Rajvaibhav Rahane

17u283 223045

SE-C Comp,Viit,Pune

**Aim: Study the various concepts of classes.**

**THEORY:-**

1. **constructor :-**

A constructor is a member function of a class which initializes objects of a class. In C++, Constructor is automatically called when object(instance of class) create. It is special member function of the class.

1. **Default constructor:-**

Default constructor is the constructor which doesn’t take any argument. It has no parameters.

1. **Copy constructor :-**

A copy constructor is a member function which initializes an object using another object of the same class. Detailed article on [Copy Constructor](https://www.geeksforgeeks.org/copy-constructor-in-cpp/).

1. **Destructor :-**

Destructor is a member function which destructs or deletes an object.

A destructor function is called automatically when the object goes out of scope:  
(1) the function ends  
(2) the program ends  
(3) a block containing local variables ends  
(4) a delete operator is called

**5. static member function:-**

By declaring a function member as static, you make it independent of any particular object of the class. A static member function can be called even if no objects of the class exist and the static functions are accessed using only the class name and the scope resolution operator ::.

A static member function can only access static data member, other static member functions and any other functions from outside the class.

Static member functions have a class scope and they do not have access to the this pointer of the class. You could use a static member function to determine whether some objects of the class have been created or not.

1. **Friend function:-**

A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions.

A friend can be a function, function template, or member function, or a class or class template, in which case the entire class and all of its members are friends.

To declare a function as a friend of a class, precede the function prototype in the class definition with keyword friend.

1. **Friend class:-**

A friend class can access private and protected members of other class in which it is declared as friend. It is sometimes useful to allow a particular class to access private members of other class. For example a LinkedList class may be allowed to access private members of Node.

1. **This pointer:-**

Every object in C++ has access to its own address through an important pointer called **this** pointer. The **this** pointer is an implicit parameter to all member functions. Therefore, inside a member function, this may be used to refer to the invoking object.

Friend functions do not have a **this** pointer, because friends are not members of a class. Only member functions have a **this** pointer.

1. **Inline code:-**

C++ inline function is powerful concept that is commonly used with classes. If a function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time.

Any change to an inline function could require all clients of the function to be recompiled because compiler would need to replace all the code once again otherwise it will continue with old functionality.

To inline a function, place the keyword inline before the function name and define the function before any calls are made to the function. The compiler can ignore the inline qualifier in case defined function is more than a line.

A function definition in a class definition is an inline function definition, even without the use of the inline specifier.

1. **Dynamic memory allocation:-**
2. Memory allocated "on the fly" during run time

2) dynamically allocated space usually placed in a program segment known as the *heap* or the *free store*

3) Exact amount of space or number of items does not have to be known by the compiler in advance.

4)For dynamic memory allocation, pointers are crucial

***CODE:***

#include<bits/stdc++.h>

using namespace std;

class Student{

static int objCount;

char \*name;

int rollNo;

char \*year;

char division;

int dd,mm,yy;

char \*address;

char \*phone;

public:

static int getObjectCount(){

return objCount;

}

Student(Student &s){

cout<<"Copy Constructor called\n";

name=year=address=phone=NULL;rollNo=dd=mm=yy=-1;

}

//int calculateLength(char \*str);

Student(char \*name,int rollNo,char \*year,char division,int dd,int mm,int yy,char \*address,char \*phone){

//Parameterised Constructor;

cout<<"Parameterised Constructor called\n";

this->name=name; //use of this pointer

this->rollNo=rollNo;

this->year=year;

this->division=division;

this->dd=dd;this->mm=mm;this->yy=yy;

this->address=address;

this->phone=phone;

}

Student(){ //Empty Constructor

cout<<"Default Constructor called\n";

name=year=address=phone=NULL;rollNo=dd=mm=yy=-1;objCount++;

}

~Student(){ //destructor;

if(name!=NULL){delete name;name=NULL;}

if(year!=NULL){delete year;year=NULL;}

if(address!=NULL){delete address;address=NULL;}

if(phone!=NULL){delete phone;phone=NULL;}

cout<<"Destructing Student\n";

}

friend istream & operator>>(istream &in,Student&); //friend fx

friend ostream & operator<<(ostream &,const Student&);

};

int Student::objCount=0;

int calculateLength(char \*str){

int i=0;

if(str!=NULL){

for(i=0;str[i]!='\0';i++);

}

return i;

}

istream & operator>>(istream &in,Student & s){

s.name=new char[16];s.year=new char[3];s.address=new char[41];s.phone=new char[11];

cout<<"Name(15) : ";scanf("%[^\n]%\*c",s.name);

s.name[calculateLength(s.name)]='\0';

cout<<"PRN No : ";scanf("%d%\*c",&s.rollNo);

cout<<"Year(2) : ";scanf("%[^\n]%\*c",s.year);s.year[calculateLength(s.year)]='\0';

cout<<"Div(1) : ";scanf("%c%\*c",&s.division);

cout<<"DOB : ";scanf("%d%d%d%\*c",&s.dd,&s.mm,&s.yy);

cout<<"Address(40) : ";scanf("%[^\n]%\*c",s.address);s.address[calculateLength(s.address)]='\0';

cout<<"Phone(10) : ";scanf("%[^\n]%\*c",s.phone);s.phone[calculateLength(s.phone)]='\0';

return in;

