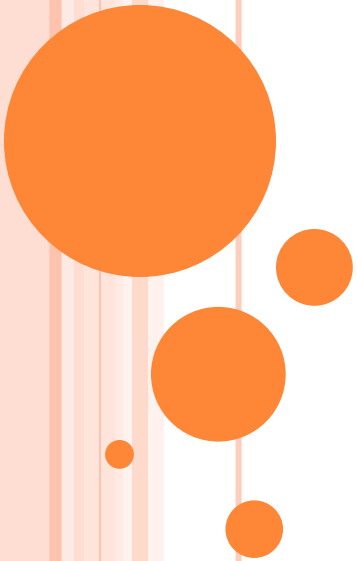


OOP/Computer Programming

By : Dr. Danish Shehzad

Review

- Class
- Objects
- Functions (Inline, Out of line)
- Interface Vs. Implementation
- Constructors
- Constructor overloading
- Default Constructors
- Copy Constructor
- Shallow Copy vs. Deep Copy
- Destructors
- This Pointer
- Constant Functions
- Constant Data Members
- Member Initializer
- Constant Objects

Today's Lecture

- STATIC MEMBERS
- ARRAY OF OBJECTS
- POINTERS TO OBJECTS

Static Data Member

Definition

"A variable that is part of a class, yet is not part of an object of that class, is called static data member"

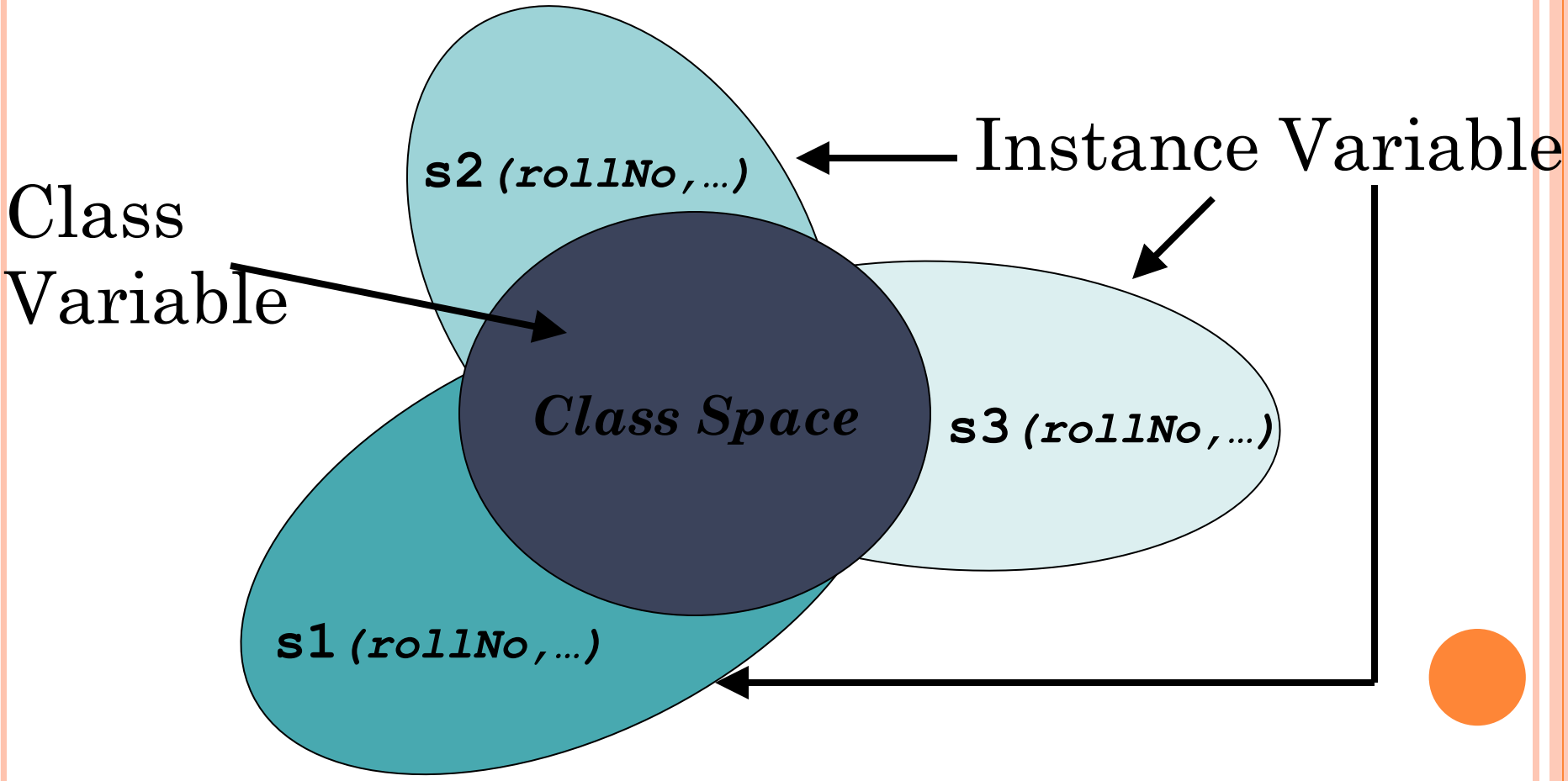
Static Data Member (Syntax)

