Programming Exercises

OPTIONAL AND EXTRA CREDIT

The objective of this assignment is to help you hone your programming skills in C and C++. This programming exercise is *Optional & Extra Credit*. There are 15 programming exercises, and each accounts for 0.1 points of extra-credit towards your cumulative course grade. The assignments are due in batches as outlined in the due dates on Canvas.

What should I prioritize: the regular HW1 and HW2 or the programming exercises?
You should prioritize the homeworks. For e.g., HW1 and HW2 are each worth 5 points towards your cumulative course grade --- i.e., individually HW1 and HW2 are worth 50 times more than each programming exercise.

Due Dates:

- C01 through C06 (i.e. 6 programming exercises) are due on Wednesday, January 29th, @ 8:00 pm MT
- C07 through C12 (i.e. 6 programming exercises) are due on Wednesday, February 5th, @ 8:00 pm MT
- C++13, C++14, and C++15 (i.e. 3 programming exercises) are due on Wednesday, February 12th, @ 8:00 pm M

Generative AI Use and Consequences
Use of AI tools such as ChatGPT, Claude, Github Co-Pilot, and/or their ilk to write or "improve" your code or written work at *any* stage is prohibited; this includes the ideation phase. It is your responsibility to ensure that you don't have the GitHub Co-Pilot extension installed in your IDE; assignment solutions generated by Co-Pilot aren't written by you. Turning in code or an essay written by generative AI tools will be treated as turning in work created by someone else, namely an act of plagiarism and/or cheating.

Ultimately, you will get out of the class what you put in. Simply copying and pasting code from generative AI tools is neither ethical nor does it contribute to your learning experience. There are multiple reasons why these generative AI tools are detrimental to your learning experience:

1. They rob you of the ability to think and learn the concepts for yourself. Solving problems is an essential step to gaining a solid understanding of the material.
2. You will struggle with the in-classroom quizzes and exams where you will not have access to these tools.
3. While we acknowledge that these tools are likely to become an important part of a software engineer's workflow in the future, you are much more likely to use these tools in an effective manner if you already have expertise in the relevant technical topics. Developing such expertise requires putting in the effort to learn these topics without the assistance of these tools.
4. These tools are prone to generating imperfect or even incorrect solutions, so trusting them blindly can lead to bad consequences.

Some helpful Infospaces videos for this assignment:

(1) Hello World in C: https://infospaces.cs.colostate.edu/watch.php?id=268

(2) Makefile for C Programs: https://infospaces.cs.colostate.edu/watch.php?id=269

(3) Run Script for C Programs: https://infospaces.cs.colostate.edu/watch.php?id=273

(4) Remote SSH Using VS Code: https://infospaces.cs.colostate.edu/watch.php?id=272

Auto-grading in seconds
Programming assignments are being autograded and the scores will be reflected in Canvas less than 30 seconds after you have submitted. You have unlimited attempts (till the submission deadline), and your highest score will be retained. Use of these autograders is predicated on you following the outputs exactly as specified. If you are having trouble printing outputs in the prescribed format, please get in touch with the TAs. Don't procrastinate and start early.

1 Programming Exercises

Naming Convention: Exercises that are in C are prefixed with a C, while those in C++ are prefixed with CC++.

1.1 C01 - Hello World

Goals:

- Familiarization with Makefile
- Familiarization with C syntax

Instructions

1. Write a C program
    - File name: main.c
    - Executable name: main
    - No includes allowed except for stdio.h
2. Include a Makefile with the following targets:
    - build
    - run
    - clean
3. Additional requirements for this exercise:
    - Print "Hello, World!"

Example

To run:

```
./main
```

Result:

```
Hello, World!
```

1.2 C02 - Sum

Goals:

Use printf and scanf

Instructions:

1. Write a C program
   - File name: main.c
   - Executable name: main
   - No includes allowed except for stdio.h
2. Include a Makefile with the following targets:
   - build
   - run
   - clean
3. Additional requirements for this exercise:
   - Read 2 integers from stdin
   - Print the sum of the 2 integers

Example

To run:

```
./main < infile
```

Contents of *infile*:

```
2 3
```

Result:

```
5
```

1.3 C03 - Even Odd

Goals

- Use `if-else` and `%` operator. User input from arguments. Use `atoi` function.

