**ASSIGNMENT – 6**

1. Write a Menu driven C program to accomplish the following functionalities in Stack using an Array:

a. Insert an element into the stack using an array (Push Operation).

b. Delete an element from the stack using an array (Pop Operation).

c. Return the value of the topmost element of the stack (without deleting it from

the stack) using an array.

d. Display the elements of a stack using an array.

#include <stdio.h>

#include <math.h>

#include <string.h>

#include <stdlib.h>

#define PI 3.14159

int main()

{

int n;

printf("Enter Number of Elements: ");

scanf("%d", &n);

int ar[n];

int front = -1, rear = -1;

while (1)

{

int ch;

printf("1. Push\n2. Pop\n3. Peek:\n");

scanf("%d", &ch);

switch (ch)

{

case 1:

if (front == -1)

{

front = 0;

rear = 0;

printf("Enter Data: ");

scanf("%d", &ar[front]);

}

else if (rear == (n - 1))

{

printf("Stack is Full\n");

}

else

{

rear++;

printf("Enter Data: ");

scanf("%d", &ar[rear]);

}

break;

case 2:

if (front == -1)

{

break;

}

else if (front == rear)

{

front = -1;

rear = -1;

}

else

{

rear--;

}

break;

case 3:

if (front == -1)

{

break;

}

else

{

printf("The Topmost Value in the Stack is: %d", ar[rear]);

}

break;

}

if (front == -1)

{

printf("Stack is Empty\n");

}

else

{

printf("Your Stack: \n");

for (int i = front; i <= rear; i++)

{

printf("%d ", ar[i]);

}

printf("\n");

}

int x;

printf("Do you want to modify the Stack?\n1. Yes\n2. No:\n");

scanf("%d", &x);

if (x == 2)

{

exit(0);

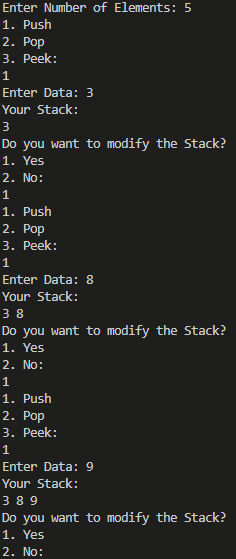
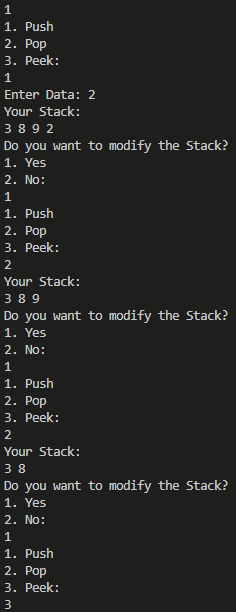
}

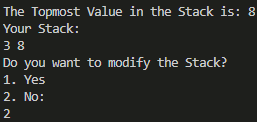
}

return 0;

}

Output:



1. Write a Menu driven C program to accomplish the following functionalities in Stack using Linked List:

a. Insert an element into the stack using a Linked List (Push Operation).

b. Delete an element from the stack using a Linked List (Pop Operation).

c. Return the value of the topmost element of the stack (without deleting it from

the stack) using a Linked List.

d. Display the elements of the stack using a Linked List.

#include <stdio.h>

#include <math.h>

#include <string.h>

#include <stdlib.h>

#define PI 3.14159

struct node

{

int data;

struct node \*link;

};

struct node \*push(struct node \*head, int n)

{

if (head == NULL)

{

int var;

printf("Enter Data: ");

scanf("%d", &var);

struct node \*temp = (struct node \*)malloc(sizeof(struct node \*));

temp->data = var;

temp->link = NULL;

head = temp;

}

else

{

struct node \*loop = head;

int count = 0;

while (loop->link != NULL)

{

loop = loop->link;

count++;

}

count++;

if (count >= n)

{

printf("Stack is Full\n");

}

else

{

int var;

printf("Enter Data: ");

scanf("%d", &var);

struct node \*temp = (struct node \*)malloc(sizeof(struct node \*));

temp->data = var;

temp->link = NULL;

loop->link = temp;

