**Assignment 3**

**Module I**

**Q1.a) Create a class INTEGER that contains the one integer data member. Overload all the four arithmetic operators so that they operate on the objects of INTEGER.**

#include<iostream>

using namespace std;

class INTEGER{

int data;

static int count;

public:

INTEGER(){

data=0;

count++;

}

INTEGER operator +(INTEGER x);

INTEGER operator -(INTEGER x);

INTEGER operator \*(INTEGER x);

INTEGER operator /(INTEGER x);

void getdata()

{

cout<<endl<<"Enter data"<<endl;

cin>>this->data;

}

void display()

{

cout<<endl<<count<<" -The data is :"<<this->data<<endl;

}

};

int INTEGER::count=0;

INTEGER INTEGER::operator +(INTEGER x)

{

INTEGER r;

r.data = x.data+this->data;

return r;

}

INTEGER INTEGER::operator -(INTEGER x)

{

INTEGER r;

r.data = this->data-x.data;

return r;

}

INTEGER INTEGER::operator \*(INTEGER x)

{

INTEGER r;

r.data = x.data\*this->data;

return r;

}

INTEGER INTEGER::operator /(INTEGER x)

{

INTEGER r;

r.data = this->data/x.data;

return r;

}

int main()

{

INTEGER x;

x.getdata();

x.display();

INTEGER y;

y.getdata();

y.display();

INTEGER a=x+y;

a.display();

INTEGER b=x-y;

b.display();

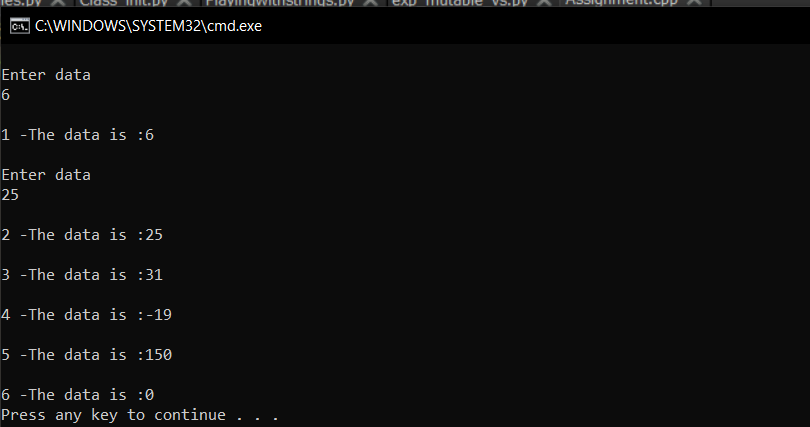
INTEGER c=x\*y;

c.display();

INTEGER d=x/y;

d.display();

}



Q1-B: A **b)Create an abstract base class shape with two members base and height, a member function for intialisation and a pure virtual function to compute area(). Derive two specific classes Triangle and Rectangle which override the function area().Use these classes in a main function and display the area of a triangle and a rectangle**

#include<iostream>

#include<iomanip>

using namespace std;

class shape{

protected:

float height,base;

public:

void getdata()

{

cout<<"Enter height"<<endl;

cin>>height;

cout<<"Enter base"<<endl;

cin>>base;

}

virtual void area()=0;

};

class Triangle:public shape{

public:

Triangle(){

}

void area()

{

cout<<"The area of triangle is: "<<fixed<<setprecision(2)<<(0.5\*height\*base)<<"cm2"<<endl;

}

};

class Rectangle:public shape{

public:

Rectangle(){

}

void area()

{

cout<<"The area of Rectangle is: "<<fixed<<setprecision(2)<<(height\*base)<<"cm2"<<endl;

}

};

int main()

{

shape \*s;

Triangle t;

Rectangle r;

s=&t;

