FINALTERM EXAMINATION
Fall 2009
CS304- Object Oriented Programming (Session - 1)

"VISIT VURANK FOR MORE"

Time: 120 min
Marks: 75

Question No: 1 (Marks: 1) - Please choose one

Which one of the following terms must relate to **polymorphism?**

- ► Static allocation
- ► Static typing
- ► **Dynamic binding**
- ► Dynamic allocation

Question No: 2 (Marks: 1) - Please choose one

Multiple inheritance can be of type

- ► Public
- ► Private
- ► Protected
- ► **All of the given**

Question No: 3 (Marks: 1) - Please choose one

When a subclass specifies an alternative definition for an attribute or method of its superclass, it is _____ the definition in the superclass.

- ► overload
- ► **overriding**
- ► copy riding
- ► none of given

Question No: 4 (Marks: 1) - Please choose one

Like template functions, a class template may not handle all the types successfully.

- ► **True**
- ► False

Question No: 5 (Marks: 1) - Please choose one

It is sometimes useful to specify a class from which no objects will ever be created.

- ► True
- ► **False**

Question No: 6 (Marks: 1) - Please choose one

Assume a class Derv that is privately derived from class Base. An object of class Derv located in main() can access

- ► public members of Derv.
- ► protected members of Derv.

- ► private members of Derv.
- ► **protected members of Base.**

Question No: 7 (Marks: 1) - Please choose one

A pointer to a base class can point to objects of a derived class.

- ► **True**
- ► False

Question No: 8 (Marks: 1) - Please choose one

A copy constructor is invoked when

- ► a function do not returns by value.
- ► an argument is passed by value.
- ► **a function returns by reference.**
- ► an argument is passed by reference.

Question No: 9 (Marks: 1) - Please choose one

A function call is resolved at run-time in_____

- ► non-virtual member function.
- ► **virtual member function.**
- ► Both non-virtual member and virtual member function.
- ► None of given

Question No: 10 (Marks: 1) - Please choose one

When the base class and the derived class have a member function with the same name, you must be more specific which function you want to call (using _____).

- ► **scope resolution operator**
- ► dot operator
- ► null operator
- ► Operator overloading

Question No: 11 (Marks: 1) - Please choose one

Each try block can have _____ no. of catch blocks.

- ► 1
- ► 2
- ► 3
- ► **As many as necessary.**

Question No: 12 (Marks: 1) - Please choose one

Two important STL associative containers are _____ and _____.

- ► set,map
- ► **sequence,mapping**
- ► setmet,multipule
- ► sit,mat

Question No: 13 (Marks: 1) - Please choose one

The mechanism of selecting function at run time according to the nature of calling object is called,

- ► late binding
- ► static binding
- ► virtual binding
- ► **None of the given options**

Question No: 14 (Marks: 1) - Please choose one

An abstract class is useful when

- ► We do not derive any class from it.
- ► There are multiple paths from one derived class to another.
- ► **We do not want to instantiate its object.**
- ► You want to defer the declaration of the class.

Question No: 15 (Marks: 1) - Please choose one

Which of the following is incorrect line regarding function template?

- ► template<class T>
- ► template <typename U>
- ► **Class<template T>**
- ► template < class T, class U>

Question No: 16 (Marks: 1) - Please choose one

Which of the following is/are advantage[s] of generic programming?

- ► Reusability
- ► Writability
- ► Maintainability
- ► **All of given**

Question No: 17 (Marks: 1) - Please choose one

By default the vector data items are initialized to ____

- ► **0**
- ► 0.0
- ► 1
- ► null

Question No: 18 (Marks: 1) - Please choose one

Which one of the following functions returns the total number of elements in a vector.

- ► length();
- ► **size();**
- ► ele();
- ► veclen();

Question No: 19 (Marks: 1) - Please choose one

Suppose you create an uninitialized vector as follows:

vector<int> evec;
After adding the statment,
evec.push_back(21);
what will happen?

► The following statement will add an element to the start (the back) of evec and will initialize it with the value 21.

► The following statement will add an element to the center of evec and will reinitialize it with the value 21.

► The following statement will delete an element to the end (the back) of evec and will reinitialize it with the value 21.

► The following statement will add an element to the end (the back) of evec and initialize it with the value 21.

Question No: 20 (Marks: 1) - Please choose one

An STL container can not be used to,

► hold objects of class employee.

► store elements in a way that makes them quickly accessible.

► compile c++ programs.

► organize the way objects are stored in memory

Question No: 21 (Marks: 1) - Please choose one

Algorithms can only be implemented using STL containers.

