**Lab – 3**

Constructor, destructor, dynamic memory management, friend function

1. **Create a class named *'Rectangle'* with two data members- *length* and *breadth* and a function to calculate the area which is 'length\*breadth'. The class has *three constructors* which are:  
   (a) having no parameter - values of both length and breadth are assigned zero.**

**(b) having two numbers as parameters - the two numbers are assigned as length and**

**breadth respectively.  
(c) having one number as parameter - both length and breadth are assigned that number.  
Now, create objects of the 'Rectangle' class having none, one and two parameters and print their areas.**

#include<iostream>

using namespace std;

class rect{

int length, breadth;

public:

rect(void){

}

rect(int a, int b){

length=a;

breadth=b;

}

rect(int a){

length=a;

breadth=a;

}

void area(void){

cout<<"Area is: "<<length\*breadth<<endl;

}

};

int main(){

rect r1,r2(2,3),r3(4);

r1.area();

r2.area();

r3.area();

return 0;

}

1. **Redefine the above program by creating an array of objects of the class Rectangle and calculate area for each object calling different constructors. Also implement constructors with default arguments and destructor in this program.**

#include<iostream>

using namespace std;

class rect{

int length, breadth;

public:

rect(void){

}

rect(int a, int b){

length=a;

breadth=b;

}

rect(int a){

length=a;

breadth=a;

}

void area(void){

cout<<"Area is: "<<length\*breadth<<endl;

}

~rect(){

cout<<"Destructor called"<<endl;

}

};

int main(){

rect r[3]={rect(),rect(2,3),rect(4)};

rect().area();

rect(2,3).area();

rect(4).area();

return 0;

}

1. **Verify the following about *destructor* by writing the program:**
2. **Name should begin with tilde sign(~) and must match class name.**
3. **There cannot be more than one destructor in a class.**
4. **Destructors do not allow any parameter.**
5. **They do not have any return type, just like constructors.**

**When you do not specify any destructor in a class, compiler generates a default destructor and inserts it into your code.**

#include<iostream>

using namespace std;

class test{

public:

test(){

cout<<"Constructor"<<endl;

}

~test(){

cout<<"Destructor"<<endl;

}

};

int main(){

cout<<"Enter main: "<<endl;

test obj1;

{

cout<<"Enter block"<<endl;

test obj2;

test obj3;

cout<<"Leave block"<<endl;

}

cout<<"Leave main"<<endl;

return 0;

}

1. **Write a program to implement (a) pointer to an object (b) this pointer. Practice both ‘.’ (dot operator) and ‘->’ (arrow operator).**

#include<iostream>

using namespace std;

class test{

int a;

public:

void setdata(int a);

void display(void);

};

void test::setdata(int a){

this->a=a;

}

void test::display(){

cout<<"num is: "<<a;

}

int main(){

test \*ptr;

test obj;

ptr=&obj;

(\*ptr).setdata(10);

ptr->display();

return 0;

}

1. **Write a program to swap private values of two classes using a friend function.**

#include<iostream>

using namespace std;

class test2;

class test1{

int num;

public:

void setvalue(int a){

num=a;

}

friend void swap(test1 obj1,test2 obj2);

};

class test2{

int num;

public:

void setvalue(int a){

num=a;

}

friend void swap(test1, test2);

};

void swap(test1 obj1, test2 obj2){

int temp;

temp=obj1.num;

obj1.num=obj2.num;

obj2.num=temp;

cout<<"Obj1 is: "<<obj1.num<<" "<<"Obj2 is: "<<obj2.num;

}

int main(){

test1 obj1;

test2 obj2;

obj1.setvalue(5);

obj2.setvalue(8);

swap(obj1, obj2);

return 0;

}

1. **Write a program to add data objects of two different classes using friend functions.**

#include<iostream>

using namespace std;

class test2;

class test1{

int num;

public:

void setnum(){

num=5;

}

friend void sum(test1, test2);

};

class test2{

int num;

public:

void setnum(){

num=6;

}

friend void sum(test1, test2);

};

void sum(test1 obj1, test2 obj2){

cout<<"sum is: "<<obj1.num+obj2.num<<endl;

}

int main(){

test1 obj1;

obj1.setnum();

test2 obj2;

obj2.setnum();

sum(obj1,obj2);

}

1. **Write a program to demonstrate the working of friend class.**

//Friend classes

#include<iostream>

using namespace std;

class frnd;

class test{

private:

int num\_pri;

protected:

int num\_pro;

public:

int num\_pub;

test(void){

num\_pri=10;

num\_pro=20;

num\_pub=30;

}

friend class frnd;

};

class frnd{

public:

void display(test &obj1){

cout<<"num\_pri is: "<<obj1.num\_pri<<endl;

cout<<"num\_pro is: "<<obj1.num\_pro<<endl;

cout<<"num\_pub is: "<<obj1.num\_pub<<endl;

}

};

int main(){

test obj1;

frnd obj2;

obj2.display(obj1);

return 0;

}

1. **Implement dynamic memory allocation. Use *new* and *delete* keywords.**

#include<iostream>

using namespace std;

class test{

int \*p;

public:

test(int num){

p= new int;

\*p=num;

cout<<"In constructor, \*p is: "<<\*p<<endl;

}

~test(){

delete p;

cout<<"In destructor, p deleted"<<endl;

}

};

int main(){

int num;

cout<<"Enter number: "<<endl;

cin>>num;

test obj(num);

return 0;

}