**Lab-5 Virtual Function, Polymorphism, and miscellaneous C++ Features**

**1. Write a program to solve the concept of virtual function.**

#include<iostream>

using namespace std;

class base

{

public:

void display ()

{

cout<<”\n display base:”;

}

virtual void show ()

{

count <<”\n show base”;

}

};

class derived: public base

{

public:

void display ()

{

cout<<”\n display derived:”;

}

void show ()

{

cout<<”\n show derived”;

}

};

int main()

{

Base B;

Derived D;

Base \*bptr;

cout<<”\n bptr point to base”;

bptr = &B;

bptr → dsplay(); //calls base version

bptr → show(); //calls base version

cout<<”\n bptr points to derived”;

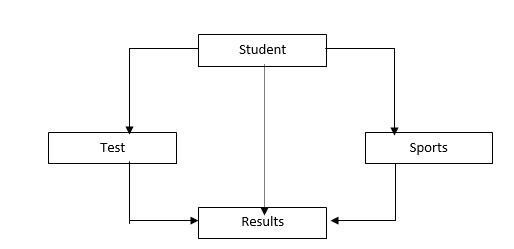
bptr = &D

bptr → display(); // calls base version

bptr → calls derived version

}

**2. Solve the following form of inheritance. [ use virtual base class]**



#include<iostream>

using namespace std;

class student

{

protected:

int roll\_number;

public:

void get\_number(int a)

{

roll\_number = a;

}

void put\_number(void)

{

cout<<"Roll No:"<<roll\_number<<"\n";

}

};

class test: virtual public student

{

protected:

float part1, part2;

public:

void get\_marks (float x, float y)

{

part1 = x; part2 = y;

}

void put\_marks(void)

{

cout <<"Marks obtained:"<<"\n"

<<"part1 = "<<part1<<"\n"

<<"part2 ="<<part2<<"\n";

}

};

class sports: public virtual student

{

protected:

float score;

public:

void get\_score(float s)

{

score = s;

}

void put\_score(void)

{

cout<<"sports wt:" <<score<<"\n";

}

};

class result: public test, public sports

{

float total;

public:

void display (void);

};

void result:: display (void)

{

total = part1 + part2 + score;

put\_number ();

put\_marks ();

put\_score ();

cout <<"\n Total score:"<< total<<"\n";

}

int main ()

{

result student1;

student1.get\_number (678);

student1.get\_marks(30.5, 25.5);

student1.get\_score(7.0);

student1.display();

}

**3.Solve Q. No. 2 by defining at least one constructor in each class**

**4. Write a program to compare the age of three persons and display the information of elder one. Use “this” pointer to return the information of the person.**

#include<iostream>

#include<string.h>

using namespace std;

class person

{

char name[20];

float age;

public:

person(char s[],float a)

{

strcpy(name,s);

age=a;

}

person greater(person &x)

{

if(x.age>=age)

return x;

else

return \*this;

}

void display(void)

{

cout<<"Name:"<<name<<"\n"<<"Age:"<<age<<"\n";

}

};

int main()

{ char name1[10]="John";

char name2[10]="Jack";

char name3[10]= "Jim";

person P1(name1,37.50),P2(name2,29.0),P3(name3,40.25);

person P=P1.greater(P3);

cout<<"Elder person is: \n";

P.display();

P=P1.greater(P2);

cout<<"\nElder person is: \n";

P.display();

return 0;

}

**5.Write a program to show the concept of pure virtual function.**

#include<iostream>

using namespace std;

class A

{

public:

virtual void show () = 0; // pure virtual function

};

class B: public A

{

public:

void show( ) // pure virtual function is overriden here

{

cout <<"show method is implemented here";

}

};

int main ()

{

A \* ptr;

// ptr = new A; Cannot create instance of abstract class A

ptr = new B;

ptr->show( );

return 0;

}

**6.Write a program to read information of three employees (name, address and salary). Make three separate classes for each employee. Now, calculate the total salary of three employee and display all information. Use the concept of friend function in your program.**

#include<iostream>

using namespace std;

class E2;

class E3;

class E1

{ char name[10];

float salary;

public:

void set()

{

cout<<"\n Enter first Employee name and salary";

cin>>name>>salary;

}

friend void process(E1,E2,E3);

};

class E2

{

char name[10];

float salary;

public:

void set()

{

cout<<"\n Enter second Employee name and salary";

cin>>name>>salary;

}

friend void process(E1,E2,E3);

};

class E3

{

char name[10];

float salary;

public:

void set()

{

cout<<"\n Enter third Employee name and salary";

cin>>name>>salary;

}

friend void process(E1,E2,E3);

};

void process(E1 x, E2 y, E3 z)

{

cout<<"\n\nFirst Employee name= "<< x.name;

cout<<"\nFirst Employee salary= "<<x.salary;

cout<<"\n\nSecond Employee name= "<< y.name;

cout<<"\nSecond Employee salary= "<<y.salary;

cout<<"\n\nThirdEmployee name= "<< z.name;

cout<<"\nThird Employee salary= "<<z.salary;

float total = x.salary + y.salary + z.salary;

cout<<"\n\n Their total salary= "<<total;

}

int main()

{

E1 A;

E2 B;

E3 C;

A.set();

B.set();

C.set();

process(A,B,C);

return 0;

}

**7.Write a program to show the concept of static function and static member variable.**

#include<iostream>

using namespace std;

class test

{

int code;

static int count; // static member variable

public:

void setcode(void)

{

code=++count;

}

void showcode(void)

{

cout<<"object member : "<<code<<endl;

}

static void showcount(void)

{ cout<<"count="<<count<<endl;

//cout<<code; //this can not be done here because static function will access only static variables

}

};

int test:: count;

int main()

{

test t1,t2;

t1.setcode( );

t2.setcode( );

test :: showcount ( );

test t3;

t3.setcode( );

test:: showcount( );//accessing static member function

t1.showcode( );

t2.showcode( );

t3.showcode( );

//test t4;

//t4.showcount(); //it can also be done

return(0);

}

**8.Write a program of virtual destructor.**

#include <iostream>

using namespace std;

class base {

public:

base()

{ cout << "Constructing base\n"; }

virtual ~base()

{ cout << "Destructing base\n"; }

};

class derived : public base {

public:

derived()

{ cout << "Constructing derived\n"; }

~derived()

{ cout << "Destructing derived\n"; }

};

int main()

{

derived d;

return 0;

}