ECS 140A Homework 5 – Problem 3

1 Haskell

Step 1: Algorithm/Pseudocode

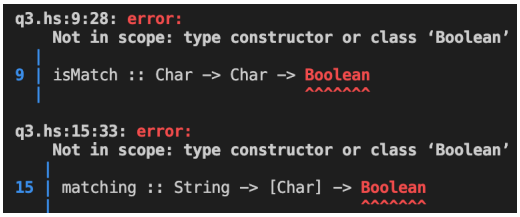
```
isMatch(char, char) -> bool
    if char1 matches parenthesis type char2, return true
    else return false

matchingHelper(string, list) -> bool
    if both string and list are empty, return true
    if first char in "({[", add to list and recurse on rest of string
    else if char in ")}]"
        call isMatch with last item in list and char
        if false returned, return false; else pop list and recurse on rest of string
    else recurse on rest of string with same list

matching(string) -> bool
    call matchingHelper with empty list and return same boolean value
```

Step 2: Actual Code

```
isMatch :: Char -> Char -> Bool
isMatch open close =
    (open == '(' && close == ')') ||
    (open == '{' && close == '}') ||
    (open == '[' && close == ']')

matchingHelper :: String -> [Char] -> Bool
matchingHelper str stack = do
    if str == [] && stack == [] then True
    else do
        let (x:xs) = str
        if x == '(' || x == '{' || x == '['
            then matchingHelper (stack ++ x) xs
        else if x == ')' || x == '}' || x == ']' then do
            if stack == [] then False
            else if isMatch (last stack) x
                then matchingHelper (take (length stack - 1) stack) xs
            else False
        else matchingHelper stack xs
```

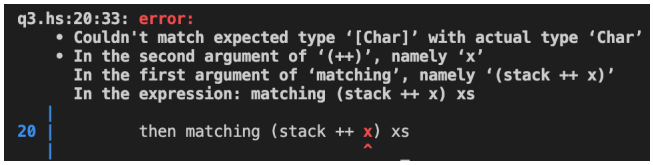
```haskell
matching :: String -> Bool
matching str = matchingHelper str []

main = do
    print (matching "()")
    print (matching "[a(b])")
```

Syntax Error 1



This error was raised because the boolean type should be `Bool` instead of `Boolean`.

Syntax Error 2



This error was raised because I forgot to enclose x (which is type `Char`) into a list in order to append to the stack.

Step 3: Working Code

```haskell
isMatch :: Char -> Char -> Bool
isMatch open close =
    (open == '(' && close == ')') ||
    (open == '{' && close == '}') ||
    (open == '[' && close == ']')

matchingHelper :: String -> [Char] -> Bool
matchingHelper str stack = do
    if str == [] && stack == [] then True
    else do
        let (x:xs) = str
        if x == '(' || x == '{' || x == '['
            then matchingHelper (stack ++ [x]) xs
```

```
        else if x == ')' || x == '}' || x == ']' then do
            if stack == [] then False
            else if isMatch (last stack) x
                then matchingHelper (take (length stack - 1) stack) xs
            else False
        else matchingHelper stack xs

matching :: String -> Bool
matching str = matchingHelper str []

main = do
    print (matching "()")
    print (matching "[a(b])")
```

Step 4: Debug Process

Bug 1

```
False
q3: q3.hs:19:13-24: Non-exhaustive patterns in x : xs
```

This is the output when running the given test cases. This is because I mixed up the order of the arguments when recursing on the `matchingHelper` function.

Buggy code:

```
matchingHelper :: String -> [Char] -> Bool
matchingHelper str stack = do
    if str == [] && stack == [] then True
    else do
        let (x:xs) = str
        if x == '(' || x == '{' || x == '['
            then matchingHelper (stack ++ [x]) xs
        else if x == ')' || x == '}' || x == ']' then do
            if stack == [] then False
            else if isMatch (last stack) x
                then matchingHelper (take (length stack - 1) stack) xs
            else False
        else matchingHelper stack xs
```

Fixed code:

```
matchingHelper :: String -> [Char] -> Bool
matchingHelper str stack = do
    if str == [] && stack == [] then True
    else do
        let (x:xs) = str
        if x == '(' || x == '{' || x == '['
            then matchingHelper xs (stack ++ [x])
        else if x == ')' || x == '}' || x == ']' then do
            if stack == [] then False
            else if isMatch (last stack) x
                then matchingHelper xs (take (length stack - 1) stack)
            else False
        else matchingHelper xs stack
```

Bug 2

```
q3: q3.hs:19:13-24: Non-exhaustive patterns in x : xs
```

This was the output when running the test case `"a{abc([])"`. This message was raised because we reached the end of the string but there was still items left in the stack. I forgot to account for this case.

Buggy code:

```
matchingHelper :: String -> [Char] -> Bool
matchingHelper str stack = do
    if str == [] && stack == [] then True
    else do
        ...
```

Fixed code:

```
matchingHelper :: String -> [Char] -> Bool
matchingHelper str stack = do
    if str == [] && stack == [] then True
    else if str == [] && stack /= [] then False
    else do
        ...
```

After fixing the above bugs, all given and additional test cases passed and behaved as expected. Below are the outputs:

```
(base) Annas-MacBook-Pro-2:hw5 annachen$ ghc q3.hs
[1 of 1] Compiling Main             ( q3.hs, q3.o )
Linking q3 ...
(base) Annas-MacBook-Pro-2:hw5 annachen$ ./q3
True
False
True
True
False
False
True
False
False
False
True
False
False
```

Step 5: Add Documentation

```haskell
-- Check if open parenthesis character matches close parenthesis character
isMatch :: Char -> Char -> Bool
isMatch open close =
    (open == '(' && close == ')') ||
    (open == '{' && close == '}') ||
    (open == '[' && close == ']')

-- Helper function that recurses on str[1:] and updates stack of open parentheses as needed
matchingHelper :: String -> [Char] -> Bool
matchingHelper str stack = do
    -- empty str check
    if str == [] && stack == [] then True
    else if str == [] && stack /= [] then False
    else do
        let (x:xs) = str
        if x == '(' || x == '{' || x == '[' -- open parenthesis char check
            then matchingHelper xs (stack ++ [x]) -- append to stack and recurse
        else if x == ')' || x == '}' || x == ']' then do -- close parenthesis char check
            if stack == [] then False -- extra closing
            else if isMatch (last stack) x
                then matchingHelper xs (take (length stack - 1) stack) -- pop stack and
                    recurse
            else False
        else matchingHelper xs stack -- if other chars, "do nothing"

-- Uses helper function in order to include stack
matching :: String -> Bool
matching str = matchingHelper str []
```

```haskell
main = do
    -- given cases
    print (matching "()") -- true
    print (matching "[a(b])") -- false

    -- additional cases
    print (matching "") -- true
    print (matching "[a(b)]") -- true
    print (matching "a{abc([])") -- false
    print (matching "aabc([])}c") -- false
    print (matching "abc") -- true
    print (matching ")") -- false
    print (matching "(") -- false
    print (matching "(]") -- false
    print (matching "({[()]})") -- true
    print (matching "({[()]}") -- false
    print (matching "{[()]})") -- false
```

Step 6: Extra Test Cases Used

- `""`
- `"[a(b)]"`
- `"a{abc([])"`
- `"aabc([])}c"`
- `"abc"`
- `")"`
- `"("`
- `"(]"`
- `"({[()]})"`
- `"({[()]}"`
- `"{[()]})"`