Static Data Members

Muhammad Hammad Waseem

m.hammad.wasim@gmail.com

Static Data Member

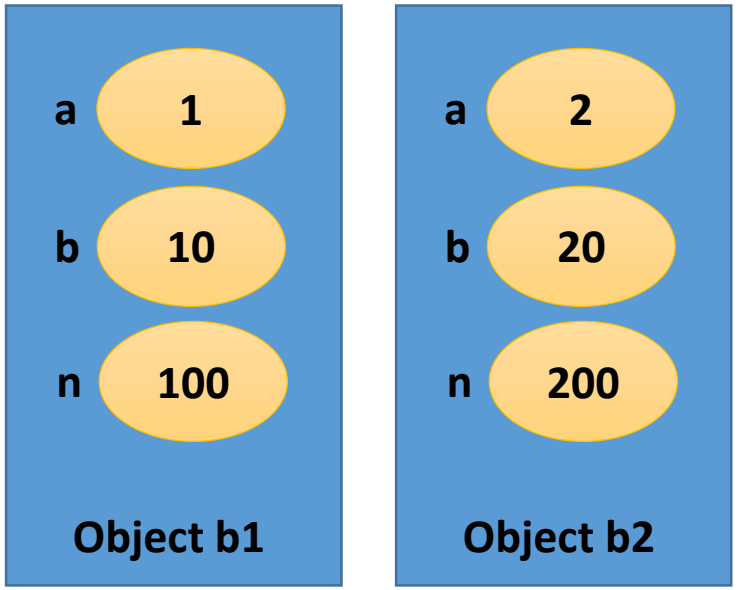
- A type of data member that is shared among all objects of class is known as **static data member**.
- The static data member is defined in the class with ***static*** keyword.
- When a data member is defined as static, only *one variable is created* in the memory even if there are *many objects* of that class.
- A static data item is useful when all objects of the same class must share a common item of information.
- The characteristics of a static data member are same as normal static variable.

Static Data Member

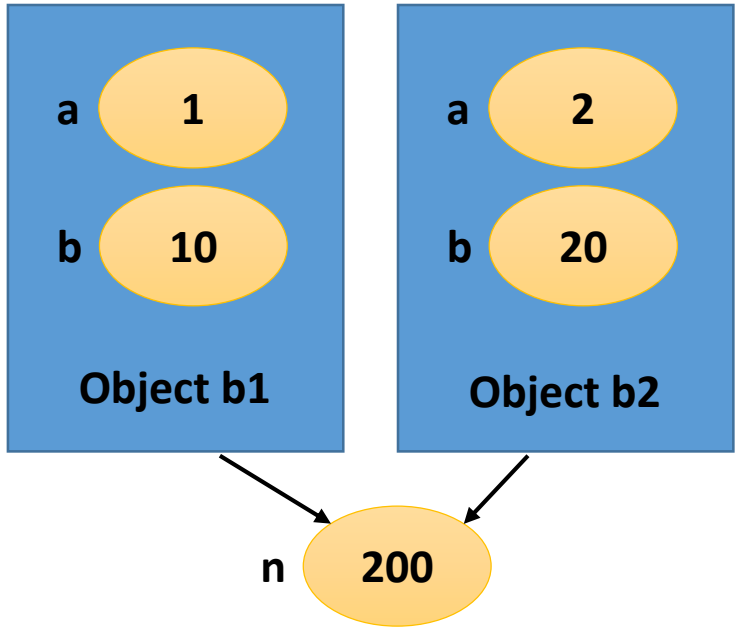
- It is visible only in the class, in which it is defined but its lifetime
  - *Starts* when the *program starts* its execution.
  - *Ends* when the entire program is *terminated*.
- It is normally used to share some data among all objects of a particular class.
- The main difference between normal data member and static data member is that
  - each object has its own variable of normal data member.
  - On the other hand, static data member is **shared among all objects** of the class.
- Only **one memory location** is created for **static data member** that is shared among all objects.

Difference Between Normal & Static Data Members

Object with three normal data members



Object with two normal data members and one static data member



Uses of Static Class Data

- Why would you want to use static member data?
- As an example, suppose an object needed to know how many other objects of its class were in the program.
- for example :
  - In a road-racing game, a race car might want to know how many other cars are still in the race.
- In this case a static variable count could be included as a member of the class. All the objects would have access to this variable.
- It would be the same variable for all of them; they would all see the same count.

Separate Declaration and Definition

- Static member data requires an unusual format.
- Ordinary variables are usually declared and defined in the same statement.
- Static member data, on the other hand, requires two separate statements.
  - The variable's declaration appears in the class definition, but the
  - Variable is defined outside the class, in much the same way as a global variable.
- Why is this two-part approach used?
  - If static member data were defined inside the class, it would violate the idea that a class definition is only a blueprint and does not set aside any memory.

Separate Declaration and Definition

- Putting the definition of static member data outside the class also serves to emphasize that
  - the memory space for such data is allocated only once, before the program starts to execute, and that
  - one static member variable is accessed by an entire class; each object does not have its own version of the variable, as it would with ordinary member data.
- In this way a static member variable is more like a global variable.

Write a program that counts the number of objects created of a particular class (1/2)

```
class yahoo
{
private:
static int n;
public:
yahoo()
{ n++; }
void show()
{
cout<<"you have created "<<n<<" objects so far "<<endl;
}
};
```

Write a program that counts the number of objects created of a particular class (2/2)

```
int yahoo::n=0;
void main()
{
yahoo x,y;
x.show();
yahoo z;
x.show();
}
```

OUTPUT:

- You have created 2 objects so far.
- You have created 3 objects so far.

How it Works

- The program declares a static data member `n` to count the number of objects that have been created.
- The statement `int yahoo::n=0;` defines the variable and initializes it to 0.
- The variable is defined outside the class because it will be **not part** of any object.
- It is *created only once* in the memory and is *shared among all* objects of the class.
- The variable definition outside the class must be preceded with class name and **scope resolution operator** `::`.
- The compiler ***does not*** display any **error** if the static data member is ***not defined***.
- The ***linker*** will generate an **error** when the ***program is executed***.
- The above program creates three objects `x, y` and `z`. each time an object is created, the constructor is executed that increases the value of `n` by 1.

Write a program that creates three objects of class student. Each of them must assigned a unique roll number. (Hint: use static data member for unique roll number) (1/2)

```
class Student
{
private:
static int r;
int rno,marks;
char name[30];
public:
Student()
{ r++;
Rno =r; }
void in()
{
cout<<"enter name:";
gets(name);
cout<<"enter marks:";
cin>>marks;
}
void show()
{
cout<<"Roll No:"<<rno<<endl;
cout<<"Name:"<<name<<endl;
cout<<"Marks:"<<marks<<endl;
}
};
```

Write a program that creates three objects of class student. Each of them must assigned a unique roll number. (Hint: use static data member for unique roll number) (2/2)

```
int Student::r=0;
void main()
{
Student s1,s2,s3;
s1.in();
s2.in();
s3.in();
s1.show();
s2.show();
s3.show();
}
```

OUTPUT

- Enter name: Farhan Khan
- Enter marks: 600
- Enter name: Zeeshan
- Enter marks: 786
- Enter name: Ali
- Enter marks: 567

- Roll no: 1
- Name: Farhan Khan
- Marks: 600
- Roll no: 2
- Name: Farhan Khan
- Marks: 786
- Roll no: 3
- Name: Ali
- Marks: 567

How it Works

- The above program uses a static data member `r` to assign unique roll numbers to each object of the `class Student`.
- The static data member is initialized to 0.
- The constructor increments its value by 1 when an object is created and then assigns the updated value of `r` to the data member `rno`.
- It ensures that each object gets a unique roll number.