C++ standard template library

string

- C++ provides a powerful alternative for the char*, string.
- It is not built-in data type, but it is container in STL.
- To use string in our program, we must include string header file.
- Examples:

```
#include<string.h>

int main() {
    string s0;     // s0=“”
    string s1=“Hello World”; //s1=“Hello World”
    string s2(s1);
    string s3(s1,1,3);  //s3=“ell”
    string s4(5,’*’);   //s4=“*****”
    string s5(s1.begin(),s1.begin()+3);   // s5=“Hel”
}
```

String – Member Functions

Functions	Description
append()	Inserts additional characters at the end of the string (can also be done using '**+**' or '**+=**' operator). Its time complexity is O(N) where N is the size of the new string.
assign()	Assigns new string by replacing the previous value (can also be done using '**=**' operator).
at()	Returns the character at a particular position (can also be done using '**[ ]**' operator). Its time complexity is O(1).
begin()	Returns an iterator pointing to the first character. Its time complexity is O(1).
clear()	Erases all the contents of the string and assign an empty string ("") of length zero. Its time complexity is O(1).
compare()	Compares the value of the string with the string passed in the parameter and returns an integer accordingly. Its time complexity is O(N + M) where N is the size of the first string and M is the size of the second string.
copy()	Copies the substring of the string in the string passed as parameter and returns the number of characters copied. Its time complexity is O(N) where N is the size of the copied string.
c_str()	Convert the string into C-style string (null terminated string) and returns the pointer to the C-style string. Its time complexity is O(1).
length()	Returns the length of the string. Its time complexity is O(1).

String – Member Functions

Functions	Description
empty()	Returns a boolean value, true if the string is empty and false if the string is not empty. Its time complexity is O(1).
end()	Returns an iterator pointing to a position which is next to the last character. Its time complexity is O(1).
erase()	Deletes a substring of the string. Its time complexity is O(N) where N is the size of the new string.
find()	Searches the string and returns the first occurrence of the parameter in the string. Its time complexity is O(N) where N is the size of the string.
insert()	Inserts additional characters into the string at a particular position. Its time complexity is O(N) where N is the size of the new string.
replace()	Replaces the particular portion of the string. Its time complexity is O(N) where N is size of the new string
resize()	Resize the string to the new length which can be less than or greater than the current length. Its time complexity is O(N) where N is the size of the new string.
size()	Returns the length of the string. Its time complexity is O(1).
substr()	Returns a string which is the copy of the substring. Its time complexity is O(N) where N is the size of the substring.

String-Example

```
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{
    string s, s1;
    s = "HELLO";
    s1= "HELLO";
    if(s.compare(s1) == 0)
      cout << s << " is equal to " << s1 << endl;
    else
      cout << s << " is not equal to " << s1 << endl;
    s.append(" WORLD!");
    cout << s << endl;
    printf("%s\n", s.c_str());
    if(s.compare(s1) == 0)
      cout << s << " is equal to " << s1 << endl;
    else
      cout << s << " is not equal to " << s1 << endl;
  return 0;
}
```

vector

- Vectors are sequence containers that have dynamic size.
- In other words, vectors are dynamic arrays.
- Just like arrays, vector elements are placed in contiguous storage location so they can be accessed and traversed using iterators.
- To traverse the vector we need the position of the first and last element in the vector which we can get through **begin()** and **end()** or we can use indexing from 0 to **size()**.
- To use vector include vector header file

```
vector<int> a; // empty vector of ints
vector<int> b (5, 10); // five ints with value 10
vector<int> c (b.begin(),b.end()); // iterating through second
vector<int> d (c); // copy of c
```

Vector – Member Functions

Functions	Description
back()	Returns the reference to the last element. Its time complexity is O(1).
clear()	Deletes all the elements from the vector and assign an empty vector. Its time complexity is O(N) where N is the size of the vector.
at()	Returns the reference to the element at a particular position (can also be done using '[ ]' operator). Its time complexity is O(1).
begin()	Returns an iterator pointing to the first element of the vector. Its time complexity is O(1).
empty()	Returns a boolean value, true if the vector is empty and false if the vector is not empty. Its time complexity is O(1).
end()	Returns an iterator pointing to a position which is next to the last element of the vector. Its time complexity is O(1).
erase()	Deletes a single element or a range of elements. Its time complexity is O(N + M) where N is the number of the elements erased and M is the number of the elements moved.
front()	Returns the reference to the first element. Its time complexity is O(1).

