Institute of Computer Technology

B. Tech. Computer Science and Engineering

Semester: III

Sub: Data Structure

Course Code: 2CSE302

Assignment - 1

1. How is the following 3*4 matrix is represented in the memory in Column major order representation? How can you find the memory address of the element 8, if the base address is 2050?

$$M = \begin{bmatrix} 5 & 7 & 3 & 9 \\ 2 & 3 & 8 & 6 \\ 4 & 2 & 4 & 1 \end{bmatrix}$$

Ans.

- In column-major order, elements are stored column by column: [5, 2, 4, 7, 3, 2, 3, 8, 4, 9, 6, 1].

To find the memory address of element 8, we use the formula:
Loc(arr[i][j])=base(arr)+w(m*j+i)

Where:- Base Address:-2050

W=1 Byte

M=3 rows

Element 8 at position i=2,j=2

Thus The answer is:- ⌊2050+1(3*2+2)=2058

2. Explain the role of stack in finding GCD of two numbers using recursion.

Ans.

In recursion, each function call stores its state in the stack. In the Euclidean algorithm, each recursive call pushes onto the stack until the base case is reached (b = 0), then the stack unwinds, returning the GCD.

3. Consider a stack of size 10. What will be the final value of stack after the following push and pop operations?
   Push(A), Push(B), Push(C), Pop(), Pop(),
   Push(D),Push(E),Pop() .

Ans.

Step-by-step:

- Push A: [A]
- Push B: [A, B]
- Push C: [A, B, C]
- Pop(): [A, B] (C is removed)
- Pop(): [A] (B is removed)
- Push D: [A, D]
- Push E: [A, D, E]
- Pop(): [A, D] (E is removed)

Final Stack: [A, D]

4. Convert the following postfix expressions into infix and prefix. Postfix: AB-CD/E*+
   Postfix: ABC*E/+F-G+

Ans.

- Postfix: AB-CD/E+*
  - Infix: (A−B)+((C/D)∗E)(A - B) + ((C / D) * E)(A−B)+((C/D)∗E)
  - Prefix: +−AB∗/CDE+ - AB * / CDE+−AB∗/CDE

- Postfix: ABC*E/+F-G+
  - Infix: ((A+(B∗C))/E)+F−G((A + (B * C)) / E) + F - G((A+(B∗C))/E)+F−G
  - Prefix: +/+A∗BCEF−G+ / + A * BCEF - G+/+A∗BCEF−G

5. Discuss the difference between Simple, Circular, Deque and Priority Queue.

- Simple Queue: Basic FIFO (First In, First Out) structure.

- Circular Queue: Last element is connected to the first, allowing efficient reuse of memory.
- Deque (Double-Ended Queue): Insert and remove elements from both ends.
- Priority Queue: Elements are dequeued based on their priority, not just order of insertion.

6. How is insertion and deletion operations are performed in Priority Queue?

Ans.

- Insertion: Insert an element based on its priority into the correct position.
- Deletion: Remove the element with the highest priority (or lowest, depending on implementation).

7. Discuss the need of Circular and Doubly linked list over singly linked list.

Ans.

- Circular Linked List: Efficient for applications that need a circular traversal (e.g., round-robin scheduling).
- Doubly Linked List: Allows traversal in both directions and efficient deletion/insertion at both ends, which is harder in singly linked lists.

8. Write down the algorithm to count total number of nodes in doubly linked list.

Ans.

- Initialize a counter:
  - Set count = 0. This will store the number of nodes in the linked list.
- Start at the head node:
  - Create a pointer current and assign it to point to the head node of the linked list.

- Traverse through the list:
  - While current is not NULL (i.e., until you reach the end of the list):
    1. Increment the counter (count++).
    2. Move to the next node in the list by updating current = current->next.

- Return the count:
  - Once the loop ends (when current becomes NULL), return the

value of count, which now holds the total number of nodes in the list.

9. Write down the algorithm to reverse the given singly linked list.

Ans.

- Initialize three pointers:
  - prev (previous node) to NULL.
  - current (current node) to the head of the list.
  - next (next node) to NULL (this will store the next node temporarily).

