# Experiment No.03

## PART A

(PART A: TO BE REFFERED BY STUDENTS)

* 1. **Aim:** To understand and implement stack-based algorithms for: Checking balanced parentheses.

###### Prerequisite: - Knowledge of any programming language

Understanding of stack data structure and its operations (push, pop, peek)

* 1. **Outcome:**

After successful completion of this experiment, students will be able to

1. Be able to implement algorithms for checking balanced parentheses using stacks.
2. Gain practical experience in using stacks to solve real-world problems.

###### Theory:

**Checking balanced parentheses.**

* Parentheses are balanced if each opening parenthesis has a corresponding closing parenthesis and they are properly nested.
* Example: {[()()]} is balanced, but {[(])} is not.

**Algorithm:**

1. Initialize an empty stack.
2. Traverse the expression from left to right.
3. For each character:
   * If it is an opening parenthesis ({, [, (), push it onto the stack.
   * If it is a closing parenthesis (}, ], )):
     + Check if the stack is empty. If it is, the expression is not balanced.
     + Otherwise, pop the top of the stack and check if it matches the current closing parenthesis.
4. After processing all characters, check if the stack is empty. If it is not empty, the expression is not balanced.

**Example:**

* Expression: {[()()]} → Balanced
* Expression: {[(])} → Not Balanced
  1. **Tasks to be completed**

Write a program in C++ to perform parenthesis check using stack

\

**PART B**

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)

|  |  |
| --- | --- |
| Roll No. C002 | Name: Kartik Sethi |
| Program : BTI | Division: B |
| Batch: B1 | Date of Experiment: |
| Date of Submission: | Grade : |

* 1. **Tasks given in PART A to be completed here**

#include <iostream>

#include <string>

using namespace std;

#define MAX\_SIZE 10

class SStack

{

public:

char stack[MAX\_SIZE];

int top = -1;

bool push(char n)

{

if (top >= MAX\_SIZE - 1)

{

return false;

}

else

{

top++;

stack[top] = n;

return true;

}

}

bool pop()

{

if (isEmpty())

{

return false;

}

else

{

top--;

return true;

}

}

char peek()

{

if (top >= 0)

{

return stack[top];

return '\0';

}

bool isEmpty()

{

return top == -1; // top is -1

}

bool isFull()

{

return top >= MAX\_SIZE - 1;

}

};

class CHECKER : protected SStack

{

public:

bool isBalanced(string expr, int length)

{

char ch;

for (int i = 0; i < length; i++)

{

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

{

push(expr[i]);

continue;

}

if (isEmpty())

return false;

ch = peek();

pop();

switch (expr[i])

{

case ')':

if (ch != '(')

return false;

break;

case '}':

if (ch != '{')

return false;

break;

case ']':

if (ch != '[')

return false;

break;

}

}

return isEmpty();

}

};

int main()

{

CHECKER c;

string expr = "{[()}}";

int length = expr.size();

if (c.isBalanced(expr, length))

{

cout << "Balanced";

}

else

{

cout << "Not Balanced";

}

return 0;

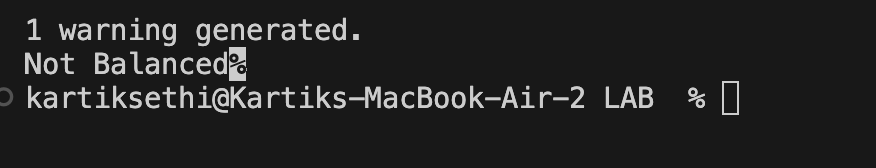
}

* 1. **Output /Observations**

**When:**

1. string expr = "{[()}}";

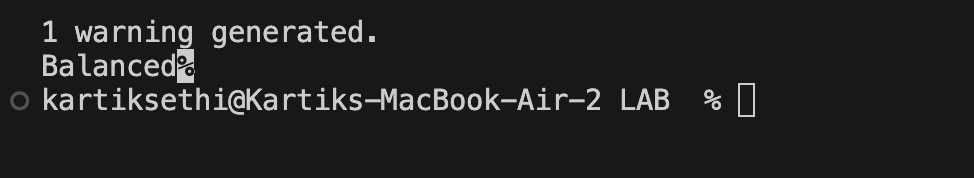
**Output:**

****

**When:**

string expr = "{[()]}";

**Output:**

****

* 1. **Conclusion:**

I learned how to implement stack-based algorithms for: Checking balanced parentheses.