► True

► False

Question No: 22 (Marks: 1) - Please choose one

The main function of scope resolution operator (**::**) is,

► To define an object

► To define a data member

► To link the definition of an identifier to its declaration

► To make a class private

Question No: 23 (Marks: 1) - Please choose one

When is a constructor called?

► Each time the constructor identifier is used in a program statement

► During the instantiation of a new object

► During the construction of a new class

► At the beginning of any program execution

Question No: 24 (Marks: 1) - Please choose one

Consider the code below,

class Fred {

```
public:
Fred();
...
};
int main()
{
Fred a[10];
Fred* p = new Fred[10];
...
}
```

Select the best option,

► Fred a[10]; calls the default constructor 09 times
Fred* p = new Fred[10]; calls the default constructor 10 times

► **Produce an error**

► Fred a[10]; calls the default constructor 11 times
Fred* p = new Fred[10]; calls the default constructor 11 times

► Fred a[10]; calls the default constructor 10 times
Fred* p = new Fred[10]; calls the default constructor 10 times

Question No: 25 (Marks: 1) - Please choose one

Associativity can be changed in operator overloading.

► True
► **False**

Question No: 26 (Marks: 1) - Please choose one

A normal C++ operator that acts in special ways on newly defined data types is said to be

► glorified.
► encapsulated.
► **classified.**
► overloaded.

Question No: 27 (Marks: 1) - Please choose one

Which operator can not be overloaded?

► The relation operator (>=)
► Assignment operator (=)
► Script operator ([])
► **Conditional operator (? :)**

Question No: 28 (Marks: 1) - Please choose one

Suppose obj1 and obj2 are two objects of a user defined class A. An + operator is overloaded to add obj1 and obj2 using the function call obj1+obj2.
Identify the correct function prototype against the given call?

- A operator + (A &obj);
- int + operator();
- **int operator (plus) ();**
- A operator(A &obj3);

Question No: 29 (Marks: 1) - Please choose one

Default constructor is such constructor which either has no ---------or if it has some parameters these have -------- values

- Parameter, temporary
- Null, Parameter
- **Parameter, default**
- non of the given

Question No: 30 (Marks: 1) - Please choose one

Public methods of base class can --------- be accessed in its derived class

- directly
- **inderectly**
- simultaniously
- non of the given

Question No: 31 (Marks: 1)

Is **Deque** a Birectional Container?

Yes, deque behaves like queue (line) such that we can add elements on both sides of it.

Question No: 32 (Marks: 1)

What is meant by Generic Programming?

Generic programming refers to programs containing generic abstractions general code that is same in logic for all data types like printArray function), then we instantiate that generic program abstraction (function, class) for a particular data type, such abstractions can work with many different types of data.

Question No: 33 (Marks: 2)

Sort the following data in the order in which compiler searches a function?
Complete Specialization, Generic Template, Partial Specialization, Ordinary Function.

Specializations of this function template, instantiations with specific types, can be called just like an ordinary function:

```
cout << max(3, 7);  // outputs 7
```

The compiler examines the arguments used to call max and determines that this is a call to max(int, int). It then instantiates a version of the function where the parameterizing type T is int, making the equivalent of the following function:

```
int max(int x, int y)
{
   return x < y ? y : x;
}
```

the C++ Standard Template Library contains the function template max(x, y) which creates functions that return either *x* or *y*, whichever is larger. max() could be defined like this:

```
template <typename T>
T max(T x, T y)
{
   return x < y ? y : x;
}
```

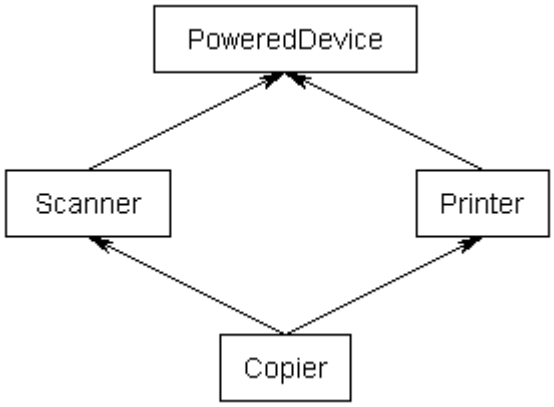
Question No: 34 (Marks: 2)

State any conflict that may rise due to multiple inheritance?

The conflict may arise is the diamond problem, which our author likes to call the "diamond of doom". This occurs when a class multiply inherits from two classes which each inherit from a single base class. This leads to a diamond shaped inheritance pattern.

For example, consider the following set of classes:

```
classPoweredDevice
{
};
classScanner: publicPoweredDevice
{
};
classPrinter: publicPoweredDevice
{
};
classCopier: publicScanner, publicPrinter
{
};
```



Scanners and printers are both powered devices, so they derived from PoweredDevice. However, a copy machine incorporates the functionality of both Scanners and Printers.

Ambiguity also cause problem.

Question No: 35 (Marks: 3)

Describe three properties necessary for a container to implement Generic Algorithms.

If you declare a container as holding pointers, you are responsible for managing the memory for the objects pointed to. The container classes will not automatically free memory for these objects when an item is erased from the container.
Container classes are expected to implement methods to do the following:

- create a new empty container (constructor),
- report the number of objects it stores (size),
- delete all the objects in the container (clear),
- insert new objects into the container,
- remove objects from it,
- provide access to the stored objects.

Question No: 36 (Marks: 3)

Write three important features of virtual functions.

With virtual functions, derived classes can provide new implementations of functions from their base classes. When someone calls a virtual function of an object of the derived class, this new implementation is called, even if the caller uses a pointer to the base class, and doesn't even know about the particular derived class.
The virtual function is an option, and the language defaults to non virtual, which

is the fastest configuration.
The derived class can completely "override" the implementation or "augment" it (by explicitly calling the base class implementation in addition to the new things it does).

Question No: 37 (Marks: 3)

Consider the code below,

```
#include <iostream>
#include <stdlib.h>
using namespace std;
class Shape{
        public:
        void Draw(){cout<<"shape"<<endl;}
    };
class Line : public Shape{
        public:
        void Draw(){cout<<"Line"<<endl;}
        };
class Circle : public Shape{
        public:
        void Draw(){cout<<"Circle"<<endl;}
        };
int main(int argc, char *argv[])
{
  Shape * ptr1 = new Shape();
  Shape * ptr2 = new Line();
  Shape * ptr3 = new Circle();

  ptr1->Draw();
  ptr2->Draw();
  ptr3->Draw();
  system("PAUSE");
  return 0;
}
```

This code shows output,

```
Shape
Shape
Shape
```

Give the reason for this output

Suppose we want to show the output,

Shape
Line
Circle

How we can change the code to do that?

```
class shape { public:
   void draw();
};
class circle : public shape { };
int main(int argc, char **argv){
   circle my_circle;
   my_circle.draw();
}
```

While this has all the usual advantages, e.g., code reuse, the real power of polymorphism comes into play when draw is declared to be virtual or pure virtual, as follows:

```
class shape{ public:
   virtual void draw()=0;
};
class circle : public shape { public:
   void draw();
}
```

Here, circle has declared its own draw function, which can define behavior appropriate for a circle. Similarly, we could define other classes derived from shape, which provide their own versions of draw. Now, because all the classes implement the shape interface, we can create collections of objects that can provide different behavior invoked in a consistent manner (calling the draw member function). An example of this is shown here.

```
shape *shape_list[3];   // the array that will
                        // pointer to our shape objects
shape[0] = new shape;  // three types of shapes
shape[1] = new line;  // we have defined
shape[2] = new circle;
for(int i = 0; i < 3; i++){
   shape_list[i].draw();
}
```

When we invoke the draw function for each object on the list, we do not need to know anything about each object; C++ handles the details of invoking the correct version of draw. This is a very powerful technique, allowing us to provide extensibility in our designs. Now we can add new classes derived from shape to provide whatever behavior we desire. The key here is that we have separated the interface (the prototype for shape) from the implementation.

Question No: 38 (Marks: 5)

There are some errors in the code given below, you have to

1. Indicate the line no. with error/s
2. Give the reason for error/s
3. Correct the error/s.

```
1.  #include <iostream>          this will be #include <iostream.h>
2.  #include <stdlib.h>

3.  using namespace std;
4.  template <typename T>
5.  class MyClass{
6.  public:
7.  MyClass(){
8.  cout<<"This is class1"<<endl;
9.  }
10. };
11. template <typename T>
12. class MyClass<int*>{
13. public:
14. MyClass(){
15. cout<<"This is class2"<<endl;
16. }
17. };
18. int main(int argc, char *argv[])
19. {
20. MyClass<int> c1;
21. MyClass<int*> c2;
22. system("PAUSE");
23. return 0;
24. }
```

Question No: 39 (Marks: 5)

Given are two classes A and B. class B is inherited from class A. Write a code snippet(for main function) that polymorphically call the method of class B. Also what changes do you suggest in the given code segment that are required to call the class B method polymorphically.

```
class A
{
public:
void method() { cout<<"A's method \n"; }

};
```

```
class B : public A
{

public:
void method() { cout<<"B's method\n"; }

};
```

Ans:

```
public class Test
{
public class A {}

public class B extends A {}

private void test(A a)
{
System.out.println("test(A)");
}

private void test(B b)
{
System.out.println("test(B)");
}

public static void main(String[] args)
{
Test t = new Test();
A a = t.new A();
A b = t.new B();

t.test(a);
t.test(b);
}
}
```

Question No: 40 (Marks: 10)

Create built-in STL (Standard Template Library) **vector class object** for **strings** and add in it some words by taking input from user, then apply the sort() algorithm to array of words stored in this vector class object.

Hint:Use push_back() to add the words in vector class object, and the [] operator and size() to display these sorted words.

The STL is the containers, iterators and algorithms component of the proposed C++ Standard Library [ANSI95]. It represents a novel application of principles which have their roots in styles of programming other than Object-orientation.

```
void listWords(istream& in, ostream& out)
{
	string s;

	while (!in.eof() && in >> s) {
		add s to some container
	}

	sort the strings in the container
	remove the duplicates

	for (each string t in container) {
		out << t;
	}
}
```

For now, assume that a word is defined as a whitespace-separated string as delivered by the stream extraction operator. Later on we will consider ways of refining this definition. Given the way this problem is expressed, we can implement this program directly, if naïvely. The STL container class vector will suffice to hold the words: applying the algorithms sort and unique provides the required result.

```
void listWords(istream& in, ostream& out)
{
	string s;
	vector<string> v;

	while (!in.eof() && in >> s)
		v.push_back(s);		// (1)

	sort(v.begin(), v.end());

	vector<string>::iterator e
		= unique(v.begin(), v.end());	// (2)

	for (vector<string>::iterator b = v.begin();
		b != e;
		b++) {
		out << *b << endl;
	}
}
```

At (1) the vector member function push_back() is used to add to the end of the vector.

This can also be done using the insert member, which takes as a parameter an iterator identifying the position in the vector at which to place the added element:

```
v.insert(v.end(), s);
```

This allows us to add at any position in the vector. Be aware, though, that adding anywhere other than the end implies the overhead of physically shifting all elements from the insertion point to the end to make room for the new value. For this reason, and given the choices made in this example, attempts to optimise this code by maintaining the vector in sorted order are unwise. Replace vector with list and this becomes possible - although in both cases a search over the container will be necessary to determine the correct position of insertion.

The unique algorithm has the surprising property of not changing the length of the container to which it is applied (it can hardly do this, as it has access not to the underlying container, but only to the pair of iterators it is passed). Instead, it guarantees that duplicates are removed by moving unique entries towards the beginning of the container, returning an iterator indicating the new end of the container. This can be used directly (as here, at (2)), conversely it can be passed to the erase member with the old end iterator, to truncate the container.

Question No: 41 (Marks: 10)

Q. Write a detailed note on Exceptions in Destructors with the help of a coding example.

Exceptions in Destructors:

An object is presumably created to do something. Some of the changes made by an object should persist after an object dies (is destructed) and some changes should not. Take an object implementing a SQL query. If a database field is updated via the SQL object then that change should persist after the SQL objects dies. To do its work the SQL object probably created a database connection and allocated a bunch of memory. When the SQL object dies we want to close the database connection and deallocate the memory, otherwise if a lot of SQL objects are created we will run out of database connections and/or memory.

The logic might look like:

```
Sql::~Sql()
{
  delete connection;
  delete buffer;
}
```

Let's say an exception is thrown while deleting the database connection. Will the buffer be deleted? No. Exceptions are basically non-local gotos with stack cleanup. The code for deleting the buffer will never be executed creating a gaping resource leak.

Special care must be taken to catch exceptions which may occur during object destruction. Special care must also be taken to fully destruct an object when it throws an exception.

Example code for exception ……

```
#include<iostream.h>
#include<conio.c>
class Exception {
private:

 char message[30]  ;
public:

 Exception() {strcpy(message,"There is not enough stock");}
 char * get_message() { return message; }
};
class Item {
private:

 int stock ;
 int required_quantity;
public:

 Item(int stk, int qty)
    {
     stock =  stk;
     required_quantity =  qty;
 }
    int get_stock()
    {
       return stock;
    }

    int get_required_quantity()
    {
       return required_quantity;
    }

    void order()
    {
        if (get_stock()< get_required_quantity())

        throw Exception();
                  else
                  cout<<"The required quantity of item is available in the stock";
    }

     ~Item(){}
};
```

```
void main()
{

   Item obj(10, 20);

 try
   {
  obj.order();
 }
 catch(Exception & exp2)
   {
    getch();
  cout << "Exception: " << exp2.get_message() << endl;
 }
 getch();
```