}

}

return head;

}

struct node \*pop(struct node \*head)

{

if (head == NULL)

{

return head;

}

else if (head->link == NULL)

{

head = NULL;

}

else

{

struct node \*loop = head;

while (loop->link->link != NULL)

{

loop = loop->link;

}

loop->link = NULL;

}

return head;

}

void peek(struct node \*head)

{

if (head != NULL)

{

struct node \*loop = head;

while (loop->link != NULL)

{

loop = loop->link;

}

printf("The Topmost Element in this Stack is: %d\n", loop->data);

}

}

void display(struct node \*head)

{

if (head == NULL)

{

printf("Stack is Empty\n");

}

else

{

struct node \*loop = head;

while (loop != NULL)

{

printf("%d ", loop->data);

loop = loop->link;

}

}

}

int main()

{

struct node \*head = NULL;

int n;

printf("Enter Number of Elements: ");

scanf("%d", &n);

while (1)

{

int ch;

printf("1. Push\n2. Pop\n3. Peek:\n");

scanf("%d", &ch);

switch (ch)

{

case 1:

head = push(head, n);

break;

case 2:

head = pop(head);

break;

case 3:

peek(head);

break;

}

display(head);

int x;

printf("\nDo you want to modify the Stack?\n1. YES\n2. NO:\n");

scanf("%d", &x);

if (x == 2)

{

exit(0);

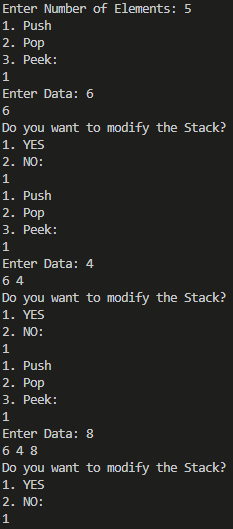
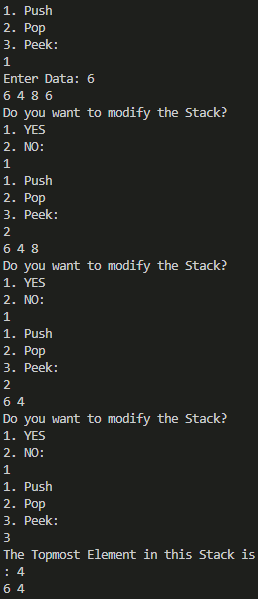
}

}

return 0;

}

Output:

1. Write a program to convert an infix expression into its equivalent postfix notation.

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

typedef struct stack

{

char \*ch;

struct stack \*next;

} stack;

stack \*head = NULL;

int top = 1;

int max;

void push(char \*x, int max)

{

if (top <= max)

{

stack \*newnode;

newnode = (stack \*)malloc(sizeof(stack));

newnode->ch = (char \*)malloc(max \* (sizeof(char)));

strcpy(newnode->ch, x);

newnode->next = NULL;

if (head == NULL)

head = newnode;

else

{

stack \*tmp = head;

newnode->next = tmp;

head = newnode;

top++;

}

}

else

printf("Stack OVERFLOW!\n");

}

char \*pop()

{

if (head == NULL)

{

printf("Stack UNDERFLOW!\n");

return " ";

}

else

{

char \*s;

stack \*tmp;

tmp = head->next;

head->next = NULL;

s = head->ch;

free(head);

head = tmp;

top--;

return s;

}

}

void popall()

{

int i = 1;

if (head == NULL)

i = 0;

else

{

stack \*tmp;

tmp = head->next;

printf("%s", head->ch);

head->next = NULL;

free(head);

head = tmp;

top--;

}

if (i)

popall();

}

int isOperator(char x)

{

switch (x)

{

case '+':

case '-':

case '/':

case '\*':

case '^':

return 1;

}

return 0;

}

char \*operate(char qs)

{

char \*a, \*b, c[2];

c[0] = qs;

c[1] = '\0';

a = pop();

b = pop();

strcat(a, b);

strcat(a, c);

return a;

}

int main()

