Stacks

Dr Yuhua Li

(Slides adapted from Dr Bailin Deng)

Recall: Arrays vs. LinkedList

- Arrays allow for **random access** of elements: each element can be accessed directly in constant time.
  - Typical illustration: a book where each page can be open independently of others

- Linked lists allow for **sequential access** of elements: each element can be accessed only in particular order.
  - Typical illustration: a roll of paper or tape — all prior material must be unrolled in order to get to the data you want

Stacks

- A stack is a collection of objects that are inserted and removed according to the **last-in, first-out** (**LIFO**) principle
  - A user may insert objects into a stack at any time, but may only access or remove the most recently inserted object (the “top”) that remains in the stack

Stacks

- A physical metaphor: a stack of plates

Stacks

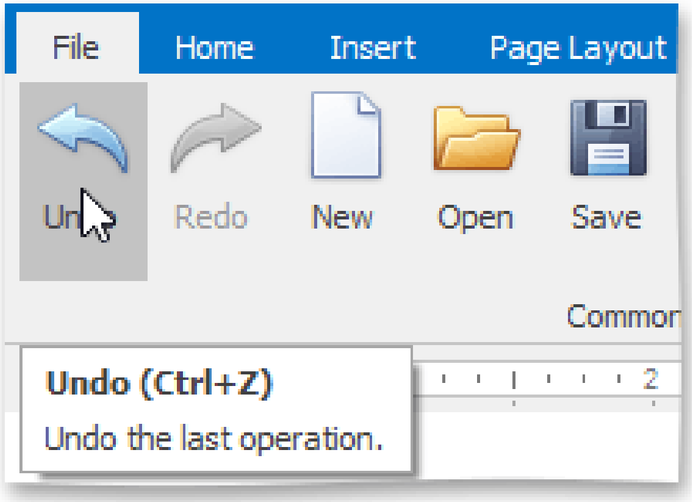
- A physical metaphor: a stack of plates
  - We can add a new plate to the top of the stack

Stacks

- A physical metaphor: a stack of plates
  - We can add a new plate to the top of the stack
  - We can only access or remove the plate at the top

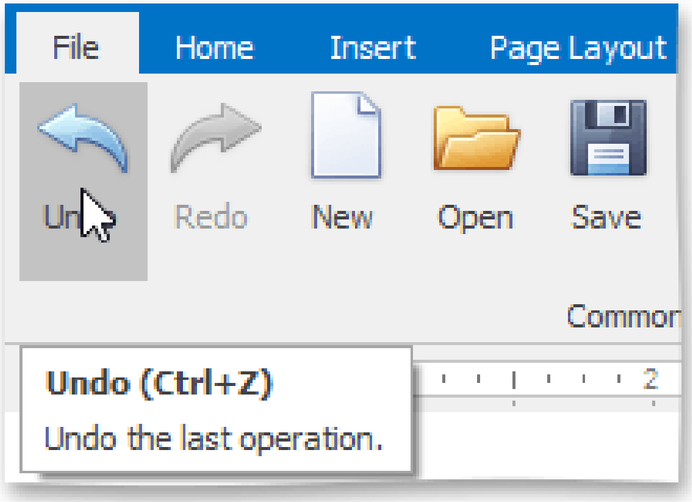
Application of Stacks

- Undo operations
  - Each operation and its results can be inserted into a stack



