Day 11 (Wed 02/16)

- exercise 10 review
- day 11 recap questions
- exercise 11

Announcements/reminders

- HW2 due Friday by 11pm
- HW3 due Friday 2/18 by 11pm
  - substantial assignment, start early!

slido.com

jhuintprog01 $\uparrow$ zero

Exercise 10 review

Important application of pointers: "pass by reference" semantics for normal variables

(Arrays are always passed by reference, but ordinary variables are passed by value by default.)

Exercise 10 review (continued)

Trace:

```
// getDate function
int getDate(int *m, int *d, int *y) {
  int month, day, year;
  int rc = scanf("%d/%d/%d", &month, &day, &year);
  *m = month;
  *d = day;
  *y = year;
  return rc;
}

// ... in main function ...

int day, mon, yr;

...

if (getDate(&day, &mon, &yr) == 3) {
  // use day, mon, yr
```

Exercise 10 review

Trace:

```
// even more concise version of getDate

int getDate(int *m, int *d, int *y) {
  return scanf("%d/%d/%d", m, d, y);
}

// ... in main function ...

int day, mon, yr;

...

if (getDate(&day, &mon, &yr) == 3) {
  // use day, mon, yr
```

Day 11 recap questions:

1. What is the difference between stack and heap memory?
2. What is dynamic memory allocation in C?
3. What is the memory leak problem?
4. What is the difference between malloc, realloc, and calloc?
5. What do we use valgrind to check for?
6. Consider the exclaim function below. Do you see any problems with this function?

1. What is the difference between stack and heap memory?

Stack memory: used for local variables, parameters, and other data required by an in-progress function call

- key characteristic: the _lifetime_ of local variables and parameters is limited to the duration of the function call for which they are allocated
- storage for local variables and parameters is allocated automatically in a _stack frame_ created when a function is called

Heap memory: chunks of memory can be allocated in a dedicated region of memory (the "heap")

- key characteristic: the lifetime of variables allocated in heap memory is under the control of the program (program decides when those variables are no longer needed)

2. What is dynamic memory allocation in C?

The program uses malloc (or calloc, or realloc) to dynamically create a chunk of memory of a specified size

Program can then use the chunk as a single variable, an array, etc.

Dynamically allocated memory can only be accessed using pointers

E.g.:

```
// dynamically allocate an array of 10 int elements
int *arr = (int *) malloc(10 * sizeof(int));
for (int i = 0; i < 10; i++) {
  arr[i] = (i + 1);
}
```

Dynamically allocated memory must be explicitly de-allocated by the program (when no longer needed) using the free function:

```
free(arr);
```

3. What is the memory leak problem?

If a program dynamically allocates memory but does not free it using free, it continues to exist in the heap.

The maximum amount of memory which can be allocated in the heap is finite, so a program which repeatedly dynamically allocates memory without freeing it could eventually exhaust the heap, causing subsequent allocation attempts to fail.

Programs must take care to de-allocate dynamically allocated memory after the last use.

4. What is the difference between malloc, realloc, and calloc?

malloc: dynamically allocate a block of memory of specified size

- Note: contents of the memory block are unspecified! (Similar to how the contents of local variables and arrays are also unspecified.)

calloc: like malloc, but contents are filled with zeroes (often used when allocating a dynamic array)

realloc: attempt to reallocate an existing chunk of memory to increase or decrease its size

- reallocation could be done "in place", or could involve allocating a new chunk of memory and copying the data from the original chunk

5. What do we use valgrind to check for?

valgrind can check for:

1. memory leaks (detected when the program exits)

2. memory errors, such as
   - out of bounds array accesses
   - use of an uninitialized value
   - accesses to heap memory not currently in use (i.e., dereferencing a pointer to a de-allocated block of dynamically allocated memory)

Testing your program regularly using valgrind is incredibly helpful!

- Just because your program "works" when you run it, doesn't mean that it is free from bugs

- The kind of bugs valgrind finds often lead to subtle data corruptions that can be difficult to track down by other means

Use it!

6. Consider the exclaim function below. Do you see any problems with this function?

```
// Return a C character string containing n exclamation points.
// n must be less than 20.
char* exclaim(int n) {
  char s[20];
  assert(n < 20);
  for (int i = 0; i < n; i++) {
    s[i] = '!';
  }
  s[n] = '\0';
  return s;
}
```