Vector – Member Functions

Functions	Description
insert()	Inserts new elements into the vector at a particular position. ts time complexity is O(N + M) where N is the number of elements inserted and M is the number of the elements moved .
pop_back()	Removes the last element from the vector. Its time complexity is O(1).
push_back()	Inserts a new element at the end of the vector. Its time complexity is O(1).
resize()	Resizes the vector to the new length which can be less than or greater than the current length. Its time complexity is O(N) where N is the size of the resized vector.
size():	Returns the number of elements in the vector. Its time complexity is O(1).

Vector-Example

- Traversing vector

```
void traverse(vector<int> v)
{
vector <int>::iterator it;
for(it = v.begin();it != v.end();++it)
    cout << *it << ' '; cout << endl;

for(int i = 0;i < v.size();++i)
   cout << v[i] << ' '; cout << endl;
}
```

Vector-Example

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector <int> v;
    vector <int>::iterator it;
    v.push_back(5);
    while(v.back() > 0)
        v.push_back(v.back() - 1);

    for(it = v.begin(); it != v.end();++it)
      cout << *it << ' '; cout << endl;
    for(int i = 0;i < v.size();++i)
      cout << v.at(i) << ' '; cout << endl;
    while(!v.empty()) {
       cout << v.back() << ' '; v.pop_back();
    }
    cout << endl;
return 0;
}
```

Output:

```
5 4 3 2 1 0
5 4 3 2 1 0
0 1 2 3 4 5
```

list

- List is a sequence container which takes constant time in inserting and removing elements. List in STL is implemented as Doubly Link List.
- The elements from List cannot be directly accessed. For example to access element of a particular position ,you have to iterate from a known position to that particular position.

```
//declaration - list is of type integer
list <int> LI;

// here LI will have 5 int elements of value 100.
list<int> LI(5, 100)
```

List – Member Functions

Functions	Description
back()	It returns reference to the last element in the list.Its time complexity is O(1).
begin()	It returns an iterator pointing to the first element in list.Its time complexity is O(1).
empty()	It returns whether the list is empty or not.It returns 1 if the list is empty otherwise returns 0.Its time complexity is O(1).
end()	It returns an iterator referring to the theoretical element(doesn't point to an element) which follows the last element.Its time complexity is O(1).
erase()	It removes a single element or the range of element from the list.Its time complexity is O(N).
front()	It returns reference to the first element in the list.Its time complexity is O(1).
assign()	It assigns new elements to the list by replacing its current elements and change its size accordingly.It time complexity is O(N).
push_back()	It adds a new element at the end of the list, after its current last element. Its time complexity is O(1).
push_front()	It adds a new element at the beginning of the list, before its current first element. Its time complexity is O(1).

List – Member Functions

Functions	Description
remove()	It removes all the elements from the list, which are equal to given element. Its time complexity is O(N).
pop_back()	It removes the last element of the list, thus reducing its size by 1. Its time complexity is O(1).
pop_front()	It removes the first element of the list, thus reducing its size by 1. Its time complexity is O(1).
insert()	It insert new elements in the list before the element on the specified position. Its time complexity is O(N).
reverse()	It reverses the order of elements in the list. Its time complexity is O(N).
size()	It returns the number of elements in the list. Its time complexity is O(1).

List-Example

```
#include <iostream>
#include <list>
using namespace std;
int main() {
    list <int> LI;
    list <int>::iterator it; //inserts elements at end of list
    LI.push_back(4);
    LI.push_back(5); //inserts elements at beginning of list
    LI.push_front(3);
    LI.push_front(5); //returns reference to first element of list
     it = LI.begin(); //inserts 1 before first element of list
    LI.insert(it,1);
    cout<<"All elements of List LI are: " <<endl;
    for(it = LI.begin();it!=LI.end();it++) {
        cout<<*it<<" ";
    }
cout<<endl;
```

```
LI.reverse();
cout<<"All elements of List LI are after reversing: " <<endl;
for(it = LI.begin();it!=LI.end();it++) {
    cout<<*it<<" ";
}
cout<<endl; //removes all occurences of 5 from list
LI.remove(5);
cout<<"Elements after removing all occurence of 5 from
List"<<endl;
for(it = LI.begin();it!=LI.end();it++) {
cout<<*it<<" ";
}
cout<<endl; //removes last element from list
LI.pop_back(); //removes first element from list
LI.pop_front();
return 0;
}
```

Output:

All elements of List LI are: 1 5 3 4 5
All elements of List LI are after reversing: 5 4 3 5 1
Elements after removing all occurrence of 5 from List 4 3 1

pair

- Pair is a container that can be used to bind together a two values which may be of different types. Pair provides a way to store two heterogeneous objects as a single unit.

```
pair <int, char> p1; // default
pair <int, char> p2 (1, 'a'); // value inititialization
pair <int, char> p3 (p2); // copy of p2
```

- We can also initialize a pair using **make_pair()** function. **make_pair(x, y)** will return a pair with first element set to x and second element set to y.

```
p1 = make_pair(2, 'b');
```

- To access the elements we use keywords, first and second to access the first and second element respectively.

```
cout << p2.first << ' ' << p2.second << endl;
```

Pair-Example

```
#include <iostream>
#include <utility>
using namespace std;
int main() {
        pair <int, char> p;
        pair <int, char> p1(2, 'b');
        p = make_pair(1, 'a');
        cout << p.first << ' ' << p.second << endl;
        cout << p1.first << ' ' << p1.second << endl;
        return 0;
}
```

Output:

```
1 a
2 b
```

sets

- Sets are containers which store only unique values and permit easy look ups.
- The values in the sets are stored in some specific order (like ascending or descending).
- Elements can only be inserted or deleted, but cannot be modified.
- We can access and traverse set elements using iterators just like vectors.

```
set<int> s1; // Empty Set
int a[]= {1, 2, 3, 4, 5, 5};
set<int> s2 (a, a + 6); // s2 = {1, 2, 3, 4, 5}
set<int> s3 (s2); // Copy of s2
set<int> s4 (s3.begin(), s3.end()); // Set created using iterators
```

Set – Member Functions

Functions	Description
begin()	Returns an iterator to the first element of the set. Its time complexity is O(1).
empty()	Returns true if the set is empty and false if the set has at least one element. Its time complexity is O(1).
end()	Returns an iterator pointing to a position which is next to the last element. Its time complexity is O(1).
erase()	Deletes a particular element or a range of elements from the set. Its time complexity is O(N) where N is the number of element deleted.
clear()	Deletes all the elements in the set and the set will be empty. Its time complexity is O(N) where N is the size of the set.
count()	Returns 1 or 0 if the element is in the set or not respectively. Its time complexity is O(logN) where N is the size of the set.
find()	Searches for a particular element and returns the iterator pointing to the element if the element is found otherwise it will return the iterator returned by end(). Its time complexity is O(logN) where N is the size of the set.
insert()	insert a new element. Its time complexity is O(logN) where N is the size of the set.
size()	Returns the size of the set or the number of elements in the set. Its time complexity is O(1).

set-Example

```
#include <iostream>
#include <set>
using namespace std;
int main() {
      set <int> s;
      set <int>::iterator it;
      int A[] = {3, 5, 2, 1, 5, 4};
      for(int i = 0;i < 6;++i)
           s.insert(A[i]);
      for(it = s.begin();it != s.end();++it)
          cout << *it << ' '; cout << endl;
      return 0;
}
```

Output:

1 2 3 4 5

map

- Maps are containers which store elements by mapping their value against a particular key. It stores the combination of key value and mapped value following a specific order.
- Here key value are used to uniquely identify the elements mapped to it.
- The data type of key value and mapped value can be different.
- Elements in map are always in sorted order by their corresponding key and can be accessed directly by their key using bracket operator ([ ]).
- In map, key and mapped value have a pair type combination, i.e. both key and mapped value can be accessed using pair type functionalities with the help of iterators.

```
//declaration. Here key values are of char type and mapped values(value of element) is of int type.
map <char ,int > mp;

//  It will map value 1 with key 'b'. We can directly access 1 by using mp[ 'b' ].
mp['b'] = 1;

mp['a'] = 2;
```