Software Engineering 265 Software Development Methods Fall 2019

Assignment 1 Due: Monday, October 14, 4:30 pm by submission via git (no late submissions accepted)

Programming environment

For this assignment, please ensure your work compiles and executes correctly on the Linux machines in ELW B238. You are welcome to use your own laptops and desktops for much of your programming; if you do this, give yourself a few days before the due date to iron out any bugs in the C program you have uploaded to the BSEng machines. (Bugs in this kind of programming tend to be platform specific, and something that works perfectly at home may end up crashing on a different hardware configuration.)

Individual work

This assignment is to be completed by each individual student (i.e., no group work). Naturally you will want to discuss aspects of the problem with fellow students, and such discussion is encouraged. However, sharing of code fragments is strictly forbidden without the express written permission of the course instructor (Murray). If you are still unsure regarding what is permitted or have other questions about what constitutes appropriate collaboration, please contact me as soon as possible. (Code-similarity analysis tools will be used to examine submitted programs.)

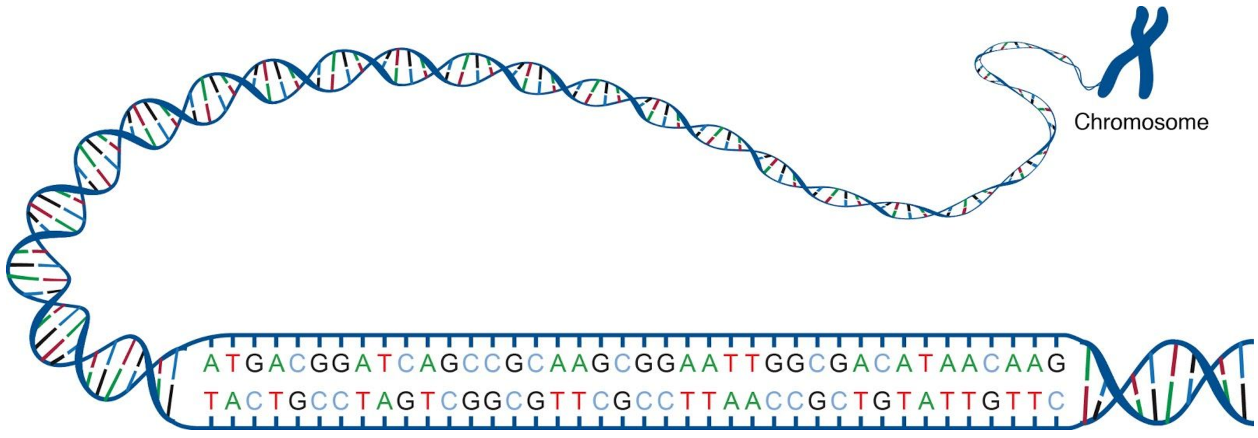
Objectives of this assignment

• Use the C programming language to write the first implementation of a file encoder (and decoder) named “RLE.c”
• Test your code against the provided test data, and other test data you create, to ensure it’s working as expected. The instructor will also be testing your program against test data that isn’t provided to you, so be sure to think creatively about how to “break” and strengthen your code!

This assignment: “RLE.c”

Bioinformatics is the management and use of biological information, particularly genetic information. It involves the application of computer sciences in order to solve problems in molecular biology. Biologists have become reliant on having access to shared data and analytical tools to do their research, but there are certain challenges, including the exponential growth of data. When performing experiments, scientists are often forced to rely on massive data warehouses and are looking for ways to transfer huge amounts of data as quickly as they can. There are numerous ways to improve this process, but one ubiquitous technique is to decrease the amount of data needing to be transferred using compression methods.

In Bioinformatics, one example of data is DNA sequences (e.g. sequences made from the letters A, C, G, and T, representing the nucleobases Adenine, Cytosine, Guanine and Thymine, the molecular building blocks of all DNA).


Chromosome
ATGACGGATCAGCCGCAAGCGGAATTGGCGACATAACAAG
TACTGCCTAGTCGGCGTTCGCCTTAACCGCTGTATTGTTC

Source: genome.gov

An example of such a sequence would be:
AACCCTAAAATTGGAA

In this assignment, we will use Run-Length Encoding as a lossless data compression technique to condense the repetitive parts of DNA sequences. RLE might not be a biologist's first choice for which compression technique to select and apply, but it will give us an entry point to discuss compression algorithms and their implementation.

In particular, we are interested in both the encoding of sequences (translating the original format into an encoded format), and the inverse decoding operation, which translates the encoded format back into the original sequences.

Encoding

To store the original example sequence "AACCCTAAAATTGGAA" in memory would require one byte per character, or 16 bytes in total

To start RLE, we start with the first character, which in this example is 'A'. Then we look at the second character, which is 'A'. Then we evaluate whether the first character is the same as the second character. In this case, it is. Since the last two characters were both the same, we're now going to look at the third character and see if it is also the same. The third character is 'C',

so we have found the end of the streak of ‘A’s. Since we have reached the end of the streak of ‘A’s, we now write out the character in the streak and number of times it occurs in the streak.
Output: **A2**
The third character ‘C’ in this example is the beginning of a streak of three characters. So we add this to the output:
Output: A2**C3**
The first character that appears after the streak of ‘C’s is a T. Only one T appears at that point so the out is now changed to:
Output: A2C3**T1**
We continue this process until we reach the end of the sequence, and can produce the final encoded output:

Final Encoded Output : A2C3T1A4T2G2A2
The length of the output is 14 bytes. This is 2 bytes less than the 16 bytes for the original sequence.

Decoding

To decode the encoded sequences, we need to iterate through the characters in the encoded string as “sub-sequences”. Each sub-sequence is a letter followed by a one-digit number. The first character is the original letter, and the next character is the number of consecutive instances of that letter in the original sequence. So, returning to the output from the encoding example, let’s run it through the decoding process.

The encoded string is: **A2C3T1A4T2G2A2.**
The *first* character is ‘A’ and the count is 2. So we build a new output string:
Decoded Output : **AA**

The *third* character is ‘C’ and the count is 3.
Decoded Output : AA**CCC**

The *fifth* character is ‘T’ and the count 1.
Decoded Output : AACCC**T**

From observing the positions of the letters in the encoded format, We can deduce that each odd-numbered character (starting from 1) in the string to be decoded is an encoded letter, and each even-numbered character is the count of the number of occurrences of that letter in the original sequence.

The seventh character is ‘A’ and the count 4.
Decoded Output : AACCCT**AAAA**

The ninth character is ‘T’ and the count 2.
Decoded Output : AACCCTAAAA**TT**

The eleventh character is ‘G’ and the count 2.
Decoded Output : AACCCTAAAATT**GG**

The thirteenth character is ‘A’ and the count 2.
Decoded Output : AACCCTAAAATTGG**AA**

Final Decoded Output : AACCCTAAAATTGGAA

Requirements

The requirements for this assignment are as follows:

1. Use arrays (and not dynamic memory allocation!) as an aggregate data type for file processing (encoding and decoding).
2. Create a `main` function that:
    a. Takes two command line arguments:
        i. A string indicating the name of the file to operate on;
        ii. A one-character flag indicating whether the program should operate in encoding or decoding mode. Use ‘e’ for encoding mode, or ‘d’ for decoding mode.
        iii. If the second argument is not provided, or is not either ‘e’ or ‘d’,
            1. print “Invalid Usage, expected: RLE {filename} [e | d]” to the console
            2. terminate the program with exit code 4.
    b. Opens the file indicated by the first argument.
        i. If there is no filename specified:
            1. print “Error: No input file specified!” to the console
            2. terminate the program with exit code 1.
        ii. If the filename specified doesn’t exist or can’t be read for any reason:
            1. print “Read error: file not found or cannot be read” to the console
            2. terminate the program with exit code 2.
    c. Reads one line from e file into a C string. The single line will be a maximum of 40 characters (we will process only the first line, there should be no new line character as input, or subsequent line processing). The encoded and decoded file formats should follow these rules:
        i. Each file must only contain upper case letters, digits and optional whitespace.
        ii. If the file contains any whitespace characters, they must be at the end of the file, and must be ignored.

        iii. If within the given file any non-whitespace character follows any whitespace character, or any non-whitespace character other than a digit or an upper-case letter, the file should be considered to be in an invalid format. Upon detecting this situation, the program should:
            1. print "Error: Invalid format" to the console
            2. terminate the program with exit code 3.
    d. Depending on the value of the second command-line argument, passes the string read from the input file to the **`encode`** or **`decode`** function as described below.

