**DS: Assignment 1**

Name: Vivian Vijay Ludrick

Branch : SE Comps A Batch B

Roll No: **9914**

**Q.1 Check for Balanced Brackets in an expression**

CODE:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define SIZE 100

*// a stack to store the opening braces and when an ending brac appers it pops one out*

typedef struct

{

char a[SIZE];

int tos;

} Bracket;

*// push the brackets into the stack*

void push(Bracket \**b*, char *ele*)

{

if (*b*->tos == SIZE - 1)

{

printf("Stack overflow");

exit(404);

}

*b*->a[++*b*->tos] = *ele*;

}

*// pops a bracket out of the stack*

char pop(Bracket \**b*)

{

if (*b*->tos == -1)

{

printf("Stack underflow");

exit(404);

}

return *b*->a[*b*->tos--];

}

*// start of the main*

int main()

{

Bracket b1;

b1.tos = -1;

char x;

char input[SIZE];

printf("Enter expression:\n");

fgets(input, SIZE, stdin);

printf("Current expression:\t%s\n", input);

int i = 0;

while (input[i] != '\0')

{

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

{

push(&b1, input[i]); *// opening -> push to tos*

}

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

{

x = pop(&b1); *// closing -> pop out one*

if (input[i] == ')' && x != '(' || input[i] == ']' && x != '[' || input[i] == '}' && x != '{')

{

printf("Not a balanced expression"); *// if no match exit*

exit(300);

}

}

i++;

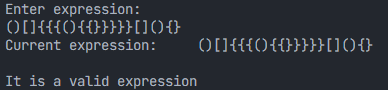
}

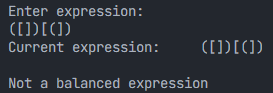
printf("It is a valid expression");

return 200;

}

OUTPUT:





**Q.2 Convert Infix expression to Postfix expression**

CODE:

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <ctype.h> *//isalnum***

**#define SIZE 20**

***// stack of operators***

**typedef struct**

**{**

**char a[SIZE];**

**int tos;**

**} Stack;**

***// push an operator onto stack***

**void push(Stack \**s*, char *ele*)**

**{**

**if (*s*->tos == SIZE - 1)**

**{**

**printf("Stack overflow");**

**return;**

**}**

**else**

**{**

***s*->tos++;**

***s*->a[*s*->tos] = *ele*;**

**}**

**}**

***// pop an operator out of stack***

**char pop(Stack \**s*)**

**{**

**char x = *s*->a[*s*->tos];**

***s*->tos--;**

**return x;**

**}**

**int isEmpty(Stack *s*)**

**{**

**if (*s*.tos == -1)**

**return 1;**

**else**

**return 0;**

**}**

**int precedence(char *c*)**

**{**

**if (*c* == '(') *// lowest precedence to (***

**return 0;**

**else if (*c* == '+' || *c* == '-')**

**return 1;**

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

**return 2;**

**else if (*c* == '^' || *c* == '$') *// highest precedence to ^***

**return 3;**

**}**

**int main()**

**{**

**Stack s1;**

**char str[SIZE], post[SIZE]; *// str for input string***

***// post is for output string***

**int i, j = 0;**

**char x;**

**s1.tos = -1;**

**printf("Enter Infix Expression: ");**

**gets(str);**

**for (i = 0; str[i] != '\0'; i++) *// or for(i = 0 ; i < strlen(str);i++)***

**{**

**if (isalnum(str[i])) *// if operand***

**{**

**post[j] = str[i]; *// put it in postfix string***

**j++;**

**}**

**else *// if operator***

**{**

**if (str[i] == '(')**

**{**

**push(&s1, str[i]);**

**}**

**else if (str[i] == ')')**

**{**

***// pop operators from stack and put it in postfix string until '(' encountered***

**while ((x = pop(&s1)) != '(') *// if popped element is / discard***

**{**

**post[j] = x;**

**j++;**

**}**

**}**

**else if (isEmpty(s1) ||**

**precedence(str[i]) > precedence(s1.a[s1.tos])) *// if str[i] is a operator***

**{**

***// stack is empty push***

***// if str[i] is operator having higher precedence than stack top operator, push it***

**push(&s1, str[i]);**

**}**

**else**

**{**

***// if str[i] is operator having less or equal precedence than stack top operator,***

***// it means stack top operators having higher or equal precedence***

***// go on popping such operators***

***// while popping you have to ensure stack is not empty***

**while (!isEmpty(s1) && (precedence(str[i])) <= precedence(s1.a[s1.tos]))**

**{**

**post[j] = pop(&s1); *// pop operators from stack having higher or equal precedence//make it a part of output string***

**j++;**

**}**

**push(&s1, str[i]); *// push the current scanned operator on stack***

**}**

**}**

**}**

**while (!isEmpty(s1)) *// at the end pop all operators and put them in postfix string***

**{**

**post[j] = pop(&s1);**

**j++;**

**}**

**post[j] = '\0'; *// put null character at the end***

**printf("Postfix expression is: %s", post); *// print output***

**return (0);**

**}**

OUTPUT:



**Q.3.Evaluation of Postfix Expression**

CODE:

#include <stdio.h>

#include <math.h> *//for pow*

#include <stdlib.h> *//for exit*

#include <ctype.h> *//for isdigit*

#define SIZE 500

*// a stack to store the numbers*

struct stack

{

int operand[SIZE];

int top;

};

*// checks whether the numbers are empty or not*

int isEmpty(struct stack \**ps*)

{

if (*ps*->top == -1)

return (1);

else

return (0);

}

*// pushes the encountered number onto the stack*

void push(struct stack \**ps*, int *a*)

{

if ((*ps*->top) == SIZE - 1)

printf("Stack overflow");

else

*ps*->operand[++(*ps*->top)] = *a*;

}

*// pops the out of stack*

int pop(struct stack \**ps*)

{

if (isEmpty(*ps*))

{

printf("Stack underflow");

exit(1);

}

return (*ps*->operand[(*ps*->top)--]);

}

*/\**

*calculates the operation based on the symbol.*

*Note: it cant directly calculate since the oprators are characters and hence cant be direclty operated upon*

*\*/*

int calculate(int *a*, int *b*, char *symb*)

{

switch (*symb*)

{

case '+':

return (*a* + *b*);

case '-':

return (*a* - *b*);

case '\*':

return (*a* \* *b*);

case '/':

return (*a* / *b*);

case '^':

return (pow(*a*, *b*));

}

}

*// start of the main*

int main()

{

struct stack p1;

p1.top = -1;

int i = 0;

char expr[SIZE];

int t1, t2;

printf("Enter the postfix expression = ");

scanf("%s", expr);

while (expr[i])

{

if (isdigit(expr[i]))

{

push(&p1, expr[i] - '0');

}

else

{

*// need to pop two operands since for each opreand there are two operators*

*// to convert to infix expression just print the operator in between the operands however there might be an issue with the brackets so might need to check it. As per my guess it wont occur.*

t1 = pop(&p1);

t2 = pop(&p1);

push(&p1, calculate(t2, t1, expr[i]));

}

i++;

}

printf("\nAnswer = %d", pop(&p1));

return 0;

}

OUTPUT:

