**HIFZA FAYYAZ**

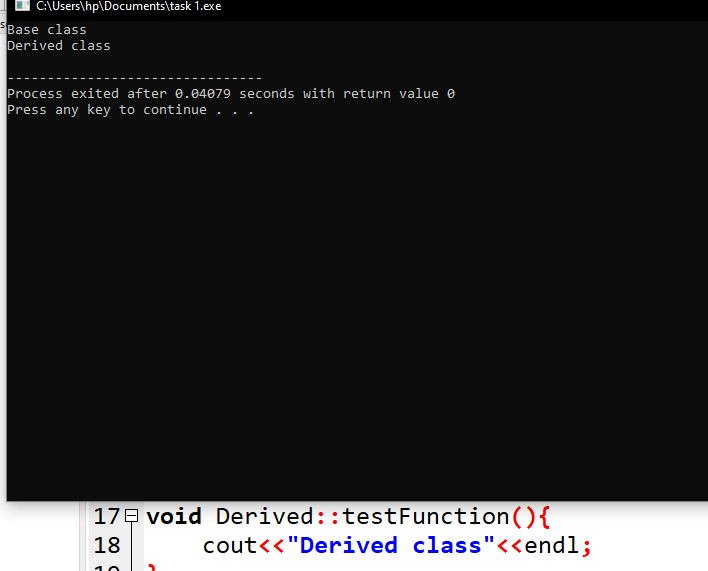
**46152**

**BSCS 2A FALL 2022**

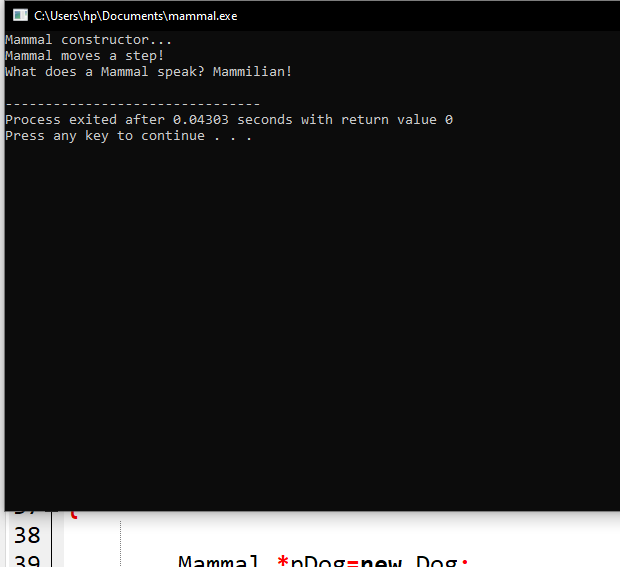
**OOP LAB TASK AFTER MIDS**

**POLYMORPHISM**

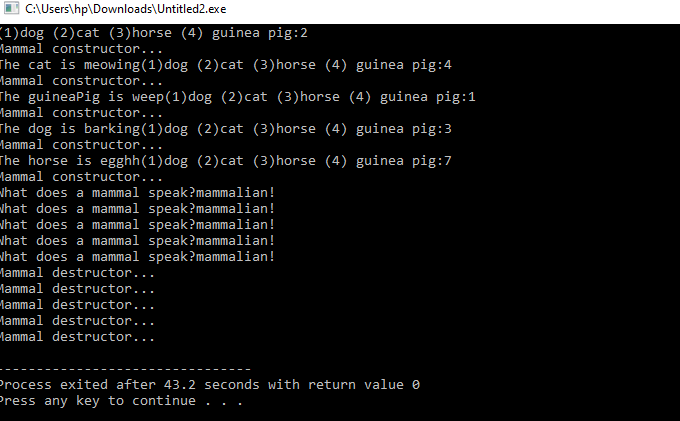
To gain a better understanding of polymorphic and virtual function start with the following simple examples. Notice we have not defined a virtual function yet.



You will first build two classes,Mammal and Dog. Dog will inherit from Mammal. Below is the mammal class code.Once you have the mammal class built, build a second class Dog that will inherit publicily from mammal.



Develop additional classes for Cat,Horse and GuineaPig overriding the move and speak methods.



**Some questions that you should start to understand:**

**Are inherited members and functions passed along to subsequent generations? If Dog derives from Mammal, and Mammal derives from Animal, does Dog inherit Animal's functions and data?**

Yes, in C++, when a class is derived from another class, it inherits all the non-private members (data members and member functions) of the base class

**Q. Can a derived class make a public base function private?**

With private inheritance makes the public and protected members of the base class private in the derived class.

