PCPF Lab
Lab Assignment number 03

Name: Aamir Ansari **Batch:** A **Roll no.** 01

Aim: Implementation of Dynamic Binding

Problem Statement:

Write a program to design a class representing the information regarding digital libraries having books and tapes. Book and Tape class are derived from a common class called Media. (Think over data members of Media)
Class Book Data Members - title, price, pages
Member functions -Parameterized constructors
-Display book details
Class Tape Data Members - title, price, time
Member functions -Parameterized constructors
- Display tape details
The program should demonstrate Runtime Polymorphism. (Hint: Use virtual function)

Theory:

Dynamic Binding

C++ provides facility to specify that the compiler should match function calls with the correct definition at the run time; this is called dynamic binding or late binding or run-time binding. Dynamic binding is achieved using virtual functions. Base class pointer points to derived class object. And a function is declared virtual in base class, then the matching function is identified at run-time using virtual table entry.

Runtime Polymorphism:

Runtime polymorphism: This type of polymorphism is achieved by Function Overriding. Function overriding on the other hand occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden.
// C++ program for function overriding

Virtual Functions:

```
#include <bits/stdc++.h>
using namespace std;
class base  {
public:
   virtual void print ()
   { cout<< "print base class" <<endl; }
   void show ()
   { cout<< "show base class" <<endl; }
};
class derived : public base  {
public:
    void print ()
   { cout<< "print derived class" <<endl; }
   void show ()
   { cout<< "show derived class" <<endl; }
};
int main()  {
   base *bptr;
   derived d;
   bptr = &d;

   //virtual function, binded at runtime (Runtime polymorphism)
   bptr->print();

   // Non-virtual function, binded at compile time
   bptr->show();

   return 0;
}
// output
print derived class
show base class
```

Pure Virtual Functions:

Sometimes implementation of all function cannot be provided in a base class because we don't know the implementation. Such a class is called abstract class. For example, let Shape be a base class. We cannot provide implementation of function draw() in Shape, but we know every derived class must have implementation of draw(). Similarly an Animal class doesn't have implementation of move() (assuming that all animals move), but all animals must know how to move. We cannot create objects of abstract classes.
A pure virtual function (or abstract function) in C++ is a virtual function for which we don't have implementation, we only declare it. A pure virtual function is declared by assigning 0 in declaration. See the following example.

Program:

```
// code


#include <iostream>
#include <cstring>

using namespace std;

// parent class Media
class Media {
   public:
      // instance variables
      int id, yearOfPublish;

      // constructor
      Media(int id, int yearOfPublish) {
         this->id = id;
         this->yearOfPublish = yearOfPublish;
      }

      //  virtual method which is overloaded
      virtual void display() {}

};
// sub-class Book
class Book : public Media {
   public:
      // instance variables
      string title;
      float price;
      int pages;
      // constructor
      Book(int id, int yearOfPublish ,string title, float price, int pages)
      :Media(id, yearOfPublish) {
         this->title = title;
         this->price = price;
         this->pages = pages;
      }
      // method to display
      virtual void display() {
         cout << "Book's ID :  " << this->id << endl;
         cout << "Book's Year of publish :  " << this->yearOfPublish << endl;
         cout << "Book's title' :  " << this->title << endl;
         cout << "Book's price :  " << this->price << " rs" << endl;
         cout << "Number of pages in book :  " << this->pages << " pages" << endl;

      }
};

// sub-class Tape
class Tape : public Media {
   public:
```

```
        // instance variables
        string title;
        float price;
        float time;
        // constructor
        Tape (int id, int yearOfPublish, string title, float price, float time)
        :Media(id, yearOfPublish) {
            this->title = title;
            this->price = price;
            this->time  = time;
        }
        // method to display
        virtual void display() {
            cout << "Tape's ID :  " << this->id << endl;
            cout << "Tape's Year of publish :  " << this->yearOfPublish << endl;
            cout << "Tape's title' :  " << this->title << endl;
            cout << "Tape's price  :  " << this->price << " rs" << endl;
            cout << "Tape's length :  " << this->time << " hours" << endl;

        }
};

int main() {
    // object of Book
    Media *myBook = new Book(1001, 2077, "To sleep in sea of stars", 399.99, 425);
    myBook->display();

    cout << "*****************************************" << endl;

    // object of Tape
    Media *myTape = new Tape(2002, 2001, "2001 Space odessey", 449.99, 2.5);
    myTape->display();
}


// output
Book's ID :  1001
Book's Year of publish :  2077
Book's title' :  To sleep in sea of stars
Book's price :  399.99 rs
Number of pages in book :  425 pages
*****************************************
Tape's ID :  2002
Tape's Year of publish :  2001
Tape's title' :  2001 Space odessey
Tape's price  :  449.99 rs
Tape's length :  2.5 hours
```