CS 201: Problem Solving & Programming II

Lab #9

The warm up

Remember we worked on Animal class on lab 6. We will use the same scenario to practice polymorphism and virtual function. Again let’s assume we have four classes: Animal, Human, Bird and Dog.

**The Animal class**

The class will have one virtual functions, which its descendents must override. Here the descendents will be: Human, Bird and Dog class.

|  |
| --- |
| **Animal** |
| - m\_speciesName : string |
| + Animal()  + Animal(string& speciesName)  + Action(): void **virtual** |

1. First create the Animal class.
2. Then create the three child classes: Human, Bird and Dog. These classes don’t need any other variables or functions, just override the virtual function of their base class.
3. The Action function of each class will print different text like-

Animal: I am a living thing.

Human: I can walk, run and tame other animals.

Dogs: Huffs! I like my master.

Bird: I just love to fly, pity the humans can’t!

1. Create a main.cpp file and create one object of each class.
2. Call the *Action* function of all the objects. Make sure each call is showing its related text.

Exercise 1

We just created four objects and called the *Action* function in them. However we can use polymorphism here. Comment in the codes that called the action functions. Then do the following:

1. Create a pointer of Animal class.
2. Have the pointer point to the reference of human object. For example:

Animal \*animal;

Human human(“human”);

animal = &human;

1. Call the action function using the pointer. See the result in the console.

animal.Action();

Is anything unusual happened? Why?

1. Similarly, use the pointer to point dog object and call the action function.
2. Why do you think different output is shown for same action call? Write it down in comments just after the code.

Exercise 2

We have three children of Animal class. However, we can’t make sure that all the children must override the Action function. To achieve that, we have to make the function pure virtual.

1. Now make the Action function of the Animal class pure. Comment in the definition of Action function in the Animal class. Recall from Rachel’s slides on how to make a function pure virtual.
2. Compile the program, is it working? Ask the instructor if not.
3. Go back to main.cpp file. You have a pointer of Animal class.
4. Create a new object of Animal class (not a pointer). Is it showing error? Why?

Why can we make a pointer of Animal class where we can’t do the same for object? Write it down in a comment.

Exercise 3

For this exercise we don’t need the Animal class to be abstract. Restate the Action method to virtual again (not pure virtual) and redefine it as you did in the warm up. To make sure, use/create an object of Animal and call its Action function. Check if it’s printing “I’m a living thing”.

1. You already have a pointer of Animal class (if not, create one and have it point to an object of animal class).
2. Now create two more pointers: one of Human, another of Dog class. Have the two pointer point to the objects you created earlier. For example,

Human human(“human”);

Human \*ptrh = &human;

Dog dog(“dog”);

Dog \*ptrd = &dog;

1. Now do the following **dynamic casting**, compile and run. If it fails to compile or run, try to find out why. State the reason in a comment.
   1. *Base class to derived class* (Assume, ptra is a pointer of Animal class):

ptra = dynamic\_cast<Human\*>(ptrh);

ptra -> Action();

* 1. *Derived class to base class:*

ptrh = dynamic\_cast<Animal\*><ptra);

ptrh->Action();

1. Do the same as 3.a and 3.b using **static casting** (use static\_cast instead of dynamic\_cast). What happens now?
2. Now use **reinterpret cast** to do the same actions. Is it working?
3. Also try this for reinterpret cast:

ptrh -> reinterpret\_cast<Dog\*>(ptrd);

ptrh -> Action();