601.220 Intermediate Programming

Spring 2023, Day 10 (February 13)

Today's agenda

- Exercise 9 review
- Pointers
- Exercise 10

Reminders

- HW2 due Friday (Feb 17th)
  - Written assignment, no late submissions

Exercise 9 review

- Good first step in debugging a program: `break main`
  - This gives you control at the very beginning of `main`
- Use `next` (`n`) to advance to the next statement
- Use `step` (`s`) to step into a called function
  - Very important if a function is misbehaving

Exercise 9 review

To debug effectively, you need a *hypothesis* about what is going wrong.

For the `transpose` function, start with the observation that the `print` function doesn't print the entire contents of the destination array.

Use `print` (`p`) to inspect the values of variables, array elements, etc.

Exercise 9 review

Next issue: the `transpose` function doesn't seem to correctly transpose the elements in the original array.

Step into the call to `transpose`.

Inspect "shape" and contents of the two arrays:

```
print start[0]
print start[1]
print start[2]
```

Look carefully at the code at line 13 (do tthe array subscripts make sense?)

Exercise 9 review

Debuggers are not magic.

They will not tell you what's wrong with your code. . . because they have no idea what your code is supposed to do!

They are *very* useful for seeing what your code is actually doing: they help make the internal state of the program visible.

Pro tip: learn how to set breakpoints:

- `break` *functionName*
- `break` *sourceFileName*:*lineNumber*

Use the `continue` (`c`) command to run the program until the next breakpoint is reached.

Day 10 recap questions

1. What is a pointer?
2. If `a` is an int variable, and `p` is a variable whose type is *pointer-to-int*, how do you make `p` point to `a`?
3. If `p` is a *pointer-to-int* variable that points to an `int` variable `a`, how can you access the value of `a` or assign a value to `a` without directly referring to `a`? Show examples of printing the value of `a` and modifying the value of `a`, but without directly referring to `a`.
4. When calling `scanf`, why do you need to put a `&` symbol in front of a variable in which you want `scanf` to store an input value?
5. Trace the little program below and determine what the output will be.

1. What is a pointer?

A pointer represents the *address*, or in other words, the *location* of a variable.

With a pointer to a variable, you can *indirectly* access the variable, either to use the value stored in the variable, or to modify the value stored in the variable.

2. If `a` is an int variable, and `p` is a variable whose type is *pointer-to-int*, how do you make `p` point to `a`?

```
int a;
int *p;
p = &a;
```

`&` is the "address-of" operator. It gives you a pointer that points to the variable to which it is applied.

Visual representation:

3. If `p` is a *pointer-to-int* variable that points to an `int` variable `a`, how can you access the value of `a` or assign a value to `a` without directly referring to `a`? Show examples of printing the value of `a` and modifying the value of `a`, but without directly referring to `a`.

To indirectly access the variable a pointer is pointing to, use the `*` operator, known as the *dereference* operator.

How to think about the derefence operator: if `p` points to `a`, then `*p` means exactly the same thing as `a`.

Dereferencing a pointer

```c
// deref.c:
#include <stdio.h>
int main(void) {
  int a = 42;
  int *p;
  p = &a;
  printf("*p = %d\n", *p); // get a's value indirectly
  *p = 17;                 // modify a's value indirectly
  printf("after assigning to *p, a = %d\n", a);
  return 0;
}
```

```
$ gcc -std=c99 -Wall -Wextra -pedantic deref.c
$ ./a.out
*p = 42
after assigning to *p, a = 17
```

4. When calling `scanf`, why do you need to put a `&` symbol in front of a variable in which you want `scanf` to store an input value?

By using the address-of operator (`&`), you are passing a pointer to the variable in which you want `scanf` to store the input value. `scanf` uses this pointer to indirectly assign to the variable.

This is a very important use of pointers: to allow a function to *indirectly* refer to a variable that it can't refer to directly. This is a way of emulating *pass by reference*.

5. Trace the little program below and determine what the output will be.

The program:

```
int func(float ra[], float x, float *y) {
    ra[0] += 10;
    x *= 20;
    *y += 30;
    return 40;
}
int main() {
    float a = 1;
    float b = 2;
    float c[] = {3, 4, 5, 6};
    int d;
    d = func(c, a, &b);
    printf("%.2f, %.2f, %.2f, %d\n", a, b, c[0], d);
}
```

Exercise 10

- Implement a `getDate` function so that its parameters are pointers to month, day, and year variables
- Talk to us if you have questions!

Notes

Notes

Notes

Notes

Notes