**\_\_\_\_\_**

**complete the program. You may need to make some changes in the code:**

class Dog {

protected:

int age;

\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

};

int main(){

//declare a dog called woofy

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// set Woofy’s age to 5

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

return 0;

}

**Find ERROREs in this programme and fix them**

#include <iostream>

#include <string>

using namespace std;

class Celcius{

private:

float c;

void readCelcius(){

cout<<"Enter temperature in Celcius: ";

cin>>c;

}

public:

//constructor

Celcius(){ c= 0};

Celcius( float c1) { c= c1;}

// destructor

float ¬ Celcius(int x);

float toFarenheit(){return((9.0/5.0)\*c+32);}

};

int main(){

Celcius c1;

c1.Celcius();

Celcius c2();

Celcius c2(3.4);

c1.readCelcius();

cout<<c1.toFarenheit()<<" F"<<endl;

f1.readFarenheit();

cout<<f1.toCelcius()<<" C";

return 0;

}

trace the following program ,show the output

#include <iostream>

#include <string>

using namespace std ;

//==================================================

class CreateAndDestroy {

public:

CreateAndDestroy(int objectNumber, string message )// constructor

~CreateAndDestroy(); // destructor

private:

int objectID;

string message;

}; // end class CreateAndDestroy

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// constructor

CreateAndDestroy::CreateAndDestroy( int objectNumber, string message )

{

objectID = objectNumber;

message = message;

cout << "Object " << objectID << " constructor runs "

<< message << endl;

} // end CreateAndDestroy constructor

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// destructor

CreateAndDestroy::~CreateAndDestroy()

{

cout << "Object " << objectID << " destructor runs "

<< message << endl;

} // end ~CreateAndDestroy destructor

//==================================================

void create( void ); // prototype

// global object

CreateAndDestroy first( 1, "(global before main)" );

int main()

{

cout << "\nMAIN FUNCTION: EXECUTION BEGINS" << endl;

CreateAndDestroy second( 2, "(local in main)" );

create(); // call function to create objects

cout << "\nMAIN FUNCTION: EXECUTION RESUMES" << endl;

CreateAndDestroy third( 3, "(local in main)" );

cout << "\nMAIN FUNCTION: EXECUTION ENDS" << endl;

return 0;

} // end main

//===================================================

// function to create objects

void create( void )

{

cout << "\nCREATE FUNCTION: EXECUTION BEGINS" << endl;

CreateAndDestroy fourth( 4, "(local in create Function )" );

CreateAndDestroy fifth(

5, "(local in create Function)" );

cout << "\nCREATE FUNCTION: EXECUTION ENDS" << endl;

} // end function create

Write the first line of the declaration for a class CAT that is publicly derived from a Class ANIMAL.

Answer the following :

* Adding a derived class to a base class requires fundamental changes to the base class. ( True / False )
* If no constructors are specified for a derived class, objects of the derived class will use the constructors in the base class. (True / False )
* A class D can be derived from a class C, which is derived from a class B, which is derived from a class A. (True / False )
* To be accessed from a member function of the derived class, data or functions in the base Class must be public or \_\_\_\_\_\_\_\_\_.

If a **base** class and a **derived** class each include a member function with the **same name**,

Which member function will be called by an object of the derived class, assuming the

Scope-resolution operator is not used?

Assume that there is a class **Derv** that is derived from a base class **Base**. Write the

Declaratory for a derived-class **constructor** that takes one argument and passes this argument

Along to the constructor in the base class.

Assume a class **Derv** that is **privately** derived from class **Base**. An object of class Derv

located in main() can access

1. public members of Derv.
2. protected members of Derv.
3. private members of Derv.
4. public members of Base.
5. protected members of Base.
6. private members of Base.

**Trace** the program then Define the differences between the function overloading and overwritten? **(find the errors )**

#include <iostream>

using namespace std;

////////////////////////////////////////////////////////////////

class A //base class

{

private:

int privdataA;

protected:

int protdataA;

public:

int pubdataA;

void funct(int x, int y, int z)

{

int a;

privdataA= x;

protdataA = y;

pubdataA= z;

}

void print(){ cout<< "base Class"; }

};

////////////////////////////////////////////////////////////////

class B : public A //publicly-derived class

{ int privatedataB;

public:

void funct(int x, int y, int z, int i)

{

funct(x,y,z);

privatedataB= i;

}

/////////////////////////////

void print(){

print();

A::print();

cout<< "deriverd Class ";

}

};

////////////////////////////////////////////////////////////////

int main()

{

int a;

B objB;

objB.funct(1,3,6);

objB.funct(7,6,2,1);

objB.print();

return 0;

}

Trace the following program and Find Errors and correct them:

#include <iostream>

using namespace std;

class Base

{ int m;

public:

Base(int mValue){ m = mValue; cout<<"Constructing base"; }

};

class Derived: public Base

{double n;

public:

Derived(double nValue)

{ n =nValue; cout<<"Constructing derived"; }

};

int main(){

Base b1;

Base b2(3.0);

Derived d1

Derived d2(4.0);

return 0;

}

Trace the following program and write the output:

class Person {

// Data members of person

public:

Person(int x) { cout << "Person::Person(int ) called" << endl; }

};

class Faculty : public Person {

// data members of Faculty

public:

Faculty(int x):Person(x) {

cout<<"Faculty::Faculty(int ) called"<< endl;

}

};

class Student : public Person {

// data members of Student

public:

Student(int x):Person(x) {

cout<<"Student::Student(int ) called"<< endl;

}

};

class TA : public Faculty, public Student {

public:

TA(int x):Student(x), Faculty(x) {

cout<<"TA::TA(int ) called"<< endl;

}

};

void main() {

TA ta1(30);

}