Instructions

1. Write a C program
   - File name: `main.c`
   - Executable name: `main`
   - No includes allowed except for `stdio.h` and `stdlib.h`
2. Include a Makefile with the following targets:
   - `build`
   - `clean`
3. Additional requirements for this exercise:
   - Read an integer from the command line arguments
   - Check if even or odd
   - Use if-else and % operator
   - Print "even" if integer is even
   - Print "odd" if integer is odd

Example

To run:

```
./main 5
```

Result:

```
odd
```

1.4 C04 - Largest

Goals

- Use comparison operators (`>`, `<`, `==`)

Instructions

1. Write a C program
   - File name: `main.c`
   - Executable name: `main`
   - No includes allowed except for `stdio.h`
2. Include a Makefile with the following targets:
   - `build`
   - `run`
   - `clean`
3. Additional requirements for this exercise:
   - Read two integers from stdin
   - Print the largest integer to stdout
   - Use comparison operators (`>`, `<`, `==`)
   - Print "equal" if integers are equal

Example

To run:

```
./main < infile
```

Contents of *infile*:

```
4 5
```

Result:

```
5
```

1.5 C05 - Multiplication

Goals

- Use for loop

Instructions

1. Write a C program
   - File name: main.c
   - Executable name: main
   - No includes allowed except for stdio.h
2. Include a Makefile with the following targets:
   - build
   - run
   - clean
3. Additional requirements for this exercise:
   - Read an integer from stdin
   - Print its multiplication table (from x1 to x10)
   - Print each number on a separate line
   - Use a for loop

Example

To run:

```
./main < infile
```

Contents of *infile*:

7

Result:

```
7

14

21

28

35

42

49

56

63

70
```

1.6 C06 - Factorial

Goals

- Use while loop, using unsigned long long

Instructions

1. Write a C program
   - File name: main.c
   - Executable name: main
   - No includes allowed except for stdio.h and stdlib.h

2. Include a Makefile with the following targets:
   - `build`
   - `clean`
3. Additional requirements for this exercise:
   - Read an integer N<=20 from command line arguments
   - Compute and print the factorial of the integer
   - Use a while loop
   - Consider using something larger than int to store the factorial number

Example

To run:

```
./main 14
```

Result:

```
87178291200
```

1.7 C07 - Reverse

Goals

- Use arrays

Instructions

1. Write a C program
    - File name: `main.c`
    - Executable name: `main`
    - No includes allowed except for `stdio.h`
2. Include a Makefile with the following targets:
    - `build`
    - `run`
    - `clean`
3. Additional requirements for this exercise:
    - Read an integer N from stdin
    - Read N integers from stdin and store them into an array
    - Print the array in reverse order
    - Each number should be printed on a separate line

Example

To run:

```
./main < infile
```

Contents of infile:

```
5 1 2 3 4 5
```

Result:

```
5

4

3

2

1
```

1.8 C08 - Vowels

Goals

- Use char arrays

Instructions

1. Write a C program
   - File name: `main.c`
   - Executable name: `main`
   - No includes allowed except for `stdio.h`
2. Include a Makefile with the following targets:
   - `build`
   - `run`
   - `clean`
3. Additional requirements for this exercise:
   - Read a string from stdin using scanf, the string will be 1000 characters or less
   - Count the number of vowels in the string before the first whitespace character
   - Remember to count both upper and lower case vowels
   - Print the number of vowels

Example

To run:

```
./main < infile
```

Contents of infile:

```
Lorem_ipsum_dolor_sit_amet,_consectetur_adipiscing_elit,_sed_do_eiusmod_tempor_incididunt_ut_labore_et_dolore_magna_aliqua
```

Result:

```
45
```

1.9 C09 - Primes

Goals

- Use functions and loops

Instructions

1. Write a C program
   - File name: `main.c`
   - Executable name: `main`
   - No includes allowed except for `stdio.h` and `<stdbool.h>`
2. Include a Makefile with the following targets:
   - `build`
   - `run`
   - `clean`
3. Additional requirements for this exercise:
   - Read an integer from stdin
   - Print all prime numbers up to that number
   - Print each prime number on a separate line
   - Create a helper function to check if a number is prime: `bool is_prime(int n)`
   - Remember to include `<stdbool.h>`

