|  |
| --- |
| **Assignment # 6**  ***Session*: Spring 2022 *Total marks*: 100**  ***Name*** : ***\_nimra maqbool\_ Roll number* : \_bsce21012\_** |

***Submission:***

• *Email instructor or TA if there are any questions. You cannot look at others’ solutions or use others’ solutions, however, you can discuss it with each other. Plagiarism will be dealt with according to the course policy.*

*• Submission after due time will not be accepted.*

**There should be a Report explaining your code and highlighting results. Follow this naming convention for your report RollNumber\_Assignment#.pdf e.g BSCE21001\_Assignment6.pdf.**

**TASKS**

**Q1** What is inheritance? In your opinion, why and when to use inheritance?

|  |
| --- |
| **Inheritance:**  **It is a key feature of object-oriented programing in c++.it is used to make a derived class from the existing parent class. By inheritance we can access to the public and the protected members of the parent class.**  **Inheritance may be single or multiple.**  **Why is it used?**  **We use inheritance when there is repetition of the same functions in the code. Then we can increase the readability of the function and maintain it. It also gives us an advantage to reuse the code and increase the efficiency of the code.** |

**Q2** We have 2 types of Ducks that are given below:

Mallard duck

RedHead duck

Both ducks quack, fly and swim. But these ducks will be different in appearance.

*Hint: Implement a Duck class with common methods. Then inherit these methods to available types of ducks (Mallard & RedHead in this case)*

Your Duck class will look like this:

class Duck

{

public:

void quack();

void fly();

void swim();

void display(); // Abstract method

};

Since Ducks possess the same quack, fly and swim behavior, you need to implement these methods in the Duck class.

All ducks look different, so each duck subtype will implement its own display and will inherit other common methods from Duck class.

//write code: MallardDuck inherits from Duck class

{

public:

void display();

};

//write code: RedHeadDuck inherits from Duck class

{

public:

void display();

};

|  |
| --- |
| **Function.h:**  class duckA { public:  void quackA();  void flyA(); //declaring  void swimA();  virtual void displayA(){ //making an virtual function of display to use abstract method  } }; class MallardDuckA :public duckA{ //class inheriting duckA class public:  void displayA(){ //making display function  cout<<"I LOOK LIKE MALLARDUCK "<<endl;  }  }; class RedHeadDuckA:public MallardDuckA{ //class inheriting MallardDuckA class public:  void displayA(){ //making display function  cout<<"I LOOK LIKE REDHEADDUCK "<<endl;  }  };   * **In this task, I have made a class duck and in it I have made the public functions of quack, swim and fly and a virtual function of display as we wanted it to be different for every function.** * **Then I have inherited the duckA class in MallardDuckA class,and made the updated display function.** * **I have made another class named RedHeadDuckA inheriting MallardDuckA.**   **function.cpp:**  void duckA::quackA(){ //making a quack function and giving it path to duckA class  cout<<"I CAN TALK , QUACK QUACK "<<endl; } void duckA::flyA(){ //making a flyA function and giving it path to duckA class  cout<<"I HAVE WINGS , I CAN FLY "<<endl; } void duckA::swimA(){ //making a swimA function and giving it path to duckA class  cout<<"I CAN FLOAT "<<endl; }   * **In function.cpp I have defined the functions swim, quack and flyA.** * **And displayed different lines in it.**   **main.cpp:**  int opt; //declaring do {  cout << "WHICH TASK DO YOU WANT TO PERFORM?" << endl;  cout << "1.TASK 1." << endl; //asking for the choice from user  cout << "2.TASK 2." << endl;  cout << "3.TASK 3." << endl;  cout << "4.EXIT." << endl;  cin >> opt; //taking input  if (opt == 1) {   cout << "-------------------------------------------------------------------------" << endl;  duckA \*a; //making an pointer obj  MallardDuckA z; //making an object  a = &z; //taking it equal to the object  a->displayA(); //pointing to the function  z.quackA(); //calling  z.swimA();  z.flyA();  cout << "-------------------------------------------------------------------------" << endl;  RedHeadDuckA Z; //making an object  a = &Z; //taking it equal to the object  a->displayA(); //pointing to the function  Z.quackA();  Z.swimA(); //calling  Z.flyA();  cout << "------------------------------------------------------------------------" << endl;  }   * **In main.cpp I have made a menu and then called the functions.**   **output:**  **A picture containing graphical user interface  Description automatically generated**  **Uml\_diagram**:  Diagram  Description automatically generated with medium confidence |

