CSC 412 – Operating Systems

Programming Assignment 01, Fall 2023

Tuesday, September 12th, 2022

Due date: Wednesday, September 22nd, 11:59pm.

1 What this Assignment is About

1.1 Objectives

The objectives of this assignment are for you to

- Get started with C programming;
- Write a small bash script to build and launch your executable;
- Make sure that you properly follow assignment specifications.

This is an individual assignment.

1.2 Handout

There is no handout for this assignment besides this document.

2 Part I: A Little C program

2.1 Location of source files

All your programs should be placed inside a folder named `Source` that should be located at the root of the project folder that you submit.

2.2 Version 1: Single argument

2.2.1 Name of the source file

To simplify the graders's job I impose that your source file should be named `prog01v1.c`.

2.2.2 What the program should compute

Your program will take as argument a single strictly positive integer and will print out the list of divisors of that number, listed in increasing order.

For example, if your executable is named `prog` and is launched from the console with the command

```
./prog 36
```

Then your program should produce the following output:

```
The list of divisors of 36 is:  1, 2, 3, 4, 6, 9, 12, 18, 36.
```

2.2.3 Data validation and error reports

You should check the number of arguments of your program and report an eventual error using one of the following messages (again assuming that you built an executable named `prog`):

- ```
  program launched with no argument.
  Proper usage:  ./prog m
  ```
- ```
  program launched with too many arguments.
  Proper usage:  ./prog m
  ```

For this assignment, you don't need to verify that the argument is valid (a strictly positive integer). This will be for extra credit.
Needless to say, the name of the executable should not be hard-coded in your C program.

2.3 Version 2: Two arguments

2.3.1 Name of the source file

Your source file should be named `prog01v2.c`.

2.3.2 What the program should compute

Your program will take as arguments two strictly positive integers and should compute and output the gcd of these two numbers. For example, if your executable is named `prog` and is launched from the console with the command

```
./prog 36 60
```

Then your program should produce the following output:

```
The gcd of 36 and 60 is 12.
```

2.3.3 Data validation and error reports

You should check the argument of your program and report an eventual error using one of the following messages (again assuming that you built an executable named `prog`):

- `program launched with too few arguments.`
  `Proper usage:  ./prog m n`
- `program launched with too many arguments.`
  `Proper usage:  prog m n`

Needless to say, the name of the executable should not be hard-coded in your C program.
Again, for this version, you don't need to verify that the arguments are valid, strictly positive integers.

2.4 Version 3: Combined version

2.4.1 Name of the source file

Your source file should be named `prog01v3.c`.

2.4.2 What the program should compute

Your program will take either one or two strictly positive integer arguments. If the program was launched with a single argument, then you should compute and output the list of all divisors of the argument. For example, if your executable is named `prog` and is launched from the console with the command

```
./prog 36
```

Then your program should produce the following output:

```
The list of divisors of 36 is:  1, 2, 3, 4, 6, 9, 12, 18, 36.
```

If the program was launched with two arguments, then your program should compute and output the gcd of these two numbers.For example, if your executable is named `prog` and is launched from the console with the command

```
./prog 36 60
```

Then your program should produce the following output:

```
The gcd of 36 and 60 is 12.
```

2.5 Data validation and error reports

You should check the argument(s) of your program and report an eventual error using one of the following messages (again assuming that you built an executable named `prog`):

- `program launched with no argument.`
  `Proper usage:  ./prog m [n]`
- `program launched with too many arguments.`
  `Proper usage:  ./prog m [n]`

Needless to say, the name of the executable should not be hard-coded in your C program.
Again, for this version, you don't need to verify that the arguments are valid, strictly positive integers.

2.6 Regarding output specifications

In this and future assignments, when I give specifications for the format of the output, you are expected to follow these specifications very precisely (whether it concerns input and output format, or the naming of functions and files), and you will be penalized if you don't.

2.7 Extra credit

2.7.1 Extra credit 1 (4 points)

This version of the program, named `prog01EC1.c`, applies to Version 1: Instead of reporting *all* divisors of the integer argument, only reports the prime divisors.

2.7.2 Extra credit 2 (3 points)

This version of the program, named `prog01EC2.c`, applies to Version 1, and simply adds one error message to the original list:

- `Program launched with no argument.`
  `Proper usage:  ./prog m`
- `Program launched with too many arguments.`
  `Proper usage:  ./prog m`
- `The program's argument is not a strictly positive integer.`

Note that we perform a "lazy" data validation: Stop as soon as an error is detected, and report. First the number of arguments, then the value of the argument. So there is no room for a composite error message: "Too many arguments, and by the way your first argument is not a strictly positive integer."

