CMSC 216
Introduction to Computer Systems
Lecture
Pointers Intro
Administrivia

• Read Reek, Chapter 6: Pointers
Chapter 6, Reek

POINTERS
Pointers

• Pointers are variables whose value is an address

• Summary
  int x = 5;
  int *ptr;     /* pointer variable */
  ptr = &x;     /* getting address of x */
  *ptr = 10;    /* equivalent to x = 10 */

• Let’s draw a diagram for the above
• Every variable is stored at an address in memory
• We use pointers to perform manipulation of memory, by accessing items at the address stored in the pointer
Declaration of pointers

• A pointer to an int value would be declared like this: `int *ip;`

• Creates a variable called ip, whose type is "pointer to int"

• We can assign the address of an int variable to be the value of this new pointer
Pointer operators

• Obtaining the address of an object (&)
  – Placed before a variable (or an object in memory)

• Accessing the value at an address (*)
  – Placed before an expression which is either a pointer or otherwise evaluates to an address

• Example:

```c
int i = 6;
int *p;
p = &i;
printf("%d %d\n", *p, *(&i));
```

<table>
<thead>
<tr>
<th>Memory</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>6</td>
</tr>
<tr>
<td>p</td>
<td>1084</td>
</tr>
</tbody>
</table>
Using a dereferenced pointer

• The * operator can be used on both the left and right sides of an assignment

```c
int i = 6;
int j;
int *p;
p = &i;
j = *p;
printf("%d %d\n", i, j);
*p = 4;
printf("%d %d\n", i, j);
```
Garbage pointers

• When a pointer is declared, it points to whatever address was in the memory location allocated for the pointer (no initialization)

• Trying to dereference this random address will generally result in one of three Bad Things:
  – accessing a memory location you don't have permission to access (a "segmentation fault")
  – violating the computer's alignment policies (a "bus error")
  – silent failure: everything appears to work right... for now
Multiple uses for *

- as multiplication operator
- * to declare a variable as a pointer variable
  ```c
  int *ptr;  /* we are not dereferencing here */
  ```
- to dereference
  ```c
  int x = 5;
  int *ptr = &x;
  *ptr = 10;
  ```
NULL pointer

- This is a pointer that points to the address 0, where nothing is allowed to be accessed
- Defined in `stddef.h`, which is included by many other header files
- Analogue to Java's `null`
  - What happens when you try to call a method of an object which is null?
  - Very similar thing happens in C when trying to dereference a NULL pointer; it's usually a segfault
- Just like in Java, you have to check pointers to see if they're NULL before dereferencing them:

  ```c
  void f(int *p) {
    if (p != NULL)
      *p = 55;
  }
  ```
Pointers to Pointers

- You can also obtain the address of a pointer variable:

  ```c
  int i = 4;
  int j = 6;
  int *p = &i;
  int *q = &j;
  int **r = &p;
  printf("%d\n", **r);
  *r = &j;
  printf("%d\n", *p);
  ```

- Let’s add some arrows to the memory map
- This technique will be useful when working with pointers as parameters
Pointers as parameters

- You can also pass addresses into a function:
  ```
  void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
  }
  ```

  ```
  .
  int x = 2;
  int y = 3;
  swap(&x, &y);
  printf("%d %d\n", x, y);
  ```

- Why do we need to use pointers here?
- Let’s draw a memory map for the above
- What would happen if after *b = tmp we set a and b to null?