CMSC 202	**Homework 5**	Spring 1996
Sections 101, 102 & 103	**due: Tuesday, 9 April**	Handout #19

This assignment is designed to give you more experience implementing and using data storage structures. The assignment has two parts: implementation of a new ADT (the *fleezle*); and writing functions that use an existing ADT (the stack).

1 Abstract Data Type Implementation

A *fleezle* is a data storage structure that holds integers. The fleezle ADT consists of the following components:

- `fleezle_type`—the type of a fleezle.
- `fleezle_item_type`—the type of number contained in a fleezle (namely, `int`).
- `fleezle_type create_fleezle(void)`—create and return a new empty fleezle.
- `void add_to_fleezle(fleezle_item_type item, fleezle_type fleezle)`—add an item to the specified fleezle.
- `fleezle_item_type get_biggest(fleezle_type fleezle)`—return the largest value stored in the fleezle. As a side–effect, remove that item from the fleezle.
- `fleezle_item_type get_smallest(fleezle_type fleezle)`—return the smallest value stored in the fleezle. As a side–effect, remove that item from the fleezle.
- `BOOL empty_fleezle(fleezle_type fleezle)`—return `TRUE` or `FALSE` according to whether the given fleezle contains any items.
- `void transform_fleezle(fleezle_type fleezle, fleezle_item_transformer transformer)`—replace every item in the given fleezle with the result of applying `transformer` to that item. A `fleezle_item_transformer` is a pointer to a function that takes a `fleezle_item_type` as its argument and returns a new `fleezle_item_type`. For example, if `my_fleezle` is a fleezle containing the numbers 12, 17, and 25, and `my_transformer` is a function that returns its argument plus seventeen, then a call to `transform_fleezle` on these arguments will cause `my_fleezle` to contain the values 29, 34, and 42.

The file `fleezle.h` appears in the standard CMSC 202 library. You must use this version of fleezle.h. To do so, you must:

- `#include "fleezle.h"` at the top of your file containing your fleezle functions (which must be called `fleezle.c`).
- Use `typedef` somewhere in your file `fleezle.c` to define a `struct fleezle_tag` (the file `fleezle.h` defines `fleezle_type` to be a pointer to such a struct).
- Compile your `.c` files using "`-I ~mayfield/lib202`."
- If you build a test program to use your functions, you will need to link your `.o` files using "`-L ~mayfield/lib202 -l202`" at the end of the compilation line.

2 Abstract Data Type Use

You are to write two versions of the `pop_nth` function. `Pop_nth` takes two arguments: a stack, and an integer. It returns as its result the item in the stack at the integer position. The item at the top of the stack is number zero, the item just below the top item is item one, and so on. As a side–effect, `pop_nth` must remove the value returned from the stack, but leave the stack otherwise unchanged. For example, if the values 17, 42, 12, 88, and 59 have been pushed on a stack called `stack` in that order, then a call to `pop_nth(stack, 3)` should return 42; after the call, `stack` should contain the values 17, 12, 88, and 59.

Your first version of `pop_nth` must use a temporary stack to hold values popped from the original stack. Your second version of `pop_nth` must use recursion to accomplish the same thing; it must not use a temporary stack.

The file `popnth.h` appears in the standard CMSC 202 library. You must use this version of popnth.h. To do so, you must:

- `#include "popnth.h"` at the top of each file containing one of your `pop_nth` functions (which must be called `popnth-stack.c` and `popnth-recursive.c`).
- Compile your `.c` files using "`-I ~mayfield/lib202`."
- If you build a test program to use your functions, you will need to link your `.o` files using "`-L ~mayfield/lib202 -l202`" at the end of the compilation line.

Submission

Submission will be electronic. You must submit the following files:

- `fleezle.c`: This file must contain the C code for the functions `create_fleezle`, `add_to_fleezle`, `get_biggest`, `get_smallest`, `empty_fleezle`, and `transform_fleezle`.
- `popnth-stack.c`: This file should contain the C code for the function `pop_nth` that uses a temporary stack.
- `popnth-recursive.c`: This file should contain the C code for the function `pop_nth` that uses recursion.

It is OK to write auxiliary functions, if you wish. However, none of the three files that you submit should contain a `main` function.

Your files must be named exactly the names given here. You should write test routines and use them to test each of your functions. However, you should not submit them. Nor should you submit your Makefile. Submit only the files `fleezle.c`, `popnth-stack.c`, and `popnth-recursive.c`.

To submit your files, use the submit command:

```
submit cs2021 hw5 fleezle.c popnth-stack.c popnth-recursive.c
```

We will automatically compile and run your program on a fixed set of data. You may submit your program more than once, but only the last version you submit will be graded. Submissions made by midnight on the due date (Tuesday, 9 April) will receive the on–time bonus. You may have an automatic extension until Thursday, 11 April; submission of homework will be disabled at midnight on the 11^{th}.