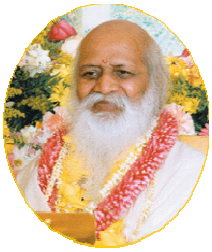
Maharishi Skills
SKILLS | EDUCATION



Certificate in C Programming

Duration : 240 Hrs / 3 Months

Overview

- 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

Function

- 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
- Parameter zing 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