{

char qs[100];

printf(“Enter Infix: ”);

scanf("%[^\n]s", qs);

int length = strlen(qs);

char \*s = (char \*)malloc(max \* sizeof(char));

char \*q = (char \*)malloc(sizeof(char));

max = length;

for (int i = length - 1; i >= 0; i--)

{

q[0] = qs[i];

q[1] = '\0';

if (isOperator(qs[i]))

{

s = operate(qs[i]);

push(s, max);

}

else

{

push(q, max);

}

}

popall();

return 0;

}

Output:



4. Write a program to convert an infix expression into its equivalent prefix notation.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX\_EXPRESSION\_SIZE 100

typedef struct

{

char data[MAX\_EXPRESSION\_SIZE];

int top;

} Stack;

void initialize(Stack \*stack)

{

stack->top = -1;

}

void push(Stack \*stack, char item)

{

if (stack->top == MAX\_EXPRESSION\_SIZE - 1)

{

printf("Stack Overflow\n");

exit(EXIT\_FAILURE);

}

stack->data[++stack->top] = item;

}

char pop(Stack \*stack)

{

if (stack->top == -1)

{

printf("Stack Underflow\n");

exit(EXIT\_FAILURE);

}

return stack->data[stack->top--];

}

int isOperator(char c)

{

return c == '+' || c == '-' || c == '\*' || c == '/';

}

int getPrecedence(char operator)

{

switch (operator)

{

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

default:

return 0;

}

}

void reverseString(char \*str)

{

int length = strlen(str);

int i, j;

char temp;

for (i = 0, j = length - 1; i < j; i++, j--)

{

temp = str[i];

str[i] = str[j];

str[j] = temp;

}

}

void infixToPrefix(char \*infixExpression, char \*prefixExpression)

{

Stack operatorStack;

initialize(&operatorStack);

reverseString(infixExpression);

for (int i = 0; i < strlen(infixExpression); i++)

{

if (infixExpression[i] == '(')

{

infixExpression[i] = ')';

}

else if (infixExpression[i] == ')')

{

infixExpression[i] = '(';

}

}

int j = 0;

for (int i = 0; i < strlen(infixExpression); i++)

{

char currentChar = infixExpression[i];

if (isalnum(currentChar))

{

prefixExpression[j++] = currentChar;

}

else if (currentChar == '(')

{

push(&operatorStack, currentChar);

}

else if (currentChar == ')')

{

while (operatorStack.top != -1 && operatorStack.data[operatorStack.top] != '(')

{

prefixExpression[j++] = pop(&operatorStack);

}

pop(&operatorStack);

}

else if (isOperator(currentChar))

{

while (operatorStack.top != -1 && operatorStack.data[operatorStack.top] != '(' &&

getPrecedence(operatorStack.data[operatorStack.top]) >= getPrecedence(currentChar))

{

prefixExpression[j++] = pop(&operatorStack);

}

push(&operatorStack, currentChar);

}

}

while (operatorStack.top != -1)

{

prefixExpression[j++] = pop(&operatorStack);

}

prefixExpression[j] = '\0';

reverseString(prefixExpression);

}

int main()

{

char infixExpression[MAX\_EXPRESSION\_SIZE];

printf("Enter Infix: ");

scanf("%s", infixExpression);

char prefixExpression[MAX\_EXPRESSION\_SIZE];

infixToPrefix(infixExpression, prefixExpression);

printf("Prefix Expression: %s\n", prefixExpression);

return 0;

}

Output:



5. Write a program to evaluate a postfix expression.

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#define MAX\_EXPRESSION\_SIZE 100

typedef struct

{

int data[MAX\_EXPRESSION\_SIZE];

int top;

} Stack;

void initialize(Stack \*stack)

{

stack->top = -1;

}

void push(Stack \*stack, int item)

{

if (stack->top == MAX\_EXPRESSION\_SIZE - 1)

{

printf("Stack Overflow\n");

exit(EXIT\_FAILURE);

}

stack->data[++stack->top] = item;

}

int pop(Stack \*stack)

{

if (stack->top == -1)

{

printf("Stack Underflow\n");

exit(EXIT\_FAILURE);

}

return stack->data[stack->top--];

}

int evaluatePostfix(char \*postfixExpression)

