TECHNOLOGIES

The "C" Programming Language course syllabus – associate level

Course description

The course fully covers the basics of programming in the "C" programming language and demonstrates fundamental programming techniques, customs and vocabulary including the most common library functions and the usage of the preprocessor.

Learning objectives

To familiarize the trainee with basic concepts of computer programming and developer tools.

To present the syntax and semantics of the "C" language as well as data types offered by the language

To allow the trainee to write their own programs using standard language infrastructure regardless of the hardware or software platform

Course outline

- Introduction to compiling and software development
- Basic scalar data types and their operators
- Flow control
- Complex data types: arrays, structures and pointers
- Structuring the code: functions and modules
- Preprocessing source code

Chapters:

- Absolute basics
- Languages: natural and artificial
- Machine languages
- High-level programming languages
- Obtaining the machine code: compilation process
- Recommended readings
- Your first program
- Variable – why?
- Integer values in real life and in "C", integer literals

Data types

- Floating point values in real life and in "C", float literals
- Arithmetic operators
- Priority and binding
- Post- and pre -incrementation and -decrementation
- Operators of type *op=*

- Char type and ASCII code, char literals
- Equivalence of *int* and *char* data
- Comparison operators
- Conditional execution and *if* keyword
- *printf()* and *scanf()* functions: absolute basics
- *Flow control*
- Conditional execution continued: the "*else*" branch
- More integer and float types
- Conversions – why?
- Typecast and its operators
- Loops – *while*, *do* and *for*
- Controlling the loop execution – *break* and *continue*
- Logical and bitwise operators

Arrays

- *Switch:* different faces of '*if*'
- Arrays (vectors) – why do you need them?
- Sorting in real life and in a computer memory
- Initiators: a simple way to set an array
- Pointers: another kind of data in "C"
- An address, a reference, a dereference and the *sizeof* operator
- Simple pointer and pointer to nothing (*NULL*) *&* operator
- Pointers arithmetic
- Pointers vs. arrays: different forms of the same phenomenon
- Using strings: basics
- Basic functions dedicated to string manipulation
- *Memory management and structures*
- The meaning of array indexing
- The usage of pointers: perils and disadvantages
- *Void* type
- Arrays of arrays and multidimensional arrays
- Memory allocation and deallocation: *malloc()* and *free()* functions
- Arrays of pointers vs. multidimensional arrays
- Structures – why?
- Declaring, using and initializing structures
- Pointers to structures and arrays of structures
- Basics of recursive data collections

Functions

- Functions – why?
- How to declare, define and invoke a function
- Variables' scope, local variables and function parameters
- Pointers, arrays and structures as function parameters
- Function result and *return* statement
- *Void* as a parameter, pointer and result
- Parameterzing the *main* function
- External function and the *extern* declarator
- Header files and their role

Files and streams

- Files vs. streams: where does the difference lie?
- Header files needed for stream operations

FILE structure

- Opening and closing a stream, open modes, *errno* variable
- Reading and writing to/from a stream
- Predefined streams: *stdin*, *stdout* and *stderr*
- Stream manipulation: f*getc()*, *fputc()*, *fgets()* and *fputs()* functions
- Raw input/output: *fread()* and *fwrite()* functions
- *Preprocessor and complex declarations*
- Preprocessor – why?
- *#include:* how to make use of a header file
- *#define:* simple and parameterized macros
- *#undef* directive
- Predefined preprocessor symbols
- Macro operators: # and ##
- Conditional compilation: *#if* and *#ifdef* directives
- Avoiding multiple compilations of the same header files
- Scopes of declarations, storage classes
- User defined types-why?
- Pointers to functions
- Analyzing and creating complex declarations

TECHNOLOGIES

Syllabus for "C++ Programming Language

"C++ programming language"

Description:

In this class, we will learn the basics about C++ programming language such as variables, data types, arrays, pointers, functions and classes etc.

Objective:

At the end of the class, we expect people to have a good understanding about the concept of object-oriented programming using C++, be able to write and read basic C++ code.

Prerequisite:

No prior knowledge about C++ is required, but people are expected to have some basic knowledge about computers, some knowledge about one or two other programming languages such as Perl, PHP, Python or Java etc is preferred.

Course Outlines:

Introduction

- What is C++?
- Why C++?
- C and C++
- Exception Handling
- Object Oriented Programming
- Standard Template Library

Types and declarations

- Types
- Booleans
- Integer Types
- Floating-Point Types
- Sizes
- Void
- Enumerations

- Declarations

Pointers, Arrays and Structures

- Pointers
- Arrays
- Pointers into Arrays
- Constants
- References
- Pointers to void
- Structures

Expressions and Statements

- A Deck Calculator
- Operator Summary
- Statement Summary
- Comments and Indentation

Functions

- Function Declarations
- Argument Passing
- Value Return
- Overloaded Function Names
- Default Arguments
- Pointer to Function
- Macros

Namespaces and Exceptions

- Namespaces
- Exceptions

Source Files and Programs

- Separate Compilation
- Linkage
- Using Header Files
- Programs

Classes

- Classes
- Access Control
- Constructors
- Member functions
- Static members
- Destructors
- Memory allocation
- Member initialization

Operator overloading

- Introduction
- Operator Functions
- A Complete Number Type
- Conversion Operators
- Friends
- Large Objects
- Essential Operators
- Subscripting
- Functions Calls
- Dereferencing
- Increment and Decrement
- A String Class

Derived class

- Introduction
- Derived Classes
- Abstract Classes
- Design of Class Hierarchies
- Class Hierarchies and Abstract Classes