Objective:

- To look into the technical and theoretical aspects of the Polymorphism.

Task 1:

What will be displayed on console for the following code segments?

Q. No. 1.1

```
class IX {
public:  virtual void f()=0;
         virtual void g()=0;
};
Class IY {
public:  virtual void h()=0;
         virtual void show()=0;
};
class concrete: public IX, public IY {
public:  void f() {cout<<"f()";}
         void g() {cout<<"g()";}
         void show()
  {cout<<"show()";}
         void h() {cout<<"h()";}
};
void main() {
        IX* ptr_IX=new concrete;
        IY* ptr_IY=(IY*)(ptr_IX);
        Ptr_IY->show();
}
```

Q. No. 1.2

```
class base {
public:
     virtual void fun() {cout<<"base";}
};
class derive: public base {
private:
    void fun() { cout<<"derive"; }
};
void main() {
        base* ptr=new derive;
        ptr->fun();
}
```

Q. No. 1.3

```
class base1 {
public:
     int a,b;
};
class base2 {
public:
     int c,d;
};
class derive:public base1, public base2 {};
void main() {
        derive obj;
        //Assume: starting address of
       //obj is 100
        cout<<sizeof(obj);
        base1 *ptrb1=&obj;
        base2 *ptrb2=&obj;
        cout<<endl<<ptrb1;
        cout<<endl<<ptrb2;
}
```

Q. No. 1.4

```
class Pet
{
public:
  void eat()        { cout<<"eat"; }
  void speak()      { cout<<"speak"; }
  void sleep()      { cout<<"sleep"; }
  void sleep(int i)  {cout<<"sleep i"; }
};
class Goldfish : Pet {
public:
using Pet::eat;
using Pet::sleep;
};
void main() {
  Goldfish bob;
  bob.eat();
  bob.sleep();
  bob.sleep(1);
  Pet *ptr=(Pet*)&bob;
  Ptr->speak();
}
```

Task-2:

The goal in this problem is to write classes and a main program to simulate different pricing plans for an internet service provider. This exercise concentrates on using inheritance and polymorphic methods (virtual functions). Suppose you work in the marketing department of the Baltimore On-Line (BOL) service provider. You want to determine which payment plan is better for different customers. The types of plans include the Customer, Hacker, and ChatRoom plans:

Table 1: Different ISP Plans

	Customer	Hacker	ChatRoom
Initial Time (hrs)	4	10	unlimited
Initial Cost (dollars)	10	20	50
Additional Rate (dollars / hr)	4.00	2.50	none
Disk Space Limit (MB)	1	50	unlimited
Charge per connection	none	none	0.10

In the Customer plan, customers get 4 hours of connection time a month for $10.00 and pay a fixed rate of $4.00 an hour for access time over 4 hours. The Hacker plan is designed so that customers who spend more time on-line save money by paying a higher monthly fee of $20.00 for the first ten hours, and $2.50 an hour after that. ChatRoom plan members pay $50.00 for unlimited monthly access and also pay an additional charge of 10 cents for each time they dial up.

Write class definitions for Customer and for the derived sub-classes Hacker, and ChatRoom. Customer should contain data members common to all customers, like the number of hours connected for a month, the customer's name, the amount of disk space the customer requires and so on. Also implement a method for computing a bill, computeBill( ) which uses the appropriate algorithm and data members to compute a monthly charge for each type of customer. In all classes be sure to also include any constructors and other methods that you think are needed. The derived classes should contain any data members and initializations specific to it's payment plan, and a specific implemenation for the computeBill() method which overrides the default method from the Customer class. Demonstrate programmatically which plan is better for the following customers:

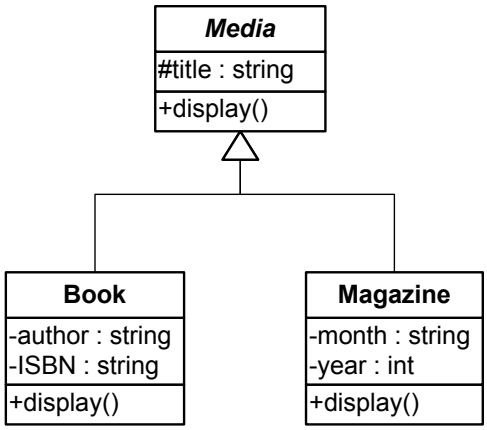
- John Dough, a customer spending 6 hours on-line per month who dials-up 35 times and does not host any web pages (uses zero megabytes of disk space).
- Jane Doe, a customer who spends 18 hours on-line per month, who dials-up 75 times and hosts 30 megabytes of web pages and images. (Note: Customer is not a valid option because of the 1 MB Disk space limit).
- Javiar Dinero, a customer who spends 48 hours on-line per month, who dials-up 90 times, and who hosts 10 megabytes of web pages. (Note: Customer is not a valid option because of the 1 MB Disk space limit).

Task-3:

[Taken: Robert Lafore book]

Task 3.1

In this task, you are required to implement the following inheritance hierarchy:



Declare and implement the *abstract* class **`Media`**. This class will have a protected member variable **title** (of string type) to store the title of the media item. Apart from the overloaded constructor, Media class will have a *pure virtual function* **`display()`**.

Inherit two classes from the Media class, namely: **`Book`** and **`Magazine`**.

The **`Book`** class will have two strings to store the **author name** and **ISBN** of the book. Apart from the overloaded constructor, this class will implement the **`display()`** function which will display all attributes of a Book on screen in a neat and readable way.

The **`Magazine`** class will have a string to store the **month name** and an integer to store the **year of publication** of the magazine. Apart from the overloaded constructor, this class will also implement the **`display()`** function which will display all attributes of a Magazine on screen in a neat and readable way.

Now, implement a main function which should ask the user how many Media items the user wants to create, and store the value entered by the user in an integer variable **n**. Then, your program will dynamically allocate an array of **`Media*`** of size **n**.

After that, your program will ask the user to create **n** Media items. The user should be asked to enter 1 if he/she wants to create a Book and 2 if he/she wants to create a Magazine. Once the choice has been entered, your program should ask the user for all the attributes necessary for creating that item (Book or Magazine). Then, that item should be dynamically allocated and its address should be stored in the array of **Media***.

Once all Media items have been created, your program should traverse the array of **Media*** to display the details of each item on screen.

At the end, your program should properly deallocate all the dynamically allocated memory.

Task # 3.2

In order to accomplish this task, you will firstly need to implement a public member function **`int getYear()`** in the **`Magazine`** class.

Modify the program that you wrote in Task # 1.1 and implement a global function:

void searchByYear (Media, int)**

which takes the array of **Media*** and its **size** as parameters. This function should ask the user to enter a year, then it should search the array for all **magazines** of that year and display their details on screen.

Now, call the above function from the main function to search the array of **Media*** once all media items have been created.

Task # 3.3

Add another class **CD** to the inheritance hierarchy. The **CD** class will have an integer member variable to store the its **capacity** in MBs. Apart from the overloaded constructor, this class will also implement the **display()** function which will display all attributes of a CD on screen in a neat and readable way.

Also modify the main function to allow the user to create a CD by entering the option 3.

Task # 3.4

Add a **Shelf** class to store a list/collection of **Media** items. It will have the following declaration:

```
class Shelf
{
private:
        Media** items;
        int maxSize;
        int currSize;
public:
        Shelf (int);
        void insert (Media*);
        void displayContents ();
        ~Shelf();
};
```

The overloaded constructor will take an integer value as argument and initialize the **maxSize** to that value, and initialize **currSize** to 0. Constructor will also dynamically allocate an array of **Media*** through the member variable **items**.

The member function **insert(Media*)** will take a **Media*** as parameter, and store that pointer into the next available index in the **items** array (use the member variable **currSize** to determine the next available index in the array). This function will also increment **currSize** to indicate the updated size of the array.

The member function **displayContents()** will display the details of all the items that are currently stored in the **items** array. This will be accomplished, simply, by calling the display member function on each Media item stored in the **items** array.

The destructor of the **Shelf** class will deallocate the dynamically allocated array which was allocated by the constructor.

Now, implement a main function, which should ask the user how many Media items the user wants to create, and declares a **Shelf** object to store those many items. After that the user should be asked to create those many Media items. After the creation of each Media item (Book, Magazine, or CD), it should be inserted into the Shelf. Once all items have been inserted, their details should be displayed by calling the **displayContents()** function.