- Keyword static is used to make a data member static

```
class ClassName{
…
static DataType VariableName;
};
```

Class vs. Instance Variable

- **`Student s1, s2, s3;`**



Static Data Member

- They are shared by all instances of the class
- They do not belong to any particular instance of a class

Defining Static Data Member

- Static data member is declared inside the class
- But they are defined outside the class

Defining Static Data Member

```
class ClassName{
…
static DataType VariableName;
};

DataType ClassName::VariableName;
```

Example

```
class Student{
private:
    static int noOfStudents;
public:
   …
};
int Student::noOfStudents = 0;
/*private static member cannot be accessed
outside the class except for initialization*/
```

Initializing Static Data Member

- If static data members are not explicitly initialized at the time of definition then they are initialized to 0

Example

```
int Student::noOfStudents;
```

is equivalent to

```
int Student::noOfStudents=0;
```

Accessing Static Data Member

- To access a static data member there are two ways
  - Access like a normal data member
  - Access using a scope resolution operator '::'

Example

```
class Student{
public:
   static int noOfStudents;
};
int Student::noOfStudents;
int main(){
   Student aStudent;
   aStudent.noOfStudents = 1;
   Student::noOfStudents = 1;
}
```

Life of Static Data Member

- They are created even when there is no object of a class
- They remain in memory even when all objects of a class are destroyed

Example

```
class Student{
public:
   static int noOfStudents;
};
int Student::noOfStudents;
int main(){
   {
     Student aStudent;
     aStudent.noOfStudents = 1;
   }
   Student::noOfStudents = 1;
}
```

Uses

- They can be used to store information that is required by all objects, like global variables

Example

- Modify the class Student such that one can know the number of student created in a system

Example

```
class Student{
…
public:
  static int noOfStudents;
  Student();
  ~Student();
…
};
int Student::noOfStudents = 0;
```

Example

```
Student::Student(){
   noOfStudents++;
}
Student::~Student(){
   noOfStudents--;
}
```

Example

```
int Student::noOfStudents = 0;
int main(){
  cout <<Student::noOfStudents <<endl;
  Student studentA;
  cout <<Student::noOfStudents <<endl;
  Student studentB;
  cout <<Student::noOfStudents <<endl;
}
```

Output:
0
1
2

Static Member Functions

- Static member function can access only static data members or functions

Practice 1:

```
#include <iostream>

using namespace std;

class Demo
{
        private:
                //static data members
                static int X;
                static int Y;

        public:
        //static member function
        static void  Print()
        {
                cout <<"Value of X: " << X << endl;
                cout <<"Value of Y: " << Y << endl;
        }
};

//static data members initializations
int Demo :: X =10;
int Demo :: Y =20;
```

```
int main()
{
        Demo OB;
        //accessing class name with object name
        cout<<"Printing through object name:"<<endl;
        OB.Print();

        //accessing class name with class name
        cout<<"Printing through class name:"<<endl;
        Demo::Print();

        return 0;
}
```

Output:
Printing through object name:
Value of X: 10
Value of Y: 20
Printing through object name:
Value of X: 10
Value of Y: 20

Array of Objects

- An object of class represents a single record in memory, if we want more than one record of class type, we have to create an array of objects.

- As we know, an array is a collection of similar type, therefore an array can be a collection of class type.

