DAY3 LAB

1. Write a C program to implement Stack operations using array such as PUSH, POP and PEEK.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100
int stack[MAX_SIZE];
int top = -1;
void push(int value) {
   if (top == MAX_SIZE - 1) {
      printf("Stack Overflow\n");
      return;
   }
   stack[++top] = value;
   printf("%d pushed to stack\n", value);
}
void pop() {
   if (top == -1) {
      printf("Stack Underflow\n");
      return;
   }
   printf("%d popped from stack\n", stack[top--]);
}
int peek() {
   if (top == -1) {
      printf("Stack is empty\n");
      return -1;
   }
   return stack[top];
}
int main() {
```

```
    push(10);
    push(20);
    push(30);
    printf("Top element: %d\n", peek());
    pop();
    pop();
    pop();
    pop();
    return 0;
}
```

OUTPUT:

10 pushed to stack

20 pushed to stack

30 pushed to stack

Top element: 30

30 popped from stack

20 popped from stack

10 popped from stack

Stack Underflow

2. Write a C program to implement Stack operations using linked list such as PUSH, POP and PEEK.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* top = NULL;
void push(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = top;
    top = newNode;
    printf("%d pushed to the stack.\n", value);
}
```

```
void pop() {
   if (top == NULL) {
      printf("Stack is empty.\n");
      return;
   }
   struct Node* temp = top;
   top = top->next;
   printf("%d popped from the stack.\n", temp->data);
   free(temp);
}
int peek() {
   if (top == NULL) {
      printf("Stack is empty.\n");
      return -1;
   }
   return top->data;
}
int main() {
   push(10);
   push(20);
   push(30);
   printf("Top element: %d\n", peek());
   pop();
   pop();
   pop();
   printf("Top element: %d\n", peek());
   return 0;
}
```

OUTPUT:

```
10 pushed to the stack.
20 pushed to the stack.
30 pushed to the stack.
Top element: 30
30 popped from the stack.
20 popped from the stack.
10 popped from the stack.
Stack is empty.
Top element: -1
```

3. Write a C program for Sorting elements using a stack (e.g., sorting a stack using recursion).

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
struct Stack {
   int data;
```

```
    struct Stack* next;
};
struct Stack* createStack() {
    return NULL;
}
int isEmpty(struct Stack* root) {
    return !root;
}
void push(struct Stack** root, int data) {
    struct Stack* newNode = (struct Stack*)malloc(sizeof(struct Stack));
    newNode->data = data;
    newNode->next = *root;
    *root = newNode;
}
int pop(struct Stack** root) {
    if (isEmpty(*root))
        return INT_MIN;
    struct Stack* temp = *root;
    *root = (*root)->next;
    int popped = temp->data;
    free(temp);
    return popped;
}
void sortedInsert(struct Stack** root, int data) {
    if (isEmpty(*root) || data > (*root)->data) {
        push(root, data);
        return;
    }
    int temp = pop(root);
    sortedInsert(root, data);
    push(root, temp);
}
void sortStack(struct Stack** root) {
    if (!isEmpty(*root)) {
        int temp = pop(root);
        sortStack(root);
        sortedInsert(root, temp);
    }
}
void printStack(struct Stack* root) {
    while (root != NULL) {
        printf("%d ", root->data);
        root = root->next;
    }
}
int main() {
    struct Stack* root = createStack();
    push(&root, 30);
```

```
    push(&root, -5);
    push(&root, 18);
    push(&root, 14);
    push(&root, -3);
    printf("Stack elements before sorting: ");
    printStack(root);
    sortStack(&root);
    printf("\nStack elements after sorting: ");
    printStack(root);
    return 0;
}
```

OUTPUT:

Stack elements before sorting: -3 14 18 -5 30

Stack elements after sorting: 30 18 14 -3 -5

4. Write a C Program to Simulate Recursive Function Calls Using a Stack

```
#include <stdio.h>

#define MAX_SIZE 100

int stack[MAX_SIZE];

int top = -1;

void push(int item) {
   if (top >= MAX_SIZE - 1) {
      printf("Stack Overflow\n");
      return;
   }
   stack[++top] = item;
}

int pop() {
   if (top < 0) {
      printf("Stack Underflow\n");
      return -1;
   }
   return stack[top--];
}

int isEmpty() {
   return top == -1;
}
```

```
void simulateRecursive(int n) {
   push(n);
   while (!isEmpty()) {
      int current = pop();
      if (current > 0) {
         printf("%d ", current);
         push(current - 1);
         push(current - 1);
      }
   }
}
int main() {
   int n = 3;
   simulateRecursive(n);
   return 0;
}
```

OUTPUT:

3 2 1 1 2 1 1

5. Write a C program to Implement undo and redo functionality using two stacks.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100
int undoStack[MAX_SIZE];
int redoStack[MAX_SIZE];
int undoTop = -1;
int redoTop = -1;
void pushUndo(int item) {
   if (undoTop == MAX_SIZE - 1) {
      printf("Undo Stack Overflow\n");
   } else {
      undoStack[++undoTop] = item;
   }
}
void pushRedo(int item) {
   if (redoTop == MAX_SIZE - 1) {
      printf("Redo Stack Overflow\n");
```

```
    } else {
        redoStack[++redoTop] = item;
    }
}
int popUndo() {
    if (undoTop == -1) {
        printf("Undo Stack Underflow\n");
        return -1;
    } else {
        return undoStack[undoTop--];
    }
}
int popRedo() {
    if (redoTop == -1) {
        printf("Redo Stack Underflow\n");
        return -1;
    } else {
        return redoStack[redoTop--];
    }
}
void undo() {
    int item = popUndo();
    if (item != -1) {
        pushRedo(item);
        printf("Undo: %d\n", item);
    }
}
void redo() {
    int item = popRedo();
    if (item != -1) {
        pushUndo(item);
        printf("Redo: %d\n", item);
    }
}
int main() {
    pushUndo(1);
    pushUndo(2);
    pushUndo(3);
    undo();
    undo();
    redo();
    redo();
    return 0;
}
```

OUTPUT:

```
Undo: 3
Undo: 2
Redo: 2
```

Redo: 3

6. Write a C program to Check if a string is a palindrome using a stack.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAX 100
struct Stack {
   int top;
   char array[MAX];
};
struct Stack* createStack() {
   struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
   stack->top = -1;
   return stack;
}
int isEmpty(struct Stack* stack) {
   return stack->top == -1;
}
void push(struct Stack* stack, char item) {
   stack->array[++stack->top] = item;
}
char pop(struct Stack* stack) {
   if (!isEmpty(stack))
      return stack->array[stack->top--];
   return '$';
}
int isPalindrome(char str[]) {
   int length = strlen(str);
   struct Stack* stack = createStack();
   int i, mid = length / 2;
   for (i = 0; i < mid; i++) {
      push(stack, str[i]);
   }
   if (length % 2 != 0) {
      i++;
   }
   while (str[i] != '\0') {
      char ele = pop(stack);
      if (ele != str[i])
         return 0;
      i++;
   }
   return 1;
}
int main() {
   char str[MAX];
```

```
   printf("Enter a string: ");
   scanf("%s", str);
   if (isPalindrome(str))
      printf("%s is a palindrome.\n");
   else
      printf("%s is not a palindrome.\n");
   return 0;
}
```

OUTPUT:

```
Enter a string: divya123
Segmentation fault
```