**Q3** Add the following ducks to the same code of Q2:

Rubber duck

Decoy duck

|  |
| --- |
| **Function.h:**  class duckA { public:  void quackA();  void flyA(); //declaring  void swimA();  virtual void displayA(){ //making an virtual function of display to use abstract method  } }; class MallardDuckA :public duckA{ //class inheriting duckA class public:  void displayA(){ //making display function  cout<<"I LOOK LIKE MALLARDUCK "<<endl;  }  }; class RedHeadDuckA:public MallardDuckA{ //class inheriting MallardDuckA class public:  void displayA(){ //making display function  cout<<"I LOOK LIKE REDHEADDUCK "<<endl;  }  }; class RubberDuckA:public RedHeadDuckA{ //class inheriting RedHeadDuckA public:  void displayA(){ //making display function  cout<<"I LOOK LIKE RUBBERDUCK "<<endl;  }  };  class DecoyDuckA:public RubberDuckA{ //class inheriting RubberDuckA class public:  void displayA(){ //making display function  cout<<"I LOOK LIKE DECOYDUCK "<<endl;  }  };   * **In function.h I have made 2 new classes named RubberDuckA and DecoyDuckA.** * **The rubberDuckA class is inheriting RedHeadDuckA class and have the display function in it.** * **The DecoyDuckA is inheriting the RubberDuckA class and have the display function in it.**   **Function.cpp:**  void duckA::quackA(){ //making a quack function and giving it path to duckA class  cout<<”I CAN TALK , QUACK QUACK “<<endl; } void duckA::flyA(){ //making a flyA function and giving it path to duckA class  cout<<”I HAVE WINGS , I CAN FLY “<<endl; } void duckA::swimA(){ //making a swimA function and giving it path to duckA class  cout<<”I CAN FLOAT “<<endl; }   * **The function.cpp is same as the uper task.**   **main.cpp:**  if(opt==2){  cout << "-------------------------------------------------------------------------" << endl;  duckA \*a; //making an pointer obj  MallardDuckA z; //making an object  a = &z; //taking it equal to the object  a->displayA(); //pointing to the function  z.quackA(); //calling  z.swimA();  z.flyA();  cout << "-------------------------------------------------------------------------" << endl;  RedHeadDuckA Z; //making an object  a = &Z; //taking it equal to the object  a->displayA(); //pointing to the function  Z.quackA();  Z.swimA(); //calling  Z.flyA();  cout << "------------------------------------------------------------------------" << endl;  RubberDuckA t; //making an object  a = &t; //taking it equal to the object  a->displayA(); //pointing to the function  t.quackA();  t.swimA();  t.flyA(); //calling  cout << "------------------------------------------------------------------------" << endl;  DecoyDuckA T; //making an object  a = &T; //taking it equal to the object  a->displayA(); //pointing to the function  T.quackA();  T.swimA();  T.flyA(); //calling  cout << "-----------------------------------------------------------------------" << endl; }   * **In main.cpp I have made a menu and then called the functions.**   **output:**  **A picture containing text  Description automatically generated**  **Uml\_diagram:**  **Diagram  Description automatically generated** |

**Q4** We need to add fly behaviour for ducks.

But we know that all the types of ducks cannot fly e.g Rubber duck cannot fly.

