Nov/Dec 2016

Module - I

2. A

a) No;

b)
```
#include <iostream>
using namespace std;
class A
{ public:
  A()
  { cout << "A"; }
  };
class B : public A
{ public : B() {}
};
class C : public B
{ public : C()
  }};
class D {
  public : D()
  { cout << "D"; }};
class E : public C, public D
{ public : E()
  { cout << "D";
  }};
```

```
class F : B, virtual E
{ public : F()
  { cout << "F";
    }};
  int main()
  { F F;
    return 0;
    }
```

output

A B C D D A B F

Demenstrate with the help of c++ code how runtime polymorphism is performed by the c++ compilier using VT table.

```
class Employee
{ public:
  virtual void raise_salary()
  { /* common raise salary code */ }
  virtual void promote
  { /* common promote code */ }
  };
class Manager : public Employee
{ virtual void raise_salary ()
  { /* Mangger specify raise salary code. */ }
  virtual void promote ()
  { /* Manager specific promote */ }
  };
```

```
void globalRaise_Salary ( Employee *emp[], int n)
{ for (int i = 0; i<n; i++)
    emp[i] -> raise_salary (); // polymorphic call
                               // : calls raise_salary()
}
int main()
{ // return 0; Employee *e = new Manager;
  e ->
}
```

compiler maintain 2 things

vtable: A table of function pointer. It is maintained per class

vptr: A pointer to vtable. It is maintained per object

compiler adds additional code at 2 places to maintain vptr

1) code in every constructor: this code sets vptr of the object being created. This code set vptr to point to vtable of the class.

2) Code with polymorphic function: whenever polymorphic call is made, compiler inserts code to first look for vptr using base class pointer or reference. Once vptr is fetched, vtable of derived class can be accessed since vptr is of derived class. Using vtable address of derived class function ~~is accessed and called~~.

Module - 2

4.a. consider the fall code

```
template <class X, class Y> void sum (X a, X b)
{ cout << a+b << endl;
}
```

```
int main()
{
  int i1, i2;
  double d1, d2;
   i1=10;
   i2 = 22;
   d1 = 12340; d2 = 499090;
   sum (i1, d2);
    return 0; }
```

is this code sum accepts 2 parameters with different data types But at runtime 2 parameters with same data type (int) have been passed. will there be an error in this program?

yes: no matching function for call to 'sum (int &, double &)'

b). What is template specilization? explain with syntax.

```
template <class T>
void fun (T a)
{ cout << "In main template fun(): " << a << endl;
} // template specialization
template <>
void fun (int a)
{
   cout << "specialized template for int type " << a << endl;
}
 int main()
 { fun <char> ('a');
   fun <int> (10);
   fun <int> 3;
```

here we have general template func() for all data type except int. for int, there is a specialized version of func().

Module - 4

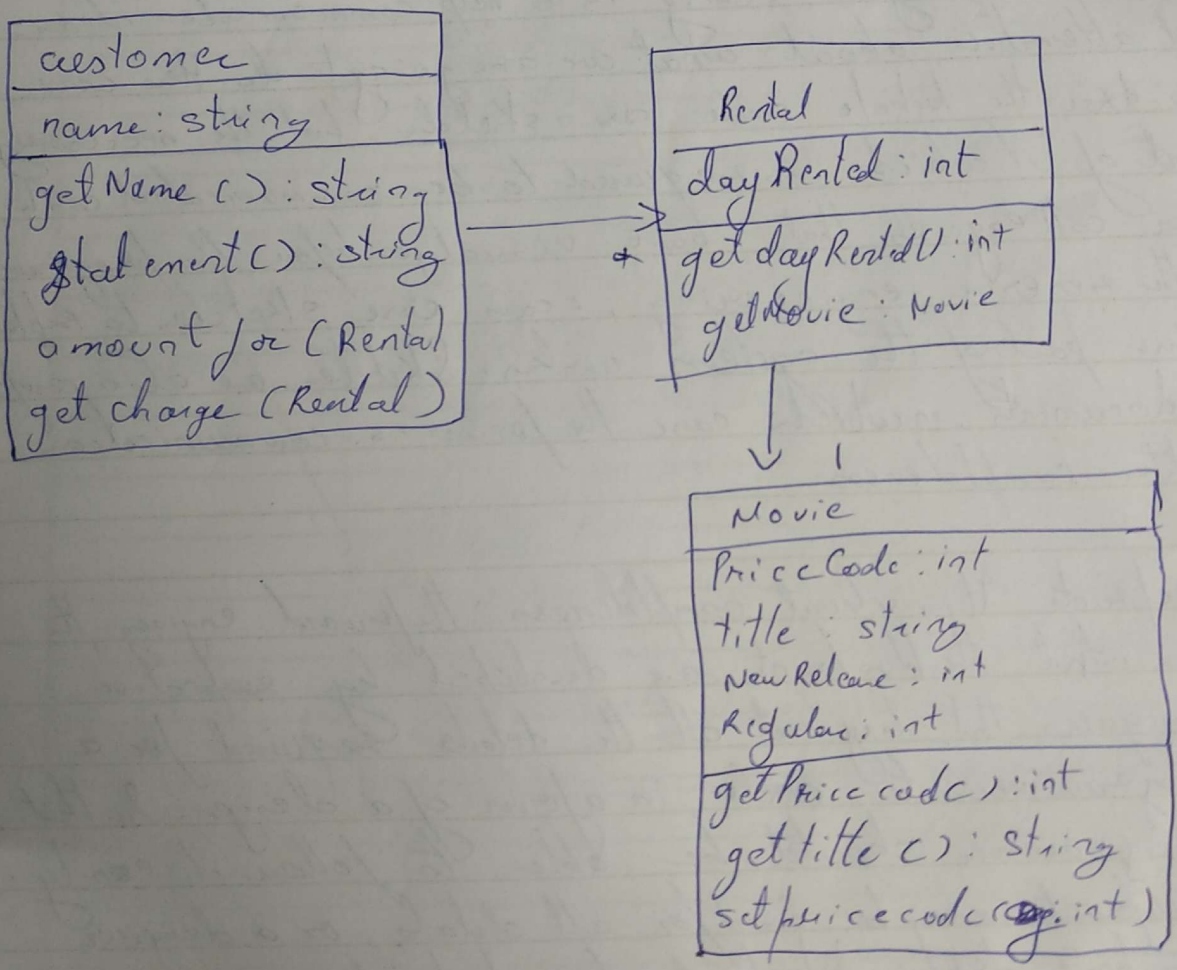
7.a Explain any 3 ways of using UML.

1) Sketch: aim of sketching is to help communicate ideas and alternative about what we are going to do. Here were do draw the whole code as a sketch but the necessary part of it which is required to be communicated with our colleagues that i have visualize before the programming. With reserve-engineering, i can use sketches to explain some part of the system works. Sketches are also used in documents, in which case the focus is communication rather completeness.

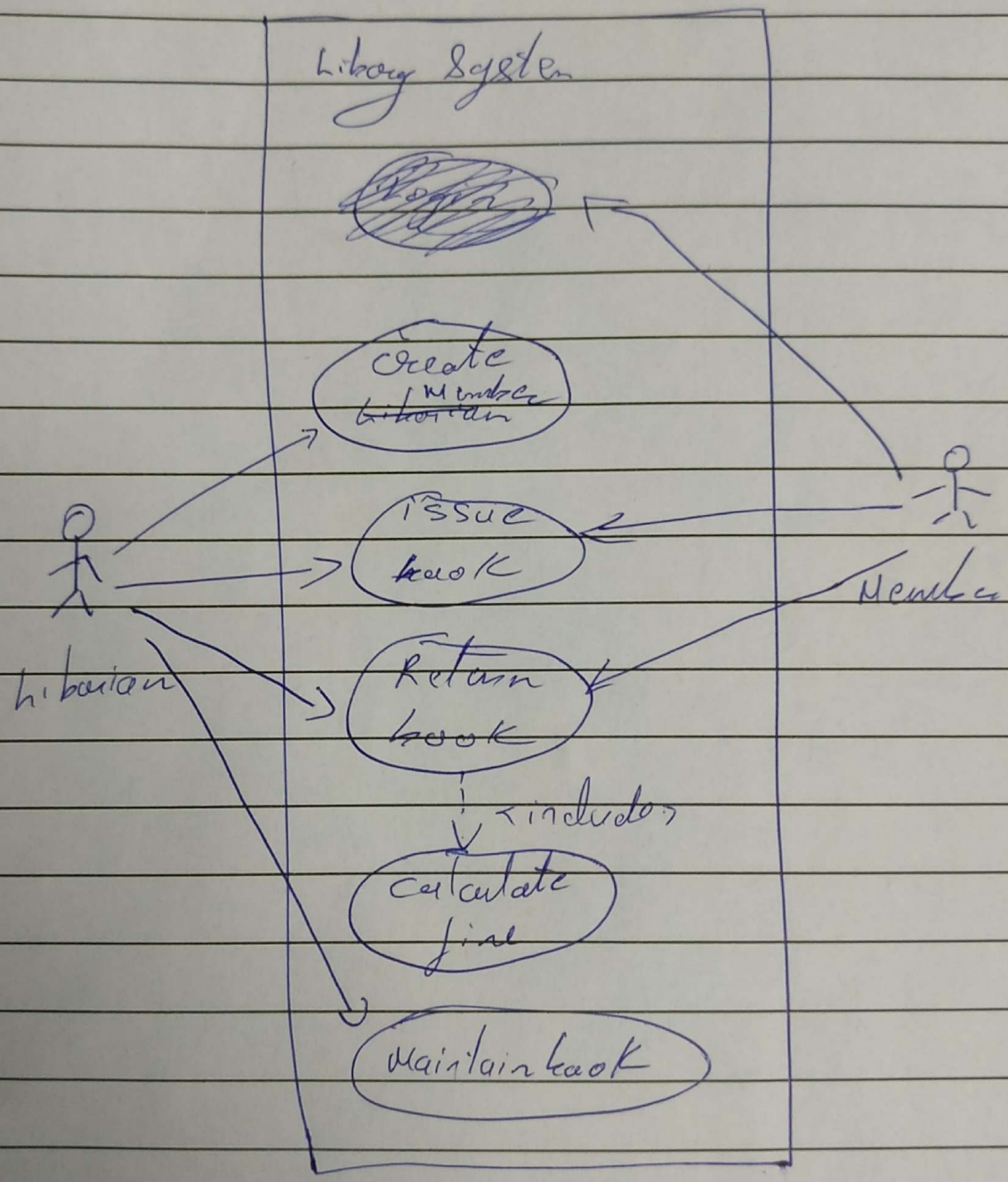
2) Blueprints: It is about completeness. the forward engineering the idea is that the blueprint are developed by respective designers that lay out all the details required for a programmer to code in a form of a design. So that programmer should be able to follow it easily. Blueprints may be used for all details, or a designer may have double blueprints to a particular area. In reverse engineering, blueprint aims to convey details information about the code either in paper document or as an interactive graphical browser. the blueprint can show every details about a class in a graphical form that's easier for developers to understand.

