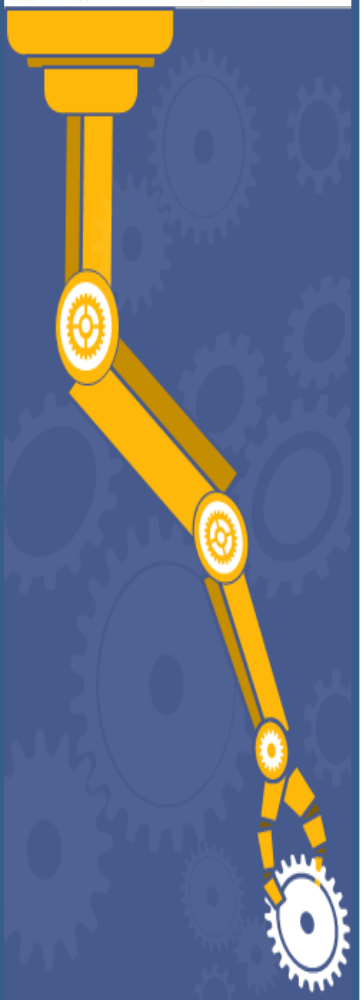
VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)



# 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, Call by reference**
  - **Functions with Default Arguments**
  - **Functions with Objects as Arguments**
  - **Friend Functions and classes**


# Friend Functions & Classes

- Note: Private members cannot be accessed from outside the class. i.e a non-member function cannot have an access to private data of a class.

- **If any situation where we would like two or more classes to share a particular function, what to do??**

- **C++ allows a common function to be made friendly with both the classes, thereby allowing the function to have access to the private data of these classes.**

- **Such a function need not be a member of any other classes.**

# Friend Functions- Declaration

```
class ABC
{
      .....
      .....
  public:
      .....
      .....
      friend void xyz(void);  // declaration
};
```

- To make an outside function friendly to a class, we have to declare this function as friend of the class.
- Function is defined anywhere like normal C++ function.
- Function definition **does not use either keyword friend or scope operator::**
- **Although not a member function, has full access rights to private member of the class.**


# Characteristics of Friend Functions

- A friend function is created by placing the keyword friend in the **function declaration** but not in function definition. Exception is if you declare and define at the same place.
- A friend function is a friend of the class in which it is declared.
- A friend function is not a member function of the class and cannot be called from any object of the class using dot operator.
- A friend function can have full access to the public, private and protected data member of the class to which it is a friend.
- The **arguments of friend functions are usually objects of the class to which it is a friend**.
- A friend function not being a member function of class is called as a normal function.


# Characteristics of Friend Functions

- A friend function can be friend of more than one class.
- A function of one class can be a friend of another class.
- We can have whole class as a friend of another class
- We use friend function usually with multiple classes but can used with single class also.
- A friend function can be declared in the public or private visibility mode without affecting its meaning.


# Syntax- Friend Functions

```
class demo
{
        data members :
        public :
        members functions;
        // friend function declaration
        friend data_type function_name (parameters);
};
data_type function_name (parameters) //definition
{
        function definition;
}
```

**Note : we are not using like this**

```
Void product :: getdata(int a, float b)
{
        number=a;
        cost=b;
}
```


# Example-1 Friend Functions

```
class demo
{
        int y;
        public :
        void input(int x)
        {
            y=x;
        }
friend int findsqr(demo);
};
```

```
int findsqr(demo d)
{
        return d.y * d.y;
}
void main( )
{
        demo F;
        F.input(30);
        cout<<"Square
        is="<<findsqr(F);
}
```

**OUTPUT:**
**Square is = 900**

*Friend function with single class*


# Example-2 Friend Functions

```
class sample
{
        int a,b;
        public :
        void setvalue()
        {
            a=10;
            b=20;
        }
friend float avg(sample s);
};
```

```
float avg(sample s)
{
return float(s.a+s.b)/2;
}
int main( )
{
        sample x;
        x.setvalue();
        cout<<"Average is =”<<avg(x);
        return 0;
}
```

**OUTPUT:**
**Average is = 15**

Friend function with single class

# Example-3 Max of two data of two different classes

```
#include <iostream.h>
class second;// declaration
class first
{
        int fx;
public :
void inputf(int x)
{
        fx=x;
}
friend void findmax(first,second);
};
```

```
class second
{
        int sx;
public :
void inputs(int x)
{
        sx = x;
}
friend void findmax(first,second);
};
```

*friend of both the class first and second*

*Friend function with two different class*


# Example-3 Max of two data of two different classes

```
void findmax(first A, second B)
{
if(A.fx>B.sx)
        cout<<A.fx<<"of class first is greater than
        "<<B.sx<<"of class second<<endl;
else
        cout<<B.sx<<"of class second is greater
        than"<<A.fx<<"of class first<<endl;
}
 void main( )
 {
        first F;
        second S;
        F.inputf(40);
        S.inputs(70);
        findmax(F,S);
 }
```

*Output:*
*70 of class second is greater than 40 of class first*


# Example-4: Function of one class- friend of another class

```
//MF of one class as a FRIEND of another class

#include<iostream>
using namespace std;
class sample;
class sam
{
   public:
   void disp();
};
class sample
{
   int a;
public:
   int b;
   sample()
   {
      a=10;
      b=20;
   }
   friend void sam::disp();
};

void sam::disp()
   {
      sample s;
      cout<<s.a<<s.b;
   }


int main()
{
   sam s;
   s.disp();

}
```


## Example-5: Function of one class- friend of another class

```
#include <iostream.h>
class second;
class first
{
        int num;
public :
void input_first( )
{
        num=20;
}
void show(second);
};
```

*Created function in class first which will be friend of class second*

```
class second
{
int num;
        public :
void input_second(int x)
{
        num==x;
}
friend void first : :show(second);
};
```

*As function will be friend of class we can have object of **class second** in the function of the first class*


# Example-5: Function of one class- friend of another class

```
void first : :show(second s)
{
cout<<"NUM OF CLASS FIRST :="<<num<<endl;
s.input_second(num*num);
cout<<"NUM OF CLASS SECOND :="<<s.num<<endl;
}

 void main( )
 {
                first f;
                f.input_first( );
                second s;
                f.show(s);
 }
```

*Output:*
*NUM OF CLASS FIRST :=20*
*NUM OF CLASS SECOND :=400*

- *As function show is friend of class second, private members functions of second class can be used in show function only through objects using dot operator.*
- *But as the function show is of class first, private data members of class first can be used directly. In the main, the function is called using an object of class first and passes an object of class second.*
- *In the function num* num (num is of class first) is assigned to num of class second object using a call to function input_second.*


## Example-6:Whole class as a friend of another class

```
#include <iostream.h>
class second;
class first
{
public :
void output( )
{
cout<<"FIRSTCLASS "<<endl;
}
friend class second;
};

class second
{
public :
void show(first s)
{
s.output( );
}
};

void main( )
{
        second s;
        first f;
        s.show(f);
}
```

*Output:First class*


# Example-7: Matrix Addition with Friend Functions

```
//A normal function as Friend Function of one class
#include<iostream>
using namespace std;
class matrix
{
   int row,col;
   int ar[5][5];

public:
   void getmatrix(int,int);
   void display();
   friend void addmatrix(matrix &, matrix &, matrix &);
};
                                        void addmatrix(matrix &n1,matrix &n2, matrix &res)
                                        {
                                           cout<<"\nADDITION\n";
                                           res.row=n1.row;
                                           res.col=n1.col;
                                           for (int i=0;i<res.row;i++)
                                           {
                                              for(int j=0;j<res.col;j++)
                                              {
                                                 res.ar[i][j]= n1.ar[i][j]+n2.ar[i][j];
                                                 cout<<res.ar[i][j];
                                              }    }
                                        }
```


# Example-7: Matrix Addition with Friend Functions

```
void matrix::getmatrix(int r,int c)
{
   row= r;
   col=c;
   cout<<"\nMatrix Input:";
   for (int i=0;i<r;i++)
      for(int j=0;j<c;j++)
         cin>> ar[i][j];
}
void matrix::display()
{
   for (int i=0;i<row;i++)
   {
      cout<<"\n";
      for(int j=0;j<col;j++)
      {
        cout<< ar[i][j]<<"\t";
      }

   }
}
```

```
int main()
{
   matrix m1, m2;
   m1.getmatrix(2,2);
   m1.display();
   m2.getmatrix(2,2);
   m2.display();
   matrix m3;
   addmatrix(m1,m2,m3);  //friend function
   cout<<"Result:";
   m3.display();
}
```

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

# Thank You