2023

DSA
PART-7

CYBER TRON

C++

(SEVEN PART)

T.I.M. HAMIM

Index: DSA < Part - 7 >

1. Intro to STL

STL :

The Standard Template Library (STL) is a set of C++ template classes to provide common programming data structures and functions such as lists, stacks, arrays, etc. It is a library of container classes, algorithms, and iterators. It is a generalized library and so, its components are parameterized.

The C++ Standard Template Library (STL) is a collection of algorithms, data structures, and other components that can be used to simplify the development of C++ programs. The STL provides a range of containers, such as vectors, lists, and maps, as well as algorithms for searching, sorting and manipulating data.
One of the key benefits of the STL is that it provides a way to write generic, reusable code that can be applied to different data types. This means that you can write an algorithm once, and then use it with different types of data without having to write separate code for each type.
The STL also provides a way to write efficient code. Many of the algorithms and data structures in the STL are implemented using optimized algorithms, which can result in faster execution times compared to custom code.

Some of the key components of the STL include:

1. **Containers:** The STL provides a range of containers, such as vector, list, map, set, and stack, which can be used to store and manipulate data.
2. **Algorithms:** The STL provides a range of algorithms, such as sort, find, and binary_search, which can be used to manipulate data stored in containers.
3. **Iterators:** Iterators are objects that provide a way to traverse the elements of a container. The STL provides a range of iterators, such as forward_iterator, bidirectional_iterator, and random_access_iterator, that can be used with different types of containers.
4. **Function Objects:** Function objects, also known as functors, are objects that can be used as function arguments to algorithms. They provide a way to pass a function to an algorithm, allowing you to customize its behavior.
5. **Adapters:** Adapters are components that modify the behavior of other components in the STL. For example, the reverse_iterator adapter can be used to reverse the order of elements in a container.

STL has 4 components:

1. Containers
2. Iterators
3. Algorithms
4. Functors

1. Containers:

A container is a holder object that stores a collection of other objects (its elements). They are implemented as class templates, which allows great flexibility in the types supported as elements.
The container manages the storage space for its elements and provides member functions to access them, either directly or through iterators (reference objects with similar properties to pointers).

1. Sequential Containers:
Implement data structures that can be accessed in a sequential manner.

- vectors
- stack
- queue
- pair (Not a container)

2. Ordered Containers:

- Maps
- Multimap
- Set
- Multiset

3. Unordered Containers:

- Unordered Map
- Unordered Set

- **Nested Containers:**

- vector < vector<int> >
- map < int , vector <int> >
- set < pair<int, string> >
- vector < map<int, set<int> >

2. Iterators:

Iterators are used to point at the memory addresses of STL containers. They are primarily used in sequences of numbers, characters etc. They reduce the complexity and execution time of the program.

Operations of iterators :-

1. begin() :- This function is used to return the beginning position of the container.

2. end() :- This function is used to return the after end position of the container.

3. Algorithms:

The header algorithm defines a collection of functions specially designed to be used on a range of elements. They act on containers and provide means for various operations for the contents of the containers.

Algorithm

- upper bound
- Lower bound
- Sorting (Comparator)
- Searching
- max_element
- min_element
- accumulate
- reverse
- count
- find
- next permutations
- prev-permutations

4. Functors:

The STL includes classes that overload the function call operator. Instances of such classes are called function objects or functors. Functors allow the working of the associated function to be customized with the help of parameters to be passed.

2. Pairs

In C++, the term "pairs" usually refers to the std::pair template class provided by the Standard Template Library (STL). A std::pair is a simple container class that can hold two values or objects of potentially different types. It is often used when you need to associate two values together, such as key-value pairs in a map or dictionary.

- **What will be the output of the following code?**

```
#include<bits/stdc++.h>
using namespace std;

int main(){
    pair<int, string> MyPair;
    MyPair = make_pair(2, "Hamim");
    cout<<MyPair.first<<" "<<MyPair.second<<endl;
}
```

Output:
2 Hamim

or,

```
#include<bits/stdc++.h>
using namespace std;

int main(){
    pair<int, string> MyPair;
    MyPair = {2, "Hamim"};
    cout<<MyPair.first<<" "<<MyPair.second<<endl;
}
```

Output:

2 Hamim

- **What will be the output of the following code?**

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

int main(){
  pair<int, string> pr = {2, "Hridi"};
  cout<<pr.first<<" "<<pr.second;
}
```

Output:
2 Hridi

- **What will be the output of the following code?**

```
#include<bits/stdc++.h>
using namespace std;

int main(){
    pair<int, string> MyPair;
    MyPair = {2, "Hamim"};
    pair<int, string> p1 = MyPair;

    cout<<p1.first<<" "<<p1.second<<endl;
}
```

Output:
2 Hamim

- **What will be the output of the following code?**

```
#include<bits/stdc++.h>
using namespace std;

int main(){
  pair<int, string> MyPair;
  MyPair = {2, "Hamim"};
  pair<int, string> p1 = MyPair;

  cout<<MyPair.first<<" "<<MyPair.second<<endl;
  cout<<p1.first<<" "<<p1.second<<endl;
}
```

Output:
2 Hamim
2 Hamim

- **What will be the output of the following code?**

```
#include<bits/stdc++.h>
using namespace std;

int main(){
  pair<int, string> MyPair;
  MyPair = {2, "Hamim"};
  pair<int, string> &p1 = MyPair;
  p1.first = 3;
  cout<<MyPair.first<<" "<<MyPair.second<<endl;
}
```

Output:
3 Hamim

- **What will be the output of the following code?**

```
#include<bits/stdc++.h>
using namespace std;

int main(){
  pair<int, int> P_array[3];
  P_array[0] = {1,2};
  P_array[1] = {2,3};
  P_array[2] = {3,4};

  for(int i=0; i<3; ++i){
    cout<<P_array[i].first<<" "<<P_array[i].second<<endl;
  }
}
```

Output:
1 2
2 3
3 4

- **What will be the output of the following code?**

```
#include<bits/stdc++.h>
using namespace std;

int main(){
  pair<int, int> P_array[3];
  P_array[0] = {1,2};
  P_array[1] = {2,3};
  P_array[2] = {3,4};
  cout<<"Before swap:"<<endl;
  for(int i=0; i<3; ++i){
    cout<<P_array[i].first<<" "<<P_array[i].second<<endl;
  }
  swap(P_array[0], P_array[2]);
  cout<<"After swap:"<<endl;
  for(int i=0; i<3; ++i){
    cout<<P_array[i].first<<" "<<P_array[i].second<<endl;
  }
}
```

Output:

```
Before swap:
1 2
2 3
3 4
After swap:
3 4
2 3
1 2
```

- **What will be the output of the following code?**

```
#include<bits/stdc++.h>
using namespace std;

int main(){
  pair<int, int> P_array[3];
  cin>>P_array[0].first;
  cout<<P_array[0].first;
}
```

Output:

5
5

3. Vectors

In C++, a vector refers to the std::vector container provided by the Standard Template Library (STL). It is a dynamic array-like data structure that can dynamically grow or shrink in size. Vectors are part of the C++ Standard Library and are widely used for storing and managing collections of data in a flexible and efficient manner.

- **Here's a basic example of how to use std::vector in C++:**

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    // Declare a vector of integers
    vector<int> myVector;

    // Add elements to the vector
    myVector.push_back(1);
    myVector.push_back(2);
    myVector.push_back(3);

    // Access elements by index
    cout << "Element at index 0: " << myVector[0]
<< endl;
```

```
    cout << "Element at index 1: " << myVector[1]
<< endl;
    cout << "Element at index 2: " << myVector[2]
<< endl;

    // Get the size of the vector
    cout << "Vector size: " << myVector.size() <<
endl;

    return 0;
}
```

Output:

```
Element at index 0: 1
Element at index 1: 2
Element at index 2: 3
Vector size: 3
```

- **Here's a basic example of how to use std::vector in C++:**

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
   vector<int> myVector(5);

   myVector.push_back(1);

   for(int i=0; i<myVector.size(); i++){
      cout << "Element at index "<<i<<": "
<<myVector[i] << endl;
   }

   cout << "Vector size: " << myVector.size() <<
endl;

   return 0;
}
```

Output:

```
Element at index 0: 0
Element at index 1: 0
Element at index 2: 0
Element at index 3: 0
Element at index 4: 0
Element at index 5: 1
Vector size: 6
```

- **Here's a basic example of how to use std::vector in C++:**

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  vector<int> myVector(5, 3);

  myVector.push_back(1);

  for(int i=0; i<myVector.size(); i++){
    cout << "Element at index "<<i<<": "
<<myVector[i] << endl;
  }

  cout << "Vector size: " << myVector.size() <<
endl;

  return 0;
}
```

Output:

```
Element at index 0: 3
Element at index 1: 3
Element at index 2: 3
Element at index 3: 3
Element at index 4: 3
Element at index 5: 1
Vector size: 6
```

Here's a list of some commonly used functions and methods of std::vector in C++, along with examples for each:

1. **push_back(element):** Add an element to the end of the vector.
2. **pop_back()**: Remove the last element from the vector.
3. **at(index):** Access an element at a specific index with bounds checking.
4. **size():** Get the number of elements in the vector.
5. **clear():** Remove all elements from the vector.
6. **empty()**: Check if the vector is empty.

1. push_back(element):

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

int main() {
    vector<int> myVector;

    myVector.push_back(1);
    myVector.push_back(2);
    myVector.push_back(3);

    for(int i=0; i<myVector.size(); i++){
        cout<<myVector[i]<<" ";
    }

    return 0;
}
```

Output:

1 2 3

2. pop_back():

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

int main() {
  vector<int> myVector = {1, 2, 3};

  cout<<"Before pop: ";
  for(int i=0; i<myVector.size(); i++){
    cout<<myVector[i]<<" ";
  }
  cout<<endl;

  myVector.pop_back();

  cout<<"After pop: ";
  for(int i=0; i<myVector.size(); i++){
    cout<<myVector[i]<<" ";
  }

  return 0;
}
```

Output:
Before pop: 1 2 3
After pop: 1 2

3. at(index):

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

int main()
{
  vector<int> myVector = {1, 2, 3};

  int element = myVector.at(1);
  cout << "Element at index 1: " << element <<
endl;

  myVector.at(1) = 4;
  element = myVector.at(1);
  cout << "Element at index 1: " << element <<
endl;

  element = myVector.at(2);
  cout << "Element at index 2: " << element <<
endl;

  myVector.at(2) = 5;
  element = myVector.at(2);
  cout << "Element at index 2: " << element <<
endl;

  element = myVector.at(0);
  cout << "Element at index 0: " << element <<
endl;
```

```
  myVector[0] = 7;
  element = myVector.at(0);
  cout << "Element at index 0: " << element << endl;

  return 0;
}
```

Output:

```
Element at index 1: 2
Element at index 1: 4
Element at index 2: 3
Element at index 2: 5
Element at index 0: 1
Element at index 0: 7
```

4. size():

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

int main() {
  vector<int> myVector = {1, 2, 3};

  int size = myVector.size();

  cout << "Vector size: " << size << endl;
  return 0;
}
```

Output:
Vector size: 3

5. clear():

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

int main()
{
  vector<int> myVector = {1, 2, 3};

  cout << "Before clear vector: ";
  for (int i = 0; i < myVector.size(); i++)
  {
    cout << myVector[i] << " ";
  }
  cout<<endl;

  myVector.clear();

  cout << "After clear vector: ";

  for (int i = 0; i < myVector.size(); i++)
  {
    cout << myVector[i] << " ";
  }

  return 0;
}
```

Output:

```
Before clear vector: 1 2 3
After clear vector:
```

6. empty():

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

int main() {
  vector<int> myVector;

  bool isEmpty = myVector.empty();

  cout << "Is the vector empty? " << (isEmpty ?
"Yes" : "No") << endl;
  return 0;
}
```

Output:
Is the vector empty? Yes

- **What will be the output of the following code?**

```
#include<bits/stdc++.h>
using namespace std;

int main(){
  vector<int> nums;
  int n;
  cin>>n;
  for(int i=0; i<n; i++){
    int x;
    cin>>x;
    nums.push_back(x); // O(1)
  }

  for(int i=0; i<n; i++){
    cout<<nums[i]<<" ";
  }
}
```

Output:

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

- **What will be the output of the following code?**

```
#include<bits/stdc++.h>
using namespace std;

void printVector(vector<int> arr){
  cout<<"[";
  for(int i=0; i<arr.size(); i++){
    cout<<arr[i];
    if(i+1==arr.size()){
      break;
    }else cout<<", ";
  }
  cout<<"]"<<endl;
}

int main(){
  vector<int> nums;
  int n;
  cout<<"Enter number of elements: ";
  cin>>n;
  for(int i=0; i<n; i++){
    int x;
    cin>>x;
 nums.push_back(x); // O(1)
 }

 int size = nums.size(); // O(1)
 cout<<"Size: "<<size<<endl;

 printVector(nums);
}
```

Output:

```
Enter number of elements: 5
1 2 3 4 5
Size: 5
[1, 2, 3, 4, 5]
```

- **What will be the output of the following code?**

```
#include<bits/stdc++.h>
using namespace std;

void printVector(vector<int> arr){
  cout<<"Size: "<<arr.size()<<endl;
  cout<<"[";
  for(int i=0; i<arr.size(); i++){
    cout<<arr[i];
    if(i+1==arr.size()){
      break;
    }else cout<<", ";
  }
  cout<<"]"<<endl;
}

int main(){
  vector<int> nums(5); // corrent vector size = 5

  printVector(nums);
}
```

Output:

```
Size: 5
[0, 0, 0, 0, 0]
```

- **What will be the output of the following code?**

```
#include<bits/stdc++.h>
using namespace std;

void printVector(vector<int> arr){
  cout<<"Size: "<<arr.size()<<endl;
  cout<<"[";
  for(int i=0; i<arr.size(); i++){
    cout<<arr[i];
    if(i+1==arr.size()){
      break;
    }else cout<<", ";
  }
  cout<<"]"<<endl;
}

int main(){
  vector<int> nums(5); // corrent vector size =5
  nums.push_back(7);
  printVector(nums);
}
```

Output:

```
Size: 6
[0, 0, 0, 0, 0, 7]
```

- **What will be the output of the following code?**

```
#include<bits/stdc++.h>
using namespace std;

void printVector(vector<int> arr){
  cout<<"Size: "<<arr.size()<<endl;
  cout<<"[";
  for(int i=0; i<arr.size(); i++){
    cout<<arr[i];
    if(i+1==arr.size()){
      break;
    }else cout<<", ";
  }
  cout<<"]"<<endl;
}

int main(){
  vector<int> nums(10, 3);
  nums.push_back(7);
  printVector(nums);
}
```

Output:

```
Size: 11
[3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 7]
```

- **What will be the output of the following code?**

```
#include<bits/stdc++.h>
using namespace std;

void printVector(vector<int> arr){
  cout<<"Size: "<<arr.size()<<endl;
  cout<<"[";
  for(int i=0; i<arr.size(); i++){
    cout<<arr[i];
    if(i+1==arr.size()){
      break;
    }else cout<<", ";
  }
  cout<<"]"<<endl;
}

int main(){
  vector<int> nums(10, 3);
  cout<<"Before pop back: ";
  nums.push_back(7);
  printVector(nums);
  cout<<"After pop back: ";
  nums.pop_back();  // O(1)
  printVector(nums);
}
```

Output:

```
Before pop back: Size: 11
[3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 7]
After pop back: Size: 10
[3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
```

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

void printVector(vector<int> arr)
{
  cout << "[";
  for (int i = 0; i < arr.size(); i++)
  {
    cout << arr[i];
    if (i + 1 == arr.size())
    {
      break;
    }
    else
      cout << ", ";
  }
  cout << "]" << endl;
}

int main()
{
  vector<int> nums;
  int n;
  cout << "Enter number of elements: ";
  cin >> n;
  for (int i = 0; i < n; i++)
  {
    int x;
```

```
    cin >> x;
    nums.push_back(x); // O(1)
  }

  int size = nums.size(); // O(1)
  cout << "Size: " << size << endl;
  cout<<"nums = ";
  printVector(nums);
  vector<int> v1 = nums;  // O(n)
  cout << "After copying nums elements to another
vector v1: "<<endl;
  cout<<"v1 = ";
  printVector(v1);
}
```

Output:

```
Enter number of elements: 5
1 2 3 4 5
Size: 5
nums = [1, 2, 3, 4, 5]
After copying nums elements to another vector v1:
v1 = [1, 2, 3, 4, 5]
```

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

void printVector(vector<int> arr)
{
  cout << "Size: " << arr.size() << endl;
  cout << "[";
  for (int i = 0; i < arr.size(); i++)
  {
    cout << arr[i];
    if (i + 1 == arr.size())
    {
      break;
    }
    else
      cout << ", ";
  }
  arr.push_back(2);
  cout << "]" << endl;
}

int main()
{
  vector<int> nums;
   nums.push_back(7);
   nums.push_back(6);
```

```
  vector<int> v1 = nums;
  v1.push_back(5);
  printVector(nums);
  printVector(nums);
  printVector(v1);
}
```

Output:

```
Size: 2
[7, 6]
Size: 2
[7, 6]
Size: 3
[7, 6, 5]
```

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

void printVector(vector<int> &arr)
{
 cout << "Size: " << arr.size() << endl;
 cout << "[";
 for (int i = 0; i < arr.size(); i++)
 {
  cout << arr[i];
  if (i + 1 == arr.size())
  {
   break;
  }
  else
   cout << ", ";
 }
 cout << "]" << endl;
}

int main()
{
 vector<int> nums;
  nums.push_back(7);
  nums.push_back(6);

  printVector(nums);
```

```
 vector<int> v1 = nums;
 printVector(v1);
}
```

Output:

```
Size: 2
[7, 6]
Size: 2
[7, 6]
```

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

void printVector(vector<int> &arr)
{
  cout << "Size: " << arr.size() << endl;
  cout << "[";
  for (int i = 0; i < arr.size(); i++)
  {
    cout << arr[i];
    if (i + 1 == arr.size())
    {
      break;
    }
    else
      cout << ", ";
  }
  cout << "]" << endl;
}

int main()
{
  vector<int> nums;
   nums.push_back(7);
   nums.push_back(6);

   printVector(nums);
```

```
    vector<int> &v1 = nums;
    v1.pop_back();
    printVector(nums);
}
```

Output:

```
Size: 2
[7, 6]
Size: 1
[7]
```

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

void printVector(vector<string> arr)
{
  cout << "[";
  for (int i = 0; i < arr.size(); i++)
  {
    cout << arr[i];
    if (i + 1 == arr.size())
    {
      break;
    }
    else
      cout << ", ";
  }
  cout << "]" << endl;
}

int main()
{
  vector<string> names;
  int n;
  cout << "Enter number of elements: ";
  cin >> n;
  for (int i = 0; i < n; i++)
  {
    string s;
```

```
    cin>>s;
    names.push_back(s); // O(1)
  }

  printVector(names);
}
```

Output:

```
Enter number of elements: 5
Hamim
Jim
Mithu
Hridi
Shanta
[Hamim, Jim, Mithu, Hridi, Shanta]
```

- **Nesting In Vectors:**

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

void printVector(vector< pair<int, int> > arr)
{
  cout << "[";
  for (int i = 0; i < arr.size(); i++)
  {
    cout << "{" <<arr[i].first<<" "<<arr[i].second<<
"}";
    if (i + 1 == arr.size())
    {
      break;
    }
    else
      cout << ", ";
  }
  cout << "]" << endl;
}
```

```
int main()
{
  vector<pair<int, int>> myVector = {{1,2}, {2,3}, {4,5}};

  printVector(myVector);
}
```

Output:

```
[{1 2}, {2 3}, {4 5}]
```

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  vector<pair<int, int>> nums;
  nums.push_back({2, 3});
  nums.push_back({4, 5});

  cout << "Size: " << nums.size() << endl;
  cout << "[";
  for (int i = 0; i < nums.size(); i++)
  {
    cout << nums[i].first << " " << nums[i].second;
    if (i + 1 == nums.size())
    {
      break;
    }
    else
      cout << ", ";
  }
  cout << "]" << endl;
}
```

Output:

```
Size: 2
[2  3, 4  5]
```

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

void printVector(vector< pair<int, int> > arr)
{
  cout << "[";
  for (int i = 0; i < arr.size(); i++)
  {
    cout << "{" <<arr[i].first<<" "<<arr[i].second<<
"}";
    if (i + 1 == arr.size())
    {
      break;
    }
    else
      cout << ", ";
  }
  cout << "]" << endl;
}

int main()
{
  vector<pair<int, int>> myVector = {{1,2}, {2,3},
{4,5}};
  printVector(myVector);
  int n;
  cout<<"Enter number of pairs: ";
  cin>>n;
```

```
  for(int i=0; i<n;++i){
    int x, y;
    cin >> x >> y;
    myVector.push_back({x,y});
  }
  printVector(myVector);
}
```

Output:

```
[{1 2}, {2 3}, {4 5}]
Enter number of pairs: 3
7 8
3 44
12 10
[{1 2}, {2 3}, {4 5}, {7 8}, {3 44}, {12 10}]
```

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

void printVector(vector< pair<int, int> > arr)
{
  cout << "[";
  for (int i = 0; i < arr.size(); i++)
  {
    cout << "{" <<arr[i].first<<" "<<arr[i].second<<
"}";
    if (i + 1 == arr.size())
    {
      break;
    }
    else
      cout << ", ";
  }
  cout << "]" << endl;
}

int main()
{
  vector<pair<int, int>> myVector = {{1,2}, {2,3},
{4,5}};
  printVector(myVector);
  int n;
  cout<<"Enter number of pairs: ";
  cin>>n;
```

