Announcements

- **Project #2**
  - Available on the web

- **Reading**
  - Chapter 6 (today)
  - Chapter 11 (Thursday)

---

Pointers Defined

- **Can declare a pointer to any type of variable**
  - Are similar to references in Java
  - Use * before variable name to create a pointer to the type
    - int *foo;
    - myTypeDef *myPtr;
    - float *bar;
  - The type determines both:
    - What is pointed to
    - The size of the object being pointed to

- **Initialization**
  - Creating a pointer doesn't create the space it points to
  - int *foo; /* These two lines will often cause an error here */
    *foo = 3; /* Because you haven't specified what space it points to */
  - int *bar = 3; /* Won't cause an error here but likely will later */
  - int a[5], b; /* These two lines are the way initialization */
    int *p = a, *q = &b; /* Should be done */
Pointers

- **Mixing types in declaration:**
  - int *p, q;
  - declares p as a pointer to an integer and q as an integer

- **Can declare a pointer to a pointer**
  - int **p; /* p pointer to a variable that points to an int */
  - useful if you want to modify a pointer in a subroutine

- **Typedefs of pointers improve readability**
  - typedef int *intPtr;
  - intPtr p, q;

- **Function return type can be a pointer**
  - Be careful of Dangling Pointers
  ```c
  int *foo() {
    int x = 42;
    return (&x);
  }
  ```

Other Pointer Declarations

- **Can declare a pointer to a pointer**
  - int **p; /* p pointer to a variable that points to an int */
  - useful if you want to modify a pointer in a subroutine
  - see pntrpntr.c

- **What declaration:**
  - int *p, q;
  - declares p as a pointer to an integer and q as an integer

- **Typedefs of pointers improve readability**
  - typedef int *intPtr;
  - intPtr p, q;
Pointer Arithmetic

- **Pointers are first class types**
  - arithmetic applies to pointers just like ints.
  - This is useful, but can get tricky
- **Adding pointer and int**
  - Adds n objects to value pointed to.
    - If objects are 100 bytes, p+1 is 100 bytes later in memory
  - Example:
    - int x = 5;
    - int a[10];
    - int *p = a;
    - *(p + 3) = 42; /* updates a[3] to 42 */
  - [] is really shorthand for pointer addition
    - a[n] is really the same as *(a + n)
- **Subtraction**
  - pointer and int
    - p - n a pointer to the nth element before p
  - pointer and pointer
    - p - q number of items between p and q

Relational Operations on Pointers

- `<`, `<=`, `>`, `>=`, `==`, `!=`
  - all compare two pointers of the same type
  - compares the memory location pointed to
- **Example:**
  - int a[] = {6, 2, 0, 1, 8}, b = 5;
  - int *p, *q, *r;
  - p = &a[1]; q = a[3]; r = &b;
  - if (p < q) {....}
  - if (*p < *q) {....}
  - if (p < r) {....}
  - if (*p < *r) {....}
  - if (*(p-1) < *r) {....}
Type Conversion

- **C lets you assign variables of different types**
  - called type conversion
  - useful when dealing with
    - external devices
      - `int *fooDev = (int *) 100;`  
      - we know the foo device is at location 100
    - when type information read from a file is based on file contents.
- **WARNING: This can be very dangerous**
- **(type *) variable:**
  - `int *x;`
  - `float *y;`
  - `x = (int *) y;`

Generic Pointers

- **void ***
  - Declare a pointer that can point to any type
  - Any use are assignment requires a cast:
    - `void *p;`
    - `int *x;`
    - `int *y;`
    - `p = (void *) x;`
    - `y = (int *) p;`
  - Allows writing generic code
    - Need to ensure only appropriate things happen:
      - `int *x;`
      - `float *y;`
      - `void *p;`
      - `p = (void *) x;`
      - `y = (float *) p;`
NULL Pointer

- **NULL is a pointer to nothing**
  - usually integer value 0
  - (void *) is its type
  - defined in stdlib.h
- **Can be used to initialize pointers**
  - int *p;
  - p = NULL;
- **Can be used in comparison**
  - if (p != NULL) {
    ....
  }
- **Often returned by functions to indicate error or completion**

Const Keyword and Pointers

- **const int *p;**
  - pointer to an integer that doesn't change:
    - const x = 42;
    - p = &x; /* legal */
    - *p = 20; /* illegal - changing the constant */
- **int *const p;**
  - constant pointer to an integer:
    - int *const p = &foo;
    - *p = 42; /* ok: just changing the value */
    - p = &bar; /* illegal: changing the pointer */
Pointers and Functions

• Parameters
  – can use const to prevent functions from changing values

    int frob(const char *data, int size) {
      if (data[0] == 'Q') { .. } /* OK: just using the data */
      data[2] = 'A'; /* Error: can't modify data */
    }

• Functions can return pointer types

    int *getStuff(int *a, int *b) {
      if (*a > *b) {
        return a;
      } else {
        return b;
      }
    }

    int *foo0() {
      int *ptr;
      ptr = calloc(sizeof(int), 42);
      return ptr;
    }

    int *foo1() {
      int x = 42;
      return (&x);
    }

    int *foo2() {
      int x = 42;
      int *p = &x;
      return (p);
    }

    int *foo3(int *a, int *b) {
      if (*a > *b) {
        return a;
      } else {
        return b;
      }
    }

    int *foo4(int a, int b) {
      if (a > b) {
        return &a;
      } else {
        return &b;
      }
    }