CS61c Spring 2013 Discussion 1 – Number Rep, C Intro

1 Unsigned Integers

By now we should all be somewhat comfortable with non-decimal bases. As a reminder, if we have an n-digit unsigned numeral $d_{n-1}d_{n-2}\dots d_0$ in radix (or base) r, then the value of that numeral is $\sum_{i=0}^{n-1} r^i d_i$, which is just fancy notation to say that instead of a 10's or 100's place we have an r's or r^2's place. For binary, decimal, and hex we just let r be 2, 10, and 16, respectively.

Recall also that we often have cause to write down unreasonably large numbers, and our preferred tool for doing that is the IEC prefixing system: Ki = 2^{10}, Mi = 2^{20}, Gi = 2^{30}, Ti = 2^{40}, Pi = 2^{50}, Ei = 2^{60}, Zi = 2^{70}, Yi = 2^{80}.

1.1 Some More Practice

1. Convert the following numbers from their initial radix into the other two common radices: 0b10010011, 0xD3AD, 63, 0b00100100, 0xB33F, 0, 39, 0x7EC4, 437

2. Write the following numbers using IEC prefixes: 2^{16}, 2^{34}, 2^{27}, 2^{61}, 2^{43}, 2^{47}, 2^{36}, 2^{58}.

3. Write the following numbers as powers of 2: 2 Ki, 256 Pi, 512 Ki, 64 Gi, 16 Mi, 128 Ei

2 Signed Integers

Unsigned binary numbers work to store natural numbers, but many calculations use negative numbers as well. To deal with this a number of different schemes have been used to represent signed numbers.

2.1 Sign and Magnitude and One's complement

Both of these schemes are relatively simple conceptually, but have been replaced by cleverer representations. Why?

- Most significant bit tells you the sign: 1 if negative, 0 if positive.
- Positive values can be treated just like unsigned integers.
- To invert the sign of a sign and magnitude number flip the MSB.
- To invert the sign of a one's complement number flip all the bits.

2.2 Biased Notation

- Like an unsigned int, but offset by $-(2^{n-1}-1)$, where n is the number of bits in the numeral. Aside: Technically we could choose any bias we please, but the choice presented here is extraordinarily common.
- Formally, if we have an n-bit biased notation number with bits $d_{n-1}d_{n-2}\dots d_0$, then the value of the numeral is $-(2^{n-1}-1)+\sum_{i=0}^{n-1} 2^i d_i$.
- Just one zero, but it's not at 0b0.
- Addition is a little weird, but not overwhelmingly so.

2.3 Two's complement

- Two's complement is the standard solution for representing signed integers.
  - Most significant bit has a negative value, all others have positive.
  - Otherwise exactly the same as unsigned integers.
- A neat trick for flipping the sign of a two's complement number: flip all the bits and add 1.
- Addition is exactly the same as with an unsigned number.
- Only one 0, and it's located at 0b0.

2.4 Exercises

For the following questions assume a 8 bit integer. Answer each question for the case of a sign and magnitude number, a one's complement number, a biased notation number, and a two's complement number.

1. What is the largest integer? The largest integer + 1?

2. How do you represent the numbers 0, 1, and -1?

3. How do you represent 17, -17?

4. What is the largest integer that can be represented by *any* encoding scheme that only uses 8 bits?

3 C Introduction

C is syntactically very similar to Java, but there are a few key differences of which to be wary:

- C is function oriented, not object oriented, so no objects for you.
- C does not automatically handle memory for you.
  - In the case of stack memory (things allocated in the "usual" way), a datum is garbage immediately after the function in which it was defined returns.
  - In the case of heap memory (things allocated with `malloc` and friends), data is freed only when the programmer explicitly frees it.
  - In any case, allocated memory always holds garbage until it is initialized.
- C uses pointers explicitly. `*p` tells us to use the value that `p` points to, rather than the value of `p`, and `&x` gives the address of `x` rather than the value of `x`.

There are other differences of which you should be aware, but this should be enough for you to get your feet wet.

3.1 At Least There Are Comments.

Write the following functions so that they perform according to the provided comment.

1.
```
/*The first function you write in any language.
 *Prints the string "Hello World\n" to standard output.*/
void hello_world() {
```

2.
```
/*Divides and takes the floor of a value exterior to this function by 2^POW.
 *Does not use the division function.*/
void div(int *y, unsigned int pow) {
```

3.
```
/*For each bit position i in [0, sizeof(int)*8) calls hello_world i times
 *iff the ith bit of the value X points to is set.*/
void HI_HI_HI_HI(int *x) {
```

4.

```c
/*Computes and returns the nth fibonacci number, using an iterative approach.*/
int fib_iter(unsigned int n) {
```

3.2 Uncommented Code? Yuck!

The following functions work correctly (note, this does not mean intelligently), but have no comments. Document the code to prevent it from causing further confusion.

1.

```c
/*
 *                                                        */
int foo(int *arr, size_t n) {
        return n ? arr[0] + foo(arr + 1, n - 1) : 0;
}
```

2.

```c
/*
 *                                                        */
int bar(int *arr, size_t n) {
        int sum = 0, i;

        for (i = n; i > 0; i--) {
                sum += !arr[i - 1];
        }

        return ~sum + 1;
}
```

3.

```c
/*
 *                                                        */
void baz(int x, int y) {
        x = x ^ y;
        y = x ^ y;
        x = x ^ y;
}
```