2.7.3 Extra credit 3 (2 points)

This version of the program, named `prog01EC3.c`, applies to Version 2, and simply adds two error messages to the original list:

- `Program launched with no argument.`
  `Proper usage:  ./prog m`
- `Program launched with too many arguments.`
  `Proper usage:  ./prog m`
- `The program's first argument is not a strictly positive integer.`
- `The program's second argument is not a strictly positive integer.`

Here again, this is a "lazy" data validation: Stop and report as soon as an error is detected: Num,ber of arguments, then first argument, then second argument. So, there is special error report in the case that both arguments are invalid.

3 Part II: a first simple `bash` script

3.1 What it is about

Here again, nothing fancy: We want you to write a `bash` script that builds an executable and launches it.

3.2 Location of the bash script files

Your script files should be placed in a folder named `Scripts` located at the root of your project folder. The scripts will be executed from the root of the project folder. Be careful that your scripts don't include any hard-coded absolute paths that can only work on your personal computer.

3.3 The script

Your script should be named `script01.sh` and take as argument a list of strictly positive integers.
Then your script should do the following:

- Build the C program of Version 1.
- Launch the divisor program for each of the integer arguments, to get the list of divisors of that number.

For example, if your script is launched with the command

```
./Scripts/script01.sh 36 10 37
```

then the output should be:

```
The list of divisors of 36 is:  1, 2, 3, 4, 6, 9, 12, 18, 36.
The list of divisors of 10 is:  1, 2, 5, 10.
The list of divisors of 37 is:  1, 37.
```

3.4 Extra credit 4: 4 points

This script, named `script01EC1.sh` should perform all the tasks of the regular script, but also build the program of Version 2 and launch it for each *pair* of arguments (ignoring order, so if you called `./prog 20 12`, don't call `./prog 12 20`, to compute the gcd of these two numbers.
For example, if your script is launched with the command

```
./Scripts/script01EC1.sh 36 10 37
```

then the output should be:

```
The list of divisors of 36 is:  1, 2, 3, 4, 6, 9, 12, 18, 36.
The list of divisors of 10 is:  1, 2, 5, 10.
The list of divisors of 37 is:  1, 37.
The gcd of 36 and 10 is 2.
The gcd of 36 and 37 is 1.
the gcd of 10 and 37 is 1.
```

3.5 Extra credit: 5 points

With this script, named `script01EC2.sh`, is an addition on the previous one: If a value is repeated in the list of arguments to your script, then use this value only once to launch the divisor program.
For example, if your script is launched with the command

```
./Scripts/script01EC2.sh 36 10 36 37 10
```

then the output should be:

```
The list of divisors of 36 is:  1, 2, 3, 4, 6, 9, 12, 18, 36.
The list of divisors of 10 is:  1, 2, 5, 10.
The list of divisors of 37 is:  1, 37.
The gcd of 36 and 10 is 2.
The gcd of 36 and 37 is 1.
the gcd of 10 and 37 is 1.
```

4 What to submit

4.1 Organization

The C programs, including extra credit versions, should go into a folder named `Source`. All the scripts should be in a folder named `Scripts`. Again, the scripts will be executed from the root folder of the project. Do **not** include any other file (the pdf of the assignment, executables, etc.).

The folder project should be named Prog01 and then compressed as a `zip` archive, **not** a `.rar` or other alternative archive format.

4.2 Grading for this assignment

For this assignment (and the next one as well), we are going to go for bark over bite, pointing at problems, telling you what is going to be penalized in future assignments. So, don't just pat yourself on the back if you get a good grade, while ignoring the comments and warnings we provide, and then act all surprised when points start coming off in future assignments.

4.2.1 Execution

We are going to test your code with different list of arguments. Your grade for this section will reflect to what extent you produce the desired output for all the test cases. So, make sure that you test your programs before you submit; don't simply replicate the examples of this handout.

4.2.2 Code quality

For this part of the grade we look at things such as:

- Proper indentation;
- Good comments;
- Good choice of identifiers (chances are that `a`, `b`, and `c` are not good identifiers, particularly if the first two are strings and the third one an integer;
- Consistent choices of identifiers. For example, if you have a variable `int beanCount` that does what it claims to do, then it would be a bad idea to have another variable `int num_of_boxes`.
- Good implementation decision: avoid super long functions (at your current skill level, if your function occupies more than one page of your screen, it's probably too long); select proper data types; create your own when appropriate.

Your grade for this section will reflect how much you actually implemented. A superbly commented and indented "Hello World!" program won't bring you many "code quality" points.

4.2.3 No syntax error

Code with syntax errors (compiler errors for the C program) will get a grade of 0 for execution and quality. Comment out the part with syntax errors and explain in the comments what you were trying to do.

4.3 Comments

There is no separate report for this assignment (coming soon), but you are expected to document properly your code with comments that explain what you are doing.

4.4 Grading

- Code quality: 35 pts
  - C programs: 25 pts
  - bash script: 10 pts
- Execution: 50 pts
  - C programs: 35 pts
  - bash script: 15 pts
- Folder organization: 15 pts