Alex Isiofia

Professor Lane

COM 210

7/3/22

Lab 2: Delimiter Matching Using Restricted Structures

**Abstract:**

In computer programming, a delimiter is a [character](https://www.techtarget.com/whatis/definition/character) that identifies the beginning or the end of a character string (a contiguous sequence of characters). Delimiters separate text strings. Common delimiters are [commas](https://www.computerhope.com/jargon/c/comma.htm) (,), [semicolon](https://www.computerhope.com/jargon/s/semicolo.htm) (;), [quotes](https://www.computerhope.com/jargon/q/quote.htm) ( ", ' ), [braces](https://www.computerhope.com/jargon/c/curlybra.htm) ({}), [pipes](https://www.computerhope.com/jargon/p/pipe.htm) (|), or [slashes](https://www.computerhope.com/jargon/s/slash.htm) ( / \ ). When a program stores [sequential](https://www.computerhope.com/jargon/s/sequence.htm) or [tabular](https://www.computerhope.com/jargon/t/tabular.htm) data, it delimits each item of data with a predefined character. A stack is a data structure that only operates on the most recent item added, also known as [LIFO](https://www.computerhope.com/jargon/l/lifo.htm) (last-in, first-out). When a new item is added to the stack, it's called "pushing" that item onto the stack. When the most recent data item is retrieved and removed from the structure, it's called "popping" the item from the stack. These "push" and "pop" operations may apply to an [array](https://www.computerhope.com/jargon/a/array.htm) if the item added or removed is at the end of the array.

**Introduction:**

In this lab, I learned how to perform delimiter matching using restricted structures. One common use for stacks is to parse certain kinds of text strings. Typically, the strings are lines of code in a computer language, and the programs parsing them are compilers. I created a program that checked the delimiters in a line of text typed by the user. This text was coded in java. The delimiters used were braces { and }, brackets [ and ], and parentheses ( and ). Each opening or left delimiter was matched by a closing or right delimiter; that is, every { was followed by a matching } and so on. Also, opening delimiters that occurred later in the string were closed before those occurring earlier.

**Procedure:**

Here, I used java to create a program that checks the delimiters in a line of text input by the user. “Boolean” was used to create a stack object and initialize the stack using constructor. The size of the stack was determined by the expression's length. I used if statements to check the delimiters. Whenever an opening bracket was found, it was put onto the stack. Whenever a closing bracket was found, an item was popped (removed) from the top of the stack to match that with the closing bracket. At the end if the delimiters were matched the word “Matched” was output by the program. If the delimiters were not matched, then the program output “Not Matched”.

**Code:**

package testdelimiter;

import java.util.\*;

class DelimiterMatching

{

String expr; //expects a string to be input

public boolean isDelimiterMatching(String expr)

{

Deque<Character> stk = new ArrayDeque<>(); //create new stack

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

{

char x = expr.charAt(i);

if (x == '(' || x == '[' || x == '{') //opening symbols

{

stk.push(x); //push them

continue;

}

if(x != ')' && x != ']' && x != '}') //closing symbols

{

continue;

}

if (stk.isEmpty()) //check if empty

return false;

char check;

switch (x)

{

case ')':

check = stk.pop(); //pop and check

if (check == '{' || check == '[')

return false;

break;

case '}':

check = stk.pop(); //pop and check

if (check == '(' || check == '[')

return false;

break;

case ']':

check = stk.pop(); //pop and check

if (check == '(' || check == '{')

return false;

break;

}

}

return (stk.isEmpty()); //end check if stack is empty

}

}

class TestDelimiter

{

public static void main(String args[])

{

DelimiterMatching D = new DelimiterMatching();

if(D.isDelimiterMatching("x\*(y+z)+[z/(z-y)]"))

System.out.print("Matched");

else System.out.print("Not Matched");

}

}

**Results:**

The program was successful. I got “Matched” as my output.

Graphical user interface, text, table

Description automatically generated with medium confidence

**Conclusion:**

From this lab, I have learned how to match delimiters using stack structure. At first, I was experiencing difficulties because I was getting a lot of errors. I had to try different methods until something finally worked. Although, I found this lab to be very tricky, I am still glad that I got to experience it. Overall, I have a better understanding of delimiters, stacks, and java as a result of completing this lab.