**Lab8ResultKenWagner.doc.** Use the file to save your answers to questions and/or work accomplished below each corresponding numbered task.

**Lab8 Topic: “Non-Polymorphism, Polymorphism and Virtual Functions, Why**

**using polymorphism?”**

**Concept/Rule 1**: A base class and derived class(es) can have same function

names.

**Concept/Rule 2**: A non-polymorphic program uses static binding which happens

at compile time to call a base-class member function that has

the same member function name as defined in a derived class.

**Concept/Rule 3**: A polymorphic program uses dynamic binding which happens

at run time to call a common member function name that is

defined in the base class and derived classes.

**Concept/Rule 4**: To make a program polymorphic, the base class must have a

virtual function and its name must be the same name as defined

in derived classes.

**Concept/Rule 5**: To make the base class to be an abstract class, a pure virtual

must be declared by assigning 0 to it. The pure virtual function

should not be implemented in the base class, but it must be

declared and defined/implemented in derived class(es)

**Concept/Rule 6**: An object of an abstract class cannot be instantiated.

**Your Tasks:**

1. Compile the following program, Lab8Program1.cpp. Display a screenshot of the program output.

#include <iostream>

#include <string>

using namespace std;

class Animal

{

public:

Animal()

{ who ="Not Human";

}

void displayID();

private:

string who;

};

void Animal::displayID()

{ cout<< who << endl;

}

class Cat :public Animal

{

public:

Cat()

{ me ="I am a CAT!!!";

}

void displayID();

private:

string me;

};

void Cat::displayID()

{ cout<< me << endl;

}

class Dog :public Animal

{

public:

Dog()

{ me ="I am a DOG!!!";

}

void displayID();

private:

string me;

};

void Dog::displayID()

{ cout<< me << endl;

}

//A regular function for the main.

void PrintID( Animal &); // function prototype

int main()

{ //Process Animal

Animal anyThing;

PrintID(anyThing);

//Process a cat

Cat catObj;

PrintID(catObj); // call the PrintID that calls a memberfunction displayID.

//Process a dog

Dog dogObj;

PrintID(dogObj);

cin.ignore();

cin.get();

return 0;

}

void PrintID( Animal &object) // function definition

{

object.displayID();

}

1. Based on Task #1 done, are some concepts and rules proven? Which ones? Explain.

*Concept 1 is proven because the function void PrintID is used with different parameters by different objects.*

*Concept 2 is proven because the argument of PrintID function will only refer to each class according to the compiler.*

1. Compile the following program, Lab8Program2.cpp. Display a screenshot of the program output

#include <iostream>

#include <string>

using namespace std;

class Animal

{

public:

Animal()

{ who ="Not Human";

}

virtual void displayID() = 0; // pure virtual function. This makes the Animal class

// an Abstract class.

virtual int numlegs();

virtual ~Animal()

{

cout <<"Virtual destructor " << endl;

}

private:

string who;

};

void Animal::displayID()

{ cout<< who << endl;

}

int Animal::numlegs()

{

return 0;

}

class Cat :public Animal

{

public:

Cat()

{ me ="I am a CAT!!!";

}

~Cat()

{

cout <<"Virtual destructor in Cat " << endl;

}

void displayID();

int numlegs();

private:

string me;

}; // end class

void Cat::displayID()

{ cout<< me << endl;

}

int Cat::numlegs()

{

return 4;

}

class Bird :public Animal

{

public:

Bird()

{ me ="I am a Bird!!!";

}

~Bird()

{

cout <<"Virtual destructor in Bird ." << endl;

}

void displayID();

int numlegs();

private:

string me;

};

void Bird::displayID()

{ cout<< me << endl;

}

int Bird::numlegs()

{

return 2;

}

class Dog :public Animal

{

public:

Dog()

{ me ="I am a DOG!!!";

}

~Dog()

{

cout <<"Virtual destructor in Dog ." << endl;

}

void displayID();

int numlegs();

private:

string me;

};

void Dog::displayID()

{ cout<< me << endl;

}

int Dog::numlegs()

{ return 4;

}

//A regular function for the main.

void PrintID( Animal &); // function prototype

int main()

{ //Process Animal

//Animal anyThing;

//Process a cat

Cat catObj;

PrintID(catObj); // call the PrintID that calls a member function displayID.

//Process a dog

Dog dogObj;

PrintID(dogObj);

cin.ignore();

cin.get();

return 0;

}

void PrintID( Animal &object) // function definition

{

object.displayID();

cout << "Legs = " << object.numlegs()<< endl;

}

1. Based on Task #3 done, are some concepts and rules proven? Which ones? Explain.

*Concept 1 is proven because the function void PrintID is used with different parameters by different objects.*

*Concept 3 is proven because it uses dynamic binding to call a member function defined in class Animal.*

*Concept 4 is proven because the base class has called function “virtual void PrintID () =0;” and since it is also in the derived classes, it is polymorphic.*

*Concept 5 is proven because the base class has called function “virtual void PrintID () =0;”*

*Concept 6 is proven because if you comment out the code “ //void displayID();”*

*The compiler cannot substantiate “cat” because it cannot override the virtual function.*

1. Uncomment the statement //Animal anyThing; in the main ( ). Compile and run to see an error message. What does it say? What concept is proven?

>>

Error 1 error C2259: 'Animal' : cannot instantiate abstract class

<<

*That animal is not a derived class and cannot override the virtual function.*

1. Write a program that proves all concepts and rules of this lab. Use two classes: Person class, and Student class. Name your program **Assessment8ProjectYourname.cpp** . Display a screenshot of the program output.

**What To Turn In**

Submit:

**Lab8ResultYourname.doc** and **Assessment8ProjectYourname.cpp**

Grading Rubric:

|  |  |
| --- | --- |
| Lab8 Completion of Tasks 1- 5 | 10 points |
| Assessment8ProjectYourname.cpp  done in Task #6   * The program runs. * The program proves all Lab 8 Concepts and Rules.. | 4 points  6 points |