl;
cout<<g<<endl;
cout<<g.find_first_of("UAE")<<endl;
cout<<g.find_first_of("AUE",11)<<endl;
cout<<g.find_last_of("UAE")<<endl;
cout<<g.find_last_of("AUE",17)<<endl;
cout<<g.find_first_not_of("AEIOU")<<endl;
cout<<g.find_first_not_of("AEIOUaeiou",3)<<endl;
cout<<g.find_last_not_of("AEIOUaeiou",40)<<endl;
g="God is great";
string m;
m=g.substr(4,2);
cout<<m<<endl;
m=g.substr(4);
cout<<m<<endl;
g="Good Bad Ugly";
m="Bad";
cout<<g.compare(m)<<endl;
cout<<m.compare(g)<<endl;
```

```
cout<<m.compare(string("Bad"))<<endl;
cout<<g.compare(5,m.length(),m)<<endl;
return 0;
}
```

The pair class

```
/*
henceforth, to compile use -std=c++11 option along with -o as taught in the classroom session
*/
#include<utility>
#include<iostream>
using namespace std;
int main()
{
pair<int,int> p1;
cout<<"First : "<<p1.first<<", Second : "<<p1.second<<endl;
p1.first=10;
p1.second=20;
cout<<"First : "<<p1.first<<", Second : "<<p1.second<<endl;
pair<int,int> p2;
p2=make_pair(1000,2000);
cout<<"First : "<<p1.first<<", Second : "<<p1.second<<endl;
cout<<"First : "<<p2.first<<", Second : "<<p2.second<<endl;
cout<<"Swapping"<<endl;
p1.swap(p2);
cout<<"First : "<<p1.first<<", Second : "<<p1.second<<endl;
cout<<"First : "<<p2.first<<", Second : "<<p2.second<<endl;
cout<<"Swapping again"<<endl;
swap(p1,p2);
cout<<"First : "<<p1.first<<", Second : "<<p1.second<<endl;
cout<<"First : "<<p2.first<<", Second : "<<p2.second<<endl;
pair<int,char> p3(10,'Z');
cout<<"First : "<<p3.first<<", Second : "<<p3.second<<endl;
pair<int,char> p4=p3;
cout<<"First : "<<p4.first<<", Second : "<<p4.second<<endl;
pair<int,char> p5;
p5=p4;
cout<<"First : "<<p5.first<<", Second : "<<p5.second<<endl;
cout<<(p4==p5)<<endl;
cout<<(p4!=p5)<<endl;
pair<int,char> p6(5,'Z');
cout<<(p6<p5)<<endl;
pair<int,char> p7(10,'Z');
cout<<(p7<p5)<<endl;
pair<int,char> p8(10,'Y');
cout<<(p8<p5)<<endl;
cout<<(get<0>(p8))<<endl;
```

```
cout<<(get<1>(p8))<<endl;
// Assignment : implement operator overloading to make cout<<p7; possible
return 0;
}
```

Student.h

```
#include<string.h>
class Student
{
private:
int rollNumber;
char *name;
void release()
{
if(this->name!=NULL) delete [] this->name;
}
void set(int rollNumber,const char *name)
{
this->rollNumber=rollNumber;
this->name=NULL;
if(name==NULL)
{
this->name=new char[1];
this->name[0]='\0';
}
else
{
this->name=new char[strlen(name)+1];
strcpy(this->name,name);
}
}
public:
Student()
{
this->rollNumber=0;
this->name=NULL;
}
Student(int rollNumber,const char *name)
{
this->set(rollNumber,name);
}
Student(const Student &other)
{
this->set(other.rollNumber,other.name);
}
Student & operator=(Student other)
{
```

```
release();
this->set(other.rollNumber,other.name);
}
virtual ~Student()
{
release();
}
int getRollNumber()
{
return this->rollNumber;
}
const char * getName()
{
return this->name;
}
};
```

tststudent.cpp

```
#include<iostream>
using namespace std;
#include "Student.h"
int main()
{
Student s1(101,"Suresh");
cout<<s1.getRollNumber()<<s1.getName()<<endl;
return 0;
}
```

City.h

```
#include<string.h>
#include<iostream>
using namespace std;
class City
{
private:
int code;
string name;
public:
City()
{
this->code=0;
}
City(int code,string name)
{
this->code=code;
this->name=name;
}
City(const City &other)
```

```
{
this->code=other.code;
this->name=other.name;
}
City & operator=(City other)
{
this->code=other.code;
this->name=other.name;
}
virtual ~City()
{
}
int getCode()
{
return this->code;
}
string getName()
{
return this->name;
}
};
```

tstcity.cpp

```
#include<iostream>
using namespace std;
#include "City.h"
int main()
{
City c1(101,"Ujjjain");
cout<<c1.getCode()<<","<<c1.getName()<<endl;
return 0;
}
```

pair2.cpp

