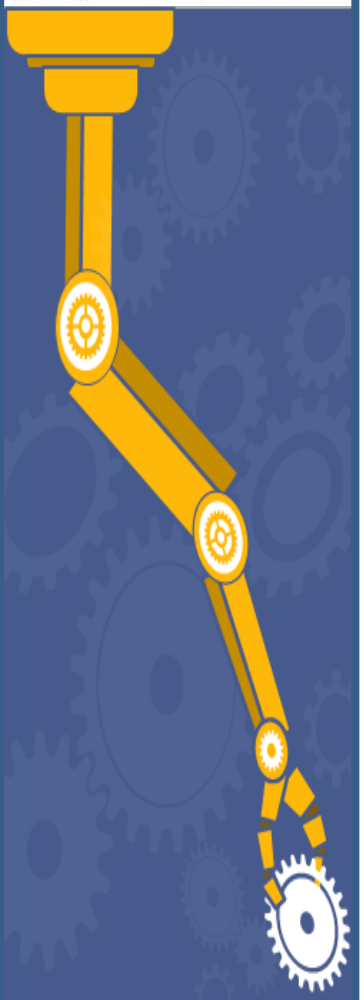
VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)



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.

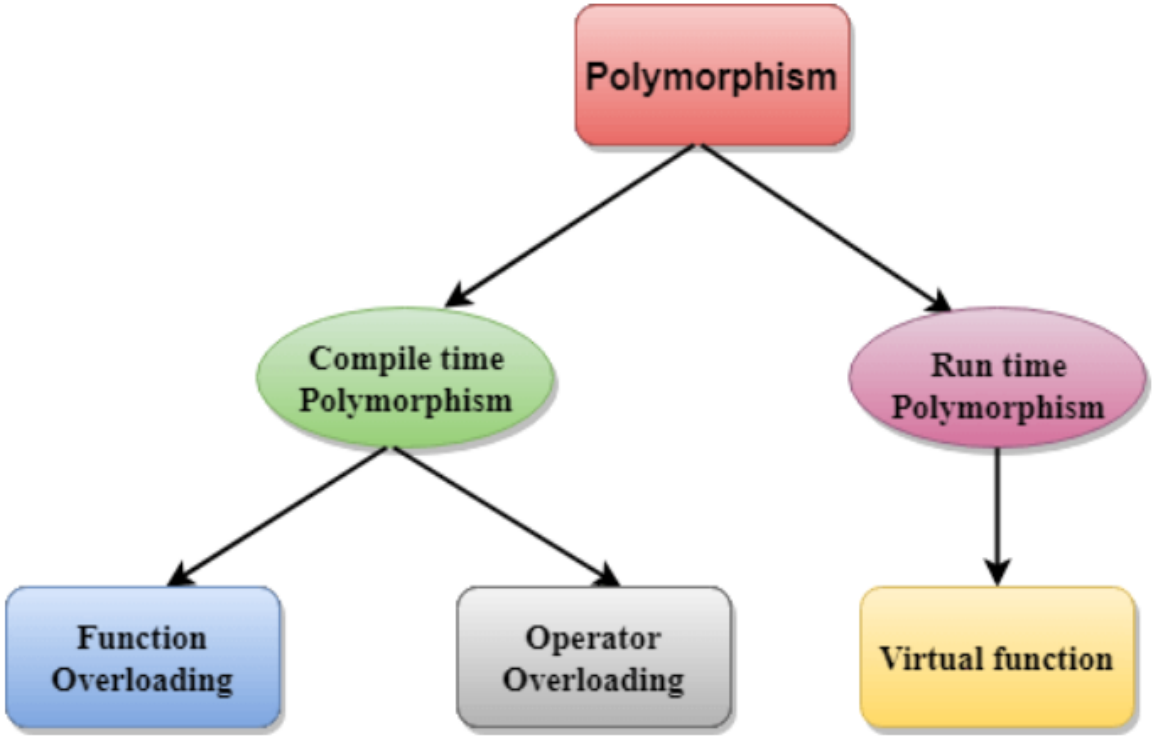
Text Book(s)	
1.	Robert C. Seacord, Effective C: An Introduction to Professional C Programming, 1st Edition, No Starch Press, 2020.
Reference Book(s)	
1.	Vardan Grigoryan and Shunguang Wu, Expert C++: Become a proficient programmer by learning coding best practices with C++17 and C++20's latest features,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

BCSE102L- Structured and Object-Oriented Programming

- **Module-7: POLYMORPHISM**
  - **Function Overloading**
  - **Operator Overloading**
  - **Dynamic Polymorphism**
  - **Virtual functions**
  - **Pure Virtual Functions**
  - **Abstract Classes**

Polymorphism

- "Polymorphism" is the combination of "poly" + "morphs" which means many forms. Ability to take more than one forms.



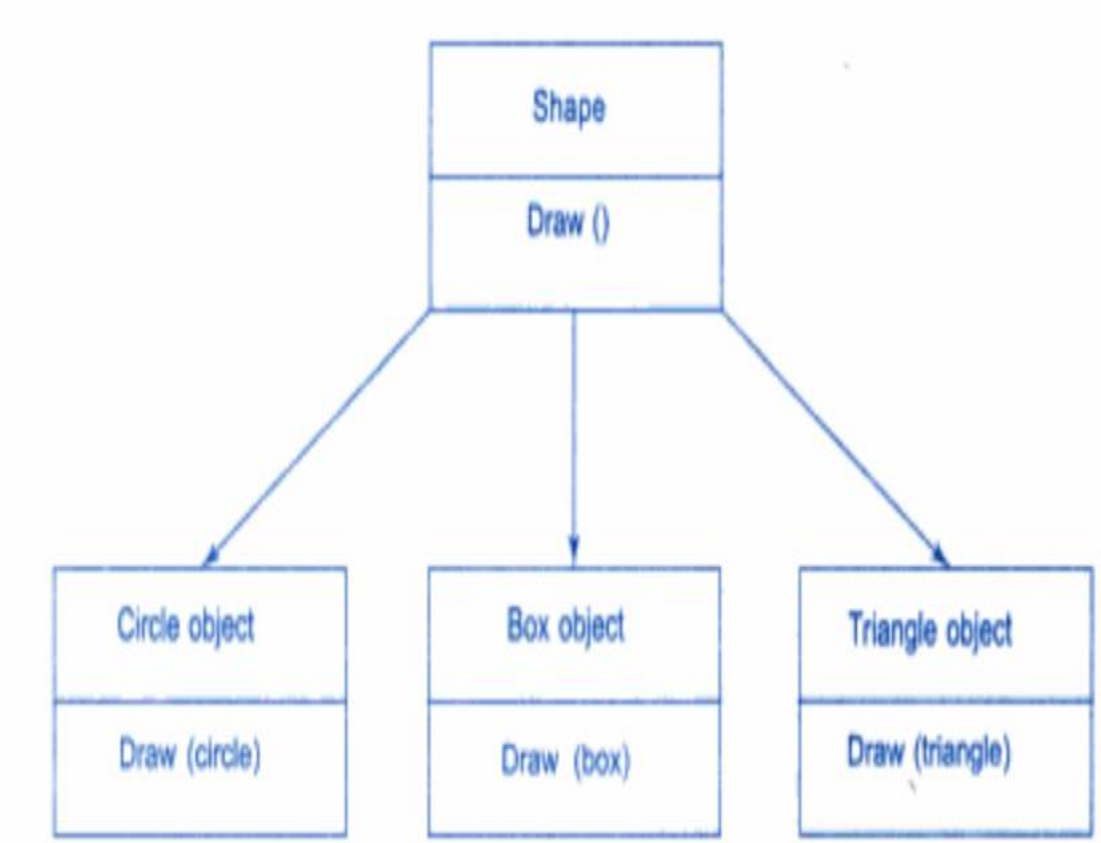
Compile time polymorphism

- Overloaded functions are invoked by matching the type and number of arguments.
- This information is available at the compile time and, therefore, compiler selects the appropriate function at the compile time.
- It is achieved by **function overloading and operator overloading** which is also known as static binding or early binding.

Run time polymorphism

- Run time polymorphism is achieved when the object's method is invoked at the run time instead of compile time.
- It is achieved by method overriding which is also known as dynamic binding or late binding.

Dynamic Binding



Dynamic Binding

- Binding refers to linking of a procedure call to the code to be executed in response to the call.
- Associated with polymorphism and Inheritance.

- For example: **The procedure " draw" in the previous slide, every object will have this method/procedure.**
- Its algorithm is unique with respect to each object and it will be redefined in each class that defines the object.
- At runtime, the code matching the object under current reference will be called.

Compile time polymorphism	Run time polymorphism
The function to be invoked is known at the compile time.	The function to be invoked is known at the run time.
It is also known as overloading, early binding and static binding.	It is also known as overriding, Dynamic binding and late binding.
Overloading is a compile time polymorphism where more than one method is having the same name but with the different number of parameters or the type of the parameters.	Overriding is a run time polymorphism where more than one method is having the same name, number of parameters and the type of the parameters.
It is achieved by function overloading and operator overloading.	It is achieved by virtual functions and pointers.
It provides fast execution as it is known at the compile time.	It provides slow execution as it is known at the run time.
It is less flexible as mainly all the things execute at the compile time.	It is more flexible as all the things execute at the run time.

Function Overloading

- Overloading refers to use of **same thing for different purpose.**
- Function overloading - Creating numbers of functions with same name which performs different tasks.
- Function overloading relieves us from remembering so many functions names with type of arguments they take.
- Create number of functions with the same name but either number of arguments or type of arguments must be different.

Function Overloading- Examples

```
int sum (int);
float sum(float);
double sum(double);
```

Three functions with the same name sum. Each function takes just one parameter but all are of different types. That is number of parameters in all overloaded function sum is same but type of parameter is different.

```
void show(int,int);
Void show(int);
Void show(int,int,int);
```

Number of parameters are not same in show functions but type is same.

```
void show(int,char);
void show (char,int,float);
void show(int);
int show(int,int);
float show(char,char,char);
```

Here we have a mix of overloaded show functions. Some have same number of arguments but type is different and some have same type of argument but numbers of argument are different.

Function Overloading

- When we have number of overloaded functions in a program, which function to call is determined by either **checking type of argument or number of argument.**
- **Note:** "return type" does not play any role in function overloading as which function to call is determined by checking type and number of argument a function accepts.
- When control is transferred to function and function is about to return after execution then return type comes to play.

Function Overloading

- In function overloading compiler first tries to find an exact match. If exact match is not find then **integral promotion/ demotion is used.**

Function Overloading - Example-1

```
void main( )
{
void show(int);
void show(float);
void show(char);
void show(char*);
int x=10;
float y=23.45;
char ch= 'p';
char * s="overload";
show(x);
show(y);
show(ch);
show(s);
}
```

```
void show(int x)
{
cout<<"int show x="<<x<<endl;
}
void show(float y)
{
cout<<"float show y="<<y<<endl;
}
void show(char ch)
{
cout<<"char show ch="<<ch<<endl;
}
void show(char*s)
{
cout<<"char *s show s="<<s<<endl;
}
```

Function Overloading - Example-11

```
void main( )
{
void show(int);
void show(double);
show(23);
show('p');
show(2.5f);
show(3.45);
}
```

// Integral Promotion

```
void show(int x)
{
cout<<"int show x="<<x<<endl;
}
void show(double s)
{
cout<<"double show s="<<s<<endl;
}
```

OUTPUT:

int show x=23

int show x=112

double show x=2.5

double show x=3.45

Function Overloading - Example-III

```
void main( )
{
void show(float);
void show(double);
show(23);
show(2.5f);
show(3.45);
}
```

Whether to call float function or double function??

```
void show(float x)
{
cout<<"float show x="<<x<<endl;
}
void show(double s)
{
cout<<"double show s="<<s<<endl;
}
```

OUTPUT:

ERROR

'show' : ambiguous call to overloaded function

Function Overloading - Example-IV

```
void main( )
{
void show(int);
void show(float);
show('p');
show(3.45);
}
```

```
void show(float x)
{
cout<<"float show x="<<x<<endl;
}
void show(int s)
{
cout<<"int show s="<<s<<endl;
}
```

Int or float version can be called- Integral Demotion

OUTPUT:

ERROR

'show' : ambiguous call to overloaded function

Function Overloading - Example-V

```
void main( )
{
void show(int);
show(5.47f);
show(3.45);
}
```

```
void show(int s)
{
cout<<"int show s="<<s<<endl;
}
```

// Integral demotion- from float and double to int

OUTPUT:

int show s=5

int show s=3

Function Overloading - Example-VI

```
void main( )
{
int x,y, intmax;
float f1,f2,fmax;
char ch1,ch2,chmax;
int max2(int,int);
float max2(float,float);
char max2(char,char);
cout<<"Enter two integers";
cin>>x>>y;
cout<<"Enter two floats";
cin>>f1>>f2;
cout<<"Enter two chars";
cin>>ch1>>ch1;
```

Function Overloading - Example-VI

```
intmax=max2(x,y);
fmax=max2(f1,f2);
chmax=max2(ch1,ch2);
cout<<"Max of two int:"<<intmax<<endl;
cout<<"Max of two float is "<<fmax<<endl;
cout<<"Max of two char is"<<chmax<<endl;
}
```

Function Overloading - Example-VI

```
int max2(int x,int y)
{
    return(x>y ?x :y);
}
float max2(float x,float y)
{
    return(x>y ?x :y);
}
char max2(char x,char y)
{
    return(x>y ?x :y);
}
```

OUTPUT:

```
Enter two integers
123     567
Enter two floats
12.34     56.78
Enter two chars
a     g
Max of two int is 567
Max of two float is 56.78
Max of two char is g
```

Function Overloading - Example-VII

```
class demo
{
    private:
        int i = 5;        double d = 6.2;
    public:
        void add(int x)
        {
            cout << "Value : " << i + x << endl;// 15
        }
       void add(double y)
        {
            cout << "Value : " << d + y << endl;// 21.7
        }
```

Function Overloading - Example-VII

```
// Differing in the number of arguments.
        void add(int a, int b)
        {
           cout << "Value :" << i +a+ b << endl;//30
        }
};
int main()
{
    demo t1;
    t1.add(10);
    t1.add(15.5);
    t1.add(10,15);
    return 0;
}
```