Data and File Structures
[Course Planner with Lab Exercises]

Academic Year 2011-12
Even Semester

Indira Gandhi Institute of Technology
Guru Gobind Singh Indraprastha University
Kashmere Gate, Delhi

Version No.	Comments	Modification Date	Modification Done By
1.0	Created lab planner – added syllabus and lab exercises	2nd Jan 2012	Rishabh Kaushal Assistant Professor Department of Information Technology, IGIT

Table: Modification History

Table of Contents

0. Guidelines

This document titled ‘Course Planner’ is written based on following guidelines.

1. Course work is divided into across weeks. Each week has certain objectives to be accomplished which shall be achieved by solving programming problems of varying complexity.

2. Each week has 4 lectures hours supplemented with 2 hours scheduled Lab work.

3. We learn inner workings of a data structure when it is at work. Therefore, it is highly recommended that each and every data structure discussed in classroom is *implemented.* Being a course which forms bedrock for Computer Science, it is expected and strongly encouraged that **students not only utilize their Lab hours efficiently, but also implement data structures in their personal computing environment**.

4. In case for any reason (holiday or otherwise), a Lab session is not held during a week, even then the stipulated lab problems for the week are to be done by students in their personal computing environments.

1. Syllabus

As taken from www.ipu.ac.in website*.

**Academics → Academics → Scheme Syllabus (Affiliated Institutes, w.e.f. 2005-06)*

UNIT – I
Fundamentals of algorithm analysis: Big 'O' notations, Time and space complexity of algorithms, linked lists: singly and doubly linked lists, stacks, queues, double stack, multistacks and multiqueues, deques, polynomial arithmetic, infix, postfix and prefix arithmetic expression conversion and evaluations. [No. of Hrs: 08]

UNIT – II
Trees: Binary trees: Definition, Binary Search Tree basic operations, Tree Traversals (recursive and stack based non-recursive), Heaps and priority queues, Threaded binary tree, AVL Trees B-Tree: need, properties, creation, uses. B+ tree, B* tree. [No. of Hrs: 10]

UNIT – III
Graphs: Representation (Matrix and Linked), Traversals, Connected components, Spanning trees, Shortest path and Transitive closure, Topological sort, Activity network, Critical path, Path enumeration. Dijkstra's Algorithm, Floyd Warshall's Algorithm, Coloring of Graphs, Spanning Tree, Minimum Spanning Tree Algorithms (Kruskal's Algorithm, Prim's Algorithm)
Searching & Sorting: Binary search, Hash function, Hash table, Search tree. Internal sort: Radixsort, Insertion sort, Selection sort, Shell sort, Quick sort, Merge sort, Heap sort.
[No. of Hrs: 16]

UNIT – IV
Files: Sequential file organization, creating updating retrieving from sequential files advantages and disadvantages of sequential file organization. Data representation and denisity, parity and error control techniques, devices and channels, double buffering and block buffering, handling sequential files in C language, seeking, positioning, reading and writing binary files in C. External Sorting and merging files k way and polyphase merge [No. of Hrs: 08]

TEXT BOOKS:
1. E. Horowitz and S. Sahani, "Fundamentals of Data Structures in C", 2nd Edition, Universities Press, 2008.
2. Mark Allen Weiss, "Data Structures and Algorithm Analysis in C", 2nd Edition Addison-Wesley, 1997.

REFERENCES:
1. Schaum's Outline Series, "Data Structure", TMH, Special Indian Ed., Seventeenth Reprint, 2009.
2. Y. Langsam et. al., "Data Structures using C and C++", PHI, 1999.
3. N. Dale and S.C. Lilly, D.C. Heath and Co., "Data Structures", 1995.
4. R. S. Salaria, Khanna, "Data Structure & Algorithms", Book Publishing Co. (P) Ltd., 2002.
5. Richard F. Gilberg and Behrouz A. Forouzan, "Data Structure A Pseudocode Approach with C", Cengage Learning, 2nd Ed., 2005.

6. Mary E. S. Loomes, “Data Management and File Structure”, PHI, 2nd Ed., 1989.

1.1 Course Coverage Plan

This section contains the topics to be covered for various examinations.

