

# Programming assignment 3 (120 points)

[Submit Assignment](#)

---

**Due** Mar 22 by 11:59pm      **Points** 120      **Submitting** a file upload      **File Types** tar  
**Available** until May 12 at 11:59pm

---

## Data representations and operations in C

Prof. Yipeng Huang

Rutgers University

2021 Spring

---

## 0. Introduction and setup

### Learning goals

You will learn about how data is represented in computers, and you will learn about the properties and limitations of data types. You will also experiment with how computers carry out mathematical operations such as addition, subtraction, and multiplication on the data representations. Such knowledge lends insight into how the electronic hardware of computers performs calculations, and allows you to gain intuition about the relative costs of different kinds of operations.

### Sources

Get started early! You can post questions to the class Piazza (please avoid sending emails to the TAs and instructor).

- You can discuss the assignment with your classmates, but you cannot copy code from your classmates.
- You can use the internet to research the mechanics of the C programming language, data types, and algorithms, but you cannot use websites that offer ready-made answers specific to this assignment and this class. Consider citing any sources that you use in your research; such a citation can be as simple as providing a link to a webpage in a code comment.

- We will be checking the assignments for identical and/or plagiarized code using automated tools.
- <https://www.cs.rutgers.edu/academics/undergraduate/academic-integrity-policy>  
(<https://www.cs.rutgers.edu/academics/undergraduate/academic-integrity-policy>)
- <https://www.cs.rutgers.edu/academics/undergraduate/academic-integrity-policy/programming-assignments>  
(<https://www.cs.rutgers.edu/academics/undergraduate/academic-integrity-policy/programming-assignments>)
- The instruction staff takes the Rutgers University Academic Integrity Policy seriously. Please review it at: <https://nbprovost.rutgers.edu/academic-integrity-students>  
(<https://nbprovost.rutgers.edu/academic-integrity-students>). We will report all violations.

## Setup

Access the programming assignment files for this assignment by pulling new files from the class Github repository, 2021\_0s\_211:

```
git pull
```

If you do not have this directory already, clone the repository:

```
git clone https://github.com/yipenghuang0302/2021_0s_211.git
```

The files for this assignment are in the directory 2021\_0s\_211/pa3/.

---

## 1. toHex: Converting signed integers to binary and hexadecimal representation (easy) (24 points)



Computers represent numbers using a base-2 binary number system to represent integers. And in order to represent both positive and negative numbers, they use a two's complement number system that overflows (wraps around) nicely and has simple rules for carrying out addition and subtraction. Because binary numbers are unwieldy for human beings to read and write, we use hexadecimal notation to make the representation more compact.

In this part of the assignment, you will take positive and negative integers that can be represented using signed short (16-bit) integer data types and print out their hexadecimal representation.

## Input format

Your program should take a single command line argument specifying the path to an input file. Test cases for your program are in the tests/ directory. In each test case there is a single integer, written in base-10, that is within the range of the signed short int data type (-32,768 to +32,767).

## Output format

Expected outputs from your program for each test case are in the answers/ directory. You should print a four-character hexadecimal number corresponding to the binary representation of the input number. In your hexadecimal representation, the digits that would be represented using letters (A-F) should be printed in uppercase.

For example, the base-10 number -4319 has a binary two's complement representation of 1110\_1111\_0010\_0001, which you should print out in hexadecimal notation as EF21.

---

## How to compile, run, and test your code

Instructions from Programming Assignment 1 and 2 carry over.

The autograder.py scripts for this assignment only work with Python version 3 on the ilab machines. This means that the correct way to invoke the autograder script is through either of these two commands:

```
./autograder.py
```

or

```
python3 autograder.py
```

---

## binSub: Finding the difference of two signed integers via operations on binary numbers (medium) (24 points)

In computer processors all arithmetic operations are carried out as basic logical operations on binary numbers. In this part of the assignment you will perform subtraction on signed char (8-bit) integers, where positive and negative numbers are encoded in the two's complement number system. Recall that for subtraction operations:

```
minuend - subtrahend = difference
```

To perform subtraction, you will negate the binary representation of the subtrahend, and then add that negated number to the minuend to find the difference. You may also want to review how to add two binary numbers using a carry bit. Your output should also be a signed char (8-bit) integer, so any overflow should wrap around and stay within the limits of the signed char data type.

It is good practice (and certainly more interesting) for you to perform this calculation on the binary numbers directly in this assignment. That is to say, you should not convert the binary numbers into integer data types and perform subtraction using the '-' operator.

## Input format


Your program should take a single command line argument specifying the path to an input file. Test cases for your program are in the tests/ directory. In each test case, the first line records the signed 8-bit two's complement minuend (the number to be subtract from). The second line records the signed 8-bit two's complement subtrahend (the number to subtract).

## Output format

You should print the difference of the two inputs as a sequence of eight binary digits, overflowing if needed to stay within the limits of the signed char data type. Expected outputs from your program for each test case are in the answers/ directory.

