



CS 240: Programming in C

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

Prof. Jeff Turkstra



Announcements

- Homework 1 final due date tonight
- You've already started Homework 2, right?
 - Due Sunday
 - Final due date Wednesday 2/5 9pm
 - If you have questions, try the examples on the web pages FIRST
 - Try the examples in your textbook
- Don't forget about the code standard
 - 20 points!

**IF YOU WAIT UNTIL THE LAST
MINUTE TO START YOUR HOMEWORK**



**YOU'RE GONNA
HAVE A BAD TIME**

imgflip.com

Notes

- Regrade request policy of one week is for initiating the request
 - If it takes longer than a week to resolve, that's okay
- Gradescope assessments will not be loaded into the gradebook until **after** regrade request window closes
- Please let me know if you are having any difficulty with TAs



Reminder

- You can attend the second half of any lab section!
 - There are lots of labs!

Feasting with Faculty

- Great opportunity to interact with me and fellow students outside of classroom
 - Lots of fun discussions in the past
- Earhart Dining Court
 - Private Dining Room B (hopefully, otherwise look around for a table)
- Every Thursday 12pm – 1pm
 - First one is **next week!**



Feasting with Faculty

- Welcome to attend any/all lunches
- Will invite individuals in batches
 - Only one invite
 - Don't have to wait (might not get one until the end!)

Comment your code

- Part of the code standard assessment will include reasonably commenting complex parts of your code
 - We've had some contradictory information posted on Ed Discussion

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
 - After a release, not while developing
- Use comments to mark incomplete implementations
 - TODO: This only works for simple inputs
 - Jeff: This doesn't look right, please review

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);
```

etc

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';
```

Reading

- Read Chapter 6.1-6.3, 6.7-6.9 in K&R
 - ...and/or Chapter 8 in Beej's
 - If you see anything about pointers,
COVER YOUR EYES!

What can go wrong?

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

```
int main() {  
    int z = 0xffffffff;  
    char buf[8];  
    int y = 0xffffffff;  
  
    scanf("%s", buf);  
  
    printf("read: %s\n", buf);  
    printf("z = %d, y = %d\n", z, y);  
  
    return 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;
```

- 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



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

`my_prog: my_prog.c:10: main: Assertion
'non_neg_value > 0' failed.`

- And it tells you exactly where it occurred

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", counter);
    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 added them. We know how to keep your program from stopping when that happens
- 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);  
}
```

- 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?

Purdue trivia

- "Harry Creighton Pepper, first head of the School of Chemical Engineering, ordered a camera through normal university channels. Purchasing Agent H. C. Mahin wrote to ask Pepper what he intended to do with it. The implications of Mahin's query so angered Pepper that he fired back one of the best letters he ever regretted. It was, Pepper wrote to Mahin in high dudgeon, 'none of your goddam business' what he did with the camera, but since Mahin wanted to know, he intended to take pictures with it." The letter eventually made its way to President Elliott's desk. "Pepper was called to the president's office where Elliott told him that writing such a letter to an administrative officer was the same thing as writing it to him. Pepper was unimpressed. He pointed out to Elliott that he too was an administrative officer and that using the same logic, Elliott could conclude that he had written the letter to himself."
'Pepper, what am I going to do with you?' the exasperated president asked.
'I haven't the slightest idea,' Pepper retorted.
'Get out of here and get back to your office,' Elliott said with finality."

- A Century and Beyond, by Robert W. Topping

Takehome Quiz 2

1. Write a function named `get_characters()` that returns an integer and accepts a string (`char *`) argument. It should:

- Open a file for read with the name specified by the argument
- If the file cannot be opened, return `ERR_ON_OPEN`
- Use `fscanf()` to read 50 *characters* from the file into an `ARRAY`
- Use `printf()` to output every other character in reverse order
- Close the file
- Return OK



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()` returns 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 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(int argc, char *argv[]) {
    FILE *fp = 0;
    long int offset = 0;
    char buffer[100];

    fp = fopen(argv[1], "r");
    fgets(buffer, 100, fp);
    offset = ftell(fp);
    while (1) {
        fgets(buffer, 100, fp);
        printf("%s", buffer);
        fseek(fp, offset, SEEK_SET);
    }
    return 0;
}
```



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 or backward by N lines other than reading them in

Tips for Homework 2

- 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;
```
 - This is the **only** time we want you to use casts (for now)



typedef

- Standard 'C' has a set of built-in **types** such as 'int', 'char', 'float', etc...
- You can create your own types as long as they are not already C keywords

- Examples:

```
typedef int number;  
typedef double array[3];
```

```
number n = 5;  
array arr = { 1.5, 2.9, 3.7 };
```

When to use typedef?

- Use typedef when you have a variable type that is used a lot and...
 - ...has a really long, cumbersome description. E.g.:
`typedef double * const *ptr;`
 - ...has a parameter type that you might change sometime later. E.g.:
`typedef double array[3];`
 - ...it is a structure (more on this today)
 - ...it is so confusing that you really can't deal with it without creating a typedef (more later this semester)

Getting the syntax of typedef...

- The syntax of typedef might seem backwards to you. Here's how to always get it right...
 - Pretend that you're defining a variable of the type that you want to see
 - Let the variable name be the name of the new type
 - Add 'typedef' to the beginning of the definition

typedef syntax cont

- For instance, if you want a type called uint5 that can be used to define an array of five unsigned integers,
 - Pretend you're defining a variable:
`unsigned int uint5[5];`
 - Make it a type:
`typedef unsigned int uint5[5];`
 - Use it:
`uint5 arr = { 1, 2, 3, 4, 5};`

Questions about typedef?

Introduction to structures

- Large programs usually have many data
(did you know the word 'data' is plural?)
- Instead of creating a separate variable for each one, it is helpful to group them together to better organize them
- A structure is the thing that allows you to use one name to refer to many variables

What does a structure look like?

- Type declaration:

```
struct my_data {  
    int age;  
    float height;
```

```
} ;
```



Note the semicolon!

- Storage definition and initialization:

```
struct my_data my_var = { 19, 5.3 };
```

Declaration/definition together

```
struct my_data {  
    int age;  
    float height;  
} my_var = { 19, 5.3 };
```

Accessing elements

- Once a structure variable has been defined, you can access its internal elements with the dot (.) operator:

```
my_var.height = 6.1;
```

```
x = my_var.age;
```

```
printf("Age: %d, Height: %f\n",  
      my_var.age, my_var.height);
```

Properties of structures

- Anything that can be defined can also be defined inside of a struct { ... }
- Except functions
- You can put arrays in structures
- You can put other structures in structures
- You can put them in any order:

```
struct stuff {  
    double d_var;  
    int     i_arr[100];  
    float   f_arr[20];  
};
```



More properties of structures

- There's no limit to the number of elements in a structure
 - Each must have a unique name, though
- Structures can be passed to functions and returned from functions:

```
struct my_data grow(struct my_data start) {  
    struct my_data finish;  
  
    finish.age = start.age + 1;  
    finish.height = start.height * 1.1;  
    return finish;  
}
```

Even more properties of struct

- You can assign one structure variable to another:

```
int main() {  
    struct my_data small_kid = { 1, 2.5 };  
    struct my_data big_kid = { 3, 4.1 };  
  
    small_kid = big_kid;  
  
    ...  
}
```

- Note: This is an *assignment*. Not an *initialization*.

```
small_kid = { 2, 3.2 }; /* nope! */
```

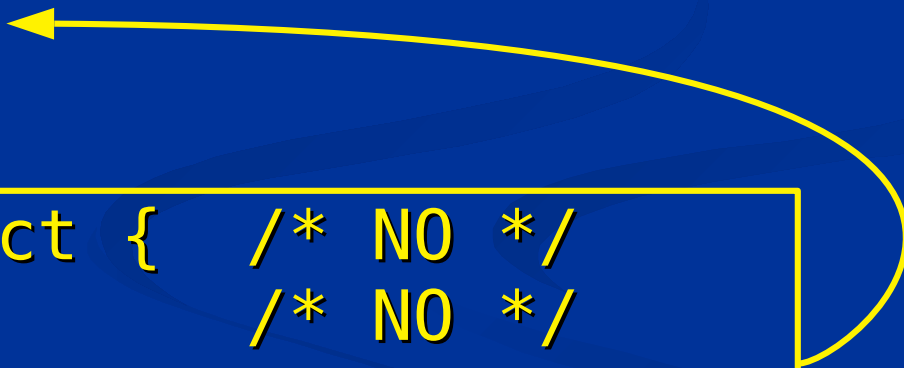
- But with C99...

```
small_kid = (struct my_data) { 2, 3.2 };
```

Where to put declarations

- Structure variables can be defined inside or outside of a function
 - We prefer to see the declaration outside

```
int func(int arg) {  
    struct my_new_struct {    /* NO */  
        int x;                /* NO */  
        int y;                /* NO */  
    };                        /* NO */  
    struct my_new_struct s1;  
}
```



For next lecture

- Read Chapter 6.1-6.3, 6.7-6.9 in K&R
 - ...and/or Chapter 8 in Beej's
 - If you see anything about pointers,
COVER YOUR EYES!

Boiler Up!