Final Examination

Important Notes – PLEASE READ FIRST

- Each submission must be a student's individual work. Collaborations between students are **not** allowed. We use MOSS and other software to detect plagiarism.
- If you adapt code or ideas from a book or website, acknowledge the book (title, edition/year, author(s), page numbers) or website (URL) in your programs or answers. You should not copy the code directly and then add some minor modifications, but to borrow ideas only.
- Complete the header in each submitted file with your student information.
- Do not modify the C function definitions in the submitted files. Do not add `printf` statements to your C code. *Note*: These mistakes will mess up our automatic grading programs and produce incorrect outputs.
- Do not use any C library functions except those for I/O (e.g., `scanf, printf, getchar, putchar`) or memory allocation (e.g., `malloc, calloc, realloc`).
- Assume that all inputs to the C programs are valid. No error checking is required on inputs.
- Do not use global variables in any program.
- Use Bourne shells for the UNIX shell scripting problems. That is, use only the commands, control structures and utilities provided in the lecture notes and tutorial notes.
- Submit each answer/program after you finish it. Submit your work periodically and frequently before the deadline to avoid last-minute technical glitches.
- **Submission instruction:**
  - **websubmit: https://webapp.eecs.yorku.ca/submit/**
  - **submit command: submit 2031N exam filename**

The exam consists of 6 questions.

Problem 1 – Converting octal string to decimal integer (10%)

Specification

Write a C program to input an **octal** number in the form of a line of characters in the form **`[+/-]0[digits]`**. Store the input characters in an array. Convert the octal number to a decimal integer.

Note: Users may or may not input the + or – sign. Prefix 0 is required in all inputs.

Implementation

- Use the given template `octal.c` and fill in your code. Submit only file `octal.c`.
- You are given an array of characters of size `MAX_SIZE` where `MAX_SIZE` = 100. The array is named `my_strg`.
- Use `getchar` and a loop to read an octal number in the form of a line of characters as specified above, and store the input characters into array `my_strg`. The loop terminates when a new line character `'\n'` is entered. The new line character `'\n'` is NOT part of the line (i.e., discard the new line character `'\n'`).
- Convert the octal number stored in array `my_strg` to a decimal integer which is stored in variable `my_int`.
- You may use either pointers or array indexing.
- See file `allOutput.txt` for sample inputs and outputs.

Problem 2 – Converting hexadecimal string to decimal integer (10%)

Repeat Problem 1, except that the input now is a line of characters containing a **hexadecimal** number in the form **`[+/-][0x/0X][digits]`**. The hexadecimal digits 'A' to 'F' can be in upper case or lower case.

Note: Users may or may not input the + or – sign. . Prefix `0x` or `0X` is required all inputs.

Use the given template `hexa.c` and fill in your code. Submit only file `hexa.c`.

You may use either pointers or array indexing.

See file `allOutput.txt` for sample inputs and outputs.

Problem 3 – Converting octal/decimal/hexadecimal string to decimal integer (25%)

Specification

Write a C program that contains two functions, one to input a string in the form of a decimal, octal or hexadecimal number and the other to convert the string into an actual integer number.

The input string has the following format: **[+/-][0/0x/0X][digits]** , where

- The plus or minus sign is optional, i.e., the user may or may not enter one.
- **0** indicates that the digits that follow represent an octal number.
- **0x** or **0X** indicates that the digits that follow represent a hexadecimal number.
- The prefix **0/0x/0X** is optional. If it is not present in the input string, then the string stores a decimal number.
- **digits** = 0-9 for decimal numbers.
- **digits** = 0-7 for octal numbers.
- **digits** = 0-9, a-f, and A-F for hexadecimal numbers. The upper case and lower case of an alphabetic digit has the same value, i.e., a = A, b = B, etc.

