KJSCE - Codecell: C++ Reference

1. Introduction
    1. Purpose of Codecell
    2. Purpose of this session / document
    3. What is C++
    4. C++ compared to other Languages
    5. Compilers: Turbo and GCC
    6. How to use GCC/ Basic program structure in GCC
    7. IDEs, Online Judges, Online IDEs
    8. Codechef's "Code, Compile and Run" online IDE
2. STL (Standard Template Library)
    1. What is STL?
    2. Why STL?
    3. Overview
    4. Vectors
        1. What are they?
        2. Comparison with Arrays
        3. Declaration and Syntax
        4. In - Built Functions
        5. Use Cases
        6. Code snippets
        7. Problems for practice
    5. Strings
        1. What are they?
        2. Declaration,Syntax and comparison with C-type strings
        3. Input / Output
        4. In - Built Functions
        5. Use Cases
        6. Code Snippets
        7. Problems for practice
    6. Sets / Multisets
        1. What are they?
        2. Difference between Sets and Multisets
        3. Declaration and Syntax
        4. Insertion and Iteration
        5. In - Built Functions
        6. Use Cases
        7. Code Snippets
        8. Problems for Practice
    7. Pairs
        1. What are they?
        2. Declaration and Syntax
        3. Use Cases
        4. Code Snippets
    8. Maps
        1. What are they?
        2. Declaration and Syntax
        3. Insertion, Access and Iteration
        4. In - Built Functions
        5. Use Cases
        6. Code Snippets
        7. Problems for Practice
    9. <algorithm>

                1. Algorithms?
                2. Max, min
                3. Sort
                4. Count
                5. Fill_n
                6. Binary_search
                7. Lower_bound, Upper_bound
        10. What more in STL?
3. Competitive Coding and C++
4. References

1] Introduction

1.1 Purpose of Codecell

Codecell is a CodeChef Campus Chapter that aims at promoting competitive programming in various colleges and schools around the world and helping students become better problem solvers which goes a long way in learning as well as in their career development.

1.2 Purpose of this session / document

There are numerous programming languages available, but C++ proves to be more efficient than most languages because it works very close to the hardware of the system. The advantage of this is that the execution time is significantly reduced and the memory consumption is decreased drastically. Now, the focus is on STL in particular.STL stands for standard template library. That's because STL provides ready-made set of common classes of algorithms and data structures.

1.3 What is C++

C++ is a procedural as well as object oriented programming (OOP) language, developed by Bjarne Stroustrup, and is an extension of C language. It is considered to be an intermediate level language, as it encapsulates both high and low level language features. The main highlight of C++ is a collection of pre-defined classes, which are data types that can be instantiated multiple times. A few of the essential concepts within C++ programming language include polymorphism, virtual and friend functions, templates, namespaces and pointers.

1.4 C++ compared to other Languages

The main languages that C++ is compared to are Java, C and Python. Java and C++ are both object oriented programming languages, but Java is strictly OO whereas C++ allows for procedural programming as well. Alongside, C++ runs very close to the hardware whereas Java runs in a VM (JVM) which makes it slower and resource heavy.

1.5 Compilers: Turbo and GCC

The main difference between turbo and GCC compilers is that turbo has not received an update since 20000. So it's quite out dated. In GCC, there are header files such as **cstdio** which can be used to incorporate the functions of c in c++. GCC has namespace std. Namespace is a block which defines the scope of the function which it contains. Std stands for standard i.e. it contains all the standard function such as **cout** and **cin**.

Using namespace will eliminate the confusion as to which version of the functions having the same name must be used in the program.

Defining a namespace

```
namespace namespace_name {
   // code declarations
}
```

Calling a function or variable in namespace

```
name::code;  // code could be variable or function.
```

GCC eliminates the use of .h extension in header files.

1.5 How to use GCC/ Basic program structure in GCC

Gcc is preinstalled in any unix based system.To invoke gcc to compile C++ files, "g++" command is used.
In Windows 10, one can enable the ubuntu unix terminal by enabling the developer mode in settings.
Say you have a file helloworld.C as follows :

```
#include <stdio.h>
void main (){
   printf("Hello World\n");
}
```

You can compile and run it from the unix prompt as follows :

```
% g++ helloworld.cpp
```

This creates an executable called "a.out". You can run it by typing

```
% ./a.out
```

Since no executable name was specified to g++, a.out is chosen by default. Use the "-o" option to change the name
:

```
% g++ -o helloworld helloworld.cpp
```

creates an executable called "helloworld".

1.6 IDEs, Online Judges, Online IDEs

An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and adebugger. Most modern IDEs have intelligent code completion. Some IDEs, such as NetBeans and Eclipse, contain acompiler, interpreter, or both; others, such as SharpDevelop and Lazarus, do not. The boundary between an integrated development environment and other parts of the broader *software development environment* is not well-defined. Sometimes a version control system, or various tools to simplify the construction of a Graphical User Interface (GUI), are integrated. Many modern IDEs also have a class browser, an object browser, and a class hierarchy diagram, for use in object-oriented software development.

There are many online cross platform IDE's .Check http://www.tutorialspoint.com/codingground.htm for more details and types of IDE's.

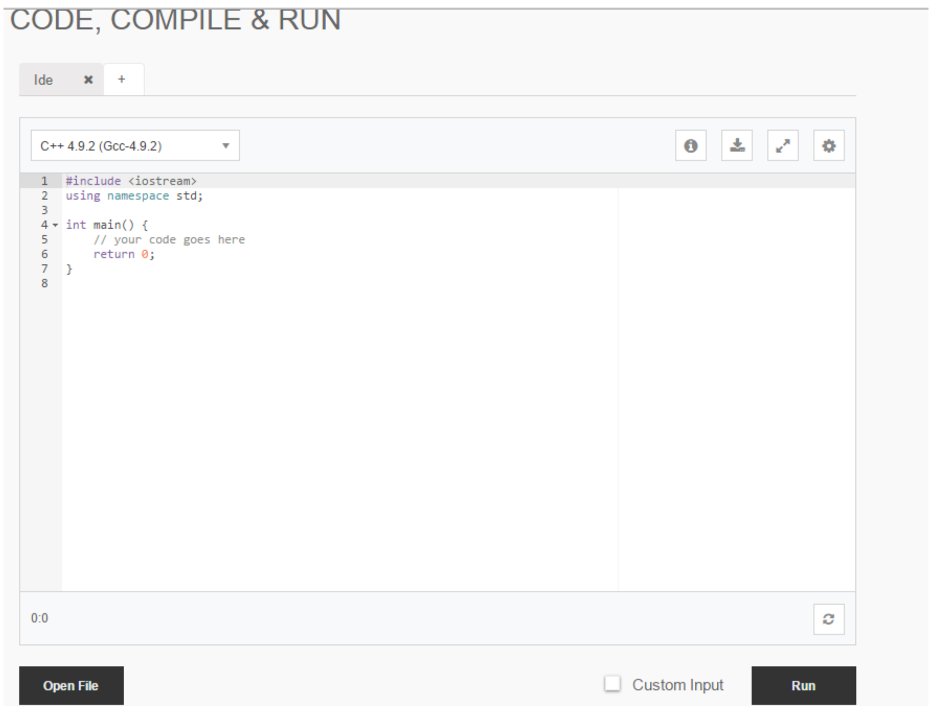
An online judge is an online system to test programs in programming contests. They are also used to practice for such contests. Many of these systems organize their own contests.

The system can compile and execute code, and test them with pre-constructed data. Submitted code may be run with restrictions, including time limit, memory limit, security restriction and so on. The output of the code will be captured by the system, and compared with the standard output. The system will then return the result. When mistakes were found in a standard output, rejudgement using the same method must be made. Many of the online judges are done by hand or locals.

Online Judges have ranklists showing users with the biggest number of accepted solutions and shortest execution time for a particular problem.

1.7 Codechef 's "Code, Compile and Run" online IDE

CodeChef has its very own IDE. The website url for it is www.codechef.com/ide. It provides support to over 50 programming languages. The page looks like in the picture:



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

int main() {
    // your code goes here
    return 0;
}

```

The custom input option is used to give customised input before compilation.Unlike turbo , the input must be predefined before compilation as the user cannot enter inputs once the program is executed.

2] STL (Standard Template Library)

2.4) VECTORS

2.4.1. Vectors?

Vectors are basically dynamic arrays. It can resize itself according to our needs.
They are placed in continuous memory locations and can be traversed using iterators.
Elements are entered from the end. So it becomes easier to insert and delete elements.

2.4.2. Comparison to array

In array if suppose we are declaring array a with 10 elements
Int a[10];
Now if we want to add the 11th element it is not possible as the memory was allocated to only 10 elements. Now that is not the case in vectors, vectors can be resized.
You all might think that malloc and calloc in array is also used for dynamic memory allocation. But the difference is in malloc and calloc altogether memory is allocated and in vectors memory is allocated as you insert.

2.4.3. Declaration And Syntax

A vector is declared as

```
vector < > vector_name;
```

< > is for= generic purpose, that means it can have any data type like integer, float, string, double, etc
If you want to create an integer vector a then the syntax is

```
vector <int> a;
```

The iterator that is used to traverse throughout the vector is declared as

```
vector < > :: iterator iterator_name;
```

2.4.4. Built in functions

- push_back(const value_type g)

It adds a new element g at the end of the vector and the vector container size increases by 1.

- pop_back()

It removes the element at the vector.

- empty()

It returns whether the container is empty.

- begin()

It returns an iterator pointing to the first element in the vector.

- end()

It returns an iterator pointing to the last element in the vector.

- front()

It returns a reference to the first element in the vector.

- back()

It returns a reference to the last element in the vector.
The difference between the functions begin() and front() is that,
begin() function will return the address of the first element that the iterator is pointing to and front() function will return the value of the first element.
Similarly end() function will return the address of the last element that the iterator is pointing to and back() function will return the value of the last element.

Example :

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
   vector <int> a;
   vector <int> :: iterator i;
   for (int i = 1; i <= 5; i++)
       a.push_back(i);
   cout << "Output using begin and end functions \t:\t";
   for (i = a.begin(); i != a.end(); ++i)
       cout << *i << '\t';
   cout << endl << endl;
   cout<<”Reversed output using front and back functions \t:\t”;
      while(!a.empty())
      {
cout<<a.back()<<’\t’;
      a.pop_back();
      }
    return 0;
}
```

Output:

```
Output using begin and end functions             1      2      3      4      5
Reversed output using front and back functions          5      4      3      2      1
```

<u>More functions</u>

- reference operator[g]

It returns a reference to the element at position ‘g’ in the vector.

- at(g)

It again returns a reference to the element at position ‘g’ in the vector.

- size()

It returns the number of elements in the vector.

- capacity()

It returns the size of the storage space currently allocated to the vector expressed as number of elements.
Example:

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
```

```
int main()
{
   vector <int> b;
   for (int i = 1; i <= 10; i++)
       b.push_back(i * 10);
   cout << "Reference operator [b] : b[2] = " << b[2];
   cout << endl;
   cout << "at : b.at(4) = " << b.at(4);
   cout << endl;
      cout << "Size : " << b.size();
   cout << "\nCapacity : " << b.capacity();
   return 0;
}
```

Output:

```
Reference operator [b] : b[2] = 30
at : b.at(4) = 50
Size : 10
Capacity : 16
```

<u>Modifiers:</u>

- assign(size_type n, const value_type g)

Assigns new content to vector and resize.
That means it will assign n elements with the value g.

- assign(input_iterator first, input_iterator last)

Assigns new content to vector and resize.
Example:

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
   vector <int> a1;
   vector <int> a2;
   vector <int> a3;
   a1.assign(5, 10);   // 5 elements with value 10 each
   vector <int> :: iterator i;
   i = a1.begin() + 1;
   a2.assign(i, a1.end() - 1); // the 3 middle values of a1
   int a[] = {1, 2};
   a3.assign(a, a + 2);   // assigning from array
   cout << "Elements of a1:\n" ;
      for (i = a1.begin(); i != a1.end(); ++i)
       cout << *i << '\t';
      cout << "\nElements of a2:\n" ;
      for (i = a2.begin(); i != a2.end(); ++i)
       cout << *i << '\t';
      cout << "\nElements of a3:\n" ;
      for (i = a3.begin(); i != a3.end(); ++i)
       cout << *i << '\t';
   return 0;
}
```

Output:

```
Elements of a1:
10      10      10      10      10
Elements of a2:
10      10      10
Elements of a3:
1       2
```

Insertion and Deletion in Vectors:

Unlike arrays insertion and deletion in vectors is very easy.
In arrays if one element has to be inserted then the elements following it should be shifted one position right.
For example
Int a = {1,2,4,5};

1	2	4	5	6

If 3 has to be inserted before 4 then
The numbers 4,5,6 should be moved one position to their right

1	2	4	5	6	

1	2		4	5	6

Now 3 can be inserted before 4

1	2	3	4	5	6

Similarly in deletion the position of the deleted element should be occupied by the element following it by shifting left.
But this is not the case of Vectors any element can be easily inserted and deleted.

- insert(const_iterator q, const value_type g)

Adds element 'g' before the element referenced by iterator 'q' and returns an iterator that points to the newly added element.

- insert(const_iterator q, size_type n, const value_type g)

Adds 'n' elements each with value 'g' before the element currently referenced by iterator 'q' and returns an iterator that points to the first of the newly added elements.

- ¨ insert(const_iterator q, InputIterator first, InputIterator last)

Adds a range of elements starting from first to last, the elements being inserted after the position currently referred by 'q'.
Example:
```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
   vector <int> a1;
   vector <int> :: iterator it;
      a1.assign(3,10);
   it = a1.begin();
   it = a1.insert(it, 20);          //insert 20 before the 1st value of a1
      a1.insert(it, 2, 30);  //insert 2 times 30 before the new 1st value
of a1
   it = a1.begin();
   vector <int> a2;
      a2.assign(2,40);
   a1.insert(it + 2, a2.begin(), a2.end()); //insert value of a2 before
it+2
   cout << "a1 contains : ";
   for (it = a1.begin(); it < a1.end(); it++)
       cout << *it << '\t';
   return 0;
}
```
Output:
```
A1 contains: 30      30    40    40    20    10    10    10
```

- erase(const_iterator q)

Deletes the element referred by 'q' and returns an iterator to the element followed by the deleted element.

- erase(const_iterator first, const_iterator last)

Deletes the elements in the range first to last, with the first iterator included in the range and the last iterator not included, and returns an iterator to the element followed by the last deleted element.

Example:

```
#include <iostream>
#include <vector>
using namespace std;
int main ()
{
   vector <int> a;
   for (int i = 1; i <= 5; i++)
       a.push_back(i * 2);
   // erase the 5th element
   a.erase(gquiz.begin() + 3);
   // erase the first 5 elements:
   gquiz.erase(a.begin(), a.begin() +2);
   cout << "gquiz contains :";
   for (int i = 0; i < a.size(); ++i)
       cout << a[i] << '\t';
   return 0;
}
```

Output:

```
a contains: 6   10
```

- swap(vector q, vector r) – Swaps the contents of ‘q’ and ‘r’.
- clear() – Removes all elements from the vector.