```
#include<utility>
#include<iostream>
using namespace std;
#include "Student.h"
#include "City.h"
int main()
{
pair<Student,City> p1(Student(101,"Suresh"),City(5001,"Pune"));
Student s=p1.first;
City c=p1.second;
cout<<"Student : Roll number -> "<<s.getRollNumber()<<", Name -> "<<s.getName()<<endl;
cout<<"belongs to City  :  Code -> "<<c.getCode()<<", Name -> "<<c.getName()<<endl;
return 0;
}
```

Iterators

string4.cpp

```
#include<iostream>
using namespace std;
int main()
{
string g;
g="God is great";
cout<<"Iterating string"<<endl;
cout<<"Technique 1"<<endl;
for(int e=0;e<g.length();e++)
{
cout<<g[e]<<endl;
}
cout<<"Technique 2"<<endl;
string::iterator i=g.begin();
while(i!=g.end())
{
cout<<*i<<endl;
++i;
}
cout<<"Technique 3"<<endl;
string::reverse_iterator k=g.rbegin();
while(k!=g.rend())
{
cout<<*k<<endl;
++k;
}
return 0;
}
```

Collection classes

The vector class

A vector manages its elements in a dynamic array. It enables random access, which means that you can access each element directly with the corresponding index. Appending and removing elements at the end of the array is very fast.3 However, inserting an element in the middle or at the beginning of the array takes time because all the following elements have to be moved to make room for it while maintaining the order.

vector1.cpp

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

```
{
vector<int> v;
v.push_back(10);
v.push_back(30);
v.push_back(20);
v.push_back(40);
v[8]=500;
for(int i=0;i<v.size();++i)
{
cout<<v[i]<<endl;
}
cout<<"Size : "<<v.size()<<",Capacity : "<<v.capacity()<<",Maximum size : "<<v.max_size()<<endl;
v.resize(10);
cout<<"After resizing "<<endl;
cout<<"Size : "<<v.size()<<",Capacity : "<<v.capacity()<<",Maximum size : "<<v.max_size()<<endl;
v[8]=203;
for(int i=0;i<v.size();++i)
{
cout<<v[i]<<endl;
}
cout<<"Iterating front to end"<<endl;
vector<int>::iterator fi=v.begin();
while(fi!=v.end())
{
cout<<*fi<<endl;
++fi;
}
cout<<"Iterating end to front"<<endl;
vector<int>::reverse_iterator ri=v.rbegin();
while(ri!=v.rend())
{
cout<<*ri<<endl;
*ri=*ri+10000;
++ri;
}
cout<<"After modifying through iterator"<<endl;
for(int i=0;i<v.size();++i)
{
cout<<v[i]<<endl;
}
v.clear();
v.push_back(10);
v.push_back(12);
v.push_back(15);
v.push_back(13);
cout<<"After clearing the vector and adding 4 elements to it"<<endl;
for(int i=0;i<v.size();++i) cout<<v[i]<<endl;
```

```
cout<<"Creating a constant iterator (Through which value cannot be modified)"<<endl;
vector<int>::const_iterator cfi=v.cbegin();
while(cfi!=v.cend())
{
cout<<*cfi<<endl;
//   *cfi=40;  // if uncommented, this code won't compile as it is a const_iterator
++cfi;
}
// try const_reverse_iterator along with crbegin and crend
return 0;
}
```

vector2.cpp

```
#include<vector>
#include<iostream>
using namespace std;
int main()
{
vector<int> v(10);
v[0]=20;
v[1]=22;
v[2]=33;
v[3]=44;
for(int e=0;e<v.size();++e)
{
cout<<v[e]<<endl;
}
cout<<"Size : "<<v.size()<<", Capacity : "<<v.capacity()<<endl;
v.shrink_to_fit();
cout<<"After shrinking"<<endl;
cout<<"Size : "<<v.size()<<", Capacity : "<<v.capacity()<<endl;
for(int e=0;e<v.size();++e)
{
cout<<v[e]<<endl;
}
v.resize(3);
cout<<"After resizing"<<endl;
cout<<"Size : "<<v.size()<<", Capacity : "<<v.capacity()<<endl;
for(int e=0;e<v.size();++e)
{
cout<<v[e]<<endl;
}
return 0;
}
```

vector3.cpp

```
#include<vector>
#include<iostream>
using namespace std;
int main()
{
vector<int> v;
v.push_back(220);
v.push_back(221);
v.push_back(222);
v.push_back(223);
v.push_back(224);
for(int i=0;i<v.size();++i) cout<<v[i]<<endl;
v.pop_back();
cout<<"After popping from back"<<endl;
for(int i=0;i<v.size();++i) cout<<v[i]<<endl;
cout<<"Creating a vector of size 3, populated with 100 as value of each element"<<endl;
vector<int> vv(3,200);
for(int i=0;i<vv.size();++i) cout<<vv[i]<<endl;
vv[0]=100;
vv[2]=300;
cout<<"After changing data"<<endl;
for(int i=0;i<vv.size();++i) cout<<vv[i]<<endl;
cout<<"Data at front : "<<vv.front()<<endl;
cout<<"Data at back : "<<vv.back()<<endl;
vv.push_back(400);
vv.push_back(500);
vv.push_back(600);
vv.push_back(700);
vv.push_back(800);
cout<<"After appending more data"<<endl;
for(int i=0;i<vv.size();++i) cout<<vv[i]<<endl;
vv.erase(vv.begin()+2);
cout<<"After erasing 3rd element"<<endl;
for(int i=0;i<vv.size();++i) cout<<vv[i]<<endl;
vv.erase(vv.begin()+1,vv.begin()+5);
// Note begin()+1  for 2nd and begin()+4 for 5th,
// but 5th won't be removed
cout<<"After erasing 2nd, 3rd and 4th element"<<endl;
for(int i=0;i<vv.size();++i) cout<<vv[i]<<endl;
vv.insert(vv.begin()+2,3000); // insert at 3rd position
cout<<"After inserting 3000 at 3rd position"<<endl;
for(int i=0;i<vv.size();++i) cout<<vv[i]<<endl;
vv.insert(vv.begin()+2,5,6000); // insert at 3rd position
cout<<"After inserting 6000, 5 times from 3rd position"<<endl;
for(int i=0;i<vv.size();++i) cout<<vv[i]<<endl;
vector<int> k(3,300);
```

