Q.2 a. Write a program to find the addition of n numbers by recursion.

Answer:

```
#include<stdio.h>

int main(){

    int n,sum;

    printf("Enter the value of n: ");
    scanf("%d",&n);

    sum = getSum(n);

    printf("Sum of n numbers: %d",sum);

    return 0;
}

int getSum(n){

    static int sum=0;

    if(n>0){
         sum = sum + n;
         getSum(n-1);
    }

    return sum;
}
```

b. Differentiate between malloc and calloc.

Answer:

Both the malloc() and the calloc() functions are used to allocate dynamic memory. Each operates slightly different from the other. malloc() takes a size and returns a pointer to a chunk of memory at least that big:

```
void *malloc( size_t size );
```

calloc() takes a number of elements, and the size of each, and returns a pointer to a chunk of memory
at least big enough to hold them all:

```
void *calloc( size_t numElements, size_t sizeOfElement );
```

There?s one major difference and one minor difference between the two functions. The major difference is that malloc() doesn?t initialize the allocated memory. The first time

malloc() gives you a particular chunk of memory, the memory might be full of zeros. If memory has been allocated, freed, and reallocated, it probably has whatever junk was left in it. That means, unfortunately, that a program might run in simple cases (when memory is never reallocated) but break when used harder (and when memory is reused). calloc() fills the allocated memory with all zero bits. That means that anything there you?re going to use as a char or an int of any length, signed or unsigned, is guaranteed to be zero. Anything you?re going to use as a pointer is set to all zero bits. That?s usually a null pointer, but it?s not guaranteed.Anything you?re going to use as a float or double is set to all zero bits; that?s a floating-point zero on some types of machines, but not on all.
The minor difference between the two is that calloc() returns an array of objects; malloc() returns one object. Some people use calloc() to make clear that they want an array.

c. What are advantages and disadvantages of external storage class?

Answer:

Advantages of external storage class

1) Persistent storage of a variable retains the latest value

2) The value is globally available

Disadvantages of external storage class

1) The storage for an external variable exists even when the variable is not needed

2) The side effect may produce surprising output

3) Modification of the program is difficult

4) Generality of a program is affected

Q.3 a. What is the main difference between STRUCTURE and UNION?

Answer:

1) Structure: The size in bytes is the sum total of size of all the elements in the structure, plus padding bytes.
2) Size of in bytes of the union is size of the largest variable element in the union.

i.e In case of Union, the elements making up the union 'overlap' in memory OR they are accessed as different name/type at different places in the program.
Whereas in case of Struct, each of the elements have a distinct identity.

b. Explain the four major operations carried out on the sequential files.

Answer: Page Number 147-148 of Text Book

c. With the help of an example, explain how is memory allocated to a structure.

Answer: Page Number 140-141 of Text Book

Q.4 a. What is meant by row major order and column major order?

Answer:

The array may be stored in memory one of the following way:-

1. Column by column i,e column major order
2. Row by row, i.e in row major order. The following figure shows both representation of the above array.

By row-major order, we mean that the elements in the array are so arranged that the subscript at the extreme right varies fast than the subscript at it's left., while in column-major order , the subscript at the extreme left changes rapidly , then the subscript at it's right and so on.

1,1
2,1
1,2
2,2
Column Major Order
1,1
1,2
2,1
2,2

b. Write an algorithm of linear search.

Answer:

```
LinearSearch(value, list)
   if the list is empty, return Λ;
   else
     if the first item of the list has the desired value, return its location;
     else return LinearSearch(value, remainder of the list)
```

c. With the help of an example, describe the merge sort technique.

Answer:

Mergesort algorithm is based on a divide and conquer strategy. First, the sequence to be sorted is decomposed into two halves (Divide). Each half is sorted independently (Conquer). Then the two sorted halves are merged to a sorted sequence (Combine)

```
void mergesort(int lo, int hi)
{
    if (lo<hi)
    {
        int m=(lo+hi)/2;
        mergesort(lo, m);
        mergesort(m+1, hi);
        merge(lo, m, hi);
    }
 }
```

Q.5 a. Describe the various applications of stack and queues.

Answer:

Stack and queue applications.

Stacks and queues have numerous useful applications.

Queue applications: Computing applications: serving requests of a single shared resource (printer, disk, CPU), transferring data asynchronously (data not necessarily received at same rate as sent) between two processes (IO buffers), e.g., pipes, file IO, sockets. Buffers on MP3 players and portable CD players, iPod playlist. Playlist for jukebox - add songs to the end, play from the front of the list. Interrupt handling: When programming a real-time system that can be interrupted (e.g., by a mouse click or wireless connection), it is necessary to attend to the interrupts immediately, before proceeding with the current activity. If the interrupts should be handles in the same order they arrive, then a FIFO queue is the appropriate data structure.

Arithmetic expression evaluation. Program Evaluate.java evaluates a fully parenthesized arithmetic expression.

