**Lab 10**

**Polymorphism**

# Objectives

**Following programming skills will be acquired in this lab:**

* To understand the use of pointer to a base class.
* To practice the syntax of defining a **virtual** function.
* To understand use of **abstract** classes.
* To understand the use of **pure** virtual functions.

**Pointers to Base Class**

One of the key features of derived classes is that a pointer to a derived class is type-compatible with a pointer to its base class. Polymorphism is the art of taking advantage of this simple but powerful and versatile feature, that brings Object Oriented Methodologies to its full potential.

# Example 10.1: Pointers to Base Class

|  |  |
| --- | --- |
| #include <iostream>  using namespace std;    class **Polygon** { protected:  int width, height;  public:  void setValues (int a, int b)  { width=a; height=b; }  };    **class Rectangle: public Polygon**  { public:  int area ()  { return (width \* height); }  };    **class Triangle: public Polygon**  { public:  int area ()  { return (width \* height / 2); }  };    int **main** () {  Rectangle rect;  Triangle trgl;    Polygon \* ppoly1 = &rect;  Polygon \* ppoly2 = &trgl;    ppoly1->setValues (4,5);  ppoly2->setValues (4,5);    cout << rect.area() << endl;  cout << trgl.area() << endl;  return 0;  } | **Output**:    20  10 |

