601.220 Intermediate Programming

Spring 2023, Day 14 (February 22nd)

Today's agenda

- Exercise 13 review
- Binary file I/O, bitwise operations
- Exercise 14

Reminders/Announcements

- HW3 due Friday
  - Note: this homework is challenging, let us know if you have questions
- HW4: will be released Friday, due next Friday (March 3rd)
- Midterm project team registration
  - See Piazza post 233 for team registration form
  - Deadline is **Sunday, Feb 26th** by 11 pm
  - If you aren't registered by the deadline, we will assign you to a team

HW3: Important advice

When solving a problem recursively, you want the subproblem to be as *large* as possible.

In the context of the `match` function in HW3, the base cases occur when either the regex or word (or both) are empty.

So, progress means reducing the size of either regex or word (or both.)

This means that your `match` function should be examining **only** the very beginning of the regex and word.

If your code is looking further than (at most) the first two characters of either the regex or word,* it is doing unnecessary (and possibly counterproductive) work.

*Important exception: when handling the * wildcard, it could be necessary to consume up to restriction characters from the word

Exercise 13 review

In a `struct` data type, "`struct`" is part of the name of the data type.

E.g.:

```
struct Stat {
  int num_of_goals;
  int num_of_assists;
  float pass_accuracy;
  int min_played;
  int num_of_shots;
  float shot_accuracy;
};
```

The data type is "`struct Stat`".

Exercise 13 review

Common shortcut: use `typedef` to avoid the need to use "`struct`" when referring to the data type. E.g.:

```
typedef struct {
  int num_of_goals;
  int num_of_assists;
  float pass_accuracy;
  int min_played;
  int num_of_shots;
  float shot_accuracy;
} Stat;
```

Now "`Stat`" can be used as the name of the data type. The `main` function assumes you have done this for each `struct` type.

Exercise 13 review

Find the player with the latest signing date:

```
index = 0;
for (int i = 1; i < TEAMSIZE; i++) {
  Player *last = &team[index];
  Player *p = &team[i];
  if ( /* p's date is later than last's date */ ) {
    index = i;
  }
}
```

Exercise 13 review

Use valgrind to analyze memory use:

```
==2049== 24 bytes in 1 blocks are definitely lost in loss record 1 of 3
==2049==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgprelo
==2049==    by 0x109314: main (main.c:10)
==2049==
==2049== 132 bytes in 11 blocks are definitely lost in loss record 2 of 3
==2049==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgprelo
==2049==    by 0x1095A0: create_player (soccer.c:10)
==2049==    by 0x10981B: create_team (soccer.c:41)
==2049==    by 0x10932B: main (main.c:11)
==2049==
==2049== 240 bytes in 10 blocks are definitely lost in loss record 3 of 3
==2049==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgprelo
==2049==    by 0x10965A: create_player (soccer.c:22)
==2049==    by 0x10981B: create_team (soccer.c:41)
==2049==    by 0x10932B: main (main.c:11)
```

Exercise 13 review

Freeing memory:

```
for (int i = 0; i < TEAMSIZE; i++) {
  free(team[i].date);
  free(team[i].stat);
}
```

Observation: for each `Player` instance in the `team` array, the `date` and `stat` fields are pointers to dynamically allocated instances of `Date` and `Stat`.

Day 14 recap questions

1. How do we read/write binary files in C?
2. What character represents the bitwise XOR operation? How does it differ from the OR operation?
3. What happens if you apply the bitwise operation on an integer value? (extra: what if we apply to floats)
4. What is the result of `(15 >> 2) || 7`?
5. What is the result of `(15 >> 2) | 7`?

1. How do we read/write binary files in C?

Use `"rb"` or `"wb"` when calling `fopen`, and use `fread` or `fwrite` to read binary data value(s).

E.g., read array of 20 `int` values from a file of binary data:

```
FILE *in = fopen("input.dat", "rb");
if (in == NULL) { /* handle error */ }
else {
  int *arr = malloc(sizeof(int) * 20);
  int rc = fread(arr, sizeof(int), 20, in);
  if (rc != 20) { /* handle error */ }
}
// arr now points to array of 20 elements read from input file
```

Warning: this code is not portable across CPU architectures due to byte ordering. For multi-byte data values some CPUs store least significant byte first ("little endian"), some CPUs store most significant byte first ("big endian")

Binary data representation

"Binary" means "base 2".