Example:

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
   vector <int> a1;
   vector <int> a2;
   vector <int> :: iterator i;
   a1.push_back(10);
   a1.push_back(20);
   a2.push_back(30);
   a2.push_back(40);
   cout << "Before Swapping, \n"
        <<"Contents of vector a1 : ";
   for (i = a1.begin(); i !=a1.end(); ++i)
       cout << *i << '\t';
   cout << "\nContents of vector a2 : ";
   for (i = a2.begin(); i != a2.end(); ++i)
       cout << *i << '\t';
   swap(a1, a2);
   cout << "\n\nAfter Swapping, \n";
   cout << "Contents of vector a1 : ";
   for (i = a1.begin(); i != a1.end(); ++i)
       cout << *i << '\t';
   cout << "\nContents of vector gquiz2 : ";
   for (i = a2.begin(); i != a2.end(); ++i)
       cout << *i << '\t';
   cout << "\n\nNow, we clear a1 and add 100 to a1\n";
   a1.clear();
     a1.push_back(100);
   cout << “a1=”<<a1.front();
   return 0;
}
```

2.4.7. Problems for practice			
https://www.codechef.com/problems/HOLES https://www.codechef.com/problems/PLANEDIV https://www.codechef.com/problems/VCS https://www.codechef.com/problems/SPIT1			

2.5) Strings

2.5.1. What are strings?

Strings are basically words or sentence. In Programming strings are array of characters.
There are two types of strings:
We all have been using the generic C strings more generally array of characters. In C if we wish to save a particular sentence or word (a string) we have to define a character array
I.e. char a[20];
The drawback of this is that we have to guess what will be the maximum length of the array which is really inconvenient at times. Also this strings have a ‘\0’ after the last character.
C ++ introduces a new concept called strings which is nothing a but a class written in c++;
In C++ Strings Strings we can declare a string like:
String a;
That’s it we don’t need to give it a predetermined size.C++ Strings solves many of our problems we face while using C strings.
The following are the changes in C strings and C++ strings

2.5.2. Declaration,Syntax and comparison with C-type Strings:

Basic Declaration Syntax:

```
    string  string_name;
```

Initialization:

In C-type Strings
char a[] = “Hello World!”;.
char a[7]= “Hello!”; (Why 7?)
char a[]

In C++ Strings

string a(“Hello World!”);
string a = “Hello World!”;

Assignment:
a = “Hello World!”;
b = a;

Concatenation:

In C-type Strings
concat(str1,str2);
or if we want to store it in a new string
strcpy(str3, concat(str1,str2));

In C++ Strings Strings

str1 + str2;
str3 = str1 + str2;
Accessing:

In C-type Strings:
 a[index];

In C++ Strings Strings
 a.at(index);
 A[index];
 a(index,count); // strings starting at index and ending at index+count
 <u>Also using iterator like other stl members</u>

Comparing:

In C-type Strings
strcmp(str1,str2)
Positive if str1 is large or not equal.
Negative if str2 is short or not equal
0 if str1 is equal to str2

In C++ Strings Strings

We can use the comparison operators like ‘<’ ,‘>’ & ‘==’

Length:
In C-type Strings
strlen(str1);
Or
sizeof(str1)/sizeof(char);

In C++ Strings Strings;
str1.length();

2.5.3) Input and Output:

Input:

In C-type Strings
scanf(“%s”,&a);

In C++ Strings
cin>>a;

Output:

In C-type Strings
printf(“The string is %s”,a);

In C++ strings:
cout<<a;

A bit about getline and get?

In what follows, keep in mind that cin ignores white space when reading a string, while cin.get(), cin.getline() and getline() do not. Remember too that cin.getline() and getline() consume the delimiter while cin.get() does not. Finally, cin can be replaced with any open input stream, since file input with inFile, say, behaves in a manner completely analogous to the corresponding behavior of cin. Analogously, in the output examples given immediately above, cout could be replaced with any text output stream variable, say outFile. In all cases, x (for char a[x];) is the maximum number of characters that will be read.

2.5.4. In-Build function of C++ strings:

clear():

Syntax:

s.clear();
Clears all the characters in the string

c_str()

Syntax:

s.c_str();
Converts the C++ string into a normal C-type Strings.

begin():

Syntax:

s.begin();
Returns an iterator pointing at the beginning

end():

Syntax:

s.end();
Return an iterator pointing at the end.

Note: It is not actually at the end but just like null character it is after the end.

size() or length():

Syntax:

s.size() ,s.length()

Returns the size of the string

push_back():
	Syntax:
		s.push_back();
	Append a new character to a strings

Substrings:

substr(X, Y) -- returns a copy of the substring (i.e. portion of the original string) that starts at index X and is Y characters long

substr(X) -- returns a copy of the substring, starting at index X of the original string and going to the end

Append -- several versions.

All of these append something onto the END of the original string (i.e. the calling object, before the dot-operator)

append(str2) -- appends str2 (a string or a c-string)
append(str2, Y) -- appends the first Y characters from str2 (a string or char array)
append(str2, X, Y) -- appends Y characters from str2, starting at index X
append(X, CH) -- appends X copies of the character CH

Assign --

there are several functions called assign which are for assigning data to a string. There are versions with the same parameter lists as with the append functions, but these assign the string to the requested data, while append adds it to the end of an existing string

Compare -- multiple versions

str1.compare(str2) -- performs a comparison, like the c-string function strcmp. A negative return means str1 comes first. Positive means str2 comes first. 0 means they are the same

str1.compare(str2, X, Y) -- compares the portions of the strings that begin at index X and have length Y. Same return value interpretation as above

Find -- multiple versions

str.find(str2, X) -- returns the first position at or beyond position X where the string str2 is found inside of str

str.find(CH, X) -- returns the first position at or beyond position X where the character CH is found in str

Insert -- multiple versions

str.insert(X, Y, CH) -- inserts the character CH into string str Y times, starting at position X
str.insert(X, str2) -- inserts str2 (string object or char array) into str at position X

2.5.5) Use Cases:

So where do we use strings? We use strings simply because strings make our life easy. We don’t have to care about where to put ‘/0’.(Couldn’t find more on this!)

2.5.6. Code Snippet:

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

int main (){
  string str1 = "Hello";
  string str2 = "World";
  string str3;
  int  len ;

  // copy str1 into str3
  str3 = str1;
  cout << "str3 : " << str3 << endl;

  // concatenates str1 and str2
  str3 = str1 + str2;
  cout << "str1 + str2 : " << str3 << endl;

  // total lenghth of str3 after concatenation
  len = str3.size();
  cout << "str3.size() :  " << len << endl;
  return 0;
}


;
}
```

2.5.7. Problems:

Easy:
https://www.codechef.com/problems/COEX04
https://www.codechef.com/problems/CLCO06

Medium:
https://www.codechef.com/problems/PALSTR
https://www.codechef.com/problems/CODE1603

Tough:
https://www.codechef.com/problems/CLCO08
https://www.codechef.com/problems/CODE1601

Reference link: http://www.cs.fsu.edu/~myers/c++/notes/stringobj.html

2.6) Sets/Multisets

2.6.1. What are they?

Sets and Multisets are one of the containers in STL. Much like the sets that we have studied in Mathematics, the container set, stores unique elements in a certain order.
The value of a set cannot be modified; however, it can be deleted and replaced with another value.
Internally, sets and multisets are always sorted according to some criterion. By default, the sets are sorted in an ascending order.
Multisets are a type of containers similar to set, with an exception that multiple elements can have same values.

2.6.2. Difference between sets and multisets?

The only difference between sets and multisets is that multisets allows you to have duplicate elements while sets don't.

2.6.3. Declaration and Syntax

STL Sets and Multisets are included in the container library called set which is declared as follows:

```
#include<set>
```

Declaration of Sets:
set<data-type> *variable-name*

Declaration of Multisets:

multiset<data-type> *variable-name*

Example:

```
#include<iostream>



#include<set>
int main()
{
   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
   return 0;
}
```

2.6.4. Insertion and Iteration

Iterators
An *iterator* is any object that, pointing to some element in a range of elements (such as an array or a container), has the ability to iterate through the elements of that range using a set of operators (with at least the increment (++) and dereference (*) operators).

The most obvious form of iterator is a *pointer*: A pointer can point to elements in an array, and can iterate through them using the increment operator (++). But other kinds of iterators are possible. For example, each container type (such as a list) has a specific*iterator* type designed to iterate through its elements.

Like vectors, sets are accessed by using iterators.
The **insert()** function is used to insert ele
ments into a set.
The function **begin()** returns an iterator pointing to the first element of the set.
The function **end()** returns an iterator pointing to a theoretical element which is present after the last element of a set.

Example:

```
int main()
{
     set<int> arr;
     set<int>:: iterator it;
     arr.insert(5);
     arr.insert(1);
     arr.insert(3);
     arr.insert(9);
     for(it=arr.begin();it!=end();it++)
     {
           cout<<*it<<endl;
     }
```

```
    return 0;
}
```

Output:

```
1
3
5
9
```

As we can see the function begin() returns an iterator it pointing to the first element and it is incremented to traverse through the set and by dereferencing operators the value is extracted.

By the property of sets in stl, the elements of the sets are arranged in ascending order and are displayed likewise.

Reverse Iterator

In c++ reverse iterators are objects pointing to elements in a set. As the name suggests it reverses the direction in which an iterator iterates through a set.

Syntax: set<datatype>:: reverse_iterator *variable name*

It is used in functions **rbegin()** and **rend()**

rbegin(): This function reverses the beginning of the set, i.e it will point to the last element of the set

rend(): This function reverses the end i.e it will point to a theoretical element just before the first element of the set. Like in the previous example:

```
set<int> arr;
set<int>:: reverse_iterator rit;
arr.insert(5);
arr.insert(1);
arr.insert(2);
arr.insert(7);
for(rit=arr.rbegin();rit!=arr.rend();rit++)
cout<<*rit;
```

Output:

```
7 5 2 1
```

2.6.5. In-Built Functions

Some in-built functions for sets are:

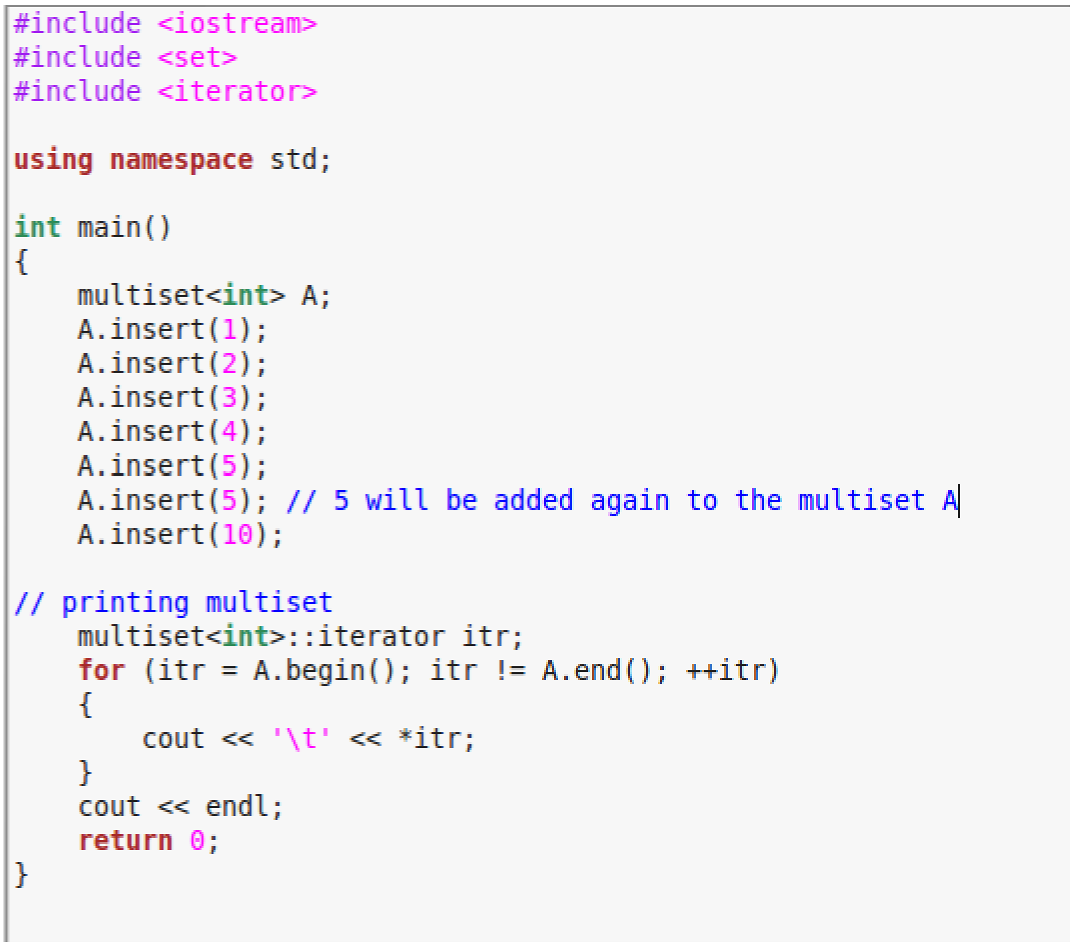
- **size()**: The function **size()** gives you the size of the set.
- **max_size()**: The function **max_size()** gives you the maximum size of the set
- **erase()**: The function **erase()** erases either 1 element or a range of elements from a set.
- **find()**: This function searches the container for an element equivalent to the value which is searched and returns an iterator to it if found, otherwise it returns an iterator pointing to the end of set.
- **swap()**: Exchanges the content two sets.
- **count(const g)**: This function returns the number of matches to element ‘g’ in the set
- **lower_bound(const g)**: This function returns an iterator to the first element that is equivalent to ‘g’ or definitely will not go before the element ‘g’ in the set

- **upper_bound(const g)**: This function returns an iterator to the first element that is equivalent to 'g' or definitely will go after the element 'g' in the set
- **empty()**: This function returns whether the set container is empty (i.e. whether its size is 0).
- **clear()**: The **clear()** function clears a set.

NOTE: Always use **clear()** function after you have finished performing required operations on a set, so that you can use the same set again.

2.6.6. Code Snippets

1. Insertion and Printing in multisets



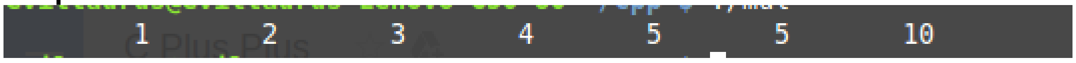
```
#include <iostream>
#include <set>
#include <iterator>

using namespace std;

int main()
{
    multiset<int> A;
    A.insert(1);
    A.insert(2);
    A.insert(3);
    A.insert(4);
    A.insert(5);
    A.insert(5); // 5 will be added again to the multiset A
    A.insert(10);

// printing multiset
    multiset<int>::iterator itr;
    for (itr = A.begin(); itr != A.end(); ++itr)
    {
        cout << '\t' << *itr;
    }
    cout << endl;
    return 0;
}
```

Output:



```
        1       2       3       4       5       5       10
```

2.6.7. Problems for Practice

[Codechef Easy: Cleaning Up](#)
[Codechef Beginner: VCS](#)

2.7 Pairs

2.7.1 What are they?

Pairs are very commonly required in programming competitions and they make certain things very easy to implement.

2.7.2 Declaration and syntax

Syntax

pair<T1,T2> pair1;

This class couples together a pair of values, which may be of different types (T1 and T2). The individual values can be accessed through its public members (first) and (second).

2.7.3 Inbuilt Functions

Here are some function for pair template :

•**Operator =** : assign values to a pair.

•**swap** : swaps the contents of the pair.

•**make_pair()** : create and returns a pair having objects defined by parameter list.

•**Operators(== , != , > , < , <= , >=)** : lexicographically compares two pairs.

2.7.4 Code Snippets

```
#include <iostream>
using namespace std;
int main ( void ) {
  pair<int,int> pair1, pair3;
  pair<int,string>  pair2;

  pair1 = make_pair(1, 2);
  pair2 = make_pair(1, "Study")
  pair3 = make_pair(2, 4)
  cout<< pair1.first << endl;
  cout<< pair2.second << endl;
  if(pair1 == pair3)
          cout<< "Pairs are equal" << endl;
  else
          cout<< "Pairs are not equal" << endl;

  return 0;
}
```

2.8 Maps

2.8.1 What are they?

The STL map class,can be thought of as an "associative array".Maps are containers that store elements formed by a combination of key value and mapped value,following a specific order.Keys are associated with particular values (e.g, you might use a student name as a key, and the student's grade as the data). Maps provide a way of using "associative arrays" that allow you to store data indexed by keys of any type you desire, instead of just integers as with arrays.

In a Map,the key values are used to uniquely identify the elements,while the mapped values store the content associated with the key.

2.8.2 Declaration and Syntax

A Map is declared as

map< datatype,datatype>map_name;

As you can see the < > symbol basically denotes the generic nature.So we can have any datatype in the two fields.(for eg int,float,string,char,etc).Suppose you make both the fields as integer type,then basically map behaves as an integer array.Hence we say that array are basically a type of map with the restriction that the index field must be an integer,which is not the case in map.

Example:

map<char,int>mp;

mp['a']=1;(This shows a map with key as character data type and value as integer)

2.8.3 Insertion,Access and Iteration

Insertion

One thing which is to remembered is that key of a map and corresponding values are always inserted as a pair, you cannot insert only key or just a value in a map.

Since keys are unique in a map, it first checks that whether the given key is already present in the map or not, if it is present the entry is not inserted in the map and the iterator to the existing key is returned otherwise new entry is inserted in the map.

Insertion can be done using the (insert) method.This method extends the container by inserting new elements, effectively increasing the container size by the number of elements inserted

There are two variations of insert() :

•insert(pair) : In this variation, a pair of key and value is inserted in the map. The inserted pair is always inserted at the appropriate position as keys are arranged in sorted order.

•insert(start_itr , end_itr) : This variation inserts the entries in range defined by start_itr and end_itr of another map.

Access

There are two ways to access the value mapped to a particular key.

1)at(key): Returns a reference to the mapped value of the element with key equivalent to key.

2)operator [ ]: Returns a reference to the value that is mapped to a key equivalent to key, performing an insertion if such key does not already exist.

Iteration

An iterator is any object that, pointing to some element in a range of elements.It has the ability to iterate through the elements of that range using a set of operators (with at least the increment (++) and dereference (*) operators).

An Iterator for map is declared as:

map<char,int>::iterator it;

2.8.4 Inbuilt Functions

Some other inbuilt functions are listed below:

erase and clear

erase() removes the entry from the map pointed by the iterator (which is passed as parameter), however if we want to remove all the elements from the map, we can use clear(), it clears the map and sets its size to 0.

There are two variations of erase :

erase(iterator_itr) : This removes entry from the map pointed by iterator (iterator_itr), reducing the size of map by 1.

erase(start_iterator,end_iterator): It removes the elements in range specified by the (start_iterator) and(end_iterator).

begin,end and find
begin, end and find returns an iterator. begin() returns the iterator to the starting entry of the map, end() returns the iterator next to the last entry in the map and find() returns the iterator to the entry having key equal to given key (passed as parameter).

2.8.5 Use Cases

The map is represented as tree, so the complexity of searching for an element in the map is O(log(n))

Hence,map is generally better for a paired search.

2.8.6 Code Snippets

```
// accessing mapped values
#include <iostream>
#include <map>
#include <string>
using namespace std;
int main ()
{
 map<char,std::string> mymap;
 mymap['a']="an element";
 mymap['b']="another element";
 mymap['c']=mymap['b'];
 cout << "mymap['a'] is " << mymap['a'] << '\n';
 cout << "mymap['b'] is " << mymap['b'] << '\n';
 cout << "mymap['c'] is " << mymap['c'] << '\n';
 cout << "mymap['d'] is " << mymap['d'] << '\n';
 cout << "mymap now contains " << mymap.size() << " elements.\n";
 return 0;
}
```

2.8.7 Problems for Practice

http://www.codechef.com/LTIME17/problems/CSS2/

http://www.codechef.com/problems/FROGV

2.9) <algorithm>

2.9.1. Algorithms?

An algorithm is a logical arrangement of steps you take in order to achieve a particular result.

