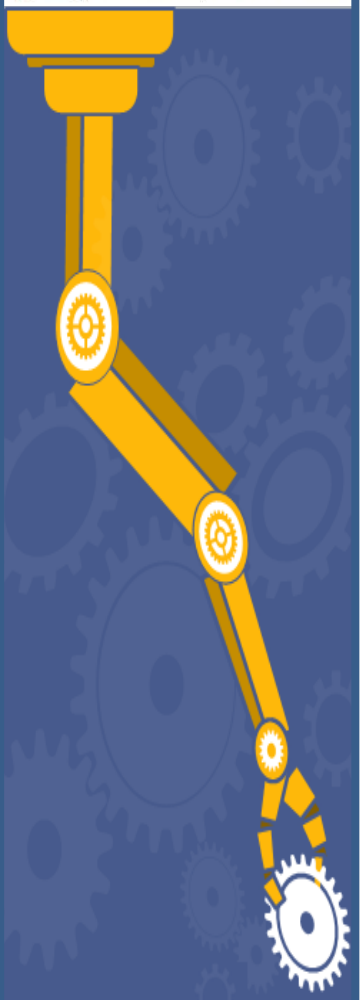
VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)



# BCSE102L- Structured and object-oriented programming

**Dr. P.Keerthika**
Associate Professor
School of Computer Science & Engineering
VIT, Vellore.

# BCSE102L- Structured and object-oriented programming

| | | |
|---|---|---|
| **Module:1** | **C Programming Fundamentals** | **2 hours** |
| Variables - Reserved words – Data Types – Operators – Operator Precedence - Expressions - Type Conversions - I/O statements - Branching and Looping: if, if-else, nested if, if-else ladder, switch statement, goto statement - Loops: for, while and do…while – break and continue statements. | | |
| **Module:2** | **Arrays and Functions** | **4 hours** |
| Arrays: One Dimensional array - Two-Dimensional Array – Strings and its operations. User Defined Functions: Declaration – Definition – call by value and call by reference - Types of Functions - Recursive functions - Storage Classes - Scope, Visibility and Lifetime of Variables. | | |
| **Module:3** | **Pointers** | **4 hours** |
| Declaration and Access of Pointer Variables, Pointer arithmetic – Dynamic memory allocation – Pointers and arrays - Pointers and functions. | | |
| **Module:4** | **Structure and Union** | **2 hours** |
| Declaration, Initialization, Access of Structure Variables - Arrays of Structure - Arrays within Structure - Structure within Structures - Structures and Functions – Pointers to Structure - | | |
| **Module:5** | **Overview of Object-Oriented Programming** | **5 hours** |
| Features of OOP - Classes and Objects - "this" pointer - Constructors and Destructors - Static Data Members, Static Member Functions and Objects - Inline Functions – Call by reference - Functions with default Arguments - Functions with Objects as Arguments - Friend Functions and Friend Classes. | | |
| **Module:6** | **Inheritance** | **5 hours** |
| Inheritance - Types of Inheritance: Single inheritance, Multiple Inheritance, Multi-level Inheritance, Hierarchical Inheritance - Multipath Inheritance - Inheritance and constructors. | | |
| **Module:7** | **Polymorphism** | **4 hours** |
| Function Overloading - Operator Overloading – Dynamic Polymorphism - Virtual Functions - Pure virtual Functions - Abstract Classes. | | |
| **Module:8** | **Generic Programming** | **4 hours** |
| Function templates and class templates, Standard Template Library. | | |
| | **Total Lecture hours:** | **30 hours** |

# BCSE102L- Structured and object-oriented programming – Text Books and Reference Books

| | |
|---|---|
| **Text Book(s)** | |
| 1. | Herbert Schildt, C: The Complete Reference, 4th Edition, McGraw Hill Education, 2017 |
| 2. | Herbert Schildt, C++: The Complete Reference, 4th Edition, McGraw Hill Education, 2017. |
| **Reference Books** | |
| 1. | Yashavant Kanetkar, Let Us C: 17th Edition, BPB Publicaitons, 2020. |
| 2. | Stanley Lippman and Josee Lajoie, C++ Primer, 5th Edition, Addison-Wesley publishers, 2012. |

VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

# BCSE102P- Structured and object-oriented programming Laboratory

| | Indicative Experiments |
|---|---|
| 1. | Programs using basic control structures, branching and looping |
| 2. | Experiment the use of 1-D, 2-D arrays and strings and Functions |
| 3. | Demonstrate the application of pointers |
| 4. | Experiment structures and unions |
| 5. | Programs on basic Object-Oriented Programming constructs. |
| 6. | Demonstrate various categories of inheritance |
| 7. | Program to apply kinds of polymorphism. |
| 8. | Develop generic templates and Standard Template Libraries. |

| **Text Book(s)** | |
|---|---|
| 1. | Robert C. Seacord, Effective C: An Introduction to Professional C Programming, 1st Edition, No Starch Press, 2020. |
| **Reference Book(s)** | |
| 1. | Vardan Grigoryan and Shunguang Wu, Expert C++: Become a proficient programmer by learning coding best practices with C++17 and C++20's latest features, 1st Edition, Packt Publishing Limited, 2020. |


# BCSE102L- Structured and Object-Oriented Programming

- **Module-5: Overview of Object Oriented Programming**
  - **Features of OOP- Classes and Objects**
  - **Constructors and Destructors**
  - **Static Data Members and Member Functions**
  - **Inline Functions**
  - **Functions with Default Arguments**
  - **Functions with Objects as Arguments**
  - **Friend Functions and classes**


# Static Data Members

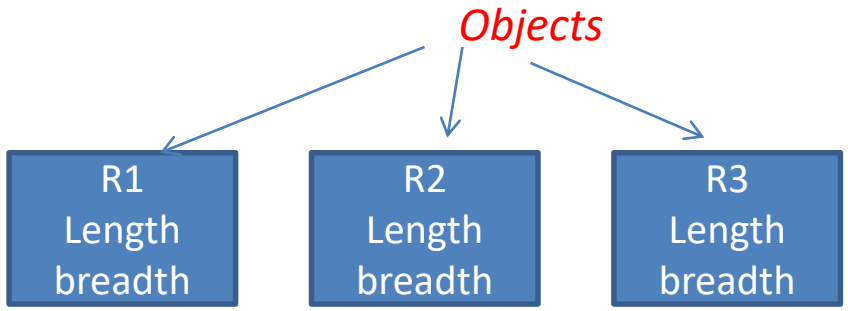
- Static data members are declared by using the **static keyword inside the class**, but as they have the lifeline until the program runs and is accessible to every class object they must have to initialize outside the class.
- To initialize static data members, we use the scope resolution operator to access them and initialize after the class declaration and before the main function.
- As the memory block of static data members is shared by every object of the class. So, if there is any change made in the static data member then it will be updated for every other object of the class.
- Static data members can be accessed anywhere in the program after the declaration of class either using the class instance or using scope resolution, class name, and variable name. There is no need to create objects to access them.

**datatype class_name:: variable name=value**


# Why Static Data Members??

```
class room
{
private:
    int length;
    int breadth;
public:
    room(int l, int b)
    {
            length =l;
            breadth=b;
     }
     int area()
     {
            return length*breadth;
      }
};
```

```
int main()
{
    room r1(10,20);
    room r2(20,30);
    room r3(15,20);
    cout<<r3.area();
}
```

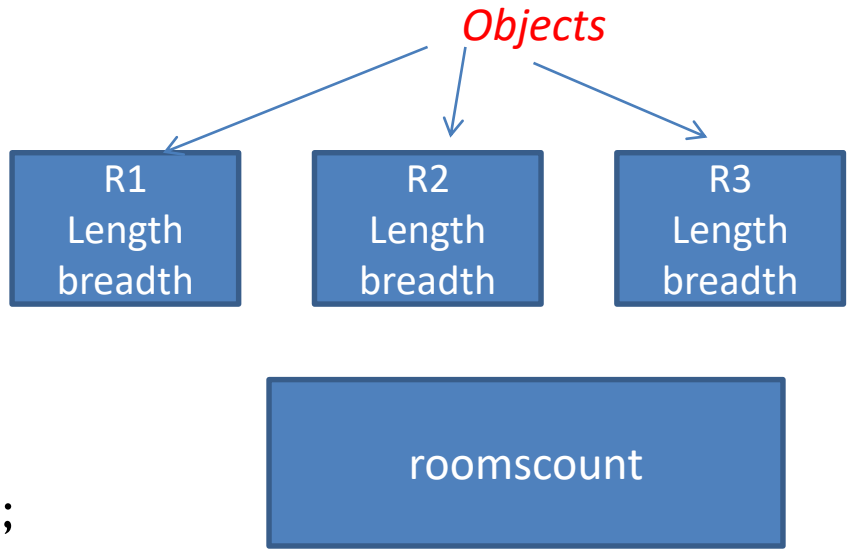



*For example: If I need one common data in class??*

# Static Data Members

```
class room
{
private:
    static int roomscount;
    int length;
    int breadth;
public:
    room(int l, int b)
    {
            length =l;
            breadth=b;
     }
     int area()
     {
            return length*breadth;
      }
};
```

```
int main()
{
    room r1(10,20);
    room r2(20,30);
    room r3(15,20);
    cout<<r3.area();
}
```