```
cout<<"Contents of vector k"<<endl;
for(int i=0;i<k.size();++i) cout<<k[i]<<endl;
k.swap(vv);
cout<<"After swapping k with vv"<<endl;
cout<<"Contents of vector k"<<endl;
for(int i=0;i<k.size();++i) cout<<k[i]<<endl;
cout<<"Contents of vector vv"<<endl;
for(int i=0;i<vv.size();++i) cout<<vv[i]<<endl;
return 0;
}
```

The deque class

Double ended queue, same as vector but elements can be added in front as well as back

deque1.cpp

```
#include<deque>
#include<iostream>
using namespace std;
int main()
{
deque<int> q;
q.push_back(10);
q.push_back(20);
q.push_back(30);
q.push_front(2);
q.push_front(4);
q.push_front(5);
for(int e=0;e<q.size();++e)
{
cout<<q[e]<<endl;
}
q.pop_front();
q.pop_back();
cout<<"After popping elements from front and back"<<endl;
for(int e=0;e<q.size();++e)
{
cout<<q[e]<<endl;
}
//Assignment : try other function as done in the examples of vector
return 0;
}
```

The array class

Fixed size. Whatever we can do in case of int x[10] can be done using the array container along with the additional benefit of functionality applicable on array class object.

array1.cpp

```
#include<array>
#include<iostream>
using namespace std;
int main()
{
array<int,5> a1;
cout<<"Default data"<<endl;
for(int e=0;e<a1.size();e++) cout<<a1[e]<<endl;
a1[0]=10;
a1[1]=20;
a1[2]=30;
a1[3]=40;
a1[4]=50;
cout<<"Without iterator"<<endl;
for(int i=0;i<a1.size();++i) cout<<a1[i]<<endl;
cout<<"With iterator"<<endl;
array<int,5>::iterator fi=a1.begin();
while(fi!=a1.end())
{
cout<<*fi<<endl;
++fi;
}
cout<<"The new range based loop"<<endl;
for(int x:a1)
{
cout<<x<<endl;
}
cout<<"More fun introduced in C++"<<endl;
for(auto x:a1)
{
cout<<x<<endl;
}
// Assignment : try other functions as discussed in the classroom session
return 0;
}
```

The list class

Internally a doubly linked list is maintained. We cannot access an element directly, we need to traverse to that element. From an element, we can move forward or backward. Since the internal implementation is not a dynamic array, insertions and remove are fast as only the links have to be managed.

list1.cpp

```
#include "student.h"
#include<list>
#include<iostream>
using namespace std;
int main()
{
Student s1(101,"Sameer");
Student s2(102,"Mohit");
Student s3(103,"Rakesh");
Student s4(104,"Reena");
Student s5(105,"Tina");
Student s6(106,"Kamal");
list<Student *> students;
students.push_back(&s2);
students.push_back(&s4);
students.push_back(&s5);
students.push_back(&s6);
students.push_front(&s1);
// logic to insert an element at index 2
list<Student *>::iterator fi;
int index=2;
int m;
fi=students.begin();
m=1;
while(m<=index)
{
++fi;
m++;
}
students.insert(fi,&s3);
for(fi=students.begin();fi!=students.end();++fi)
{
cout<<(*fi)->getRollNumber()<<","<<(*fi)->getName()<<endl;
}
return 0;
}
```

list2.cpp

```
#include<list>
#include<iostream>
using namespace std;
int main()
{
list<int> aList;
aList.push_back(10);
aList.push_back(20);
aList.push_back(30);
aList.push_back(40);
aList.push_back(50);
aList.push_back(60);
aList.push_back(40);
aList.push_back(70);
aList.push_back(30);
aList.push_back(80);
for(list<int>::iterator fi=aList.begin();fi!=aList.end();++fi)
{
cout<<*fi<<endl;
}
aList.sort();
cout<<"After calling sort function"<<endl;
for(list<int>::iterator fi=aList.begin();fi!=aList.end();++fi)
{
cout<<*fi<<endl;
}
aList.unique();
cout<<"After calling unique function"<<endl;
for(list<int>::iterator fi=aList.begin();fi!=aList.end();++fi)
{
cout<<*fi<<endl;
}
// Assignment, try calling unique without calling sort function
// and see how list behaves
return 0;
}
```

list3.cpp