You might have learnt Binary Search or Bubble sort in your first year. These are both algorithms, i.e. steps taken to achieve the detection of an element in case of the former and arrangement of elements by magnitude in the latter.

Competitive coding problems involve a lot of these algorithms, and it is essential to learn and understand them to be a good programmer. However, competitive coders and programmers in general have time constraints and cannot spend time on the implementation and debugging of a few simple, commonplace algorithms. Enter <algorithm>.

<algorithm> is a C++ header file which contains ready - to - use implementations of common algorithms. It helps developers and competitive coders cut down their implementation time drastically. Following are some algorithms already implemented.

2.9.2. Max and min

Max and min return the largest and smallest element of the two elements passed to it.
If at all both the elements are equal, the first element or parameter is displayed.
The function uses operator <,> (or comp, if provided) to compare the values.
For example,

```
#include <iostream>     // std::cout
#include <algorithm>    // std::max


Using namespace std;
int main () {
 cout << "max(1,2)==" << max(1,2) <<endl;
 cout << "min(2,1)==" << min(2,1) <<endl;
 cout << "min('a','z')==" << min('a','z') <<endl;
 cout << "max(3.14,2.73)==" << max(3.14,2.73) <<endl;
 return 0;
}
```

```
Output:
max(1,2)==2
min(2,1)==1
min('a','z')==a
max(3.14,2.73)==3.14
```

2.9.3. Sort

Sorts the elements in the range [first,last) into ascending order.
The elements are compared using operator < for the first version, and comp for the second.
This function can be used with any stl structure like vector,map or even an array.
For example,

```
// sort algorithm example
#include <iostream>     // std::cout
#include <algorithm>    // std::sort
#include <vector>       // std::vector


using namespace std;

int main () {
 int myints[] = {32,71,12,45,26,80,53,33};
 vector<int> myvector (myints, myints+8);               // 32 71 12 45 26 80 53 33

 // using default comparison (operator <):
 sort (myvector.begin(), myvector.begin()+4);           // (12 32 45 71) 26 80 53 33



 return 0;
}
```

2.9.4. Count

Count appearances of value in range
Returns the number of elements in the range [first,last) that compare equal to val.
The function uses operator== to compare the individual elements to val.
For example,

```
// count algorithm example
#include <iostream>     // std::cout
#include <algorithm>    // std::count
#include <vector>       // std::vector
using namespace std;


int main () {
 // counting elements in array:
 int myints[] = {10,20,30,30,20,10,10,20};   // 8 elements
 int mycount = count (myints, myints+8, 10);
 cout << "10 appears " << mycount << " times.\n";

 // counting elements in container:
vector<int> myvector (myints, myints+8);
 mycount =count (myvector.begin(), myvector.end(), 20);
cout << "20 appears " << mycount  << " times.\n";

 return 0;
}
```

Output

```
10 appears 3 times.
20 appears 3 times
```

2.9.5. Fill_n

Fill sequence with value
Assigns val to the first n elements of the sequence pointed by first.
For example,

```
vector<int> myvector (8,10);        // myvector: 10 10 10 10 10 10 10 10
fill_n (myvector.begin(),4,20);     // myvector: 20 20 20 20 10 10 10 10
fill_n (myvector.begin()+3,3,33);   // myvector: 20 20 20 33 33 33 1
```

2.9.6 binary_search

This algorithm searches for an element in a sorted container. If found, returns True, else returns false;

For example:
vector<int> v(8);
for(inti =0; i<8; i++) { v[i]=i; }
bool present = binary_search(v.begin(), v.end(), 19);

2.9.7 lower_bound, upper_bound

lower_bound returns an iterator to the smallest element greater than or equal to the given element.

upper_bound returns an iterator to the smallest element strictlt greater than given element.

Note: The container must be sorted before lower_bound and upper_bound are used

For example:
vector<int> v(8);
for(inti =0; i<8; i++) { v[i]=i; }
vector<int>::iterator it=lower_bound(v.begin(), v.end(), 6) //returns iterator to 6
it=upper_bound(v.begin(), v.end(), 6) //returns iterator to 7

2.10) What more in STL?

STL has a low more containers including but not limited to queues, stacks, priority_queues, unordered_maps, unordered_sets, heaps etc.

These are all available for study on various online resources.

3] Competitive Coding and C++

C++, being a compiled language with so many in-built containers, is a super fast, heavy utility language with relatively simpler syntax. In Competitive Coding, speed is of the essence, and hence C++ is preferred by competitive coders.
It allows for their solutions to be:

a. Simpler to read and debug
b. Quicker at execution (runtime)
c. Less memory heavy
d. Able to accomplish more in less code

For these same reasons, KJSCE Codecell also promotes C++ as the language of choice as far as Competitive Coding is concerned.

FIN