**ARRAY OF CLASS OBJECTS**

#include <iostream>

using namespace std;

class Rectangle

{

int length;

int width;

public:

Rectangle(int =1,int=1);

void printValues();

};

Rectangle::Rectangle(int a, int b)

{

length=a; width=b;

}

void Rectangle::printValues()

{cout<<“length=“<<length<<endl;

cout<<“width=“<<width;

}

void main()

{

Rectangle R1; //Default constructor is called

**Rectangle RectArray[3];**

**//Default constructor is called for each object**

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

**RectArray[i].print(); //for each object, call print function**

**}**

**AN EXAMPLE ABOUT CONSTRUCTORS**

#include <iostream>

using namespace std;

class A

{

public:

void print() const;

**A (int =0, int =0); //constructor with default parameter**

private:

int a1;

int a2;

};

int main ()

{

**A object\_1(10); //which constructor should be called??**

object\_1.print();

return 0;

}

void A::print()const

{ cout<<"a1 = "<<a1<<"\ta2 = "<<a2<<endl; }

A::A(int x, int y)

{

a1=x;

a2=y;

**}**

**Output:**

**a1 = 10**

**a2 = 0**

**CONSTANT DATA MEMBERS OF CLASS**

#include <iostream>

using namespace std;

class A

{

public:

void print() const;

**A (int =0, int =0); //constructor with default parameter**

private:

int a1;

**const int a2;**

};

void A::print()const

{ cout<<"a1 = "<<a1<<"\ta2 = "<<a2<<endl; }

**SYNTAX ERROR!!!!!**

a.cpp: In constructor `A::A(int, int)':

a.cpp:32: assignment of read-only data-member `A::a2'

A::A(int x, int y)

{

a1=x;

a2=y;

**}**

int main ()

{

A object\_1(1,2); //OKAY, WE WANT: a1🡨1 , a2🡨 2

object\_1.print();

return 0;

}

When constant member is declared in private data member, the constructor should be defined as follows:

A::A(int x, int y):a2(y)

{

**Member initiallize**

a1=x;

**}**

**ARRAYS AS DATA MEMBERS OF CLASS**

Consider a class of Courses that stores roll numbers of registered student. Let us assume that in a course maximum 35 students can be registered and this maximum size is known at the compile time. But we also need to put a check for the number of currently registered students.

When a new course is offered, the number of students is zero. As students registers in the course, number of students increases.

class Course

{

private:

string CourseName;

int students[35];

int currently\_registered;

public:

Course(string=“-”);//default value

void RegisterStudent(int);

void main()

{

Course calculus(“calculus-1”);

calculus.RegisterStudent(212);

calculus.RegisterStudent(213);

calculus.RegisterStudent(214);

calculus.PrintDetails();

};

void PrintDetails() const;

};

Course::Course(string cname)

{

currently\_registered=0;

CourseName=cname;

for (int i=0; i<35; i++)

students[i]=0;

**OUTPUT:**

coursename: calculus-1

current number of registered students: 3

list of registered students

212

213

214

}

void Course::RegisterStudent(int x)

{

if(currently\_registered<35)

{students[currently\_registered]=x;

currently\_registered++;}

}

void Course::PrintDetails()

{

cout<<”coursename: ”<<CourseName;

cout<<”current number of registered students”<<currently\_registered;

cout<<”list of registered students”;

for (int i=0; i<currently\_registered; i++)

cout<<students[i];}

**POINTERS AS DATA MEMBERS OF CLASS**

Now we don’t know the maximum number of students that can be registered. So we will make a dynamic array of students. **We should customize our copy constructor for such classes**

class Course

{

private:

string CourseName;

int \*students;

int capacity;

int currently\_registered;

public:

Course(string=”-“, int =35);//default values

void RegisterStudent(int);

void main()

{

Course calculus(“calculus-1”,40);

calculus.RegisterStudent(212);

calculus.RegisterStudent(213);

calculus.RegisterStudent(214);

calculus.PrintDetails();

};

void PrintDetails() const;

};

Course::Course(string cname, int c){

currently\_registered=0;

CourseName=cname;

capacity=c;

students=new int[capacity];

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

students[i]=0;}

**OUTPUT:**

coursename: calculus-1

capacity: 40

current number of registered students: 3

list of registered students

212

213

214

void Course::RegisterStudent(int x)

{

if(currently\_registered< capacity)

{

students[currently\_registered]=x;

currently\_registered++;}

}

void Course::PrintDetails()

{cout<<”coursename: ”<<coursename;

cout<<”Capacity: “<<capacity;

cout<<”current number of registered students”<<currently\_registered;\

cout<<”list of registered students”;

for (int i=0; i<currently\_registered; i++)

cout<<students[i];}

**EXERCISE: Create a copy constructor, and a DropStudent() function for this class**

**DESTRUCTOR**

* Destructors are opposite of constructors
* A constructor is invoked when an object is created, where as a destructor is invoked when an object is destroyed.
* A Destructor releases the memory of object allocated by the constructor
* A destructor is called by the constructor, when an object goes out of scope
* Every class has a default destructor if the destructor is not explicitly defined
* It cannot not have any parameters and no return value
* We normally write destructors when we have pointers as data members
* Destructors are named the same way as constructors, but we put a tilde character (~) in the beginning
* Example:

class Course

{

private:

string CourseName;

int \*students;

int capacity;

int currently\_registered;

public:

Course(string=”-“, int =35);//default values

void RegisterStudent(int);

void PrintDetails() const;

~Course();

};

Course::~Course()

{

Cout<<”\n Destructor is called!!! \n”;

delete [] students;

}

**Output:**

Destructor is called!!!

Destructor is called!!!

void f1()

{

Course C;

};

void main()

{

Course C1;

f1();

}

**Order in which Constructors and Destructors are called**

#include <iostream>

using namespace std;

class test

{

private:

int value;

public:

test(int);

~test();

};

test::test(int a)

{

value=a;

cout<<"\n Contructor value="<<value;

}

test::~test()

{

cout<<"\n Destructor value="<<value;

}

test t1(1); //Global Variable

void func1()

{

Output:

Contructor value=1

Contructor value=2

Contructor value=3

Destructor value=3

Contructor value=4

Contructor value=5

Destructor value=5

Destructor value=4

Destructor value=2

Destructor value=1

test t3(3);

}

void func2()

{

test t5(5);

}

int main()

{

test t2(2);

func1();

test t4(4);

func2();

return 0;

}