```cpp
/*
Daniel Avila April 29th, 2020 Section 19
Lab: Virtual Interfaces
Description: Using virtual functions to test for abstraction
Description: from using inheritance
*/
//#include "Creature.h" //can include but isn't necessary
#include "Player.h"//for player object
//#include "Monster.h" //can include but isn't necessary
#include "WildPig.h"//for wildpig object
#include "Dragon.h"//for dragon object
#include <memory>
int main()
{

    //makes pointer with overloaded argument using a name-Timmy
    shared_ptr<Player> player = make_shared<Player>("Timmy");
    player->DrawOnScreen();//Points to function that prints the action while also
                           // calling another virtual function that prints the name

    //makes pointer with overloaded argument using a name-UFO
    shared_ptr<Monster> monster = make_shared<Monster>("UFO");
    monster->DrawOnScreen();//Points to function that prints the action while also
                           // calling another virtual function that prints the name

    //makes pointer with overloaded argument using a name-Piglet
    shared_ptr<WildPig> wildpig = make_shared<WildPig>("Piglet");
    wildpig->DrawOnScreen();//Points to function that prints the action while also
                           // calling another virtual function that prints the name

    //makes pointer with overloaded argument using a name-Viserion
    shared_ptr<Dragon> dragon = make_shared<Dragon>("Viserion");
    dragon->DrawOnScreen();//Points to function that prints the action while also
                           // calling another virtual function that prints the name
    system("pause>nul");
    return 0;
}
```

```h
#ifndef CREATURE_H
#define CREATURE_H
#include <string>
#include<iostream>
using namespace std;

class Creature
{
protected:
    string CreatureName;//variable accessible to other classes
public:
    Creature(string cN);//overloaded constructor
    virtual void DoAction() = 0;//pure virtual function
    virtual void DrawOnScreen() = 0;//pure virtual function
};
Creature::Creature(string cN)//name is accessed within the class
{
    CreatureName = cN;//parameter is to the protected variable
}

#endif // !CREATURE_H
```

```
#ifndef PLAYER_H
#define PLAYER_H
#include "Creature.h"

class Player : public Creature//to inherit from Creature
{
public:
    Player(string name) : Creature(name)//overloaded constructor that
    {//uses the parameter from the base class Creature which sets the name to
      CreatureName

    }
    virtual void DoAction();//virtual function being used for abstraction
    virtual void DrawOnScreen();//second virtual function being used for
      abstraction
};

void Player::DoAction()//the action the pointer will be doing
{
    cout << "is attacking!!" << endl << endl;//action
}
void Player::DrawOnScreen()//printing the outputs
{
    cout << "Player " << CreatureName << " ";//uses protected variable from base
      class
    DoAction();//calls the above function for the action the object is doing
}

#endif // !PLAYER_H
```

```
#ifndef MONSTER_H
#define MONSTER_H
#include "Creature.h"

class Monster : public Creature//to inherit from Creature
{
public:
    Monster(string name) : Creature(name)//overloaded constructor that
    {//uses the parameter from the base class Creature which sets the name to
      CreatureName

    }
    virtual void DoAction();//function for testing abstraction
    virtual void DrawOnScreen();//function to test for abstract
};
void Monster::DoAction()
{
    cout << "is doing monster stuff!!" << endl << endl;//action object is doing
}
void Monster::DrawOnScreen()
{
    cout << "Monster " << CreatureName << " ";//protected variable from base class
    DoAction();//calls the action
}
#endif // !MONSTER_H
```

```
#ifndef WILDPIG_H
#define WILDPIG_H
#include "Monster.h"

class WildPig : public Monster//to inherit from Monster class
{
public:
    WildPig(string name) : Monster(name)//sets the name to Monster class
      constructor
    {//that sets it to the base class constructor

    }
    virtual void DoAction();//virtual to test abstraction from Monster to Creature
    virtual void DrawOnScreen();//virtual to test abstraction from Monster to
      Creature
};
void WildPig::DoAction()
{
    cout << "is running!!" << endl << endl;//action object doing
}
void WildPig::DrawOnScreen()
{
    cout << "WildPig " << CreatureName << " ";//protected variable used
    DoAction();//calls the function above
}
#endif // !WILDPIG_H
```

```cpp
#ifndef DRAGON_H
#define DRAGON_H
#include "Monster.h"

class Dragon : public Monster//to inherit from Monster that inherits from Creature
{
public:
    Dragon(string name) : Monster(name)//overloaded constructor that uses
      overloaded
    {//Monster constructor that uses Creature's overloaded constructor

    }
    virtual void DoAction();//test for abstraction function
    virtual void DrawOnScreen();//test for abstraction function
};
void Dragon::DoAction()
{
    cout << "is breathing fire!!" << endl << endl;//action the object is doing
}
void Dragon::DrawOnScreen()
{
    cout << "Dragon " << CreatureName << " ";
    DoAction();//calls the action to do
}
#endif // !PLAYER_H
```