**Lab Assignment-5**

**UTA018: Object Oriented Programming**

**Topics covered: Constructors in Inheritance, abstract class and virtual base class**

**Q1. Implement a C++ program to define three classes Alpha, Beta and Gamma, each class**

**having private data members. Gamma is a class derived from Alpha and Beta (by applying**

**multiple inheritance). Use constructors and destructors to read and display data.**

#include<iostream>

using namespace std;

class alpha{

protected:

int a;

public:

alpha(int x){

a=x;

cout<<"alpha constructor executed"<<endl;

}

~alpha(){

cout<<"Alpha Destructor executed"<<endl;

}

};

class beta{

protected:

int b;

public:

beta(int x){

b=x;

cout<<"Beta constructor executed"<<endl;

}

~beta(){

cout<<"Beta Destructor executed"<<endl;

}

};

class gamma: public alpha, public beta{

int g;

public:

//Not needed if base class constructor default. Necessary if parameterized.

gamma(int x):alpha(1),beta(2){

g=x;

cout<<"gamma constructor executed"<<endl;

cout<<"a: "<<a<<" b: "<<b<<endl;

}

~gamma(){

cout<<"Gamma Destructor executed"<<endl;

}

};

int main(){

gamma obj(3);

return 0;

}

**Q2. Write a program to define class X, Y and Z. Each class contains one character array as a data**

**member. Using multiple inheritance, concatenate strings of class X and Y and store it in class Z.**

**Using constructor and destructors, show all the three strings.**

#include<iostream>

#include<string.h>

using namespace std;

class x{

protected:

char x\_str[10];

public:

x(char x\_new\_str[10]){

strcpy(x\_str,x\_new\_str);

cout<<"x constructor executed"<<endl;

}

~x(){

cout<<"x Destructor executed"<<endl;

}

};

class y{

protected:

char y\_str[10];

public:

y(char y\_new\_str[10]){

strcpy(y\_str,y\_new\_str);

cout<<"y constructor executed"<<endl;

}

~y(){

cout<<"y Destructor executed"<<endl;

}

};

class z: public x, public y{

char z\_str[20];

public:

z():x("hello"),y("world"){

strcpy(z\_str,x\_str);

strcat(z\_str,y\_str);

cout<<"z constructor executed"<<endl;

cout<<"z\_str: "<<z\_str<<endl;

}

~z(){

cout<<"z Destructor executed"<<endl;

}

};

int main(){

z obj;

return 0;

}

**Q3. Write a C++ program to implement the diamond problem (hybrid inheritance). State the**

necessary assumptions using comments. – (HINT: use virtual base class)

#include<iostream>

using namespace std;

class A{

public:

int a;

A(){

a=10;

}

};

class B: virtual public A{

};

class C: virtual public A{

};

class D: public B, public C{

};

int main(){

D obj;

cout<<"a is: "<<obj.a;

return 0;

}

**Q4. Write a C++ program creating an abstract class Student. Create three derived classes**

**Science, Art and Commerce from the base class. Create the objects of the derived classes and**

**process them and access them using array of pointer of type Student.**

/\*

#include<iostream>

using namespace std;

class student{

//abstract class

public:

virtual void display()=0;

};

class science: public student{

//need to define or else will become abstract

void display(){

cout<<"Science"<<endl;

}

};

class arts: public student{

//diff classes can give own definition to display

void display(){

cout<<"arts"<<endl;

}

};

class commerce: public student{

void display(){

cout<<"Commerce"<<endl;

}

};

int main(){

student \*ptr;

ptr= new science;

ptr->display();//science display func called. ptr of type base but calling contents of derived

ptr= new arts;

ptr->display();

ptr= new commerce;

ptr->display();

delete ptr;

return 0;

}

\*/

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include<iostream>

using namespace std;

class student{

int x;

public:

virtual void show()=0;

};

class science: public student{

public:

void show(){

cout<<"science"<<endl;

}

};

class arts: public student{

public:

void show(){

cout<<"arts"<<endl;

}

};

class commerce: public student{

public:

void show(){

cout<<"commerce"<<endl;

}

};

int main(){

student \*ptr[3];

science obj1;

arts obj2;

commerce obj3;

ptr[0]=&obj1;

ptr[1]=&obj2;

ptr[2]=&obj3;

ptr[0]->show();

ptr[1]->show();

ptr[2]->show();

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*