

Announcements

- Program #2
 - Available on the web
- Midterm #1 is today
 - 6:00 PM (in ARM 0131)
- Reading
 - Chapter 6 (today)
 - Chapter 11 (Thursday)

Pointers

- Can declare any variable to be of type pointer
 - 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;`
 - `*foo = 3;` /* error: haven't defined what it points to */

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;`

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
- 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 Assignments

- Pointers a normal variables
 - they are not initialized unless assigned
- Sample Assignments:
 - `int *p, *q, i1, i2;`
 - `p = &i1; /* p now points to i1 */`
 - `q = p; /* q now points to i1 too */`
 - `p = &i2; /* p now points to i2 */`
 - `*p = *q; /* the value of i1 is copied to i2 */`

Generic Pointers

- `void *`

- Declare a pointer that can point to any type
- Any use or 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
 - (void *) 0 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

Pointer Examples

```
size_t strlen(char *str)
{
    size_t length;

    for (length=0; *str != '\0'; str++) {
        length++;
    }
    return length;
}
```

Array of Strings Example

```
int findIt(char **table, char *it)
{
    int count = 0;
    while (*table) {
        if (!strcmp(*table, it)) {
            return count;
        }
        table++;
        count++;
    }
    return -1;
}
```

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 */`

Pointer Aliases

- Sometimes two pointers point to the same thing\

- This can be useful
- This can cause problems:

```
int *p, *q;
```

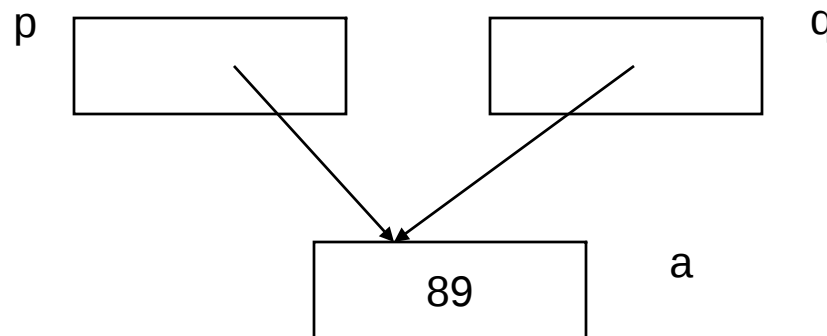
```
int a= 89;
```

```
p = &a;
```

```
q = p;
```

```
free(p);
```

```
if (*q) { ... }      /* error: *q points to freed space */
```



Other Pointer Operations

- Subtraction

- pointer and int
 - $p - n$ a pointer to the n th element before p
- pointer and pointer
 - $p - q$ number of items between p and q

- Relational operations:

- $<$, $<=$, $>$, $>=$, $==$, $!=$
 - all compare two pointers of the same type
 - compares the memory location pointed to

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

Dangling Reference

- Problem: Stack variables go out of scope on return
- Consider:

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