Topics to be covered in **Minor-1**:
Unit I and Unit II (till Priority Queue)

Topics to be covered in **Minor-2**:
Unit-II (Priority Queue onwards) and Unit III

2. Lecture Plan

Given below is a tentative plan for the conduct of lectures in classroom for this course. Please note that this plan may change during the course progress.

Week 1: Introduction to data structures	
L1	Introduction, Need for data structures and their impact
L2	Asymptotic notations (Big-oh, Omega and Theta)
L3	Time and Space complexity of data structures
L4	Review of pointers (memory maps)
Week 2: Linear data structures	
L5	Stacks & Queues
L6	Double Stacks
L7	Linked List (singly)
L8	Linked List (doubly)
Week 3: Linear data structures (contd.) & its applications	
L9	Multi queues, Multi stacks & Deque
L10	Polynomial arithmetic
L11	Infix, Postfix and Prefix arithmetic conversions & evaluations
L12	Infix, Postfix and Prefix arithmetic conversions & evaluations (contd.)
Week 4: Non linear data structures	
L13	Recursive functions & its implications
L14	Binary Trees: definitions and review of concepts
L15	Binary Search Tree: basic operations
L16	Tree traversals:
Week 5: Tree data structures	
L17	Tree traversals: recursive
L18	Tree traversals: non-recursive using stacks
L19	Heaps
L20	Priority queue
Week 6: Review of Data Structures	
L21	Linear data structures – arrays, linked lists
L22	Linear data structures – queue, stacks
L23	Non-Linear data structures – tree
L24	Non-Linear data structures – tree
MINOR EXAM - 1	
Week 7: Tree (advanced topics)	
L25	Threaded binary tree
L26	AVL Tree
L27	B Tree: need, creation, properties & uses
L28	B+ tress & B* trees
Week 8: Graph – Introduction	
L29	Representation – Matrix and Linked
L30	Traversal of Graphs - BFS
L31	Traversal of Graphs - DFS

L32	Graph implementation illustrated
Week 9: More on Graphs	
L33	Connected components of Graphs
L34	Topological sort
L35	Activity network
L36	Implementation illustrated
Week 10: Spanning Tree	
L37	Spanning Tree – Concept
L38	Kruskal’s algorithm
L39	Prim’s algorithm
L40	Implementation illustration
Week 11: Paths in Graphs	
L41	Shortest path and Transitive closure
L42	Critical path, Path enumeration
L43	Dijikstra’s algorithm
L44	Floyd Warshall’s algorithm
Week 12: Searching and Sorting	
L45	Linear & Binary Search
L46	Hash Function, Hash table & Search Tree
L47	Internal sort, Radix sort, Insertion sort and Selection sort
L48	Shell sort, Quick sort, Merge sort and Heap sort
MINOR EXAM - 2	
Week 13: Files	
L49	Sequential file organization
L50	Creating updating retrieving advantages and disadvantages
L51	Data representation and density, parity and error control techniques
L52	Devices and channels, double buffering and block buffering
Week 14: More on Files	
L53	Handling sequential files in C language,
L54	Seeking, positioning, reading and writing binary files in C
L55	External Sorting and merging files
L56	K-way and poly-phase merge

Above lecture hours shall cover the course till Minor-1, plan for remaining topics shall be updated later.

3. Lab Work Guidelines and Plan

Guiding Principle:

- The way to learn data structures is to write programs implementing them. It is only when we implement data structure, we understand and appreciate its intricate workings.
- Data structure follow the following life cycle:
    a. Design / Proposal of data structure
    b. Implementation (C/C++) of data structure
    c. Analysis – Finding the asymptotic time and space complexity of data structure

3.1 Submission Guidelines

Following are the modalities for evaluation:-

1. All programs for implementing data structure have to be submitted **online to a web server** hosted in the Lab.
2. Programs submitted will undergo *online* compilation and *offline* (optional) testing along with necessary *plagiarism checks* in order to evaluate the program from time to time.
3. Students would maintain a **handwritten *lab file*** to keep a record of their ***weekly progress***. For each problem, write the following in *lab file*:-
    a. Problem statement
    b. Data structure design (with optional relevant code snippets)
    c. Asymptotic time and space complexity analysis
4. **Printouts of data structure implementation will NOT be accepted.**
5. There would be **zero tolerance** for plagiarism of any kind. In any event of detection of plagiarism, all students involved would be penalized including the perceived "original" writer. It is responsibility of a student to secure their work.

