# **Lab 3 Using a Stack to check for balanced parentheses**

# **Week beginning 2nd October 2023**

When analyzing arithmetic expressions, it is important to determine whether an expression is balanced with respect to parentheses

( a + b \* ( c / ( d – e ) ) ) + ( d / e )

The problem is further complicated if braces or brackets are used in conjunction with parentheses

( a + b \* { c / ( d – e )} ) + [ d / e]

The solution is to use stacks!

Likewise, Java compilers use a stack in parsing the syntax of programs to check that (, {, [ and < match up.

Write a method:

public static boolean isBalanced(String expression)

This method returns true if expression is balanced with respect to parentheses, false if not.

When we speak about parentheses here, we will refer to any one of (, {, [ and <.

The following algorithm works by pushing an open parenthesis onto the stack. When we find a close parenthesis, it should match the open one on the top of the stack. If it doesn’t or the stack is empty, then the expression is not balanced.

Algorithm for isBalanced

Create an empty stack of characters

Set balanced to true

Set index to 0 // index indicates current position in expression

while balanced and index < length of expression

ch = next char in expression

if ch is an opening parenthesis

push ch onto stack

else if ch is a closing parenthesis

if stack is empty

set balanced to false

else

pop item from top of stack – save it to topChar

if topChar does not match ch

set balanced to false

endif

endif

endif

increment index

endwhile

if balanced and stack is empty

return true

else

return false

endif

To check for opening and closing parentheses, use methods:

private static boolean isOpen(char ch) – returns true if ch is an opening parenthesis – any one of (, {, [ and <

private static boolean isClose(char ch) – returns true if ch is a closing parenthesis – any one of ), }, ] and >

private static boolean areMatching(char c1, char c2)-This method returns true if c1 and c2 are matching open and close parentheses etc e.g. ‘(‘ and ‘)’ or ‘<’ and ‘>’ etc.

To test your code (use JUnit):

Provide a variety of input expressions displaying the result true or false

Try several levels of nested parentheses

Try nested parentheses where corresponding parentheses are not of the same type – i.e. use (, { , [ and < as Java allows.

e.g. try (2+[5+7]+1) – balanced

Try unbalanced parentheses

No parentheses at all!

Stack implementation in Java:

Note (from the slides):

* java.util provides a class *Stack*
* But the java documentation states:
  + A more complete and consistent set of LIFO stack operations is provided by the [Deque](file:///C:\Program%20Files\Java\jdk1.6.0_21\docs\api\java\util\Deque.html) interface and its implementations, which should be used in preference to this class.
* Use LinkedList as the implementation of Deque interface