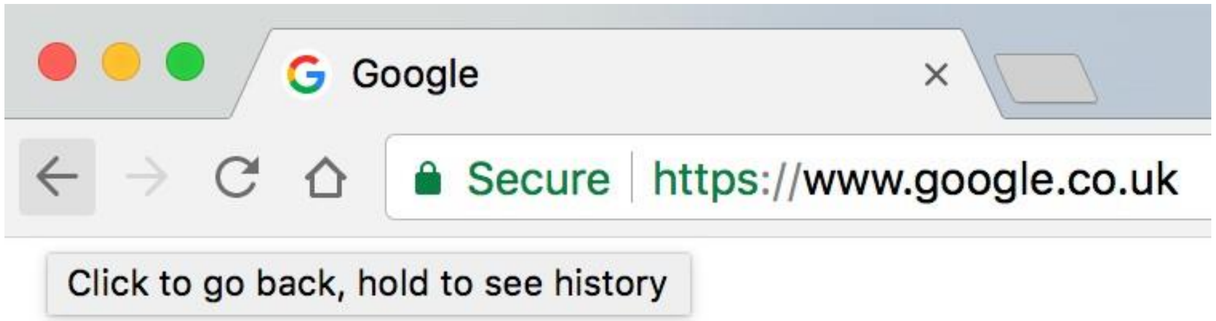
Application of Stacks

- Undo operations
  - Each operation and its results can be inserted into a stack
  - When a user press “undo”, the most recent operation and results are retrieved from the stack



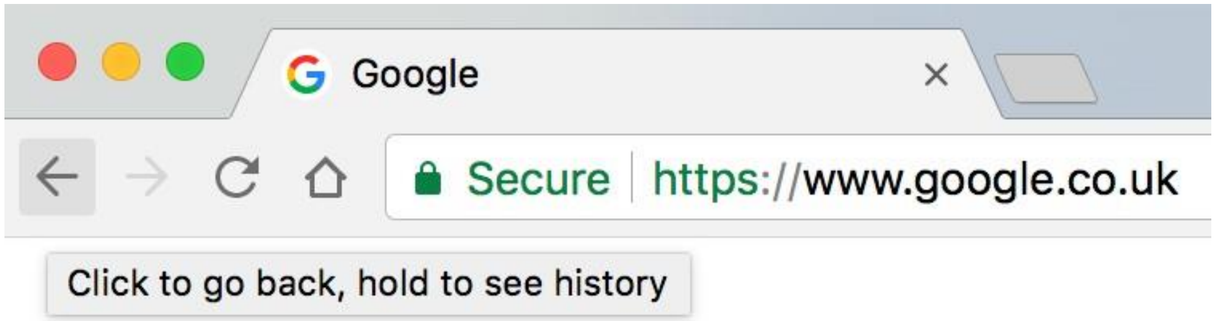
Application of Stacks

- Web browsing history can be stored in a stack



Application of Stacks

- Web browsing history can be stored in a stack
  - When a user clicks “go click”, the most recent webpage is retrieved from the top of the stack



Stack ADT

- A stack should provide the following operations

push(*e*): Adds element *e* to the top of the stack.

pop( ): Removes and returns the top element from the stack (or null if the stack is empty).

Stack ADT

- Additional operations for convenience:

top():	Returns the top element of the stack, without removing it (or null if the stack is empty).
size():	Returns the number of elements in the stack.
isEmpty():	Returns a boolean indicating whether the stack is empty.

Implementation

- A stack can be implemented using a linked list as the underlying container for its elements

```
public class Stack
{
   private DoublyLinkedList elementList;
   …
}
```

Implementation

- A stack can be implemented using a linked list as the underlying container for its elements
  - At initialisation, the underlying list is empty —> the stack is empty

```
public class Stack
{
   private DoublyLinkedList elementList;
   …
}
```

Implementation

- push: new elements is stored at the back of the underlying list

```
void push(e)
{
   elementList.addLast(e);
}
```

Implementation

- push: new elements is stored at the back of the underlying list

```
void push(e)
{
   elementList.addLast(e);
}
```

Time complexity: O(1)

Implementation

- pop: remove and return the last element in the underlying list

```
Element pop()
{
   return elementList.deleteLast();
}
```

Implementation

- pop: remove and return the last element in the underlying list

```
Element pop()
{
   return elementList.deleteLast();
}
```

Time complexity: O(1)

Implementation

- top: return the last element of the underlying list

```
Element top()
{
   return elementList.last();
}
```

Implementation

- top: return the last element of the underlying list

```
Element top()
{
   return elementList.last();
}
```

Time complexity: O(1)

Implementation

- size: the size of the underlying list

```
int size()
{
   return elementList.size();
}

bool isEmpty()
{
   return elementList.size() == 0;
}
```

Implementation

- size: the size of the underlying list

Time complexity: O(1)

```
int size()
{
   return elementList.size();
}

bool isEmpty()
{
   return elementList.size() == 0;
}
```

Implementation

- Time complexity summary:

Method	Running Time
size	$O(1)$
isEmpty	$O(1)$
top	$O(1)$
push	$O(1)$
pop	$O(1)$

Implementation

- A stack can also be implemented using other types of underlying container

Implementation

- A stack can also be implemented using other types of underlying container
- For example, we can use a dynamic array to store the elements
  - push: add an element to the back
  - pop: remove and return an element from the back
  - top: return the last element

Implementation

- A stack can also be implemented using other types of underlying container
- For example, we can use a dynamic array to store the elements
  - push: add an element to the back
  - pop: remove and return an element from the back
  - top: return the last element

Time complexity: O(1)

Implementation

- A stack can also be implemented using other types of underlying container
- The **adapter** design pattern: reusing the interface of an existing class to match a related but different interface

Implementation

- A stack can also be implemented using other types of underlying container
- The **adapter** design pattern: reusing the interface of an existing class to match a related but different interface
  - Example: reusing the linked list / dynamic array interface to match the stack interface
  - The original class is used as a hidden instance in the implementation of the new interface

Application

- A valid mathematical expression should have matching parenthesis
  - ‘(’ and ‘)’
  - ‘[’ and ‘]’
  - ‘{’ and ‘}’

Application

- A valid mathematical expression should have matching parenthesis

- Valid: $[(5 + x) - (y + z)]$

Application

- A valid mathematical expression should have matching parenthesis

- Valid: $[(5 + x) - (y + z)]$

- Invalid: $[(5 + x] - [y + z)]$

Application

- A valid mathematical expression should have matching parenthesis

- Valid: $[(5 + x) - (y + z)]$

opening *closing*

- Invalid: $[(5 + x] - [y + z)]$

Observation: when we see a ‘closing’ parenthesis, the last ‘opening’ parenthesis must match

Application

- A valid mathematical expression should have matching parenthesis

- Valid: $[(5 + x) - (y + z)]$

opening *closing*

- Invalid: $[(5 + x] - [y + z)]$

Algorithm to verify expressions: scan the expression, and match any closing parenthesis with the last opening parenthesis

```
boolean isMatched (String expression)
{
   String opening = “({[”;
   String closing = “)}]”;
   Stack<char> buffer = new Stack<char>();
   for(char c : expression.toCharArray()){
      if(opening.indexOf(c) != -1)
         buffer.push(c);
      else if (closing.indexOf(c) != -1
        if(buffer.isEmpty())
           return false;
        if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
           return false;
      }
   }
   return buffer.isEmpty()
}
```

```
boolean isMatched (String expression)
{
   String opening = "({[";
   String closing = ")}]";
   Stack<char> buffer = new Stack<char>();
   for(char c : expression.toCharArray()){
      if(opening.indexOf(c) != -1)  buffer.push(c);
      else if (closing.indexOf(c) != -1){
        if(buffer.isEmpty())
           return false;
        if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
           return false;
      }
   }
   return buffer.isEmpty()
}
```

Opening parenthesis and their respective closing parenthesis

```
boolean isMatched (String expression)
{
    String opening = “({[”;
    String closing = “)}]”;
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
        if(opening.indexOf(c) != -1)  buffer.push(c);
        else if (closing.indexOf(c) != -1){
          if(buffer.isEmpty())
             return false;
          if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
             return false;
        }
    }
    return buffer.isEmpty()
}
```

stack for opening parenthesis

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
        if(opening.indexOf(c) != -1)  buffer.push(c);
        else if (closing.indexOf(c) != -1){
          if(buffer.isEmpty())
             return false;
          if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
             return false;
        }
    }
    return buffer.isEmpty()
}
```

go through all characters in the expression

```
boolean isMatched (String expression)
{
    String opening = “({[”;
    String closing = “)}]”;
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
        if(opening.indexOf(c) != -1)  buffer.push(c);
        else if (closing.indexOf(c) != -1){
          if(buffer.isEmpty())
             return false;
          if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
             return false;
        }
    }
    return buffer.isEmpty()
}
```

If it is an opening parenthesis, push to the stack

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
        if(opening.indexOf(c) != -1)  buffer.push(c);
        else if (closing.indexOf(c) != -1){
            if(buffer.isEmpty())
                return false;
            if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
                return false;
        }
    }
    return buffer.isEmpty()
}
```

