CSCB09: Software Tools and Systems Programming

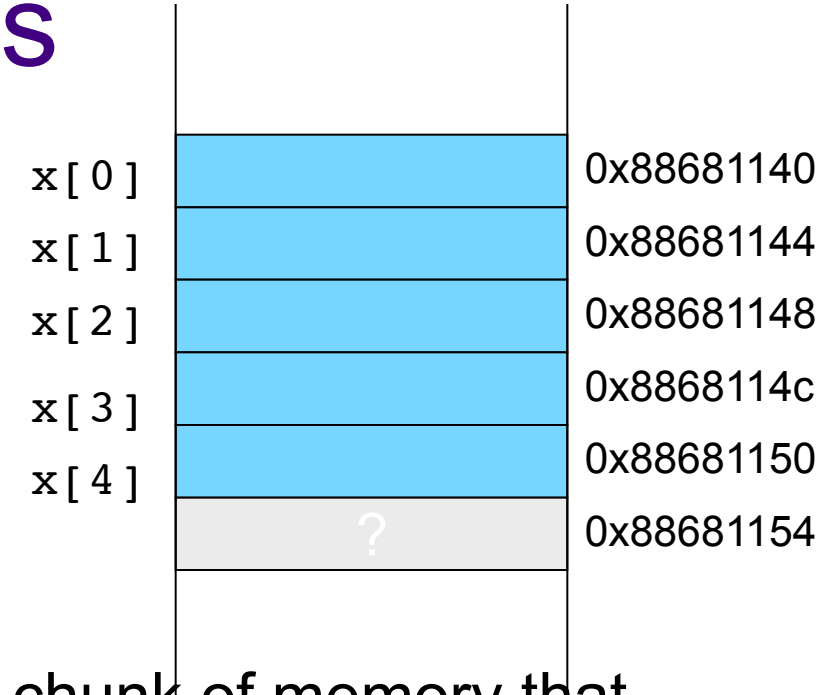
Bianca Schroeder
bianca@cs.toronto.edu
IC 460

Tips for A1

- Writing code that follows specifications is a big part of software development.
  - Make sure
    - to test on different input images.
    - can handle any valid ppm image as input.
    - the only output produced is a valid ppm file.
    - program never hangs when run on valid ppm image.
  - Solutions that don’t follow specifications might fail automarker tests, resulting in zero points.

Arrays

```
int x[5];
for (i = 0; i < 5; i++) {
    x[i] = i*i;
}
```



- Arrays in C are a contiguous chunk of memory that contain a list of items of the same type.
- If an array of ints contains 10 ints, then the array is 40 bytes. There is nothing extra.
- In particular, the size of the array is not stored with the array. There is *no* runtime checking.
- So you can access `x[99999999]`
  - What does that mean?

Pointer Arithmetic

- The array access operator [ ] is really only a shorthand for pointer arithmetic + dereference
- These are equivalent in C:

```
a[i]  ==  *(a + i)
```

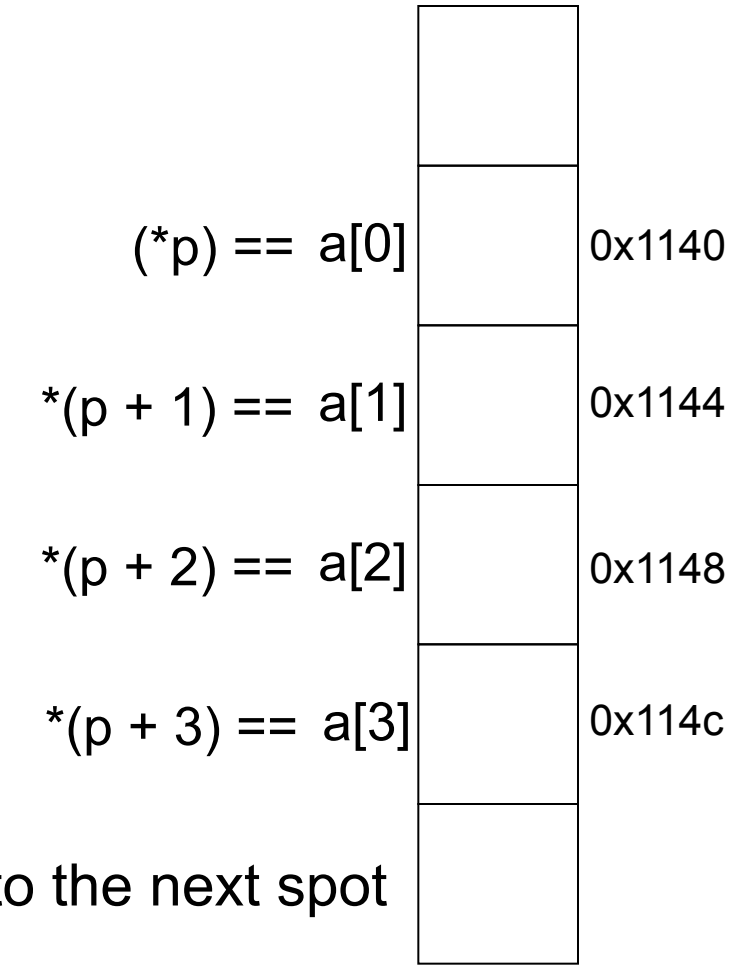
- The compiler resolves the name of an array to the starting address of the array and adds to it.
- So for `a[9999999]`, the program will happily try to access contents at address `*(a+9999999)`

