SE001CPP: *C++ Basic*

DURATION: 5 Days

WHAT YOU WILL LEARN
This five-day instructor-led course provides to teach the basic of C/C++ language

The approach used is theory followed by powerful seriously designed exercises to reinforce the understanding of the theory.

Besides the language syntax, many aspect of the course are intended to make the participants as better software developer, especially using C++ as programming tool. Many of the aspects covered in this course will benefit the participants in many other project developments.

This course does not cover the OOP aspect of C++ in detail. It is focusing on most of the language construct that can be used for both C and C++

Upon completion of this course, participants should be able to:

- Apply different programming constructs in C++ appropriately.
- Solve moderate size problems using C++.
- Understand the how to work in a team in software development with C++.
- To use C++ as better C.
- Use IDE to create a moderate size of project.

AUDIENCE
This course is for programmers who intend to learn C++.

PREREQUISITES
Before attending this course, students must have:

- Basic understanding of operating system.
- Basic knowledge of computer hardware.
- Basic programming concept will be added advantage.

COURSE OUTLINES

Module 1: Introducing to the IDE
This module explains how to use the Integrated Development environment for developing C++ program/project
Lessons

- The project templates
- The editor
- The compiler
- The main() function
- Different type of programming errors
- Using debugger
- How to get help
- Running the program

Lab: *Using IDE features*

- Exercise 1: *GCD solution*
- Exercise 2: *How to use debugger*
- Exercise 3: *Using help from the IDE*

After completing this module, students will be able to:

- Use the IDE to create a simple C/C++ program
- Use Debugger to find the programming mistakes
- Run programs from the IDE
- Seeking help from the IDE

Module 2: C/C++ fundamentals
This module will brief students the fundamental features of C/C++ language constructs
Lessons

- Identifier and the rules in naming
- Single line and block comments
- Command line arguments
- The return value from main() function
- Functions as building block
- Naming Convention
- Sequential Statements
- Simple I/O with cin and cout classes
- Preprocessing
    - Macro constant
    - Macro Functions
- **const** Vs. macro constants

Lab: *Basic language constructs*

- Exercise 1: *GCD solution with input from command line arguments*
- Exercise 2: *GCD solution with user input*
- Exercise 3: *Using macro constant or **const** to minimize changes*
- Exercise 4: *Using macro function*

After completing this module, students will be able to:

- Utilize arguments to pass values from command line
- Get input from user during running time instead of hardcode the values in programs.
- Apply macro constant to minimize hard coding and increase flexibility of coding
- Differentiate **const** and macro constant and apply them appropriately

Module 3: Conditional statements
This module will cover various conditional statements in C/C++
Lessons

- Conditions, Composite conditions, and Boolean logic
- Zero is false, none zero is true
- "Short circuit" in composite conditions
- Branching Statement

- Unconditional **goto** Statement
- **if**-Then, **if**-Then-**else**, and Nested If
- **switch..case..default** Statement

Lab: *Understand conditions*

- Exercise 1: *Conditional AND, OR, XOR and NOT*
- Exercise 2: *Short Circuit*
- Exercise 3: *Fall through in switch case statement*

After completing this module, students will be able to:

- Understand various conditional logics
- Avoid problem caused by the short circuit in composite conditions
- Provide code readability in the switch case statement.

Module 4: *Iteration Statements*

This module explains various types of iteration statements in C/C++

Lessons

- **do-while** statement
- **while–do** statement
- **for** statement
- **goto** statement
- **break** statement in iteration
- **continue** statement in iteration
- Busy loop
- The principal of "*One in one out*"

Lab: *Understand iteration statements*

- Exercise 1: *Iteration with goto statement*
- Exercise 2: *Busy loop and break statement*
- Exercise 3: *Using continue statement*

After completing this module, students will be able to:

- Understand various conditional and unconditional iteration statement
- Preserve good practice using the principal of "One in one out"
- Apply busy loop adequately.

Module 5: *Data Types*

This module explains various data types in C/C++.

Lessons

- Data Types Concept
  - Number of bytes needed for representing the data
  - Value range represented
  - Acceptable Value types
  - Valid operations on the data
- Simple/native data types
- The **sizeof** keyword
- Variables
  - Variable naming: Pascal and camel styles
  - Declaration and Scoping
  - Shadowing and Scope resolution operator
- Type casting
- Bit Fields
- **Union**
- New types in C++

Lab: Understand data types

- Exercise 1: *Apply appropriate data types*
- Exercise 2: *Using **sizeof** keyword*
- Exercise 3: *Variable shadowing and scope resolution operator*
- Exercise 4: *Type casting*
- Exercise 5: *Union and bit fields*

After completing this module, students will be able to:

- Select adequate data types to represent the data
- Make use of **sizeof** in writing better code
- Avoid the shadowing issue and resolve it using scope resolution operator
- Understand the meaning of type casting and its consequences
- Use union and bit fields

Module 6: Storage class

This module explains the meaning of storage class, and relates it to the variable/object life cycle.

Lessons

- Data segment, Stack, and Heap
- **auto**, **static**, **extern** and **register** keywords
- Constraint on the register variables
- Register parameters
- Variable/Object Life Cycle

Lab: Understand storage classes

- Exercise 1: *Automatic variables*
- Exercise 2: *Register variables/parameters*
- Exercise 3: *Hiding variables in the module*
- Exercise 4: *Variable/Object life cycle*

After completing this module, students will be able to:

- Know where is the variables/objects defined during program runtime
- Understand various types of storage classes in C/C++ and choose the right one appropriately
- Appreciate the relationship between storage class and variable/object life cycle.

Module 7: Expressions and operators

This module explains expression and operator concept in C/C++ programs

Lessons

- Operators and operands
- The expression
- Operators
  - Arity
  - Precedence
  - Association
- Operator Types
  - Assignment
  - Arithmetic
  - Comparison
  - Logical

    - Bitwise
- Pitfall of the / operator
- The new rules of assignment operator in C++

Lab: *Using operators*

- Exercise 1: *Operator precedence*
- Exercise 2: The ?: operator
- Exercise 3: *The ++ and -- operators*
- Exercise 4: *Integer division vs floating point division*
- Exercise 5: *Assignment operator in conditional expression*
- Exercise 6: *Dealing with bits with bitwise operators*
- Exercise 7: *New face of assignment operator in C++*

After completing this module, students will be able to:

- Apply common operators used in expressions.
- Differentiate the sequence of ++ and - - operators in expression
- Avoid common errors caused by / operator
- Use ?: in expression to simplify coding
- Understand the use of assignment operator
- Distinguish the differences of assignment operator between C and C++.

Module 8: Compiler directives and constants

This module explains various important compiler directives and constants used in C/C++ programs

Lessons

- Compiler directives
    - #include
    - #define
    - #if .. #elif .. #else..#endif
    - #ifdef
    - #error
    - #undef
    - #pragma
    - ##
- Compiler Constants
    - __FILE__
    - __LINE__
    - __DATE__
    - __TIME__
    - __cplusplus

Lab: *Using compiler directives*

- Exercise 1: *Compiler directive for portability*
- Exercise 2: *Conditional compilation*
- Exercise 3: *Forcing error during compilation*
- Exercise 4: *Forget the definition*
- Exercise 5: *Control reentrant*
- Exercise 6: *Source level debugging using compiler constants*

After completing this module, students will be able to:

- Use compiler directive to create portable code
- Apply conditional compilation by using compiler directives.
- Display custom error massage during compile time
- Undefine definition from the compiler
- Avoid reentrant by using compiler directives
- Utilize compiler constants in source level debugging

Module 9: Pointers

This module explains pointer concept in C/C++, which is one of the most challenging aspects of the language.

Lessons

- Pointers is simple/native type
- The value Vs address of the storage.
- Parameter passing by value for pointer
- The power of indirect access
- Addressing Arithmetic
- Pointer casting
- **void** pointer
- Pointer of pointers
- Pointer to functions

Lab: *Showing the power of pointers*

- Exercise 1: *Variable value and address*
- Exercise 2: *Pointer is a native type*
- Exercise 3: *Parameter passing by value*
- Exercise 4: *The power of indirect access*
- Exercise 5: *Addressing arithmetic*
- Exercise 6: *Playing naughty with pointer casting*
- Exercise 7: *The power of pointer to void*
- Exercise 8: *Dynamic behavior with pointer to function*

After completing this module, students will be able to:

- Differentiate the value and address of variables
- Agree that pointer is a native type
- Appreciate the power of using pointer for indirect access
- Understand how the pointer arithmetic works
- Use pointer casting in special cases
- Apply pointer to void appropriately
- Write dynamic behavior code using pointer to function.

Module 10: Character and String

This module explains the character and string concepts

Lessons

- **char** as integer type
- Coding scheme
    - ASCII
    - Unicode
- Code Vs value
- Control characters
- Characters and String
- String terminator

- Escape characters in string
- String manipulation
  - Concatenation
  - Trimming
  - Etc.
- String and pointer
- Standard string manipulation functions

Lab: *Dealing with Strings*

- Exercise 1: *Use char as one byte integer*
- Exercise 2: ASCII code and ASCII value
- Exercise 3: *SToUppper( ) function*
- Exercise 4: *char**
- Exercise 5: *Using standard string manipulation functions*

After completing this module, students will be able to:

- Understand the relationship between char type and integer type
- Appreciate what are the ASCII and Unicode coding schemes
- Understand the string concept in C/C++
- Understand the relationship between string and pointer
- Write basic string manipulation functions
- Use some standard string manipulation functions

Module 11: Complex types

This module explains the concept of complex types in C/C++. More details explanation covered both homogenous and heterogeneous complex types.

Lessons

- Homogeneous Complex Data Type：Array
- Array indexing and out of bound issue
- Open declaration
- Optional last comma
- The use of **sizeof** keyword on array
- Multidimensional Array
- Array and pointers
- Heterogeneous Complex Types: **struct** (Record)
- Accessing items in a record
- Pointer to record
- The . and -> operator
- Record of record
- Array of Record

Lab: *Using operators*

- Exercise 1: *Sorting*
- Exercise 2: *Matrix multiplication*
- Exercise 3: *char* buf Vs char buf[ ]*
- Exercise 4: *Student records*

After completing this module, students will be able to:

- Understand the concept of complex types in C/C++
- Distinguish the arrays and pointers
- Define multi dimensional arrays.
- Use **struct** types
- Apply special notation for complex types correctly
- Construct complex in complex types

Module 12: Dynamic Allocation

This module explains the concept of dynamic allocation and understand why it is needed in C/C++.

Lessons

- Non deterministic during compile time: *Dynamic Allocation*
- The Heap's story
- **new** operator Vs **malloc** function
- **delete** operator Vs **free** function
- The different between **delete** and **delete**[]
- Dynamic allocation and pointers
- Issues with dynamic allocation
  - Memory leaking
  - Memory fragmentation
  - Pointer overrun

Lab: *Apply Dynamic Allocation*

- Exercise 1: *Variable inputs*
- Exercise 2: *Memory leaking*
- Exercise 3: *Pointer overrun*

After completing this module, students will be able to:

- Understand the why dynamic allocation is needed
- Use dynamic allocation for both C and C++ styles
- Appreciate the pitfalls of Dynamic Allocation
- Relate the Dynamic Allocation with pointers

Module 13: *Functions*

This module explains more features about functions in C++

Lessons

- Function prototyping
- Multiple modules project
- **static** functions
- Header files
  - What should be there?
- Parameter passing by value
- Passing address as value using pointer parameter
- The Synonym and Parameter passing by reference
- Function Overloading
- Default parameter value
- Function return types
  - Complex return
  - Return by reference
- Recursive function call
- **auto** variables and **static** variables
- Stack overflow

Lab: More Function challenges

- Exercise 1: *Multi module project*
- Exercise 2: *Swap( ) function*
- Exercise 3: *Sum( ) function*
- Exercise 4: *Return by Complex*
- Exercise 5: *Return by reference*

- Exercise 6: *Magic3x3*
- Exercise 7: *static variable*
- Exercise 8: *Stack overflow*

After completing this module, students will be able to:

- Appreciate the needs of function prototyping
- Handle multiple modules projects
- Understand why header files is needed and what should be kept in there.
- Differentiate various types of parameter passing in C/C++
- Differentiate the return type of C and C++
- Apply function overloading and default parameter value, and aware the conflict occurs
- Use recursive technique in solving complex problems
- Understand the stack overflow issue.
- Use **auto** and **static** appropriately.