Lab 03

September 20, 2017

Testing your assumptions

Very often when programming, certain assumptions are made. While this helps simplify the task at hand, sometimes our assumptions can be wrong. For example, if you are certain that a list will never have more than n elements, you can state this with an assertion:

```
assert(list_size <= n);
```

Now `list_size` will be compared with `n` every time the `assert()` call is reached.

A few important notes about using assertions:

- you must `#include <assert.h>`
- `assert()`s should only be used by you (the developer) to validate your assumptions – they should not be used for error handling
- `assert()`s can be disabled by adding `-DNDEBUG` to your compiler flags
- the golden rule of using assertions: *they cannot change any state in your program* (had the above assertion used `n++` instead of `n` the program would no longer have the same semantics)

If `malloc()` fails that is typically not recoverable; checking the value returned from `malloc()` is **not** a good candidate for using assertions (even though we do it in the sample code). Checking that a pointer has a valid value before you dereference it may be acceptable but only if you expect it to be valid, i.e., that is the behavior you expect from your program.

Download assert_test.c and get it working. Don't forget to turn on all warnings with the `-Wall` option to either `clang` (recommended) or `gcc` (OK, too).

Grades

Write a grading program that asks for a student's grades. Use `malloc()` to allocate enough space for all of them. Use this web page to only print out two digits of precision (search for precision).

```
How many grades does the student have? 4
Enter the next grade: 92
Enter the next grade: 98
Enter the next grade: 100
Enter the next grade: 93
The average grade is 95.75
```

In addition to asserting the memory returned from `malloc()` is valid, can you think of any other valid assumptions you can use to test your code? A student can definitely have five grades – can that student have zero grades? Would that be better handled by normal error checking? Can you think of any other interesting assumptions you can make?

Better Grades Program

Repeat the program above but this time create a `struct student` that should be defined like so:

```
struct student
{
    char name[100]; // No one should have a name longer than this
    int *grades;    // This is the pointer to hold the (integer) grades
    int count;      // The number of grades
};
```

Ask how many students and allocate them. For each student, ask how many grades and allocate those as well. Sample output below:

```
How many students are there? 2

Enter the name of student 0: Jenny
How many grades does Jenny have? 4
Enter the next grade: 92
Enter the next grade: 98
Enter the next grade: 100
Enter the next grade: 93

Enter the name of student 1: Bill
How many grades does Bill have? 3
Enter the next grade: 91
Enter the next grade: 88
Enter the next grade: 94

Jenny has an average grade of 95.75
Bill has an average grade of 91.00
```