- Behaviour of exceeding array bounds is "undefined"
  - program might appear to work
  - program might crash (segmentation fault)
  - program might do something apparently random

Why do pointers need a type, e.g. `int*` or `char*`?

```
int i = 0;
int a[4] = {0, 1, 2, 3};
int *p;
p = a;

for(i = 0; i < 4; i++) {
  printf("%d\n", *(p + i));
}
```



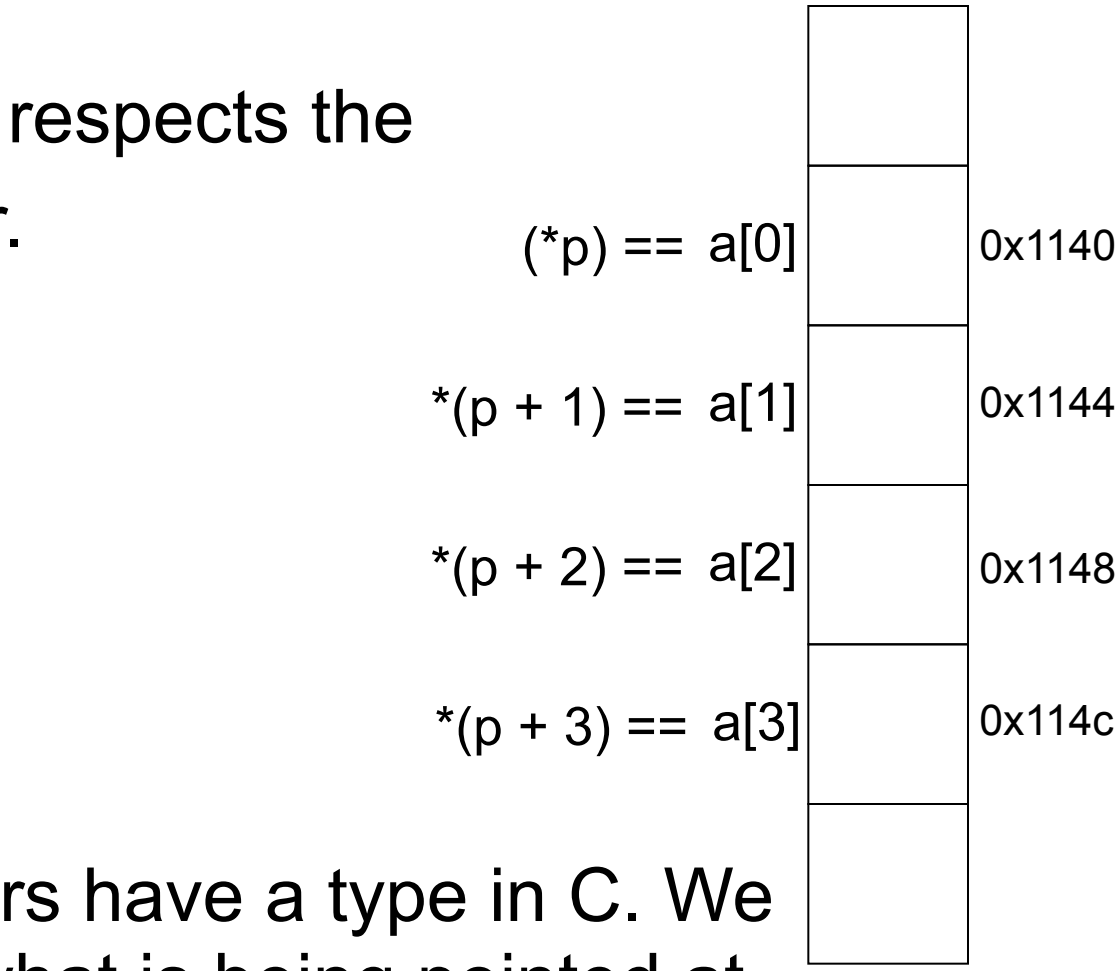
- Hint: Why does adding 1 to `p` move it to the next spot for an int, assuming an int is 4 bytes?

Why do pointers need a type, e.g. `int*` or `char*`?

- Pointer arithmetic respects the type of the pointer.
- E.g.,

```
int *p;
*(p+1) …;
```

really adds 4 to `p`



- This is why pointers have a type in C. We know the size of what is being pointed at from the *type* of the pointer.