For a closing parenthesis, match it with the top opening parenthesis in the stack

```
boolean isMatched (String expression)
{
    String opening = “({[”;
    String closing = “)}]”;
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
        if(opening.indexOf(c) != -1)  buffer.push(c);
        else if (closing.indexOf(c) != -1){
            if(buffer.isEmpty())
                return false;
            if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
                return false;
        }
    }
    return buffer.isEmpty()
}
```

No opening parenthesis in the stack: invalid

```
boolean isMatched (String expression)
{
    String opening = “({[”;
    String closing = “)}]”;
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
        if(opening.indexOf(c) != -1)  buffer.push(c);
        else if (closing.indexOf(c) != -1){
          if(buffer.isEmpty())
             return false;
          if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
             return false;
        }
    }
    return buffer.isEmpty()
}
```

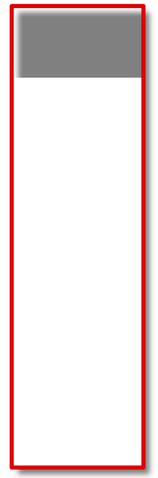
The top opening parenthesis in the stack does not match: invalid

```
boolean isMatched (String expression)
{
    String opening = “({[”;
    String closing = “)}]”;
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
        if(opening.indexOf(c) != -1)  buffer.push(c);
        else if (closing.indexOf(c) != -1){
          if(buffer.isEmpty())
             return false;
          if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
             return false;
        }
    }
    return buffer.isEmpty();
}
```

There is unmatched opening parenthesis in the end: invalid

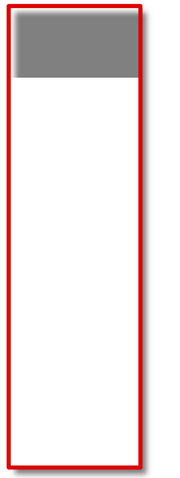
Input: $[(5 + x] - [y + z)]$

```
boolean isMatched (String expression)
{
   String opening = “({[”;
   String closing = “)}]”;
   Stack<char> buffer = new Stack<char>();
   for(char c : expression.toCharArray()){
      if(opening.indexOf(c) != -1)  buffer.push(c);
      else if (closing.indexOf(c) != -1){
        if(buffer.isEmpty())
           return false;
        if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
           return false;
      }
   }
   return buffer.isEmpty()
}
```



buffer

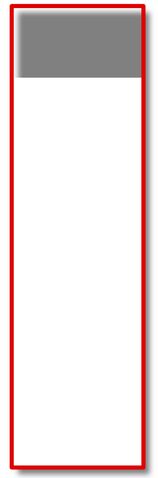
$[(5 + x] - [y + z)]$



buffer

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```

$$[(5 + x] - [y + z)]$$

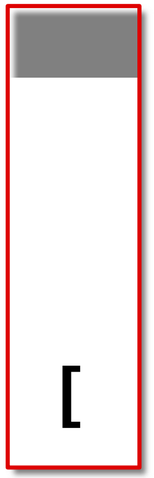


buffer

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
      if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```

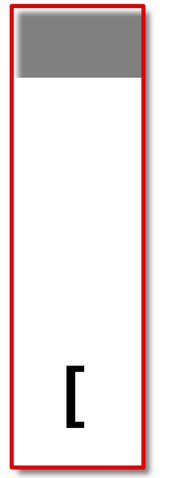
$[(5 + x] - [y + z)]$

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
      if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```