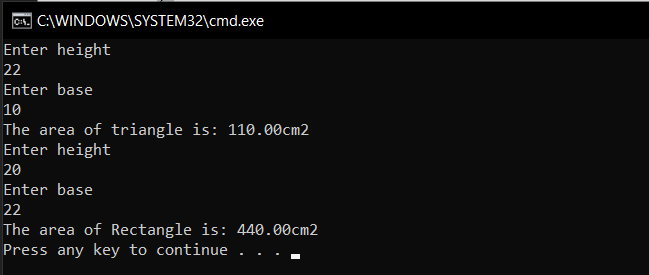
s->getdata();

s->area();

s=&r;

s->getdata();

s->area();

}

**Q2.a) Assume that a bank maintains two kinds of accounts for customers. One called as saving account and the other as current account. The savings account provides compound interest and withdrawal facility but no cheque book facility. The current account provides cheque book but no interest. Current account holders should also minimum balance and if the balance and if the balance falls below this level a service charge is imposed.**

**Create a class account that stores customer name, account number and type of account. From this derive the classes current\_account and savings\_account to make them more specific to their requirements.Include necessary member functions in order to achieve the following tasks:**

1. **Accept deposit from a customer and update the balance**
2. **Display the balance**
3. **Compute and desposit interest.**
4. **Permit withdrawal and update the balance**
5. **Check for minimum balance, impose penalty, if necessary, and update the balance.**

**Do not use any consructors. Use member functions to initialize the class members.**

#include<iostream>

using namespace std;

class account{

protected:

char name[100];

long long int accno;

float balance;

char type[100];

public:

void getdata()

{

fflush(stdin);

cout<<"Enter name"<<endl;

gets(name);

cout<<"Enter acc no"<<endl;

cin>>accno;

cout<<"Type?"<<endl;

cin>>type;

cout<<"Balance"<<endl;

cin>>balance;

}

void display()

{

cout<<"---------------------------------------------------------------------"<<endl;

cout<<"Name:"<<name<<endl;

cout<<"Type:"<<type<<endl;

cout<<"Acc no:"<<accno<<endl;

cout<<"Balance:"<<balance<<endl;

cout<<"---------------------------------------------------------------------"<<endl;

}

void deposit()

{

float s;

cout<<"Amount to be deposited"<<endl;

cin>>s;

this->balance+=s;

}

};

class saveacc:public account{

float interest;

public:

void withdraw(){

float s;

cout<<"Amount to be withrawn"<<endl;

cin>>s;

if(this->balance>=s)

{

cout<<"Succesfful"<<endl;

this->balance-=s;

}

else{

cout<<"Unsuccesfull"<<endl;

}

}

void computerInterest(){

int m;

cout<<"Enter the no of months"<<endl;

cin>>m;

interest = ((2/(balance))\*100)\*m;

}

void addInterest(){

this->balance+=interest;

cout<<endl<<"UPDATED Successfully"<<endl;

}

};

class curracc:public account{

public:

void check(){

if(balance<500){

cout<<"Balance"<<balance<<endl;

cout<<"LESS than minimum balance, PENALTY!"<<endl;

this->balance-=100;

}

else{

cout<<"No penalty"<<endl;

}

}

void cheque()

{

float amount;

cout<<"Enter the amount"<<endl;

cin>>amount;

if(this->balance>=amount)

{

cout<<"Succesfful"<<endl;

this->balance-=amount;

}

else{

cout<<"Unsuccesfull"<<endl;

}

}

};

int main()

{

curracc \*CA[100];

saveacc \*SA[100];

int i=0,j=0;

while(1)

{

int c,ch;

cout<<"1-Current Account"<<endl;

cout<<"2-Saved Account"<<endl;

cin>>c;

if(c==1)

{

CA[i]=new curracc();

CA[i]->getdata();

CA[i]->display();

cout<<"1-Deposit"<<endl;

cout<<"2-Check"<<endl;

cout<<"3-Display"<<endl;

cout<<"4-Cheque"<<endl;

cin>>ch;

switch(ch){

case 1:

CA[i]->deposit();

CA[i]->display();

break;

case 2:

CA[i]->check();

break;

case 3:

CA[i]->display();

break;

case 4:

CA[i]->cheque();

CA[i]->display();

break;

}

i++;

}

else

{

SA[j]=new saveacc();

SA[j]->getdata();