An important application of stacks is in parsing. For example, a compiler must parse arithmetic expressions written using infix notation. For example the following infix expression evaluates to 212.

$$2 + ((3 + 4) * (5 * 6)))$$

We break the problem of parsing infix expressions into two stages. First, we convert from infix to a different representation called postfix. Then we parse the postfix expression, which is a somewhat easier problem than directly parsing infix.

b. Write the algorithm for converting an infix expression to postfix, using a stack.

Answer:

Infix to Postfix Conversion:

In normal algebra we use the infix notation like a+b*c. The corresponding postfix notation is abc*+. The algorithm for the conversion is as follows:

Scan the Infix string from left to right.

Initialise an empty stack.

If the scannned character is an operand, add it to the Postfix string. If the scanned character is an operator and if the stack is empty Push the character to stack.

If the scanned character is an Operand and the stack is not empty, compare the precedence of the character with the element on top of the stack (topStack). If topStack has higher precedence over the scanned character Pop the stack else Push the scanned character to stack. Repeat this step as long as stack is not empty and topStack has precedence over the character.

Repeat this step till all the characters are scanned.

(After all characters are scanned, we have to add any character that the stack may have to the Postfix string.) If stack is not empty add topStack to Postfix string and Pop the stack. Repeat this step as long as stack is not empty.
Return the Postfix string.

c. What is the limitation of an array implementation of a queue? How is it overcome?

Answer: Page Number 244 of Text Book

Q.6 a. Explain the singly linked list and write an algorithm to insert an element in the beginning of a singly linked list.

Answer:
Inserting at the beginning of the list requires a separate function. This requires updating firstNode.

```
function insertBeginning(List list, Node newNode) // insert node before current first node
    newNode.next   := list.firstNode
    list.firstNode := newNode
```

b. With the help of an example, explain how a linked list can be sorted.

Answer: Page Number 269-274 of Text Book

Q.7 a. With the help of an example differentiate between singly and doubly linked list.

Answer:
Each element in the singly linked list contains a reference to the next element in the list, while each element in the doubly linked list contains references to the next element as well as the previous element in the list. Doubly linked lists require more space for each element in the list and elementary operations such as insertion and deletion is more complex since they have to deal with two references. But doubly link lists allow easier manipulation since it allows traversing the list in forward and backward directions.

Each element in a singly linked list has two fields as shown in Figure 1. The data field holds the actual data stored and the next field holds the reference to the next element in the chain. The first element of the linked list is stored as the head of the linked list.

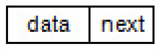


Figure 1: Element of a singly linked list

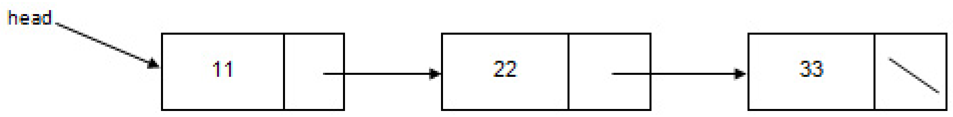


Figure 2: A singly linked list

Figure 2 depicts a singly linked list with three elements. Each element stores its data and all elements except the last one store a reference to the next element. Last element holds a null value in its next field. Any element in the list can be accessed by starting at the head and following the next pointer until you meet the required element.

Each element in a doubly linked list has three fields as shown in Figure 3. Similar to singly linked list, the data field holds the actual data stored and the next field holds the reference to the next element in the chain. Additionally, the previous field holds the reference to the previous element in the chain. The first element of the linked list is stored as the head of the linked list.

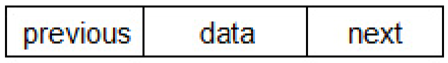


Figure 3: Element of a doubly linked list

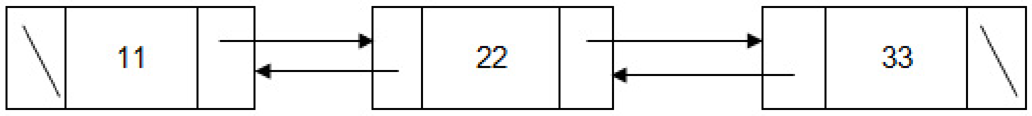
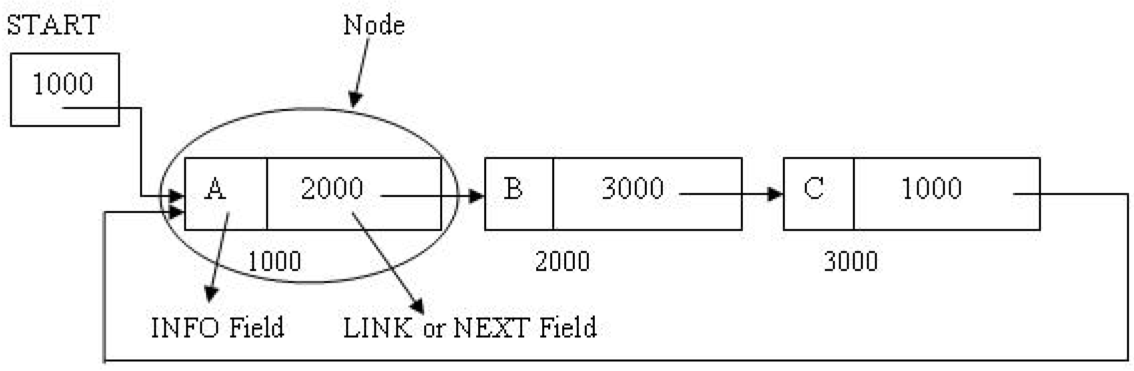


Figure 4: A doubly linked list

b. What is Circular Linked List? What are Advantages and Disadvantages of Circular Linked List?