If we add the fly behaviour to the parent class, what will be the problem in using inheritance now?

What solution do you suggest for solving this problem? (Hint: Find out what is multiple inheritance, when and how do we use it)

|  |
| --- |
| **Function.h:**  **(method -1)**  class duck { //creating another class of name duck public:  void quack();  virtual void fly()=0; //pure virtual function of fly  void swim(); //declaring  virtual void display(){ //virtual function of duck  } }; class MallardDuck :public duck{ //class inheriting duck class public:  void display(){  cout<<"I LOOK LIKE MALLARDUCK "<<endl; //making display function  }  void fly(){ //making fly function as we make its pure virtual function  cout<<"I HAVE WINGS , I CAN FLY "<<endl;  } }; class RedHeadDuck:public MallardDuck{ //class inheriting MallardDuck public:  void display(){  cout<<"I LOOK LIKE REDHEADDUCK "<<endl; //making display function  }  void fly(){ //making fly function as we make its pure virtual function  cout<<"I HAVE WINGS , I CAN FLY "<<endl;  } }; class RubberDuck:public RedHeadDuck{ //class inheriting RedHeadDuck public:  void display(){  cout<<"I LOOK LIKE RUBBERDUCK "<<endl; //making display function  }  void fly(){ //making fly function as we make its pure virtual function  cout<<"I HAVE WINGS , BUT I CAN'T FLY "<<endl;  } };  class DecoyDuck:public RubberDuck{ //class inheriting RubberDuck public:  void display(){ //making display function  cout<<"I LOOK LIKE DECOYDUCK "<<endl;  }  void fly(){ //making fly function as we make its pure virtual function  cout<<"I HAVE WINGS , I CAN'T FLY "<<endl;  } }; //cout<<" -----------------------------------------------------------------------------------"<<endl;   * **In the function.h I have made the duck class having quack swim, virtual display and pure virtual fly function.** * **Then I have made the MallarDuck inherited by the duck class and having 2 functions of display and fly.** * **Then I have made the RedHeadDuck inherited by the MallardDuck class and having the display and fly function in it .** * **Then I have made the RubberDuck class inherited by RedHeadDuck class having fly and display function.** * **Then I have made the DecoyDuck class inherited by the RubberDuck class.**   **(method -2)**  class flyDuck{ //making a class of flyDuck public:  void flyB(){ //a public function of flyB  cout<<”I HAVE WINGS,I CAN FLY”<<endl;  } }; class duck1 { //making a class of name duck! public:  void quack1(){ //making quack function in duck1  cout<<”I CAN TALK , QUACK QUACK “<<endl;  }  void swim1(){ //making a swim function in duck1  cout<<”I CAN FLOAT “<<endl;  }  virtual void display1(){  } //making virtual function of display };  class MallardDuck1 :public duck1 ,public flyDuck{ //inheriting multiple classes(duck1 and flyDuck) public:  void display1(){ //making function of display  cout<<”I LOOK LIKE MALLARDUCK “<<endl;  }  };  class RedHeadDuck1:public MallardDuck1{ //inheriting MallardDuck1 public:  void display1(){ //making function of display  cout<<”I LOOK LIKE REDHEADDUCK “<<endl;  } }; class RubberDuck1:public RedHeadDuck1 { //inheriting RedHeadDuck public:  void display1(){ //making function of display  cout<<”I LOOK LIKE RUBBERDUCK “<<endl;  }  void flyB(){  cout<<”I HAVE WINGS , BUT I CAN’T FLY”<<endl; //an verridden function of flyB  }  }; class DecoyDuck1:public RubberDuck1 { //inheriting RubberDuck1 public:  void display1(){ //making function of display  cout<<”I LOOK LIKE DECOYDUCK “<<endl;  }  void flyB(){  cout<<”I HAVE WINGS , BUT I CAN’T FLY”<<endl; //a verridden function of flyB  } };   * **I have made 2 classes named duck1 and flyDuck.** * **In flyDuck I have made a function of fly in public.** * **Then in duck1 I have made the functions quack, swim, and virtual function display.** * **Then I have made the MallardDuck1 class inherited by the flyDuck and the duck1, having display function in it.