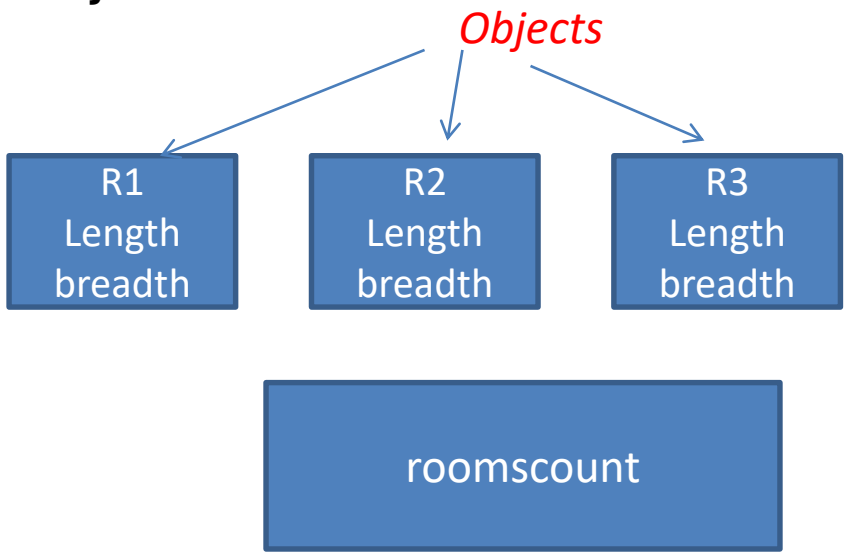



*Next question: where memory is allocated??*

# Static Data Members

```
class room
{
private:
    static int roomscount;
    int length;
    int breadth;
public:
    room(int l, int b);
    {
            length =l;
            breadth=b;
     }
     int area()
     {
            return length*breadth;
      }
};
```

```
int  room::roomscount=0;
int main()
{
     room r1(10,20);
     room r2(20,30);
     room r3(15,20);
     cout<<r3.area();
}
```

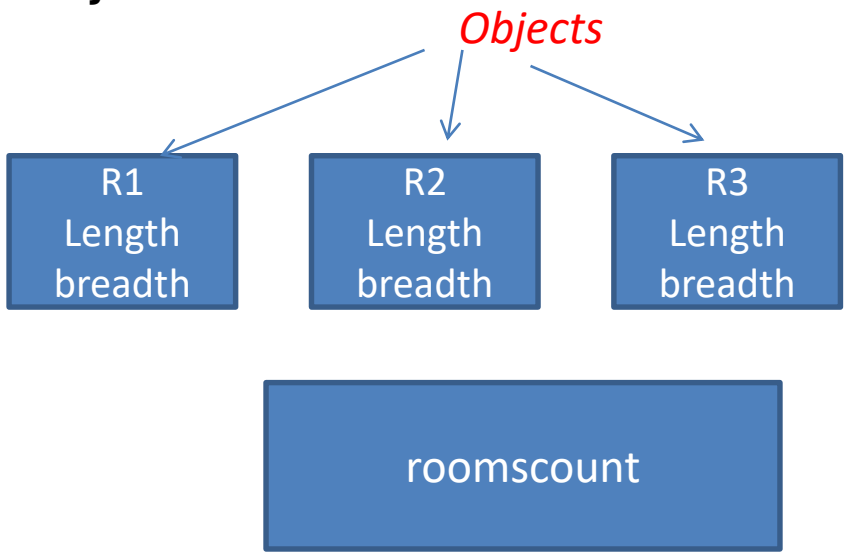



*Next question: how to track the count??*

# Static Data Members

```
class room
{
private:
    static int roomscount;
    int length;
    int breadth;
public:
    room(int l, int b)
    {
            length =l;
            breadth=b;
            roomscount++;
     }
     int area()
     {
            return length*breadth;
      }
};
```

```
int  room::roomscount=0;
int main()
{
     room r1(10,20);
     room r2(20,30);
     room r3(15,20);
     cout<<r3.area();
}
```

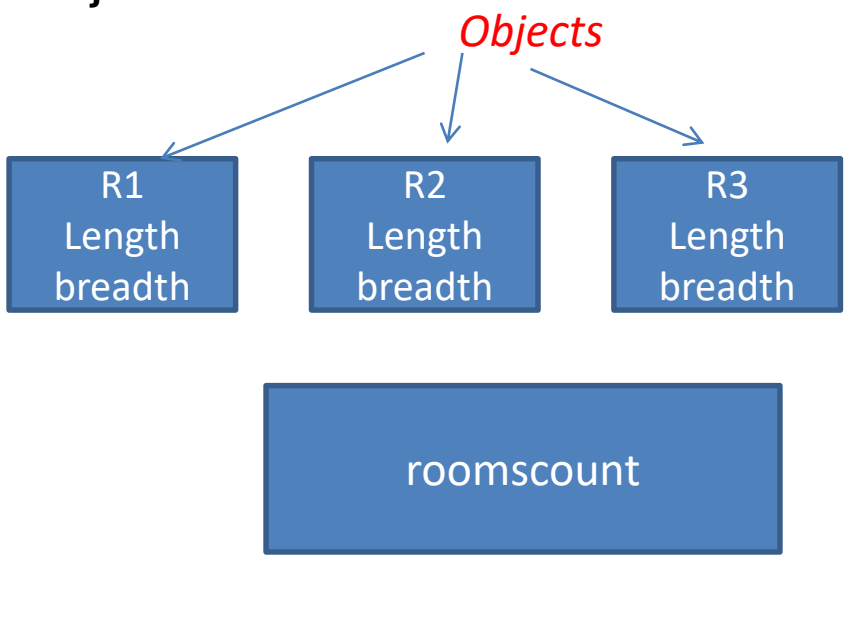



*Next question: how to verify the rooms count??*

# Static Data Members

```
class room
{
private:
    static int roomscount;
    int length;
    int breadth;
public:
    room(int l, int b)
    {
                length =l;
                breadth=b;
                roomscount++;
     }
     int getroomscount()
     {
                return roomscount;
     }
     int area()
     {
                return length*breadth;
      }
};
```

```
int  room::roomscount=0;
int main()
{
      room r1(10,20);
      room r2(20,30);
      room r3(15,20);
      cout<<r3.area();
      cout<< r1.getroomscount();
}
```

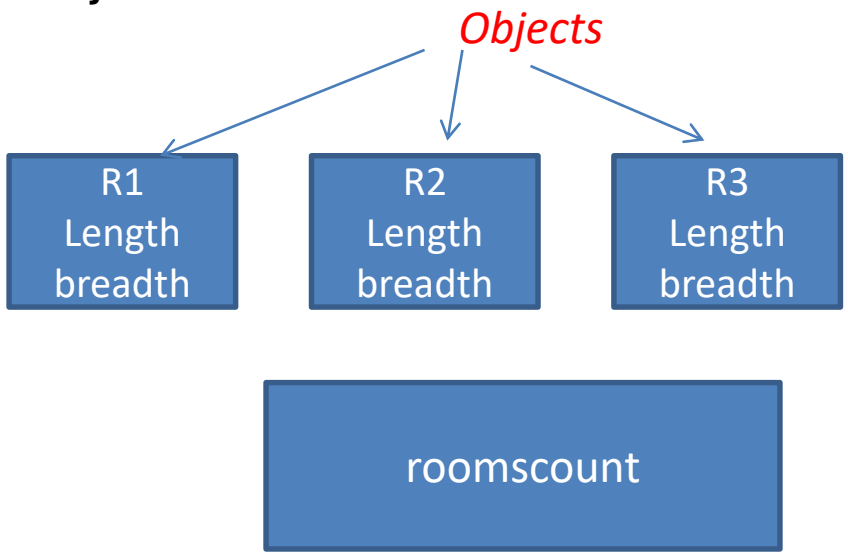



*Next question: is this sensible?? If not what to do??*

# Static Members functions

```
class room
{
private:
    static int roomscount;
    int length;
    int breadth;
public:
    room(int l, int b)
    {
                length =l;
                breadth=b;
                roomscount++;
     }
     static int getroomscount()
     {
                return roomscount;
     }
     int area()
     {
                return length*breadth;
      }
};
```

```
int  room::roomscount=0;
int main()
{
      room r1(10,20);
      room r2(20,30);
      room r3(15,20);
      cout<<r1.area();
      cout<< room::getroomscount();
}
```




**Output:**

**200**

**3**


# Static Member Functions

- Static members functions are defined by using the static keyword. As we have defined these functions as static, they have class properties rather than object properties.
- They can be called even if no object of class be created.
- So we can access static member functions by using the class name and scope resolution operator.
- A static member function can only access static data members and other static member functions from outside the class.
- A static member function is independent of any object of the class.
- A static member function can be called even if no objects of the class exist.
- A static member function can also be accessed using the class name through the scope resolution operator.
- A static member function can access static data members and static member functions inside or outside of the class.
- Static member functions have a scope inside the class and cannot access the current object pointer.
- You can also use a static member function to determine how many objects of the class have been created.

**class_name:: function_name()**