CS 300 Fall 2014 – Programming Assignment 5

Assigned: October 3, 2014 **Due: October 22, 2013**

Stacks implemented based on Array Lists: This assignment involves using a stack of characters to determine whether or not all pairs of brackets, braces, and parentheses in a text file are properly matched and nested.

Write new files `pairs.c` and `pairs.h` which will contain the main program. As a starting point for the stack module (which will go in files `ar_stack.c` and `ar_stack.h`), look at your files `array_list.c` and `array_list.h` from assignment 3. In place of these files, write new files `ar_stack.c` and `ar_stack.h` which will implement a stack based on an array list, but it will be a stack of characters (rather than a stack of `Part`s).

Requirements:

- Create the file `ar_stack.c` containing functions which implement a stack based on an array list, with the following prototypes:

```
void stack_init(Charstack *s);
char stack_top(Charstack *s);
char stack_pop(Charstack *s);
void stack_push(char new_char, Charstack *s);
int stack_empty(Charstack *s);
```

  The function `stack_top` returns the top element of the stack, but doesn't change the stack. The function `stack_pop` returns what was the top element of the stack and also removes this element from the stack. The above prototypes will go in the file `ar_stack.h`, together with the type definitions for the stack:

```
/* Type definitions */
#define MAX_CHARS    1000

typedef struct charstack
{
    char elements[MAX_CHARS];
    int last;
} Charstack;
```

- Your functions `stack_push` and `stack_pop` must have run-time complexity of $O(1)$.
- The `main()` function of your program must go in the file `pairs.c` and must contain a variable of type `Charstack` to which characters read from the input file are pushed and popped as necessary (using the functions `stack_push` and `stack_pop`).
- All access to, and changing of, the data in your `Charstack` done in the `pairs.c` file must make use of functions from `ar_stack.c` and **not** access `Charstack` data members directly. For example, adding a character to the stack might be done by:

```
stack_push(new_char, &stack);
```

- The program must prompt the user for the name of a text file which it will use as the input file to check.
- Some sample runs of your program, for various input files, should give the following results:

  If the input file contains the text:

```
FILE *in_fp = fopen(infile_name, "r");
if (in_fp == NULL)
{
   printf("Error: Could not open input file %s for reading.\n", infile_name);
   exit(EXIT_FAILURE);
}
```

then the program run should look like:

```
This program determines whether or not pairs of brackets, braces, and
parentheses in a text file are properly matched and nested.

Enter the filename of a text file to check: <FILENAME_GOES_HERE>

All pairs of (), [], and {} in the file <FILENAME_GOES_HERE> are properly matched.
```

If the input file contains the text:

```
if (in_fp == NULL)
{
   printf("Error", };
}
```

then the program run should look like:

```
This program determines whether or not pairs of brackets, braces, and
parentheses in a text file are properly matched and nested.

Enter the filename of a text file to check: <FILENAME_GOES_HERE>

Mis-matched pair: opening ( had closing }.
```

If the input file contains the text:

```
if (in_fp == NULL)
{
   printf("Error: Could not open input file %s for reading.\n", infile_name);
   exit(EXIT_FAILURE);
```

then the program run should look like:

```
This program determines whether or not pairs of brackets, braces, and
parentheses in a text file are properly matched and nested.

Enter the filename of a text file to check: <FILENAME_GOES_HERE>

Mis-match: opening { had no closing symbol.
```

- Copy the `makefile` from one of your previous assignments and update it to work for compiling your program.

Hints:

- Although the details are different, some of the ideas shown in class from the application of a stack for converting an arithmetic expression from infix to postfix notation, and then evaluating the postfix expression, are relevant here.
- Once your program is working, try running it with the various `.c` and `.h` files for this assignment as the input file. Most of these files should have perfectly matched opening and closing symbols – there is one possible exception to this – what is it and why? (*This question is for your interest and you don't need to submit answers to it.*)

Reminder:

- Be sure that your program includes your name and ID and the necessary comments, and follows the style standards as listed in the document posted on the Information page in Blackboard, called "Requirements for Programs Submitted for Assignments". Part of the grade will be for style and quality.
- Carefully test your program.
- You are welcome to write your program at home. If you do, be sure to compile and test it in the lab before submitting it.

How to submit your program:

- Submit your files electronically using **~cs300d/bin/handin 5 pairs.c pairs.h ar_stack.c ar_stack.h makefile**
  The first parameter to the handin program ("5") is the assignment number, and the remaining parameters are the file names.