**CSCI 4210 — Operating Systems**
**CSCI 6140 — Computer Operating Systems**
**Homework 4 (document version 1.0)**
**Multi-threading with Pthread in C**

## Overview

- This homework is due by 11:59:59 PM on **Thursday, November 5, 2015**. Homeworks are to be submitted electronically.
- This homework will count as 4% of your final course grade.
- This homework is to be completed **individually**. Do not share your code with anyone else.
- You **must** use C for this homework assignment.
- Your program **must** successfully compile and run on Ubuntu v14.04.3 LTS or v15.04.
- Your program **must** successfully compile via `gcc` with no warning messages when using the `-Wall` compiler option.

## Homework Specifications

Based on the third homework, in this fourth homework, you will use C and the POSIX thread (Pthread) library to implement a single-process multi-threaded calculator program. The goal is to work with threads and synchronization.

Overall, your program will read an input file that specifies a set of calculations to be made by using a Scheme/LISP-like format. More specifically, your program will create child threads to perform each calculation, thus forming a thread tree that represents the parsed mathematical expression.

The input file is specified on the command-line as the first argument and consists of one expression to be evaluated.

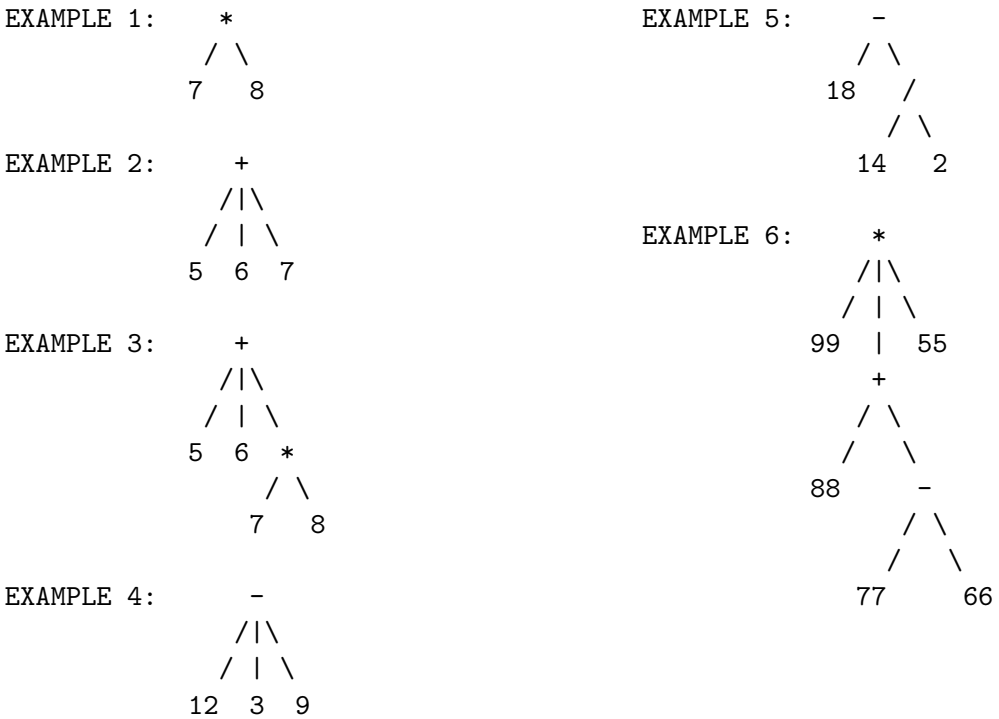
As with the previous homework, you must support addition, subtraction, multiplication, and division, adhering to the standard mathematical order of operations. Note that each of these operations must have at least two operands (display an error if this is not the case).

Example expressions are shown below.

```
# EXAMPLE 1: 7 * 8
#
(* 7 8)

# EXAMPLE 2: 5 + 6 + 7
#
(+ 5 6 7)

# EXAMPLE 3: 5 + 6 + 7 * 8
#
(+ 5 6 (* 7 8))

# EXAMPLE 4: 12 - 3 - 9
#
(- 12 3 9)
```

```
# EXAMPLE 5: 18 - 14 / 2
#
(- 18 (/ 14 2))

# EXAMPLE 6: 99 * (88 + 77 - 66) * 55
#
(* 99 (+ 88 (- 77 66)) 55)
```

The six examples shown above have corresonding "parse trees" shown below.




