Wesleyan University, Fall 2021, COMP 211
Homework 4: Arrays and searching
Due by 11:59pm on October 5, 2021

---

1. Written problems (5 points)

Problem 1. *Consider the pseudocode in Program 1 and assume execution starts at the first line of* `main` *(i.e., assume you have a stack with a single empty binding table just before executing line 12). Recall that malloc is used to dynamically allocate memory.*

Explain what the values of A and B are at line 16. Describe any issues that might arise from the execution of function `f`*. Justify your answer by describing how the environment changes as the program executes (and in particular when* `f` *is called). Assume a character requires two bytes of storage.*

Solution: At line 16, A is the array `['a', 'b', 'c', 'q']` and B is the array `['d', 'e', 'f']`. A major issue will arise after function `f` is executed is that after the function returns, there will be no way to reach the memory allocated in the function, creating a memory leak. Let's analyze how the environment changes to see that this is the case.

(1) Executing the declarations just creates uninitialized entries in the current binding table for A and B, so after the declarations the environment is

$$S = \{A \rightarrow ?, B \rightarrow ?\}.$$

(2) After executing the assignments in `main`, there are changes to both the current binding table and to memory. Evaluation of an array expression reserves a block of memory that is large enough to hold the array elements; the value of the array expression is the first memory location of the block. Suppose the evaluation of `['a', 'b', 'c', 'm']` allocates memory starting at location 400. Then the (representations of the) characters `'a'`, `'b'`, `'c'`, and `'m'` are stored in memory starting at memory location 400. Thus memory location 400 stores the (representation of) `'a'`, memory location 402 stores `'b'`, memory location 404 stores `'c'`, and memory location 406 stores `'m'`. And the value assigned to A in the current binding table is the memory address 400. Similarly, if the evaluation of `['d', 'e', 'f']` allocates memory starting at location 500, then memory locations 500, 502, and 504 store `'d'`, `'e'`, and `'f'`, respectively, and the value assigned to B in the current binding table is memory address 500. Thus the environment is

$$S = \{A \rightarrow 400, B \rightarrow 500\}.$$

(3) When we call `f`, we add a new binding table, S', to the stack, with initial bindings that bind the values of the arguments in the call to the corresponding parameters. Thus the environment is

$$S = \{A \rightarrow 400, B \rightarrow 500\}; S' = \{A \rightarrow 400, B \rightarrow 500\}$$

(4) To execute line 3, the representation of `'q'` is stored at the memory address $S'(A) + 3 \cdot 2$, where S' is the current binding table, 3 is the index being assigned to, and 2 is the size of the representation of a `char` value. Since $S'(A) = 400$, this memory address is 406, and so the memory at location 406 is set to the representation of `'q'`. So now the memory

```
1  void f(char* A, char* B) {
2
3     A[3] = 'q';
4     B = malloc((sizeof(char) * 3));
5     B[0] = 'x';
6     B[1] = 'y';
7     B[2] = 'z';
8  }
9
10 int main(void) {
11
12    char A[] = {'a', 'b', 'c', 'm'};
13    char B[] = {'d', 'e', 'f'};
14    f(A, B);
15
16    // What are values of A and B here?
17
18    return 0;
19 }
```

PROGRAM 1. A program with a function that manipulates arrays.

starting at location 400 stores the characters `'a'`, `'b'`, `'c'`, `'q'` and the memory starting at location 500 stores the characters `'d'`, `'e'`, `'f'`. There is no change to the stack of binding tables.

(5) To execute line 4 a new block of memory is allocated say starting at location 900, so now S'(B) = 900 rather than 500. Lines 4-6 then fill it with the representations of 'x', 'y', and 'z'. So the memory locations 900, 902, and 904 store the characters `'x'`, `'y'`, and `'z'`, respectively. Note however, that the data starting at memory locations 400 and 500 has not changed. Thus the environment is

$$S = \{A \to 400, B \to 500\}; S' = \{A \to 400, B \to 900\}$$

(6) When we return, we pop the top binding table from the stack, so the environment is

$$S = \{A \to 400, B \to 500\};$$

The contents of the memory starting at locations 400, 500, and 600 have not changed.

So now we can see what array A represents. Since the value of A in the current binding table is 400, the array consists of the four characters that start at memory location 400, which are `['a', 'b', 'c', 'q']`. Since the value of B is the current binding table is 500, the array consists of the three characters that start at memory location 500, which are `['d', 'e', 'f']`. The value, however of B in the binding table S', however, has been lost, and there is now no way to reach the memory that was allocated.

2. Coding problems (15 points)

When you are given pre-conditions to functions, you may assume that these conditions hold when your function is called. You do not have to verify that they are true of the arguments, and your functions will only be tested on arguments that meet the pre-conditions.

Problem 2. *Write a function* `merge` *that satisfies the following specification:*

- **Function header.** `void merge(int A[], int m, int B[], int n, int C[])`
- **Pre-condition.** $A[0] \leq \cdots \leq A[m-1]$ *and* A *has size* m*;* $B[0] \leq \cdots \leq B[n-1]$ *and* B *has size* n*; and* C *has size* $m+n$*. In other words, the pre-conditions state that* A *and* B *are sorted in non-decreasing order.*
- **Function body.** *This should satisfy the following: when* $\texttt{merge}(A, m, B, n, C)$ *returns,* C *comprises all elements of* A *and* B*, and furthermore* $C[0] \leq \cdots \leq C[m+n-1]$*, i.e.,* C *is also sorted in non-decreasing order.*

Your function must have cost $O(m+n)$*. The basic idea here is to "zipper" together the elements of* A *and* B*. You will need two variables, one of which marches through the indices of* A *and the other of which marches through the indices of* B*. The indices tell you which elements of* A *and* B *could be the next one to be added to* C*; which one is actually added will depend on how they compare to each other. You will need to do the marching simultaneously with an appropriate loop. You should write out some examples on paper (pictures, not code) to get a feel for how your algorithm should behave before trying to write this code.*

Problem 3. *Write a function* `bin_search` *that satisfies the following specification:*

- **Function header.** `int bin_search(int A[], int n, int x)`
- **Pre-condition.** A *has size* n*, and for* $0 \leq i < n-1$*,* $A[i] \leq A[i+1]$
- **Function body.** *This should satisfy the following:*

$$\texttt{bin\_search}(A, n, x) = \begin{cases} i, & \text{where } A[i] = x \text{ and for all } j < i,\ A[j] \neq x \\ -1, & \text{there is no } i \text{ such that } 0 \leq i < n \text{ and } A[i] = x. \end{cases}$$

In other words, the function you will implement is similar to the binary search algorithm described in class and the readings, except that it must return the smallest *index* i *such that* $A[i] = x$*. For full credit, your implementation must have* $O(\lg n)$ *cost.*

Problem 4. *Write a function* `unimodal_search` *that satisfies the following specification:*

- **Function header.** `int unimodal_search(int A[], int x)`
- **Pre-condition.** A *has size* n*, and there is some* i *such that*

$$A[0] < A[1] < \cdots < A[i-1] < A[i] > A[i+1] > \cdots > A[n-1].$$

- **Function body.** *This should satisfy the following:*

  `unimodal_search`$(A, n) = i$, *where* $A[i] = \max(A[0], \ldots, A[n-1])$.

In other words, A consists of values that are strictly increasing up to some index i, and then strictly decreasing after that. `unimodal_search`(A, n) *will return i. Note that it could be that* $i = 0$ *or* $i = n - 1$. *There are at least a couple of ways of doing this. One is a fairly direct adaptation of binary search and has cost* $O(\log_2 n)$. *A cleverer implementation has cost* $O(\log_3 n)$.

3. Code distribution

This assignment comes with your first *code distribution*, comprising files that you will need to complete this assignment:

- `hw4.h`: header file for the code you will write. This file declares the functions that you must implement. Do not change the contents of this file; if it appears to be causing problems with compilation, the problem is with your solution.

- `hw4.c`: The only file you will submit for this homework. Function stubs matching hw4.h have been implemented for you to get you started.

- `comp211.h`: a header file that defines `dotest`, which is used in `tests.c`.

- `tests.c`: a small testing program. This program provides just a few tests. You should certainly add more. To compile the testing program, use the command

  ```
  gcc --std=c99 -o tests tests.c hw4.c
  ```

- `driver.c`: a small driver program. This program provides a simple interactive program that uses your `unimodal_search` function. You may modify it however you like. To compile the driver program, use the command

  ```
  gcc --std=c99 -o driver driver.c hw4.c
  ```

- `Makefile`: a Makefile for this assignment. `make` is a program for simplifying compilation of complex programs. From our perspective, its primary role is to shorten the command for compiling a program. Instead of using the above commands for compiling your code, you can use the commands `make tests` and `make driver`. `Makefile` is a file with instructions for `make` that say, in effect, to execute the compilation commands above when the corresponding `make` commands are given. We may not have covered the use of `make` by the time of this assignment, and you do not need to use it if you don't want to.

4. Submission

Submit your written work as `hw4.pdf` and your code as `hw4.c` to the Google Drive directory I have created for you named `comp211-f21-USERNAME/hw4/`. You should replace `USERNAME` with your Wesleyan username.

Do not forget that your written work must be submitted as a PDF! And make sure that at the top of each file you have put your name! Do not, however, change the names of the files.

You should *not* submit `hw4.h` or any test or driver programs. When we test your code, we will add in our copy of `hw4.h` and our own testing program. In particular, if you change `hw4.h` in order to make your code compile, then your code will probably fail to compile with our `hw4.h`, and hence you will receive little to no credit for the coding portion of this assignment.