programming language: developers draw UML diagram that are compiled directly to executable code, and the UML become the source code.

b) Draw class diagram for a video store.



c Draw use case diag for a college library system



MAY/JUNE 2016

a. How is dynamic binding achived in c++? Explain what will be the output of the foll code

```
#include <iostream>
using namespace std;
class nonvirtual
{ int x;
  public: void func();
};
class withvirtual
{ int x;
  public: virtual void func();
};
int main()
{ cout<< "sizeof(nonvirtual) = "<< sizeof(nonvirtual)<<endl;
cout<< "sizeof(withvirtual) = "<< sizeof(withvirtual)<<endl;
return 0;
}
```

output:

sizeof nonvirtual 14

sizeof virtual 16.

Dynamic binding is the result of virtual function. Dynamic binding means that the bodies of the code in a member function invocation is determined at the last possible moment based on dynamic type of the object at runtime.

b) Trace the output or correct if error.

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{ float i;
  for (i=0.25; i<=1; i=i+0.25)
  { cout.precision(5);
    cout.width(7);
    cout<<i;
    cout<<width(10);
    cout<<i+i<<"\n";
  }
cout<<setw(10)<<"Total ="<<setw(7)<<setprecisia
(3)<<setiosflag(ios::showpoint|ios::fixed)
  <<1234.57;
  return 0;
}
```

c) Difference between manipulators and ios functions in influencedable

1) ios function return value which manipulators doesnot
2. we can not create own ios function while we can create our own manipulator.
3) ios function are single and not possible to be combined while manipulators are possible to be applied in chain.
4) ios function need <iostream> while manipulator need <iomanip>
5) ios function are memba function while manipulator are non-memba functions

4.a. Give the pointers associated with the file, write and explain the position of this pointer when the file is opening in

(i) Read mode

ii write mode

iii Append mode.

All file object hold 2 file pointers that associated with the file. The pointers are get pointer (input pointer) and put pointer (output pointer)

Read mode: when a file is opened in read mode, the get pointer is set at the beginning of the file. Hence, it is possible to read the file from the first character of the file.

Write mode: when a file is in write mode, the put pointer is set at the begining of the file. Thus, it allows the write operation from the begining of the file. In case the specific file is already exists, its content will be deleted.

Appends Mode: this mode allow addition of data at the end of the file. when the file is opened in append mode the output pointer is set at the end of the file. In case the ~~file~~ specified file already exists a new file is created and the output is ~~set at the append mode~~ set at the begining of the file. when a pre-existing file is successfully opend in append mode, its content remain safe and new data are appended at the end of the file.