The "C" Programming Language course syllabus - associate level
(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 size of 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 deal location: 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: fgetc(), 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

C Graphics, Sound and Animation

Be Practical and Knowledgeable PS SOFTECH