CSE 220: Systems Fundamentals I

Stony Brook University
Homework Assignment #2
Fall 2023

Due: Monday, Sept 25, 2023 by 11:59 PM Eastern Time sharp!

No extension will be given–Prof. Malik

Learning Outcomes

After completion of this homework assignment you should be able to:

- Use basic git commands to manage code revisions.
- Use bitwise operators to perform bit-level computations.
- Use bit-manipulation operators in C to implement bit-level computations.

Preliminaries

Complete the basic git tutorial provided on the git home page.

Download the Template Repository

You will receive a link from the Computer-Fundamentals-220-FALL2023 organization on the Github Classroom. On your Linux VM, click this link to access the template repository. You'll need to accept the invitation by clicking the button marked “Accept this Assignment”. After GitHub sets up the repository, you can go to https://github.com/Computer-Fundamental-220-FALL2023/ to find your repository.

To clone the repository onto your VM , look for a green button in the repository browser marked **<> Code**. Copy the URL for **SSH** access. Type `git clone` with a space to the right of the word “clone”. Then paste the URL you just copied. Your command line should resemble the following (all on one line):

```
git clone git@github.com:Computer-Fundamental-220-FALL2023/
                                          cse-220-homework-2-USERNAME.git
```

where USERNAME is your GitHub username.

Part 1: Bit Manipulation

File: `BitManipulation.c`

To access this part of the assignment, cd to the directory containing your work (cd cse-220-homework-2-USERNAME) and then cd BitManipulation.

When compressing and uncompromising data, often we need to perform operations on less than a single byte. Therefore it is important to become familiar with some basic operations for working with data one bit at a time. In this part of the assignment, you will implement the following basic bit operation functions:

1. **int GetBit(int num, int pos)** :- should give the bit value in num at position pos.
    a. Example GetBit(12, 2) should return 1.
2. **int SetBit(int num, int pos)**:- should set the bit in num at position pos and should return the new integer value if the pattern is changed.
    a. Explanation :- if K-th bit is 0, then set it to 1 and if it is 1 then leave it unchanged.
3. **int ClearBit(int num, int pos):-** should clear the bit in num at position pos and should return the new integer value.
    a. Explanation:- if K-th bit is 1, then clear it to 0 and if it is 0 then leave it unchanged.
4. **int Reversbit(int num):-** should reverse the bit pattern of num and return the new integer value.
5. **int CountBit(int num):-** should give the number of ones in num.
6. **int PowerofTwo(unsigned int num)** should return 1 if num is power of 2 else 0.

Note:

1. You will not use any library to implement these operations.
2. The command line you take the option and the required number of parameters. For example:
    a. ./a.out 4 6 << should reverse the bit pattern of integer 6>>
    b. ./a.out 2 7 3 <<should set the bit 3 in the bit pattern for integer 7>>

Remember *→ LSB (Least Significant Bit) has position 0 and the MSB (Most Significant Bit) is at position 31 for this assignment. Don't worry about the endiness of your machine (Good Read).*

Running and Testing Your Code:

1. You will provide enough test cases of your implementation as well to show it covers most of the cases.

Grading Notes

1. The TAs will test your code on Linux VM using their own tests. Marks depend on how many test cases are successfully passed by your code. 100% pass ensures full marks for the Testing part.
2. The submitted code will be checked for plagiarism using tools. Any plagiarism will lead to zero marks.

Part 2: Construct IEEE 754 Single-precision Number

File: `float.c`

To access this part of the assignment, cd to the directory containing your work (`cd cse-220-homework-3-USERNAME`) and then `cd Float`.

Implement the following function in C:

```
float construct_float_sf(char sign_bit, char exponent, unsigned int
fraction);
```

Provided for you are the following arguments:

- `sign_bit`: The sign bit zero-extended to 8-bit binary representation. For example, `0000000`**`0`** represents `0` which is positive and `0000000`**`1`** represents `1` which is negative.
- `exponent`: The exponent field in 8-bit, excess-127 binary representation. For example, `10001000` which is 136 or an exponent of 9 (136 - 127 = 9).
- `fraction`: The fraction field, or matissa, zero-extended to 32-bit representation. For example, `000000000`**`01100000000000000000000`**. Remember, only the first 23 bits matter!

Within this function, there are two statements already written for you that you **must** not change:

- The declaration and initialization of the variable `f`. Here, `f` is initialized to `0` and has a binary value of `'00000000000000000000000000000000'`.
- The `return` statement that returns the binary representation of `f` as a `float` (We will cover how that works later in this class)

Start writing your code after the "Start coding here" comment. Your job is to use bitwise operations to construct the float number `'f'` using the arguments provided. After you write your code, the `return` statement will do the rest of the work for you.

Examples:

Function Arguments	**Value Of `'f'`**
'`0x00`', '`0x81`', '`0x300000`'	'`0x40B00000`' or decimal '5.5'
'`0x01`', '`0x81`', '`0x300000`'	'`0xC0B00000`' or decimal '-5.5'
'`0x00`', '`0x7F`', '`0x200000`'	'`0x3FA00000`' or decimal '1.25'
'`0x01`', '`0x7F`', '`0x200000`'	'`0xBFA00000`' or decimal '-1.25'
'`0x00`', '`0x76`', '`0x299B6F`'	'`0x3B299B6F`' or decimal '0.002588'
'`0x01`', '`0x76`', '`0x299B6F`'	'`0xBB299B6F`' or decimal '-0.002588'
'`0x00`', '`0x89`', '`0xABCDEF`'	'`0x44ABCDEF`' or decimal '1374.44'
'`0x01`', '`0x89`', '`0xABCDEF`'	'`0xC4ABCDEF`' or decimal '-1374.44'
'`0x00`', '`0x90`', '`0x7973C0`'	'`0x487973C0`' or decimal '255439'
'`0x01`', '`0x90`', '`0x7973C0`'	'`0xC87973C0`' or decimal '-255439'

Note:

1. You will not use any library to implement these operations.
2. Test the correctness of your function by printing the output of the function using a `printf()` statement. For example, `printf("f = %g\n", construct_float_sf(0x00, 0x76, 0x299B6F))` and see whether this prints 0.002588 or not.

