CS 221 C and Systems Programming
Assignment 3

Due date: September 25, 2015
Submit your `zip` file to `Canvas` before 11:59 p.m. on the due date.

1 Stack (40 marks)

Implement a C program to check for balanced parenthesis and brackets. By balanced we mean that every open parenthesis is matched with a corresponding close parenthesis, and parenthesis are properly nested. The program accepts 4 types of parenthesis and bracket: `< >`, `[ ]`, `( )` and `f g`. All other characters are simply ignored. For example, `(x[0(0)0]fabcg)` is balanced while `(a<b)c>` is not balanced.

To discover whether an input string is balanced, the input characters are read one at a time. The character is categorized as either an opening parenthesis, a closing parenthesis, or other type of character. Values of the third category are ignored. When a character of the first category is encountered, it is `pushed` to the stack. When a closing parenthesis is encountered, the top most element of the stack is `pop-ed` and compared with the closing parenthesis. If they match, the program continues processing the next character. If they do not match, the program reports that the input is not balanced. If a closing parenthesis is read and the stack is empty, the input is not balanced. If the stack is empty when the end of the input is reached, the input string is well balanced.

Your program should allow users to enter the input as many times as he/she wishes and then report if the input is balanced or not balanced. The program terminates when users typed `'Q'`. Declare your stack with size 20. As long as the stack is not full, the program is able to read and process the input. Once the stack is full, the program reports that the input is not balanced. When an input is not balanced, remember to call `getchar()` to process all current input and call `make_empty()` to get a flesh stack for the next input. Feel free to use the `stack` code provided in the class for this assignment. Name your file `stack.c`. Use the following 5 inputs to test your program:

```
> stack
Enter your input that contains parenthesis or Q to quit:
(x[0(0)0]fabcg)
The input is balanced.
Enter your input that contains parenthesis or Q to quit:
(a<b)c>
The input is not balanced.
Enter your input that contains parenthesis or Q to quit:
Q(123)
The input is balanced.
Enter your input that contains parenthesis or Q to quit:
((((x[0(0)0]fabcg))))
The input is balanced.
Enter your input that contains parenthesis or Q to quit:
Q
```

2 Recursion (40 marks)

Write the following C functions using recursion and create a library `libutils.a`:

- `void ascending(int n)`: the function prints the numbers $1 \cdots n$ in ascending order.
- `int min(int a[], int start, int end)`: the function returns the smallest element between the indices `start` and `end` (including `start` and `end`) in the parameter array `a`.
- `int mul(int a, int b)`: the function computes the product of two integers `a` and `b`. You are not allowed to use the multiplication operator (`*`) in your program.
- `int count(char s[], char c)`: the function returns the number of times `c` appears in `s`.

Use the following `main` function to test your code:

```
main()
{
   ascending(10);
   printf(``\n'');
   printf(``Expected output is:  12345678910\n'');
   int a[] = { 9, 2, 6, 7, 5, 4, 0, 2, 7, 5 };
   printf(``%d \n'',min(a, 2, 8));
   printf(``Expected output is:  0\n");
   printf(``%d \n'',mul(2,8));
   printf(``Expected output is:  16\n'');
   printf(``%d \n'',mul(-2,8));
   printf(``Expected output is:  -16\n'');
   printf(``%d \n'',mul(2,-8));
   printf(``Expected output is:  -16\n'');
   printf(``%d\n'',mul(-2,-8));
   printf(``Expected output is:  16\n'');
   char email[] =``dai.yo.hhh@gmail.com'';
   printf(``%d \n'',count(email,'.'));
   printf(``Expected output is:  3\n'');
}
```

Name your file `ascending.c`, `min.c`, `mul.c`, `count.c` and `main.c`. Submit your library `libutils.a` with the 5 C files. Your code will be tested by a different `main.c` with different test cases.

3 Macro (20 marks)

Define a macro `swap_m(t,x,y)` that interchanges the two arguments `x`, `y` of type `t`. Also, implement a C function `swap_f(int x, int y)` that behaves the same way for integers. In your `main` function, first test the correctness of your implementation by calling each of the `swap_m` and `swap_f` once. Next, use loops to call `swap_m(t,x,y)` and `swap_f(int* x, int* y)` a large number (`INT_MAX` defined in ⟨limits.h⟩) of times and report their execution time by calling `system("date")` prior and after the macro/function calls. Discuss about your results in terms of which of the two implementations is more efficient and why. Name your file `macro.c`.

Submission Instructions

Similar to Assignment 1, you will generate 1 `typescript` file for compiling and running `stack.c`, `ascending.c`, `min.c`, `mul.c`, `count.c`, `main.c`, `libutils.a` and `macro.c`. Zip your files to `ass3.zip`. Finally, submit your `ass3.zip` to `Canvas`.