**Q. Why not make all class functions virtual?**

Virtual functions cannot be in-lined because inlining have to happen at runtime. This have performance impacts when you expect you functions benefits from inlining.

**Q. If a function (SomeFunc()) is virtual in a base class and is also overloaded, so as to take either an integer or two integers, and the derived class overrides the form taking one integer, what is called when a pointer to a derived object calls the two-integer form?**

When a pointer to a derived object calls the two-integer form of the overloaded virtual function that is defined in the base class, the version of the function that is called depends on how the derived class overrides the function. If the derived class overrides only the one-integer form of the function, and does not provide an implementation for the two-integer form, then the two-integer form in the base class will be called when invoked through a pointer to the derived class object.

**Here are some more questions:**

**What is a v-table?**

The v is a virtual table. This is a lookup table of functions used to resolve function calls in a dynamic/late binding manner.

**2.What is a virtual destructor?**

Virtual Destructor in C++ is a member function that is used to free up the memory allocated used by the object of a child class or derived class when it is removed from the memory using the parent class pointer object.

**How do you show the declaration of a virtual constructor?**

The constructor cannot be virtual, because when a constructor of a class is executed there is no virtual table in the memory, means no virtual pointer defined yet. So, the constructor should always be non-virtual.

**How can you create a virtual copy constructor?**

No you can't, constructors can't be virtual.

**How do you invoke a base member function from a derived class in which you've overridden that function?**

We must use the scope resolution operator, “::” to access the overridden function. Another way to access the overridden function is by using the pointer of the base class to point to an object of the derived class and calling the function through the pointer.

**How do you invoke a base member function from a derived class in which you have not overridden that function?**

If you have not overridden the function in the derived class, calling the function using the base class name will invoke the base class implementation.

**If a base class declares a function to be virtual, and a derived class does not use the term virtual when overriding that class, is it still virtual when inherited by a third-generation class?**

Yes, if a base class declares a function to be virtual, and a derived class overrides that function without using the virtual keyword, the function is still considered virtual. This means that when the function is called on an object of the derived class or any further derived class, the most-derived implementation of the function will be called, regardless of whether the virtual keyword was used in the overriding function or not

**What is the protected keyword used for**?

The protected keyword specifies access to class members in the member-list up to the next access specifier ( public or private ) or the end of the class definition

**Some more exercises:**

**Show the declaration of a virtual function that takes an integer parameter and returns void.**

virtual void myFunction(int myParam) = 0;

**Show the declaration of a class Square, which derives from Rectangle, which in turn derives from Shape**.

class Shape {

public:

virtual double getArea() const = 0;

};

class Rectangle : public Shape {

public:

Rectangle(double width, double height) : width\_(width), height\_(height) {}

virtual double getArea() const override { return width\_ \* height\_; }

protected:

double width\_;

double height\_;

};

class Square : public Rectangle {

public:

Square(double side) : Rectangle(side, side) {}

};

**If, in Exercise 2, Shape takes no parameters, Rectangle takes two (length and width), but Square takes only one (length), show the constructor initialization for Square**

class Shape {

public:

Shape() {}

virtual double area() = 0;

};

class Rectangle : public Shape {

public:

Rectangle(double l, double w) : length(l), width(w) {}

virtual double area() { return length \* width; }

private:

double length;

double width;

};

class Square : public Rectangle {

public:

Square(double l) : Rectangle(l, l) {}

};

Write a virtual copy constructor for the class Square (in Exercise 3).

class Shape {

public:

virtual Shape\* clone() const = 0;

virtual double area() const = 0;

virtual ~Shape() {}

};

class Rectangle : public Shape {

public:

Rectangle(double l, double w) : length(l), width(w) {}

virtual double area() const { return length \* width; }

virtual Rectangle\* clone() const { return new Rectangle(\*this); }

private:

double length;

double width;

};

class Square : public Rectangle {

public:

Square(double l) : Rectangle(l, l) {}

virtual Square\* clone() const { return new Square(\*this); }

};

**BUG BUSTERS: What is wrong with this code snippet? void SomeFunction (Shape); Shape \* pRect = new Rectangle; SomeFunction(\*pRect);**

void SomeFunction(Shape& shape);

Shape\* pRect = new Rectangle;

SomeFunction(\*pRect);

**6. BUG BUSTERS: What is wrong with this code snippet? class Shape() { public: Shape(); virtual ~Shape(); virtual Shape(const Shape &); };**

class Shape {

public:

Shape();

virtual ~Shape();

Shape(const Shape &);

};