Project 01, CSCI 1730

Exhaustive search
Due date/time are on the Breakout Labs and
Projects webpage

Learning Objectives

- Design and implement a C++ project of moderate size, consisting of a main driver class and multiple files.
- Use the "make" utility, a software engineering tool for managing and maintaining computer programs.
- Use the UNIX command line interface to spawn processes that redirect output or communicate through a pipe.
- Demonstrate knowledge of variable scope rules by predicting program output (i.e. testing your code since it will contain lots of scopes).
- Process command line argument flags in any order
- Decomposing a complicated problem into simpler parts

Problem / Exercise

**You are encouraged to work with one partner on this project,
but you and your partner must both be registered for the same lab section
(note: only one submission on nike is needed for a group of two students).**

Exhaustive search is a problem that is often encountered in computer science and many other disciplines. Exhaustive search involves having a program consider all possible permutations (or combinations) of a problem to find a set of solutions. In this project, you will write a program that will exhaustively search all possible permutations of expressions to print out all expressions that evaluate to a certain value based on command line inputs.

Your program must process command line arguments with flags: "-a" for using a specific set of arithmetic operators or "-b" for using a specific set of bitwise operator, "-e" follow by a single integer, and "-v" followed by 5 integer values where each integer is different (no repeats) and greater than 0. The flags can be in any order, but you may assume the following about the input (which is similar to many Linux/Unix commands).

- Only "-a" or "-b" (not both) will be used for input (this means your program will only consider arithmetic or bitwise operations (not both) during a single run of your program).
- The argument after "-e" will be an integer.
- The next 5 arguments after "-v" will be valid integers as aforementioned.

Given the set of arithmetic operator `A = {+, -, *, /, % }`, bitwise operators `B = {<<, >>, &, |, ^}`, input set of 5 integer values `V` (arguments following the -v flag), and input integer `e`, print all expressions involving all 5 values of `V` and 4 operators from either `A` or `B` (not both) that evaluate to `e` using following the rules.

1. Each expression must contain all 5 values of V without any repeated operands.
2. The 4 operators used in each expression can contain repeated operators, and we are only using the binary operators (operators that use two operands).
3. Each expression must be parenthesized as shown in the example.

4. Each expression is evaluated based on C and C++ rules.

Study the examples in the Examples section to understand the problem before you start to code. After printing all expressions, your program should print the total number (as shown in the examples). Your program's I/O must match the examples given in the Examples section. Also note that a few examples with many lines of output use the `'>'` symbol to redirect output to a file (output file on the breakout labs and projects webpage) and `'|'` symbol to pipe output to wc -l to get a line count. Also note that the redirect output and pipe are not passed as command line arguments to your main method by the operating system. When you test out your program, experiment with piping and redirecting output to a file.

Project Requirements

Your program must adhere to all requirements stated below.

1. Your program must handle command line arguments as shown in the Examples section, and your I/O must be exactly like the examples shown.
2. Your program must run to completion in less than 10 seconds on a cf node (or nike). If your program runs longer than 10 seconds, then use the Unix kill command to end the program (look up the kill command if you don't know how to use it).
3. Your program must compile with all targets, dependencies, and commands given in the Makefile on the breakout labs and projects webpage for this project. This means that you must download, read, understand, and use the Makefile. You CANNOT modify the Makefile, which means you'll need to split your program into multiple files (`proj1.h`, `main.cpp`, `proj1.cpp`), and details about these files are below.
   (a) `proj1.h` must contain all of your prototypes (and other information as needed)
   (b) `main.cpp` must contain the main function that processes command line arguments and passes these arguments to one function found in `proj1.cpp`. The main function cannot do more than process command line arguments and pass them to one function in `proj1.cpp`, the main function must be the only function in this file, and the main function's source code cannot exceed 30 lines of code (this does not include comments). Otherwise, points will be deducted for poor programming style.
   (c) `proj1.cpp` contains all other functions necessary to complete this project.
   (d) All functions/methods should be commented properly.