Looking at Questions 4-9 from the arrays worksheet

```
#include <stdio.h>

int main() {
    int ages[4] = {5,7,18,20};
    int i;

    for(i=0; i<4; i++){
        ages[i] += 1;
    }

    for(i=0; i<4; i++){
        printf("The element at index %d now has the
                value %d\n", i, ages[i]);
    }

    return 0;
}
```

Before we look at the Command-line Worksheet, a few words about strings..

Strings

- Strings are not a built-in data type
- A string is an array of chars terminated by null character (‘\0’).

- Initializing a string:

```
char course_name[8] = {‘c’,’s’,’c’,’b’,’0’,’9’,’h’,’\0’};
```

Or more conveniently:

```
char course_name[8] = “cscb09h”;
```

Now you can do all the usual array operations, e.g.

```
course_name[3] = ‘z’;
```

- Other ways to initialize the string:

```
char course_name[80] = “cscb09h”;
char course_name[2] = “cscb09h”;      // OUCH!!!
```

Strings

After initializing a string:

```
char course_name[8] = {'c','s','c','b','0','9','h','\0'};
```

What is the type of `course_name`?

```
char *
```

What type would an array of strings be?

```
char **
```

So what is `argv` in `int main(int argc, char **argv)`?

An array of strings.

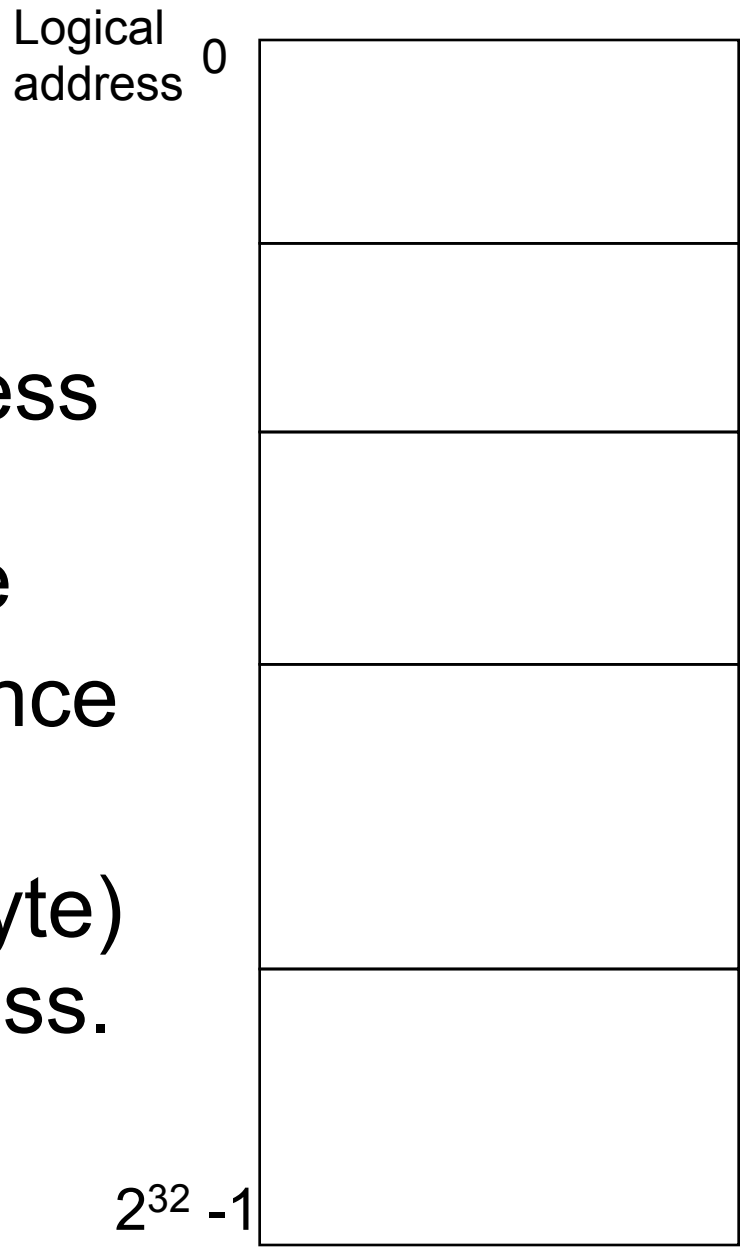
Now try to work through Q2-5 on the command-line argument sheet!

Questions about pointers before we start with Q 1 of the Pointer Worksheet?

