CS 240: Programming in C

Lecture 5: assert(),
Random-access File I/O

PURDUE UNIVERSITY

Announcements

- Homework 2 was released on Monday
  - It requires significantly more time and effort!
  - If you have questions, try the examples on the web pages
  - Try the examples in your textbook
  - Don’t forget about the code standard
- Have you started it yet?

Announcements

- Homework 1 is due tonight at 9:00 pm!
- You have one week to submit a regrade request
  - after grades have been released
- To request a regrade, email me
- For code style regrades, use Gradescope

Reminder

- You can attend any lab section for help
  - There are a lot of sections -- see the course webpage
  - Students registered for that section are given priority

Comment your code

- Part of the code standard assessment includes reasonably commenting complex parts of your code

Good comments

- Put the code in context and describe its impact on execution and data
- Do not restate in English what can be ascertained immediately by reading the code
- Comments should add clarity, not confusion
- They must be maintained along with the code
- Add comments when fixing bugs
- Use comments to mark incomplete implementations
  - /* TODO: This only works for simple inputs */

Bad examples

```
i = i + 1; // Add one to i

// iterate from 0 to 9
for (int i = 0; i < 10; i++) {
    array[i]++;
}

// read in an integer from stdin
scanf(“%d\n”, &my_int);
```

Good examples

```
// Increment first 10 elements of array
for (int i = 0; i < 10; i++) {
    array[i]++;
}

// Handle malformed input
if (status < 12) {
    fclose(fp);
    return BAD_RECORD;
}

// Trim trailing spaces (s is not resized)
while (*s && *s != ‘ ’) s++;
*s = ‘\0’;
```

Homework 2 hints

- What is wrong with this?

```
in_fp = fopen(input, “r”);
if (in_fp == NULL) return 0;
out_fp = fopen(output, “w”);
if (out_fp == NULL) return 0;
```

Homework 2 hints

- What is wrong with this?
  - in_fp won't be closed if out_fp fails to open!

```
in_fp = fopen(input, “r”);
if (in_fp == NULL) return 0;
out_fp = fopen(output, “w”);
if (out_fp == NULL) return 0;
```

- The return value for fprintf() tells you how many characters it printed
  - It should always be > 0
- Remember, for help on any function (e.g., fprintf), type:

```
$ man fprintf
```

Homework 2 hints

- Read about strcmp()
- Remember, fscanf() needs pointers

```
int array[3];
scanf("%d %d %d", array, &array[1], &array[2]);
```

  - `array == &array[0]`
  - More on this later
- Like Java, C will do integer division when given integers
  - Cast the denominator to a float:

```
float my_f = some_int / (float) another_int;
```

The assert() macro

- Use assert() in your code to say, “I make an assertion that this **must** be true.”

```
assert(non_neg_value >= 0);
```

- If the condition is not true, the program aborts
  - And it tells you exactly where it occurred

```
my_prog: my_prog.c:10: main: Assertion
‘non_neg_value >= 0’ failed.
```

Use assert() iff you have to

- If there is any way of detecting and recovering from an error condition, do so
- If you cannot recover, it is better to stop execution rather than to continue on
  - Especially while your code is being developed
- If you can proactively insert checks into your program, you'll more easily detect problems.
- When you "ship" your program, you can disable assertion checks

When to use assert()

- **Do** use assert() to ensure you detect error conditions that you cannot handle

```
assert(impossible == 0);
```

- **Do** use assert() to double check expectations for your code

```
assert(non_negative >= 0);
```

- **Do not** use assert() to handle conditions that you can take care of otherwise...

```
fp = fopen(“my.file”, “w”);
assert(fp != 0);
```

A complete example

```
#include <stdio.h>
#include <assert.h> /* Must include this! */

int main() {
    int x = 1;

    printf(“Here I make a true assertion.\n”);
    assert(x == 1);

    printf(“Right. Nothing happened.\n”);
    assert(x == 12);

    return 0;
}
```

A non-trivial example

```
#include <stdio.h>
#include <assert.h>

int main() {
    short int counter = 0;
    int sum = 0;

    do {
        counter++;
        sum = sum + counter;
        assert(counter > 0);
    } while (counter != 0);

    printf(“Sum is %d\n”, sum);
    return 0;
}
```

Turning off assert() for the “customer”

- To turn off assertion checks:

```
$ gcc -DNDEBUG=TRUE -o my_prog my_prog.c
```

- All assertion checks will be skipped

When we grade things...

- When we grade your programs, we'll do things that force assertion checks to fail to make sure you properly add them.
- You should try your program with assertion checks turned off before turning it in!
  - Why?

Bad example

```
/*
 *  If x is even, add 1 to it to make it odd.
 *  Check that we still have a positive number.
 */
if (x % 2 == 0) {
  assert(++x > 0);
}
```

Bad example

```
/*
 *  If x is even, add 1 to it to make it odd.
 *  Check that we still have a positive number.
 */
if (x % 2 == 0) {
  assert(++x > 0);
}
```

- The problem is that if assertion checks do not happen, that increment doesn’t happen either
- Only put **comparisons** inside assert()

assert() is your friend

- Any questions?

Random file access

- We can use an open file pointer to say the following things:
  - Where are we at in the file? (what's our offset?)
  - Go back to the beginning of the file
  - Go directly to the end of the file
  - Go to some arbitrary position in the file

ftell()

- The ftell() function is used to find out where we are in the file

```
int ftell(FILE *file_pointer);
```

- The return value is either:
  - The current offset from the beginning of the file
  - -1 in the event of an error

fseek()

- The fseek() function is used to “go somewhere” in the file

```
int fseek(FILE *fp, long int offset, int whence);
```

- fseek() return 0 on success and -1 in the event of an error
  - If unsuccessful, file position stays the same
- “whence?”

fseek() from whence

- The “whence” value can be one of three things:
  - SEEK_SET: new offset will be relative to the beginning of the file
  - SEEK_CUR: new offset will be relative to the current position of the file
  - SEEK_END: new offset will be relative to the end of the file

Examples

```
fseek(file_pointer, 0, SEEK_SET);
```

- Move to the beginning of the file

```
fseek(file_pointer, 0, SEEK_END);
```

- Move to the end of the file

```
fseek(file_pointer, -14, SEEK_CUR);
```

- Move backward fourteen bytes

```
fseek(file_pointer, 28, SEEK_SET);
```

- Move to the 28th byte of the file

```
fseek(file_pointer, -12, SEEK_END);
```

- Move to 12 bytes before the end of the file

Some tricks

- Find out how long a file is...

```
fseek(file_pointer, 0, SEEK_END);
len = ftell(file_pointer);
fseek(file_pointer, 0, SEEK_SET);
```

- Have we hit EOF?

```
fgets(buffer, 100, file_pointer);
if (ftell(file_pointer) == len) { ... }
```

- You probably don’t need to use these tricks in your homework submissions or on exams

Strange example

```
#include <stdio.h>

int main() {
  FILE* fp = 0;
  long int offset = 0;
  char buffer[100];

  fp = fopen(“input.txt”, “r”);
  fscanf(fp, “%99s”, buffer);
  offset = ftell(fp);
  while (1) {
    fscanf(fp, “%99s”, buffer);
    printf(“%s\n”, buffer);
    fseek(fp, offset, SEEK_SET);
  }
  return 0;
}
```

input.txt

```
abcdefg\n
hijklmnop\n
qrstuv\n
wxyz\n
```

Uses for random file access

- Accessing a file as a database
- Extracting parts of a file
- Modifying a few bytes in the file without writing the whole thing
  - Only works on byte ranges (not lines)
  - There is no way to move forward to backward by N lines other than reading them in

Takehome Quiz #1

- Write a function named print_chars() that returns an integer and accepts a string (char *) argument.
  It should:
  - Open a file for reading with the name specified by the argument
  - If the file cannot be opened, return ERR_ON_OPEN
  - Use fscanf() to read **exactly 50 characters** from the file into an array
    - Including any whitespace! Read scanf documentation carefully (K&R 7.4)
  - If 50 characters can't be read, return ERR_ON_READ
  - Use printf() to output every non-whitespace character in reverse order
    - Hint: see functions in K&R 7.8
  - Close the file
  - Return OK

Quiz Reminder

- Due in 24 hours
- Handwritten answers only

For next lecture

- Read Chapter 6.1-6.3, 6.7-6.9 in K&R
  - ...and/or Chapter 8 in Beej's
  - Ignore anything about pointers!

Slides

- Slides are heavily based on Prof. Turkstra's material from previous semesters.