- Traverse through the list:
  - While current is not NULL:
    1. Store the next node: next = current->next.
    2. Reverse the current node's pointer: current->next = prev.
    3. Move prev and current one step forward:
       - prev = current
       - current = next

- Set the head to the new head (prev):
  - Once the loop ends, prev will point to the new head of the reversed list.
  - Set head = prev.

10. Write down the algorithm to delete specific node from singly circular linked list by position.

Ans.

- Check if the list is empty:
  - If head == NULL, the list is empty, and nothing can be deleted.

- Handle deletion of the head node (position = 1):

- If the list has only one node (head->next == head), set head = NULL.
- If the list has more than one node:
  1. Traverse the list to find the last node.
  2. Update the last node's next pointer to the node after head.
  3. Set head = head->next.

➢ Handle deletion of a node at a given position (position > 1):
- Traverse the list until you reach the node just before the target position.
- Update the next pointer of the previous node to skip the node at the target position.

➢ Free the memory of the deleted node:
- Free the node being deleted to release memory.

➢ End.

11. How linked list is used to represent tree data structure? Explain with example.

Ans.

➢ In a tree data structure, each node can have multiple children, whereas in a linked list, each node generally has a reference to only one node. To represent a tree using a linked list in C, the key idea is to use a linked list of child pointers for each node.

➢ For example, in a binary tree, each node contains:
- A data value.
- A pointer to the left child.
- A pointer to the right child.

➢ For a generic tree (where nodes can have more than two children), we can extend this idea by using a linked list for the children of each node. This approach is often referred to as a first-child/next-sibling representation. In this representation:

- Each node points to its first child.

- Each node also points to its next sibling.
- Structure of a Node in a Tree (Using Linked List in C)
- For a binary tree, the node structure looks like this:

➢ **Code:-**

```
struct TreeNode {
 int data;
 struct TreeNode* left; // Pointer to left child
 struct TreeNode* right; // Pointer to right child
};
```

➢ For a general tree, the structure looks like this:

Code:-

```
struct TreeNode {
 int data;
 struct TreeNode* firstChild; // Pointer to the first child
 struct TreeNode* nextSibling; // Pointer to the next sibling
};
```

12. Suppose you want to store some integer numbers and want to retrieve them for processing in reverse order of their appearance, then which data structure is suitable and why?

Ans.

➢ The Stack (LIFO - Last In, First Out) is suitable for this task

because it retrieves elements in the reverse order of their insertion.

13. Implement the concept of stack (Last in First Out) using singly linked list.

Ans.

- **Code:-**

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
 int data;
 struct Node* next;
};
struct Stack {
 struct Node* top; // Pointer to the top node of the stack
};
struct Node* createNode(int data) {
 struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
 newNode->data = data;
 newNode->next = NULL;
 return newNode;
}
struct Stack* createStack() {
 struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
 stack->top = NULL;
 return stack;
}
void push(struct Stack* stack, int data) {
 struct Node* newNode = createNode(data);
 newNode->next = stack->top; current top
```

```
 stack->top = newNode;
 printf("%d pushed to stack\n", data);
}
int pop(struct Stack* stack) {
 if (stack->top == NULL) {
 printf("Stack underflow\n");
 return -1;
 }
 struct Node* poppedNode = stack->top;
 stack->top = stack->top->next;
 int poppedData = poppedNode->data;
 free(poppedNode);
 return poppedData;
}
int isEmpty(struct Stack* stack) {
 return stack->top == NULL;
}
void display(struct Stack* stack) {
 struct Node* current = stack->top;
 if (isEmpty(stack)) {
 printf("Stack is empty\n");
 return;
 }
 printf("Stack elements: ");
 while (current != NULL) {
 printf("%d -> ", current->data);
 current = current->next;
 }
 printf("NULL\n");
```

```
}
int main() {
 struct Stack* stack = createStack();
 push(stack, 10);
 push(stack, 20);
 push(stack, 30);
 display(stack);
 printf("%d popped from stack\n", pop(stack));
 display(stack);
 return 0;
}
```