**Tutorial #4**

1. Check the access privilege (define whether is OK or not accessible :

#include <iostream>

using namespace std;

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

class A //base class

{

 private:

 int privdataA;

 protected:

 int protdataA;

 public:

 int pubdataA;

};

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

class B : public A //publicly-derived class

{

public:

void funct()

{int a ,b ,c;

a = privdataA;

b = protdataA;

c = pubdataA;}

};

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

class C : private A //privately-derived class

{public:

void funct()

{

int a , b ,c;

a = privdataA;

b = protdataA;

c = pubdataA;}

};

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

int main()

{

int a;

B objB;

a = objB.privdataA;

a = objB.protdataA;

a = objB.pubdataA;

C objC;

a = objC.privdataA;

a = objC.protdataA;

a = objC.pubdataA;

return 0;

1. **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;

}

1. Trace the following program and Find Errors and correct them:

#include <iostream>

using namespace std;

class Base

{ int m\_nValue;

public:

 Base(int nValue)

 { m\_nValue= nValue; cout<<"Constructing base"; }

};

class Derived: public Base

{double m\_dValue;

public:

 Derived(double dValue) // :

 { m\_dValue =dValue; cout<<"Constructing derived"; }

};

int main(){

 Base b1;

 Base b2(3.0);

 Derived d1;

 Derived d2(4.0);

return 0;

}

1. Trace the following program and write the output:

#include<iostream>

using namespace std;

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;

 }

};

int main() {

 TA ta1(30);

}