Implementation

- Use the given template `converts2i.c` and fill in your code. Submit only file `converts2i.c`.
- The first function to be implemented is `myStrInput()`. See file `converts2i.c` for its specification. Use `getchar` and a loop to read a line of characters as specified above, and store the input characters into the array. The loop terminates when a new line character `'\n'` is entered. The new line character `'\n'` is NOT part of the line (i.e., discard the new line character `'\n'`).
- The second function to be implemented is `str2int()`. See file `converts2i.c` for its specification. The function converts the string passed to the function to an integer number and returns the integer number to the caller.
- You may use either pointers or array indexing.
- See file `allOutput.txt` for sample inputs and outputs.

Problem 4 – Command line arguments and string concatenation (30%)

Write a C program to input several strings as command-line arguments. Concatenate these strings into a long string in the same input order. Assume that the final output string has size less than `MAX_SIZE = 5000`.

Use the given template `concat.c` and fill in your code. Submit only file `concat.c`.

See file `allOutput.txt` for sample inputs and outputs.

Do not use any C string library functions such as `strlen, strcpy, strcat`, etc. You have to copy each character of the input strings to the output string.

You have to use pointers in this program. Do not add square brackets to the file, even in comments. The grading script will use the command `grep` to check for `'['` and `']'`, and give zero if the square brackets appear in the program.

Problem 5 – Bourne Shell Scripting (15%)

Specification

Given a directory, write a UNIX script named **`mychex`** that processes each ordinary file inside that directory as follows:

- Check if each file is executable or not. If not then ask the user if it should be changed to executable. If the user responds with 'y' or 'Y' then change the file permission to be executable by the user/owner. Leave the file as is for any other response.
- After all files have been processed, display the detailed information of all the executable files in the given directory using `"ls –l"` for the user's verification.
- If the given directory is empty then there is nothing to be done and nothing will be displayed.

Note: Assume that there are no sub-directories inside the given directory (so that we do not have to worry about whether `mychex` is recursive or not).

How is a directory specified?

- If the user does not enter any command-line argument then the given directory is the current working directory.
- If the user enters one command-line argument then the given directory is the command-line argument. The directory name can be an absolute or relative path name. If the directory name is

not valid (i.e., the directory does not exist), display an error message on the standard output and quit the program.

- If the user enters more than one command-line argument, consider only the first argument (even if it is an invalid directory name, which will cause an error message to be displayed and the program to terminate) and ignore the rest.

Sample Inputs/Outputs

Use the provided script `mktmp` to create a directory `Temp` with several files inside it for your testing. See file `allOutput.txt` for some examples of running the script `mychex` and for the message to be displayed in each scenario.

Note 1: When we grade this program, we will use the "`grep`" command on your outputs to search for the names of the executable files (which are displayed by "`ls –l`") and check the file attributes for the executable status.

Note 2: Use only the commands, control structures and utilities provided in the lecture notes and tutorial notes.

Problem 6 – Bourne Shell Scripting (10%)

Specification

Given a set of file names as command line arguments, write a UNIX shell script called **`counting`** to

- display the detailed information of the empty files using "`ls –l`"
- count the number of empty files in the command line arguments and display the following message, where **`n`** is the number of empty files.

  `Found n empty file(s)`

Notes:

- The file names (command line arguments) can be relative or absolute path names, and may include files that do not exist (e.g., `alien`, `ghost` and `shadow` in the sample output). If a file does not exist, ignore it. Display and count only existing empty files.
- The number of command line arguments can be more than 9.
- If the user does not enter any argument after the command **`counting`**, display the following usage message and exit.

  `Usage: counting file1 file2 ...`

Sample Inputs/Outputs

Use the provided script `mktmp2` to create a directory `Temp2` with several files and subdirectories inside it for your testing. Assuming that `Temp2` is the current working directory, see file `allOutput.txt` for some examples of running the script **`counting`**.

Note 1: When we grade this program, we will use the "`grep`" command on your outputs to search for the names of the empty files (which are displayed by "`ls –l`"), and for the output phrase "`Found n empty file(s)`". So make sure that the spelling is correct and there is only one white space between any two words in the phrase.

Note 2: Use only the commands, control structures and utilities provided in the lecture notes and tutorial notes.