DAY 3

C PROGRAMMING – DATA STRUCTURES

1. Write a c program to implement the SINGLY LINKED LIST with the following operations:
   a. Insert an element into the list [beginning, middle, last].

```c
#include <stdio.h>
#include <stdlib.h>

// Structure for a node in the linked list
struct Node {
    int data;
    struct Node *next;
};

// Function to insert a new node at the beginning of the linked list
struct Node *insertAtBeginning(struct Node *head, int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = head;
    return newNode;
}

// Function to insert a new node at the middle of the linked list
struct Node *insertAtMiddle(struct Node *head, int value, int position) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;

    struct Node *current = head;
    for (int i = 1; i < position - 1 && current != NULL; ++i) {
        current = current->next;
    }

    if (current == NULL) {
        printf("Invalid position for insertion.\n");
        free(newNode);
        return head;
    }

    newNode->next = current->next;
    current->next = newNode;

    return head;
}

// Function to insert a new node at the end of the linked list
struct Node *insertAtEnd(struct Node *head, int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
```

```c
    newNode->data = value;
    newNode->next = NULL;

    if (head == NULL) {
        return newNode;
    }

    struct Node *current = head;
    while (current->next != NULL) {
        current = current->next;
    }

    current->next = newNode;
    return head;
}

// Function to display the linked list
void displayList(struct Node *head) {
    struct Node *current = head;
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node *head = NULL;

    // Insert at the beginning
    head = insertAtBeginning(head, 20);
    head = insertAtBeginning(head, 10);
    printf("Linked List after inserting at the beginning: ");
    displayList(head);

    // Insert at the middle
    head = insertAtMiddle(head, 15, 2);
    printf("Linked List after inserting at the middle: ");
    displayList(head);

    // Insert at the end
    head = insertAtEnd(head, 30);
    printf("Linked List after inserting at the end: ");
    displayList(head);

    return 0;
}
```

input

```
Linked List after inserting at the beginning: 10 -> 20 -> NULL
Linked List after inserting at the middle: 10 -> 15 -> 20 -> NULL
Linked List after inserting at the end: 10 -> 15 -> 20 -> 30 -> NULL
```

b. Delete an element into the list [beginning, middle, last].

main.c

```c
#include <stdio.h>
#include <stdlib.h>

// Structure for a node in the linked list
struct Node {
    int data;
    struct Node *next;
};

// Function to insert a new node at the beginning of the linked list
struct Node *insertAtBeginning(struct Node *head, int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = head;
    return newNode;
}

// Function to delete a node from the beginning of the linked list
struct Node *deleteFromBeginning(struct Node *head) {
    if (head == NULL) {
        printf("List is already empty.\n");
        return NULL;
    }

    struct Node *temp = head;
    head = head->next;
    free(temp);

    return head;
}

// Function to delete a node from the middle of the linked list
struct Node *deleteFromMiddle(struct Node *head, int position) {
    if (head == NULL) {
        printf("List is empty.\n");
        return NULL;
    }

    struct Node *prev = NULL;
    struct Node *current = head;

    for (int i = 1; i < position && current != NULL; ++i) {
```

```
    for (int i = 1; i < position && current != NULL; ++i) {
        prev = current;
        current = current->next;
    }

    if (current == NULL) {
        printf("Invalid position for deletion.\n");
        return head;
    }

    if (prev != NULL) {
        prev->next = current->next;
        free(current);
    } else {
        head = current->next;
        free(current);
    }

    return head;
}

