ANNEXURE – I

C with Data Structures

Objective: At the end of the ten day session participants will be very comfortable with programming C at advanced level for systems programming and with applying Advanced Data Structures & Algorithms for industry standard applications.

The session has two inter-leaved threads of topics presented. First thread includes 'C' language topics while the second thread has Data Structures and Algorithms at an advanced level. Data Structures and Algorithm are introduced early in the session to provide sufficient time for implementations.

Prerequisites

The participants are expected to have acquired good practical working knowledge of the prerequisites listed below, prior to attending the session.

Basic Types:
- Integer types: char, short, int, long and the unsigned integers
- Two's complement system
- Floating types, float, double,long double,

Control flow
- if-else-if, for(), while(), do{}while(), switch, pitfalls with goto
- Infinite loops

Arrays
- Single dimensional arrays
- Initialization
- Operations on array elements

Pointers
- Declaration, Initialization,
- Pointers to elements of arrays
- Dynamic memory allocation

Functions
- Call by value and reference

Derived Types
- Structures and Unions
- Initializations
- Operations on members

DAY1:

History of C

C standards
- Non-ANSI C and ANSI C, C89 and ISO C99.

Overview of the language and similarities to assembly
Core language and the C library
Automatic usage of the C library
Calling library functions and including their header files

Data types

Integer types, Unsigned inter types and Floating types
Character types: char, signed char and unsigned char.

Limits for the numerical types as defined in the header files <limits.h>
Optimal numeric types for a given CPU

Constants and Their types

Default types for integer constants
Octal and hexadecimal prefixes
Unsigned , long and long long suffixes
Default types for floating constants
Float and long double suffixes
Hexadecimal floating constants
Character constants
Escape sequences for newline, carriage return, vertical tab ,
backspace, tab and form feed.
Octal and hexadecimal escape sequences

Identifiers

Definition
Rules and conventions to be followed for naming
Purpose of identifiers

Standard I/O Functions

Buffered I/O by the c library
Line buffered I/O to terminal
Char functions: getc(), putc() and ungetc()
String I/O functions: gets() and puts()
Buffer overflow defect with gets() and using fgets()
Formatted I/O functions: printf() and scanf()
Displaying pointer values
Length modifiers for short, long and long long types
White space separations between values
Width specifications for integers and floating types
Precession specification for floating types
Avoiding buffer overflow with scanf() for string input
Flag characters : #, 0, -, + and space

memset(), memcpy(), memmove() and memcmp()

Multidimensional arrays

Two dimensional arrays, array of arrays
Initialization
I/O using two dimensional arrays
Implementing matrix addition and multiplication
Extending to three and multi-dimensional arrays

Exercise: - Expression **evaluations**

Postfix expressions
Conversion from infix notation to post fix notation

DAY2:

Strings

Strings as an array of chars
Definition and storage
String literals (constant strings) and storage
Using string to initialize char pointers and char arrays
Revisiting string I/O with gets(),puts(), printf() and scanf().
Contrasting string I/O with character I/O

List of strings as a two dimensional array of chars
String I/O with two dimensional array of chars

Character processing with
isalpha(), sialnum(), isdigit(),. ....toupper() and tolower()

String processing with
strlen(), strcpy(), strncpy(), strcmp ()and strncmp()
strchr(),strrchr(),strstr(), strtok(),
atoi() and atof()

Functions for I/O with string target
Input from string with sscanf()
Output to string with sprintf() and snprintf()

Expressions:

Operators:

Relational
Arithmetic
logical
Assignment
Increment and Decrement
Conditional
Bit wise
Comparison of logical and bit wise operators
Special operators
Coma operator
Sizeof
Definition of [ ] operator
Type cast operator

Through study of operator precedence and associativity
Order of evaluation of expressions: left to right & right to left
Demonstration of order of evaluation with a simple yacc & lex program
Order of evaluation for &&,|| and ?: operators
Side effect operators and implementation dependencies
Increment, decrement and assignment
sizeof() operator and non evaluations of operand
Lvalues, Modifiable Lvalues and Rvalues

Sequence points

Effects of sequence points
List of sequence point definitions

Type Conversions

Implicit type conversion in expressions and promotions
Explicit type casting
Conversions with expressions having signed and unsigned operands
Sign preserving Vs Value preserving conversions

Functions

Definitions, Invocation and parameter passing
Concepts of call by value and by reference
Stack frames
Buffer over flow vulnerability in functions like gets()
Function prototypes
Void return type and void arguments
main() function specifics, return type
Command line arguments
Recursive functions, Tail recursion
Variable argument list functions
Defining printf() functions using va_arg() macros
Library functions for error processing
errno variable, perror() and strerror() functions

DAY3:

Algorithm Analysis

Need for theoretical Asymptotic Analysis
Inadequacy of practical tests
Big O, Omega and Theta Notations
Time complexity
Space complexity
Behavior of standard mathematical functions that occur frequently with Big O analysis
Abstract Data Structure Definition

Linked lists

Single linked lists
Doubly linked lists
Circular linked lists
Implementation of the above lists

Stacks

Implementation with arrays and linked lists
Constant time O(1) operations

Queues

Implementation with arrays and linked lists
Constant time O(1) operations

Exercise: - Merging Two Sorted Lists
Stack Packet (Stack with packet size)

DAY4:

Pointers

Definitions
Invalid pointers, illegal memory access and segmentation faults
Pointers to integers & characters
Pointer initialization
Pointers into single dimensional integer and character arrays.

Pointer Arithmetic, addition and subtractions

Dynamic memory allocation with malloc() family of functions
- Freeing of dynamic memory
- Adjusting memory requirements, using realloc()
- Allocating n elements of a given size

Pointer to a constant Vs constant pointer
Arrays and pointers: Similarities and differences
Void pointers and alignment restrictions for data types
- Argument types of memset() and memcpy() kind of functions

Array of pointers
- Allocating memory for pointers in the array of pointers

Pointer to array
- Arithmetic of pointer to an array
- Allocating memory for pointer to an array

Double pointers
- Allocating memory for two dimensional array using double pointers

Function pointers
- Definitions and Initializations
- Calling functions using function pointers
- Passing function as an argument to another function
- Callback functions

Exercise: - Registering For Multicast callbacks
Writing higher order functions to curry parameters

DAY5:

Sorting algorithms

Numeric and string keys
Stable sorts
Internal and external sorts

Bubble sort
Insertion sort
Selection sort
Implement one of the above

Merge sort
Quick sort
Implement one of the above

Exercise: - Sorting of linked list
Insertion Sort with linked list

Storage classes, qualifiers and scope of identifiers
Automatic storage class
Static storage class
Register storage class
Block scope, Function scope and File (global) scope
Comparison with dynamic memory allocation
Constant and volatile qualifiers

Linkage
Symbol linkage across multiple source files
External linkage
Static linkage
A revisit of function prototypes and #include files

Trees
Generic trees
Binary trees
Mapping generic trees to Binary trees
Minimum and maximum heights of binary trees

Binary trees traversals
Preorder, inorder and post order
Level order traversal for binary heap implementation

Exercise: - Representing polished notation in Binary tree
Evaluating the expression tree

Binary search trees
Insertion, search and deletion operations
Implementation
Finding the height of a binary search tree
Implementation
Degenerate unbalanced binary search trees
Similarity to linked lists
Reference to AVL Trees and Red-Black Trees