Lesson:

Arrays – 2

PW SKILLS

Pre-Requisites

- Loops
- Functions
- Basics of arrays

List of Concepts Involved

- Vectors in C++
- Basic STL(Standard Template Library) operations
- Looping in Vector
- Problems in vector

Since we have come this far in arrays, it is worth mentioning here that there is a provision in C++ wherein we can store the elements/data dynamically and sequentially. That means the length of this container is dynamic. These are called vectors ! Let's learn about them.

Topic: Vectors in C++

Vectors are dynamic arrays that can be resized whenever an element is inserted or deleted. Vectors also have contiguous storage like arrays but their size can change dynamically according to the requirements of user. Another important point to note here is that vectors are not compulsorily bound by index values.

Declaration Syntax:

```
vector < data_type >  name;
```

Example:

```
vector<int> v;
vector<char> a;
```

Note: We can even declare a vector of fixed size also with the following syntax:

```
vector<int> v(n);
```

Here, a vector of size n is declared.

Topic: Basic Operations on Vectors

1. Just like in arrays, it returns the size of vector.

   Example:

   ```
   v = {10,20,30,40,50};
   cout << v.size() << endl;
   ```

   Output :

   ```
   5
   ```

2. Resizing operation in vector

Unlike the fixed size approach of arrays, we can resize the vector according to our requirement.

Example:

```
vector<int> v;
v.resize(n);
```

Note: This is extremely useful when we do not know the size initially (at the time of creating the vector). It can be resized anytime as per need.

Vector capacity() function: It returns the size of the storage space allocated to the vector. This capacity is greater than or equal to the size of the vector catering the need for extra space to allow growth without the need to reallocate on each insertion.

Note: This capacity is not the limit for growth and can be automatically expanded.

Code Example:

```
vector<int> v;
v.resize(10);
cout << v.capacity() << endl;
v.resize(90);
cout << v.capacity() << endl;
```

Output:

```
16
128
```

Note: Whenever we resize and increase the size of the vector, the total capacity gets to the next power of 2 or the remaining memory. For example, in the above case, the vector size is 10 but its capacity is the closest next power of 2 i.e.16. When increased to 90, its capacity went to the closest next power of 2 i.e. 128.

3. Inserting an element at the back of the vector using push_back operation.

Code:

```
vector<int> v = {10,20,30,40};
v.push_back(50);
```

Now the new vector will have values: {10,20,30,40,50}.

4. Popping the last element of the vector using pop_back operation.

Code:

```
vector<int> v = {10,20,30,40,50};
v.pop_back();
```

Now the new vector will have values: {10,20,30,40}.

6. Insert an element somewhere in the vector's middle/ (in between the elements).

Syntax:

```
vector<int> v;
v.insert(position, value);
```

Position specifies to the iterator which points to the position where the insertion is to be done.

Example Code:

```
vector<int> v = {10,20,30,40,50};
v.insert(v.begin() + 2, 100);
```

The new vector will look like this:

```
{10,20,100,30,40,50}
```

7. Deleting an element in a vector at a specific position

Syntax:

```
v.erase(position);
```

Example Code:

```
vector<int> v = {10,20,30,40,50};

v.erase(v.begin() + 2);
```

The new vector will look like this:

```
{10,20,40,50}
```

8. Clearing the vector : clear() function is used to remove all the elements of the vector container, thus making its size 0.

Code:

```
vector<int> v = {10,20,30,40,50};
v.clear();
```

Now v will be empty.

Topic: Looping in Vector

There are many ways to traverse through the elements of a vector. The most common and widely used ways of looping are:

- **For Loop**

```
vector<int> v;
for(int i = 0; i < 5; i++) {
 v.push_back(i);
}
// now by the above loop v will have the values : {0,1,2,3,4}
for(int i = 0; i < v.size(); i++){
 cout << v[i] << " ";
}
```

Output :

```
0 1 2 3 4
```

- **For each loop**

This loop helps in iterating over iterable objects like string, array and so on.

```
vector<int> a = { 1, 2, 3, 4, 5, 6, 7, 8 };
 for (int i : a) {
     // accessing each element of the vector
     cout << i << endl;
}
```

Output:

1
2
3
4
5
6
7
8

Explanation : Here, all the elements present in the array will be printed.

- **While loop**

```
int i = 5;
vector<int> v;
while(i > 0) {
  v.push_back(i--);
}
```

Explanation: **Here, we initialize the value of 'i' to be 5 and keep decrementing it using the post-decrement** operator and at each step insert it at the back of the vector. Therefore, in the end, the vector will contain elements from 5 to 0.

Topic: Basic Problems in Arrays

Example 1: Find the last occurrence of an element x in a given array.
Code :

```
int solve(vector<int> &a, int x)
{
  int index = -1;
  for(int i = 0; i < a.size(); i++) {
    if(a[i] == x)
      index = i;
  }
  return index;
}
```

Explanation: Traverse through the whole vector and compare the current element with the target element 'x' and if it matches then it will be our last seen index.

Example 2: Count the number of occurrences of a particular element x.

Code :

```
int solve(vector<int> &a, int x)
  int count = 0;
  for(int i = 0; i < a.size(); i++) {
    if(a[i] == x)
      count++;
  }
  return count;
}
```

Explanation: Just check if the element is equal to the element x and increment the count variable and in the end return it.

Example 3: Count the number of elements strictly greater than value x.

Code :

```
int solve(vector<int> &a, int x)
  int count = 0;
  for(int i = 0; i < a.size(); i++) {
    if(a[i] > x)
      count++;
  }
  return count;
}
```

Explanation: Traverse the array and just check if the element is greater than x and increment the count variable, in the end just return it.

Example 4: Check if the given array is sorted or not.

Code :

```
bool solve(vector<int> &a)
  bool ans = true;
  for(int i = 1; i < a.size(); i++) {
    if(a[i] < a[i-1])
      ans = false;
  }
  return ans;
}
```

Explanation: Consider the answer to be true, then traverse the array and at each point check if the previous element is greater than the current. The answer will become false as the array will not be sorted.

Example 5: Find the difference between the sum of elements at even indices to the sum of elements at odd indices.

Code :

```
int solve(vector<int> &a)
  int sum = 0;
  for(int i = 0; i < a.size(); i++) {
    if(i % 2 == 0) {
      sum += a[i]; // adding values at even indices
    } else {
      sum -= a[i]; // subtracting values at odd indices
    }
  }
  return sum;
}
```

Explanation: Traverse the vector and add values at even indices and subtract values at the odd indices.

Upcoming Class Teasers:

- Problem Solving in Arrays