Example

```
class Test{
public:
  Test();
};
int main(){
  Test array[2]; // OK
}
```

Practice 2:

```
class Employee
{
	int Id;
	char Name[25];
	int Age;
	long Salary;

	public:
	void GetData()		//Statement 1 : Defining GetData()
	{
		cout<<"\n\tEnter Employee Id : ";
		cin>>Id;

		cout<<"\n\tEnter Employee Name : ";
		cin>>Name;

		cout<<"\n\tEnter Employee Age : ";
		cin>>Age;

		cout<<"\n\tEnter Employee Salary : ";
		cin>>Salary;
	}

	void PutData()		//Statement 2 : Defining PutData()
	{
		cout<<"\n"<<Id<<"\t"<<Name<<"\t"<<Age<<"\t"<<Salary;
	}

};
```

```
void main()
{

    int i;

    Employee E[3];          //Statement 3 : Creating Array of 3 Employees

    for(i=0;i<3;i++)
    {
        cout<<"\nEnter details of "<<i+1<<" Employee";
        E[i].GetData();
    }

    cout<<"\nDetails of Employees";
    for(i=0;i<3;i++)
    E[i].PutData();

}
```

II. Pointer to Objects

- A variable that holds an address value is called a pointer variable or simply pointer

- Pointer to objects are similar as pointer to built-in types

Example

```
class Student{
…
public:
  Student();
  Student(char * aName);
  void setRollNo(int aNo);
};
```

```
int main(){
  Student obj;
  Student *ptr;
  ptr = &obj;
  obj.setRollNo(5);
  ptr->setRollNo(10);
  (*ptr).setRollNo(10);
  return 0;
}
```

Allocation with new Operator

- Sometimes we donot know, at the time that we write the program , how many objects we want to create.
- When this is the case we can use new to create objects while the program is running.
- ‘new’ returns a pointer to unnamed objects.

Example

```
int main(){
  Student *ptr;
  ptr = new Student;
  ptr->setRollNo(10);
  return 0;
}
```

```
class student
{
private:
        int rollno;
        string name;
public:
        student():rollno(0),name("")
        {}
        student(int r, string n): rollno(r),name (n)
        {}
        void get()
        {
                cout<<"enter roll no";
                cin>>rollno;
                cout<<"enter  name";
                cin>>name;
        }
        void print()
        {
                cout<<"roll no is "<<rollno;
                cout<<"name is "<<name;
        }
};
```

```
void main ()
{
        student *ps=new student;
        (*ps).get();
        (*ps).print();
        delete ps;
}
```

Practice 3:

Example array of Objects

```
#include <iostream>
using namespace std;
class Box {
public: Box()
 {
cout << "Constructor called!" <<endl; }
 ~Box() { cout << "Destructor called!" <<endl;
} };
int main()
 {
Box* myBoxArray = new Box[4];
delete [] myBoxArray; // Delete array return 0;
 }
```

III. Case Study

Design a class date through which user must be able to perform following operations

- Get and set current day, month and year
- Increment by x number of days, months and year
- Set default date

Attributes

- Attributes that can be seen in this problem statement are
  - Day
  - Month
  - Year
  - Default date

Attributes

- The default date is a feature shared by all objects
  - This attribute must be declared a static member

Attributes in Date.h

```
class Date
{
  int day;
  int month;
  int year;
  static Date defaultDate;
…
};
```

Interfaces

- getDay
- getMonth
- getYear
- setDay
- setMonth
- setYear
- addDay
- addMonth
- addYear
- setDefaultDate

Interfaces

- As the default date is a static member the interface setDefaultDate should also be declared static

Interfaces in Date.h

```
class Date{
…
public:
  void setDay(int aDay);
  int getDay() const;
  void addDay(int x);
  …
…
};
```

Interfaces in Date.h

```
class Date{
…
public:
Static void setDefaultDate(
int aDay,int aMonth, int aYear);
…
};
```

Implementation of Date Class

- The static member variables must be initialized

Date Date::defaultDate (01,01,2019);

Constructors

```
Date::Date(int aDay, int aMonth, int aYear)
 {
   if(aDay==0) {
         this->day = defaultDate.day;
   }
   else{
         setDay(aDay);
   }
   //similarly for other members
}
```

Destructor

- We are not required to do any house keeping chores in destructor

```
Date::~Date
{
}
```

Getter and Setter

```
void Date::setMonth(int a){
  if(a > 0 && a <= 12){
      month = a;
}
```