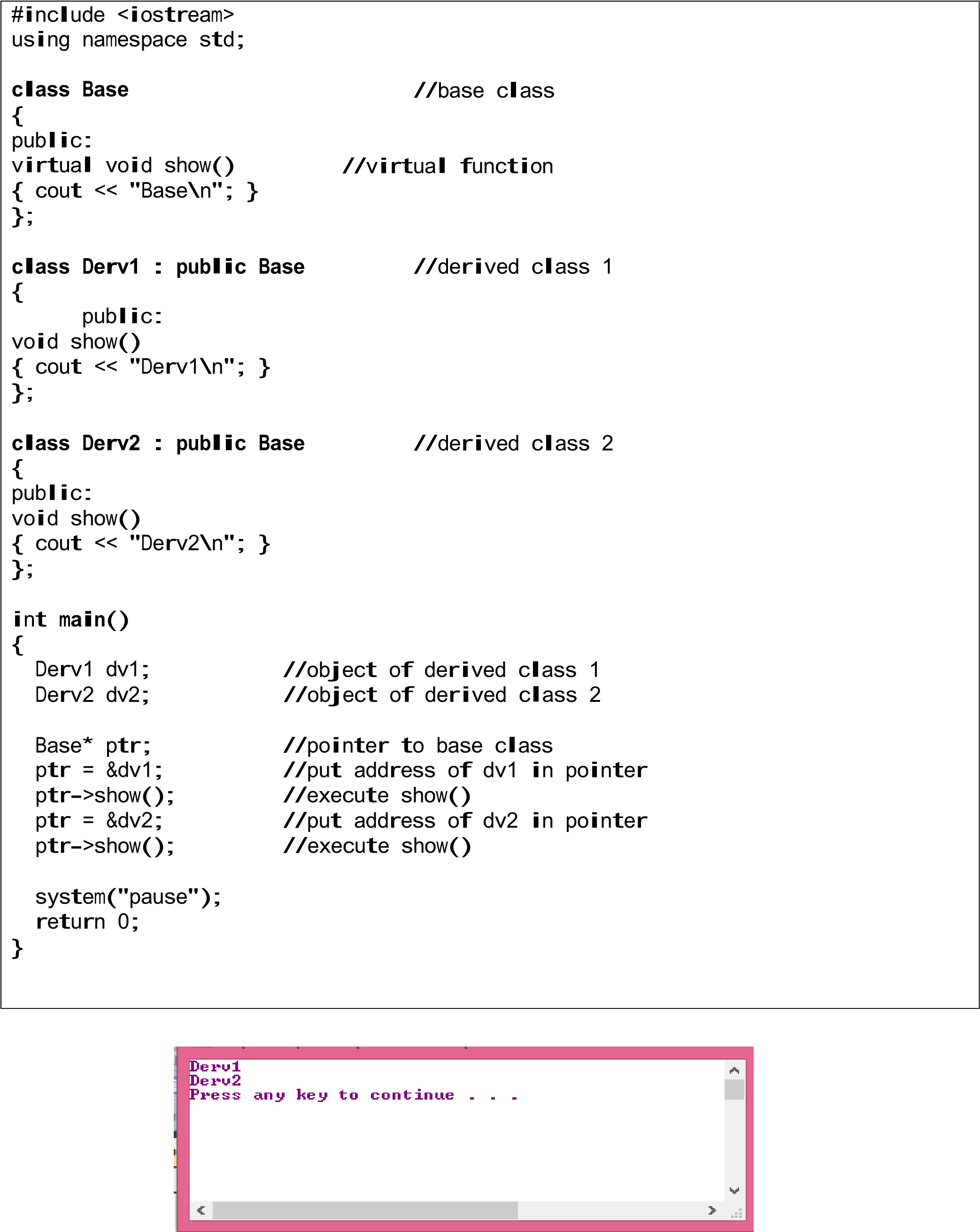
**Virtual Members**

A member of a class that can be redefined in its derived classes is known as a virtual member. In order to declare a member of a class as virtual, we must precede its declaration with the keyword virtual. A class that declares or inherits a virtual function is called a *polymorphic class*.Note that despite of its virtuality, we have also been able to declare an object of type Polygon and to call its own area() function, which always returns 0.

# Example 10.2

|  |  |
| --- | --- |
| #include <iostream> using namespace std;    class **Polygon** {  protected:  int width, height;  public:  void **setValues** (int a, int b)  { width=a; height=b; }    **virtual int area** ()  { return (0); }  };    **class Rectangle: public Polygon**  { public:  int area ()  { return (width \* height); } };    **class Triangle: public Polygon**  { public:  int area ()  { return (width \* height / 2); } };    int **main** ()  {  Rectangle rect;  Triangle trgl;  Polygon poly;    Polygon \* ppoly1 = &rect;  Polygon \* ppoly2 = &trgl;  Polygon \* ppoly3 = &poly; | ppoly1->setValues (4,5);  ppoly2->setValues (4,5);  ppoly3->setValues (4,5);    cout << ppoly1->area() << endl;  cout << ppoly2->area() << endl;  cout << ppoly3->area() << endl;  return 0;  }    **Output**:  20  10  0 |

**Example 10.3 Virtual Functions Assessed with Pointers**



# Pure Virtual Function

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.

# Abstract Base Classes

An abstract class is a class that is designed to be specifically used as a base class. An abstract class contains at least one pure virtual function. We cannot create objects of an abstract class.

Virtual members and abstract classes grant C++ the polymorphic characteristics that make object-oriented programming such a useful instrument in big projects.

## Example 10.4

|  |  |
| --- | --- |
| #include <iostream>  using namespace std;    class **Polygon**  { protected:  int width, height;  public:  void setValues (int a, int b)  { width=a; height=b; }  **virtual int area (void) = 0;**  void printarea (void) { cout << this->area() << endl; } };    **class Rectangle: public Polygon**  { public:  int area (void) { return (width \* height); }  };    **class Triangle: public Polygon** {  public:  int area (void) { return (width \* height / 2); } };    int **main** () {  Rectangle rect;  Triangle trgl;  Polygon \* ppoly1 = &rect; Polygon \* ppoly2 = &trgl; ppoly1->setValues (4,5);  ppoly2->setValues (4,5);  ppoly1->printarea();  ppoly2->printarea();  return 0;  } | **Output**:    20  10 |

**Example 10.5 Using Abstract Base Class**

|  |  |
| --- | --- |
| #include <iostream>  using namespace std;    **class person** //person  class  { protected:  char name[40]; public:  void getName()  {  cout << "Enter name: ";  cin >> name; }  void putName()  {  cout << "Name is: " << name << endl; }  virtual void getData() = 0; //pure virtual  virtual bool isOutstanding() = 0; //pure  };    **class student : public person**  { private:  float gpa;  public:  void getData()  {  person::getName();  cout << "Enter student’s GPA: ";  cin >> gpa;  }  bool isOutstanding()  {  If (gpa > 3.5) { return true; }  else { return false; }  };    **class professor : public person**  {  private:  int numPubs;  public:  void getData()  {  person::getName();  cout<< "Enter number of professor's publications: ";  cin >> numPubs;  }  }; | bool isOutstanding()  {  if(numPubs > 100) {return true;}  else {return false;}  }    int **main**()  {  person\* persPtr[10];  int n = 0;  char choice;  do  {  cout << "Enter student or professor (s/p): ";  cin >> choice;  if(choice=='s')  persPtr[n] = new student;  else  persPtr[n] = new professor;  persPtr[n++]->getData();  cout << "Enter another (y/n)? ";  cin >> choice;  }    while( choice=='y' );  for(int j=0; j<n; j++)  {  persPtr[j]->putName();  if( persPtr[j]->isOutstanding() )  **{**  cout << "This person is outstanding\n";  }  }    return 0;  } |