SA[j]->display();

cout<<"1-Deposit"<<endl;

cout<<"2-Interest"<<endl;

cout<<"3-Add interest"<<endl;

cout<<"4-Withdraw"<<endl;

cout<<"5-Display"<<endl;

cin>>ch;

switch(ch){

case 1:

SA[j]->deposit();

CA[i]->display();

break;

case 2:

SA[j]->computerInterest();

break;

case 3:

SA[j]->addInterest();

CA[i]->display();

break;

case 4:

SA[j]->withdraw();

CA[i]->display();

break;

case 5:

SA[j]->display();

break;

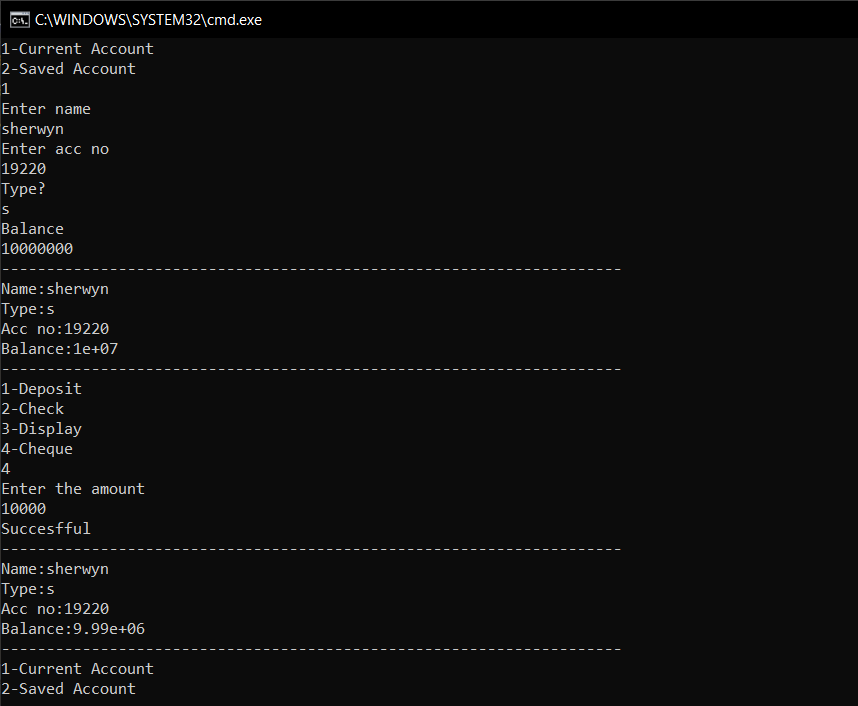
}

j++;

}

}

}



Q2-B **b) Create a class hierarchy for the worker and supervisor using C++. The Worker class should include details like workerID, Name, address of the worker, department ID and supervisor ID. The classes should have necessary functions to input the required information and a polymorphic function to print the details of the worker’s this print function should output the worker’s ID, Name and supervisor ID. Whereas for the supervisor this function should output the Name and Department ID of the supervisor. Please note that a supervisor is also a worker. Make and state suitable assumptions, if any.**

#include<iostream>

#include<iomanip>

using namespace std;

class Worker{

protected:

char name[100],address[100];

long long int workerid,departmentid,supervisorid;

public:

void getdata(){

fflush(stdin);

cout<<"Enter the name"<<endl;

gets(name);

cout<<"Enter the address"<<endl;

gets(address);

cout<<"Enter the id of supervisor"<<endl;

cin>>this->supervisorid;

cout<<"Enter the id of worker"<<endl;

cin>>this->workerid;

cout<<"Enter the id of department"<<endl;

cin>>this->departmentid;

}

virtual void display()

{

cout<<"-----------------------------------------------------------"<<endl;

cout<<"Worker Name: "<<right<<setw(24)<<this->name<<endl;

cout<<"WorkerID: "<<right<<setw(24)<<this->workerid<<endl;

cout<<"SupervisorID: "<<right<<setw(24)<<this->supervisorid<<endl;

cout<<"-----------------------------------------------------------"<<endl;;