buffer

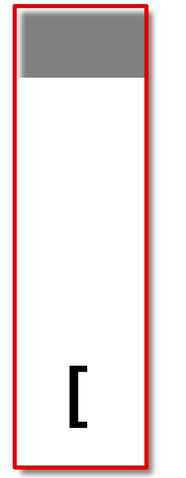
$$[(5 + x] - [y + z)]$$



buffer

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```

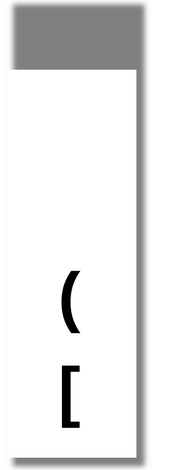
$$[(5 + x] - [y + z)]$$



buffer

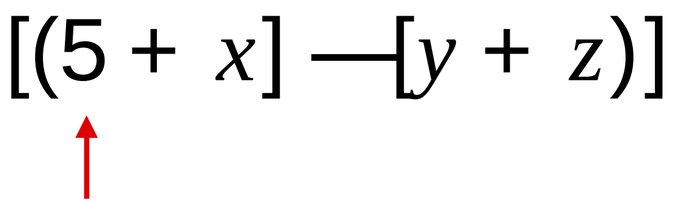
```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
      if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```

$$[(5 + x] - [y + z)]$$



buffer

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
      if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```




```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```

buffer

$$[(5 + x] - [y + z)]$$


```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```

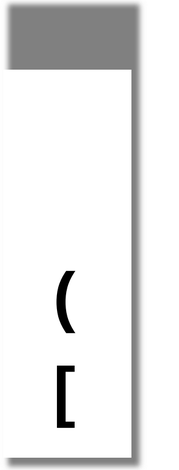
buffer

$$[(5 + x] - [y + z)]$$

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```

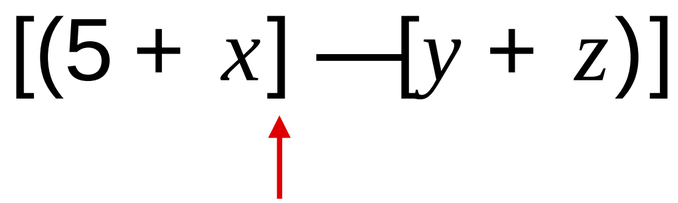
buffer

$$[(5 + x] - [y + z)]$$



buffer

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```

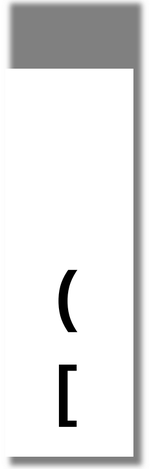



```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
      if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```

buffer

$$[(5 + x] - [y + z)]$$

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
          if(buffer.isEmpty())
             return false;
          if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
             return false;
       }
    }
    return buffer.isEmpty()
}
```



buffer

$$[(5 + x] - [y + z)]$$

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```

buffer

$$[(5 + x] - [y + z)]$$

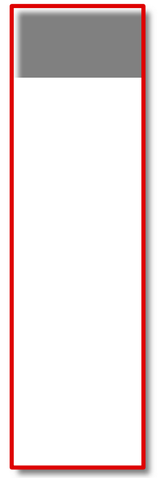
```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```

buffer

Mismatch: invalid

Input: $[(5 + x) - (y + z)]$

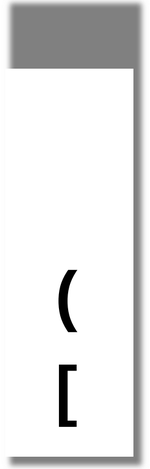
```
boolean isMatched (String expression)
{
   String opening = "({[";
   String closing = ")}]";
   Stack<char> buffer = new Stack<char>();
   for(char c : expression.toCharArray()){
      if(opening.indexOf(c) != -1)  buffer.push(c);
      else if (closing.indexOf(c) != -1){
        if(buffer.isEmpty())
           return false;
        if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
           return false;
      }
   }
   return buffer.isEmpty()
}
```



buffer

$$[(5 + x) - (y + z)]$$

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```



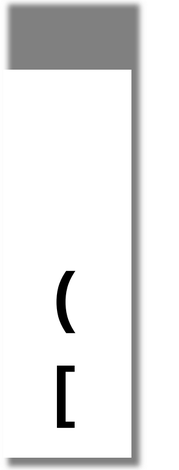
buffer

$$[(5 + x) - (y + z)]$$

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
          if(buffer.isEmpty())
             return false;
          if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
             return false;
       }
    }
    return buffer.isEmpty()
}
```

buffer

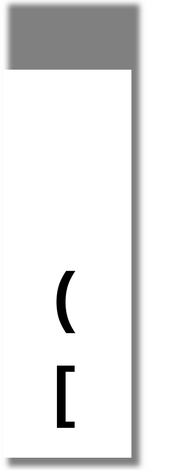
$$[(5 + x) - (y + z)]$$



buffer

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
          if(buffer.isEmpty())
             return false;
          if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
             return false;
       }
    }
    return buffer.isEmpty()
}
```