3.2 Evaluation Policy

Following shall be the break-up of the internal assessment.

Lab File work = 25%

Viva* = 25%

Online submission = 25%

Implementation (Evaluation Days) = 25%

* Viva (oral examination) shall be conducted ***randomly*** during the stipulated lab hours. Each student shall undergo two such evaluations, one before Minor-1 and another before Minor-2.

Following shall be the remarks corresponding to the range of marks (in percentage) awarded eventually towards the end of the course.

Grade	**Range of Marks**	**Remarks**
EX	95-100	Excellent
A	85-94	Very Good
B	75-84	Good
C	65-74	Average
D	55-64	Poor

4. Lab Exercises

This section includes a consolidated list of programming exercises based on topics covered every week.

Note: Please ensure that you have installed the programming environment as mentioned in Appendix A.

4.1 Week-1: Introduction to data structures

1. **Array operations.** Array is an important data structure for static allocation. Write C program to perform the following operations on an array:-
   a. Adding an element into an array after ith index
   b. Updating an element in array
   c. Deleting an element from an array from ith index

2. **Large integer arithmetic.** Arithmetic operations are to be performed on very large integers of N digits (where $0 < N < 20$). Write C program that performs the operations of addition, subtraction, multiplication and division on such large integers.

3. **Searching**. Write C programs to perform both linear and binary search on a given random set of integers. Your program should do the following:-
   - Take as input an integer, N, which would decide number of integers to be processed and another input an integer, X ($0 < X < N+1$), which is the key to be searched
   - Randomly generate N integers whose values are between 1 to N, multiple entries are allowed
   - Output all the indexes (positions) of key in given set of random integers
   - Count number of comparisons in the linear and binary searching process, please note comparisons involved in sorting process (in case of binary search) are not to be included
   - Output the result in following table:-

Input size (N)	**Number of Comparisons**	
	Linear Search	**Binary Search**
10		
100		
1000		
10000		
100000		

4.2 Week-2: Linear data structures

1. **Stack (using arrays) operations.** Stack is a data structure operating in LIFO manner. Write C program to perform the following operations on the stack:-
   a. Pushing an element into stack.
   b. Popping an element out of stack.
   c. Displaying all elements of stack.

   Implement stack data structures using *arrays*.

2. **Two stacks using single array.** Implement two stacks using a single array such that neither overflows unless total number of elements in both the stacks is equal to the size of the array.

3. **Queue (using arrays) operations.** Queue is a data structure operating in FIFO manner. Write C program to perform the following operations on the queue:-
   a. Inserting an element into queue.
   b. Deleting an element from queue.
   c. Displaying all elements of queue.

   Implement queue data structures using *arrays.*

4. **Linked List operations.** Linked list (singly) is an important data structure for dynamic allocation. Write C program to perform the following operations on the linked list:-
   a. Adding an element at beginning, after i^{th} node and end of linked list.
   b. Updating an element in linked list at i^{th} node and data.
   c. Deleting an element from linked list at beginning, at i^{th} node, based on data and end of linked list.
   d. Search for a data in linked list and return its node position from start.
   e. Traversal of all elements of the list.

5. **Stack (using linked list) operations.** Stack is a data structure operating in LIFO manner. Write C program to perform the following operations on the stack:-
   a. Pushing an element into stack.
   b. Popping an element out of stack.
   c. Displaying all elements of stack.

   Implement stack data structures using *linked list*.

6. **Queue (using linked list) operations.** Queue is a data structure operating in FIFO manner. Write C program to perform the following operations on the queue:-
   a. Inserting an element into queue.
   b. Deleting an element from queue.
   c. Displaying all elements of queue.

