601.220 Intermediate Programming

Spring 2023, Day 6 (February 3rd)

Today's agenda

- Exercise 5 review
- File I/O, functions, command line arguments
- Exercise 6

Reminders

- HW0 due *this evening* by 11pm
- HW1 due Friday, Feb 10th

Exercise 5 review

- Copying `.bashrc` and `.bash_profile` from the public repo
- These are “shell startup scripts”
  - Make `emacs` the default editor
  - Add `gccc` and `g+++` aliases for running `gcc` and `g++` with the recommended compiler options

Exercise 5 review

`count1.c`: iterate backwards over original sequence, build complement sequence:

```
for (int i = dna_len - 1; i >= 0; i--) {
  char complement = '?';
  switch (dna[i]) {
  case 'A': complement = 'T'; break;
  case 'T': complement = 'A'; break;
  case 'C': complement = 'G'; break;
  case 'G': complement = 'C'; break;
  default: /* bad data */; break;
  }
  rev_comp[rci] = complement;
  rci++;
}
```

Exercise 5 review

Set NUL terminator at end of complement sequence:

```
rev_comp[rci] = 0;
```

Exercise 5 review

count2.c: classify characters using <ctype.h> functions:

```
for (int i = 0; i < text_len; i++) {
  char c = text[i];
  if (isalpha(c)) { num_alpha++; }
  if (isdigit(c)) { num_digits++; }
  if (isspace(c)) { num_space++; }
}
```

Exercise 5 review

Could also use knowledge of how characters are encoded in ASCII:

```
for (int i = 0; i < text_len; i++) {
  char c = text[i];
  if ((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z'))
    { num_alpha++; }
  if (c >= '0' && c <= '9')
    { num_digits++; }
  // ...etc...
}
```

Using the `<ctype.h>` functions is simpler, and makes the program more readable.

Exercise 5 review

`count3.c`: initializing array of per-character counts:

```
int ascii_count[256] = {0};
```

Note that when an array initializer has fewer initial values than the array has elements, the remaining elements are initialized to zero.

So, all of the elements of `ascii_count` are set to 0 initially.

Exercise 5 review

Tabulating occurrence counts for each character value:

```
for (int i = 0; i < text_len; i++) {
  int c = text[i];
  assert(c >= 0);
  ascii_count[c]++;
}
```

Note `char` values can be negative, but using a negative array index would result in an invalid access. Hence, the use of `assert`. It is a precondition that all of the character values must be non-negative.

Also note that gcc will complain if you try to use a `char` value as an array index.

Exercise 5 review

Finding most frequent and second most frequent characters:

```
for (int i = 0; i < 256; i++) {
  if (ascii_count[i] > top_freq) {
    top_char = (char) i;
    top_freq = ascii_count[i];
  } else if (ascii_count[i] > next_freq) {
    next_char = (char) i;
    next_freq = ascii_count[i];
  }
}
```

Day 6 recap questions

1. Is `fprintf(stdout, "xxx")` the same as `printf("xxx")`?
2. When should we use assertions instead of an *if* statement?
3. What will happen if you pass an `int` variable to a function that takes a `double` as its parameter? What will happen if a `double` is passed to an `int` parameter?
4. What is "pass by value"?
5. How do you change the *main* function so that it can accept command-line arguments?

1. Is `fprintf(stdout, "xxx")` the same as `printf("xxx")`?

Yes.

2. When should we use assertions instead of an *if* statement?

An assertion (use of the `assert` macro) means "this condition must be true, or else we have proved that there is a bug in the program."

Assertions are useful for checking *invariants*. They are also useful for *unit testing*. A unit test is an automated test for a small "unit" of the program, typically a single function. Assertions are used in a unit test to verify that the code being tested behaved correctly.

Assertions should *never* be used to check for conditions that could be true in the normal operation of the program. For example, it would be incorrect to use an assertion to check whether a file was opened successfully.

3. What will happen if you pass an `int` variable to a function that takes a `double` as its parameter? What will happen if a `double` is passed to an `int` parameter?

An `int` value can be freely converted to a `double` with no loss of information. The `double` value will be numerically the same as the original `int` value.

If a `double` is converted to an `int`, it is *truncated*, i.e., the fractional part is discarded.

Conversions

```
// conversions.c:
#include <stdio.h>
#include <assert.h>

int as_int(int x) { return x; }
double as_double(double x) { return x; }

int main(void) {
  assert(as_double(3) == 3.0);
  assert(as_int(6.79) == 6);
  printf("tests passed\n");
  return 0;
}
```

```
$ gcc -std=c99 -Wall -Wextra -pedantic conversions.c
$ ./a.out
tests passed
```

4. What is “pass by value”?

“Pass by value” means that a parameter is a variable that is distinct from any other variable in the program. Changing the value of a parameter does not change the value of any other variable in the program.

C uses pass-by-value for all parameters except for array parameters.

Pass by value example

```
// pbv.c:
#include <stdio.h>

void f(int x) {
  x = 42;
  printf("x=%d\n", x);
}

int main(void) {
  int y = 17;
  f(y);
  printf("y=%d\n", y);
  return 0;
}
```

```
$ gcc -std=c99 -Wall -Wextra -pedantic pbv.c
$ ./a.out
x=42
y=17
```

5. How do you change the *main* function so that it can accept command-line arguments?

Declare it like this:

```
int main(int argc, char *argv[]) {
  // ...
}
```

`argc` is one greater than the number of command line arguments.

`argv` is an array of strings, where the strings are the command line arguments. (Note that `argv[0]` is always the name of the program.)

Command line arguments example

```c
// cmdargs.c:
#include <stdio.h>

int main(int argc, char *argv[]) {
  for (int i = 0; i < argc; i++) {
    printf("argv[%d] is '%s'\n", i, argv[i]);
  }
  return 0;
}
```

```
$ gcc -std=c99 -Wall -Wextra -pedantic cmdargs.c
$ ./a.out C is a "fun language"
argv[0] is './a.out'
argv[1] is 'C'
argv[2] is 'is'
argv[3] is 'a'
argv[4] is 'fun language'
```

A quick synopsis of C file I/O

- The `FILE*` type represents an open file (for reading or writing)
- Opening a file for reading:
  `FILE *in; in = fopen(`*filename*`, "r");`
- Opening a file for writing:
  `FILE *out; out = fopen(`*filename*`, "w");`
- If `fopen` returns `NULL`, it means the file wasn't opened successfully
- Use `fscanf` to read from a file, `fprintf` to write to a file
- When the program is done with a file, use `fclose` to close it

Exercise 6

Compound interest ($\leftarrow$ click for formulas to calculate)

Talk to us if you have questions!

Notes

Notes

Notes

Notes

Notes