**NAME – SHUBHAM GARG**

**ENROLL. NO-9919103057**

**BATCH -F2**

**Q1**

#include<stdio.h>

int size1;

int size2;

pushstack1(int n,int top1,int a[],int size1,int size2)

{

if(top1>=size1-1)

{

printf("The stack is full");

}

else

{

top1++;

a[top1]=n;

}

}

pushstack2(int n,int top2,int a[],int size1,int size2)

{

if(top2>=size1+size2-2)

{

printf("The stack is overflow");

}

else

{

top2++;

a[top2]=n;

}

}

popstack1(int top1)

{

if(top1<0)

{

printf("stack is underflow");

}

else{

top1--;

}

}

popstack2(int top2,int size1)

{

if(top2<size1)

{

printf("The stack is underflow");

}

else

{

top2--;

}

}

int main()

{

int i,top1,top2,n;

top1=-1;

printf("Enter the maximum elements in the stack1\n");

scanf("%d",&size1);

printf("Enter the maximum elements in stack 2\n");

scanf("%d",&size2);

top2=size1;

int a[size1+size2];

printf("Elements you want to enter into the stack1\n");

for(i=0;i<size1;i++)

{

scanf("%d",&n);

pushstack1(n,top1,a,size1,size2);

}

printf("Enter elements in stack 2\n");

for(i=0;i<size2;i++)

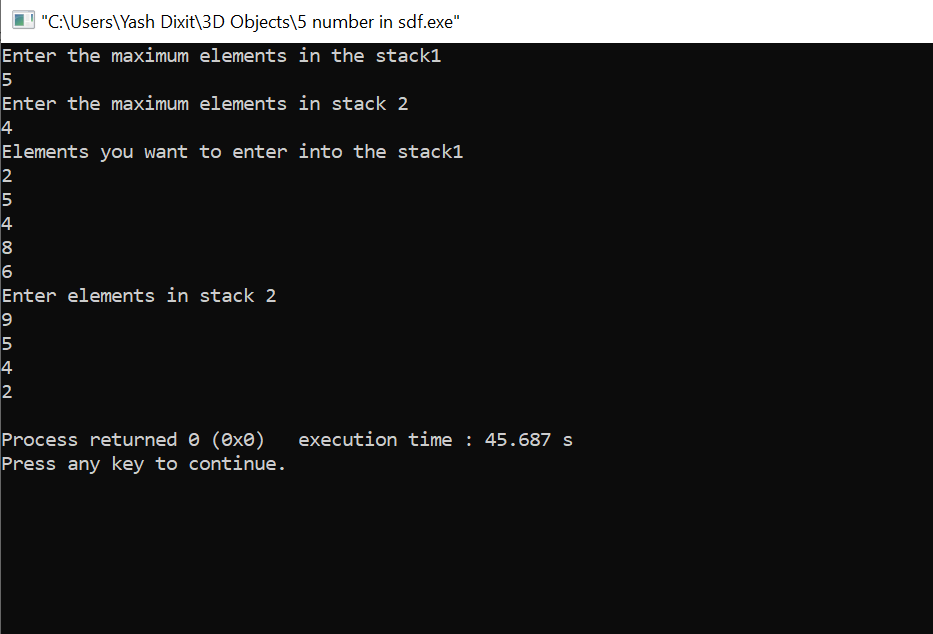
{

scanf("%d",&n);

pushstack2(n,top2,a,size1,size2);

}

}



**Q4**

#include<stdio.h>

#include<string.h>

int size;

int stack1[100];

int stack2[100];

int stack3[100];

int top1=-1,top2=-1,top3=-1;

int pop(int stack[],int size,int top)

{

if(top==-1)

{

printf("Stack is underflow\n\n");

}

else

top--;

return stack[top];

}

void push(int stack[],int size,int n,int top)

{

if(top>=size-1)

{

printf("The stack is full\nElement cannot be inserted into the stack\*\*\*\*");

}

else

{

top++;

stack[top]=n;

}

}

void copy\_a\_b()

{

int x;

while(top1>=0)

{

x=pop( stack1, size, top1);

push( stack2, size, x, top2);

}

while(top2>=0)

{

x=pop( stack2, size, top2);

push( stack3, size, x, top3);

}

}

int main()

{

int i,n;

printf("Enter the maximum size of the stack\n");

scanf("%d",&size);

printf("Enter the elements into the stack\n");

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

{

scanf("%d",&n);

push(stack1,size,n,top1);

}

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

{

printf("%d",stack1[i]);

}

copy\_a\_b();

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

{

printf("%d",stack3[i]);

}

}

**Q5**

#include<stdio.h>

#include<stdlib.h>

#include<ctype.h>

#include<string.h>

#define SIZE 100

char stack[SIZE];

int top = -1;

void push(char item)

{

if(top >= SIZE-1)

{

printf("\nStack Overflow.");

}

else

{

top = top+1;

stack[top] = item;

}

}

char pop()

{

char item ;

if(top <0)

{

printf("stack under flow: invalid infix expression");

getchar();

exit(1);

}

else

{

item = stack[top];

top = top-1;

return(item);

}

}

int is\_operator(char symbol)

{

if(symbol == '^' || symbol == '\*' || symbol == '/' || symbol == '+' || symbol =='-')

{

return 1;

}

else

{

return 0;

}

}

int precedence(char symbol)

{

if(symbol == '^')

{

return(3);

}

else if(symbol == '\*' || symbol == '/')

{

return(2);

}

else if(symbol == '+' || symbol == '-')

{

return(1);

}

else

{

return(0);

}

}

void InfixToPostfix(char infix\_exp[], char postfix\_exp[])

{

int i, j;

char item;

char x;

push('(');

strcat(infix\_exp,")");

i=0;

j=0;

item=infix\_exp[i];

while(item != '\0')

{

if(item == '(')

{

push(item);

}

else if( isdigit(item) || isalpha(item))

{

postfix\_exp[j] = item;

j++;

}

else if(is\_operator(item) == 1)

{

x=pop();

while(is\_operator(x) == 1 && precedence(x)>= precedence(item))

{

postfix\_exp[j] = x;

j++;

x = pop(); }

push(x);

push(item);

}

else if(item == ')')

{

x = pop();

while(x != '(')

{

postfix\_exp[j] = x;

j++;

x = pop();

}

}

else

{

printf("\nInvalid infix Expression.\n");

getchar();

exit(1);

}

i++;

item = infix\_exp[i];

}

if(top>0)

{

printf("\nInvalid infix Expression.\n");

getchar();

exit(1);

}

if(top>0)

{

printf("\nInvalid infix Expression.\n");

getchar();

exit(1);

}

postfix\_exp[j] = '\0';

}

int main()

{

char infix[SIZE], postfix[SIZE];

printf("ASSUMPTION: The infix expression contains single letter variables and single digit constants only.\n");

printf("\nEnter Infix expression : ");

gets(infix);

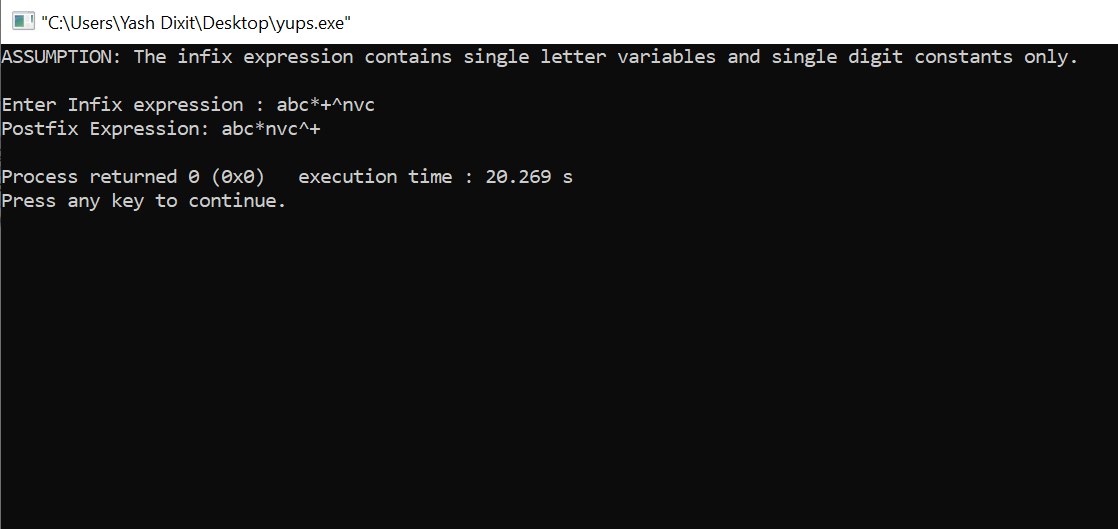
InfixToPostfix(infix,postfix);

printf("Postfix Expression: ");

puts(postfix);

return 0;

}



**Q6**

#include<stdio.h>

#include<string.h>

char stack[100];

int top=-1;

void push(char n;)

{

top++;

stack[top]='n';

}

void pop()

{

if(top==-1)

{

printf("stack is underflow");

}

else

{

top--;

}

}

int main()

{

int i;

char s[100];

printf("Enter the expression whose validity you want to check");

scanf("%s",s);

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

{

if(s[i]=='{'||s[i]=='['||s[i]=='(')

{

push(s[i]);

}

else if(s[i]=='}'||s[i]==']'||s[i]==')')

{

pop();

}

}

if(top==-1)

{

printf("The expression is valid");

}

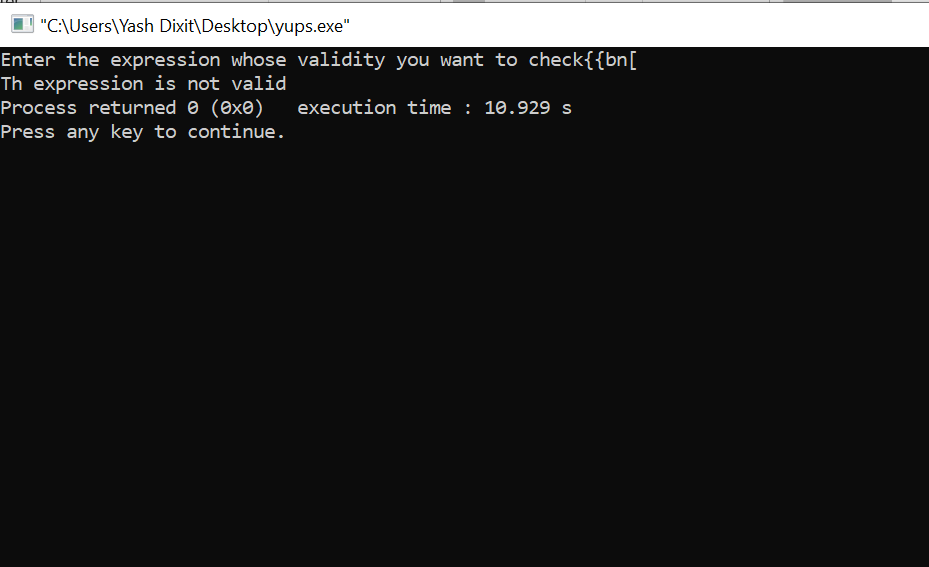
else

{

printf("Th expression is not valid");

}

}



Q7

#include <stdio.h>

#include <stdlib.h>

struct node

{

int a;

struct node \*next;

};

void generate(struct node \*\*);

void display(struct node \*);

void stack\_reverse(struct node \*\*, struct node \*\*);

void delete(struct node \*\*);

int main()

{

struct node \*head = NULL;

generate(&head);

printf("\nThe sequence of contents in stack : \n");

display(head);

printf("\n\nInversing the contents of the stack\n");

if (head != NULL)

{

stack\_reverse(&head, &(head->next));

}

printf("\nThe contents in stack after reversal\n");

display(head);

delete(&head);

return 0;

}

void stack\_reverse(struct node \*\*head, struct node \*\*head\_next)

{

struct node \*temp;

if (\*head\_next != NULL)

{

temp = (\*head\_next)->next;

(\*head\_next)->next = (\*head);

\*head = \*head\_next;

\*head\_next = temp;

stack\_reverse(head, head\_next);

}

}

void display(struct node \*head)

{

if (head != NULL)

{

printf("%d ", head->a);

display(head->next);

}

}

void generate(struct node \*\*head)

{

int num, i;

struct node \*temp;

printf("Enter length of list: ");

scanf("%d", &num);

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

{

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

temp->a = i;

if (\*head == NULL)

{

\*head = temp;

(\*head)->next = NULL;

}

else

{

temp->next = \*head;

\*head = temp;

}

}

}

void delete(struct node \*\*head)

{

struct node \*temp;

while (\*head != NULL)

{

temp = \*head;

\*head = (\*head)->next;

free(temp);

}

}