   Implement queue data structures using *linked list.*

4.3 Week-3: More on Linear Data Structure and Applications

1. **Doubly Linked List operations.** Linked list (doubly) is an important data structure for dynamic allocation wherein elements can be traversed by either direction. Write C program to perform the following operations on the linked list:-
   a. Adding an element at beginning, after i^{th} node from both directions and end of linked list.
   b. Updating an element in linked list at i^{th} node from both directions and data.
   c. Deleting an element from linked list at beginning, at i^{th} node from both directions, based on data and end of linked list.
   d. Search for a data in linked list and return its node position from start.
   e. Traversal of all elements in the list.

2. **Deque.** Deque is a queue which allows insertions and deletions at both ends. Write a C program that implements deque using both *arrays* and *linked list*. Each implementation should provide support for following operations:-
   a. Insertion of an element at both ends.
   b. Deletion of an element from both ends.
   c. Displaying all elements of deque.

3. **Evaluation of arithmetic expression.** Write a C program that reads an input arithmetic expression in given below notations and outputs its result.
   a. Infix notation
   b. Prefix notation
   c. Postfix notation

4. **Arithmetic expression notation conversion.** Write a C program that reads an input arithmetic expression in infix notation (fully parenthesized) and converts it into an output arithmetic expression in postfix notation.

5. **Multiple queues.** In an operating system, different queues are maintained based on process priority. A process with priority level p is inserted into a queue Q_p and while servicing a process, the operating system first services all processes from queue with highest priority level by de-queuing each process at a time.

   Write a C program using *multiple queues* that simulate the above behavior by providing support for following operations:-
   a. Insertion of a process into appropriate queue
   b. Deletion of a process from appropriate queue
   c. Displaying the contents of all the queues

   Each process may be assumed to have attributes as process ID, process name and process priority. Your program takes as input an integer, N, which represents number of priority levels with priority level 1 representing highest priority.

6. **Multiple stacks.** In an operating system X, different stacks are maintained for each newly spooned process. A *parent process* during its execution may spoon multiple *child processes* each of which is maintained in a stack.

   Write a C program using *multiple stacks* that simulate the above behavior by providing support for following operations:-
   a. Spooning of a new process, thereby creating a stack
   b. Begin executing an instruction for a process, thereby pushing a node in the stack corresponding to the process
   c. Finish executing an instruction for a process, thereby popping a node from the stack corresponding the process
   d. Process execution completes, deleting its corresponding stack.
   e. Displaying the contents of various stacks.

   Each process may be assumed to have attributes as process ID and process name. Each instruction may be assumed to have attributes as process ID, instruction ID and instruction name.

7. **Redo and Undo support.** In a text editor application, support is to be provided for *undo* and *redo* operations. Write a C program using *multiple stacks* that reading sequence of characters from user and provide above mentioned support by ensuring that any character typed can be undone and redone by pressing special keys.

4.4 Week-4: Advanced Operations using Linked List

1. **Maximum and Minimum.** Write a program to find minimum and maximum number, given a list of numbers as input.

2. **Reverse Linked List.** Write a C program that reverses a linked list using both the approaches given below.
   i. Recursive solution
   ii. Iterative solution

3. **Union and Intersection.** Given two linked lists of numbers, write a program that finds a resultant linked list which is union of the two input linked lists and another resultant linked list which is intersection of the two input linked lists.

4. **Detecting cyclic linked list.** Write a C program that detects whether a given linked list is cyclic or not, if yes, then return the node where the cycle begins.

5. **In-place Sort.** Write a C program that sorts the given list of numbers in a linked list *in-place,* thereby making a given unsorted linked list into the sorted linked list.

4.5 Week-5: Tree data structure

1. **Recursive review.** Write recursive programs for each of the following:-
   a. Decimal to Binary conversion
   b. Reversing a string
   c. Sorting

   Note: *Submit a single program using switch-break statement for each of the above cases.*

2. **Recursive Tree traversals.** Write C program to display a tree using all the following methods of traversals:-
   a. Inorder traversal
   b. Preorder traversal
   c. Postorder traversal

3. **Non-Recursive (stack based) Tree traversals.** Write C program to display a tree using all the following methods of traversals:-
   a. Inorder traversal
   b. Preorder traversal
   c. Postorder traversal

4. **Binary Search Tree.** Binary Search Tree is an important data structure for dynamic allocation and optimized searching. Write C program to perform the following operations on binary search tree (BST):-
   b. Adding an element.
   c. Updating an element.
   c. Deleting an element.
   d. Search for an element.
   e. Displaying all elements (in-order).