3. Create a second function called **`encode`** with the following characteristics:
    a. Take a C string as a function parameter.
    b. Encode the provided string using the algorithm described above - to simplify, the sequences in this assignment will be limited to the range of [2-9] in length
    c. If the string was encoded successfully:
        i. print the resulting encoded representation to the console
        ii. terminate the program with exit code 0, indicating success
    d. If the string could not be encoded for any reason (e.g. badly formatted):
        i. print "Error: String could not be encoded" to the console
        ii. terminate the program with exit code 5

4. Create a third function called **`decode`** with the following characteristics:
    a. Take a C string as a function parameter
    b. Decode the provided string using the algorithm described above
    c. If the string was decoded successfully:
        i. print the resulting decoded representation to the console
        ii. terminate the program with exit code 0, indicating success
    d. If the string could not be decoded for any reason (e.g. badly formatted):
        i. print "Error: String could not be decoded" to the console
        ii. terminate the program with exit code 5

Provided Materials

We have created a git repository for you containing a test suite and these instructions.

Test Suite

We will supply a suite of test cases that you can use to evaluate your own program, and get a feel for the types of inputs you can expect.

The test suite is structured as a set of directories, one per test case. Each test case directory will have the following characteristics:

- The directory is named according to the type of test, e.g. **`encode_badInput_letterAfterWhitespace`**

- The first word in the directory name will be **`encode`** or **`decode`**, indicating which mode your program should be in when running this test case.
- Any additional text in the directory name (e.g. **`_badInput_letterAfterWhitespace`** in this example) is informational and can be ignored.
- The directory always contains a single file named **`input.txt`** that your program is expected to read as its input
- If your program is expected to run successfully given this input, the directory will contain a file named **`output.txt`** that your program's output is expected to precisely match
- If your program is expected to terminate with a non-zero (unsuccessful) error code when given this input, the directory will contain one of the following:
    - a file named ***n*`.err.txt`**, where n is the expected exit code, containing text that your program's output is expected to precisely match, **or;**
    - a file named ***n*`.err`**, where n is the expected exit code. In this case your output doesn't need to match any specific error message.

Process

To get started, the assignment repository has already been set up in your person GitLab space, listed as /assignment1.git. We set this up in our local machine space with **`git clone`**:

```
git clone
https://gitlab.csc.uvic.ca:courses/2019091/SENG265/assignments/<NETLIN
K_ID>/assignment1.git
```

(all one line, with <NETLINK_ID> replaced with your personal Netlink identifier)

Note that by cloning from your personal repository, this automatically sets up your remote on GitLab so when you **`git push`** later, your changes will automatically be visible to both you and the teaching team.

Use a text editor to write your program.

Once you're able to compile your program, test it by running it with the **`input.txt`** files in the provided test suite, and any other files you can think of. Be creative; try to predict what kinds of inputs will break your code! Try random files lying around on your computer or the internet to see what happens!

If it seems like your changes are having no effect, don't forget to recompile your program after changing it!

When you've made changes to the file that you want to keep, **`git commit`** your changes to your local copy of the repository. Git will save the history of past commits. When you want to upload your changes to your repository, use **`git push`**.

Deliverables

- Write a C program in a file named RLE.c (case sensitive), placed in the root of your repository, that implements the above requirements.
- The program must compile successfully using **gcc**.
- The assignment will be marked based on the last version of your program pushed before the deadline.

Evaluation

Compilation

(10% of the mark, if your program compiles at all)

Correctness

(70% of the mark, based on the proportion of provided and secret test cases that your program successfully passes)
The teaching team will run your code using the provided test suite to verify whether your program meets the requirements:

- When run with correct parameters and given input in the expected format, it produces the expected output
- When run with incorrect parameters or badly formatted input data, it produces the expected error messages and the expected exit codes

We will also evaluate your program using ***an additional test suite that is not provided to you***, to ensure that your solution isn't over-fit to the provided test suite.

Style

(20% of the mark, based on the subjective assessment of the teaching team)

The teaching team will read your code to judge whether it is clean, original, elegant and fit to purpose.

A few examples of bad programming style:

- **inconsistent formatting**, indentation, etc. -- pick a style and use it consistently
- **misleading or non-descriptive names** for variables and functions
- **insufficient whitespace** impairing readability (no IOCCC entries please)
- long sections of **repetitive code** (sometimes it's better to repeat short sections than introduce an abstraction!)
- **"dead" code**, i.e. present in the file but never called
- **complex code without explanatory comments** (don't go overboard; simple code often doesn't need comments)