**Programming in C++**

**Program Duration**:18 days.

**Contents**:

* Introduction to Object Oriented Programming
  + Why Object-oriented Programming?
  + What are the limitations of procedural programming like C?
  + Understand basic concepts of OOP – Encapsulation, Abstraction, Polymorphism, Inheritance
* C++ Basic Constructs
  + Why C++?
  + C++ Basic Constructs – datatypes, class, object, constructor, destructor, this pointer, access specifier, iostreams, namespace, reference, struct etc.
  + Develop an application using basic constructs to create objects, use member data and functions to perform operations on it.
* Constructor and Destructors
  + Why constructor?
  + Constructor Basics – types (default, overloaded, copy and explicit), reference in copy constructor.
  + Why destructor?
  + Develop an application using constructors of different types to create objects .
  + ***Coding Guidelines***:
    - *Need to define a default and overloaded constructor.*
    - *Use initializer list to initialize the class members.*
    - *Should implement copy constructor to perform deep copy in case of classes with pointer members.*
    - *Copy constructor must use call-by-reference not call-by-value.*
    - *Free allocated memory in destructor.*
    - *A member function that does not affect the state of an object (its instance variables) is to be declared const.*
    - *Constructors and destructors must not be inline.*
* Variable Scope, Functions, Static members
  + - Understand variable scope – local, static, extern etc.
    - Understand the concepts of function – function, pass by value, pass by reference, pass by address, recursion, inline function and function chaining.
    - Static function vs. non-static/member function
    - Understand the concepts of static function – static members, static objects, initialization, static accessors.
    - Develop an application using static variables, static accessors and function using reference arguments to perform operations.
* Overloading
  + Why overloading?
  + Understand the concept of overloading - function overloading, function with default arguments, operator overloading (unary, binary), operator overloading restrictions.
  + Develop an application using overloaded functions (operator and function) to perform given operations.
  + Dynamic Memory Management
  + Why dynamic memory management?
  + Understand the concept of dynamic memory management – allocation, deallocation, memory leak, overloading new and delete.
  + Usage of relevant calls – new(), delete(), operator new().
  + Develop an application using new(), delete() and operator new() to manage dynamic memory.
  + ***Coding Guidelines***:
    - * *Do not use malloc and free.*
      * *Use delete [] to free arrays.*
      * *Set pointer to 0 after deleting the object.*
      * *Check return value after new or handle exception.*
  + Friend
  + Why friend?
  + Friend class, friend function, operator overloading using friend function.
  + Develop an application using friend function and friend class to access private members and to perform given operations.
  + ***Coding Guidelines***:
    - *Follow the below order to place data members.*