{

Stack operandStack;

initialize(&operandStack);

for (int i = 0; postfixExpression[i] != '\0'; i++)

{

char currentChar = postfixExpression[i];

if (isdigit(currentChar))

{

push(&operandStack, currentChar - '0');

}

else if (currentChar == ' ')

{

continue;

}

else

{

int operand2 = pop(&operandStack);

int operand1 = pop(&operandStack);

switch (currentChar)

{

case '+':

push(&operandStack, operand1 + operand2);

break;

case '-':

push(&operandStack, operand1 - operand2);

break;

case '\*':

push(&operandStack, operand1 \* operand2);

break;

case '/':

push(&operandStack, operand1 / operand2);

break;

default:

printf("Invalid operator: %c\n", currentChar);

exit(EXIT\_FAILURE);

}

}

}

return pop(&operandStack);

}

int main()

{

char postfixExpression[MAX\_EXPRESSION\_SIZE];

printf("Enter Postfix Expression:");

scanf("%s", postfixExpression);

int result = evaluatePostfix(postfixExpression);

printf("Result: %d\n", result);

return 0;

}

Output:



6. Write a program to evaluate a prefix expression.

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#define MAX\_SIZE 50

int stack[MAX\_SIZE];

int top = -1;

void initialize()

{

top = -1;

}

int isEmpty()

{

return top == -1;

}

int isFull()

{

return top == MAX\_SIZE - 1;

}

void push(int value)

{

if (isFull())

{

printf("Stack overflow. Cannot push %d.\n", value);

}

else

{

stack[++top] = value;

}

}

int pop()

{

if (isEmpty())

{

printf("Stack underflow. Cannot pop from an empty stack.\n");

return -1; // Return an invalid value

}

else

{

return stack[top--];

}

}

int evaluatePrefix(char \*expression)

{

int len = strlen(expression);

for (int i = len - 1; i >= 0; i--)

{

if (isdigit(expression[i]))

{

push(expression[i] - '0');

}

else if (expression[i] == ' ')

{

continue;

}

else

{

int operand1 = pop();

int operand2 = pop();

switch (expression[i])

{

case '+':

push(operand1 + operand2);

break;

case '-':

push(operand1 - operand2);

break;

case '\*':

push(operand1 \* operand2);

break;

case '/':

push(operand1 / operand2);

break;

default:

printf("Invalid operator: %c\n", expression[i]);

return -1; // Return an error code

}

}

}

return pop();

}

int main()

{

char expression[MAX\_SIZE];

printf("Enter a prefix expression: ");

scanf("%[^\n]%\*c", expression);

int result = evaluatePrefix(expression);

if (result != -1)

{

printf("Result: %d\n", result);

}

else

{

printf("Error in evaluating the expression.\n");

}

return 0;

}

Output:



7. Write a program to print the Fibonacci series using recursion.

#include <stdio.h>

int fibonacci(int n)

{

if (n <= 1)

{

return n;

}

else

{

return fibonacci(n - 1) + fibonacci(n - 2);

}

}

void printFibonacciSeries(int n)

{

printf("Fibonacci Series up to %d terms:\n", n);

for (int i = 0; i < n; i++)

{

printf("%d ", fibonacci(i));

}

printf("\n");

}

int main()

{

int terms;

printf("Enter the number of terms in the Fibonacci series: ");

scanf("%d", &terms);

printFibonacciSeries(terms);

return 0;

}

Output:



8. Write a program to solve the tower of Hanoi problem using recursion.

#include <stdio.h>

void towerOfHanoi(int n, char source, char auxiliary, char destination)

{

if (n == 1)

{

printf("Move disk 1 from %c to %c\n", source, destination);

return;

}

towerOfHanoi(n - 1, source, destination, auxiliary);

printf("Move disk %d from %c to %c\n", n, source, destination);

towerOfHanoi(n - 1, auxiliary, source, destination);

}

int main()

{

int n;

printf("Enter the number of disks: ");

scanf("%d", &n);

printf("Steps to solve the Tower of Hanoi problem with %d disks:\n", n);

towerOfHanoi(n, 'A', 'B', 'C');

return 0;

}

Output:

