ECS 140A Homework 1 – Problem 1

1 Python

Step 1: Algorithm/Pseudocode

```
matching_parentheses(string):
    # use an array as a stack
    # iterate over input string
        # if character is open parenthesis, push onto stack
        # if character is close parenthesis, check stack to see if item to pop is the
        # same type of parenthesis
            # if parenthesis type matches, do pop operation
            # if no match, return false
        # ignore all other characters since only parentheses matter
    # return true if stack is empty after iterating over entire input string
```

Step 2: Actual Code

```
def is_matching_type(open_paren, close_paren):
    return (open_paren == '(' and close_paren == ')') \
        or (open_paren == '{' and close_paren == '}') \
        or (open_paren == '[' and close_paren == ']')

def matching_parentheses(string):
    stack = []
    for ch in string:
        if ch in "({[":
            stack.append(ch)
        elif ch in ")}]":
            if is_matching_type(stack[-1], ch):
                stack.pop()
            else:
                return False
    return True if len(stack) == 0 else False
```

Step 3: Working Code

There were no syntax errors, so the initial working code was the same as the previous step.

Step 4: Debug Process

Error when trying to run `matching_parentheses("aabc([])}c")`:

```
Traceback (most recent call last):
  File "matching_parentheses.py", line 50, in <module>
    print(matching_parentheses(string))
  File "matching_parentheses.py", line 37, in matching_parentheses
    if is_matching_type(stack[-1], ch):
IndexError: list index out of range
```

This error was raised because the program tried to access an empty stack; hence an IndexError. More specifically in this test case, it was due to finding a close parenthesis and having all open parentheses seen thus far already popped off the stack. To fix this error, I checked to see if the stack is non-empty before accessing items in the stack as seen in the following:

```
def is_matching_type(open_paren, close_paren):
    return (open_paren == '(' and close_paren == ')') \
        or (open_paren == '{' and close_paren == '}') \
        or (open_paren == '[' and close_paren == ']')

def matching_parentheses(string):
    stack = []
    for ch in string:
        if ch in "({[":
            stack.append(ch)
        elif ch in ")}]":
            if len(stack) == 0 or not is_matching_type(stack[-1], ch):
                return False
            else:
                stack.pop()
    return True if len(stack) == 0 else False
```

```
achen00@ad3.ucdavis.edu@pc20:~/ecs140a/hw1$ python3 matching_parentheses.py
(): True
[a(b)]: True
[a(b]): False
a{abc([]): False
aabc([])}c: False
: True
abc: True
): False
(: False
(]: False
({[()]}): True
({[()]}: False
{[()]}): False
```

Above shows the execution of the program after fixing the IndexError and test cases (behaves as expected).

Step 5: Add Documentation

```
def is_matching_type(open_paren, close_paren):
    '''
    Takes 2 characters as input (a close parenthesis and an open parenthesis) and checks to see if
    they are of the same parenthesis type. If they are, this function returns True; otherwise,
    returns False
    '''
    return (open_paren == '(' and close_paren == ')') \
        or (open_paren == '{' and close_paren == '}') \
        or (open_paren == '[' and close_paren == ']')
```

```
def matching_parentheses(string):
    '''
    Takes a string as input and checks to see if all parentheses in it match. If all parentheses
    match, this function returns True; otherwise, returns False.
    '''
    stack = []
    for ch in string:
        if ch in "({[": # open parenthesis case
            stack.append(ch)
        elif ch in ")}]": # close parenthesis case
            if len(stack) == 0 or not is_matching_type(stack[-1], ch):
                return False
            else: # if ch and stack[-1] match parenthesis type
                stack.pop()
    return True if len(stack) == 0 else False
```

Step 6: Extra Test Cases Used

- `""`
- `"abc"`
- `")"`
- `"("`
- `"(]"`
- `"({[()]})"`
- `"({[()]}"`
- `"{[()]})"`