```
#include<iostream>
#include<list>
using namespace std;
int main()
{
list<int> aList;
aList.push_back(10);
aList.push_back(20);
aList.push_back(30);
aList.push_back(40);
aList.push_back(50);
aList.push_back(60);
for(list<int>::iterator fi=aList.begin();fi!=aList.end();++fi) cout<<*fi<<endl;
list<int>::iterator ei=aList.begin();
++ei;
++ei;
cout<<"Calling erase with one iterator argument pointing to "<<*ei<<endl;
aList.erase(ei);
for(list<int>::iterator fi=aList.begin();fi!=aList.end();++fi) cout<<*fi<<endl;
aList.push_back(70);
aList.push_back(80);
aList.push_back(90);
cout<<"After adding some more elements"<<endl;
for(list<int>::iterator fi=aList.begin();fi!=aList.end();++fi) cout<<*fi<<endl;
list<int>::iterator ei1=aList.begin();
++ei1;
++ei1;
list<int>::iterator ei2=ei1;
++ei2;
++ei2;
cout<<"Calling erase with two iterator arguments pointing to "<<*ei1<<" and "<<*ei2<<endl;
aList.erase(ei1,ei2);
for(list<int>::iterator fi=aList.begin();fi!=aList.end();++fi) cout<<*fi<<endl;
return 0;
}
```

---

list4.cpp

```
#include<iostream>
#include<list>
using namespace std;
int isToBeRemoved(int);
int main()
{
list<int> aList;
aList.push_back(10);
aList.push_back(20);
```

```
aList.push_back(30);
aList.push_back(40);
aList.push_back(30);
aList.push_back(50);
aList.push_back(30);
aList.push_back(60);
for(list<int>::iterator fi=aList.begin();fi!=aList.end();++fi) cout<<*fi<<endl;
aList.remove(30);
cout<<"After removing value 30 "<<endl;
for(list<int>::iterator fi=aList.begin();fi!=aList.end();++fi) cout<<*fi<<endl;
aList.push_back(70);
aList.push_back(80);
aList.push_back(90);
aList.push_back(100);
cout<<"After appending  value 70,80,90 and 100 "<<endl;
for(list<int>::iterator fi=aList.begin();fi!=aList.end();++fi) cout<<*fi<<endl;
aList.remove_if(isToBeRemoved);
cout<<"After removing with the help of a predicate function "<<endl;
for(list<int>::iterator fi=aList.begin();fi!=aList.end();++fi) cout<<*fi<<endl;
return 0;
}

int isToBeRemoved(int data)
{
return data>=50 && data<=70;
}
```

---

list5.cpp

```
#include<iostream>
#include<list>
#include<string.h>
#include "City.h"
using namespace std;
bool cityComparator(City *,City *);
int main()
{
list<int> aList;
aList.push_back(10);
aList.push_back(20);
aList.push_back(30);
aList.push_back(40);
aList.push_back(50);
aList.push_back(60);
for(list<int>::iterator fi=aList.begin();fi!=aList.end();++fi) cout<<*fi<<endl;
aList.reverse();
cout<<"After reversing the contents of the list "<<endl;
for(list<int>::iterator fi=aList.begin();fi!=aList.end();++fi) cout<<*fi<<endl;
```

```
City c1(101,"Pune");
City c2(102,"Goa");
City c3(103,"Ujjain");
City c4(104,"ahmedabad");
City c5(105,"Mumbai");
City c6(106,"Dewas");
City c7(107,"indore");
list<City *> cities;
cities.push_back(&c1);
cities.push_back(&c2);
cities.push_back(&c3);
cities.push_back(&c4);
cities.push_back(&c5);
cities.push_back(&c6);
cities.push_back(&c7);
cout<<"After adding some citites"<<endl;
for(list<City *>::iterator fi=cities.begin();fi!=cities.end();++fi)
{
cout<<"Code : "<<(*fi)->getCode()<<", City : "<<(*fi)->getName()<<endl;
}
cities.sort(cityComparator);
cout<<"After sorting citites"<<endl;
for(list<City *>::iterator fi=cities.begin();fi!=cities.end();++fi)
{
cout<<"Code : "<<(*fi)->getCode()<<", City : "<<(*fi)->getName()<<endl;
}
return 0;
}
bool cityComparator(City *left,City *right)
{
int x=stricmp(left->getName().c_str(),right->getName().c_str());
if(x<0) return true;
return false;
}
```

---

list6.cpp

```
#include<list>
#include<iostream>
using namespace std;
int main()
{
list<int> firstList;
firstList.push_back(10);
firstList.push_back(20);
firstList.push_back(30);
firstList.push_back(40);
firstList.push_back(50);
```