Examples

```
./proj1.out -a -e 71 -v 7 1 2 3 6
(((7+2)+3)*6)-1
(((7+3)+2)*6)-1
(((2+7)+3)*6)-1
(((2+3)+7)*6)-1
(((3+7)+2)*6)-1
(((3+2)+7)*6)-1
6 arithmetic expressions found that evaluate to 71

./proj1.out -e 71 -v 7 1 2 3 6 -a
(((7+2)+3)*6)-1
(((7+3)+2)*6)-1
(((2+7)+3)*6)-1
(((2+3)+7)*6)-1
(((3+7)+2)*6)-1
(((3+2)+7)*6)-1
6 arithmetic expressions found that evaluate to 71

./proj1.out -v 7 1 2 3 6 -a -e 71
(((7+2)+3)*6)-1
(((7+3)+2)*6)-1
(((2+7)+3)*6)-1
```

```
(((2+3)+7)*6)-1
(((3+7)+2)*6)-1
(((3+2)+7)*6)-1
6 arithmetic expressions found that evaluate to 71

./proj1.out -b -v 5 2 3 11 19 -e 204
(((5<<3)^11)|19)<<2
(((11>>3)<<5)|19)<<2
(((11>>3)<<5)^19)<<2
3 bitwise expressions found that evaluate to 204

./proj1.out -v 5 2 3 11 19 -b -e 204
(((5<<3)^11)|19)<<2
(((11>>3)<<5)|19)<<2
(((11>>3)<<5)^19)<<2
3 bitwise expressions found that evaluate to 204

./proj1.out -e 204 -v 5 2 3 11 19 -b
(((5<<3)^11)|19)<<2
(((11>>3)<<5)|19)<<2
(((11>>3)<<5)^19)<<2
3 bitwise expressions found that evaluate to 204

./proj1.out -v 18 26 17 28 100 -a -e 12 > output1.txt
tail output1.txt
(((100%17)*26)%18)%28
(((100%17)*28)-18)%26
(((100%17)*28)*26)%18
(((100%17)%28)*26)%18
(((100%28)+26)*17)%18
(((100%28)+17)*26)%18
(((100%28)*17)-26)%18
(((100%28)*17)%26)%18
(((100%28)%18)*17)%26
461 arithmetic expressions found that evaluate to 12

./proj1.out -a -e 17 -v 2 3 6 4 12 | wc -l
324

./proj1.out -e 250 -v 10 20 30 40 50 -a | wc -l
115

./proj1.out -b -v 45 23 12 11 9 -e 9 | wc -l
3930

./proj1.out -b -v 45 23 12 11 9 -e 9 > output2.txt
tail output2.txt
(((9>>12)>>23)^11)&45
(((9>>11)|45)&23)^12
(((9>>11)|23)&45)^12
(((9>>11)|23)^12)&45
(((9>>11)|12)^23)&45
(((9>>11)^45)&23)^12
(((9>>11)^23)&45)^12
(((9>>11)^23)^12)&45
(((9>>11)^12)^23)&45
3929 bitwise expressions found that evaluate to 9

./proj1.out -a -e 349 -v 5 7 3 1 9
0 arithmetic expressions found that evaluate to 349

./proj1.out -e 217 -v 5 7 3 1 9 -b
0 bitwise expressions found that evaluate to 217
```

1 Submission

Before the date/time stated on the Breakout Labs and Projects webpage, you need to submit your code on nike. Make sure your work is on `nike.cs.uga.edu` in a directory called `LastName-FirstName-proj01` if you are working alone. If you are working with a partner, then place all of your group work in a directory called `LastName1-FirstName1-LastName2-FirstName2-proj01`. These directories must include all of the following.

1. All source files required for this lab
2. A copy of Makefile for this project
3. A README file filled out correctly (fill out a copy as you did in a previous lab, but remove all brainstorm sections since there are no brainstorms for this project).

To submit from within the parent directory, execute the one of the following commands:

```
$ submit LastName-FirstName-proj01 cs1730
$ submit LastName1-FirstName1-LastName2-FirstName2-proj01 cs1730
```

If the submission is successful, then you'll see that a file that starts with `rec` (your receipt file) was created in the directory that you submitted. It is also a good idea to email a copy to yourself (to do this, modify the command found in Lab 01 for this lab).

2 Grading (50 points)

If your program does not compile on nike or a cf node with an unmodified copy of our Makefile, then you'll receive a grade of 0 on this assignment. Otherwise, your program will will be graded using the criteria below.

Makefile (working as aforementioned)	3 points
README file (filled out correctly)	2 points
All functions/methods are commented properly	5 points
Program runs correctly on various test cases on nike or a cf node	40 points
Late Penalty	-10 points for each 24 hour period late
Penalty for not following instructions (invalid I/O, etc.)	Penalty decided by grader

You must test, test, and retest your code to make sure it compiles and runs correctly on nike or a cf node with any given set of valid inputs. This means that you need to create many examples on your own (that are different than the aforementioned examples) to ensure you have a correctly working program. Your program's I/O must match the examples given in the Examples section. We will only test your program with valid sets of inputs; however, the command line argument flags can be in any order as shown in the examples.