Digital computers represent all numbers using base-2 rather than base 10. For example:

42 in base 10: $4 \times 10^1 + 2 \times 10^0$

42 in base 2:

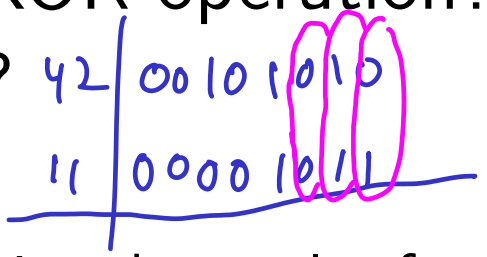
$$1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

$$= 1 \times 32 + 0 \times 16 + 1 \times 8 + 0 \times 4 + 1 \times 2 + 0 \times 1$$

42 as a sequence of binary digits ("bits"): 101010

All data values are represented in binary (base-2) at the machine level. "Bitwise" operators allow you to work directly with binary values.

2. What character represents the bitwise XOR operation? How does it differ from the OR operation?



Bitwise OR: combine two binary values by computing the result of the OR operation on each pair of binary digits.

Operator: `|`

Logic: `0|0=0, 0|1=1, 1|0=1, 1|1=1`

42|11 00101011

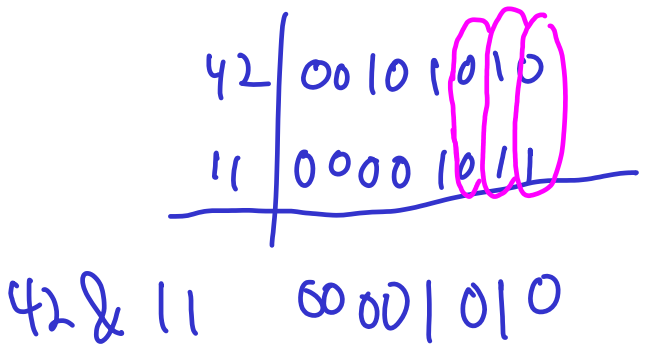
42^11 00100001

Bitwise XOR: combine two binary values by computing the result of the XOR (exclusive or) operation on each pair of binary digits.

Operator: `^`

Logic: `0^0=0, 0^1=1, 1^0=1, 1^1=0`

Bitwise AND



Bitwise AND: combine two binary values by computing the result of the AND operation on each pair of binary digits.

Operator: `&`

Logic: `0&0=0`, `0&1=0`, `1&0=0`, `1&1=1`

3. What happens if you apply the bitwise operation on an integer value? (extra: what if we apply to floats)

The two-operand bitwise operators (`|`, `^`, `&`) perform a logical operation (OR, XOR, AND) on each pair of bits in the two operands.

The operands must be values belonging to an "integral" (integer-like) type.

E.g., `int`, `unsigned`, `long`, `unsigned long`, `char`, `unsigned char`, etc.

Bitwise operations may *not* be performed on floating point (`float` or `double`) values.

4. What is the result of `(15 >> 2) || 7`?

15 in base-2 is 1111 ⇒ 0011

"`>>`" is the right shift operator, shifting 1111 two bits to the right yields 0011, which is equal to 3.

Any non-zero integer is considered TRUE, so 3 is true.

If the left operand of `||` is true, the entire expression is true (and the right operand is not evaluated.) All logical and relational operators yield 0 when false and 1 when true.

So, the result of `(15 >> 2) || 7` is 1.

5. What is the result of `(15 >> 2) | 7`?

`|` is the bitwise OR operator

15 in binary is 1111

(15 >> 2) is 0011

7 in binary is 0111

```
  0011
| 0111
------
  0111   -- bit is 1 where either operand has a 1 bit
```

which is equal to 7

Exercise 14

- arrays and strings
- bitwise operations
- Talk to us if you have questions!

Notes

Notes

Notes

Notes

Notes