** * **Then I have made RedHeadDuck class inherited by the MallardDuck class, having display function in it.** * **Then I have made the rubberDuck class inherited by the RedHead class then I have made display function and the overriden fly function display I can’t fly.** * **Then I have made an decoyDuck class inherited by the RubberDuckClass and having the display and overriden function of fly displaying I can’t fly.**   **function.cpp(method-1)**  void duck::quack(){ //making a quack function and giving it path to duck class  cout<<"I CAN TALK , QUACK QUACK "<<endl; } void duck::swim(){ //making a swim function and giving it path to duck class  cout<<"I CAN FLOAT "<<endl; }   * **I have made 2 functions of quack and swim of the class duck.**   **main.cpp:**  if (opt == 3) {  int opt1; //declaring  cout << "I HAVE PERFORMED TASK 2 FROM 2 METHODS " << endl;  cout << "FROM WHICH METHOD DO YOU WANT TO PERFORM IT?" << endl;  cout << "1.METHOD 1" << endl; //asking for the choice from user  cout << "2.METHOD 2" << endl;  cout << "3.EXIT" << endl;  cin >> opt1; //taking input  if (opt1 == 1) {  cout << "-------------------------------------------------------------------------" << endl;  duck1 \*f; //making an pointer obj  MallardDuck1 M; //making an object  f = &M; //taking it equal to the object  f->display1(); //pointing to the function  M.swim1();  M.quack1(); //calling  M.flyB();  cout << "--------------------------------------------------------------------------" << endl;  RedHeadDuck1 R; //making an object  f = &R; //taking it equal to the object  f->display1(); //pointing to the function  R.swim1(); //calling  R.quack1();  R.flyB();  cout << "--------------------------------------------------------------------------" << endl;  RubberDuck1 E; //making an object  f = &E; //taking it equal to the object  f->display1(); //pointing to the function  E.swim1();  E.quack1();  E.flyB(); //calling  cout << "--------------------------------------------------------------------------" << endl;  DecoyDuck1 g; //making an object  f = &g; //taking it equal to the object  f->display1(); //pointing to the function  g.swim1(); //calling  g.quack1();  g.flyB();  cout << "--------------------------------------------------------------------------" << endl;   }  if (opt1 == 2) {  cout << "-----------------------------------------------------------------------------" << endl;  duck \*c; //making an pointer obj  MallardDuck m; //making an obj  c = &m; //taking it equal to the object  c->display(); //pointing to the function  m.quack(); //calling  m.swim();  m.fly();  cout << "------------------------------------------------------------------------" << endl;  RedHeadDuck d; //making an object  c = &d; //taking it equal to the object  c->display(); //pointing to the function  d.swim();  d.quack(); //calling  d.fly();  cout << "------------------------------------------------------------------------" << endl;  RubberDuck r; //making an object  c = &r; //taking it equal to the object  c->display(); //pointing to the function  r.swim();  r.quack(); //calling  r.fly();  cout << "------------------------------------------------------------------------" << endl;  DecoyDuck D; //making an object  c = &D; //taking it equal to the object  c->display(); //pointing to the function  D.swim();  D.quack(); //calling  D.fly();  cout << "-----------------------------------------------------------------------------" << endl;  }  if (opt1 == 3) {  cout << "YOU CHOOSE TO EXIT .." << endl;  exit(5);  } }   * **In main.cpp I have made a menu and then called the functions.**   **output:(method 1)**  **Text  Description automatically generated**  **Output(method -2)**  **Text  Description automatically generated**  **Uml\_diagram(method-1)**  **Diagram  Description automatically generated with low confidence**  **Uml\_diagram:**  **Diagram  Description automatically generated** |