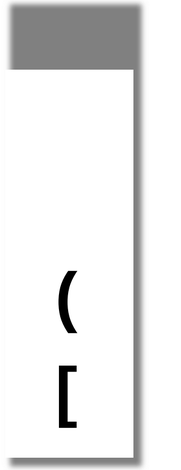
$$[(5 + x) - (y + z)]$$



buffer

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```

$$[(5 + x) - (y + z)]$$

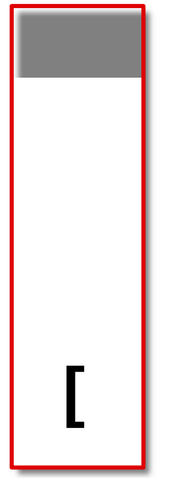


buffer

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```

Matched: remove top element from stack

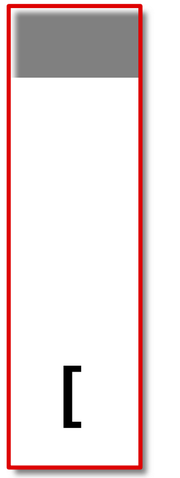
$$[(5 + x) - (y + z)]$$



buffer

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```

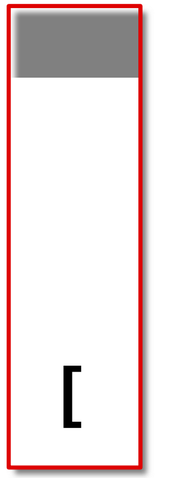
$$[(5 + x) - (y + z)]$$



buffer

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```

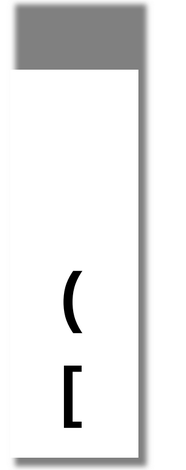
$$[(5 + x) - (y + z)]$$



buffer

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
        else if (closing.indexOf(c) != -1){
          if(buffer.isEmpty())
             return false;
          if(closing.indexOf(c)  != opening.indexOf(buffer.pop()))
             return false;
        }
    }
    return buffer.isEmpty()
}
```

$$[(5 + x) - (y + z)]$$



buffer

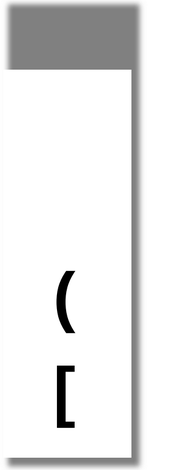
```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
        else if (closing.indexOf(c) != -1){
          if(buffer.isEmpty())
             return false;
          if(closing.indexOf(c)  != opening.indexOf(buffer.pop()))
             return false;
        }
    }
    return buffer.isEmpty()
}
```

$$[(5 + x) - (y + z)]$$


```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```

buffer

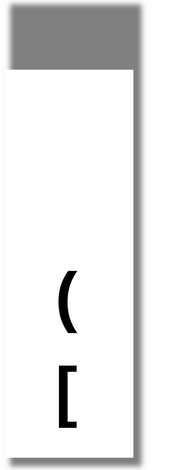
$$[(5 + x) - (y + z)]$$



buffer

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
        else if (closing.indexOf(c) != -1){
          if(buffer.isEmpty())
             return false;
          if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
             return false;
        }
    }
    return buffer.isEmpty()
}
```

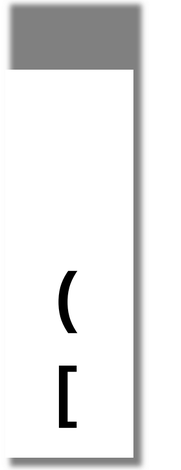
$$[(5 + x) - (y + z)]$$



buffer

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
          if(buffer.isEmpty())
             return false;
          if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
             return false;
       }
    }
    return buffer.isEmpty()
}
```

