CSC B09 Assignment 1, Winter 2018

Due by the end of Friday February 2, 2018; no late assignments without written explanation.

Please re-read the statement about academic offences at the end of the course information sheet!

Background: A little arithmetic puzzler

1+2+3+4+5+6 is 21. What other values can be made like this? 1–2–3–4–5–6 is –19. But if we don't need an operator between particular digits at all, we can get higher numbers: 12+34+56 is 102, and for that matter we could just write 123456.

Can we use a combination of '+', '–', and juxtaposition to get the value 47?

In this assignment we will write a program to find out.

Your tasks

1. Examine /cmsfaculty/ajr/b09/a1/rightside. This is a shell script which will test whether an input formula yields 47. It misbehaves if given bad input; only give it appropriate input. For example, if the input is the two lines 42+5 and 41+5, it will output only the first, but if the input is the two lines 42+5 and 41+6, it will output both.

Or, if the input is the file /cmsfaculty/ajr/b09/a1/allsix then it will answer the question posed in the introduction to this assignment above.

2. Write "leftside", which is a shell script which will produce all 243 possibilities to supply to rightside for testing; i.e. its output is the contents of allsix. The idea is that you can execute the pipeline "`sh leftside | sh rightside`" to solve the originally-posed problem.

The algorithm for leftside is to consider the numbers from 0 to 242 (inclusive) in base 3. If we consider the numbers as five-digit base 3 numbers, then we can use this string of five digits to separate the six digits 1•2•3•4•5•6. Each base 3 digit indicates whether a particular pair of digits is separated by a minus sign, a plus sign, or nothing at all. If a zero represents a "nothing at all", a one represents a plus sign, and a two represents a minus sign, then the allsix file is the values of 0 to 242 in order.

To explain this further, let's temporarily instead consider the case where we only have the digits 1, 2, and 3. There are three possibilities for each of the two gaps, so we have nine possibilities, encoded as follows, with the left column being a number written in base three:

00	123
01	12+3
02	12–3
10	1+23
11	1+2+3
12	1+2–3
20	1–23
21	1–2+3
22	1–2–3

The "00", for example, represents two juxtapositions; whereas the "11", for example, represents two plus signs.

Actually, it's easier to extract the base three number backwards, by repeatedly dividing by three using integer arithmetic (and integer arithmetic is what *expr* uses for division). So to return to the original problem with digits 1 through 6, the twelfth line (for example) of "allsix" is based on the number eleven:

- we output the digit 1 (always output this first)

(continued)

- 11%3 is 2, so we output a minus sign before the digit 2 (when x%3 is 2 we always output a minus sign)
- 11/3 is 3 (in integer arithmetic), so 3 is our value for the next time around the loop
- 3%3 is 0, so don't output anything before the digit 3 (when x%3 is 0 we always don't output anything before the next digit)
- 3/3 is 1, so 1 is our value for the next time around the loop
- 1%3 is 1, so output a plus sign before the digit 4 (when x%3 is 1 we always output a plus sign)
- 1/3 is 0, so 0 is our value for the next time around the loop
- 0%3 is 0, so don't output anything before the digit 5
- 0/3 is 0, so 0 is our value for the next time around the loop
- 0%3 is 0, so don't output anything before the digit 6

Altogether this yields the string 1–23+456.

Note that you can test your program by typing something like

```
sh leftside | diff - allsix
```

3. Implement leftside in C now, calling it leftside.c. (Be careful not to compile it to the binary file name "leftside" because you don't want to overwrite your shell script!)

Note that there is no need to accumulate the string; just output it as you go. Put the individual characters with putchar as you go. Remember that for a value between 0 and 9 inclusive, the appropriate ASCII character code can be computed as `'0'+x`.

4. Instead of outputting the strings, in C it's quite feasible to compute the value as you go, although this is a bit trickier. You need to keep track of a current number you're assembling by juxtaposition, as well as the sum so far and whether you are adding or subtracting. Try some computations on paper, with a friend or imaginary friend saying the string of juxtaposition/plus/minus symbols as you compute; then implement this in C. If the sum is 47, output your string using your previous leftside.c code; otherwise, output nothing. Thus, this program all by itself has the same output as "`leftside|rightside`". This program should be called "sum47.c".

5. Not part of the assignment but worth thinking about: How would you make a single shell script to perform both the leftside and rightside operations, without external leftside and rightside files?

6. Also not part of the assignment but a recommended exercise because it's more fun than sum47: Modify your sum47.c to become "sum100.c" which does this same exercise for strings containing the digits 1 through 9, outputting which strings yield 100. This is the original form of this problem and works out to be a bit more interesting. (It's feasible in C, but not as a shell script.)

To submit

I suggest you begin by making a new directory to hold your assignment files, plus any other working copies and associated files. You should call your assignment solution files leftside, leftside.c, and sum47.c.

Once you are satisfied with your files, you can submit them for grading with the command

```
submit -c cscb09w18 -a a1 leftside leftside.c sum47.c
```

You may still change your files and resubmit them any time up to the due time. You can check that your assignment has been submitted with the command

```
submit -l -c cscb09w18 -a a1
```

You can also submit files individually instead of all at once, and you can resubmit a particular file by using the –f option, e.g.

```
submit -c cscb09w18 -a a1 -f leftside.c
```

(continued)

This is the only submission method; you do not turn in any paper.

Other notes

For now, we will run our shell scripts by typing "sh file" or "sh file args ...".

Your programs must run on the UTSC linux machines, and they should use standard *sh* and C features only. For the *sh* script, avoid *bash* extensions, and please test it in different shells, not only in *bash*. E.g. also try "dash leftside".

Do not use awk, perl, python, or any other programming language in your shell script besides *sh*. (These are valuable languages to know, but are not the point of question 2 in the current assignment, in which you will be graded on your *sh* programming.)

Please see the assignment Q&A web page at

`https://mathlab.utsc.utoronto.ca/courses/cscb09w18/a1/qna.html`

for other reminders, advice, and answers to common questions.

Remember:

This assignment is due at the end of Friday, February 2, by midnight. Late assignments are not ordinarily accepted and *always* require a written explanation. If you are not finished your assignment by the submission deadline, you should just submit what you have (for partial marks).

Despite the above, I'd like to be clear that if there *is* a legitimate reason for lateness, please do submit your assignment late and send me that written explanation. (For medical notes, I need to see the original, in person. E.g. in office hours or after the next class.)

I'd also like to point out that even a zero out of 7% is far better than cheating and suffering an academic penalty. Don't cheat even if you're under pressure. Whatever the penalty eventually applied for cheating, it will be worse than merely a zero on the assignment. Do not look at other students' assignments, and do not show your assignment (complete or partial) to other students. Collaborate with other students only on material which is not being submitted for course credit.