***Remember** → LSB (Least Significant Bit) has position 0 and the MSB (Most Significant Bit) is at position 31 for this assignment. Don't worry about the endiness of your machine (Good Read).*

Running and Testing Your Code:

1. You will provide enough test cases of your implementation as well that shows that it takes care of most of the case

Grading Notes

1. The TAs will test your code on Linux VM using their own tests. Marks depend on how many test cases are successfully passed by your code. 100% pass ensures full marks for the Testing part.
2. The submitted code will be checked for plagiarism using tools. Any plagiarism will lead to zero marks.

Part 3: Convert Integer Representations

File: `integers.c`

To access this part of the assignment, cd to the directory containing your work (`cd cse-220-homework-3-USERNAME`) and then `cd Integers`.

Implement the following function in C:

```
void repr_convert(char source_repr, char target_repr,
                  unsigned int repr)
```

This function takes an unsigned integer (`repr`) that provides the 32-bit representation of an integer and converts it from one integer representation to another. The function prints the number in the target representation as an eight-digit hexadecimal value. Include a single newline at the end of the output. Include leading zeros in the output to pad out the eight digits. Any digits which are given as letters must be printed in lowercase.

- `source_repr`: the character '2' or 'S' , which indicates the integer representation the function is converting from. '2' indicates that the input is a 32-bit two's complement representation. 'S' indicates that the input is a 32-bit sign/magnitude representation.

- `target_repr`: the character '2' or 'S', which indicates the integer representation the function is converting into. '2' and 'S' have similar interpretations for the `source_repr` argument, except that this character indicates the target representation.

- `repr`: an unsigned, 32-bit value that represents an integer in the provided source representation.

If the number in the provided source representation cannot be represented in the target representation, print the string `"undefined\n"` and return.

If `source_repr` or `target_repr` is not one of '2' or 'S', print the string `"error\n"` and return.

Recall that some integer representations have two representations of zero. Always output "positive" zero when converting into one of these representations.

When implementing this part, you may use functionality found only in the following C header files: `<stdio.h>`, `<stdlib.h>` and `<stdbool.h>`.

Examples

Function Arguments	Output
`'S', '2', 0x80000001`	`ffffffff`
`'S', '2', 0x80000000`	`00000000`
`'2', '2', 0x59f2ca50`	`59f2ca50`
`'F', '2', 0x00394812`	`error`
`'2', 'S', 0x80000000`	`undefined`

Running and Testing Your Code

- First read the contents of the provided makefile in the `Integers` directory. Note the various gcc compiler flags that have been provided. Take a moment and read about what these flags do.
- To build your C code with GCC, execute the command `make` (or `make all`).

Grading Notes

- During grading, only your integers.c file will be copied into the grading framework's directory for processing. Make sure all of your code is self-contained in that file.
- The TAs will test your code on Linux VM using their own tests. Marks depend on how many test cases are successfully passed by your code. 100% pass ensures full marks for the Testing part.
- The submitted code will be checked for plagiarism using tools. Any plagiarism will lead to zero marks.
- Provide enough test cases to show that your implementation is robust.

Submitting Your Work

To submit your work, do a final `git commit` to "save" your work locally, followed by a `git push` to send your work to the server for eventual grading.

Extensions, resubmissions and regrades will not be granted because a student did not use git properly to submit the assignment.

Some Advice about Testing and Grading

- Consider the following questions while writing your own test cases:
    - What normal classes (categories) of input are not tested by the current test cases?
    - What are special (“edge”) cases that are not covered by the current test cases? (e.g., erroneous function arguments, inputs that straddle the boundary between two cases in an if-statement)
    - What should happen when an argument has an invalid value?
    - Did I cover every possible path through some complicated logic?

Academic Honesty Policy

Academic honesty is taken very seriously in this course. By submitting your work for grading you indicate your understanding of, and agreement with, the following Academic Honesty Statement:

1. I understand that representing another person's work as my own is academically dishonest.
2. I understand that copying, even with modifications, a solution from another source (such as the web or another person) as a part of my answer constitutes plagiarism.
3. I understand that sharing parts of my homework solutions (text write-up, schematics, code, electronic or hard-copy) is academic dishonesty and helps others plagiarize my work.
4. I understand that protecting my work from possible plagiarism is my responsibility. I understand the importance of saving my work such that it is visible only to me.
5. I understand that passing information that is relevant to a homework/exam to others in the course (either lecture or even in the future!) for their private use constitutes academic dishonesty. I will only discuss material that I am willing to openly post on the discussion board.
6. I understand that academic dishonesty is treated very seriously in this course. I understand that the instructor will report any incident of academic dishonesty to the University's Academic Judiciary.
7. I understand that the penalty for academic dishonesty might not be immediately administered. For instance, cheating on a homework assignment may be discovered and penalized after the grades for that homework have been recorded.
8. I understand that buying or paying another entity for any code, partial or in its entirety, and submitting it as my own work is considered academic dishonesty.
9. I understand that there are no extenuating circumstances for academic dishonesty.