Answer:
In it the last node does not contain NULL pointer. Instead the last node contains a pointer that has the address of first node and thus points back to the first node.

It is shown below:



Advantages:

1. If we are at a node, then we can go to any node. But in linear linked list it is not possible to

go to previous node.
2. It saves time when we have to go to the first node from the last node. It can be done in single step because there is no need to traverse the in between nodes. But in double linked list, we will have to go through in between nodes.

Disadvantages:

1. It is not easy to reverse the linked list.
2. If proper care is not taken, then the problem of infinite loop can occur.
3. If we at a node and go back to the previous node, then we can not do it in single step. Instead we have to complete the entire circle by going through the in between nodes and then we will reach the required node.

Q.8 a. Construct a binary tree for the given:

In-order trasversal	= Q, A, Z, Y, P, C, X, B
Pre-order trasversal	= Z, A, Q, P, Y, X, C, B

Write the post order traversal for the created binary tree.

Answer: Page Number 363-364 of Text Book

b. In the given binary tree:

```
  14
  / \
 2   11
/ \  / \
1 3 10 30
  /  /
 7  40
```

Write the order of the nodes visited in:
(i) An in-order traversal
(ii) A pre-order traversal
(iii) A post-order traversal

Answer:
Inorder: 1 2 3 14 7 10 11 40 30
Preorder: 14 2 1 3 11 10 7 30 40
Postorder: 1 3 2 7 10 40 30 11

c. Explain the binary search tree.

Answer:
Binary Search tree is a binary tree in which each internal node x stores an element such that the element stored in the left subtree of x are less than or equal to x and elements stored in the right subtree of x are greater than or equal to x. This is called binary-search-tree property.

The basic operations on a binary search tree take time proportional to the height of the tree. For a complete binary tree with node n, such operations runs in Θ(lg n) worst-case time. If the tree is a linear chain of n nodes, however, the same operations takes (n) worst-case time.
The height of the Binary Search Tree equals the number of links from the root node to the deepest node.
Implementation of Binary Search Tree
Binary Search Tree can be implemented as a linked data structure in which each node is an object with three pointer fields. The three pointer fields left, right and p point to the nodes corresponding to the left child, right child and the parent respectively NIL in any pointer field signifies that there exists no corresponding child or parent. The root node is the only node in the BTS structure with NIL in its p field.

Q.9 a. Write the algorithm for a DFS traversal of a graph.

Answer:

Depth-first search (DFS) is an algorithm for traversing or searching a tree, tree structure, or graph. One starts at the root (selecting some node as the root in the graph case) and explores as far as possible along each branch before backtracking.

```
procedure DFS(G,v):
    label v as explored
    for all edges e in G.incidentEdges(v) do
        if edge e is unexplored then
            w ← G.opposite(v,e)
            if vertex w is unexplored then
                label e as a discovery edge
                recursively call DFS(G,w)
        else
            label e as a back edge
```

b. How the graphs are represented in the memory?

Answer:

It is possible to represent graphs in computer memory with a variety of different data structures. One strategy is to use an adjacency matrix. An adjacency matrix is a two dimensional array in which the row and column headers represent different vertexes in the graph. A one-way edge between, for example, vertex one and three, is denoted by a positive value in array position (1, 3). In a weighted graph the value stored in each array location corresponds to the weight or cost of each particular edge. If an edge is bi-directional it has two entries in the matrix. One entry represents the (source, destination) route while the other handles the (destination, source) return route.

Another method for representing graphs is as a more complicated linked list structure. Each vertex in the graph is a node in a master linked list. Another linked list emanates from each vertex node and denotes the vertexes directly adjacent to a given source vertex. This method, often called an adjacency list, is more space efficient than the adjacency matrix for graphs which do not have very many edges .

c. Explain the minimal cost spanning tree.

Answer:

Given a connected, undirected graph, a spanning tree of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. We can also assign a weight to each edge, which is a number representing how unfavorable it is, and use this to assign a weight to a spanning tree by computing the sum of the weights of the edges in that spanning tree. A minimum spanning tree (MST) or minimum weight spanning tree is then a spanning tree with weight less than or equal to the weight of every other spanning tree. More generally, any undirected graph (not necessarily connected) has a minimum spanning forest, which is a union of minimum spanning trees for its connected components.

Text Book

C& Data Structures, P.S. Deshpande and O.G. Kakde, Dreamtech Press, 2007