4.6 Week-6: Application of Trees

1. **Building heap.** Using a C program, build a *max-heap*, given N random integers. Display the heap thus formed in its in-order form.

2. **Heap sort.** Using the heap data structure, sort the given N random integers.

3. **Priority queue.** Implement the priority queue functionality (insertion, deletion, update and display) using the *min-heap* property.

4.7 Week-7: Advanced Trees

1. **Threaded Binary Tree.** Using a C program, perform the following operations on the *threaded binary tree* data structure:-
   a. Inserting an element
   b. Deleting an element

   c. Update an element
   d. Searching an element
   e. Displaying an element

2. **AVL Trees.** Using a C program, perform the following operations on the *AVL tree* data structure:-
   a. Inserting an element
   b. Deleting an element
   c. Update an element
   d. Searching an element
   e. Displaying an element

3. **B Trees.** Using a C program, perform the following operations on the *B tree* data structure:-
   a. Inserting an element
   b. Deleting an element
   c. Update an element
   d. Searching an element
   e. Displaying an element

4. **B+ Trees.** Using a C program, perform the following operations on the *B+ tree* data structure:-
   a. Inserting an element
   b. Deleting an element
   c. Update an element
   d. Searching an element
   e. Displaying an element

5. **B* Trees.** Using a C program, perform the following operations on the *B* tree* data structure:-
   a. Inserting an element
   b. Deleting an element
   c. Update an element
   d. Searching an element
   e. Displaying an element

4.8 Week-8: Graphs Algorithms

1. **Adjacency Matrix representation.** Represent a given input graph in its adjacency matrix form and print the contents of the matrix. Your program should provide support for both directed as well as undirected graphs.

   Note: *Input about the graph is to be read from a file mentioning the number of vertices (each separated by a comma) on first line, after which each edge is represented in a vertex-pair in a separate line that follows.*

eg. Input files for graph with five vertices and four edges.
1,2,3,4,5
1,2
2,3
4,2
5,1

2. **Adjacency List representation.** Represent a given input graph in its adjacency list form and print the contents of the matrix. Your program should provide support for both directed as well as undirected graphs.

   ***Note:** Input about the graph is to be read from a file as per the format suggested in Question No. 1 above.*

3. **Graphical representation*.** Given input graph information in form of an input file as explained in Note above, can you suggest and implement an algorithm that actually draws the input graph in its diagrammatic form, the *graphics.h* library learnt as part of Computer Graphics course may be used to draw the vertices, edges and labels.

4. **Breadth First Search.** Given an input graph (in form of an input file as explained in Note above), perform breadth first search and print its result.

5. **Depth First Search.** Given an input graph (in form of an input file as explained in Note above), perform breadth first search and print its result.
6. **Topological Sorting.** In a university curriculum, often each course has a set of pre-requisites. Given a set of courses along with their respective set of pre-requisites, prepare a curriculum such that no course appears before its pre-requisite.

4.9 Week-9: Spanning Tree Algorithms

1. **Strongly connected components.** Given an input graph, write a program that prints its strongly connected components.

   ***Note:** Input about the graph is to be read from a file mentioning the number of vertices (each separated by a comma) on first line, after which each edge is represented in a vertex-pair in a separate line that follows.*
   eg. Input files for graph with five vertices and four edges.
   1,2,3,4,5
   1,2
   2,3
   4,2
   5,1

2. **Kruskal algorithm.** Generate minimum spanning tree (print it in an in-order traversal order) from a given input graph as per Kruskal algorithm.

   Note: *Input about the graph is to be read from a file mentioning the number of vertices (each separated by a comma) on first line, after which each edge is represented in a vertex-pair in a separate line that follows.*
   eg. Input files for graph with five vertices and four edges.
   1,2,3,4,5
   1,2
   2,3
   4,2
   5,1

3. **Prim algorithm.** Generate minimum spanning tree (print it in an in-order traversal order) from a given input graph as per Prim algorithm.

   Note: *Input about the graph is to be read from a file mentioning the number of vertices (each separated by a comma) on first line, after which each edge is represented in a vertex-pair in a separate line that follows.*
   eg. Input files for graph with five vertices and four edges.
   1,2,3,4,5
   1,2
   2,3
   4,2
   5,1

4.10 Week-10: Shortest Path Algorithms

1. **Shortest Route Map.** A city's map with all the roads along with distances (in kilometers) between various places in the city is given. Using *Bellman Ford algorithm* and *Dijikstra's algorithm*, find the shortest path from a given input place fed as a source in the algorithm.

2. **Dynamic Shortest Route Map.** Suppose that in addition to the distance in the routes, real time status of the traffic conditions are also being provided at regular time intervals. Find the shortest route from a given input place fed as source. You can use any of the algorithm either *Bellman Ford algorithm* and *Dijikstra's algorithm.*

3. **Floyd Warshall Algorithm.** Given an input graph, using *Floyd Warshall algorithm*, find all pair shortest path.

4.11 Week-11: Searching and Sorting

1. **Merge Sort.** Logging activity files of two users are given as input, merge them into a single file. Assume that the format of logging activity file is two column with first

column representing the date-time record and second column the event description. Merging is to be done with respect to the date-time record field.

2. **Quick Sort with pivot selection.** Implement quick sort that sorts N random numbers using different pivot elements based on CTRL variable. Your program takes two inputs N and CTRL variables with possible values as 1 < N < 10000 and CTRL = {1: pivot as last element, 2: pivot as first element, 3: pivot as middle element and 4: random pivot}. Comment on the asymptotic analysis of quick sort for the case when last element is taken as pivot. What is the running time of quick sort when elements are already sorted and when all elements are same?

3. **Sorting Algorithms.** Write a C program to implement the following sorting algorithms over a given set of N random integers:-
   a. Radix sort
   b. Shell sort
   c. Selection sort
   d. Insertion sort

4.12 Week-12: Handling Files

1. **File Processing.** Write a program to read a text file as input and count number of characters, words and lines in the file.

2. **File Copying.** Write a program to copy a source text file into a target text file.

3. **Improved file copying.** Modify the file copy program to avoid over writing the existing target file, instead if target file has some contents, then target file is appended by contents of source file.

4. **File handling API.** Write a program to read and write a file using following combinations of functions:-
   a. fgetc( ) and fputc( )
   b. fprintf( ) and fscanf( )
   c. fgets( ) and fputs( )
   d. fread( ) and fwrite( )

Appendix A: Setting up programming environment

Aim:

Setting up Linux environment in Windows Operating System using Virtual Box Software

Step 1: Installation of Virtual Box

1. Download Virtual Box Software
http://download.virtualbox.org/virtualbox/4.1.2/VirtualBox-4.1.2-73507-Win.exe

2. Run the .exe file downloaded in Step 1 and install Virtual Box Software using the default options appearing during the installation procedure.

Note: By default, Virtual Box will install in C: drive, recommended free space that should be available in C: drive is around 15GB, if you don't have this much free space, select a different drive during the Virtual Box installation.

Step 2: Creating a Virtual Machine and installing Guest Operating System (ubuntu-10.04)

1. Download guest operating system (ubuntu-10.04 desktop i386)
http://www.psychocats.net/ubuntu/getting

2. Follow the steps in link below for creating virtual machine and installing guest operating system
http://www.psychocats.net/ubuntu/virtualbox

Step 3: Installing Virtual Box Guest Additions

1. Download virtual box guest additions
http://download.virtualbox.org/virtualbox/4.1.2/VBoxGuestAdditions_4.1.2.iso

2. Attach the guest addition iso file to CD/DVD device as explained in Step 2's 2. Next follow steps are given below
http://helpdeskgeek.com/linux-tips/install-virtualbox-guest-additions-in-ubuntu/

Step 4: Installing Software Packages (compiler, editor, etc.)

1. Go to Applications → Accessories → Terminal and type following at command prompt to install packages build-essential and vim editor.

```
sudo apt-get install build-essential
sudo apt-get install vim
```