Data Structure and Algorithm
Lab Experiment 2
Balanced parenthesis

Name: Om Ashish Mishra

Registration No.: 16BCE0789

Slot: G2

The Pseudo code:

- First we write the header files following the pre-processer directive.
- Then we define the macro for SIZE 50.
- Then we declare the stack(s[50]) and the ‘top’ pointer to -1.
- Then the push function is defined to accept the elements into the stack.
- Then the pop function is defined to remove the elements from the stack.
- Then MAIN function is defined to make a MENU-DRIVEN program.
- We declare a flag operator to check for elements between the parenthesis.
- We take the expression from the user and run a for loop.
- If the string’s element is an open parenthesis we push the elements into the stack.
- If the string’s element is a closed parenthesis we check if at the top we have similar open parenthesis if so then we pop the elements else we declare flag as zero and break from the loop.
- At the end we check the condition, if top is not equal to -1 or flag equal to 0.
- If so we print unbalanced parenthesis or we print balanced parenthesis.
- Then we ask the user whether he/she wants to continue if yes it repeats or else it stops.

The Code:

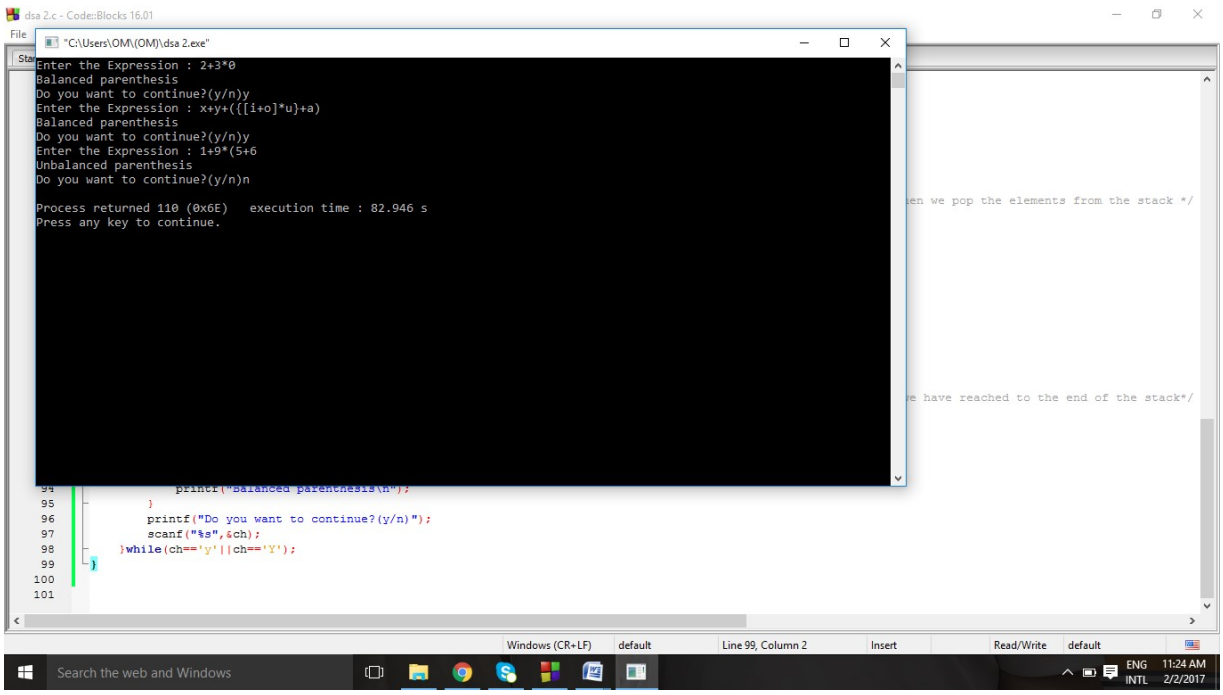
```
#include <stdio.h>
#include <string.h>
#define SIZE 50            /* Macro to give SIZE a fixed constant value */

char s[SIZE];              /* Stack with size of 50 */
int top = -1;              /* Top is used to point to the top of the stack */

void push(char c)          /* Push the elements into the stack */
{
        if(top < SIZE)         /* Checking for OVERFLOW */
        {
                top++;
                s[top] = c;
        }
        if(top >= SIZE-1)
        {
                printf("Overflow or the stack is full");
        }
}

void pop()                 /* Pop is used to remove the elements from the stack */
{
        if(top > -1)           /* Checking for Underflow*/
        {
                top--;
```

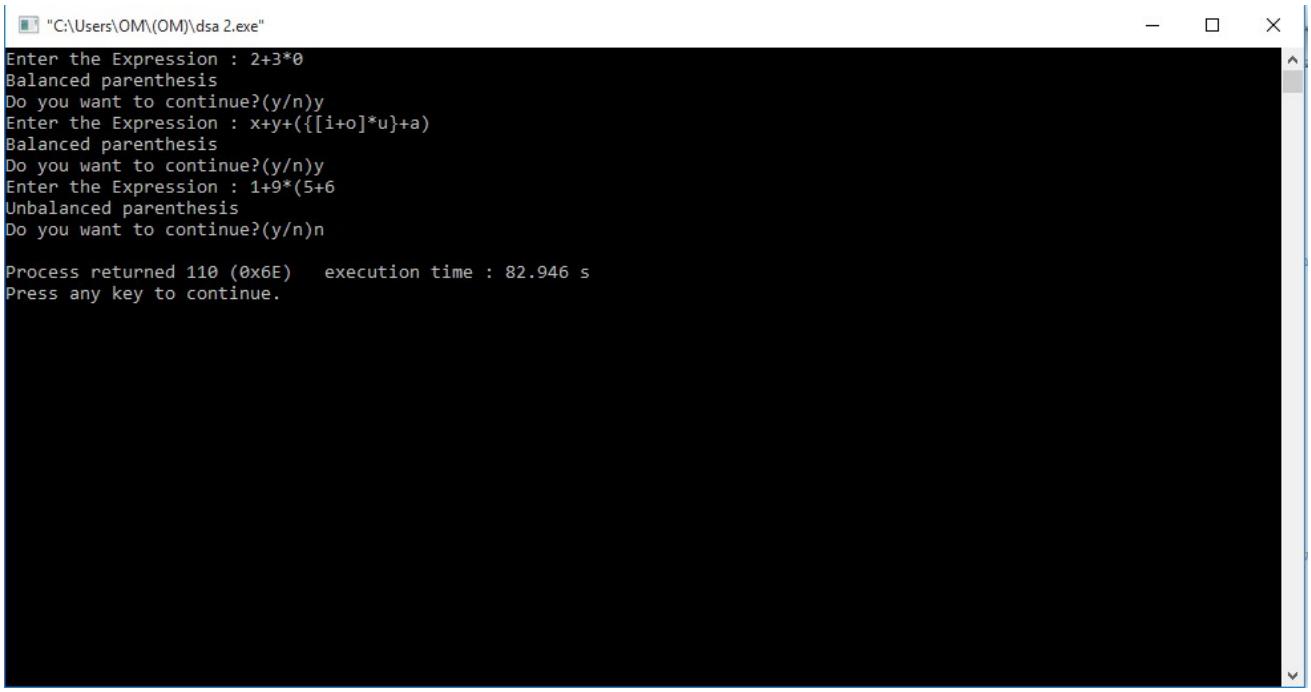
```
            }
            else
            {
                        printf("Underflow or the stack is empty \n");
            }
}

void main()
{
   char ch='y';
   do                    /* MENU DRIVEN PROGRAM */
   {
      int i;
      char exp[50];          /* The string expression */
      printf("Enter the Expression : ");
      scanf("%s", exp);
      int l = strlen(exp);       /* The length of the string */
      int flag = 1;              /* flag operator to check whether the elements in the stack within the
parenthesis are present or not*/
      for(i = 0; i<l; i++)
      {
         if(exp[i] == '(' || exp[i] == '{' || exp[i] == '[')     /* Checking the Parenthesis pushing the elements
into the stack */
         {
            push(exp[i]);
         }
         else if(exp[i] == ')')     /* Here we check whether the ending parenthesis is reached or not if so
then we pop the elements from the stack */
         {
```

```
            if(s[top] == '(')
            {
               pop();
            }
            else
            {
               flag = 0;            /* The flag operator is made to 0 */
               break;
            }
         }
         else if(exp[i] == '}')     /* Here we check whether the ending parenthesis is reached or not if so
then we pop the elements from the stack */
         {
            if(s[top] == '{')
            {
               pop();
            }
            else
            {
               flag = 0;            /* The flag operator is made to 0 */
               break;
            }
         }
         else if(exp[i] == ']')     /* Here we check whether the ending parenthesis is reached or not if so
then we pop the elements from the stack */
         {
            if(s[top] == '[')
            {
```

```
            pop();
          }
          else
          {
            flag = 0;          /* The flag operator is made to 0 */
            break;
          }
        }
    }                       /*End of for loop*/
    if(top != -1 || flag == 0)    /* Here we are checking for the inside part of the parenthesis and
whether we have reached to the end of the stack*/
    {
      printf("Unbalanced parenthesis\n");
    }
    else
    {
      printf("Balanced parenthesis\n");
    }
    printf("Do you want to continue?(y/n)");
    scanf("%s",&ch);
  }while(ch=='y'||ch=='Y');        /* Checks whether the uuser wants any other input or not*/
}
```

The Output


dsa 2.c - Code::Blocks 16.01
"C:\Users\OM\(OM)\dsa 2.exe"
Enter the Expression : 2+3*0
Balanced parenthesis
Do you want to continue?(y/n)y
Enter the Expression : x+y+({[i+o]*u}+a)
Balanced parenthesis
Do you want to continue?(y/n)y
Enter the Expression : 1+9*(5+6
Unbalanced parenthesis
Do you want to continue?(y/n)n
Process returned 110 (0x6E) execution time : 82.946 s
Press any key to continue.
printf("Balanced parenthesis\n");
}
printf("Do you want to continue?(y/n)");
scanf("%s",&ch);
}while(ch=='y'||ch=='Y');
}
Windows (CR+LF)
default
Line 99, Column 2
Insert
Read/Write
default
Search the web and Windows
ENG
INTL
11:24 AM
2/2/2017


"C:\Users\OM\(OM)\dsa 2.exe"
Enter the Expression : 2+3*0
Balanced parenthesis
Do you want to continue?(y/n)y
Enter the Expression : x+y+({[i+o]*u}+a)
Balanced parenthesis
Do you want to continue?(y/n)y
Enter the Expression : 1+9*(5+6
Unbalanced parenthesis
Do you want to continue?(y/n)n
Process returned 110 (0x6E) execution time : 82.946 s
Press any key to continue.