Courtney Greene

Professor Lane

COM 210

July 07, 2022

Project 1 - Delimiters

Delimiters

The program below illustrates the use of the push and pop algorithm that helps in checking the delimiters that are typed by a user in Java. Matching delimiters is very important in programming because all brackets must be matching one another in order for a program to execute. When opening brackets, the closing brackets must be matching in order for the code that is being executed in between them, can work. When opening brackets are identified in a program, a push algorithm pushes these brackets that are opened, so that when the pop algorithm comes along, it helps in determining if the opening brackets that were pushed are then closed by the appropriate brackets so that these brackets are matching.

The first part of this program is the push algorithm. In this part of the code, we push each of the different brackets which include ({[. These brackets are opening brackets. Since we push these brackets, we have a starting point we can use to determine which closing bracket is needed for this program to work. First we look to see if the stack is empty, meaning that if there is no opening bracket or no closing bracket. If this is the case, then it is immediately incorrect because nothing is being opened or closed in the first place.

In the next part of this program we incorporate the pop algorithm. This part allows us to check what we are matching the opening bracket to in the typed code. As each bracket is checked for its matching opening and closing brackets, if there are any errors where the brackets are not

matching, then the program will print out a message that this typed code is incorrect. This will do the same thing for if the bracket is empty, a message will still come up and say that the brackets are incorrect.

In the last part of this code, we have a few examples that are used to illustrate how this code will run to determine if the delimiters are matching or not. The first line we test out is: String at 1 = "a[b+{c*{u-r)]}". Here we can see that there are a pair of curly brackets that open, yet this programmer must have tried to close them with a parenthesis instead of another curly bracket. When the program below runs through this line of code, it will push each of the opening brackets we have here which is { and [. From here, the program already notices that a (is not used to open anything. When the program further goes into the pop algorithm, it notices the closings for the { and] but not the). Therefore, at the end of this example we get a message output that states that the brackets are incorrect here.

In the next example we have the code String at2 = "a+b+c*u-r])]". Here we can see that there are no opening brackets, and only closing brackets for the given brackets which include])]. With that being said, when the code is run and the push algorithm begins, it will push for opening brackets but there are not any opening brackets. Since there are no opening brackets, the algorithm cannot push anything, and therefore stops and creates a message that there are incorrect brackets within this line of code. When nothing gets pushed with the push algorithm, then there is not anything that can be popped off with the pop algorithm, and that is why the message immediately appears that there are incorrect brackets within the line of code.

In the last incorrect example of code, both the push and pop algorithm are used. The third example shows the line of code: String at3 = "a[{b+{c*[u-r)}]". Here we can see the opening brackets include two [opening brackets, and two { opening brackets. With that being said, when

this program begins running, the push algorithm pushes the two [opening brackets and the two { opening brackets, then the pop algorithm comes next to check the closing brackets. While the program identifies the closing brackets, we can see that there is only one) closing bracket, one } closing bracket, and one] closing bracket. With this, as the pop algorithm runs through this line of code, it will notice that there are missing one closing } bracket and one] bracket, in addition to a) closing bracket which was never opened in the first place. As the other two brackets were pushed in the push algorithm, a (bracket was never pushed in the first place, so when the pop algorithm identifies the) closing bracket along with the missing] and } brackets, a message pops up stating that these brackets are incorrect.

In the last example of code, we have a correct line of code with matching delimiters. With that being said, there is no message that would pop up stating that anything is incorrect since we can clearly see that each bracket that opens has a matching one to close them. This examples line is: String at4 = "a[{b+c*(u-r)}]". This line of code illustrates matching delimiters since we have one [bracket, one { bracket, and one (bracket. With that, the push algorithm pushes each of these brackets, and then when the pop algorithm comes into play, it checks each of the closing brackets. In this line of code we can see that each of the opening brackets matches all of the closing brackets with no extra or missing brackets! With that said, we do not get any messages indicating an incorrect bracket issue, and all of the delimiters match!

In conclusion, the push and pop algorithms are very useful algorithms when it comes to matching delimiters in typed code. There are many steps that are to be taken in order for this process to be done successfully, however without these algorithms typing code would be more difficult than normal. Matching delimiters makes typed code successful and allows for the code within the delimiters to be executed! The three examples that are provided in the code below

illustrate how exactly the push and pop algorithms work hand in hand so that people who are typing out their code know what they're missing or doing wrong when it comes to matching delimiters in their work. There are three thorough examples that provide issues with matching the brackets in their code, and one example that illustrates how a correct line of code would be executed without any delimiter issues. In the end, these algorithms play a very important role when it comes to coding properly.

JAVA CODE:

```
package com210proj1;

public boolean checkCorrectBrackets (String s) {
Stack<Character> at = new Stack<Character>();
for(int i = 0; i < a.length(); i++) {
    char c = a.charAt(i);
    switch (c) {
case '{':
case '[':
case '(':
    st.push(c);
    break;
case '}':
case ']':
case ')':
if(!st.isEmpty()) [
        char ch = st.pop()
char ch == '{' && c != '}' | |
    if(ch == '[' && c != ']' | |
     ch == '(' && c != ')') {

     System.out.println("Incorrect Backets"): + c);
    return false;
}

} else {
 System.out.println("Incorrect Backets");
}

break;
}
}

if(!st.isEmppty()) {
 System.out.println("Incorrect Backets");
return false;
}
```

```
return true;
}
}

public class COM210Proj1 {

public static void main(String [] args) {

    StackImp1 a = new StackImp1 ();

   String at = "a[b+{c*{u-r)]}";
   System.out.println(a.checkCorrectBracket(at));

   String at1 = "a[b+{c*{u-r)]}";
   System.out.println(a.checkCorrectBracket(at1));

   String at2 = "a+b+c*u-r]|)]";
   System.out.println(a.checkCorrectBracket(at2));

   String at3 = "a[{b+{c*[u-r)}]";
   System.out.println(a.checkCorrectBracket(at3));

   String at4 = "a[{b+c*(u-r)}]";
   System.out.println(a.checkCorrectBracket(at4));
}

}
```