}

};

class Supervisor:public Worker{

public:

void display(){

cout<<"-----------------------------------------------------------"<<endl;

cout<<"DepartmentID: "<<right<<setw(20)<<this->departmentid<<endl;

cout<<"Worker Name: "<<right<<setw(20)<<this->name<<endl;

cout<<"-----------------------------------------------------------"<<endl;

}

};

int main()

{

Worker \*w,w2;

Supervisor s;

w=&w2;

w->getdata();

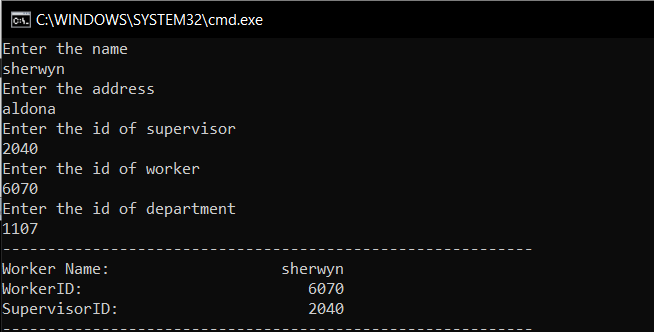
w->display();

w=&s;

w->getdata();

w->display();

}



**Module II**

**Write a program that reads a file and create another file that is identical except the every sequence of consecutive blank spaces is replaces by a single space.**

#include<iostream>

#include<fstream>

using namespace std;

int main()

{

char p,c;

fstream f1;

f1.open("f1",ios::out);

f1<<"This Great Wall Of China"<<endl;

f1<<"This Great Wall Of China"<<endl;

f1.close();

f1.open("f1",ios::in);

if(f1.good())

{

fstream fout;

fout.open("f2",ios::out);

while(f1.get(c))

{

if(p==' '&&c==' ')

continue;

else fout<<c,p=c;

}

fout.close();

fout.open("f2",ios::in);

while(fout.get(c))

{

cout<<c;

}

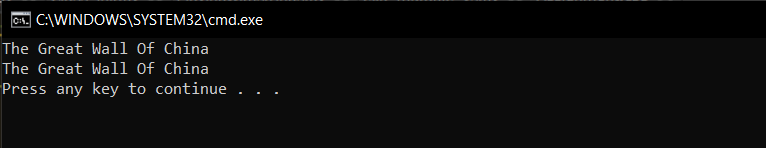
}

else{

cout<<"invalid";

}

}



**Q3 b Write a program to read file and display no of lines no of columns and no of characters in column**

#include<iostream>

#include<string.h>

using namespace std;

int main()

{

char c,s[100];

int i,a,d,ch;

i=0;

cin.get(c);

while(c!='^'){

s[i] = c;

cin.get(c);

i++;

}

int n = strlen(s);

i=a=d=ch=0;

while(i<n)

{

if(s[i]=='\n')

d++;

if(s[i]==' '||s[i]=='\t'||s[i]=='\n')

a++;

if((s[i]>='a'&&s[i]<='z')||(s[i]>='A'&&s[i]<='Z')|| (s[i]>='0'&&s[i]<='9'))

ch++;

i++;

}

cout.setf(ios::left,ios::adjustfield);

cout.width(17);

cout<<"Number of lines";

cout.width(17);

cout<<"Number of words";

cout.width(21);

cout<<"Number of characters"<<endl;

cout.setf(ios::right,ios::adjustfield);

cout.width(13);

cout<<d;

cout.width(17);

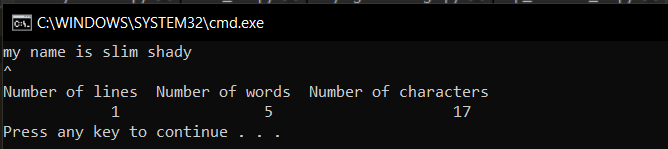
cout<<a;

cout.width(22);

cout<<ch;

cout<<endl;

}



**Q4 a How is exception handling performed in c++.**

Write a program that throws an arithmetic error when number entered is greater than 9999

Exceptions are runtime anomalies or abnormal conditions occur while executing a program.

An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: **try, catch,** and **throw**.

* **throw** − A program throws an exception when a problem shows up. This is done using a **throw** keyword.
* **catch** − A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The **catch** keyword indicates the catching of an exception.
* **try** − A **try** block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

Throwing Exceptions

Exceptions can be thrown anywhere within a code block using **throw** statement. The operand of the throw statement determines a type for the exception and can be any expression and the type of the result of the expression determines the type of exception thrown.

Catching Exceptions

The **catch** block following the **try** block catches any exception. You can specify what type of exception you want to catch and this is determined by the exception declaration that appears in parentheses following the keyword catch.

Program:

#include<iostream>

using namespace std;

int main()

{

int num;

try{

cout<<"Enter Number"<<endl;

cin>>num;

if(num>9999)

throw num;

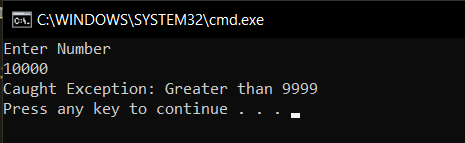
}

catch(int c)

{

cout<<"Caught Exception !!!! Greater than 9999"<<endl;

}



**Q4b Write a c++ template function called exchange() that accepts two arguments of generic type and swaps content**;

#include<iostream>

using namespace std;

template<class T>

void exchange(T &a,T &b)

{

a = b-a;

b = b-a;

a = a+b;

}

int main()

{

int a,b;

cout<<"Enter two int numbers"<<endl;

cin>>a>>b;

cout<<"a: "<<a<<" b: "<<b<<endl;

exchange(a,b);

cout<<"After swapping"<<endl;

cout<<"a: "<<a<<" b: "<<b<<endl;

float x,y;

cout<<"Enter two floats"<<endl;

cin>>x>>y;

cout<<"a: "<<x<<" b: "<<y<<endl;

exchange(x,y);

cout<<"After swapping"<<endl;

cout<<"a: "<<x<<" b: "<<y<<endl;

char c1,c2;

cout<<"Enter two characters"<<endl;

cin>>c1>>c2;

cout<<"a: "<<c1<<" b: "<<c2<<endl;

exchange(c1,c2);

cout<<"After swapping"<<endl;

cout<<"a: "<<c1<<" b: "<<c2<<endl;

double d1,d2;

cout<<"Enter two doubles"<<endl;

cin>>d1>>d2;

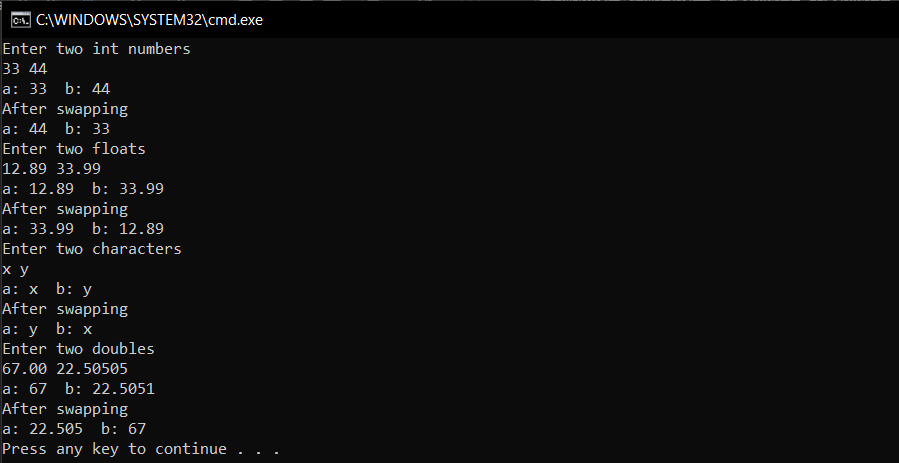
cout<<"a: "<<d1<<" b: "<<d2<<endl;

exchange(d1,d2);

cout<<"After swapping"<<endl;

cout<<"a: "<<d1<<" b: "<<d2<<endl;

}



**Explain the concept of stack unwinding in case of exception handling with appropriate example**

# Stack Unwinding in C++

The process of removing function entries from function call stack at run time is called [Stack Unwinding](http://en.wikipedia.org/wiki/Call_stack#Unwinding). Stack Unwinding is generally related to Exception Handling. In C++, when an exception occurs, the function call stack is linearly searched for the exception handler, and all the entries before the function with exception handler are removed from the function call stack. So exception handling involves Stack Unwinding if exception is not handled in same function (where it is thrown).

For example, output of the following program is:

#include <iostream>

using namespace std;

// A sample function f1() that throws an int exception

void f1() throw (int) {

cout<<"\n f1() Start ";

throw 100;

cout<<"\n f1() End ";

}

// Another sample function f2() that calls f1()

void f2() throw (int) {

cout<<"\n f2() Start ";

f1();

cout<<"\n f2() End ";

}

// Another sample function f3() that calls f2() and handles exception thrown by f1()

void f3() {

cout<<"\n f3() Start ";

try {

f2();

}

catch(int i) {

cout<<"\n Caught Exception: "<<i;

}

cout<<"\n f3() End";

}

// A driver function to demonstrate Stack Unwinding process

int main() {

f3();

getchar();

return 0;

}

In the above program, when f1() throws exception, its entry is removed from the function call stack (because it f1() doesn’t contain exception handler for the thrown exception), then next entry in call stack is looked for exception handler. The next entry is f2(). Since f2() also doesn’t have handler, its entry is also removed from function call stack. The next entry in function call stack is f3(). Since f3() contains exception handler, the catch block inside f3() is executed, and finally the code after catch block is executed. Note that the following lines inside f1() and f2() are not executed at all.

**Module III**

**Q5.a)A palindrome is a word or group of words that read the same forward and backward. For example “madam” or “wow”. Write a program that takes a string object from keyboard and, using the member functions defined standard C++ library string class, to determine whether the string was a palindrome or not.**

#include<bits/stdc++.h>

using namespace std;

int main()

{

string str;

cout<<"Enter the string"<<endl;

cin>>str;

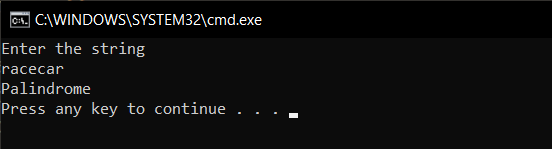
string c = string(str.rbegin(),str.rend());

if(str==c)

cout<<"Pallindrom"<<endl;

else cout<<"Not a pallindrom"<<endl;

}



Q5-B**b)Define using STL, a vector v with a maximum size of 10.**

1. **Sets the first element to of v to 100**
2. **Sets the last element of to 200**
3. **Sets the other elements to 10**
4. **Displays the contents of v**

#include<iostream>

#include<vector>

using namespace std;

int main()

{

vector<int> v(10);

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

(\*it)=100;

it=v.end()-1;

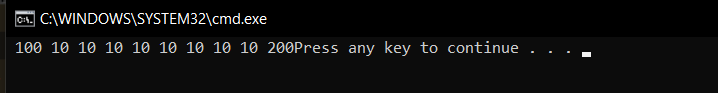
(\*it)=200;

fill(v.begin()+1, v.end()-1, 10);

for(it=v.begin();it!=v.end();it++)

cout<<" "<<(\*it);

}



Q6-A **What is an algorithm? How STL algorithms are different from conventional algorithms?**

**Algorithms**

* Algorithms act on containers. They provide the means by which you will perform initialization, sorting, searching, and transforming of the contents of containers.
* Algorithms in STL is a collection of functions specially designed to be used on ranges of elements. A range is any sequence of objects that can be accessed through iterators or pointers, such as an array or an instance of some of the STL containers. Examples of algorithms in STL: sort (Sort elements in range), binary\_search (Test if a value exists in sorted sequence), min\_element (Return smallest element in range), etc. Note that all these algorithms can be applied to any data type accepted as a template.
* In the C++ programming language, the C++ Standard Library is a collection of classes and functions, which are written in the core language and part of the C++ ISO Standard itself. The C++ Standard Library provides several generic containers, functions to utilize and manipulate these containers, function objects, generic strings and streams (including interactive and file I/O), support for some language features, and functions for everyday tasks such as finding the square root of a number.

They're optimized for use with STL containers, and they're faster, clearer, and more idiomatic than anything you can write yourself. The only situation you should consider rolling your own is if you can articulate a very specific, mission-critical need that the STL algorithms don't satisfy.

Q 6B **Distinguish between the following :**

List vs vectors

| **Vector** | **List** |
| --- | --- |
| It has contiguous memory. | While it has non-contiguous memory. |
| It is synchronized. | While it is not synchronized. |
| Vector may have a default size. | List does not have default size. |
| In vector, each element only requires the space for itself only. | In list, each element requires extra space for the node which holds the element, including pointers to the next and previous elements in the list. |
| Insertion at the end requires constant time but insertion elsewhere is costly. | Insertion is cheap no matter where in the list it occurs. |
| Vector is thread safe. | List is not thread safe. |
| Deletion at the end of the vector needs constant time but for the rest it is O(n). | Deletion is cheap no matter where in the list it occurs. |
| Random access of elements is possible. | Random access of elements is not possible. |
| Iterators become invalid if elements are added to or removed from the vector. | Iterators are valid if elements are added to or removed from the list. |

Sets vs maps

Sets and map in STL are similar in the sense that they both use of red black tree (A self balancing bst). Note that the time complexities of search, insert and delete are O(Log n).

**Differences:**The difference is set is used to store only keys while map is used to store key value pairs.

| **Map** | **Set** |
| --- | --- |
| Map is used to store key value pairs. | Set is used to store only keys |

Maps vs Multimaps

| **Map** | **Multimap** |
| --- | --- |
| The map will only store each value once for a specific key. To do that, it will have to be able to compare the values, not just the keys | The multimap stores pairs of (key,value) where both key and value can appear several times |

Queue vs Dequeue

| **Queues** | **Dequeue** |
| --- | --- |
| Queues only allow insertion in one end and retrieval from the other. | A deque is a double-ended queue, which allows easy insertion/removal from either end |

Vectors vs Arrays

| **Vector** | **Arrays** |
| --- | --- |
| Vector is a sequential container to store elements and not index based. | Array stores a fixed-size sequential collection of elements of the same type and it is index based. |
| Vector is dynamic in nature so, size increases with insertion of elements. | As array is fixed size, once initialized can’t be resized. |
| Vector occupies more memory. | Array is memory efficient data structure. |
| Vector takes more time in accessing elements. | Array access elements in constant time irrespective of their location as elements are arranged in a contiguous memory allocation. |

Q 6 c **Compare the performance characteristics of the three sequence containers.**

In standard template library they refer to the group of **container class template**, we use to them store data. One common property as the name suggests is that elements **can be accessed sequentially.**  
Each of the following containers use different algorithm for data storage thus for different operations they have different speed. And all the elements in the containers should be of **same**

 The std::vector: This is a dynamically-resized array of elements. All the elements are contiguous in memory. If an element is inserted or removed it at a position other than the end, the following elements will be moved to fill the gap or to open a gap. Elements can be accessed at random position in constant time. The array is resized so that it can several more elements, not resized at each insert operation. This means that insertion at the end of the container is done in amortized constant time.

 The std::deque: The deque is a container that offer constant time insertion both at the front and at the back of the collection. In current c++ libraries, it is implementation as a collection of dynamically allocated fixed-size array. Not all elements are contiguous, but depending on the size of the data type, this still has good data locality. Access to a random element is also done in constant time, but with more overhead than the vector. For insertions and removal at random positions, the elements are shifted either to the front or to the back meaning that it is generally faster than the vector, by twice in average.

 The std::list: This is a doubly-linked list. It supports constant time insertions at any position of the collection. However, it does not support constant time random access. The elements are obviously not contiguous, since they are all allocated in nodes. For small elements, this collection has a very big memory overhead.