**Strings in C++:**

getline() This function is used to store a stream of characters as entered by the user in the object memory.

push\_back() This function is used to input a character at the end of the string.

pop\_back() Introduced from C++11(for strings), this function is used to delete the last character from the string.

resize() This function changes the size of the string, the size can be increased or decreased.

length() This function finds the length of the string.

begin() This function returns an iterator to the beginning of the string.

end() This function returns an iterator to the next to the end of the string.

rbegin() This function returns a reverse iterator pointing at the end of the string.

rend() This function returns a reverse iterator pointing to the previous of beginning of the string.

copy(“char array”, len, pos) This function copies the substring in the target character array mentioned in its arguments. It takes 3 arguments, target char array, length to be copied, and starting position in the string to start copying.

swap() This function swaps one string with another

**Unordered\_map/Hashmap in C++(unordered\_map<string, int> umap;):**

(umap["GeeksforGeeks"] = 10;)

the cost of search, insert, and delete from the hash table is O(1).

at() This function in C++ unordered\_map returns the reference to the value with the element as key k

begin() Returns an iterator pointing to the first element in the container in the unordered\_map container

end() Returns an iterator pointing to the position past the last element in the container in the unordered\_map container

count() basically checks if there exists an element in the unordered\_map with a given key or not.

find() Returns iterator to the element

empty() Checks whether the container is empty in the unordered\_map container

erase() Erase elements in the container in the unordered\_map container

**Maps in C++:**

(map<int, int> gquiz1; interate - map<int, int>::iterator itr;)

gquiz1.insert(pair<int, int>(1, 40));

for (itr = gquiz1.begin(); itr != gquiz1.end(); ++itr) {

// first is for key , second is for value

cout << '\t' << itr->first << '\t' << itr->second<< '\n';

}

begin() – Returns an iterator to the first element in the map.

end() – Returns an iterator to the theoretical element that follows the last element in the map.

size() – Returns the number of elements in the map.

empty() – Returns whether the map is empty.

erase(iterator position) – Removes the element at the position pointed by the iterator.

erase(const g)– Removes the key-value ‘g’ from the map.

clear() – Removes all the elements from the map.

**Queue in C++ (queue<int> g;):**

queue::size() Returns the size of the queue.

queue::swap() Exchange the contents of two queues but the queues must be of the same data type, although sizes may differ.

queue::front() Returns a reference to the first element of the queue.

queue::back() Returns a reference to the last element of the queue.

queue::push(g) Adds the element ‘g’ at the end of the queue.

queue::pop() Deletes the first element of the queue.

**Stack in C++ (stack<int> stack;):**

empty() – Returns whether the stack is empty – Time Complexity : O(1)

size() – Returns the size of the stack – Time Complexity : O(1)

top() – Returns a reference to the top most element of the stack – Time Complexity : O(1)

push(g) – Adds the element ‘g’ at the top of the stack – Time Complexity : O(1)

pop() – Deletes the top most element of the stack – Time Complexity : O(1)

**Vector in C++ :**

// element access:

reference operator [g] – Returns a reference to the element at position ‘g’ in the vector

at(g) – Returns a reference to the element at position ‘g’ in the vector

front() – Returns a reference to the first element in the vector

back() – Returns a reference to the last element in the vector

//Modifyers:

push\_back() – It push the elements into a vector from the back

pop\_back() – It is used to pop or remove elements from a vector from the back.

insert() – It inserts new elements before the element at the specified position. vector\_name.insert (position, val)

erase() – It is used to remove elements from a container from the specified position or range. vectorname.erase(position) , vectorname.erase(startingposition, endingposition)

swap() – It is used to swap the contents of one vector with another vector of same type. Sizes may differ. myvector1.swap(myvector2);

clear() – It is used to remove all the elements of the vector container

//capacity

size() – Returns the number of elements in the vector.

resize(n , val\*\*) – Resizes the container so that it contains ‘n’ elements. and assign val to all new elements(\*\* is optional parameter).

empty() – Returns bool whether the container is empty.

shrink\_to\_fit() – Reduces the capacity of the container to fit its size and destroys all elements beyond the capacity.

reserve() – Requests that the vector capacity be at least enough to contain n elements.reverse(ans.begin(), ans.end());

// Itreator Methods:

begin() – Returns an iterator pointing to the first element in the vector

end() – Returns an iterator pointing to the theoretical element that follows the last element in the vector

rbegin() – Returns a reverse iterator pointing to the last element in the vector (reverse beginning). It moves from last to first element

rend() – Returns a reverse iterator pointing to the theoretical element preceding the first element in the vector (considered as reverse end)

**List in c++ (list<int> gqlist1;):**

front() Returns the value of the first element in the list.

back() Returns the value of the last element in the list.

push\_front(g) Adds a new element ‘g’ at the beginning of the list.

push\_back(g) Adds a new element ‘g’ at the end of the list.

pop\_front() Removes the first element of the list, and reduces size of the list by 1.

pop\_back() Removes the last element of the list, and reduces size of the list by 1.

**Priority Queues:**

(priority\_queue<int> gquiz; , priority\_queue<int, vector<int>, greater<int> > gquiz;)

The major advantage of using a priority queue is that you will be able to quickly access the highest priority item with a time complexity of just O(1).

The only disadvantage of using Priority Queues are that the enqueue and dequeue operations are slow and have a time complexity of O(log n)

g.top(); to get highest priority or lowest priority element based on what type of queue it is (max heap or min heap).

**Case transformation of string:**

transform(s1.begin(), s1.end(), s1.begin(), ::toupper);

toupper(char c): Returns the upper case version of character c. If c is already in uppercase, return c itself. tolower(char c) : Returns lower case version of character c.

**Conversion of an integer into a string by using to\_string() method.**

The to\_string() method accepts a single integer and converts the integer value or other data type value into a string.

**Convert the string str variable to have an int value**

//place the new value in a new variable that holds int values, named num

int num = stoi(str);

The main difference between new and malloc is that new invokes the object's constructor and the corresponding call to delete invokes the object's destructor.