601.220 Intermediate Programming

Spring 2023, Day 15 (February 24th)

Today's agenda

- Exercise 14 review
- Number representation, type conversion/casting
- Exercise 15

Reminders/Announcements

- HW3 due **this evening**
- HW4 released later this evening, due Friday March 3rd no late subs
- Reminder: register your midterm project team by **11 pm on Sunday Feb 26th**
    - See Piazza post 233 for link to Google form

Exercise 14 review

'0' == 48
'1' == 49

Converting from string of 0 and 1 digits to a binary integer:

```
int str_to_int(char msg[], int len) {
  int result = 0;

  for (int i = 0; i < len; i++) {
    int index = len - i - 1;
    char c = msg[index];
    if (c == '1') {
      result |= (1 << i);
    }
  }

  return result;
}
```

Exercise 14 review

Converting from binary integer to string of 0 and 1 digits:

```
void int_to_str(int num_encrypted, char msg_encrypted[],
                int len) {
  for (int i = 0; i < len; i++) {
    int bit_pos = (len - i - 1);
    char bit =
      (num_encrypted & (1 << bit_pos)) == 0 ? '0' : '1';
    msg_encrypted[i] = bit;
  }
  msg_encrypted[len] = 0; // NUL terminator
}
```

"mask"

Exercise 14 review

Performing the encryption:

```
~~for (int i = 1; i < n; i++) {~~
  ~~num_encrypted ^= (num_encrypted << 1);~~
~~}~~
```

this code is incorrect

Day 15 recap questions

1. What is two's complement representation?
2. How does representation of integers and floating-point values differ in C?
3. What is type narrowing?
4. What is type promotion?
5. What is type casting?
6. What is the output of the code segment below?

1. What is two's complement representation?

Two's complement is used as the representation of signed integers on all modern computer architectures.

Idea: most significant bit makes a *negative* contribution to the value of the integer.

Consider the bit string 10000101:

- As an 8 bit unsigned value: $128 + 4 + 1 = 133$
- As an 8 bit signed two's complement value: $-128 + 4 + 1 = -123$

unsigned $\left\{ \begin{array}{l} 1011 \\ 2^3\,2^2\,2^1\,2^0 \\ 8+0+2+1 = 11 \end{array} \right.$

$\boxed{1}011$

$\boxed{-2^3}\,2^2\,2^1\,2^0$

$\boxed{-8}+0+2+1 = -5$

Big advantage of two's complement representation: addition and subtraction work the same way for both unsigned and signed values.

Negating a two's complement value

$$a - b \Longrightarrow a + -b$$

$$\begin{array}{r} 1111 \\ -\,0110 \\ \hline 1001 \end{array}$$

To ~~invert~~ negate a two's complement value, invert all of the bits and add 1.

Why?

A bit string where every bit is 1 has the value −1.

a is an integer, ~a is the "complement" of a (all bits inverted).

For any a, a + ~a = -1 (e.g., 10010110 + 01101001 = 11111111)

Rearranging: -a = ~a + 1

$$\begin{aligned} \sim a &= -1 - a \\ \sim a + 1 &= \cancel{-1} - a \cancel{+1} \\ &= -a \end{aligned}$$

2. How does representation of integers and floating-point values differ in C?

Integer representation: either unsigned or signed two's complement.

Floating point representation: IEEE 754.

IEEE 754 is essentially base-2 scientific notation. "Normalized" floating point values have the form $\pm 1.\underbrace{x}_{\text{"mantissa"}} \times 2^{\underbrace{y}_{\text{"exponent"}}}$

x is the fraction (represented in base 2)

y is the exponent (representeed in base 2, can be positive or negative)

Limitations of floating point

Arithmetic on floating point values may involve rounding. Results should generally be considered to be approximate.

Also: some numbers can't be represented exactly. For example, 0.1 has no exact representation (becomes a "repeating decimal" in the fraction.)

3. What is type narrowing?

Type narrowing is converting a value belonging to a “larger” numeric type to a “smaller” numeric type. E.g., converting a `double` value to an `int`.

Narrowing conversions may lose information.

For example:

```
float f_val = 3.5;
int i_val = f_val;  // narrowing conversion, i_val=3
```

4. What is type promotion?

A type promotion is converting a value belonging to a "smaller" numeric type to a "larger" numeric type. E.g., converting an `int` value to `double`.

Will *generally* not lose information, although some promotions (e.g., `int` to `float`) may lose information in some cases.

For example:

```
int i_val = 3;
double d_val = i_val; // promotion, d_val=3.0
```

5. What is type casting?

Type casting is an *explicit* conversion from one type to another.

Can be used to eliminate warnings in some cases:

```
// Without the cast, there is a warning
// (comparison of signed and unsigned values)
// in the loop condition
size_t len = strlen(str);
for (int i = 0; i < (int) len; i++) {
  char c = str[i];
  // ...
}
```

Other motivations for casts

In addition to avoiding compiler warnings, casts can also be useful to explicitly indicate where narrowing conversions are happening in the program.

6. What is the output of the code segment below?

```
int n = 32065; // in binary: 111110101000001
float x = 24.79;
                                      65


printf("int n = %d but (char) n = %c\n", n, (char) n);
                                  A

printf("float x = %f but (long) x = %ld\n", x, (long) x);
                                    24
```

Exercise 15

- Integer representation, random number generation
- Note that Part 3 is optional!
- Talk to us if you have a question!

Notes

Notes

Notes

Notes

Notes