}

ostream & operator<<(ostream &out,const Student& s){

if(s.name!=NULL)

out<<s.name<<endl;

else

out<<"NULL Name";

out<<s.rollNo<<" ";

if(s.year!=NULL)

out<<s.year<<"\t";

else

out<<"NULL Year";

out<<s.division<<endl;

if(s.address!=NULL)

out<<s.address<<endl;

else

out<<"NULL Address";

out<<s.dd<<" "<<s.mm<<" "<<s.yy<<" ";

if(s.phone!=NULL)

out<<s.phone<<endl;

else

out<<"NULL Phone";

return out;

}

void createStudent(){

char \*name=new char[16],\*year=new char[3],\*address=new char[41],\*phone=new char[11];

int rollNo,dd,mm,yy;char division;

cout<<"Name(15) : ";scanf("%[^\n]%\*c",name);

name[calculateLength(name)]='\0';

cout<<"PRN No : ";scanf("%d%\*c",&rollNo);

cout<<"Year(2) : ";scanf("%[^\n]%\*c",year);year[calculateLength(year)]='\0';

cout<<"Div(1) : ";scanf("%c%\*c",&division);

cout<<"DOB : ";scanf("%d%d%d%\*c",&dd,&mm,&yy);

cout<<"Address(40) : ";scanf("%[^\n]%\*c",address);address[calculateLength(address)]='\0';

cout<<"Phone(10) : ";scanf("%[^\n]%\*c",phone);phone[calculateLength(phone)]='\0';

{cout<<"Creating parameterised object\n";Student parameterised(name,rollNo,year,division,dd,mm,yy,address,phone);}

cout<<"Leaving createStudent()\n";

}

void printMenu(){

cout<<"1)Create Student\t";

cout<<"2)End\tChoice? : ";

}

int main(){

int choice;

createStudent();

{cout<<"Creating simple object\n";Student s2;cout<<endl;cout<<"Creating copy\n";Student s1(s2);cout<<endl;}

printMenu();

do{

scanf("%d%\*c",&choice);

switch(choice){

case 1:{

Student s;//=createStudent();

cout<<"Object No : "<<Student::getObjectCount()<<endl;

cin>>s;

cout<<s<<endl;

break;

}

case 2:{break;}

}

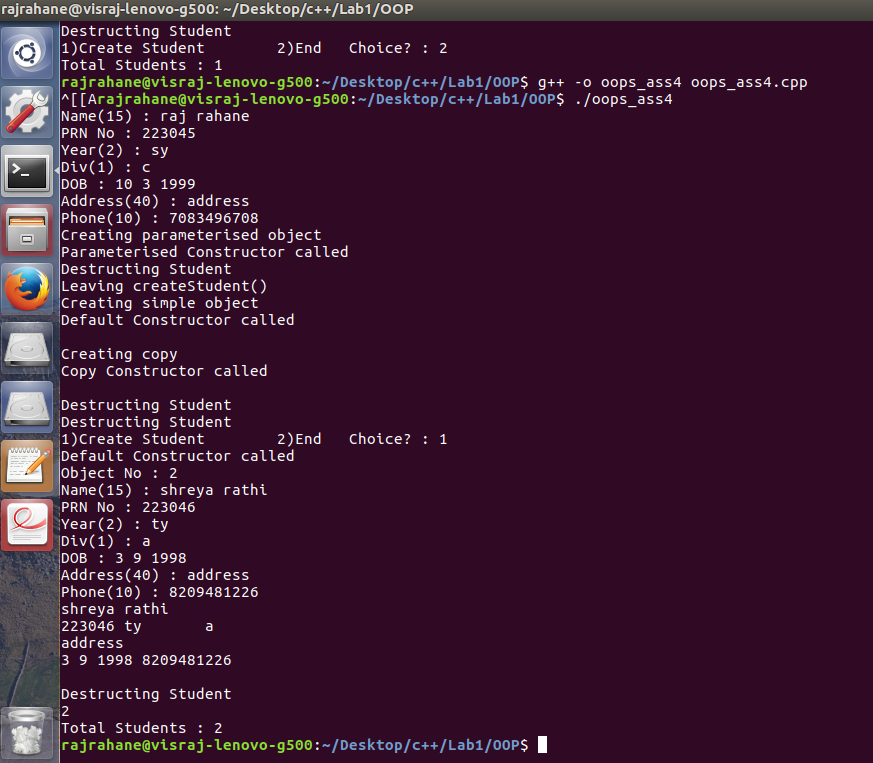
}while(choice!=2);

cout<<"Total Students : "<<Student::getObjectCount()<<endl;

return 0;

}

***Output:***



Conclusion-The Various features of a Class in C++ were studied