National University of Computer and Emerging Sciences

Lab Manual 11
Object Oriented Programming

Course Instructor	Mr. Usama Hassan
Lab Instructor (s)	Ms. Fariha Maqbool Mr. Sohaib Ahmad
Section	BSE-2C
Semester	Spring 2023

Department of Computer Science
FAST-NU, Lahore, Pakistan

Objectives

After performing this lab, students shall be able to:

- ✓ Learn and identify the need of polymorphism.
- ✓ Implement virtual functions.
- ✓ Know the difference of calling virtual function by pointer and by object.

Exercise 1:

For the first part of this lab, we are going to see if base class pointers can point to an object of derived class and vice versa. Perform the following steps

- Create a class called Animal.
- Message is data member in Animal class.
- An animal can speak so create a public method named speak in the which returns a "Message"
- Modify the definition of the speak method so that it returns the string "speak() called.".
- A Dog is an Animal. Create a class named Dog. Use public inheritance.
- The Dog class will inherit the speak method from Animal. Override this method in the Dog class so that it returns "woof!" when called.
- Add the following lines in the main function of your program, note the output.

```
Animal objAnimal;
Dog objDog;
Animal *ptrAnimal = &objAnimal;
Dog *ptrDog = &objDog;

cout << objAnimal.speak() << endl;
cout << objDog.speak() << endl;
cout << ptrAnimal->speak() << endl;
cout << ptrDog->speak() << endl;
```

You can see that we have created two objects, one for each class and two pointers in the same manner. The pointer to Animal is pointing to an object of the class Animal and the pointer to Dog is pointing to the object of class Dog. In this example, ptrAnimal called the speak method of the class Animal where as ptrDog called the speak method of the class Dog. If we want to use the definition of the base class method that we have overloaded in a derived class from a derived class pointer, we have to follow this syntax.

Exercise 2:

Change the main function of your program and replace it with the code below. Execute it and paste the output in the box below.

```
Dog lassie;
Animal *myPet = &lassie;
cout << myPet->speak() << endl;
```

You will see that as expected, the speak method of the base class was called. What if we wanted to use the definition of the derived class function? To accomplish this, we can add the keyword virtual to the declaration of the speak() method in the Animal class. Specifying a function as

virtual makes sure that whenever we use a base class pointer pointing to an object of a derived class to call a function, the definition of the method declared in the derived class is used. Modify the speak method of the Animal class as shown below, compile your code, execute it and paste the output in the space provided below.

```
virtual char *  speak()
{
      return "speak() called.";
}
```

Exercise 3:

In the above exercises, we have seen a very simple implementation of Polymorphism. The real power of this feature is realized when we have a collection of objects of multiple derived classes and we use a pointer of the base class to call their respective overloaded methods. A Cat is an animal too. Let's see how we can use an array of base class pointers to utilize the essence of polymorphism.

- Define a class Cat. Inherit publicly from Animal just like we did in class Dog.
- Overload the speak() method so that it returns "mew!".
- Modify the main function as shown below.
- Compile, execute and paste the output in the space given below.

```
const int size = 2;
Animal * myPets[size];
Cat whiskers;
Dog mutley;

myPets[0] = &whiskers;
myPets[1] = &mutley;

for(int i=0; i<size; i++)
      cout << myPets[i]->speak() << endl;
```

Exercise 4:

Modify the main function so that the size of array myPets is 5. Display a menu to the user asking him the type of pet for each of the 5 pets. For each input, create the object of the respective class dynamically and point to it by the corresponding pointer of the array myPets. Once you have

taken all 5 inputs, use a loop to call the speak() method and delete each of the objects. You can use the following code to take the input. The declaration of getch() is in the header file conio.h.

```
int i = 0;
while (i<size)
{
	cout << "Press 1 for a Dog and 2 for a Cat." << endl ;
	switch (getch())
	{
		case '1':
			myPets[i] = new Dog;
			cout << "Dog added at position "<< i <<endl<<endl;
			i++;
			break;
		case '2':
			myPets[i] = new Cat;
			cout << "Cat added at position "<< i <<endl<<endl;
			i++;
			break;
		default:
			cout<<"Invalid input. Enter again." <<endl<<endl;
			break;
	}
}
```

Exercise 5:

Although things seem to be fine on the surface, there is a problem in the program we just wrote. To observe this problem, we must add destructors for all classes. Paste the following inline definitions of the destructors in their corresponding classes, execute the program and paste the output below.

```
~Animal()	{ cout << "~Animal() called."<<endl;	}
```

```
~Cat()		{ cout << "~Cat() called."<<endl;		}
```

```
~Dog()		{ cout << "~Dog() called."<<endl;		}
```

Can you see what went wrong? When using delete to deallocate memory, only the base class destructor is called whereas the derived class destructor is not called at all. Although this is fine in the example we are using here but it will create memory leaks if there are any dynamically allocated variables in any of the derived class. To avoid this we declare the base class destructor as virtual. Doing this will make sure that the derived class destructor is called even if you are using a base class pointer to call the destructor. Now change the definition of the base class destructor to make it virtual, execute the program and paste the output in the box given below. Make sure you can see the derived class destructors being called in the output.