CSCI 1730 Project 1

Compressing and Decompressing DNA Strings Using Bitwise Operators

Learning Outcomes

- Understand binary representation of integers in memory and use bitwise operators to solve problems.
- Design and implement a program that uses bitwise operators on unsigned integers.
- Trace, design, and implement software solutions to non-trivial problems using the C programming language.
- Understand lossless text compression and decompression.
- Using a Makefile to compile and run a program with more than one header file and more than one source code file.
- Compile, link, and run programs via the command line.
- Design and implement programs that use command line arguments.
- Design and implement header files that use preprocessor directives.
- Design and implement programs that use basic programming and flow of control structures.

Problem

A DNA string d is a string consisting only of characters from the set $\{A, T, C, G\}$ whose length is greater than 0. Let $|d|$ denote the length of DNA string d. The purpose of this assignment is to compress and decompress DNA strings inputted via command line arguments using bitwise operators. The use of bitwise operators is REQUIRED for this assignment. The following binary encoding will be used for this assignment.

Character in d	binary encoding
A	00
T	01
C	10
G	11

Your task is to figure out how compression and decompression work based on the examples and what was discussed in a previous lecture class. The flag "-c" denotes compression, the flag "-d" denotes decompression, and your program will need to perform both operations on valid inputs based on the examples shown in the Examples section. This type of compression is called lossless text compression because no characters are lost when we decompress a compressed DNA string. You should work through all of the examples in this assignment using paper and pencil and practice bitwise operators before you write any source code. Also, for each compression example you should calculate the number of bytes saved (or lost) when a DNA string is compressed.

Project Requirements

Your program must adhere to all requirements stated below, and you must follow other instructions and requirements in this document.

1. Your program must use at least three different bitwise operators for compressing and decompressing. Your program should NOT compress or decompress using only arithmetic operators.
2. Your program must run to completion in less than 5 seconds on odin. If your program runs longer than 5 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, which is provided for you on eLC. This means that you must download, read, understand, and use the Makefile. You CANNOT modify the provided Makefile, which means you'll need to split your program into multiple files (`compress.c`, `compress.h`, `decompress.c`, `decompress.h`, and `proj1.c`), and details about these files are below.
   (a) `compress.c` must contain an implementation of at least one function to compress a DNA string using bitwise operators. Any helper functions needed for compression should be implemented here too.
   (b) `compress.h` must contain all of your prototypes for the functions in `compress.c`, and these prototypes must be commented. Appropriate header guards and include statements should be placed in this file.
   (c) `decompress.c` must contain an implementation of at least one function to decompress a compressed DNA string using bitwise operators. Any helper functions needed for decompression should be implemented here too.
   (d) `decompress.h` must contain all of your prototypes for the functions in `decompress.c`, and these prototypes must be commented. Appropriate header guards and include statements should be placed in this file.
   (e) `proj1.c` must contain the main function that processes command line arguments and passes these arguments to an appropriate function implemented in `compress.c` or `decompress.c`. The main function cannot do more than process command line arguments and pass them to other functions implemented in other files. The main function must be the only function in this file. 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.
4. All files needed for this assignment should be placed in a directory called `proj1`.
5. You cannot use a break, continue, or goto statement for this assignment. If your source code contains one or more break, continue, or goto statment(s), then you will receive a grade of zero for this assignment.
6. You cannot use any global variables or static variables for this assignment. If your source code contains one or more global variable(s) or static variable(s), then you will receive a grade of zero for this assignment.
7. The only include directives that you are authorized to use in your source code for this assignment are listed below.

   ```
   #include <stdio.h>
   #include <string.h>
   #include <stdlib.h>
   #include <stdbool.h>
   #include "compress.h"
   #include "decompress.h"
   ```

   You CANNOT use any other include directives anywhere in your source code. Using an unauthorized include directive anywhere in your source code will result in a grade of zero on this assignment.
8. All written and verbal instructions stated by the teaching staff (lecture instructor, lab instructor(s), etc.) must be followed for this assignment. Failure to follow instructions may result in losing points.

Examples

Your C source code should be in multiple files as aforementioned, and it should be compiled into an executable called `proj1.out` using the provided Makefile. Your C program must look exactly like the examples below when run on the command line on odin, and it must compile and run correctly with any valid sequence of inputs. You may assume that the command line arguments will have valid inputs (we won't input invalid things to your program). Your program's I/O must match the following examples. Each example shows a single run of a correctly working program on odin.

```
./proj1.out -c ATTCGG
6 22 240
```

```
./proj1.out -c CAT
3 132
```

```
./proj1.out -c AGTCCCAGATTTCCC
15 54 163 21 168
```

```
./proj1.out -c GTTAACCGGTTAGGCCTCCTC
21 212 43 212 250 105 128

./proj1.out -c CCCCTAGAGAGAGAGAGCCGAGTTCAAAGTCAAAACCCATTCTCTCTCCTCG
52 170 76 204 204 235 53 128 216 2 161 102 102 155

./proj1.out -d 6 22 240
ATTCGG

./proj1.out -d 3 132
CAT

./proj1.out -d 15 54 163 21 168
AGTCCCAGATTTCCC

./proj1.out -d 21 212 43 212 250 105 128
GTTAACCGGTTAGGCCTCCTC

./proj1.out -d 52 170 76 204 204 235 53 128 216 2 161 102 102 155
CCCCTAGAGAGAGAGAGCCGAGTTCAAAGTCAAAACCCATTCTCTCTCCTCG
```

1 C Program

1.1 Coding Style Requirements

1. All functions must be commented. Comments must include a brief description of what the function does, its input(s), its output, and any assumptions associated with calling it. If your function has a prototype and an implementation separated into different parts of your source code, then you only need to put the comments for that function above its prototype (there is no need to comment both the prototype and implementation; commenting the prototype is sufficient).
2. All structs, unions, and enums must be commented.
3. All identifiers must be named well to denote their functionality. Badly named identifiers are not allowed. For example, identifiers like a, aaa, b, bbb, bbbb are bad names for identifiers.
4. Every line of source code must be indented properly and consistently.

1.2 README.txt File Requirements

Make sure to include a `README.txt` file (use a .txt file extension) that includes the following information presented in a reasonably formatted way:

- Your First and Last Name (as they appear on eLC) and your 810/811#
- Instructions on how to compile and run your program. Since we are using a Makefile for this assignment, then these instructions should pretty simple based on the provided Makefile.

2 Submission

Submit your files before the due date and due time stated on eLC. Submit your files on eLC under the Assignment named Project 1. Submit only the following files.

1. All source files required for this assignment: compress.c, compress.h, decompress.c, decompress.h, and proj1.c
2. A README.txt file filled out correctly

Do not submit any compiled code. Also, do not submit any zipped files or directories. Do not submit a Makefile. We will use our own, unmodified copy of the provided Makefile to compile your program. We only need the files mentioned above with the file extensions aforementioned.

3 Grading: 30 points

If your program does not compile on odin using the provided Makefile and using the required compiler for this course `(gcc version 11.2.0)`, then you'll receive a grade of zero for this assignment. Otherwise, your program will be graded using the criteria below.

Program runs correctly with various test cases on odin	30 points
README.txt file missing or incorrect in some way	-3 points
Not submitting to the correct Assignment on eLC	-3 points
Late penalty for submitting 0 hours to 24 hours late	-6 points
One or more compiler warnings	-6 points
Not adhering to one or more coding style requirements	-6 points
Using one or more unauthorized include directive(s)	-30 points
Source code contains a break, continue, or goto statement	-30 points
Not submitting this assignment before its late period expires	-30 points
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 odin 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.