C Advice

- Draw stuff out
  - Variables are boxes, pointers are arrows
- Give a type your variables!
- & returns a value whose type has one more star than the type of the variable
  - int quux; int* baz = &quux;
- Execute the fundamental operations one at a time
  - variable lookup, pointer deference, etc

Tracing Pointers – Warm Up

What will \( y \) contain?

```c
int main(int argc, char** argv) {
    int y, *z;
    y = 4;
    z = &y;
    y = *z + 9;
    return 0;
}
```

Tracing Pointers – Warm Up

What will \( y \) contain?

```c
int main(int argc, char** argv) {
    int y, *z;
    y = 4;
    z = &y;
    y = *z + 9;
    return 0;
}
```
Tracing Pointers – Warm Up

What will y contain?

```c
int main(int argc, char** argv)
{
    int y, *z;
    y = 4;
    z = &y;
    y = *z + 9;
    return 0;
}
```

It contains 0xD. What is that in binary? In decimal?

---

Tracing Pointers – More Levels

What is in foo and bar at the end of this program?

```c
int main(int argc, char** argv)
{
    int foo, *bar, **baz, quux;
    bar = &quux;
    foo = 4;
    baz = &bar;
    **baz = 13;
    bar = &foo;
    **baz = 9;
    return 0;
}
```

---

Tracing Pointers – More Levels

What is in foo and quux at the end of this program?

```c
int main(int argc, char** argv)
{
    int foo, *bar, **baz, quux;
    bar = &quux;
    foo = 4;
    bar = &foo;
    **baz = 9;
    return 0;
}
```
Tracing Pointers – More Levels

What is in `foo` and `quux` at the end of this program?

```c
int main(int argc, char** argv)
{
    int foo, *bar, **baz, quux;
    bar = &quux;
    foo = 4;
    bar = &foo;
    **baz = 9;
    return 0;
}
```

What’s wrong with this program?

```c
int modifyCount(int x)
{
    x = x - 1;
}

int main(int argc, char** argv)
{
    int x = 4;
    /* want to change x */
    modifyCount(x);
    return 0;
}
```

What’s wrong with this program?

```c
int modifyCount(int x)
{
    x = x - 1;
}

int main(int argc, char** argv)
{
    int x = 4;
    /* want to change x */
    modifyCount(x);
    return 0;
}
```

What’s wrong with this program?

```c
int modifyCount(int x)
{
    x = x - 1;
}

int main(int argc, char** argv)
{
    int x = 4;
    /* want to change x */
    modifyCount(x);
    return 0;
}
```

We never changed `x`! How do we fix this?
Use Pointers!

```c
int modifyCount(int* x)
{
    *x = *x - 1;
}

int main(int argc, char** argv)
{
    int x = 4;
    /* want to change x */
    modifyCount(&x);
    return 0;
}
```

What’s wrong with this program?

```c
int modifyCount(int* x)
{
    *x = *x - 1;
}

int main(int argc, char** argv)
{
    int x = 4;
    /* want to change x */
    modifyCount(&x);
    return 0;
}
```

Pointers and `++/--`

Suppose we have the following program:

```c
int main(int argc, char** argv)
{
    int i, j;
    int* p = &argc; /* argc = 1 */
    i = (*p)++;
    argc = 1;
    j = ++(*p);
    return 0;
}
```

What is in `i` and `j`?

• `i = 1` and `j = 2`

Pointers and `++/--`

Assuming `x` and `y` have type `int`...

• `y = x++;` is equivalent to `y=x; x=x+1;`
• `y = ++x;` is equivalent to `x=x+1; y=x;`

Pointers and `[]`

• `x[i]` can always be rewritten as `*(x+i)` and vice versa

• Array types can often be converted into their corresponding pointer counterparts
  - `int foo[]` is equivalent to `int* foo`
  - `int* bar[]` is equivalent to `int** bar`

• You can at most change one set of `[]` safely
  - Changing more requires knowing how the array looks in memory

• `x[i]` can always be rewritten as `*(x+i)` and vice versa

• Array types can often be converted into their corresponding pointer counterparts
  - `int foo[]` is equivalent to `int* foo`
  - `int* bar[]` is equivalent to `int** bar`

• You can at most change one set of `[]` safely
  - Changing more requires knowing how the array looks in memory
Pointers and ++/--

Suppose we have the following program:

```c
int main(int argc, char** argv) {
    int i, j;
    int* p = &argc; /* argc = 1 */
    i = (*p)++;
    argc = 0;
    j = ++(*p);
    return 0;
}
```

What is in `i` and `j`? **Both contain 1**

Printf, scanf, and their cousins

- printf (and its cousins) are special functions that do not have a fixed argument list
  - for each format specifier (i.e. %d), an additional argument needs to be supplied

- Examples:
  - printf("%d", 4);
  - printf("%s%d%c", "CS", 0x3D, ‘c’);

C Program Walkthrough

What happens with this program?

```c
void quux(int foo)
{
    char a[4];
    char* baz = (char*)(&foo);
    printf("%c%c%c%c",
           baz[0], *(baz + 1), baz[1+1],
           baz[sprintf(a, "123")]);
}

int main(...) {
    quux(0x4d495053);
}
```

0x4d495053

```
int foo
```
C Program Walkthrough

What happens with this program?

```c
void quux(int foo) {
    char a[4];
    char* baz = (char*)(&foo);
    printf("%c%c%c%c", baz[0], *(baz + 1), baz[1+1], baz[sprintf(a, "123")]);
}

int main() {
    quux(0x4d495053);
}
```

It will print out “MIPS”