*Friend->public->protected->private*

* + - *Avoid circular dependency in case of multiple class reference by just adding forward class reference instead of #include for .h files.*
    - *Friend functions cannot use this pointer.*
    - *Friend functions is not inherited.*
    - *Use friend function to overload stream operators*
* Inheritance
  + Why to inherit?
  + Understand inheritance basic concepts – inheritance, is a kind of, types (based on access mode specifier (public, protected, private), based on number of classes (single, multiple, multi-level, hierarchical, hybrid)), constructor and destructor call sequence.
  + Composition vs. inheritance
  + What is diamond issue?
  + Design and develop an application designed using inheritance to create objects of different class type, access members and perform operations.
  + ***Coding Guidelines***:
    - *Do not expose data members by specifying as public or protected, rather expose them via accessor methods only to avoid minimal change in derived class.*
    - *All classes that are used as base classes and which have virtual functions, must define a virtual destructor.*
    - *The destructor should be virtual if and only if the class contains virtual methods.*
    - *Avoid defining a virtual function as friend.*
    - *Virtual function cannot be a static member.*
* Polymorphism
  + Understand basic concepts of runtime polymorphism – virtual & pure virtual functions, virtual function access (using vtable, vptr), access of derived class objects using base class pointers, cast operators, etc.
  + Why virtual destructor?
  + What is an abstract class?
  + Develop an application to use virtual functions of different classes and to manage operations on the derived class objects.
  + Analyze and fix the type cast conversion specific issues in code.
  + ***Coding Guidelines***:
    - *Do not call pure virtual function in constructor of an abstract base class.*
    - *Don’t use an abstract class as an argument type, as a function return type, or as the type of an explicit conversion.*
* Exception Handling
  + Why to handle exception?
  + Understand exception handling basic concepts – raising and handling specific and generic exception, handling uncaught exception, types (standard, runtime etc.), user defined exception, stack unwinding, nested try..catch, exception rethrow, etc.
  + Understand the usage of relevant calls – try….catch(), catch(..), std::bad\_alloc()….
  + Analyze the given code for exception issue(s).
  + Develop an application using relevant calls to handle exceptions and perform operations.
  + ***Coding Guidelines***:
    - *Be careful throwing exceptions in Destructors since any destructive behaviour that has already taken place probably cannot be reversed.*
    - *Handle specific exceptions and then handle generic exceptions.*
* I/O Streams, File I/O Handling
  + Understand the basics of streams – default stream objects (cin, cerr, clog), file streams (ofstream, fstream, ifstream), file I/O handling and exception.
  + Understand usage of relevant calls – open (), close(), get(), read(), …
  + Develop an application using relevant file I/O calls to manage I/O operations.
* Class information Handling
  + Why namespace?
  + Understand basics about namespace and RunTime Type Information (RTTI) – name collision, anonymous namespace, runtime type information, check data type.
  + Understand the usage of relevant keywords and functions.
  + Develop an application using namespace to manage name collisions and to compare the type using RTTI.
* Generic Programming
  + Why templates?
  + Understand the basic concepts of Generic Programming – function & class template, function template with specialization, applications.
  + Usage of function template, specialization, and class template.
  + Develop an application using function template with specialization and class template to implement the given generic programming requirements.
* Standard Template Library
  + Why STL?
  + Understand the basic concepts of STL – Components (Containers, Iterators, Algorithms, Function Objects), Container Types (sequence, adaptor, associative, unordered) and Functions (vector, list, map, queue etc.), benefits.
  + Understand the usage of relevant algorithms and container functions.
  + Develop an application using appropriate STL container(s) and algorithm(s) to perform given operations.
* Modern C++
  + Overview of Modern C++ - benefits, C++ Standards
  + C++11 Standard – Features, overview, usage of important features (range based for loop, initializer list, Lambda expression, auto type deduction, move semantics and rvalue references, STL algorithms (all\_of, any\_of, none\_of, copy\_n and iota).
* Smart Pointers and Dynamic Memory Management
  + Why Smart pointers?
  + Understand the basics of smart pointers – types (auto\_ptr, unique\_ptr, shared\_ptr, weak\_ptr), memory management, application
  + Usage of relevant smart pointer calls
  + Develop an application using appropriate smart pointer to manage dynamic memory.
* Miscellaneous
  + Handling of Mixed C and C++ code - extern “C”, usage of #ifdef \_\_cplusplus….#endif
  + Develop an application using mixed code (C and C++).
  + ***Coding Guidelines***:
    - *While creating or while calling C functions in C++ use extern "C".*
    - *Include C header files in extern “C”{} block.*
* Design Principles and Design Patterns
  + Understand the concepts of OOPS Design – Design principles (SOLID Design), Coupling and Cohesion, Aggregation vs. Composition etc.
  + Why do we need design pattern?
  + Understand basics of Design Patterns – types (creational, structural, behavioural), commonly used design patterns, applications.
  + Usage of commonly used design pattern – Creational (Factory, Singleton), Structural (Adapter, Composite), Behavioural (Observer).
  + Analyse a given code using design pattern, identify the impact of changes in one or more class(es), extend the current design add support for a new design pattern etc.
* UML Modelling
  + Understand the basics of UML Modelling – UML building blocks (Things, Relationship, Diagrams), UML Diagrams (Usecase, Class, Activity, Sequence etc.)
  + Develop a UML Model of an application using usecase, class and sequence diagrams for the given requirements.
* Concurrency in CPP
* Multithreaded Programming Basics – concept of thread, Thread Vs Process, thread attributes, shared resources.
* Thread Management using C++11 Thread Library - Thread creation mechanisms (Function callback, Function objects, Lambda Function).
* Develop a multithreaded application using C++ 11 Thread Library.
* Thread Synchronization Basics – understand need for synchronization, race condition, critical section, synchronization mechanisms (mutex, semaphore, lock\_guard etc.), RAII.
* Develop a multithreaded application with synchronized updates to global variable.
* Debug a given multithreaded application(s) and fix issues (crash, memory leak etc) in it using tools (gdb, valgrind).
* Take care of thread and mutex lock specific coding guidelines.
* ***Coding Guidelines***:
  + *Parent should wait for all joinable child threads to exit and then exit.*
  + *Do not join an already joined thread as it will terminate program.*
  + *Do not pass stack variable as thread parameter rather allocate and pass a pointer to heap block.*
  + *Do not return variable in stack, rather use static variable, or return a pointer to heap block.*
  + *Do not rely on thread output sequence.*
  + *Hold lock for very short duration and release lock after use.*
  + *Do not attempt lock on an already acquired lock.*
  + *Use std::lock\_guard for RAII support.*

**Data Structure and Algorithms**

**Program Duration**: 7days.

**Contents**:

* Introduction to Data Structure
  + Why do we need data structures?
  + Understand the concept of Data Structure - Types (Linear, Nonlinear), Access mechanisms and operations.
  + What is ADT?
* List
* Why List?
* Understand the concept of List – implementation, operations, applications
* Develop an application using STL list container and perform operations to process data.
* Take care of stack specific coding guidelines.
* Stack
* Why Stack?
* Understand the concept of Stack – implementation, operations, applications
* Develop an application using STL stack container and perform operations to process data.
* Take care of stack specific coding guidelines.
* ***Coding Guidelines***:
  + - *Check for overflow (stack full) and underflow (stack empty) cases and handle them.*
* Queue (Concept)
* Why Queue?
* Stack vs. Queue
* Understand concept of Queue basics – implementation using array/list, operations, applications, circular queue, priority queue.
* Take care of queue specific coding guidelines.
* ***Coding Guidelines***:
  + - *Check for queue empty and queue full conditions and handle them.*
* Tree (Concept)
* Why Tree?
* Understand the concept of Tree – types (AVL, BST...), height, degree, depth, balanced/unbalanced, operations (insert, search, update, delete), traversal mechanisms (pre-order, post-order and in-order), applications etc.
* Hash
* Why Hash?
* Understand the concept of Hash – hashing, operations (insert, delete, search), collision handling mechanisms and application.
* Develop an application using STL map container and perform operations to process data.
* Algorithm Analysis and Selection
* Why Data Structure Algorithms?
* Algorithm Evaluation Basics

- Time & space complexity

- Big-O Notation

* Analyze and select data structure algorithm as per requirement.
* Searching Algorithms
* Why Searching Algorithms?
* Types (Linear, Binary), time complexity, application.
* Develop an application using Binary Search STL algorithm to process data.
* Sorting Algorithms
* Why Sorting Algorithms?
* Types (Quick Sort, Merge Sort, Heap Sort etc), time complexity, application.
* Develop an application using STL sort algorithm to sort the data.