For the above parse tree diagrams, your program must create child threads that match these trees. More specifically, each node of the tree must correspond to a running thread. Regarding the edges of the tree, each child thread sends a return value to its parent; i.e., each parent thread must join the child thread and collect its integer return value. The return value is either zero (indicating success) or a nonzero value indicating that a failure occurred in the child thread.

Operator threads (i.e., `+`, `-`, `*`, and `/`) must dynamically allocate memory to store intermediate results. The corresponding child threads (i.e., the operands) must modify this intermediate result (thus each child thread must receive a pointer to this dynamically allocated memory).

In the `(* 7 8)` example, the parent thread dynamically allocates memory for a `result` variable, then calls `pthread_create()` twice, i.e., for the `7` and `8` operands. These two child threads modify `result` and (hopefully) return zero (via `pthread_exit()`) to indicate success. The parent thread calls `pthread_join()` for each of its children.

In the `(- 18 (/ 14 2))` example, the parent thread also calls `pthread_create()` twice, this time for the `18` and `(/ 14 2)` operands. The second child thread then calls `pthread_create()` twice, i.e., for the `14` and `2` operands.

Note that you can assume all values given and all values calculated will be integers (including negative values). Therefore, use integer division (i.e., truncate any digits after the decimal point).

## Required Output

When you execute your program, each parent thread must display a message when it parses an operator (i.e., starts an operation). Each child thread must then display a message when it updates the intermediate result in its parent thread.

Further, each parent thread must display a message when an operation is complete (i.e., all operands have been applied to the intermediate result). The topmost (main) thread must display the final answer for the given expression.

For the example `(+ 5 6 (* 7 8))` expression, your program must display the following (though note that thread IDs will likely be different and the order of some lines of output could be different, too, since thread output will be interleaved):

```
THREAD 31091: Starting '+' operation
THREAD 31092: Adding '5'
THREAD 31093: Adding '6'
THREAD 31094: Starting '*' operation
THREAD 31095: Multiplying by '7'
THREAD 31096: Multiplying by '8'
THREAD 31094: Ended '*' operation with result '56'
THREAD 31094: Adding '56'
THREAD 31091: Ended '+' operation with result '67'
THREAD 31091: Final answer is '67'
```

As with previous homeworks, do not include any other debugging statements, though you should be using `#ifdef DEBUG_MODE` to organize your debugging code. See the `main-with-debug.c` example on the course website.

### Synchronization

You must parallelize your program to the extent possible, which will require synchronization among threads.

With multiple child threads updating a shared result (whose memory was dynamically allocated in the parent thread), synchronization among these child threads is required.

Further, the `-` and `/` operators require left-to-right processing, thus further synchronization is required for the operands of these operators.

### Handling Errors

Your program must ensure that the correct number of command-line arguments are included. If not, display an error message and usage information as follows:

```
ERROR: Invalid arguments
USAGE: ./a.out <input-file>
```

If system calls fail, use `perror()` to display the appropriate error message, then exit the program by returning `EXIT_FAILURE`.

If given an invalid expression, display an error message using the format shown below. For example, given `(* 3)`, display the following:

```
THREAD 6431: Starting '*' operation
THREAD 6431: ERROR: not enough operands
```

As another example error, given `(QRST 5 6)`, display the following:

```
THREAD 7330: ERROR: unknown 'QRST' operator
```

In both of the above cases, your program should exit.

Upon return from `pthread_join()`, if the child thread's exit status was not zero, display the following error message:

```
THREAD 1234: child <child-tid> terminated with nonzero exit status <exit-status>
```

Also be sure that all dynamically allocated memory is freed via `free()`.

### Submission Instructions

To submit your homework, please create a single compressed and zipped file (using `tar` and `gzip`). Please include only source and documentation files (i.e., do not include executables or binary files!).

To package up your submission, use `tar` and `gzip` to create a compressed `tar` file using your RCS userid, as in `goldsd.tar.gz`, that contains your source files (e.g., `main.c` and `file2.c`); include a `readme.txt` file only if necessary.

Here's an example showing how to create this file:

```
bash$ tar cvf goldsd.tar main.c file2.c readme.txt
main.c
file2.c
readme.txt
bash$ gzip -9 goldsd.tar
```

Submit the resulting `goldsd.tar.gz` file via the corresponding homework submission link available in LMS (`http://lms9.rpi.edu`). The link is in the Assignments section.