---

# 3. binToFloat: Finding the value of a floating point number binary representation (medium) (24 points)

In this part of the assignment, you will decode the 32-bit binary representation of an IEEE 754 single-precision float point number. You may wish to review how floating point numbers are encoded using the  lecture slides, the textbook, and supplementary reading material, among other resources. You should decode the sign, exponent, and mantissa portions of the floating point number to reconstruct the number's value. You may not use memory operations and pointer referencing and dereferencing to cast the bitstring into a float. This restriction has also been noted in the binToFloat\_provided.c code.

## Input format

Your program should take a single command line argument specifying the path to an input file. Test cases for your program are in the tests/ directory. Each test case consists of a string of 32 1's and 0's

giving the binary representation of a floating point number. The sign bit is given in the leftmost position in the string.

## Output format

You should print the value of the floating point number. You can use the "%f" (for floats) and "%e" (for floats in scientific notation) printf format specifiers for printing. The autograder will be able to recognize both formats. Expected outputs from your program for each test case are in the answers/ directory.


---

## 4. doubleToBin: Find the binary representation of a double-precision floating point number (hard) (24 points)

In this fourth part of the assignment, you will encode a given real number as a IEEE 754 double-precision floating point number, and print out its 64-bit binary bitstream. This type of engineering task would be important and typical in developing testbenches that help validate whether computer hardware designs for field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs) are functioning correctly.

You may want to use trial division to find the exponent value such that mantissa of the floating point number is in its normalized range of:

$$1.0 \leq \text{mantissa} < 2.0$$

If you are able to find the correct exponent and print the correct numbers for the exp field, you will receive partial credit for passing a portion of the given test cases. To complete this part of the assignment, you need to also properly encode the mantissa in the frac field of the floating point number. For maximum points, you will also have to pass test cases for encoding denormalized numbers, and special cases in the floating point format such as +0.0 and -0.0. You may not use memory operations  pointer referencing and dereferencing to cast the float into a bitstream. This restriction has also been noted in the doubleToBin\_provided.c code. The provided code also has suggested assertion statements that help you validate that your encoding adheres to the IEEE 754 specification.

## Input format

Your program should take a single command line argument specifying the path to an input file. Test cases for your program are in the tests/ directory. Each test case consists of a positive or negative real number that can be encoded as an IEEE 754 double-precision floating point number.

## Output format

You should print the 64-bit binary representation of the double-precision floating point number that encodes the given floating point number. Expected outputs from your program for each test case are in the `answers/` directory. You may add '\_' underscore characters to your printout to aid yourself in reading the bitstream. The autograder will automatically ignore these characters.

---

## 5. floatMul: Finding the product of two single-precision floating numbers via operations in binary (hard) (24 points)

In this fifth and final part of the assignment, you will bring together ideas you have used throughout this assignment to implement floating point multiplication. Floating point multiplication is a foundational arithmetic operation that underpins science simulations, machine learning, and nearly every computer science discipline. The ability to perform floating point multiplication in hardware is measured in floating point operations per second, or FLOPS. The world leading supercomputers are pushing 500 petaFLOPS ( $10^{15}$  FLOPS), while even in consumer hardware one can purchase computers capable of 10 teraFLOPS ( $10^{12}$  FLOPS).

It is good practice (and certainly more interesting) for you to perform this calculation on the binary numbers directly in this assignment. You may want to review the class lecture slides, the posted supplementary reading, and the textbook to understand how floating point multiplication works—treating the exp and the frac fields separately. An alternative approach would be to decode the input bitstreams into a floating point representation, then use the '\*' multiplication operator in the C language to perform the calculation, and re-encode the floating point number to a bitstream. The only approach that is not permitted is to use memory operations and pointer referencing and dereferencing to cast between floats and bitstreams. This restriction has also been noted in the `floatMul_provided.c` code.

## Input format

Your program should take a single command line argument specifying the path to an input file. Test cases for your program are in the `tests/` directory. In each test case, the first line records a multiplier as a 32-bit floating point number binary representation. The second line records a multiplicand as a 32-bit floating point number binary representation. The sign bit is given in the leftmost positions in the strings.

## Output format

You should print the 32-bit binary representation of a single-precision floating point number that encodes the product of the inputs.. Expected outputs from your program for each test case are in the answers/ directory. You may add '\_' underscore characters to your printout to aid yourself in reading the bitstream. The autograder will automatically ignore these characters.

---

## How to submit

From the pa3/ directory, you can run this command to check on the outputs of our autograder script.

```
./assignment_autograder
```

or

```
python3 assignment_autograder.py
```

When you are ready to submit, from the 2021\_0s\_211/ directory where you see the pa2/ directory, run this command:

```
tar cvf pa3.tar pa3/
```

Upload the file pa3.tar here on Canvas.

By submitting your assignment, you are agreeing to the Rutgers Honor Pledge: “On my honor, I have neither received nor given any unauthorized assistance on this assignment.”

We will not be accepting late assignments. The Canvas submission site will close to enforce the deadline, so be sure to submit early.



If anything is not clear, reach out to your classmates and the instructors on the class Piazza!