Homework 7

Due Sunday, October 30 by 10:00pm

Handout 16
CSCI 334: Fall 2022

Turn-In Instructions

For each question in this assignment, create a project directory. For example, the source for question 1 should be in a folder called "`q1`". The source for question 2 should be in a folder called "`q2`".

For C questions, supply a single `Makefile` for each directory. The `Makefile` should contain one rule per C program. The naming convention for targets should be the name of the source file without the `.c` extension. For example, `q1a.c` should compile to `q1a`. You must also provide an `all` target that builds all targets and a `clean` target that removes all of the files generated by the build. For full credit, be sure that your code compiles without emitting warnings even when using the `-Wall` flag.

For F# questions, you should be able to `cd` into the project directory and then run the program by typing the command "`dotnet run`". Be sure to split the program into two pieces: a "`Program.fs`" file that contains the `main` method and associated program-startup helpers (if needed), and another "`Library.fs`" file that contains the function(s) of interest in the question. All library code should be in a module named "`CS334`". For full credit, your program should build without error.
Turn in your work using the Gitlab repository assigned to you. The name of the Github repository will have the form `https://evolene.cs.williams.edu/cs334-f22/<YOUR_USERNAME>/lab07.git`. For example, if your CS username is `22abc1`, the repository would be `https://evolene.cs.williams.edu/cs334-f22/22abc1/lab07.git`.

Honor Code

This is a partner lab. You may work with another classmate if you wish, and you may co-develop solutions. Remember: although you can work on code together, you must each independently write up and submit your solution. No code copying is allowed. **Be sure to tell me who your partner is** by committing a `collaborators.txt` file to your repository (5 points). Be sure to include this file even if you work by yourself.

This assignment is due on Sunday, October 30 by 10:00pm.

Sanity Check: Students sometimes submit incomplete assignments, accidentally forgetting to run `git add` for all of their files. Fortunately, there is an easy way to make sure that this does not happen to you. Before you are done, `git clone` your repository to a new folder and then try building/running everything. It only takes a couple minutes and can spare you from headaches later on.

Reading

1. **(Required)** Read "C: A Language Built Around a Memory Model" from the course packet.
2. **(Required)** Read "Passing Pointers by Value" from the course packet.

Problems

Q1. (40 points) ..... Allocation

Write code to perform the following. Refer to `man 3 rand` for documentation on sampling a random `int`.

(a) Generate a random `int` and copy it to every element of an array of length 10. The array should have automatic storage duration. Print the array.

(b) Generate a random `int` and copy it to every element of an array of length 10. The array should have allocated storage duration. Print the array. Be sure to deallocate when done.

(c) Generate a random C string (a `char *`) and copy it to every element of an array of length 10. Make sure that every element is a copy of the string, not a copy of the pointer. The array should have automatic storage duration. Print the array.

(d) Generate a random C string (a `char *`) and copy it to every element of an array of length 10. Make sure that every element is a copy of the string, not a copy of the pointer. The array should have allocated storage duration. Print the array. Be sure to deallocate when done.

As usual, be sure to anticipate and handle runtime errors in your program. Since this is C, a number of errors can occur with respect to memory handling. See the readings for details.

"Boxes and arrows" drawings should accompany each program; name these drawings `q1a.pdf`, etc. Please draw these diagrams neatly. If we can't read them, we can't give you credit for doing them! The diagram should reflect the state at the point in your program that you think best demonstrates how memory is being used. Be sure to indicate the appropriate line number in your diagram.

Supply your answers as a C source code programs called `q1a.c`, `q1b.c`, `q1c.c`, and `q1d.c`. Be sure to include your `Makefile`.

Q2. (20 points) ..... Counting characters

Write a program that counts characters. After starting the program, the user should be prompted to type (or paste) input into the program. When the `Enter` key is pressed, the program should read in and store the user string, then prompt for more input.

When the user presses `Ctrl-D`, the program should print the character count of the line, followed by the line itself. The very last number printed by the program is the total character count.

The program can detect when the user has pressed `Ctrl-D` by checking for the `EOF` character. The program should use the `getchar` function (see `man 3 getchar` for documentation) to get characters from the keyboard buffer.

Here is a sample session

```
$ ./q2
enter input> The quick brown fox jumps over the lazy dog.[Enter key pressed]
enter input> This is a test of the emergency broadcast system.[Enter key pressed]
enter input> Neat.[Enter key pressed]
enter input> [CTRL-D pressed]

44: The quick brown fox jumps over the lazy dog.
49: This is a test of the emergency broadcast system.
5: Neat.
98
$
```

Supply your answer as a C source code program called `q2.c`. Be sure to include your `Makefile`. As usual, try to anticipate and handle any runtime errors that may arise.

Some hints:

- The keyboard buffer in most terminals can store at most 1024 characters at a time. So you may assume that no single entered string is longer than 1024 characters.
- The natural data structure for storing a sequence of unknown length is a linked list, and if you want to write a linked list in C, please do. But you can alternatively assume that the user will never enter more than 10,000 lines total into the program.
- Making a 1024 $\times$ 10,000 byte array (especially an automatic array of that size) is not the right approach, mostly because it is egregiously wasteful of space. Most strings will probably not be 1024 characters long, and most inputs will not be 10,000 lines long. Use allocated memory, and do your best to be thrify (i.e., no wasted space).
- Lastly: you can either count characters as you read out the strings, or count as you read in the strings. Both approaches are acceptable. The second one is more interesting, but requires storing something like an array of tuples. C does not have tuples (but it does have structs!).

Q3. (35 points) ..... A Calculator

Write an evaluator in F# for a simple arithmetic calculator. The calculator should support the following arithmetic operations:

- addition,
- subtraction,
- multiplication, and
- integer division.

It should be possible to apply the above operations to integer data. For example, the calculator should be able to compute $1 + 5$. It should also be possible to apply the above operations to other arithmetic expressions. For example, the caluclator should be able to compute $1 + 5 * 3$.

Start by drawing the abstract syntax tree (AST) for some arbitrary expressions, like $1 + 5 * 3$ or $6/7 - 4 + 1$. You do not need to include these trees in your homework submission. Do them in order to reason about the next step. Be sure that your ASTs reflect the correct parse (i.e., pay attention to precedence and associativity for arithmetic).

(a) Write an F# algebraic data type (i.e., a `type`) that represents the four operations above, as well as numeric data (i.e., 5 cases). Your type should start with

```
type Expr =
| ...
```

(b) Provide an implementation for the following function, which takes an AST for an arithmetic expression and recursively computes the result.

```
let rec eval(e: Expr): int = ...
```

Hint: since `Expr` is an algebraic data type, `expr` is probably best approached as a pattern matching problem.

(c) In your `main` function, be sure to provide a small number of tests that demonstrate that your calculator works. You do not need to convert (i.e., "parse") string data (like `"1 + 5 * 3"`) into the appropriate `Expr` value. Instead, generate the test ASTs by hand. Your `eval` method should correctly compute the results of all of the arithmetic expressions mentioned in this question. When encoding those expressions, be sure to correctly encode precedence and associativity in your AST.

The project directory for this question should be called "`q3`". You should be able to run your program on the command line by typing, for example, "`dotnet run`".

Q4. ($\frac{1}{10}^{\text{th}}$ bonus point) ..... Optional: Feedback

I always appreciate hearing back about how easy or difficult an assignment is.

For $\frac{1}{10}^{\text{th}}$ of a bonus to your final grade, please fill out the following Google Form.