DSP-LAB - Assignment

1. Implementation of Stack using Arrays [Push, Pop, Display].

Source code:

#include<stdio.h>

#include<stdlib.h>

#define MAX 100//Maximum number of elements

void push();

void pop();

void display();

int top=-1,stack[MAX];

void push()

{

int val;

if(top==MAX-1)

{

printf("\nStack is full");

}

else

{

printf("\nEnter element to push:");

scanf("%d",&val);

top=top+1;

stack[top]=val;

}

}

void pop()

{

if(top==-1)

{

printf("\nStack is empty");

}

else

{

printf("\nDeleted element is : %d",stack[top]);

top=top-1;

}

}

void display()

{

int i;

if(top==-1)

{

printf("\nStack is empty");

}

else

{

printf("\nItems present in the Stack : \n");

for(i=top;i>=0;--i)

printf("%d\n",stack[i]);

}

}

int main()

{

int n,c;

while(1)

{

printf("\n\nStack operations: \n1. Push \n2. Pop\n3. Display all elements of stack\n4. Exit ");

printf("\nEnter your Choice : ");

scanf("%d", &c);

if(c==1)

{

push();

}

else if(c==2)

{

pop();

}

else if(c==3)

{

display();

}

else if(c==4)

{

exit(0);

}

else

{

printf("Invalid choice");

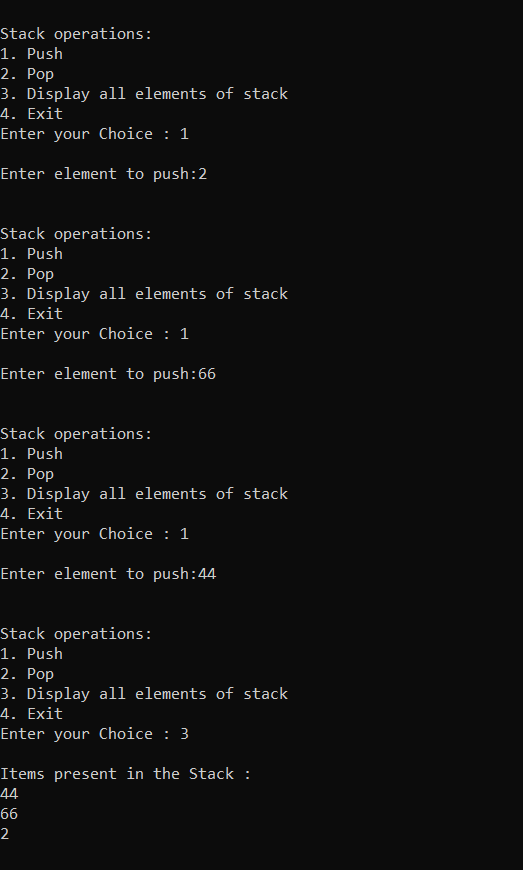
}

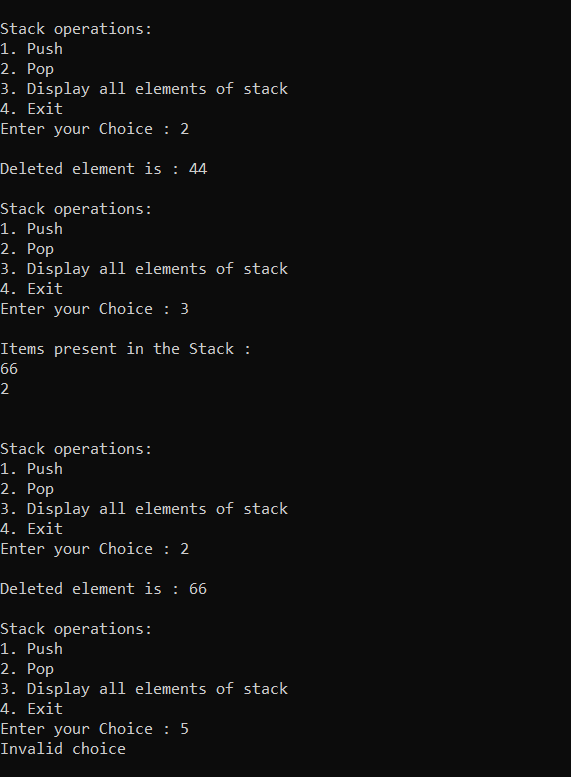
}

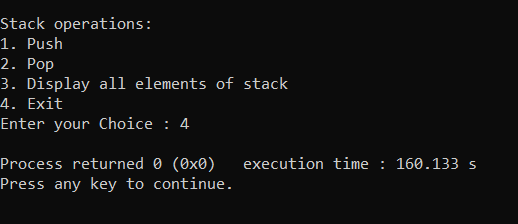
return 0;