```
firstList.push_back(60);
firstList.push_back(70);

list<int> secondList;
secondList.push_back(22);
secondList.push_back(23);
secondList.push_back(24);
cout<<"First list"<<endl;
for(list<int>::iterator fi=firstList.begin();fi!=firstList.end();++fi) cout<<*fi<<endl;
cout<<"Second list"<<endl;
for(list<int>::iterator fi=secondList.begin();fi!=secondList.end();++fi) cout<<*fi<<endl;
list<int>::iterator ii=firstList.begin();
++ii;
++ii;
cout<<"Calling the splice function for firstList using iterator pointing to "<<*ii<<endl;
firstList.splice(ii,secondList);
for(list<int>::iterator fi=secondList.begin();fi!=secondList.end();++fi) cout<<*fi<<endl;
cout<<"First list"<<endl;
for(list<int>::iterator fi=firstList.begin();fi!=firstList.end();++fi) cout<<*fi<<endl;
cout<<"Second list"<<endl;
for(list<int>::iterator fi=secondList.begin();fi!=secondList.end();++fi) cout<<*fi<<endl;
return 0;
}
```

---

list7.cpp

```
#include<list>
#include<iostream>
using namespace std;
int main()
{
list<int> firstList;
firstList.push_back(10);
firstList.push_back(20);
firstList.push_back(30);
firstList.push_back(40);
firstList.push_back(50);
firstList.push_back(60);
firstList.push_back(70);
list<int> secondList;
secondList.push_back(22);
secondList.push_back(23);
secondList.push_back(24);
secondList.push_back(25);
secondList.push_back(26);
secondList.push_back(27);
secondList.push_back(28);
cout<<"First list"<<endl;
```

```
for(list<int>::iterator fi=firstList.begin();fi!=firstList.end();++fi) cout<<*fi<<endl;
cout<<"Second list"<<endl;
for(list<int>::iterator fi=secondList.begin();fi!=secondList.end();++fi) cout<<*fi<<endl;
list<int>::iterator ii=firstList.begin();
++ii;
++ii;
list<int>::iterator si=secondList.begin();
++si;
++si;
++si;
cout<<"Calling the splice function for firstList "<<endl;
cout<<"firstList iterator pointing to "<<*ii<<endl;
cout<<"secondList iterator pointing to "<<*si<<endl;
firstList.splice(ii,secondList,si);
for(list<int>::iterator fi=secondList.begin();fi!=secondList.end();++fi) cout<<*fi<<endl;
cout<<"First list"<<endl;
for(list<int>::iterator fi=firstList.begin();fi!=firstList.end();++fi) cout<<*fi<<endl;
cout<<"Second list"<<endl;
for(list<int>::iterator fi=secondList.begin();fi!=secondList.end();++fi) cout<<*fi<<endl;
return 0;
}
```

---

list8.cpp

```
#include<list>
#include<iostream>
using namespace std;
int main()
{
list<int> firstList;
firstList.push_back(10);
firstList.push_back(20);
firstList.push_back(30);
firstList.push_back(40);
firstList.push_back(50);
firstList.push_back(60);
firstList.push_back(70);
list<int> secondList;
secondList.push_back(22);
secondList.push_back(23);
secondList.push_back(24);
secondList.push_back(25);
secondList.push_back(26);
secondList.push_back(27);
secondList.push_back(28);
cout<<"First list"<<endl;
for(list<int>::iterator fi=firstList.begin();fi!=firstList.end();++fi) cout<<*fi<<endl;
cout<<"Second list"<<endl;
```

```
for(list<int>::iterator fi=secondList.begin();fi!=secondList.end();++fi) cout<<*fi<<endl;
list<int>::iterator ii=firstList.begin();
++ii;
++ii;
list<int>::iterator si1=secondList.begin();
++si1;
++si1;
list<int>::iterator si2=si1;
++si2;
++si2;
++si2;
cout<<"Calling the splice function for firstList "<<endl;
cout<<"firstList iterator pointing to "<<*ii<<endl;
cout<<"2 secondList iterators pointing to "<<*si1<<" and "<<*si2<<endl;
firstList.splice(ii,secondList,si1,si2);
cout<<"First list"<<endl;
for(list<int>::iterator fi=firstList.begin();fi!=firstList.end();++fi) cout<<*fi<<endl;
cout<<"Second list"<<endl;
for(list<int>::iterator fi=secondList.begin();fi!=secondList.end();++fi) cout<<*fi<<endl;
return 0;
}
```

The forward_list class

Forward lists are sequence containers that allow constant time insert and erase operations anywhere within the sequence.

Forward lists are implemented as singly-linked lists; Singly linked lists can store each of the elements they contain in different and unrelated storage locations. The ordering is kept by the association to each element of a link to the next element in the sequence.
The main design difference between a forward_list container and a list container is that the first keeps internally only a link to the next element, while the latter keeps two links per element: one pointing to the next element and one to the preceding one, allowing efficient iteration in both directions, but consuming additional storage per element and with a slight higher time overhead inserting and removing elements. forward_list objects are thus more efficient than list objects, although they can only be iterated forwards.

Compared to other base standard sequence containers (array, vector and deque), forward_list perform generally better in inserting, extracting and moving elements in any position within the container, and therefore also in algorithms that make intensive use of these, like sorting algorithms.

The main drawback of forward_lists and lists compared to these other sequence containers is that they lack direct access to the elements by their position; For example, to access the sixth element in a forward_list one has to iterate from the beginning to that position, which takes linear time in the distance between these. They also consume some extra memory to keep the linking information associated to each element (which may be an important factor for large lists of small-sized elements).

The forward_list class template has been designed with efficiency in mind: By design, it is as efficient as a simple handwritten C-style singly-linked list, and in fact is the only standard container to deliberately lack a size member function for efficiency considerations: due to its nature as a linked list, having a size member that takes constant time would require it to keep an internal counter for its size (as list does). This would consume some extra storage and make insertion and removal operations slightly less efficient. To obtain the size of a forward_list object, you can use the distance algorithm with its begin and end, which is an operation that takes linear time. And the good part is that the iterator doesn't get invalidated if the list grows while iterating.

The functionality behind the following functions is same as we discussed earlier, so I won't provide examples with those.

begin, end, cbegin, cend, empty, front, push_front, pop_front, clear, remove, remove_if, unique, sort and reverse

Note : it doesn't have push_back, pop_back and back functions applicable to it.

I am just providing examples with some functionality that is not part of the earlier discussed classes.

forward1.cpp

```
#include<forward_list>
#include<iostream>
using namespace std;
bool isNegative(int);
int main()
{
forward_list<int> aList;
forward_list<int>::iterator fi;
fi=aList.before_begin();
for(int x:aList) ++fi;
aList.insert_after(fi,10);
++fi;
aList.insert_after(fi,20);
++fi;
aList.insert_after(fi,30);
++fi;
aList.insert_after(fi,40);
++fi;
aList.insert_after(fi,50);
cout<<"After inserting some elements at end"<<endl;
for(forward_list<int>::iterator fi=aList.begin();fi!=aList.end();++fi) cout<<*fi<<endl;
aList.insert_after(aList.before_begin(),5);
cout<<"After calling insert_after"<<endl;
for(forward_list<int>::iterator fi=aList.begin();fi!=aList.end();++fi) cout<<*fi<<endl;
fi=aList.before_begin();
for(auto x:aList) ++fi;
aList.insert_after(fi,1000);
++fi;
aList.insert_after(fi,2000);
++fi;
aList.insert_after(fi,3000);
cout<<"After appending more items "<<endl;
for(forward_list<int>::iterator fi=aList.begin();fi!=aList.end();++fi) cout<<*fi<<endl;
aList.push_front(-1);
aList.push_front(-2);
aList.push_front(-3);
cout<<"After pushing more items at front"<<endl;
for(forward_list<int>::iterator fi=aList.begin();fi!=aList.end();++fi) cout<<*fi<<endl;
fi=aList.begin();
++fi;
aList.insert_after(fi,30);
cout<<"After inserting more items in between at 3rd position"<<endl;
for(forward_list<int>::iterator fi=aList.begin();fi!=aList.end();++fi) cout<<*fi<<endl;
aList.remove(30);
cout<<"After removing 30"<<endl;
for(forward_list<int>::iterator fi=aList.begin();fi!=aList.end();++fi) cout<<*fi<<endl;
```

```
aList.remove_if(isNegative);
cout<<"After removing using a predicate"<<endl;
for(forward_list<int>::iterator fi=aList.begin();fi!=aList.end();++fi) cout<<*fi<<endl;
return 0;
}
bool isNegative(int x) { return x<0; }
```

forward2.cpp

```
#include<forward_list>
#include<iostream>
using namespace std;
int main()
{
forward_list<int> aList;
forward_list<int>::iterator fi;
fi=aList.before_begin();
aList.insert_after(fi,10);
++fi;
aList.insert_after(fi,20);
++fi;
aList.insert_after(fi,30);
++fi;
aList.insert_after(fi,40);
forward_list<int> bList;
fi=bList.before_begin();
bList.insert_after(fi,1000);
++fi;
bList.insert_after(fi,2000);
++fi;
bList.insert_after(fi,3000);
++fi;
bList.insert_after(fi,4000);
cout<<"Contents of aList"<<endl;
for(auto x:aList) cout<<x<<endl;
cout<<"Contents of bList"<<endl;
for(auto x:bList) cout<<x<<endl;
aList.merge(bList);
cout<<"Contents of aList after merging bList into it"<<endl;
for(auto x:aList) cout<<x<<endl;
cout<<"Contents of bList after merging it into aList"<<endl;
for(auto x:bList) cout<<x<<endl;
return 0;
}
```

forward3.cpp

```
#include<forward_list>
#include<iostream>
using namespace std;
int main()
{
forward_list<int> aList;
forward_list<int>::iterator fi;
fi=aList.before_begin();
aList.insert_after(fi,10);
++fi;
aList.insert_after(fi,4000);
++fi;
aList.insert_after(fi,30);
++fi;
aList.insert_after(fi,2000);
forward_list<int> bList;
fi=bList.before_begin();
bList.insert_after(fi,1000);
++fi;
bList.insert_after(fi,2000);
++fi;
bList.insert_after(fi,3000);
++fi;
bList.insert_after(fi,4000);
cout<<"Contents of aList"<<endl;
for(auto x:aList) cout<<x<<endl;
cout<<"Contents of bList"<<endl;
for(auto x:bList) cout<<x<<endl;
aList.merge(bList);
cout<<"Contents of aList after merging bList into it"<<endl;
for(auto x:aList) cout<<x<<endl;
cout<<"Contents of bList after merging it into aList"<<endl;
for(auto x:bList) cout<<x<<endl;
return 0;
}
```

Assignment : learn how to use splice_after method

The queue class

queue1.cpp

```
#include<queue>
#include<iostream>
using namespace std;
int main()
{
queue<int> aQueue;
aQueue.push(10);
aQueue.push(20);
aQueue.push(30);
aQueue.push(40);
aQueue.push(50);
while(!aQueue.empty())
{
cout<<aQueue.front()<<endl;
aQueue.pop();
}
return 0;
}
```

The priority_queue class

p_queue1.cpp

```
#include<iostream>
#include<queue>
using namespace std;
int main()
{
priority_queue<int> aQueue;
aQueue.push(25);
aQueue.push(20);
aQueue.push(10);
aQueue.push(40);
aQueue.push(50);
while(!aQueue.empty())
{
cout<<aQueue.top()<<endl;
aQueue.pop();
}
return 0;
}
```

---

p_queue2.cpp

```
#include "City.h"
#include<functional>
#include<iostream>
#include<queue>
using namespace std;
bool cityComparator(City *,City *);
int main()
{
City c1(101,"Ujjain");
City c2(102,"Agra");
City c3(103,"Panvel");
City c4(104,"Pune");
City c5(105,"Ahmedabad");
City c6(106,"Baroda");
City c7(107,"Bombay");
priority_queue<City *,vector<City *>,function<bool(City *,City *)>> aQueue(cityComparator);
aQueue.push(&c1);
aQueue.push(&c2);
aQueue.push(&c3);
aQueue.push(&c4);
aQueue.push(&c5);
aQueue.push(&c6);
aQueue.push(&c7);
City *c;
while(!aQueue.empty())
{
c=aQueue.top();
cout<<c->getCode()<<","<<c->getName()<<endl;
aQueue.pop();
}
return 0;
}
bool cityComparator(City *left,City *right)
{
return left->getName()<right->getName();
}
```

---

p_queue3.cpp

```
#include "City.h"
#include<functional>
#include<iostream>
#include<queue>
using namespace std;
class CityComparator
{
public:
```

```
int operator()(City *left,City *right) const
{
return left->getName()<right->getName();
}
};
int main()
{
City c1(101,"Ujjain");
City c2(102,"Agra");
City c3(103,"Panvel");
City c4(104,"Pune");
City c5(105,"Ahmedabad");
City c6(106,"Baroda");
City c7(107,"Bombay");
priority_queue<City *,vector<City *>,CityComparator> aQueue;
aQueue.push(&c1);
aQueue.push(&c2);
aQueue.push(&c3);
aQueue.push(&c4);
aQueue.push(&c5);
aQueue.push(&c6);
aQueue.push(&c7);
City *c;
while(!aQueue.empty())
{
c=aQueue.top();
cout<<c->getCode()<<","<<c->getName()<<endl;
aQueue.pop();
}
return 0;
}
```

The stack class

stack1.cpp

```
#include<iostream>
#include<stack>
using namespace std;
int main()
{
stack<int> aStack;
aStack.push(25);
aStack.push(20);
aStack.push(10);
aStack.push(40);
aStack.push(50);
while(!aStack.empty())
```

```
{
cout<<aStack.top()<<endl;
aStack.pop();
}
return 0;
}
```

The set class

Sets are containers that store unique elements following a specific order.

set1.cpp

```
#include<set>
#include<iostream>
using namespace std;
int main()
{
set<int> aSet;
aSet.insert(10);
aSet.insert(20);
aSet.insert(30);
aSet.insert(10);
aSet.insert(30);
aSet.insert(50);
aSet.insert(40);
aSet.insert(20);
cout<<"After inserting some elements in the set"<<endl;
for(auto x:aSet) cout<<x<<endl;
cout<<"Size : "<<aSet.size()<<endl;
cout<<"10 exists : "<<aSet.count(10)<<endl;
cout<<"92 exists : "<<aSet.count(92)<<endl;
cout<<"Iterating from 30"<<endl;
set<int>::iterator i=aSet.find(30);
while(i!=aSet.end())
{
cout<<*i<<endl;
++i;
}
aSet.erase(aSet.find(30));
cout<<"After removing 30"<<endl;
for(auto x:aSet) cout<<x<<endl;
aSet.erase(40);
cout<<"After removing 40"<<endl;
for(auto x:aSet) cout<<x<<endl;
aSet.erase(92);
cout<<"After removing 92"<<endl;
for(auto x:aSet) cout<<x<<endl;
return 0;
}
```

---

set2.cpp

```
#include<set>
#include<iostream>
#include "Student.h"
using namespace std;
class StudentComparator
{
public:
bool operator()(Student *left,Student *right)
{
cout<<"Got called left roll number "<<left->getRollNumber()<<", and right roll number "<<right-
>getRollNumber()<<endl;
return left->getRollNumber()<right->getRollNumber();

}
};
int main()
{
set<Student *,StudentComparator> students;
Student s1(101,"Sameer");
Student s2(102,"Lokesh");
Student s3(104,"Mahesh");
Student s4(103,"Gopal");
Student s5(102,"Ramu");
Student s6(105,"Mohan");
Student s7(104,"Aman");
students.insert(&s1);
cout<<"1------------------------"<<endl;
students.insert(&s2);
cout<<"2------------------------"<<endl;
students.insert(&s3);
cout<<"3------------------------"<<endl;
students.insert(&s4);
cout<<"4------------------------"<<endl;
students.insert(&s5);
cout<<"5------------------------"<<endl;
students.insert(&s6);
cout<<"6------------------------"<<endl;
students.insert(&s7);
cout<<"7------------------------"<<endl;
cout<<"After inserting students"<<endl;
for(auto s:students) cout<<s->getRollNumber()<<","<<s->getName()<<endl;
return 0;
}
```

The multiset class for Multiple-key set

Multisets are containers that store elements following a specific order, and where multiple elements can have equivalent values.

multiset1.cpp

```
#include<set>
#include<iostream>
using namespace std;
int main()
{
multiset<int> aSet;
aSet.insert(10);
aSet.insert(40);
aSet.insert(20);
aSet.insert(30);
aSet.insert(10);
aSet.insert(40);
aSet.insert(30);
aSet.insert(50);
aSet.insert(40);
aSet.insert(20);
cout<<"After inserting some elements in the set"<<endl;
for(auto x:aSet) cout<<x<<endl;
cout<<"Size : "<<aSet.size()<<endl;
aSet.erase(40);
cout<<"After removing 40"<<endl;
for(auto x:aSet) cout<<x<<endl;
return 0;
}
```

---

multiset2.cpp

```
#include<set>
#include<utility>
#include<iostream>
using namespace std;
class CourseComparator
{
public:
bool operator()(pair<string,string> *left,pair<string,string> *right)
{
return left->first<right->first;
}
};

int main()
{
multiset<pair<string,string> *,CourseComparator> aSet;
```

```
pair<string,string> c1("JAVA","Berlin");
pair<string,string> c2("PYTHON","Paris");
pair<string,string> c3("JAVA","London");
pair<string,string> c4("PYTHON","Barcelona");
pair<string,string> c5("ML","Prague");
pair<string,string> c6("PYTHON","Amsterdam");
pair<string,string> c7("ML","Madrid");
pair<string,string> c8("JAVA","Bombay");
aSet.insert(&c1);
aSet.insert(&c2);
aSet.insert(&c3);
aSet.insert(&c4);
aSet.insert(&c5);
aSet.insert(&c6);
aSet.insert(&c7);
aSet.insert(&c8);
for(auto p:aSet) cout<<"Course : "<<p->first<<", City : "<<p->second<<endl;
return 0;
}
```

The map class

map1.cpp

```
#include<iostream>
#include<map>
#include "Student.h"
#include<utility>
using namespace std;
int main()
{
map<int,Student *> students;
Student s1(101,"Sameer");
Student s2(102,"Rakesh");
Student s3(103,"Suresh");
Student s4(111,"Mahesh");
Student s5(104,"Ganesh");
Student s6(105,"Rohan");
Student s7(106,"Samuel");
Student s8(107,"Bharat");
students.insert(pair<int,Student *>(101,&s1));
students.insert(pair<int,Student *>(102,&s2));
students.insert(pair<int,Student *>(103,&s3));
students.insert(pair<int,Student *>(111,&s4));
students.insert(pair<int,Student *>(104,&s5));
students.insert(pair<int,Student *>(105,&s6));
students.insert(pair<int,Student *>(106,&s7));
students.insert(pair<int,Student *>(104,&s8));
```

```
for(auto m:students)
{
cout<<m.second->getRollNumber()<<","<<m.second->getName()<<endl;
}
cout<<"Looking for student with roll number as 192"<<endl;
map<int,Student *>::iterator fi;
fi=students.find(192);
if(fi!=students.end())
{
cout<<(*fi).second->getRollNumber()<<","<<(*fi).second->getName()<<endl;
}
else
{
cout<<"192 not found"<<endl;
}

cout<<"Looking for student with roll number as 104"<<endl;
fi=students.find(104);
if(fi!=students.end())
{
cout<<(*fi).second->getRollNumber()<<","<<(*fi).second->getName()<<endl;
}
else
{
cout<<"104 not found"<<endl;
}
cout<<"Erasing student with roll number : 103"<<endl;
students.erase(103);
cout<<"After erasing student will roll number 103"<<endl;
for(auto m:students)
{
cout<<m.second->getRollNumber()<<","<<m.second->getName()<<endl;
}
return 0;
}
```

Assignment :

1) Try out other functions of the map class
2) Learn to make the key of complex data type
3) learn how and when to use the multimap class