Dalhousie University
CSCI 2132 — Software Development
Winter 2018
Assignment 4

Distributed Wednesday, February 14 2018.

Due 9:00PM, Wednesday, February 28 2018.

Instructions:

1. The difficulty rating of this assignment is *silver*. Please read the course web page for more information about assignment difficulty rating, late policy (no late assignments are accepted) and grace periods before you start.
2. Each question in this assignment requires you to create one or more regular files on bluenose. Use the exact names (case-sensitive) as specified by each question.
3. C programs will be marked for correctness, design/efficiency, and style/documentation. Please refer to the following web page for guidelines on style and documentation for short C programs (which applies to this assignment):

   `http://web.cs.dal.ca/~mhe/csci2132/assignments.htm`

   You will lose a lot of marks if you do not follow the guidelines on style and documentation.
4. Create a directory named `a4` that contains the following files (these are the files that assignment questions ask you to create): `a4q1.c` and `a4q2.c`. Submit this directory electronically using the command `submit`. The instructions of using `submit` can be found at:

   `http://web.cs.dal.ca/~mhe/csci2132/assignments.htm`
5. Do NOT submit hard copies of your work.

Questions:

1. [15 marks] This question asks you to write a calculator for linear polynomials with integer coefficients.

   A linear polynomial is a polynomial of degree 1. Some examples are $32x - 5$, $-x + 19$, and $75x$. For this question, we also consider constants, i.e., polynomials with degree 0, to be a special case of linear polynomials (they are however not in mathematics). For example, we consider polynomials such as 5 as a special case for linear polynomial.

The operations we support include the addition, subtraction and multiplication of two linear polynomials with integer coefficients. These polynomials are all polynomials in variable x.

The syntax of the language of the input for this calculator is:

```
option left1 left2 right1 right2
```

There are four and only four space characters in the syntax shown above. Here `option`, `left1`, `left2`, `right1` and `right2` are all integers.

The value of `option` can only be one of the following three integers: 1, 2 and 3, which correspond to addition, subtraction and multiplication, respectively. The left and right operands are $\text{left1} \times x + \text{left2}$ and $\text{right1} \times x + \text{right2}$, respectively.

For example, the input `2 4 7 5 15` means $(4x + 7) - (5x + 15)$.

The result of the operation on two linear polynomials can be a polynomial of degrees 2 (in the case of multiplication), 1, or 0. Thus your program will consider the result as a polynomial of degree 2, and print the coefficients from the term of the highest degree to the term of lowest degree. More precisely, the output is always of the following form:

```
result1 result2 result3
```

There is one single space between two consecutive values in the output. There is no space at the end of the line, but there is a trailing newline.

The above output means that the result is $\text{result1} \times x^2 + \text{result2} \times x + \text{result3}$. If the result is a linear polynomial, then result1 is 0. If the result is a constant, then both result1 and result2 are zeroes. **Your program is required to print all three values even if they are zeroes.**

General Requirements: Use `a4q1.c` as the name of your C source code file.

We will use the following command on bluenose to compile your program:

```
gcc -o poly a4q1.c
```

Your program should NOT print anything to prompt for user input. It accepts user input that meets the syntax specified above.

Therefore, when you run your program by entering `./poly`, the program will wait for your input. If you enter `2 4 7 5 15`, the program will output `0 -1 -8`

Error Handling: Your program should print an appropriate error message and terminate if the user inputs a value for `option` that is not 1, 2, or 3.

Testing: To test your program automatically, we will use it as a UNIX filter.

For the example above, we will test it using the following command in the directory containing the executable program:

```
echo "2 4 7 5 15" | ./poly
```

The output should be:

```
0 -1 -8
```

To help you make sure that the output format of your program is exactly what this questions asks for, several files are provided in the following folder on bluenose to show how exactly your program will be automatically tested:

```
/users/faculty/prof2132/public/a4test/
```

In this folder, open the file `a4q1test` to see the commands used to test the program with two test cases. The output files, generated using output redirection, are also given.

To check whether the output of your program on any test case is correct, redirect your output into a file, and use the UNIX utility `diff` (learned in Lab 3) to compare your output file with the output files in the above folder, to see whether they are identical.

Since these output files are given, we will **apply a penalty to any program that does not strictly meet the requirement on output format**. This includes the case in which your output has extra spaces, or has a missing newline at the end.

We will use different test cases to test your program thoroughly. Therefore, construct a number of test cases to test your program thoroughly before submitting it.

Examples: The following are some examples on running the executable file:

```
mhe@bluenose:~/csci2132/a4$ echo "1 5 11 2 0" | ./poly
0 7 11
mhe@bluenose:~/csci2132/a4$ echo "1 0 11 2 0" | ./poly
0 2 11
mhe@bluenose:~/csci2132/a4$ echo "2 0 11 2 0" | ./poly
0 -2 11
mhe@bluenose:~/csci2132/a4$ echo "3 10 11 2 20" | ./poly
20 222 220
mhe@bluenose:~/csci2132/a4$ echo "5 1 2 5 4" | ./poly
Invalid operation.
```

2. [15 marks] This question asks you to write a program to compute the numeric root of a given nonnegative integer x.

   The numeric root of x is computed as follows:

   a) Compute the sum, y, of all of x's (decimal) digits;

   b) If y is greater than 10, then set x to y and go to step a). Otherwise, y is x's numerical root.

   Thus, the numeric root of 10, 202 and 875 are 1, 4 and 2, respectively.

   General Requirements: Use `a4q2.c` as the name of your C source code file.

   We will use the following command on bluenose to compile your program:

   ```
   gcc -o root a4q2.c
   ```

   Your program should NOT print anything to prompt for user input. It accepts user input that is a single integer. It prints a single number that is the numeric root of the number that the user entered, followed by the newline character.

   Therefore, when you run your program by entering `./root`, the program will wait for your input. If you enter `875`, the program will output `2`

   Error Handling: Your program should print an appropriate error message and terminate if the user inputs a value that is negative.

   Testing: To test your program automatically, we will use it as a UNIX filter.

   For the example above, we will test it using the following command in the directory containing the executable program:

   ```
   echo 875 | ./root
   ```

   The output should be:

   ```
   2
   ```

   To help you make sure that the output format of your program is exactly what this questions asks for, several files are provided in the following folder on bluenose to show how exactly your program will be automatically tested:

   ```
   /users/faculty/prof2132/public/a4test/
   ```

In this folder, open the file `a4q2test` to see the commands used to test the program with three test cases. The output files, generated using output redirection, are also given.

To check whether the output of your program on any test case is correct, redirect your output into a file, and use the UNIX utility `diff` (learned in Lab 3) to compare your output file with the output files in the above folder, to see whether they are identical.

Since these output files are given, we will **apply a penalty to any program that does not strictly meet the requirement on output format**. This includes the case in which your output has extra spaces, or has a missing newline at the end.

We will use different test cases to test your program thoroughly. Therefore, construct a number of test cases to test your program thoroughly before submitting it.

Examples: The following are some examples on running the executable file:

```
mhe@bluenose:~/csci2132/a4$ echo 875 | ./root
2
mhe@bluenose:~/csci2132/a4$ echo 10 | ./root
1
mhe@bluenose:~/csci2132/a4$ echo 99 | ./root
9
mhe@bluenose:~/csci2132/a4$ echo -5 | ./root
The input number must be nonnegative.
```