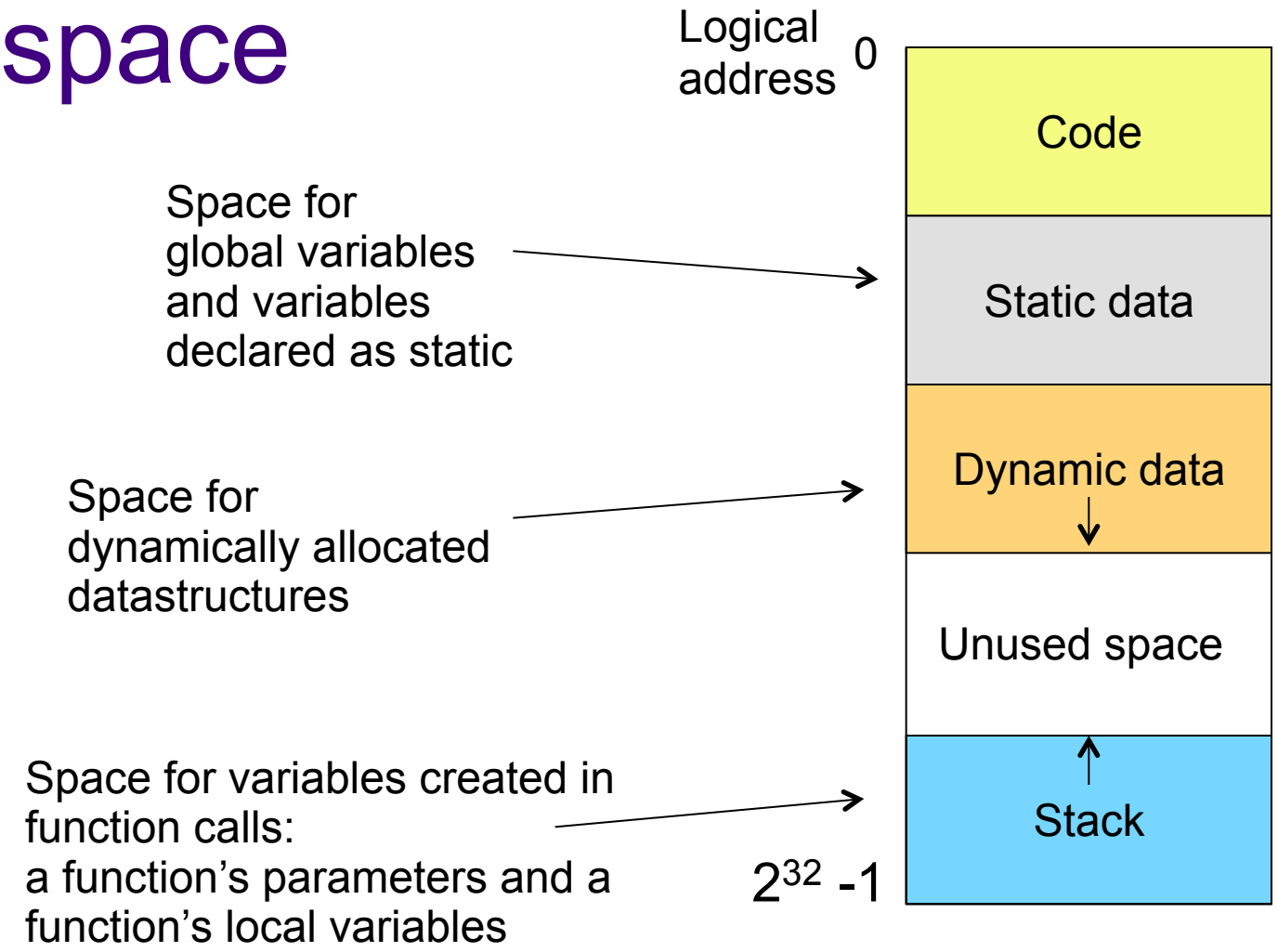
Let’s see why lie() on the worksheet did not work as expected…

Memory model

- The memory for a process (a running program) is called its address space
- Memory is just a sequence of bytes
- A memory location (a byte) is identified by an address.



The address space



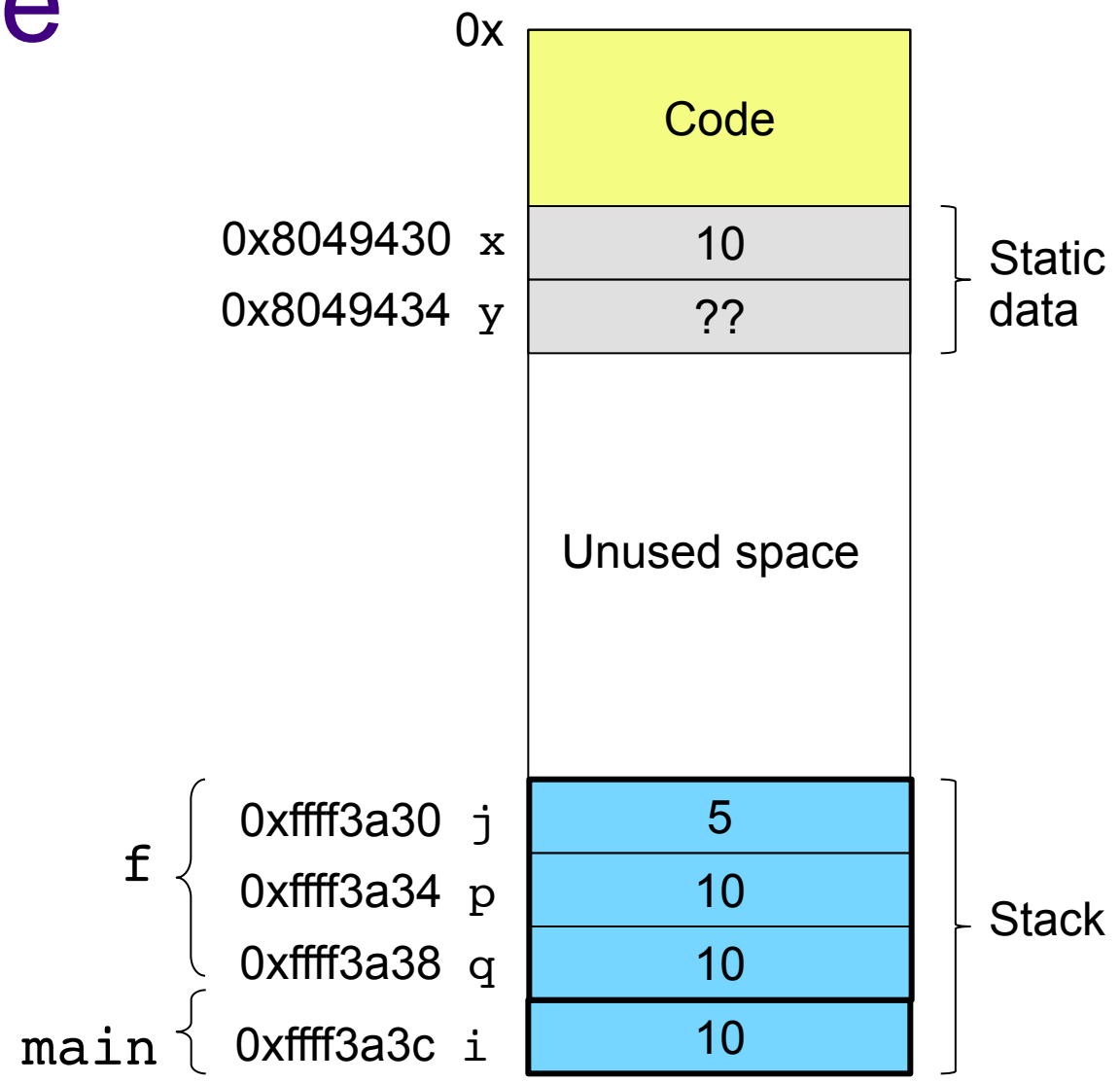
An example

```
int x = 10;
int y;

int f(int p, int q) {
    int j = 5;
    return p * q + j;
}

int main() {

    int i = x;
    y = f(i, i);
    return 0;
}
```



Passing function arguments

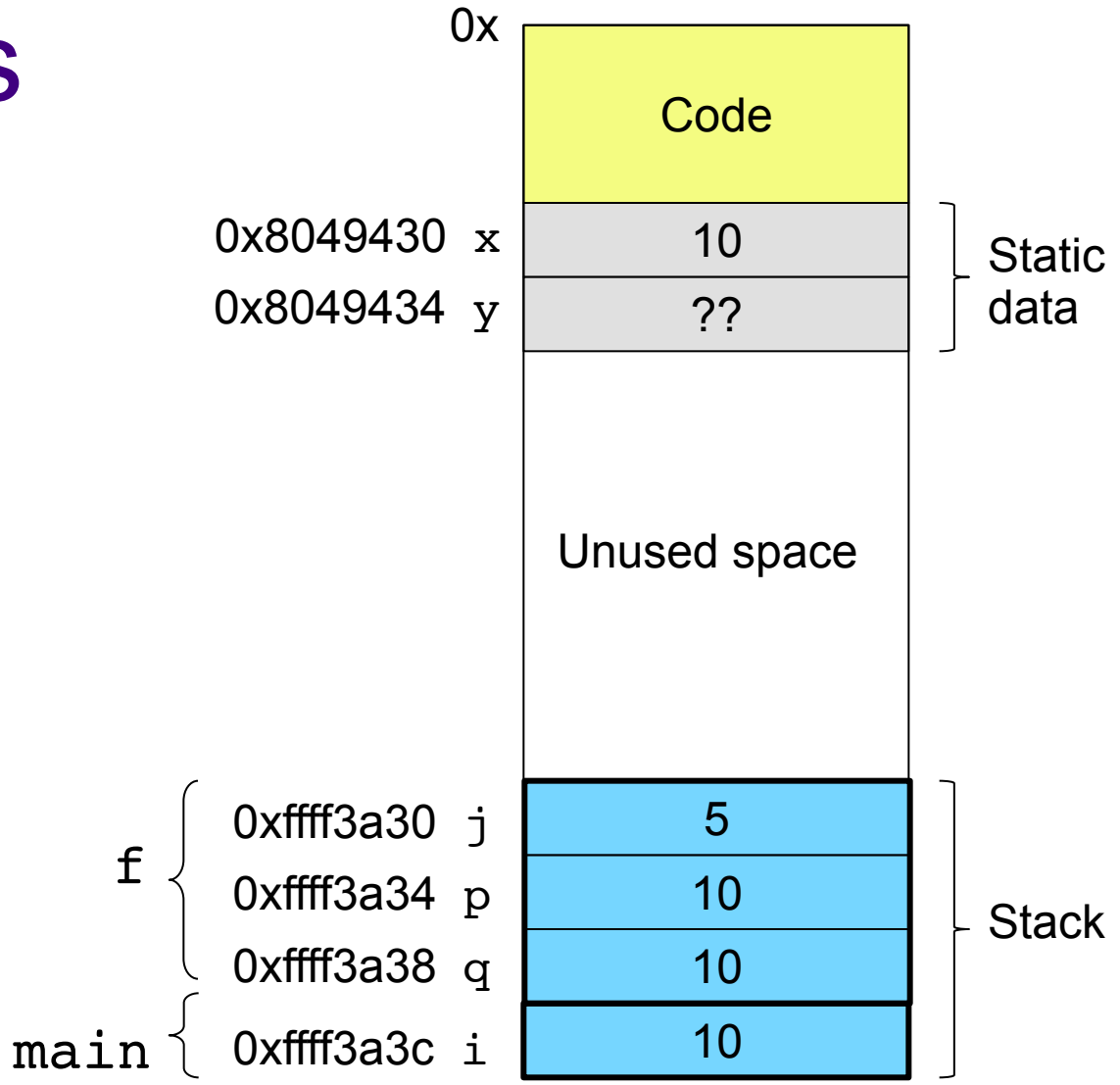
```
int x = 10;
int y;

int f(int p, int q) {
    int j = 5;
    p = 5;
    return p * q + j;
}

int main() {

    int i = x;
    y = f(i, i);
    return 0;
}
```

If f() were to modify p or q, will that change the value of main's `int i`?



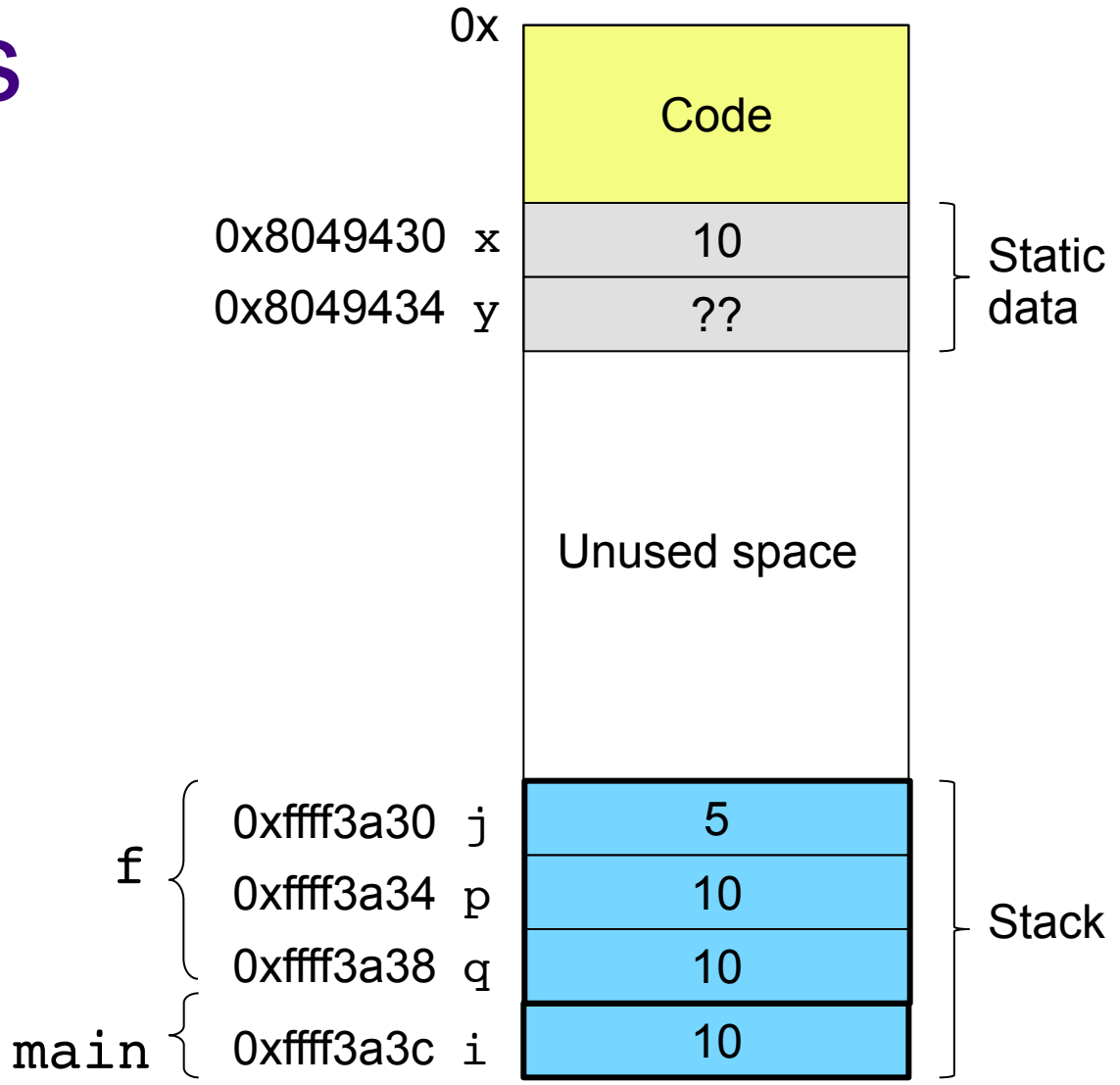
Passing function arguments

```
int x = 10;
int y;

int f(int p, int q) {
    int j = 5;
    x = 5;
    return p * q + j;
}

int main() {

    int i = x;
    y = f(i, i);
    return 0;
}
```



If `f()` were to modify `x` or `y` would this change be permanent?

Passing function arguments

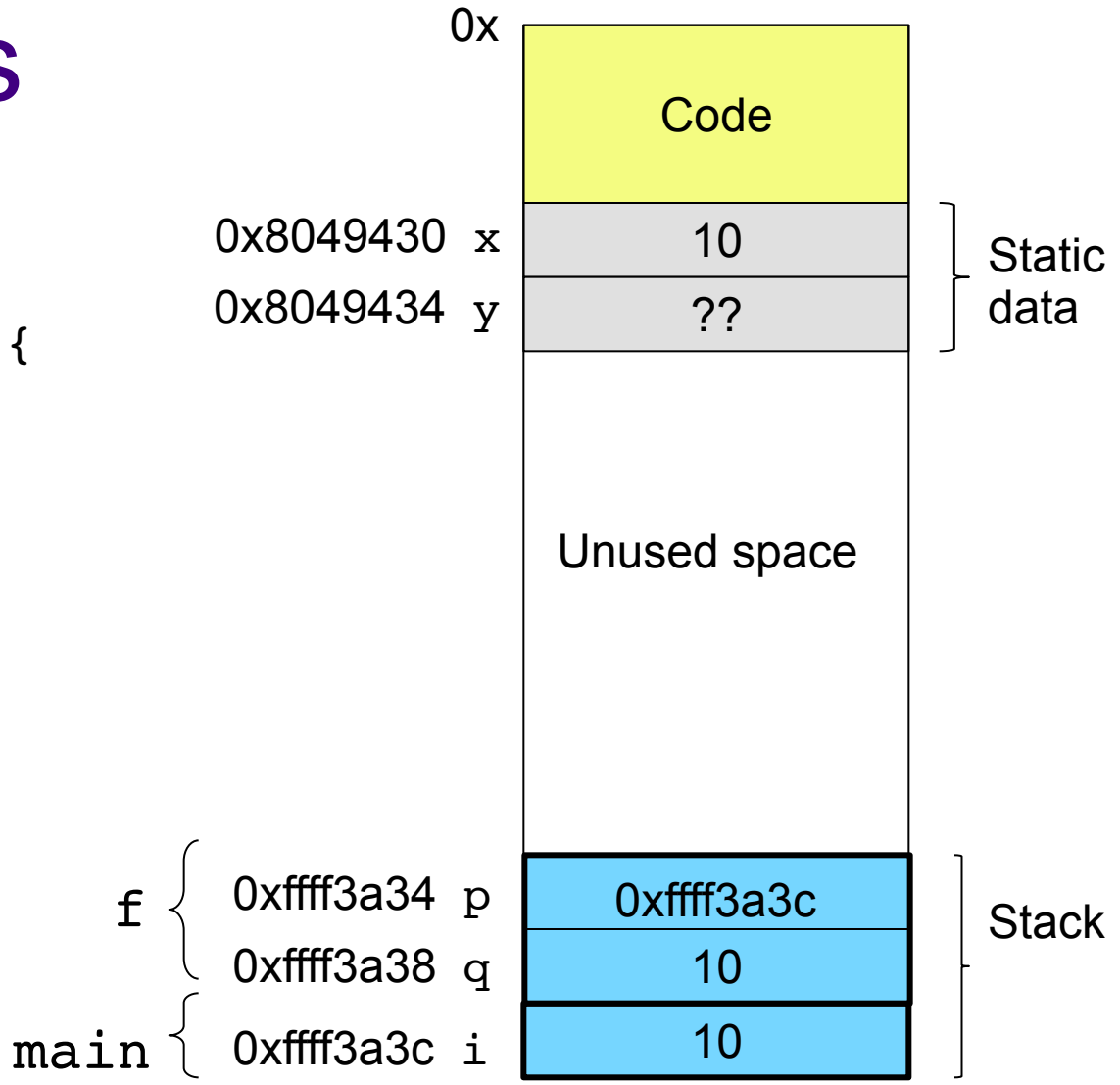
```
int x = 10;
int y;

void f(int *p, int q) {
    *p = 5;
}

int main() {

    int i = x;
    f(&i, i);
    return 0;
}
```

When calling f():



Passing function arguments

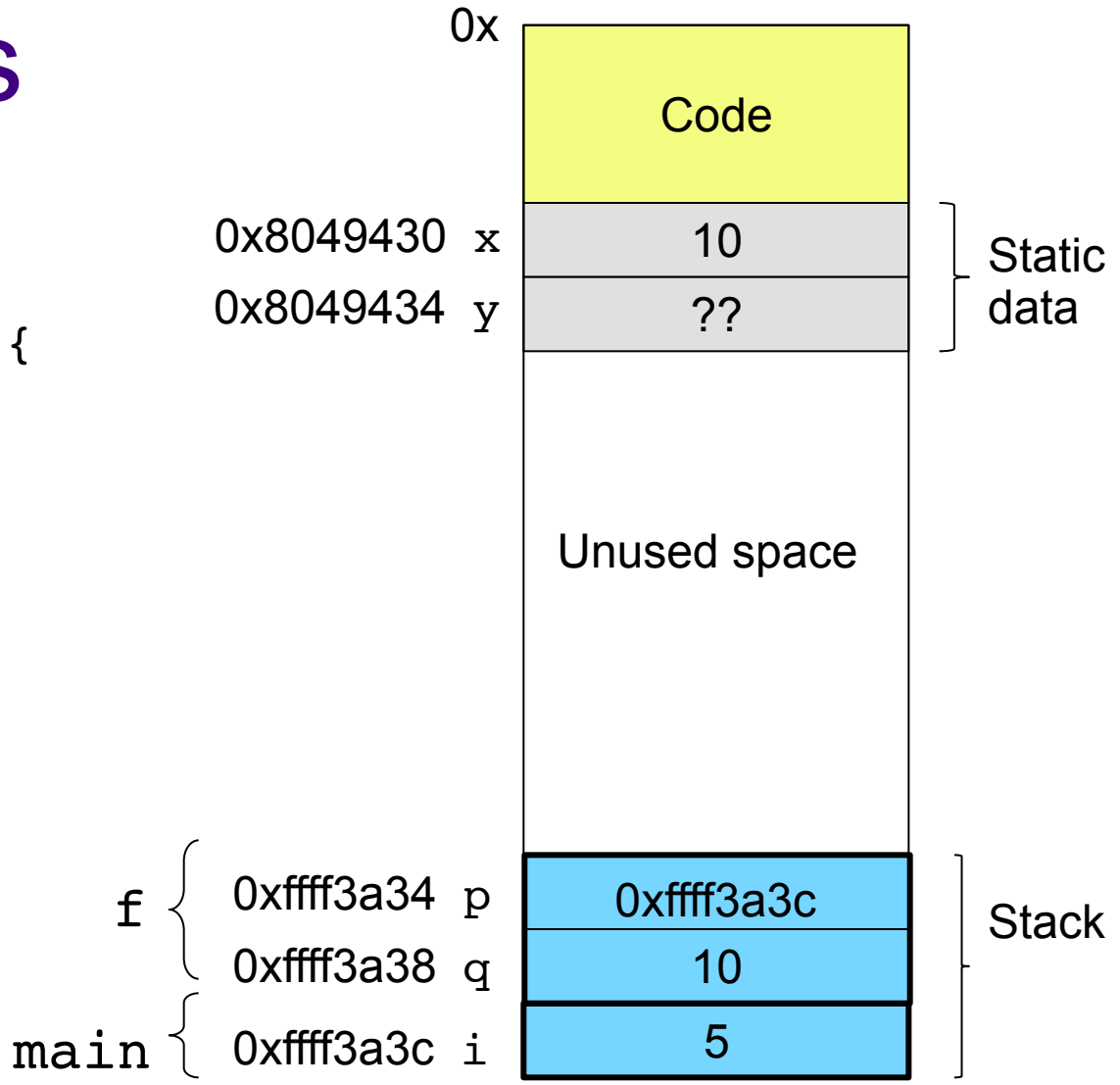
```
int x = 10;
int y;

void f(int *p, int q) {
    *p = 5;
}

int main() {

    int i = x;
    f(&i, i);
    return 0;
}
```

After f() returns:



With this in mind try to work through the remainder of the Pointer Worksheet!

That’s it for today!