**PSG College of Technology**

**Department of Applied Mathematics and Computational sciences**

**Object Oriented Programming with C++**

**Worksheet-4 (Inheritance / Operator Overloading)**

**Inheritance**

Each of the class declarations and/or member function definitions below has errors. Find

as many as you can.

1. class Car, public Vehicle

{

public:

Car();

~Car();

protected:

int passengers;

}

2. class Truck, public : Vehicle, protected

{

private:

double cargoWeight;

public:

Truck();

~Truck();

};

3. class SnowMobile : Vehicle

{

protected:

**962** Chapter 15 Inheritance, Polymorphism, and Virtual Functions

int horsePower;

double weight;

public:

SnowMobile(int h, double w), Vehicle(h)

{ horsePower = h; }

~SnowMobile();

};

4. class Table : public Furniture

{

protected:

int numSeats;

public:

Table(int n) : Furniture(numSeats)

{ numSeats = n; }

~Table();

};

5. class Tank : public Cylinder

{

private:

int fuelType;

double gallons;

public:

Tank();

~Tank();

void setContents(double);

void setContents(double);

};

6. class Three : public Two : public One

{

protected:

int x;

public:

Three(int a, int b, int c), Two(b), Three(c)

{ x = a; }

~Three();

};

**Analyse the code and write the output.**

1. class base{

public:

int bval;

base(){ bval=0;}

};

class deri:public base{

public:

int dval;

deri(){ dval=1;}

};

void SomeFunc(base \*arr,int size){

for(int i=0; i<size; i++,arr++)

cout<<arr->bval;

cout<<endl;

}

int main(){

base BaseArr[5];

SomeFunc(BaseArr,5);

deri DeriArr[5];

SomeFunc(DeriArr,5);

}

2. class base{

public:

void baseFun(){ cout<<"from base"<<endl;}

};

class deri:public base{

public:

void baseFun(){ cout<<"from derived"<<endl;}

};

void SomeFunc(base \*baseObj){

baseObj->baseFun();

}

int main(){

base baseObject;

SomeFunc(&baseObject);

deri deriObject;

SomeFunc(&deriObject);

}

1. class base{

public :

out() {

cout<<"base ";

}

};

class deri : public base{

public : out(){

cout<<"deri ";

}

};

void main(){

deri dp[3];

base \*bp = (base\*)dp;

for (int i=0; i<3;i++)

(bp++)->out();

}

1. class base{

public:

virtual void baseFun(){ cout<<"from base"<<endl;}

};

class deri:public base{

public:

void baseFun(){ cout<<"from derived"<<endl;}

};

void SomeFunc(base \*baseObj){

baseObj->baseFun();

}

int main(){

base baseObject;

SomeFunc(&baseObject);

deri deriObject;

SomeFunc(&deriObject);

}

1. class base

{

public:

int n;

virtual void foo(){n=1;}

void print(){cout <<n<<endl;}

};

class derived: base

{

public:

void foo(){n=2;}

void print(){cout <<n<<endl;}

};

void main()

{

derived y;

base \*bp =dynamic\_cast<base \*>(&y);

bp->foo();

bp->print();

}

1. class base {/\*….\*/};

class derived :public base{/\*….\*/};

void foo()

{ try

{

throw derived();

}

catch (base b)

{

cout<<”Received exception, but can’t handle\n”;

throw;

}

};

void main()

{

try

{

foo();

}

catch (derived d)

{

cout<<”In derived handler”;

}

catch (base b)

{

cout <<“In Base handler”;

}

}

**Operator overloading**

Each of the class declarations and/or member function definitions below has errors. Find

as many as you can.

1. class Box

{

private:

double width;

double length;

double height;

public:

Box(double w, l, h)

{ width = w; length = l; height = h; }

// Overloaded prefix ++ operator

void operator++()

{ ++width; ++length;}

// Overloaded postfix ++ operator

void operator++()

{ width++; length++;}

*... Other member functions follow ...*

};

2. class Yard

{

private:

float length;

public:

yard(float l)

{ length = l; }

// float conversion function

void operator float()

{ return length; }

*... Other member functions follow ...*

};

**Analyse the code and write the output.**

1. class opOverload{

public:

bool operator==(opOverload temp);

};

bool opOverload::operator==(opOverload temp){

if(\*this == temp ){

cout<<"The both are same objects\n";

return true;

}

cout<<"The both are different\n";

return false;

}

void main(){

opOverload a1, a2;

a1== a2;

}

1. class fig2d{

int dim1, dim2;

public:

fig2d() { dim1=5; dim2=6;}

virtual void operator<<(ostream & rhs);

};

void fig2d::operator<<(ostream &rhs){

rhs <<this->dim1<<" "<<this->dim2<<" ";

}

class fig3d : public fig2d{

int dim3;

public:

fig3d() { dim3=7;}

virtual void operator<<(ostream &rhs);

};

void fig3d::operator<<(ostream &rhs){

fig2d::operator <<(rhs);

rhs<<this->dim3;

}

void main(){

fig2d obj1;

fig3d obj2;

obj1 << cout;

obj2 << cout;

}