$$[(5 + x) - (y + z)]$$

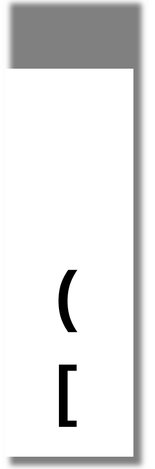


buffer

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```

$$[(5 + x) - (y + z)]$$

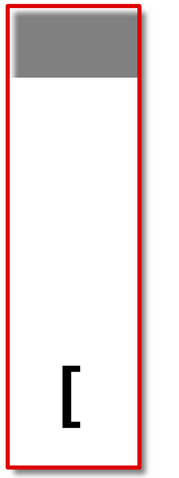
```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```



buffer

Matched: remove top element from stack

$$[(5 + x) - (y + z)]$$

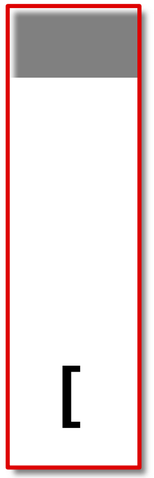


buffer

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```

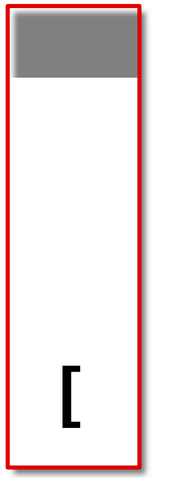
$$[(5 + x) - (y + z)]$$

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
      if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```



buffer

$$[(5 + x) - (y + z)]$$

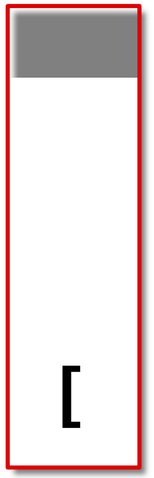


buffer

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)   buffer.push(c);
       else if (closing.indexOf(c) != -1){
          if(buffer.isEmpty())
             return false;
          if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
             return false;
       }
    }
    return buffer.isEmpty()
}
```

$$[(5 + x) - (y + z)]$$

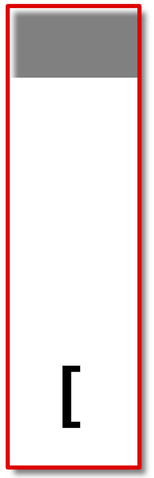
```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```



buffer

$$[(5 + x) - (y + z)]$$

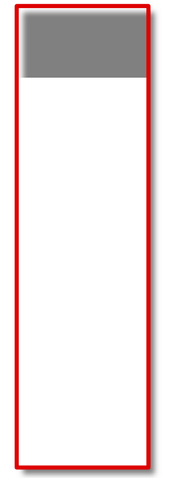
```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```



buffer

Matched: remove top element from stack

$$[(5 + x) - (y + z)]$$

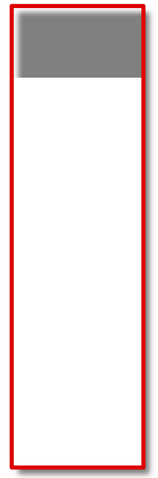


buffer

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```

$$[(5 + x) - (y + z)]$$

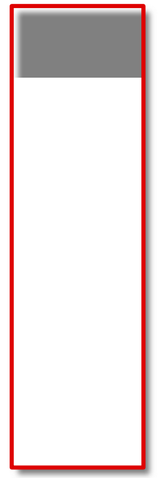
```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c) != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```



buffer

$$[(5 + x) - (y + z)]$$

```
boolean isMatched (String expression)
{
    String opening = "({[";
    String closing = ")}]";
    Stack<char> buffer = new Stack<char>();
    for(char c : expression.toCharArray()){
       if(opening.indexOf(c) != -1)  buffer.push(c);
       else if (closing.indexOf(c) != -1){
         if(buffer.isEmpty())
            return false;
         if(closing.indexOf(c)  != opening.indexOf(buffer.pop()))
            return false;
       }
    }
    return buffer.isEmpty()
}
```



buffer

No opening parenthesis left: matching succeeded