```
  for(int i=0; i<n;++i){
    int x, y;
    cin >> x >> y;
    myVector.push_back(make_pair(x,y));
  }
  printVector(myVector);
}
```

Output:

```
[{1 2}, {2 3}, {4 5}]
Enter number of pairs: 3
7 8
3 44
12 10
[{1 2}, {2 3}, {4 5}, {7 8}, {3 44}, {12 10}]
```

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

void printVector(vector<int> arr)
{
  cout << "Size: " << arr.size() << endl;
  for (int i = 0; i < arr.size(); i++)
  {
    cout << arr[i] <<" ";
  }
  cout<<endl;
}

int main()
{
  int N;
  cin >> N;
  vector<int> v[N];
  for (int i = 0; i < N; ++i)
  {
    int n;
    cin >> n;
    for (int j=0; j < n; ++j)
    {
      int x;
      cin >> x;
      v[i].push_back(x);
    }
  }
```

```
  for (int i = 0; i < N; ++i)
  {
    printVector(v[i]);
  }
}
```

Output:

```
3
3
1 2 3
3
3 4 5
2
1 2
Size: 3
1 2 3
Size: 3
3 4 5
Size: 2
1 2
```

or,

```
#include <bits/stdc++.h>
using namespace std;

void printVector(vector<int> arr[])
{
    for (int i = 0; i < arr->size(); i++)
    {
        cout << "Size: " << arr[i].size() << endl;
        for (int j = 0; j < arr[i].size(); j++)
        {
            cout << arr[i][j] << " ";
        }
        cout << endl;
    }
}

int main()
{
    int N;
    cin >> N;
    vector<int> v[N];
    for (int i = 0; i < N; ++i)
    {
        int n;
        cin >> n;
        for (int j = 0; j < n; ++j)
        {
            int x;
            cin >> x;
```

```
            v[i].push_back(x);
        }
    }

    printVector(v);
}
```

Output:

```
3
3
1 2 3
3
3 4 5
2
1 2
Size: 3
1 2 3
Size: 3
3 4 5
Size: 2
1 2
```

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

void printVector(vector<int> arr)
{
  cout << "Size: " << arr.size() << endl;
  for (int i = 0; i < arr.size(); i++)
  {
    cout << arr[i] << " ";
  }
  cout << endl;
}

int main()
{
  int N;
  cin >> N;
  vector<vector<int>> v;
  for (int i = 0; i < N; ++i)
  {
    int n;
    cin >> n;
    vector<int> temp;
    for (int j = 0; j < n; ++j)
    {
      int x;
      cin >> x;
```

```
      temp.push_back(x);
    }
    v.push_back(temp);
  }

  for (int i = 0; i < v.size(); ++i)
  {
    printVector(v[i]);
  }
}
```

Output:

```
3
3
1 2 3
3
3 4 5
2
1 2
Size: 3
1 2 3
Size: 3
3 4 5
Size: 2
1 2
```

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

void printVector(vector<int> arr)
{
  cout << "Size: " << arr.size() << endl;
  for (int i = 0; i < arr.size(); i++)
  {
    cout << arr[i] << " ";
  }
  cout << endl;
}

int main()
{
  int N;
  cin >> N;
  vector<vector<int>> v;
  for (int i = 0; i < N; ++i)
  {
    int n;
    cin >> n;
    vector<int> temp;
    for (int j = 0; j < n; ++j)
    {
      int x;
      cin >> x;
```

```
      temp.push_back(x);
    }
    v.push_back(temp);
  }

  v[0].push_back(10);
  v.push_back(vector<int> ());
  v.push_back(vector<int> (11));
  v.push_back(vector<int> (11,3));

  for (int i = 0; i < v.size(); ++i)
  {
    printVector(v[i]);
  }
}
```

Output:

3
3
1 2 3
3
3 4 5
2
1 2
Size: 4
1 2 3 10
Size: 3
3 4 5
Size: 2
1 2
Size: 0

Size: 11
0 0 0 0 0 0 0 0 0 0 0
Size: 11
3 3 3 3 3 3 3 3 3 3 3

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

void printVector(vector<int> arr)
{
  cout << "Size: " << arr.size() << endl;
  for (int i = 0; i < arr.size(); i++)
  {
    cout << arr[i] << " ";
  }
  cout << endl;
}

int main()
{
  int N;
  cin >> N;
  vector<vector<int>> v;
  for (int i = 0; i < N; ++i)
  {
    int n;
    cin >> n;
    v.push_back(vector<int> ());
    for (int j = 0; j < n; ++j)
    {
      int x;
      cin >> x;
      v[i].push_back(x);
```

```
    }
  }

  for (int i = 0; i < v.size(); ++i)
  {
    printVector(v[i]);
  }
}
```

Output:

```
3
3
1 2 3
3
3 4 5
2
1 2
Size: 3
1 2 3
Size: 3
3 4 5
Size: 2
1 2
```

4. ITERATORS

Pointer like structure in C++ STL

In C++, iterators are objects that allow you to iterate (traverse) through the elements of a container, such as an array, vector, list, or any other sequence-based container. Iterators provide a way to access the elements of a container without exposing the underlying details of the container's implementation.

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
   vector<int> v = {2,3,4,5,6};
   for(int i=0; i<v.size(); i++){
      cout<< v[i] << " ";
   }

   cout<<endl;

   vector<int> ::iterator it = v.begin();

   cout<<"in index 0 : "<<(*(it+0))<<endl;
   cout<<"in index 1 : "<<(*(it+1))<<endl;
   cout<<"in index 2 : "<<(*(it+2))<<endl;
   cout<<"in index 3 : "<<(*(it+3))<<endl;
   cout<<"in index 5 : "<<(*(it+4))<<endl;
}
```