2 C++

Step 2: Actual Code

```
#include <iostream>
#include <string>
#include <vector>

bool is_matching_type(char open_paren, char close_paren) {
    return (open_paren == '(' && close_paren == ')')
        || (open_paren == '{' && close_paren == '}')
        || (open_paren == '[' && close_paren == ']');
}

bool matching_parentheses(std::string str) {
    std::vector<char> stack;
    for (int i = 0; i < str.length(); i++) {
        if (str[i] == '(' || str[i] == '{' || str[i] == '[') {
            stack.push_back(str[i]);
        } else if (str[i] == ')' || str[i] == '}' || str[i] == ']') {
            if (stack.size() == 0 or !is_matching_type(stack.back(), str[i]))
                return false;
            else
                stack.pop_back();
        }
    }
    return stack.size() == 0 ? true : false;
}
```

```
matching_parentheses.cpp: In function 'bool matching_parentheses(std::__cxx11::string)':
matching_parentheses.cpp:12:23: error: comparison between signed and unsigned integer expressions [-Werror=sign-compare]
     for (int i = 0; i < str.length(); i++) {
                     ~~^~~~~~~~~~~~~~
cc1plus: all warnings being treated as errors
```

This warning was raised due to a comparison between a signed integer and unsigned integer type. One way to fix this is to make `i` of type `size_t`. However, I decided to fix this issue by simplifying the code and improving readability using a range-based for loop.

Step 3: Working Code

```
#include <iostream>
#include <string>
#include <vector>

bool is_matching_type(char open_paren, char close_paren) {
    return (open_paren == '(' && close_paren == ')')
        || (open_paren == '{' && close_paren == '}')
        || (open_paren == '[' && close_paren == ']');
}

bool matching_parentheses(std::string str) {
    std::vector<char> stack;
    for (char ch : str) {
        if (ch == '(' || ch == '{' || ch == '[') {
            stack.push_back(ch);
```

```
        } else if (ch == ')' || ch == '}' || ch == ']') {
            if (stack.size() == 0 or !is_matching_type(stack.back(), ch))
                return false;
            else
                stack.pop_back();
        }
    }
    return stack.size() == 0 ? true : false;
}
```

Step 4: Debug Process

All test cases (given and extra) passed and program behaved as expected.

```
achen00@ad3.ucdavis.edu@pc20:~/ecs140a/hw1$ ./match
(): true
[a(b)]: true
[a(b]): false
a{abc([]): false
aabc([])}c: false
: true
abc: true
): false
(: false
(]: false
({[()]}): true
({[()]}: false
{[()]}): false
```

Step 5: Add Documentation

```
#include <iostream>
#include <string>
#include <vector>

/*
Takes 2 characters as input (a close parenthesis and an open parenthesis) and checks to see if
they are of the same parenthesis type. If they are, this function returns True; otherwise,
returns False
*/
bool is_matching_type(char open_paren, char close_paren) {
    return (open_paren == '(' && close_paren == ')')
        || (open_paren == '{' && close_paren == '}')
        || (open_paren == '[' && close_paren == ']');
}

/*
Takes a string as input and checks to see if all parentheses in it match. If all parentheses
match, this function returns True; otherwise, returns False.
*/
bool matching_parentheses(std::string str) {
    std::vector<char> stack;
    for (char ch : str) {
        if (ch == '(' || ch == '{' || ch == '[') { // open parenthesis case
            stack.push_back(ch);
        } else if (ch == ')' || ch == '}' || ch == ']') { // close parenthesis case
```

```
            if (stack.size() == 0 or !is_matching_type(stack.back(), ch))
                return false;
            else // if ch and last item pushed in stack are matching parenthesis type
                stack.pop_back();
        }
    }
    return stack.size() == 0 ? true : false;
}
```

3 Rust

Step 2: Actual Code

```
fn is_matching_type(open_paren: char, close_paren: char) -> bool {
    (open_paren == '(' && close_paren == ')') ||
    (open_paren == '{' && close_paren == '}') ||
    (open_paren == '[' && close_paren == ']')
}

fn matching_parentheses(s: String) -> bool {
    let mut stack = Vec::new();
    for ch in s {
        if "({[".contains(ch)
            stack.push(ch);
        else if ")}]".contains(ch)
            if stack.len() == 0 || !is_matching_type(stack[stack.len()-1], ch)
                return false;
            else
                stack.pop();
    }
    return if stack.len() == 0 { true } else { false };
}
```

```
error[E0277]: `String` is not an iterator
 --> src/main.rs:9:15
  |
9 |     for ch in s {
  |               ^ `String` is not an iterator; try calling `.chars()` or `.bytes()`
  |
  = help: the trait `Iterator` is not implemented for `String`
  = note: required because of the requirements on the impl of `IntoIterator` for `String`
```

This error was raised because type String is not iterable. I wanted to iterate over the characters in variable `s`. I incorporated the suggested fix the error message had of using `.chars()`.

```
error: expected `{`, found `stack`
  --> src/main.rs:11:13
   |
10 |         if "({[".contains(ch)
   |         -- this `if` expression has a condition, but no block
11 |             stack.push(ch);
   |             ^^^^^----------
   |             |
   |             expected `{`
   |             help: try placing this code inside a block: `{ stack.push(ch); }`
```

This error was raised because Rust is a block-scoped language. So, even when an if block only has 1 statement, curly braces are still required and cannot be omitted. To fix this, I added curly braces to all if-else blocks.

Step 3: Working Code

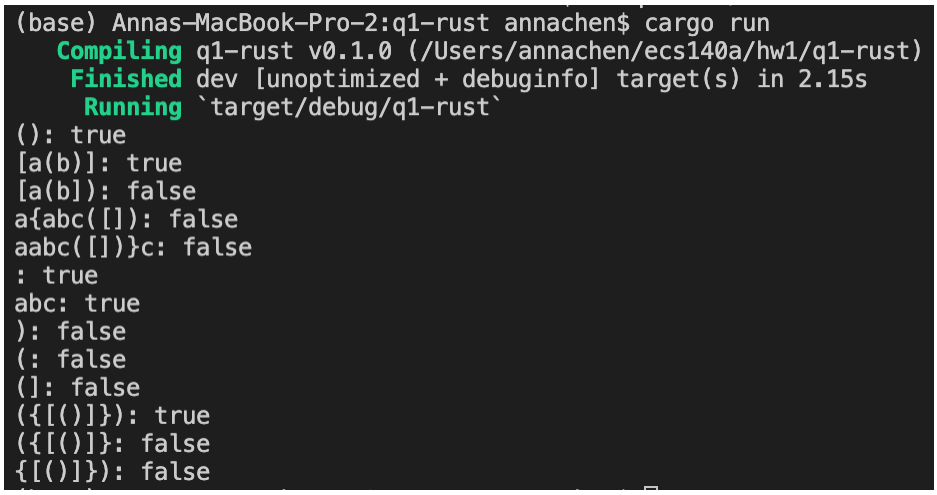
```
fn is_matching_type(open_paren: char, close_paren: char) -> bool {
    (open_paren == '(' && close_paren == ')') ||
```

```
    (open_paren == '{' && close_paren == '}') ||
    (open_paren == '[' && close_paren == ']')
}

fn matching_parentheses(s: String) -> bool {
    let mut stack = Vec::new();
    for ch in s.chars() {
        if "({[".contains(ch) {
            stack.push(ch);
        } else if ")}]".contains(ch) {
            if stack.len() == 0 || !is_matching_type(stack[stack.len()-1], ch) {
                return false;
            }
            else {
                stack.pop();
            }
        }
    }
    return if stack.len() == 0 { true } else { false };
}
```

Step 4: Debug Process

All test cases (given and extra) passed and program behaved as expected.



Step 5: Add Documentation

```
/// Takes 2 characters as input (a close parenthesis and an open parenthesis) and checks to see if
/// they are of the same parenthesis type. If they are, this function returns True; otherwise,
/// returns False
fn is_matching_type(open_paren: char, close_paren: char) -> bool {
    (open_paren == '(' && close_paren == ')') ||
    (open_paren == '{' && close_paren == '}') ||
    (open_paren == '[' && close_paren == ']')
}

/// Takes a string as input and checks to see if all parentheses in it match. If all parentheses
/// match, this function returns True; otherwise, returns False.
```

```
fn matching_parentheses(s: String) -> bool {
    let mut stack = Vec::new(); // using a vector for stack
    for ch in s.chars() {
        if "({[".contains(ch) { // open parenthesis case
            stack.push(ch);
        } else if ")}]".contains(ch) { // close parenthesis case
            if stack.len() == 0 || !is_matching_type(stack[stack.len()-1], ch) {
                return false;
            }
            else { // if ch and last item pushed to stack match parenthesis type
                stack.pop();
            }
        }
    }
    return if stack.len() == 0 { true } else { false };
}
```