```

```
Linked List after inserting at the beginning: 10 -> 20 -> 30 -> NULL
Linked List after deleting from the beginning: 20 -> 30 -> NULL
Linked List after deleting from the middle: 20 -> NULL
Linked List after deleting from the end: NULL
```

c. Search for an element in the list.

```c
#include <stdio.h>
#include <stdlib.h>

// Structure for a node in the linked list
struct Node {
    int data;
    struct Node *next;
};

// Function to insert a new node at the beginning of the linked list
struct Node *insertAtBeginning(struct Node *head, int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = head;
    return newNode;
}

// Function to search for an element in the linked list
int searchElement(struct Node *head, int target) {
    struct Node *current = head;
    int position = 1;
```

```c
    while (current != NULL) {
        if (current->data == target) {
            return position;
        }
        current = current->next;
        position++;
    }

    return -1; // Element not found
}

// Function to display the linked list
void displayList(struct Node *head) {
    struct Node *current = head;
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node *head = NULL;

    // Insert elements at the beginning
    head = insertAtBeginning(head, 30);
    head = insertAtBeginning(head, 20);
    head = insertAtBeginning(head, 10);

    printf("Linked List: ");
    displayList(head);

    int target;

    printf("Enter the element to search: ");
    scanf("%d", &target);

    int position = searchElement(head, target);

    if (position != -1) {
        printf("Element %d found at position %d.\n", target, position);
    } else {
        printf("Element %d not found in the linked list.\n", target);
    }

    return 0;
}
```

```
Linked List: 10 -> 20 -> 30 -> NULL
Enter the element to search: 30
Element 30 found at position 3.
```

d. Display the elements.

```c
#include <stdio.h>
#include <stdlib.h>

// Structure for a node in the linked list
struct Node {
    int data;
    struct Node *next;
};

// Function to insert a new node at the beginning of the linked list
struct Node *insertAtBeginning(struct Node *head, int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = head;
    return newNode;
}

// Function to display the elements in the linked list
void displayList(struct Node *head) {
    struct Node *current = head;
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node *head = NULL;

    // Insert elements at the beginning
    head = insertAtBeginning(head, 30);
    head = insertAtBeginning(head, 20);
    head = insertAtBeginning(head, 10);

    printf("Linked List: ");
    displayList(head);

    return 0;
}
```

```
Linked List: 10 -> 20 -> 30 -> NULL
```

2. Write a c program to implement the stack data structure with the following.
   a. Pop an element into the list [beginning, middle, last].

```c
#include <stdio.h>
#include <stdlib.h>

#define MAX_SIZE 100

// Structure for a stack
struct Stack {
    int items[MAX_SIZE];
    int top;
};

// Function to initialize an empty stack
void initialize(struct Stack *stack) {
    stack->top = -1;
}

// Function to check if the stack is empty
int isEmpty(struct Stack *stack) {
    return stack->top == -1;
}

```

```c
// Function to check if the stack is full
int isFull(struct Stack *stack) {
    return stack->top == MAX_SIZE - 1;
}

// Function to push an item onto the stack
void push(struct Stack *stack, int item) {
    if (isFull(stack)) {
        printf("Stack overflow\n");
        exit(1);
    }
    stack->items[++stack->top] = item;
}

// Function to display the stack
void displayStack(struct Stack *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty.\n");
        return;
    }
    printf("Stack contents: ");
```

```
    printf("Stack contents: ");
    for (int i = 0; i <= stack->top; ++i) {
        printf("%d ", stack->items[i]);
    }
    printf("\n");
}

int main() {
    struct Stack stack;
    initialize(&stack);

    int option, element;

    do {
        printf("\nStack Operations:\n");
        printf("1. Push at beginning\n");
        printf("2. Push at middle\n");
        printf("3. Push at last\n");
        printf("4. Display stack\n");
        printf("5. Quit\n");
        printf("Enter your choice: ");
        scanf("%d", &option);

        switch (option) {
            case 1:
                printf("Enter element to push: ");
                scanf("%d", &element);
                for (int i = stack.top; i >= 0; --i) {
                    stack.items[i + 1] = stack.items[i];
                }
                stack.items[0] = element;
                stack.top++;
                printf("Element %d pushed at the beginning.\n", element);
                break;
            case 2:
                if (isFull(&stack)) {
                    printf("Stack is full.\n");
                    break;
                }
                printf("Enter element to push: ");
                scanf("%d", &element);
```

```
            scanf("%d", &element);
            int position;
            printf("Enter position (1 to %d) to push at: ", stack.top + 2);
            scanf("%d", &position);
            if (position < 1 || position > stack.top + 2) {
                printf("Invalid position.\n");
                break;
            }
            for (int i = stack.top; i >= position - 1; --i) {
                stack.items[i + 1] = stack.items[i];
            }
            stack.items[position - 1] = element;
            stack.top++;
            printf("Element %d pushed at position %d.\n", element, position);
            break;
        case 3:
            printf("Enter element to push: ");
            scanf("%d", &element);
            stack.items[++stack.top] = element;
            printf("Element %d pushed at the end.\n", element);
            break;
        case 4:
            displayStack(&stack);
            break;
        case 5:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
} while (option != 5);

return 0;
}
```

```
Stack Operations:
1. Push at beginning
2. Push at middle
3. Push at last
4. Display stack
5. Quit
Enter your choice: 1
Enter element to push: 2
Element 2 pushed at the beginning.

Stack Operations:
1. Push at beginning
2. Push at middle
3. Push at last
4. Display stack
5. Quit
Enter your choice: 2
Enter element to push: 3
Enter position (1 to 2) to push at: 2
Element 3 pushed at position 2.

Stack Operations:
1. Push at beginning
2. Push at middle
```

```
3. Push at last
4. Display stack
5. Quit
Enter your choice: 3
Enter element to push: 4
Element 4 pushed at the end.

Stack Operations:
1. Push at beginning
2. Push at middle
3. Push at last
4. Display stack
5. Quit
Enter your choice: 4
Stack contents: 2 3 4

Stack Operations:
1. Push at beginning
2. Push at middle
3. Push at last
4. Display stack
5. Quit
Enter your choice: 5
Exiting...
```

b. Search for an element in the stack.

```c
#include <stdio.h>
#include <stdlib.h>

#define MAX_SIZE 100

// Structure for a stack
struct Stack {
    int items[MAX_SIZE];
    int top;
};

// Function to initialize an empty stack
void initialize(struct Stack *stack) {
    stack->top = -1;
}

// Function to check if the stack is empty
int isEmpty(struct Stack *stack) {
    return stack->top == -1;
}
```

```c
// Function to check if the stack is full
int isFull(struct Stack *stack) {
    return stack->top == MAX_SIZE - 1;
}

// Function to push an item onto the stack
void push(struct Stack *stack, int item) {
    if (isFull(stack)) {
        printf("Stack overflow\n");
        exit(1);
    }
    stack->items[++stack->top] = item;
}

// Function to pop an item from the stack
int pop(struct Stack *stack) {
    if (isEmpty(stack)) {
        printf("Stack underflow\n");
        exit(1);
    }
```

```
42      return stack->items[stack->top--];
43  }
44
45  // Function to search for an element in the stack
46  int search(struct Stack *stack, int target) {
47      for (int i = stack->top; i >= 0; --i) {
48          if (stack->items[i] == target) {
49              return i;
50          }
51      }
52      return -1; // Element not found
53  }
54
55  int main() {
56      struct Stack stack;
57      initialize(&stack);
58
59      int option, element, target;
60
61      do {
```

```
62          printf("\nStack Operations:\n");
63          printf("1. Push\n");
64          printf("2. Pop\n");
65          printf("3. Search\n");
66          printf("4. Display stack\n");
67          printf("5. Quit\n");
68          printf("Enter your choice: ");
69          scanf("%d", &option);
70
71          switch (option) {
72              case 1:
73                  printf("Enter element to push: ");
74                  scanf("%d", &element);
75                  push(&stack, element);
76                  printf("Element %d pushed onto the stack.\n",
                        element);
77                  break;
78              case 2:
79                  if (isEmpty(&stack)) {
80                      printf("Stack is empty.\n");
```

```
81                  break;
82              }
83              element = pop(&stack);
84              printf("Element %d popped from the stack.\n",
                    element);
85              break;
86          case 3:
87              printf("Enter element to search: ");
88              scanf("%d", &target);
89              int position = search(&stack, target);
90              if (position != -1) {
91                  printf("Element %d found at position %d
                        from top of the stack.\n", target,
                        stack.top - position + 1);
92              } else {
93                  printf("Element %d not found in the stack
                        .\n", target);
94              }
95              break;
96          case 4:
97              printf("Stack contents: ");
98              for (int i = stack.top; i >= 0; --i) {
99                  printf("%d ", stack.items[i]);
100             }
101             printf("\n");
102             break;
103         case 5:
104             printf("Exiting...\n");
105             break;
106         default:
107             printf("Invalid choice. Please try again.\n");
108     }
109 } while (option != 5);
110
111 return 0;
112 }
```

```
/tmp/1PfCkQSRW1.o
Stack Operations:
1. Push
2. Pop
3. Search
4. Display stack
5. Quit
Enter your choice: 1
Enter element to push: 3
Element 3 pushed onto the stack.

Stack Operations:
1. Push
2. Pop
3. Search
4. Display stack
5. Quit
Enter your choice: 1
Enter element to push: 4
Element 4 pushed onto the stack.
```

```
Stack Operations:
1. Push
2. Pop
3. Search
4. Display stack
5. Quit
Enter your choice: 1
Enter element to push: 5
Element 5 pushed onto the stack.

Stack Operations:
1. Push
2. Pop
3. Search
4. Display stack
5. Quit
Enter your choice: 4
Stack contents: 5 4 3
```

```
Stack Operations:
1. Push
2. Pop
3. Search
4. Display stack
5. Quit
Enter your choice: 3
Enter element to search: 4
Element 4 found at position 2 from top of the stack.

Stack Operations:
1. Push
2. Pop
3. Search
4. Display stack
5. Quit
Enter your choice: 5
Exiting...
```

c. Display the stack.

```c
#include <stdio.h>
#include <stdlib.h>

#define MAX_SIZE 100

// Structure for a stack
struct Stack {
    int items[MAX_SIZE];
    int top;
};

// Function to initialize an empty stack
void initialize(struct Stack *stack) {
    stack->top = -1;
}

// Function to check if the stack is empty
int isEmpty(struct Stack *stack) {
    return stack->top == -1;
}

```

```c
// Function to check if the stack is full
int isFull(struct Stack *stack) {
    return stack->top == MAX_SIZE - 1;
}

// Function to push an item onto the stack
void push(struct Stack *stack, int item) {
    if (isFull(stack)) {
        printf("Stack overflow\n");
        exit(1);
    }
    stack->items[++stack->top] = item;
}

// Function to pop an item from the stack
int pop(struct Stack *stack) {
    if (isEmpty(stack)) {
        printf("Stack underflow\n");
        exit(1);
    }
    return stack->items[stack->top--];
}

// Function to display the stack
void displayStack(struct Stack *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty.\n");
        return;
    }
    printf("Stack contents: ");
    for (int i = stack->top; i >= 0; --i) {
        printf("%d ", stack->items[i]);
    }
    printf("\n");
}

int main() {
    struct Stack stack;
    initialize(&stack);

    int option, element;
```

```
63
64      do {
65          printf("\nStack Operations:\n");
66          printf("1. Push\n");
67          printf("2. Pop\n");
68          printf("3. Display stack\n");
69          printf("4. Quit\n");
70          printf("Enter your choice: ");
71          scanf("%d", &option);
72
73          switch (option) {
74              case 1:
75                  printf("Enter element to push: ");
76                  scanf("%d", &element);
77                  push(&stack, element);
78                  printf("Element %d pushed onto the stack.\n", element);
79                  break;
80              case 2:
81                  if (isEmpty(&stack)) {
82                      printf("Stack is empty.\n");
83                      break;
84                  }
85                  element = pop(&stack);
86                  printf("Element %d popped from the stack.\n", element);
87                  break;
88              case 3:
89                  displayStack(&stack);
90                  break;
91              case 4:
92                  printf("Exiting...\n");
93                  break;
94              default:
95                  printf("Invalid choice. Please try again.\n");
96          }
97      } while (option != 4);
98
99      return 0;
100 }
101
102
103
```

```
Stack Operations:
1. Push
2. Pop
3. Display stack
4. Quit
Enter your choice: 1
Enter element to push: 2
Element 2 pushed onto the stack.

Stack Operations:
1. Push
2. Pop
3. Display stack
4. Quit
Enter your choice: 1
Enter element to push: 3
Element 3 pushed onto the stack.

Stack Operations:
1. Push
2. Pop
3. Display stack
4. Quit
Enter your choice: 1
```

tab-stdin

```
Enter your choice: 1
Enter element to push: 4
Element 4 pushed onto the stack.

Stack Operations:
1. Push
2. Pop
3. Display stack
4. Quit
Enter your choice: 2
Element 4 popped from the stack.

Stack Operations:
1. Push
2. Pop
3. Display stack
4. Quit
Enter your choice: 3
Stack contents: 3 2

Stack Operations:
1. Push
2. Pop
3. Display stack
4. Quit
Enter your choice: 4
Exiting...
```

3. Write a c program to implement queue data structure with the following operations.

a. Enqueue

b. Dequeue

c. Display

```c
#include <stdio.h>
#include <stdlib.h>

#define MAX_SIZE 100

// Structure for the queue
struct Queue {
    int items[MAX_SIZE];
    int front;
    int rear;
};

// Function to initialize a new queue
void initialize(struct Queue* queue) {
    queue->front = -1;
    queue->rear = -1;
}

// Function to check if the queue is empty
int isEmpty(struct Queue* queue) {
    return (queue->front == -1);
}

// Function to check if the queue is full
int isFull(struct Queue* queue) {
    return (queue->rear == MAX_SIZE - 1);
}

// Function to enqueue an element into the queue
void enqueue(struct Queue* queue, int value) {
    if (isFull(queue)) {
        printf("Queue is full. Cannot enqueue.\n");
        return;
    } else if (isEmpty(queue)) {
        queue->front = queue->rear = 0;
    } else {
        queue->rear++;
    }

    queue->items[queue->rear] = value;
    printf("%d enqueued to the queue.\n", value);
}
```

```c

// Function to dequeue an element from the queue
int dequeue(struct Queue* queue) {
    int dequeuedValue;

    if (isEmpty(queue)) {
        printf("Queue is empty. Cannot dequeue.\n");
        return -1;
    } else {
        dequeuedValue = queue->items[queue->front];

        if (queue->front == queue->rear) {
            queue->front = queue->rear = -1;
        } else {
            queue->front++;
        }
    }

    return dequeuedValue;
}

```

```c
// Function to display the contents of the queue
void display(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return;
    }

    printf("Queue elements: ");
    for (int i = queue->front; i <= queue->rear; i++) {
        printf("%d ", queue->items[i]);
    }
    printf("\n");
}

int main() {
    struct Queue queue;
    initialize(&queue);

    // Enqueue some elements
    enqueue(&queue, 10);
    enqueue(&queue, 20);
```

```
    enqueue(&queue, 20);
    enqueue(&queue, 30);

    // Display the queue
    display(&queue);

    // Dequeue an element
    int dequeuedValue = dequeue(&queue);
    if (dequeuedValue != -1) {
        printf("Dequeued: %d\n", dequeuedValue);
    }

    // Display the queue again
    display(&queue);

    return 0;
}
```

```
10 enqueued to the queue.
20 enqueued to the queue.
30 enqueued to the queue.
Queue elements: 10 20 30
Dequeued: 10
Queue elements: 20 30
```

