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-8: GENERIC PROGRAMMING**
  - **Function Template**
  - **Class Template**
  - **Standard Template Library**

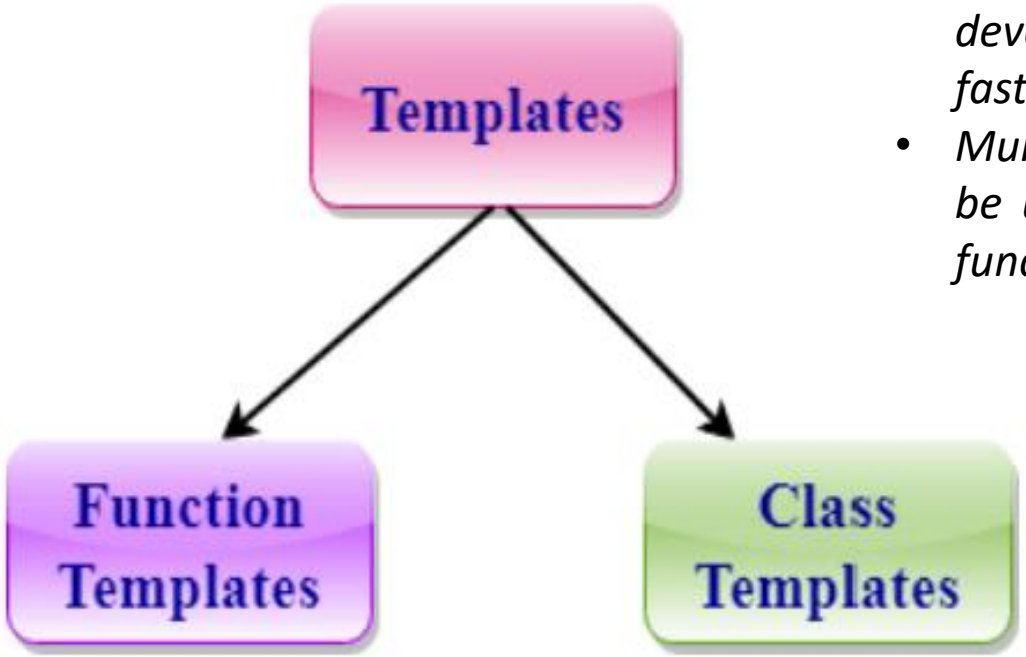
Generic Programming

- Templates is one of the features which is added in C++ to enable us to define generic classes and functions and thus provides support generic programming.
- **Generic programming** is an approach where generic types are used as parameters in algorithms so that they work for a variety of suitable data types and data structures.
- Instead of **specifying the actual data type**, we supply a **placeholder in templates**, and that placeholder is substituted by the data type that was utilized during compilation.
- A template can be used to create a family of classes or family functions to handle different data types.

Generic Programming

Templates can be represented in two ways:

- Function templates
- Class templates



- *Template classes and functions eliminate the code duplication of different data types and thus makes the development easier and faster.*
- *Multiple parameters can be used in both class and function template.*

Function Templates

- Function templates are special functions that can operate with generic types. This allows us to create a function template whose functionality can be adapted to more than one type or class without repeating the entire code for each type.
- **In other words, We can define a template for a function.**
- For example, if we have an add() function, we can create versions of the add function for adding the int, float or double type values.
- C++ routines work on specific types. We often need to write different routines to perform the same operation on different data types.

Case 1:

```
int maximum(int a, int b, int c)
{
        int max = a;
        if (b > max) max = b;
        if (c > max) max = c;
        return max;
}
```

Case 2:

```
float maximum(float a, float b, float c)
{
        float max = a;
        if (b > max) max = b;
        if (c > max) max = c;
        return max;
}
```

```
double maximum(double a, double b, double c)
{
        double max = a;
        if (b > max) max = b;
        if (c > max) max = c;
        return max;
}
```

Function Templates

- In all the cases, logic is exactly the same, but the data type is different.
- Function templates allow the logic to be written once and used for all data types – generic function.
- **SYNTAX:**

```
template <class type>
returntype functionname(parameter list/arguments of type T)
{
        // body of function
        // with type T
        // wherever appropriate
}
```

Function Templates

- Generic function to find a maximum value

```
template <class T>
T maximum(T a,T b,T c)
{
        T max = a;
        if (b > max) max = b;
        if (c > max) max = c;
        return max;
}
```

Function Templates- How it works??

```
template <typename T>
T myMax(T x, T y)
{
   return (x > y)? x: y;
}

int main()
{
   cout << myMax<int>(3, 7) << endl;
   cout << myMax<char>('g', 'e') << endl;
   return 0;
}
```

Compiler internally generates and adds below code

```
int myMax(int x, int y)
{
   return (x > y)? x: y;
}
```

Compiler internally generates and adds below code.

```
char myMax(char x, char y)
{
   return (x > y)? x: y;
}
```

Function Templates- Bubble sort

```
template <class T>
Void bubble(T a[], int n)
{
        for (int i=0;i<n-1;i++)
        {
                for(int j=n-1;i<j;j--)
                {
                        if(a[j]<a[j-1])
                        {
                                swap(a[j],a[j-1]);
                        }
                }
        }

}
```

Function Templates- Bubble sort

```
template <class X>
Void swap(X &a, X &b)
{
        X temp=a;
        a=b;
        b=temp;
}
```

Function Templates- Bubble sort

```
int main()
{
        int x[5]={15,25,12,10,20};
        float y[5]={ 1.1,5.5,3.3,2.2, 4.4};
        bubble(x,5);// calling template function for int values
        bubble(y,5);// calling template function for float values
        for (int i=0;i<5;i++)
        {
                cout<<x[i]<<endl;
        }
        for (int j=0;j<5;j++)
        {
                cout<<y[j]<<endl;
        }
        return 0;
}
```

Function Templates- Overloading

```
using namespace std;
template<class X>
void fun(X a)
{
    cout << "Value of a is : " <<a<< endl;
}
template<class X,class Y>
void fun(X b ,Y c)
{
    cout << "Value of b is : " <<b<< endl;
    cout << "Value of c is : " <<c<< endl;
}
```

```
int main()
{
   fun(10);
   fun(20,30.5);
   return 0;
}
```

OUTPUT:
Value of a is : 10
Value of b is : 20
Value of c is : 30.5