Lab 3.1 – Matching Parentheses

Learning Goals

1) Develop your ability to program in C and use a Unix shell.
2) Develop your understanding of the stack data structure.

Your Task

In any well-formed document, program, or math expression, the parentheses and brackets match. Each opening parenthesis has a matching closing parenthesis, and so forth.

Your task is to write a program that checks if all of the parentheses (), brackets [], and curly brackets {} in a document match. The program reads from standard input. The input can contain multiple lines and arbitrary text between and among the opening and closing grouping symbols.

An example legal document is:

```
[ Hello { } ( [ 5+6])]
```

An example document with a mismatching brace is:

```
( [ { ] } )
```

Sample Outputs

1) If the input is well formed, the program should output "Well formatted input".

```
$ ./paren < valid2.in
Well formatted input.
```

2) If an invalid character is found, the program must report the line and character number of the error. The program exits after hitting the error.

```
$ ./paren < invalid1.in
Line 2, Char 28: Found ), expected }

$ ./paren < invalid4.in
Line 1, Char 27: Found }, expected ]
```

3) If the stack is full, the program must report the error and exit.

```
$ ./paren < invalid3.in
Error: Stack Full!
```

4) If the stack is empty and a closing token is read, an error must also be reported.

```
$ ./paren < invalid2.in
Line 3, Char 12: Found ). No matching character.
```

Design

A stack is a very good data structure for solving this type of problem. Stacks have the property that the last element pushed on the stack will be the first element popped off of the stack. (LIFO – Last In First Out)

In our case, if the last opening grouping symbol we saw was a parenthesis, then the only current valid closing grouping symbol is a parenthesis.

Or, alternatively, we might see a different opening grouping symbol which would then need to be closed before our parenthesis can close.

Remember: getchar() returns an int that contains the next character from standard input (stdin). The value EOF indicates when the end of the file have been reached.

Starting Code

```
#define STACK_SIZE 100
#include <stdio.h>

char stack[STACK_SIZE];
int sp=0;

/*
 * Pushes character c onto the stack.
 * Returns 0 if successful.
 * Returns -1 if an error occurs.  (Stack full).
 */
int push (char c)
{}

/*
 * Pops next character off the stack.
 * Returns the char if successful.
 * Returns -1 if an error occurs.  (Stack empty).
 */
int pop ()
{}

/*
 * Reads one character at a time from stdin, checking that all (),
 * {}, and [] in the file properly match.
 *
 * If an error in the input is found, an error specifying the line and
 * character position in the file is printed for the user.
 */
int main ()
{}
```