**Example 10.6**

|  |
| --- |
| #include <iostream>  using namespace std;  **class vehicle**  {  int wheels;  float weight;  public:  virtual void **message**(void)  {cout<<"Vehicle message, from vehicle, the base class\n";}  };  **class car : public vehicle**  { int passengerLoad;  public:  void **message**(void) // second message()  {cout<<"Car message, from car, the vehicle derived class\n";}  };  **class truck : public vehicle**  {  int passengerLoad;  float payload;  public:  int passengers(void) {return passengerLoad;}  };  **class boat : public vehicle**  {  int passengerLoad;  public:  int passengers(void)  {return passengerLoad; }  void **message**(void) // third message()  {cout<<"Boat message, from boat, the vehicle derived class\n";}  }; |

|  |
| --- |
| **int** **main**()  {  vehicle unicycle;  car sedanCar;  truck trailer;  boat sailboat;  cout<<"Adding virtual keyword at the base class method\n";  cout<<" \n";  unicycle.message();  sedanCar.message();  trailer.message();  sailboat.message();  **system("pause");**  **return 0;**  **}** |

**Exercises**

**Tasks 1 :**

Answer the following/Write C++ code where required.

|  |  |
| --- | --- |
| 1. | Consider a class Square derived from a class Quad. Declare a pointer to Quad and assign the address of an object of class Square to this pointer. |
| Quad\* q1;  Square s1;  q1 = &s1; | |
| 2. | Repeat the above using dynamic memory allocation, i.e. allocate memory for an object of class Square and assign the address to a pointer of class Quad. |
| Quad\* quadPtr = new Square(); | |
| 3. | Both Quad and Square have a method int area(); Which of the methods is called by the following statement.  Quad quad, \*ptrQuad;  Square sq;  ptrQuad = &sq;  quad.area()  ptrQuad -> area(); |
| The function inside the Quad class will be called | |
| 4. | Repeat exercise 3 assuming that area() is declared as virtual function. |
| The functions from the respective classes will be called. | |
| 5. | What is an abstract class? What makes a class abstract? |
| An abstract class is a class whose object cannot be made directly. A class becomes an abstract class when it has at-least one pure virtual function. It is to be used as a base class. | |

### Exercise 10.1

Create an abstract class Faculty with the fields name and ID. Provide a pure virtual function salary (). Derive a class Permanent Faculty from Faculty. The class has additional attributes years of service and basic pay.

The salary of permanent faculty is computed as the sum of basic pay, medical allowance, and house rent. Medical allowance is 10% of basic pay and house rent is 25% of the basic pay.

Derive another class visiting faculty from Faculty. The class has attributes per hour rate and number of hours taught. The salary of visiting faculty is computed by multiplying the per hour rate with the number of hours taught.

Write a program to declare two pointers of class Faculty. Create an object each of visiting and permanent faculty, as sign their addresses to pointers of base class, set the value of data members and call the salary function for each.

Code:

#include <iostream>

using namespace std;

class Faculty

{

private:

string name;

int ID;

public:

virtual void salary() = 0;

};

class permanentFaculty : public Faculty

{

private:

int years\_of\_service;

int basic\_pay;

public:

void setValue(int a, int b)

{

years\_of\_service = a;

basic\_pay = b;

}

void salary() override

{

int sal;

sal = basic\_pay + basic\_pay \* (10 / 100) + basic\_pay \* (25 / 100);

cout << "Salary : " << sal << endl;

}

};

class visitingFaculty : public Faculty

{

private:

int per\_hour\_rate;

int number\_of\_hours\_taught;

public:

void setValue(int a,int b)

{

per\_hour\_rate = a;

number\_of\_hours\_taught = b;

}

void salary() override

{

float sal;

sal = per\_hour\_rate \* number\_of\_hours\_taught;

cout << "Salary : " << sal << endl;

}

};

int main()

{

Faculty\* f1;

Faculty\* f2;

permanentFaculty p1;

visitingFaculty v1;

f1 = &p1;

f2 = &v1;

p1.setValue(10, 20000);

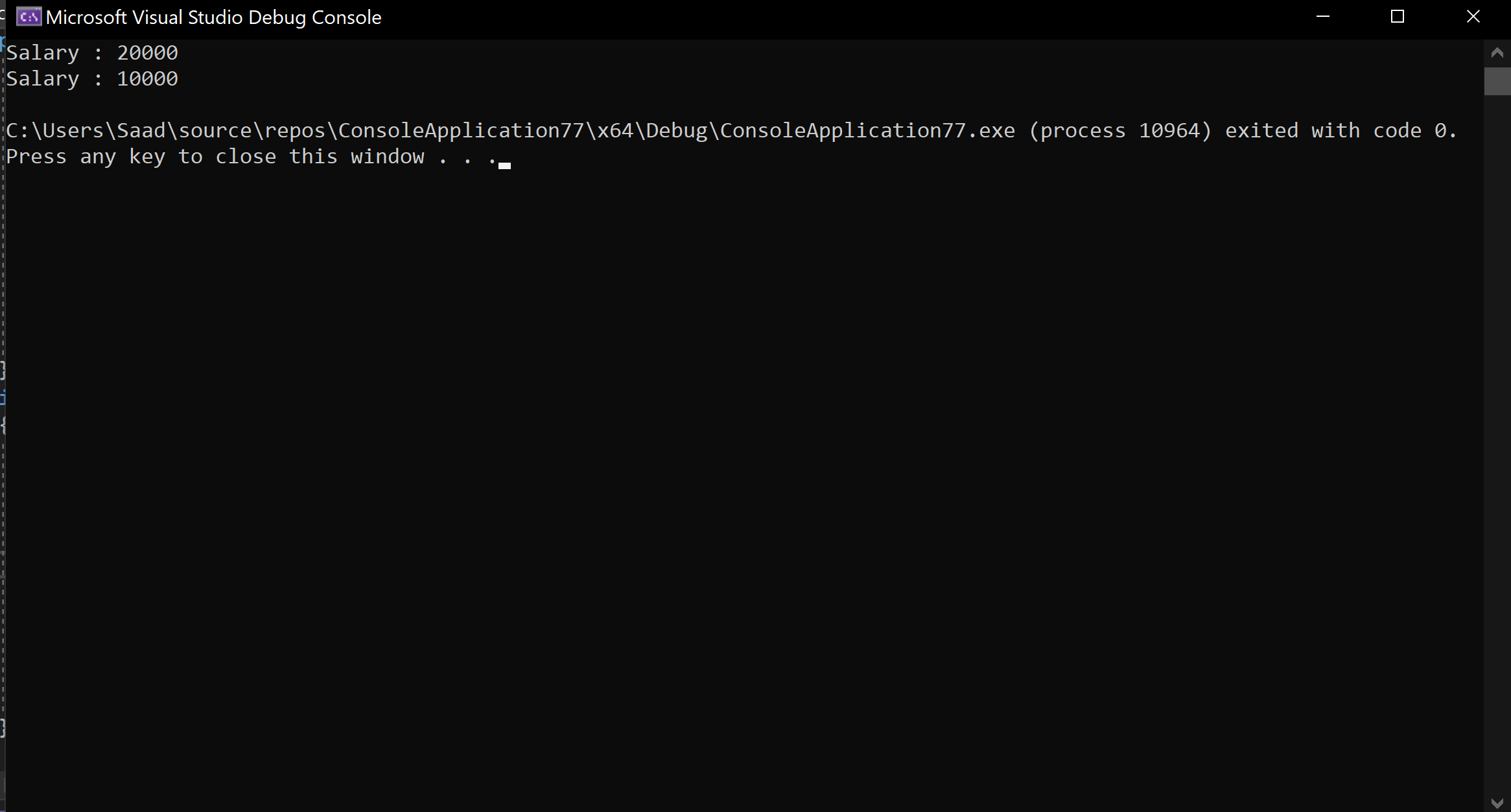
v1.setValue(500, 20);

f1->salary();

f2->salary();

}

Output:



### Exercise 10.2

Modify the above program to declare an array of pointers to Faculty. Using dynamic memory allocation, create an object of permanent or visiting faculty as indicated by the user (get user choice). Once the user has entered data for faculty, call the salary method for each object and display the salary.

**Code:**

#include <iostream>

using namespace std;

class Faculty

{

private:

string name;

int ID;

public:

virtual void salary() = 0;

virtual void setValue(int a, int b) = 0;

};

class permanentFaculty : public Faculty

{

private:

int years\_of\_service;

int basic\_pay;

public:

void setValue(int a, int b)

{

years\_of\_service = a;

basic\_pay = b;

}

void salary() override

{

int sal;

sal = basic\_pay + basic\_pay \* (10 / 100) + basic\_pay \* (25 / 100);

cout << "Salary : " << sal << endl;

}

};

class visitingFaculty : public Faculty

{

private:

int per\_hour\_rate;

int number\_of\_hours\_taught;

public:

void setValue(int a,int b)

{

per\_hour\_rate = a;

number\_of\_hours\_taught = b;

}

void salary() override

{

float sal;

sal = per\_hour\_rate \* number\_of\_hours\_taught;

cout << "Salary : " << sal << endl;

}

};

int main()

{

Faculty \*f[10];

int n = 0;

char choice;

int a, b;

do

{

cout << "Enter visitng faculty or permanent faculty (v/p): ";

cin >> choice;

if (choice == 'v') {

f[n] = new visitingFaculty;

cout << "Enter per hour rate and number of hours taught" << endl;

}

else {

f[n] = new permanentFaculty;

cout << "Enter years of service and basic salary" << endl;

}

cin >> a >> b;

f[n++]->setValue(a,b);

cout << "Enter another (y/n)? ";

cin >> choice;

}

while (choice == 'y');

for (int j = 0; j < n; j++)

{

f[j]->salary();

}

}

Output:

