601.220 Intermediate Programming

Summer 2022, Meeting 5 (June 14th)

Today's agenda

- Exercises 7 and 8 review
- "Day 9" material
  - Multidimensional arrays, gdb
  - Exercise 9
- "Day 10" material
  - Pointers
  - Exercise 10

Reminders

- HW1 due *Tomorrow*
- HW3 due Thursday, June 22nd
- After HW3 submission, you will be working in your first group project. You will be working in groups of 2 or 3 people.
- Projects will require a new github repository for the group. We will send google form on Friday to get the names in each group.

Exercise 7 review

Adding a function declaration (a.k.a. "function prototype") for the `div` function:

or
function signature

```
float div(float a, float b);
```

A function declaration makes the compiler aware of the name, parameter type(s), and return type of a function so that calls to the function can be checked for correct usage.

Exercise 7 review

`mult` function declaration:

```
float mult(float a, float b);
```

`mult` function definition:

```
float mult(float a, float b) {
  return a * b;
}
```

Exercise 7 review

`fac` declaration:

```
long fac(int a);
```

`fac` definition (observations: $0! = 1$, $n! = (n-1)! \times n$ when $n > 0$):

```
// Precondition: a >= 0
long fac(int a) {
  assert(a >= 0);
  if (a == 0) { return 1; }
  return fac(a - 1) * a;
}
```

Exercise 7 review

bsearch function:

```
int bsearch(float ra[], int low, int high, float target) {
  // base cases
  if (low > high)  { return -1; }
  if (low == high) { return (ra[low] == target) ? low : -1; }
  int mid = low + ((high-low)+1) / 2;
  if (ra[mid] == target) { return mid; }
  // ...recursive cases left as exercise for reader...
}
```

Exercise 7 review

`bsearch2`: The caller of `bsearch2` can't know how many values were added to the `results` array because the `size` parameter is passed by value.

Exercise 8 review

```
int concat(const char word1[], const char word2[],
    char result[], int result_capacity){
  int word1_len = strlen(word1);
  int word2_len = strlen(word2);
  if (word1_len + word2_len + 1 > result_capacity) {
    return 1; // not enough room in result array
  }
  int pos = 0;
  for (int i = 0; i < word1_len; i++) {
    result[pos] = word1[i];
    pos++;
  }
  for (int i = 0; i < word2_len; i++) {
    result[pos] = word2[i];
    pos++;
  }
  result[pos] = 0;
  return 0;
}
```

Step 1

Step 2

Exercise 8 review

`string_functions.h`:

```
#ifndef STRING_FUNCTIONS_H
#define STRING_FUNCTIONS_H

int concat(const char word1[], const char word2[],
   char result[], int result_capacity);

#endif // STRING_FUNCTIONS_H
```

] Header guards

Exercise 8 review

`string_functions.c`:

```c
#include <string.h>
#include "string_functions.h"

int concat(const char word1[], const char word2[],
    char result[], int result_capacity){
  // ...code omitted...
}
```

Exercise 8 review

run_concat.c:

```
#include <stdio.h>
#include <string.h>
#include "string_functions.h"

int main() {
  // ...code omitted...
}
```

dir:

- string_functions.h
- string_functions.c
- run_concat.c

Exercise 8 review

dir:

- string_functions.h
- string_functions.c
- run_concat.c

```
# Makefile
CC = gcc
CFLAGS = -std=c99 -pedantic -Wall -Wextra

run_concat: run_concat.o string_functions.o
    $(CC) -o run_concat run_concat.o string_functions.o

run_concat.o: run_concat.c string_functions.h
    $(CC) $(CFLAGS) -c run_concat.c

string_functions.o: string_functions.c string_functions.h
    $(CC) $(CFLAGS) -c string_functions.c

clean:
    rm -f *.o run_concat
```

Day 9 recap questions

1. How do you declare a multi-dimensional array and pass it to a function?
2. How do you initialize a multi-dimensional array using array initialization?
3. What is the compile flag needed to compile a program such that we can debug it using gdb?
4. How do you set a break point using gdb and check the call stack?
5. Check the gdb cheat sheet and find the command to print the content of a variable per step, instead of only printing it once using `print`.

1. How do you declare a multi-dimensional array and pass it to a function?

Declaring a two-dimensional array:

```
char board[3][3];
```

Accessing an element:

```
board[0][2] = 'X';
```

Note that by convention, the first index is “rows” and the second index is “columns”.

2-D array as parameter

```
void print_board(char board[3][3]) {
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
      printf("%c", board[i][j]);
    }
    printf("\n");
  }
}
```

Note that the first dimension can be omitted, but the other dimensions are required.

2. How do you initialize a multi-dimensional array using array initialization?

Example:

```
char board[3][3] = {
  {'O', 'X', 'X'},
  {'X', 'O', 'O'},
  {'X', 'X', 'O'},
};
```

3. What is the compile flag needed to compile a program such that we can debug it using gdb?

The `-g` option causes the compiler to generate debug information.

Strongly recommended for all `Makefiles` for C programs:

```
CFLAGS = -g -std=c99 -pedantic -Wall -Wextra
```

4. How do you set a break point using gdb and check the call stack?

Set breakpoint at beginning of function:

```
break main
break bsearch
```

Set breakpoint at specific source line:

```
break functions.c:74
```

Print call stack (all of these are equivalent):

```
where
backtrace
bt
```

5. Check the gdb cheat sheet and find the command to print the content of a variable per step, instead of only printing it once using `print`.

```
display
```

Exercise 9

- Two-dimensional arrays
- Debugging using `gdb`
- Breakout rooms 1–10 are “social”
- Use Slack to let us know if you have questions

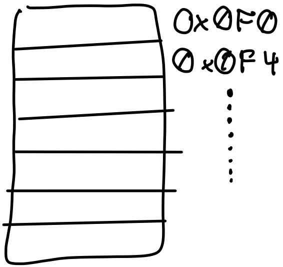
Day 10 recap questions

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.



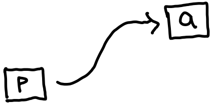
2. If a is an int variable, and p is a variable whose type is *pointer-to-int*, how do you make p point to a?

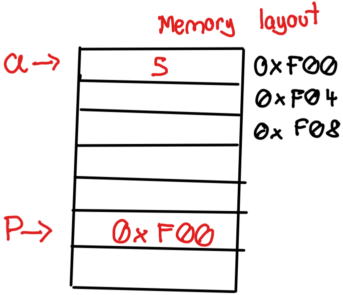
```
int a;
int *p;
p = &a;
```

printf (" % d", *p);

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

Visual representation:





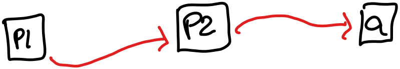
* dereference operator.

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



*P1 ⇒ address stored in P2

```
// deref.c:
#include <stdio.h>
int main(void) {
  int a = 42;
  int *p;
  p = &a; → store address of a in p
  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*.

```
int x;
scanf("%d", &x);
```

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

*Very important

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);
}
```

C[0] ⇝ 13

b ⇒ 32

a ⇝ 1

Exercise 10

- Implement a `getDate` function so that its parameters are pointers to month, day, and year variables
- Breakout rooms 1–10 are "social"
- Use Slack to let us know if you have questions