C Language

The "C" Language is currently one of the most widely used programming languages. Designed as a tool for creating operating systems (with its help the first Unix systems were constructed) it quickly proved that it is suitable wherever you need high performance, speed, compactness and portability. Despite the fact that shortly after its release it was followed by a worthy descendant, the C + + language, it did not lose its importance and it still remains an essential tool for developers and designers in many applications. Wherever a code strongly associated with the operations of equipment is created, the C language proves its flexibility and adaptability. Network card drivers, graphics card software, operating systems, microcontrollers, which can be found everywhere around you, on your desk and in your car, in the kitchen and in the garage, simply – everywhere where intelligent electronics works – you are sure to find the effects of work of programmers who write in "C". The heart of Linux is nearly 15 million lines of code in "C". What better proof of the language's longevity.

Even in places where modern software with much more powerful abilities work, the C language was, is and will be present because it is the language in which runtimes (runtime environments) are written, responsible for performance, economical memory usage and reliability. The "C" language, niche extends from single-chip microcomputers controlling your coffee machine, to your laptop onto which you have just installed the latest graphics card drivers, to supercomputers that forecast the weather for your desired holiday.

The "C" language is not planning on growing old. New standards are still being created, and the language itself changes according to how the hardware development and how its usage possibilities change.

C++ Language

C++ is a general-purpose programming language designed by Bjarne Stroustrup as an extension to the C language with object-oriented data abstraction mechanisms and strong static type safety. Compliance with the C language at the level of source code remains one of the primary design goals of subsequent language standards.

Since 1998 the ISO / IEC 14882:1998 standard (Standard for the C + + Programming Language) with minor amendments approved in 2003 (ISO / IEC 14882:2003) has remained applicable. In 2009 a new standard was announced (known as C++0x), which came into effect as of 12 August 2011.

It is a highly developed programming language in terms of operators, simplicity, and the ease of notation. This allows for data abstraction and the use of several programming paradigms: procedural, object-oriented and generic. It is characterized by high performance of the object code, direct access to hardware resources and system functions, ease of creation and use of libraries (written in C++, C, or other languages), the independence of a specific hardware or system platform (which ensures high portability of source codes) and a small execution environment. The main areas of its application or applications and operating systems.

The C and C++ programming languages are now among the most popular languages used for creating all kinds of software. TIOBE specializes in assessing and tracking the quality of software. The company also carries out a continuous popularity ranking of programming languages, available here. To the surprise of many, the C language is often the most popular programming language, even ahead of JAVA. One may certainly observe a slight dip of C++ from the third position to the fourth.

Course Description

This course provides a fast-paced introduction to the C and C++ programming languages. You will learn the required background knowledge, including memory management, pointers, preprocessor macros, object-oriented programming, and how to find bugs when you inevitably use any of those incorrectly. There will be daily assignments and a small-scale individual project.

Associate – which is the foundation level. A holder of a certificate at the associate level possesses the knowledge of the basics of programming in the C (CLA) or C++ (CPA) language, demonstrates fundamental programming techniques, customs, vocabulary and the most common library functions.

Professional – which is the advanced level. A holder of a certificate at the advanced level possesses the knowledge of advanced programming in the C (CLP) or C++ (CPP) language, demonstrates advanced programming techniques, customs, and vocabulary as well as advanced library functions. He or she is capable of using and creating complex algorithms and is able to deal with complex coding problems.

Senior – which is the expert level. A holder of a certificate at the senior level possesses knowledge of heavily advanced programming in the C (CLS) or C++ (CPS) language, demonstrates remarkably advanced programming techniques and is able to deal with highly complex coding problems. He or she is capable of creating and implementing highly complex algorithms.

Benefits

The certification program provided by C++ Institute and Pearson VUE brings **huge value** to both an **individual holding the certificate** and **the company employing the individual**.

- First, **the certificate helps you validate your knowledge**, which helps you objectively prove that you present some specific level of C/C++ programming proficiency. It is **a signal for you that you have attained a certain skill**, and it is an indication for your employer that you do possess appropriate knowledge. The certificate is **an investment in your personal brand**.

- Second, the certificate lets your employer know that **you care about self-improvement and self-development**. It also **significantly helps you** go through the very first recruitment stage **while you're looking for a job in programming**. After all, recruiters have to use some screening method and the certificate is a perfect tool for that.

- Third, **the certificate helps you earn more**. Documenting your skills and presenting the certificate strengthens your bargain position while you're negotiating for a higher salary.

- Fourth, the certificate benefits the company as **it increases working efficiency**, **productivity and profit**. And because qualified workers are always well-perceived, **it increases company's reputation and adds to its good brand image**.

- Fifth, the certificate enables you to **become a part of the C/C++ certified community**, which does not only let you feel a great sense of achievement, but also make interesting relationships and actively contribute to the world of certified professionals.

CLA – C Programming Language Certified Associate

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
- Parameterizing 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
- Macrooperators: # 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

CPA – C++ Certified Associate Programmer

Course description

The course fully covers the basics of programming in the "C++" programming language and presents the fundamental notions and techniques used in object-oriented programming. It starts with universal basics, not relying on object concepts and gradually extends to advanced issues observed in the objective approach.

Prerequisite Courses

The "C" programming language course – associate level (suggested)

Learning objectives

- To familiarize the trainee with the universal concepts of computer programming.
- To present the syntax and semantics of the "C++" language as well as basic data types offered by the language
- To discuss the principles of the object-oriented model and its implementation in the "C++" language
- To demonstrate the means useful in resolving typical implementation problems with the help of standard "C++" language libraries

Course outline

- Introduction to compiling and software development
- Basic scalar data types, operators, flow control, streamed input/output, conversions
- Declaring, defining and invoking functions
- Strings processing, exceptions handling, dealing with namespaces
- Object-oriented approach and its vocabulary
- Dealing with classes and objects
- Defining overloaded operators
- Introduction to STL

Chapters:

Absolute basics

- Machine and high-level programming languages, compilation process
- Obtaining the machine code: compilation process
- Recommended readings
- Your first program
- Variable – why?
- Integers: values, literals, operators
- Characters: values, literals, operators
- Dealing with streams and basic input/output operations

Flow control and more data types

- How to control the flow of the program?
- Floating point types: values, literals, operators
- More integral types: values and literals
- Loops and controlling the loop execution
- Logic, bitwise and arithmetic operators

Functions

- Functions: why do you need them?
- Declaring and invoking functions
- Side effects
- Different methods of passing parameters and their purpose
- Default parameters
- Inline functions
- Overloaded functions

Accessing data and dealing with exceptions

- Converting values of different types
- Strings: declarations, initializations, assignments
- String as the example of an object: introducing methods and properties
- Namespaces: using and declaring
- Exception handling

Fundamentals of the object-oriented approach

- Class: what does it actually mean?

- Where do the objects come from?
- Class components
- Constructors
- Referring to objects
- Static members
- Classes and their friends
- Defining and overloading operators

Class hierarchy

- Base class, superclass, subclass
- Inheritance: how does it work?
- Types of inheritance
- Inheriting different class components
- Multiple inheritance

Classes – continued

- Polymorphism: the notion and the purpose
- Virtual methods: declaring and using
- Inheriting virtual methods
- Abstraction and abstract classes

Exceptions – dealing with expected and unexpected problems

- What is *an exception*?
- Catching and throwing exceptions
- Different classes and hierarchy of exceptions
- Defining your own exceptions

CPP – C++ Certified Professional Program

Course description

The purpose of the course is to familiarize students with C++ advanced topics which are templates and Standard Template Library.

Prerequisite Courses

The Student should have good knowledge of the C++ language, including inheritance and operator overloading – recommended to complete C++ language course at the associate level.

Learning objectives

- To gain knowledge of C++ template mechanism,
- To be able to read and understand definitions of template functions and classes,
- To be able to use property template classes and methods including third party templates,
- To know how to create template functions and classes
- To gather good knowledge of C++ STL library including the IO part
- To be able to solve common programming problems with STL predefined classes and methods

Chapters:

Templates

- What are templates?
- Basic syntax
- Function templates
- Class templates
- When should we use them?
- Typical problems when using templates

STL Sequential containers

- Types of sequential containers
- vector, deque, list and their API
- Sequential container adapters – stack, queue and priority queue
- Dealing with objects as container elements
- Usage guidelines – when to use what

STL Associative containers

- Types of associative containers
- Set and multiset – behavior and api
- Map and multimap – behavior and api
- Putting objects into set and map
- Usage guidelines – when to use what

Non modifying STL algorithms

- Definition of non modifying algorithm
- List of non modifying algorithms: for_each, find, find_if, find_end, find_first_of, adjacent_find, count, count_if, mismatch, equal, search, search_n
- Examples
- Container compatibility

Modifying STL algorithms

- Definition of modifying algorithm
- List of non modifying algorithms: transform, copy, copy_backward, swap, swap_ranges, iter_swap, replace, fill, fill_n, generate, generate_n, remove, remove_if, unique, unique_copy, reverse, reverse_copy, rotate, partition, stable_partition
- Examples
- Container compatibility

Sorting STL operations

- List of sorting algorithms: random_shuffle, sort, stable_partition, lower_bound, upper_bound, equal_range, binary_search
- Examples
- Containers compatibility
- Sorting of objects

STL merge operations

- List of merging algorithms: merge, includes, min_element, max_element, inplace_merge

- STL operations for sets
- Examples
- Container compatibility

STL utilities and functional library

- STL “small” tools
- List of useful functors
- Examples

STL advanced I/O

- Classes which provide the input and output capability
- Console I/O
- Formatting
- File I/O
- Strings I/O
- Examples