4. To convert infix to postfix using stack

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAX_SIZE 100

// Structure for a stack
struct Stack {
    char items[MAX_SIZE];
    int top;
};

// Function to initialize an empty stack
void initialize(struct Stack *stack) {
    stack->top = -1;
}

// Function to check if the stack is empty
int isEmpty(struct Stack *stack) {
    return stack->top == -1;
```

```c
}

// Function to push an item onto the stack
void push(struct Stack *stack, char item) {
    if (stack->top >= MAX_SIZE - 1) {
        printf("Stack overflow\n");
        exit(1);
    }
    stack->items[++stack->top] = item;
}

// Function to pop an item from the stack
char pop(struct Stack *stack) {
    if (isEmpty(stack)) {
        printf("Stack underflow\n");
        exit(1);
    }
    return stack->items[stack->top--];
}

// Function to get the precedence of an operator
int getPrecedence(char op) {
    switch (op) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
    }
    return 0;
}

// Function to convert infix to postfix expression
void infixToPostfix(char infix[], char postfix[]) {
    struct Stack stack;
    initialize(&stack);

    int i, j;
    i = j = 0;

    while (infix[i] != '\0') {
        if (isalnum(infix[i])) {
            postfix[j++] = infix[i];
        } else if (infix[i] == '(') {
            push(&stack, '(');
        } else if (infix[i] == ')') {
            while (!isEmpty(&stack) && stack.items[stack.top] != '(') {
                postfix[j++] = pop(&stack);
            }
            pop(&stack); // Pop '('
        } else {
            while (!isEmpty(&stack) && getPrecedence(infix[i]) <= getPrecedence(stack.items[stack.top])) {
                postfix[j++] = pop(&stack);
            }
            push(&stack, infix[i]);
        }
        i++;
    }

    while (!isEmpty(&stack)) {
```

```
    while (!isEmpty(&stack)) {
        postfix[j++] = pop(&stack);
    }

    postfix[j] = '\0';
}

int main() {
    char infix[MAX_SIZE], postfix[MAX_SIZE];

    printf("Enter an infix expression: ");
    scanf("%s", infix);

    infixToPostfix(infix, postfix);

    printf("Postfix expression: %s\n", postfix);

    return 0;
}
```

```
Enter an infix expression: ((a*b)/(c-d))
Postfix expression: ab*cd-/
```

5. To evaluate the given expression using stack.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#define MAX_SIZE 100

typedef struct{
    int items[MAX_SIZE];
    int top;
}Stack;

void push(Stack*stack,int value){
    if(stack->top == MAX_SIZE-1){
        printf("Stack overflow\n");
        exit(1);
    }
    stack->top++;
    stack->items[stack->top]=value;
}

int pop(Stack*stack){
```

```
    if(stack->top==-1){
        printf("Stack underflow\n");
        exit(1);
    }
    int value=stack->items[stack->top];
    stack->top--;
    return value;
}


int evaluateExpression(char*expression){
    Stack stack;
    stack.top=-1;

for(int i=0;i<strlen(expression);i++){
    if(expression[i]>='0'&& expression[i]<='9'){
        push(&stack,expression[i]-'0');
    }else{
        int operand2=pop(&stack);
        int operand1=pop(&stack);
        switch(expression[i]){
            case '+':
            push(&stack,operand1+operand2);
            break;
            case'-':
            push(&stack,operand1-operand2);
            break;
            case'*':
            push(&stack,operand1*operand2);
            break;
            case'/':
            push(&stack,operand1/operand2);
            break;
        }
    }
}
    return pop(&stack);
}

int main(){
    char expression[MAX_SIZE];
    printf("enter an arithemetic expression:");
    scanf("%s",expression);

    int result= evaluateExpression(expression);
    printf("result: %d\n",result);
    return 0;
}
```

```
enter an arithemetic expression:(a+b/c*(d-e))
Stack underflow
```