Example

To run:

```
./main < infile
```

Contents of infile:

```
43
```

Result:

```
2
```

```
3

5

7

11

13

17

19

23

29

31

37

41

43
```

1.10 C10 - Sort

Goals

- Use multiple files

Instructions

1. Write a C program
   - File names: `main.c` and `sort.h`
   - Executable name: `main`
   - No includes allowed except for `stdio.h` and `sort.h`
2. Include a Makefile with the following targets:
   - `build`

- `run`
- `clean`

3. Additional requirements for this exercise:
    - `sort.h`
        - Declare function `void sort(int arr[], int size)`
        - The function should sort the array in place
        - Implement any sorting algorithm you like
        - You cannot use any library
    - `main.c`
        - Include `sort.h`
        - Read an integer N from stdin
        - Read N ints from stdin and store them into an array
        - Call `sort` function
        - Print each number in the array on a separate line

Example

To run:

```
./main < infile
```

Contents of infile:

```
7 1 2 -3 4 5 -6 7
```

Result:

```
-6

-3

1

2

4

5

7
```

1.11 C11 - Pointers

Goals

- Use pointers

Instructions

1. Write a C program
   - File names: main.c and pointers.h
   - Executable name: main
   - No includes allowed except for stdio.h, stdlib.h, and pointers.h,
2. Include a Makefile with the following targets:
   - build
   - run
   - clean
3. Additional requirements for this exercise:
   - pointers.h
     - Create function float* new_float()
     - The function should use malloc to allocate memory
     - The function should return a pointer to a new float
     - The float should be initialized to 0.0
     - Create function void swap(float *a, float *b)
     - The function should swap the values of a and b
   - main.c
     - Include pointers.h
     - Create variable f1 and f2 of type float* and and allocate space for them by calling new_float()
     - Read 2 floats from stdin and store them into the f1 and f2
     - Call swap with the 2 floats
     - Print the 2 floats
     - You must not use any other variables other than f1 and f2 in main.c

Example

To run:

```
./main < infile
```

Contents of infile:

```
4.3 6.7
```

Result:

```
6.700000 4.300000
```

1.12 C12 - Matrix

Goals

- File operations

Instructions

1. Write a C program
    - File name: `main.c`
    - Executable name: `main`
    - No includes allowed except for `stdio.h`
2. Include a Makefile with the following targets:
    - `build`
    - `clean`
3. Additional requirements for this exercise:
    - Read `input_path` and `output_path` from command line arguments
    - Read a matrix of integers from the file `input_path`
    - Write the transposed matrix to the file `output_path`
    - Both input and output matrices should be formatted as follows:
        - The first line contains 2 integers equal to the number of rows and columns respectively
        - On the following lines:
            - Values in a row are separated by a space
            - Rows are separated by a newline

Example

To run:

```
./main infile outfile
```

Contents of infile:

```
2 3

1 2 3

4 5 6
```

Contents of outfile (after run);

```
3 2

1 4

2 5

3 6
```

1.13 C++01 - Calculator

Goals

- Familarize with `cin` and `cout`. Use `switch`.

Instructions

1. Write a C++ program
   - File name: `main.cpp`
   - Executable name: `main`
   - No includes allowed except for `iostream` and `string`
2. Include a Makefile with the following targets:
   - `build`
   - `run`
   - `clean`
3. Additional requirements for this exercise:
   - Create an infinite while loop
     - Read the user input from stdin
     - If the user input is `exit`, break the loop
     - Otherwise the input is formatted as follows: `number operator number`. For example: `5.1 + 3`, `10 - 4.7`, `7 * 2.5`, `8 / 2`,
     - Use `double` for the numbers
     - The possible operators are `+`, `-`, `*`, `/`
     - Use a `switch` statement to determine the operation to perform
     - Print the result of the operation followed by a newline
     - Repeat the loop
   - use `cin` and `cout` for input and output

Example

```
4 + 3

7

6 - 2

4

3 / 4

0.75

0.75 * 5

3.75

exit
```

1.14 C++02 - Stats

Goals

- Use references. File operations.

Instructions

1. Write a C++ program
   - File names: `main.cpp` and `stats.h`.
   - Executable name: `main`
   - No includes allowed except for `cmath`, `fstream`, `vector`, and `stats.h`.
2. Include a Makefile with the following targets:
   - `build`
   - `clean`
3. Additional requirements for this exercise:
   - `stats.h`
     - Create a function `void stats(const std::vector<double>& data, double& min, double& max, double& avg, double& std_dev)` that receives a vector of doubles and calculates the minimum, maximum, average, and standard deviation values.
     - Make use of the references to return the computed values.
     - Any loop in the function must be a for loop by reference. For example: `for(const double& value : data)`.
   - `main.cpp`
     - Your program will receive a list of files as arguments.
     - The list will contain at least 2 files.
     - The last file is the output file.
     - For each file except the last one:
       - Read the data into a vector.
       - The data in the file is formatted with one double per line. You should read until the end of the file.
       - Call the `stats` function.
       - Write the stats to the output file in the following format: `<min>,<max>,<avg>,<std_dev>\n`.
     - At the end of the program, the output file should be a csv file with 4 columns and as many rows as input files.
     - No header in the output file.

Example

To run:

```
./main infile1 infile2 outfile
```

Contents of infile1:

```
1 2 3 4 5
```

Contents of infile2:

```
-1 -2 -3 -4.0 -5
```

Contents of outfile2:

```
1,5,3,1.41421

-5,-1,-3,1.41421
```

1.15 C++03 - Bank

Goals

- Use classes and objects.

Instructions

1. Write a C++ program
   - File names: `main.cpp`, `bank_account.h`, and `bank_account.cpp`
   - Executable name: `main`
   - No includes allowed except for `string`, `iostream` and `bank_account.h`
2. Include a Makefile with the following targets:
   - `build`
   - `run`
   - `clean`
3. Additional requirements for this exercise:
   - `bank_account.h`
     - Declare class `BankAccount`
     - No public attributes
     - Constructor and 3 public methods:
       - `BankAccount()`: constructor, initializes balance to 0.0
       - `double balance()`: returns the balance
       - `void deposit(double amount)`: adds amount to balance. If amount is negative, does nothing.
       - `void withdraw(double amount)`: removes amount from balance. If amount is negative or greater than the balance, does nothing.

- `bank_account.cpp`
    - Implement the methods declared in `bank_account.h`
- `main.cpp`
    - Infinite while loop
        - Read string from input
        - If string is `exit`, break loop
        - if string is `balance`, print balance followed by a newline
        - if string is `deposit`, read double from input and deposit (e.g., `deposit 100.0` or `deposit 30`)
        - if string is `withdraw`, read double from input and withdraw (e.g., `withdraw 50.0` or `withdraw -7000`)

Example

```
balance

0

deposit 10

withdraw 20

balance

10

withdraw -10

balance

10

withdraw 10

balance

0

exit
```

2 What to Submit

Use the CS370 *Canvas* to submit a single .zip file that contains:

- All .c and .h files related to the assignment (please document your code),
- a Makefile that performs both a *make clean* as well as a *make all,*
- a README.txt file containing a description of each file and any information you feel the grader needs to grade your program.

Filename Convention: You should keep the .c and .h filenames as they are in the skeleton code; any additional files can have the names you want. The archive file should be named as <FirstName>-<LastName>-PE**X**.zip . E.g. if you are Cameron Doe and submitting for programming exercise 1, then the zip file should be named Cameron-Doe-PE1.zip.

3 Grading

The assignments much compile and function correctly on machines in the CSB-120 Lab. Assignments that work on your laptop on your particular flavor of Linux, but not on the Lab machines are considered unacceptable.

Each programming exercise accounts for 0.1 points of extra credit towards your cumulative course grade. Together, the 15 programming exercises in this assignment accounts for 1.5 points of extra credit towards your cumulative course grade.

You are required to **work alone** on this assignment.

4 Late Policy

All assignments are due at 8:00 PM on the due date. There is a late penalty of 10% per-day for up to a maximum of 2 days.