Output:

```
2 3 4 5 6
in index 0 : 2
in index 1 : 3
in index 2 : 4
in index 3 : 5
in index 5 : 6
```

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  vector<int> v = {2,3,4,5,6};
  for(int i=0; i<v.size(); i++){
    cout<< v[i] << " ";
  }

  cout<<endl;

  vector<int> ::iterator it = v.begin();
  vector<int> ::iterator ite = v.end();

  cout<<"First index element : "<<(*it)<<endl;

  cout<<"Last index element : "<<(*(ite-1))<<endl;

}
```

Output:
2 3 4 5 6
First index element : 2
Last index element : 6

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  vector<int> v = {2,3,4,5,6};
  for(int i=0; i<v.size(); i++){
    cout<< v[i] << " ";
  }

  cout<<endl;

  vector<int> ::iterator it;

  for(it = v.begin(); it != v.end(); ++it){
    cout<<(*it)<<" ";
  }
}
```

Output:

2 3 4 5 6
2 3 4 5 6

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
 vector<int> v = {2,3,4,5,6};
 for(int i=0; i<v.size(); i++){
  cout<< v[i] << " ";
 }

 cout<<endl;

 //auto is a Dynamic data type
 for(auto it = v.begin(); it != v.end(); ++it){
  cout<<(*it)<<" ";
 }

}
```

Output:
2 3 4 5 6
2 3 4 5 6

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
 vector<pair<int, int>> v_p = {{1,2},{2,3},{3,4}};
 vector<pair<int, int>> :: iterator it;

 for(it = v_p.begin(); it != v_p.end(); ++it){
  cout<<"{"<<(*it).first<<", "<<(*it).second<<"} ";
 }
}
```

Output:

{1, 2} {2, 3} {3, 4}

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
 vector<pair<int, int>> v_p = {{1,2},{2,3},{3,4}};

 for(auto it = v_p.begin(); it != v_p.end(); ++it){
 cout<<"{"<<(*it).first<<", "<<(*it).second<<"} ";
 }
}
```

Output:

{1, 2} {2, 3} {3, 4}

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
 vector<pair<int, int>> v_p = {{1,2},{2,3},{3,4}};
 vector<pair<int, int>> :: iterator it;

 for(it = v_p.begin(); it != v_p.end(); ++it){
  cout<<"{"<<(it->first)<<", "<<(it->second)<<"} ";
 }
}
```

Output:
{1, 2} {2, 3} {3, 4}

Here,
(*it).first == (it->first)

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
 vector<int> v = {2,3,4,5,6};
 for(int i=0; i<v.size(); i++){
  cout<< v[i] << " ";
 }

 cout<<endl;

 vector<int> ::iterator it;

 for(it = v.begin(); it != v.end(); ++it){
  cout<<(*it)<<" ";
 }

  cout<<endl;

 //using foreach loop
 for(int value : v){
  cout<<value<<" ";
 }
}
```

Output:

```
2 3 4 5 6
2 3 4 5 6
2 3 4 5 6
```

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
 vector<int> v = {2,3,4,5,6};
 for(int i=0; i<v.size(); i++){
 cout<< v[i] << " ";
 }

 cout<<endl;

 //using foreach loop
 for(auto value : v){
 cout<<value<<" ";
 }
}
```

Output:

```
2 3 4 5 6
2 3 4 5 6
```

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
 vector<int> v = {2,3,4,5,6};
 for(int i=0; i<v.size(); i++){
  cout<< v[i] << " ";
 }

 cout<<endl;

 vector<int> ::iterator it;

 for(it = v.begin(); it != v.end(); ++it){
  cout<<(*it)<<" ";
 }

  cout<<endl;

 //using foreach loop
 for(int &value : v){
  value++;
  cout<<value<<" ";
 }
}
```

Output:

2 3 4 5 6
2 3 4 5 6
3 4 5 6 7

5. Maps

In C++, std::map is a part of the Standard Template Library (STL) and is used to implement an associative container that stores key-value pairs. It is also known as a sorted associative container, as the elements in the map are always sorted based on the keys.

Maps:

1. Ordered MAP
2. Unordered MAP
3. Multimap

1. Ordered MAP :

In this map data are stored in orderly.
It uses the Re-Black Tree data structure.

Insertion and deletion Time complexity:
O(logN)

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  map<int, string> m;
  //[int] string
  m[1] = "abc";
  m[5] = "cdc";
  m[3] = "acd";

  cout<<"map size : "<<m.size()<<endl;

  map<int, string> :: iterator it;

  for(it = m.begin(); it!=m.end(); it++){
    cout<<(*it).first<<" "<<(*it).second<<endl;
  }
}
```

Output:

```
map size : 3
1  abc
3  acd
5  cdc
```

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    map<int, string> m;
    //[int]   string
    m.insert({1, "abc"});
    m.insert({5, "cdc"});
    m.insert({3, "acd"});
    m.insert({4, "afg"});

    map<int, string> :: iterator it;

    it = m.find(1); //O(logN)
    cout<<(*it).first<<" "<<(*it).second<<endl;

    it = m.find(3);
    if(it == m.end()){
        cout<<"No value"<<endl;
    }else{
        cout<<(*it).first<<" "<<(*it).second<<endl;
    }

    it = m.find(9);
    if(it == m.end()){
        cout<<"No value"<<endl;
```

```
    }else{
        cout<<(*it).first<<" "<<(*it).second<<endl;
    }
}
```

Output:

```
1 abc
3 acd
No value
```

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  map<int, string> m;
  //[int] string
  m[1] = "ABC"; //O(logN)
  m[5] = "cdc";
  m[3] = "acd";
  m.insert({4, "afg"});

  map<int, string> :: iterator it;

  for(it = m.begin(); it!=m.end(); it++){
    cout<<(*it).first<<" "<<(*it).second<<endl;
  }
}
```

Output:

```
1  abc
3  acd
4  afg
5  cdc
```

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  map<int, string> m;
  //[int] string
  m[1] = "abc";
  m[5] = "cdc";
  m[3] = "acd";

  map<int, string> :: iterator it;

  for(it = m.begin(); it!=m.end(); it++){
    cout<<(*it).first<<" "<<(*it).second<<endl;
  }
  cout<<endl;

  m.erase(5);
  for(it = m.begin(); it!=m.end(); it++){
    cout<<(*it).first<<" "<<(*it).second<<endl;
  }
}
```

Output:

```
1  abc
3  acd
5  cdc

1  abc
3  acd
```

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  map<int, string> m;

  m.insert({1, "abc"});
  m.insert({5, "cdc"});
  m.insert({3, "acd"});
  m.insert({4, "afg"});

  map<int, string> :: iterator it;

  for(it = m.begin(); it!=m.end(); it++){
    cout<<(*it).first<<" "<<(*it).second<<endl;
  }
}
```

Output:

```
1  abc
3  acd
4  afg
5  cdc
```

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  map<int, string> m;
  //[int] string
  m[1] = "abc";
  m[5] = "cdc";
  m[3] = "acd";

  cout<<"map size : "<<m.size()<<endl;

  map<int, string> :: iterator it;

  for(it = m.begin(); it!=m.end(); it++){
    cout<<(*it).first<<" "<<(*it).second<<endl;
  }

  m.clear();
  cout<<"After clear the map : "<<endl;
  cout<<"map size : "<<m.size()<<endl;
  for(it = m.begin(); it!=m.end(); it++){
    cout<<(*it).first<<" "<<(*it).second<<endl;
  }
}
```

Output:

```
map size : 3
1  abc
3  acd
5  cdc
After clear the map :
map size : 0
```

2. Unordered MAP :

In this map data are stored in unorderly.

It uses the Hashing technique.

All functions are same as ordered map.

Insertion and deletion Time complexity:
O(1)

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
 unordered_map<int, string> m;
 //[int] string
 m[1] = "abc";
 m[5] = "cdc";
 m[3] = "acd";

 cout<<"unordered_map size : "<<m.size()<<endl;

 unordered_map<int, string> :: iterator it;

 for(it = m.begin(); it!=m.end(); it++){
  cout<<(*it).first<<" "<<(*it).second<<endl;
 }
}
```

Output:

```
map size : 3
3 acd
5 cdc
1 abc
```

6. SET

In C++, std::set is another container provided by the Standard Template Library (STL). It is an associative container that contains a sorted set of unique elements. The elements in a set are always sorted in ascending order, and each element must be unique.

set:

1. Ordered set
2. Unordered set
3. Multiset

1. Ordered SET :

In this set data are stored in orderly.

It uses the Red-Black Tree.

Insertion and deletion Time complexity:
O(logN)

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    set<string> s;
    s.insert("Hamim"); // O(logN)
    s.insert("Jim");
    s.insert("Rahim");

    set<string> :: iterator it=s.find("Hamim"); //
O(logN)
    if(it != s.end()){
        cout<<(*it)<<endl;
    } else{
        cout<<"Not found"<<endl;
    }
}
```

Output:
Hamim

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  set<string> s;
  s.insert("Hamim");
  s.insert("Jim");
  s.insert("Rahim");

  set<string> :: iterator it;

  for(it = s.find("Hamim"); it != s.end(); it++){
    cout<<(*it)<<" ";
  }
}
```

Output:
Hamim Jim Rahim

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  set<string> s;
  s.insert("Hamim"); // O(logN)
  s.insert("Jim");
  s.insert("Rahim");

  for (auto it = s.find("Hamim"); it != s.end(); it++){
    cout << (*it) << " ";
  }

}
```

Output:
Hamim Jim Rahim

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  set<string> s;
  s.insert("Hamim"); // O(logN)
  s.insert("Jim");
  s.insert("Rahim");

  set<string>::iterator it = s.find("Hamim"); //
O(logN)
  for (it = s.find("Hamim"); it != s.end(); it++){
    cout << (*it) << " ";
  }
  cout<<endl;
  s.erase("Jim");
  for (it = s.find("Hamim"); it != s.end(); it++){
    cout << (*it) << " ";
  }
}
```

Output:
Hamim Jim Rahim
Hamim Rahim

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  set<string> s;
  s.insert("Hamim"); // O(logN)
  s.insert("Jim");
  s.insert("Rahim");

  set<string>::iterator it = s.find("Hamim"); //
O(logN)
  for (it = s.find("Hamim"); it != s.end(); it++){
    cout << (*it) << " ";
  }
  cout<<endl;
  it = s.find("Hamim");
  if (it != s.end())
  {
    cout << (*it) << " deleted" << endl;
    s.erase(it);
  }
  else
  {
    cout << "Not found" << endl;
  }
```

```
  for (it = s.find("Jim"); it != s.end(); it++){
    cout << (*it) << " ";
  }
}
```

Output:

Hamim Jim Rahim
Hamim deleted
Jim Rahim

2. Unordered SET :

In this set data are stored in unorderly.

It uses the Hashing technique.

All functions are same as ordered set.

Insertion and deletion Time complexity:
O(1)

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  unordered_set<string> s;
  s.insert("abc"); // O(1)
  s.insert("fgh");
  s.insert("tws");

  unordered_set<string>::iterator it =
s.find("tws");
  cout << (*it) << endl;
}
```

Output:
tws

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  unordered_set<string> s;
  s.insert("abc"); // O(1)
  s.insert("fgh");
  s.insert("tws");

  int i=0;
  while (i++<s.size()){
    string str;
    cin>>str;
    if(s.find(str) == s.end()){
      cout<<"No\n";
    }else{
      cout<<"Yes\n";
    }
  }
}
```

Output:

```
abc
Yes
poc
No
tws
Yes
```

3. Multiset :

In this set data are stored in orderly.

In this set, we can store multiple duplicate values.

All functions are same as ordered set.

Insertion and deletion Time complexity:
O(logN)

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
 multiset<string> s;
 s.insert("abc"); //O(logN)
 s.insert("zsd");
 s.insert("bcdf");
 s.insert("abc");

 multiset<string> :: iterator it;

 for(it = s.find("abc"); it != s.end(); it++){
 cout<<(*it)<<" ";
 }
}
```

Output:
abc abc bcdf zsd

- **What will be the output of the following code?**

```
#include <bits/stdc++.h>
using namespace std;

void print(multiset<string> &s){
  for(string value : s){
    cout<<value<<endl;
  }
}

int main()
{
 multiset<string> s;
 s.insert("abc");
 s.insert("zsd");
 s.insert("bcdf");
 s.insert("abc");

 multiset<string> :: iterator it=s.find("abc");

  // deleting only one “abc”
 if(it != s.end()){
  s.erase(it);
 }

 print(s);
}
```

Output:

```
abc
bcdf
zsd
```

- **What will be the output of the following code?**

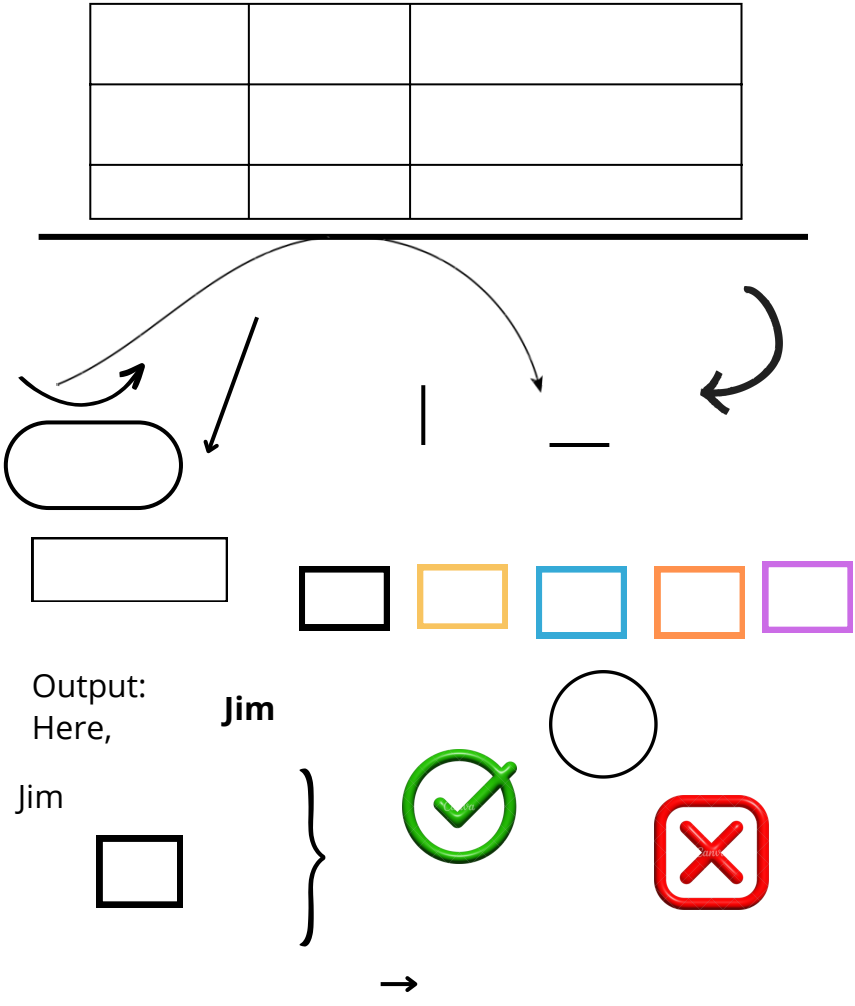
```
#include <bits/stdc++.h>
using namespace std;

void print(multiset<string> &s){
  for(string value : s){
    cout<<value<<endl;
  }
}

int main()
{
 multiset<string> s;
 s.insert("abc");
 s.insert("zsd");
 s.insert("bcdf");
 s.insert("abc");

  //deleting all “abc”
 s.erase("abc");

 print(s);
}
```

Output:

```
bcdf
zsd
```

Operator Precedence and Associativity:

- **What will be the output of the following code?**



Output:
Here,

Jim

Jim

- **What is the output of the following Java program fragment:**

Operator

C++
(SEVEN PART)
T.I.M. HAMIM

ABC
PROKASHONI