}

Output:







1. Implementation of Stack using LinkedList[Push, Pop, Display].

Source Code:

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*top = NULL;

void display();

void push(int);

void pop();

int main()

{

int n,c;

while(1)

{

printf("\n\nStack operations using Linkedlist: \n1. Push \n2. Pop\n3. Display all elements of stack\n4. Exit");

printf("\nEnter your Choice : ");

scanf("%d", &c);

if(c==1)

{

printf("\nEnter Item: ");

scanf("%d", &n);

push(n);

}

else if(c==2)

{

pop();

}

else if(c==3)

{

display();

}

else if(c==4)

{

exit(0);

}

else

{

printf("Invalid choice");

}

}

}

void push(int item)

{

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

ptr->data = item;

ptr->next = top;

top = ptr;

printf("\n%d is added to stack",top->data);

}

void display()

{

struct node \*ptr;

ptr = top;

if(ptr!=NULL)

{

printf("Items present in the stack:");

while (ptr != NULL)

{

printf("\n%d", ptr->data);

ptr = ptr->next;

}

}

else

{

printf("No item to display");

}

}

void pop()

{

if (top == NULL)

{

printf("\nStack is empty ");

}

else

{

struct node \*temp;

temp = top;

top = top->next;

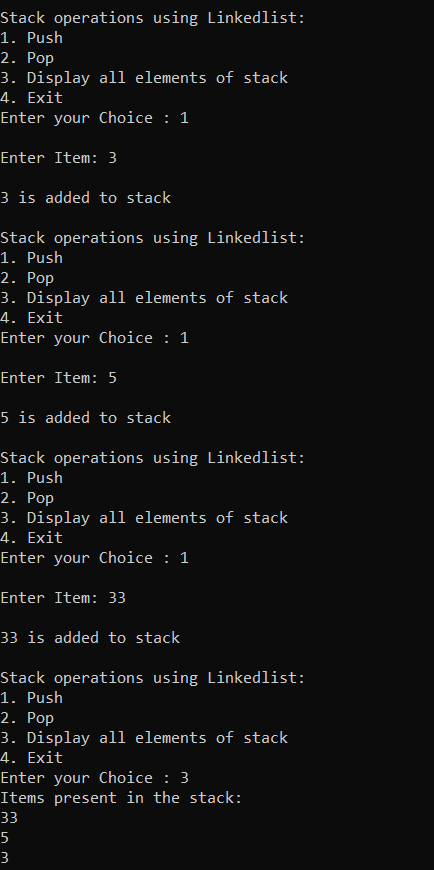
printf("\n%d deleted", temp->data);

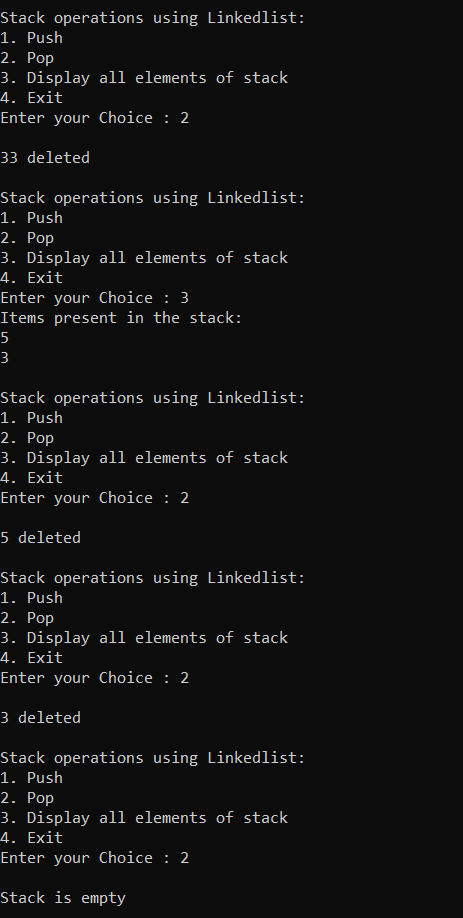
free(temp);

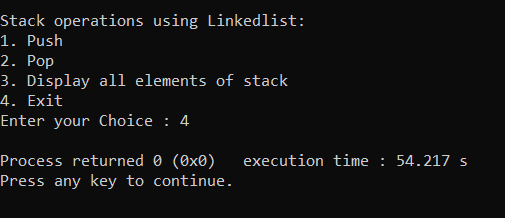
}

}

Output:







1. Implementation of Queue using Arrays [Enqueue and Dequeue].

Source code:

#include <stdio.h>

#define MAX 50

void enqueue();

void dequeue();

void display();

int queue[MAX];

int rear = - 1;

int front = - 1;

main()

{

int c;

while (1)

{

printf("\n\nQueue operations:");

printf("\n1.Enqueue \n");

printf("2.Dequeue \n");

printf("3.Display all elements of queue \n");

printf("4.Quit from program \n");

printf("\nEnter your choice : ");

scanf("%d", &c);

if(c==1)

{

enqueue();

}

else if(c==2)

{

dequeue();

}

else if(c==3)

{

display();

}

else if(c==4)

{

exit(1);

}

else{

printf("Invalid choice");

}

}

}

void enqueue()

{

int item;

if (rear == MAX - 1)

printf("Queue Overflow \n");

else

{

if (front == - 1)

front = 0;

printf("Insert the element in to queue : ");

scanf("%d", &item);

rear = rear + 1;

queue[rear] = item;

}

}

void dequeue()

{

if (front == - 1 || front > rear)

{

printf("Queue is Empty \n");

return ;

}

else

{

printf("Element deleted from queue is : %d\n", queue[front]);

front = front + 1;

}

}

void display()

{

int i;

if (front == - 1)

printf("Queue is empty \n");

else

{

if(front>rear)

{

printf("Queue is empty\n");

}

else{

printf("Items in Queue: \n");

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

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

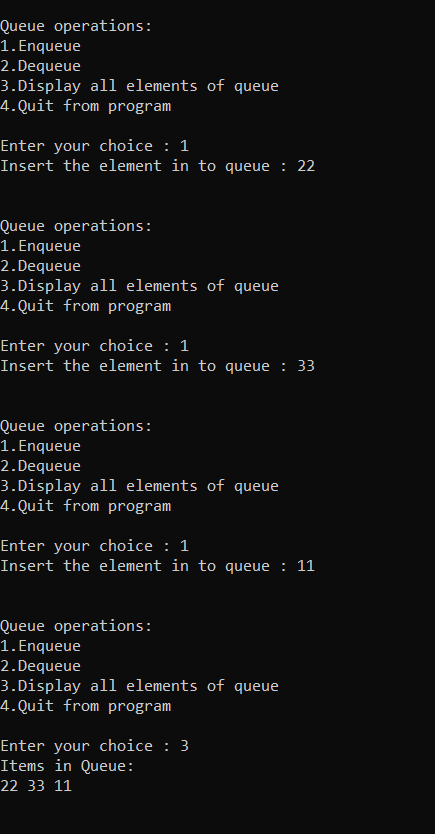
printf("\n");

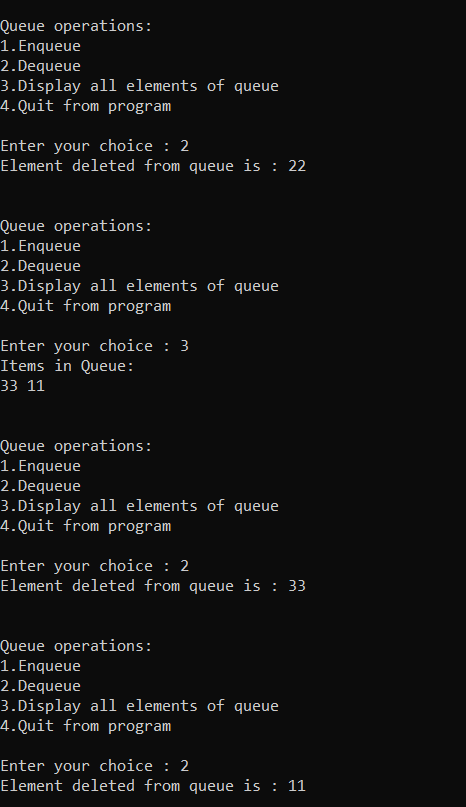
}

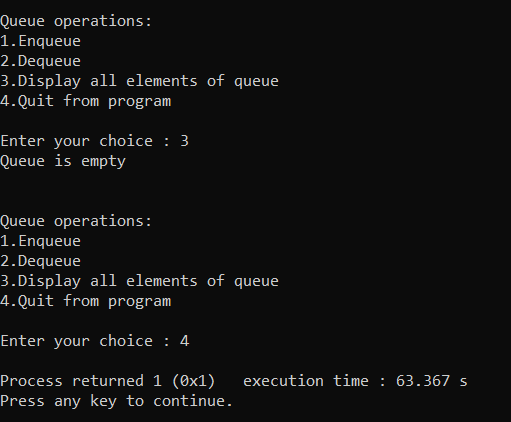
}

}

Output:







1. Implementation of Queue using LinkedList [Enqueue and Dequeue].

Source code:

#include<stdio.h>

#include<conio.h>

struct Node

{

int data;

struct Node \*next;

};

struct Node \*front = NULL,\*rear = NULL;

void enqueue(int);

void dequeue();

void display();

void main()

{

int c,item;

while (1)

{

printf("\n\nQueue operations Using Linked List:");

printf("\n1.Enqueue \n");

printf("2.Dequeue \n");

printf("3.Display all elements of queue \n");

printf("4.Quit from program \n");

printf("\nEnter your choice : ");

scanf("%d", &c);

if(c==1)

{

printf("\nEnter your item to queue: \n");

scanf("%d",&item);

enqueue(item);

}

else if(c==2)

{

dequeue();

}

else if(c==3)

{

display();

}

else if(c==4)

{

exit(1);

}

else{

printf("Invalid choice");

}

}

}

void enqueue(int value)

{

struct Node \*newnode;

newnode = (struct Node\*)malloc(sizeof(struct Node));

newnode -> data = value;

newnode -> next = NULL;

if(front == NULL)

front = rear = newnode;

else

{

rear -> next = newnode;

rear = newnode;

}

printf("\n Item inserted in Queue: \n");

}

void dequeue()

{

if(front == NULL)

printf("\n Queue is Empty\n");

else

{

struct Node \*temp = front;

front = front -> next;

printf("\n Deleted Item is: %d\n", temp->data);

free(temp);

}

}

void display()

{

if(front == NULL)

printf("\n Queue is Empty\n");

else

{

printf("Items in Queue: \n");

struct Node \*temp = front;

while(temp->next != NULL)

{

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

temp = temp -> next;

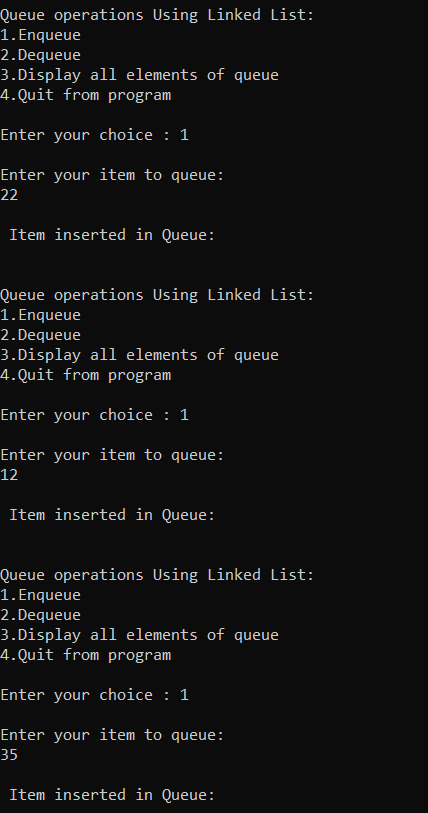
}

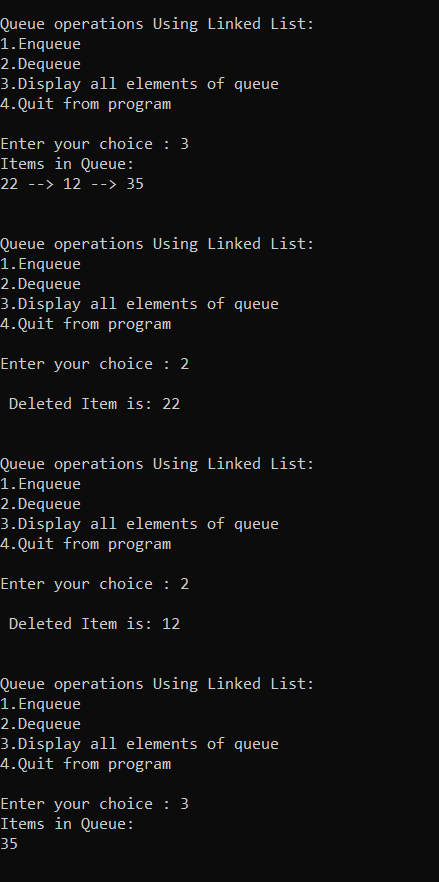
printf("%d \n",temp->data);

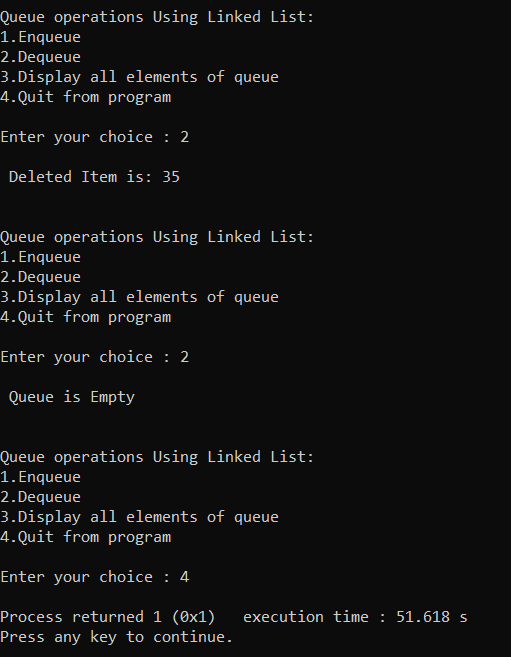
}

}

Output:







1. Implementation of Tower of Hanoi without recursion. Debugged the code.

Source code:

#include <stdio.h>

#include <limits.h>

#include <stdlib.h>

#include <math.h>

struct Stack

{

int top;

int \*arr;

int max;

};

struct Stack\* Stack\_create(int max)

{

struct Stack\* stackTOH = (struct Stack\*)malloc(sizeof(struct Stack));

stackTOH -> max = max;

stackTOH -> top = -1;

stackTOH -> arr = (int\*)malloc(stackTOH -> max\*sizeof(int));

return stackTOH;

}

int isFull(struct Stack\* stackTOH)

{

return (stackTOH->top == stackTOH->max - 1);

}

int isEmpty(struct Stack\* stackTOH)

{

return (stackTOH->top == -1);

}

void display\_shift(char fromPole, char toPole, int disk)

{

printf("Move Disk %d from \'%c\' to \'%c\'\n", disk, fromPole, toPole);

}

void push\_item(struct Stack \*stackTOH, int item)

{

if(isFull(stackTOH))

{

return;

}

stackTOH -> arr[++stackTOH -> top] = item;

}

int pop\_item(struct Stack\* stackTOH)

{

if(isEmpty(stackTOH))

{

return INT\_MIN;

}

return stackTOH -> arr[stackTOH -> top--];

}

void Disk\_shift(struct Stack \*sourcePole, struct Stack \*destPole, char source, char dest)

{

int pole1 = pop\_item(sourcePole);

int pole2 = pop\_item(destPole);

if(pole1 == INT\_MIN)

{

push\_item(sourcePole, pole2);

display\_shift(dest, source, pole2);

}

else if(pole2 == INT\_MIN)

{

push\_item(destPole, pole1 );

display\_shift(source, dest, pole1);

}

else if(pole1 > pole2)

{

push\_item(sourcePole, pole1);

push\_item(sourcePole, pole2);

display\_shift(dest, source, pole2);

}

else

{

push\_item(destPole, pole2);

push\_item(destPole, pole1);

display\_shift(source, dest, pole1);

}

}

void Tower\_of\_Hanoi(int limit, struct Stack \*sourcePole, struct Stack \*tempPole, struct Stack \*destPole)

{

int count, shift;

char dest = 'D', source = 'S', temp = 'A';

if(limit % 2 == 0)

{

char x = dest;

dest = temp;

temp = x;

}

shift = pow(2, limit) - 1;

for(count = limit; count >= 1; count--)

{

push\_item(sourcePole, count);

}

for(count = 1; count <= shift; count++)

{

if(count%3 == 1)

{

Disk\_shift(sourcePole, destPole, source, dest);

}

else if(count%3 == 2)

{

Disk\_shift(sourcePole, tempPole, source, temp);

}

else if(count%3 == 0)

{

Disk\_shift(tempPole, destPole, temp, dest);

}

}

}

int main()

{

int n;

struct Stack \*sourcePole, \*destPole, \*tempPole;

printf("\nEnter the Number of Disks : ");

scanf("%d", &n);

if(n<=0)

{

printf("\nInvalid Input");

return 0;

}

printf("\nSequence of Disk Moves:\n");

sourcePole = Stack\_create(n);

tempPole = Stack\_create(n);

destPole = Stack\_create(n);

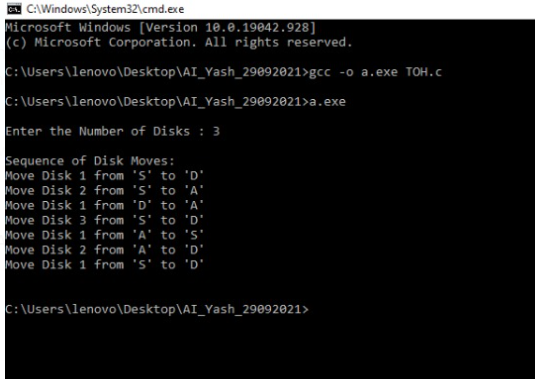
Tower\_of\_Hanoi(n, sourcePole, tempPole, destPole);

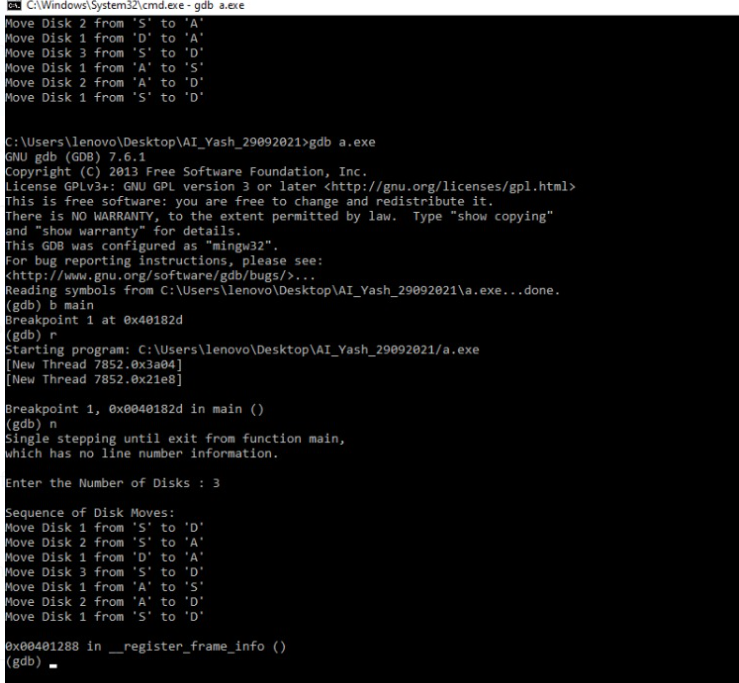
printf("\n");

return 0;

}

Output:





4.

Write a program that convert the infix Expression in to Postfix Expression and then evaluate the postfix expression using stacks. [Note: There should be atleast one two digit operand in the expression]

Source code:

#include<stdio.h>

#include<string.h>

#include<math.h>

#include<stdlib.h>

#define BLANK ' '

#define TAB '\t'

#define MAX 50

void push(long int symbol);

long int pop();

void infix\_to\_postfix();

long int postfix\_evalu();

int priority(char symbol);

int isEmpty();

int white\_space(char );

char infix[MAX], postfix[MAX];

long int stack[MAX];

int top;

int main()

{

long int value;

top=-1;

printf("Enter infix : ");

gets(infix);

infix\_to\_postfix();

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

value=postfix\_evalu();

printf("Value of expression : %ld\n",value);

return 0;

}

void infix\_to\_postfix()

{

unsigned int i,p=0;

char next;

char symbol;

for(i=0;i<strlen(infix);i++)

{

symbol=infix[i];

if(!white\_space(symbol))

{

switch(symbol)

{

case '(':

push(symbol);

break;

case ')':

while((next=pop())!='(')

postfix[p++] = next;

break;

case '+':

case '-':

case '\*':

case '/':

case '%':

case '^':

while( !isEmpty( ) && priority(stack[top])>= priority(symbol) )

postfix[p++]=pop();

push(symbol);

break;

default: /\*if an operand comes\*/

postfix[p++]=symbol;

}

}

}

while(!isEmpty( ))

postfix[p++]=pop();

postfix[p]='\0';

}

int priority(char symbol)

{

switch(symbol)

{

case '(':

return 0;

case '+':

case '-':

return 1;

case '\*':

case '/':

case '%':

return 2;

case '^':

return 3;

default :

return 0;

}

}

void push(long int symbol)

{

if(top>MAX)

{

printf("Stack overflow\n");

exit(1);

}

stack[++top]=symbol;

}

long int pop()

{

if( isEmpty() )

{

printf("Stack underflow\n");

exit(1);

}

return (stack[top--]);

}

int isEmpty()

{

if(top==-1)

return 1;

else

return 0;

}

int white\_space(char symbol)

{

if( symbol == BLANK || symbol == TAB )

return 1;

else

return 0;

}

long int postfix\_evalu()

{

long int a,b,temp,result;

unsigned int i;

for(i=0;i<strlen(postfix);i++)

{

if(postfix[i]<='9' && postfix[i]>='0')

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

else

{

a=pop();

b=pop();

switch(postfix[i])

{

case '+':

temp=b+a; break;

case '-':

temp=b-a;break;

case '\*':

temp=b\*a;break;

case '/':

temp=b/a;break;

case '%':

temp=b%a;break;

case '^':

temp=pow(b,a);

}

push(temp);

}

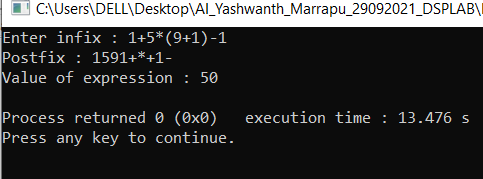
}

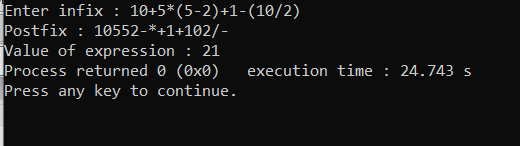
result=pop();

return result;

}

Output:





5. Given an array of integers and a number ‘n’, find the smallest subarray with sum greater than the given value. Display the smallest subarray by adding the size of the subarray to each element in the subarray.

Source Code:

#include <stdio.h>

#include <limits.h>

int l,r;

int minsubarraylen(int arr[], int n, int k)

{

int subsum;

int Sum = 0;

int len = INT\_MAX;

int flag=1;

for (r = 0; r < n; r++)

{

Sum += arr[r];

while (Sum > k && l <= r)

{

subsum = r - l + 1;

if(len>subsum)

{

len = subsum;

}

Sum = Sum - arr[l];

l++;

flag = 0;

}

}

// invalid input

if (flag==1) {

printf("No subarray exists");

return len;

}

// return result

return len;

}

int main()

{

int num,i,arr[1000],n;

printf("Enter no of numbers in an Array:");

scanf("%d",&num);

printf("\nEnter the numbers:");

for(i=0;i<num;i++)

{

scanf("%d",&arr[i]);

}

printf("\nEnter n Values:");

scanf("%d",&n);

// find the length of the smallest subarray

int len = minsubarraylen(arr, num, n);

if (len != INT\_MAX) {

printf("The smallest subarray length is %d:\n", len);

//printf("%d..%d..%d\n",l,r,len);

printf("smallest subarray is :\n");

for(int i=l-1;i<r;i++)

{

printf("%d\t",arr[i]);

}

printf("\n");

printf("smallest subarray after adding length is:\n");

for(int i=l-1;i<r;i++)

{

printf("%d\t",arr[i]+len);

}

}

return 0;

}

Output:

