*Q1.WAP Write a menu driven program to perform the following operations of a stack using array by using suitable user defined functions for each case*

*a) Check if the stack is empty*

*b) Display the contents of stack*

*c) Push*

*d) Pop*

#include <stdio.h>

#include <stdlib.h>

#define MAX 10

**int** arr[MAX];

**int** top = -1;

**void** push(**int** item);

**int** pop();

**int** Empty();

**int** isFull();

**void** display();

**int** main()

{

**int** choice, item;

    while (1)

    {

        printf("\n1.Push\n");

        printf("2.Pop\n");

        printf("3.Display the top element\n");

        printf("4.Quit\n");

        printf("\nEnter your choice : ");

        scanf("%d", &choice);

        switch (choice)

        {

        case 1:

            printf("\nEnter the item to be pushed : ");

            scanf("%d", &item);

            push(item);

            break;

        case 2:

            item = pop();

            printf("\nPopped item is : %d\n", item);

            break;

        case 3:

            display();

            break;

        case 4:

            exit(1);

        default:

            printf("\nWrong choice\n");

        }

    }

    return 0;

}

**void** push(**int** item)

{

    if (isFull())

    {

        printf("\nStack Overflow\n");

        return;

    }

    top = top + 1;

    arr[top] = item;

}

**int** pop()

{

**int** item;

    if (Empty())

    {

        printf("\nStack Underflow\n");

        exit(1);

    }

    item = arr[top];

    top = top - 1;

    return item;

}

**int** Empty()

{

    if (top == -1)

        return 1;

    else

        return 0;

}

**int** isFull()

{

    if (top == MAX - 1)

        return 1;

    else

        return 0;

}

**void** display()

{

**int** i;

    if (Empty())

    {

        printf("\nStack is empty\n");

        return;

    }

    printf("\nStack elements :\n\n");

    for (i = top; i >= 0; i--)

        printf(" %d\n", arr[i]);

    printf("\n");

}

OUTPUT

1.Push

2.Pop

3.Display the top element

4.Quit

Enter your choice : 1

Enter the item to be pushed : 5

1.Push

2.Pop

3.Display the top element

4.Quit

Enter your choice : 1

Enter the item to be pushed : 3

1.Push

2.Pop

3.Display the top element

4.Quit

Enter your choice : 1

Enter the item to be pushed : 1

1.Push

2.Pop

3.Display the top element

4.Quit

Enter your choice : 1

Enter the item to be pushed : 4

1.Push

2.Pop

3.Display the top element

4.Quit

Enter your choice : 1

Enter the item to be pushed : 2

1.Push

2.Pop

3.Display the top element

4.Quit

Enter your choice : 3

Stack elements :

2

4

1

3

5

1.Push

2.Pop

3.Display the top element

4.Quit

Enter your choice : 2

Popped item is : 2

1.Push

2.Pop

3.Display the top element

4.Quit

Enter your choice : 3

Stack elements :

4

1

3

5

1.Push

2.Pop

3.Display the top element

4.Quit

Enter your choice: 4

*Q2.WAP Write a menu driven program to perform the following operations of a stack using linked list by using suitable user defined functions for*

*each case.*

#include <stdio.h>

#include <stdlib.h>

**struct** node

{

**int** info;

**struct** node \*ptr;

} \* top, \*top1, \*temp;

**int** topelement();

**void** push(**int** data);

**void** pop();

**void** isEmpty();

**void** display();

**void** element\_count();

**void** create();

**int** count = 0;

**void** main()

{

**int** no, ch, e;

    printf("\n1.Push");

    printf("\n2.Pop");

    printf("\n3.Dipslay");

    printf("\n4.Exit");

    create();

    while (1)

    {

        printf("\nEnter choice: ");

        scanf("%d", &ch);

        switch (ch)

        {

        case 1:

            printf("Enter data: ");

            scanf("%d", &no);

            push(no);

            break;

        case 2:

            pop();

            break;

        case 3:

            display();

            break;

        case 4:

            exit(0);

        default:

            printf("Enter correct choice!");

            break;

        }

    }

}

**void** create()

{

    top = NULL;

}

**void** push(**int** data)

{

    if (top == NULL)

    {

        top = (**struct** node \*)malloc(1 \* sizeof(**struct** node));

        top->ptr = NULL;

        top->info = data;

    }

    else

    {

        temp = (**struct** node \*)malloc(1 \* sizeof(**struct** node));

        temp->ptr = top;

        temp->info = data;

        top = temp;

    }

    count++;

}

**void** display()

{

    top1 = top;

    if (top1 == NULL)

    {

        printf("Stack is empty");

        return;

    }

    while (top1 != NULL)

    {

        printf("%d ", top1->info);

        top1 = top1->ptr;

    }

}

**void** pop()

{

    top1 = top;

    if (top1 == NULL)

    {

        printf("\n Error: Trying to pop from empty stack ");

        return;

    }

    else

        top1 = top1->ptr;

    printf("\nPopped value: %d \n", top->info);

    free(top);

    top = top1;

    count--;

}

OUTPUT



*Q3. WAP to convert an infix expression into its equivalent postfix notation*

#include<stdio.h>

#include<string.h>

**char** stack[50];

**int** top=-1;

**void** post(**char** infix**[]**);

**void** push(**char**);

**char** pop();

**void** main()

{

**char** infix[25];

    printf("\nENTER THE INFIX EXPRESSION = ");

    gets(infix);

    post(infix);

}

**void** push(**char** symb)

{

    if(top>=49)

    {

    printf("\nSTACK OVERFLOW");

    return;

    }

    else

    {

    top=top+1;

    stack[top]=symb;

    }

}

**char** pop()

{

**char** item;

    if(top==-1)

    {

       printf("\nSTACK IS EMPTY");

       return(0);

    }

     else

    {

       item=stack[top];

       top--;

    }

    return(item);

}

**int** preced(**char** ch)

{

    if(ch==47)

    {

      return(5);

    }

    else if(ch==42)

    {

      return(4);

    }

    else if(ch==43)

    {

      return(3);

    }

    else

    return(2);

}

**void** post(**char** infix**[]**)

{

**int** l;

**int** index=0,pos=0;

**char** symbol,temp;

**char** postfix[40];

    l=strlen(infix);

    push('#');

    while(index<l)

    {

      symbol=infix[index];

      switch(symbol)

      {

        case '(': push(symbol);

        break;

        case ')': temp=pop();

        while(temp!='(')

        {

            postfix[pos]=temp;

            pos++;

            temp=pop();

        }

        break;

        case '+':

        case '-':

        case '\*':

        case '/':

        case '^':

        while(preced(stack[top])>=preced(symbol))

        {

            temp=pop();

            postfix[pos]=temp;

            pos++;

        }

        push(symbol);

        break;

        default: postfix[pos++]=symbol;

        break;

      }

      index++;

    }

    while(top>0)

    {

        temp=pop();

        postfix[pos++]=temp;

    }

    postfix[pos++]='\0';

    puts(postfix);

    return;

}

OUTPUT



*Q4. WAP to convert an infix expression into its equivalent prefix notation*

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

#define BLANK ' '

#define TAB '\t'

#define MAX 50

**long** **int** pop();

**char** infix[MAX], prefix[MAX];

**long** **int** stack[MAX];

**int** top;

**int** isempty();

**int** white\_space(**char** symbol);

**void** infix\_to\_prefix();

**int** priority(**char** symbol);

**void** push(**long** **int** symbol);

**long** **int** pop();

**int** main()

{

**long** **int** value;

    top = -1;

    printf("Enter infix : ");

    gets(infix);

    infix\_to\_prefix();

    printf("prefix : %s\n", prefix);

    return 0;

}

**void** infix\_to\_prefix()

{

**int** i, j, p, n;

**char** next;

**char** symbol;

**char** temp;

    n = strlen(infix);

    p = 0;

    for (i = n - 1; i >= 0; i--)

    {

        symbol = infix[i];

        if (!white\_space(symbol))

        {

            switch (symbol)

            {

            case ')':

                push(symbol);

                break;

            case '(':

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

                    prefix[p++] = next;

                break;

            case '+':

            case '-':

            case '\*':

            case '/':

            case '%':

            case '^':

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

                    prefix[p++] = pop();

                push(symbol);

                break;

            default:

                prefix[p++] = symbol;

            }

        }

    }

    while (!isempty())

        prefix[p++] = pop();

    prefix[p] = '\0';

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

    {

        temp = prefix[i];

        prefix[i] = prefix[j];

        prefix[j] = temp;

    }

}

**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);

    }

    else

    {

        top = top + 1;

        stack[top] = symbol;

    }

}

**long** **int** pop()

{

    if (top == -1)

    {

        printf("Stack underflow \n");

        exit(2);

    }

    return (stack[top--]);

}

**int** isempty()

{

    if (top == -1)

        return 1;

    else

        return 0;

}

**int** white\_space(**char** symbol)

{

    if (symbol == BLANK || symbol == TAB || symbol == '\0')

        return 1;

    else

        return 0;

}

OUTPUT



*Q5.WAP to determine whether the input sequence of brackets is balanced or not. If a string*

*Is balanced, it prints YES on a new line; otherwise, print NO on a new line. Example: Input:*

*{[()]} and OUTPUT: YES Input: {[(])} and OUTPUT: NO*

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

**int** same(**char** a, **char** b)

{

    if (a == '[' && b == ']')

        return 1;

    if (a == '{' && b == '}')

        return 1;

    if (a == '(' && b == ')')

        return 1;

    return 0;

}

**int** check(**char** \*a)

{

**char** stack[1001], top = -1;

    for (**int** j = 0; j < strlen(a); j++)

    {

        if (a[j] == '[' || a[j] == '{' || a[j] == '(')

            stack[++top] = a[j];

        if (a[j] == ']' || a[j] == '}' || a[j] == ')')

        {

            if (top == -1)

            {

                return 0;

            }

            else

            {

                if (!same(stack[top--], a[j]))

                {

                    return 0;

                }

            }

        }

    }

    if (top != -1)

    {

        return 0;

    }

    return 1;

}

**int** main()

{

**char** a[1001];

**int** n, valid;

    printf("enter number of choice of bracket you need to input: ");

    scanf("%d", &n);

    for (**int** i = 0; i < n; i++)

    {

        scanf("%s", a);

        valid = check(a);

        if (valid == 1)

            printf("YES\n");

        else

            printf("NO\n");

    }

    return 0;

}

OUTPUT

