What do we need to make the String class?

Structs

```
struct Point {
    int x;
    int y;
};
```

struct def write at toplevel

```
typedef struct Point Point;
```

abbreviation

type name

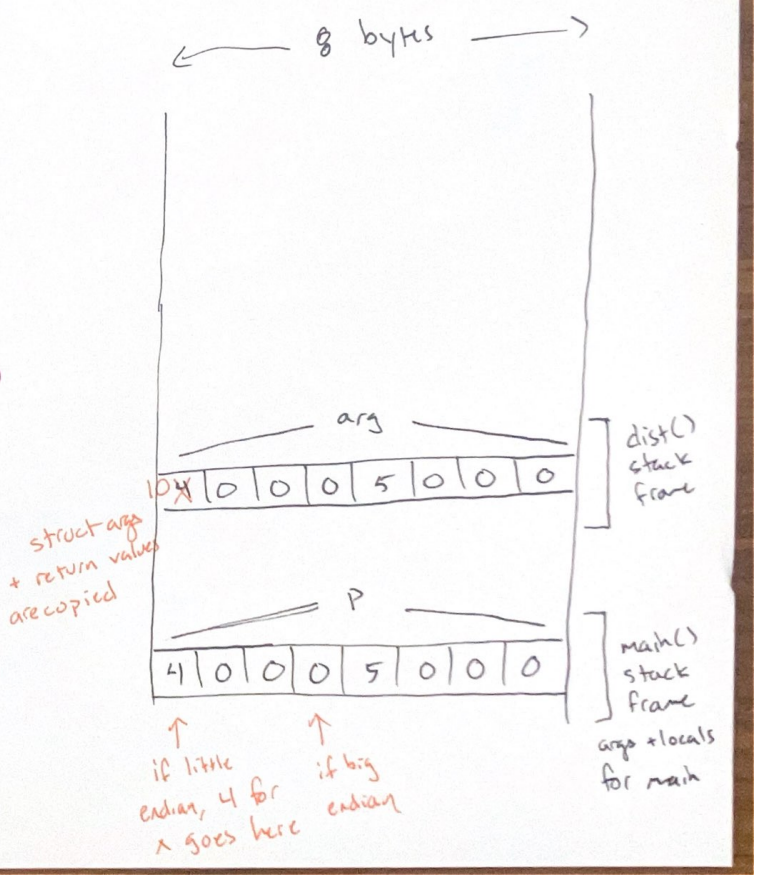
```
int main() {
    Point p = {4, 5};
    dist(p);
}

float dist(Point arg) {
    ... arg.x  arg.y ...
}
```

p.x — look up in memory at the "x" offset in p (0 bytes)

p.y — look up in memory at the "y" offset from p (4 bytes)

arg.x = 10 — imagine dist changes x field



In contrast to C, languages like Python and Java (and others) have *immutable* strings that support operations like concatenation with +. How does that work in the machine?

```
$ python3
>>> a = "hello "
>>> b = "cse29"
>>> c = a + b
>>> c
'hello cse29'
>>> a
'hello '
>>> b
'cse29'
```

```c
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

struct String {
    uint64_t length; // should always equal strlen(contents)
    char* contents;  // should always have allocated space of length + 1
};

typedef struct String String;

String new_String(char* init_contents) {
    uint64_t size = strlen(init_contents);
    String r = { size, init_contents };
    return r;
}

String plus(String s1, String s2) {
    uint64_t new_size = s1.length + s2.length + 1;
    char new_contents[new_size];
    strncpy(new_contents, s1.contents, s1.length);
    strncpy(new_contents + s1.length, s2.contents, s2.length);
    new_contents[new_size - 1] = 0;
    String r = { new_size - 1, new_contents };
    return r;
}

int main() {
    String s = new_String("hello");
    printf("%s\n", s.contents);

    String s2 = new_String("cse29");

    String hello_cse = plus(s, s2);
    String hello_bang = plus(s, new_String("!!!!"));

    printf("%s\n", hello_cse.contents);
    printf("%s\n", hello_bang.contents);
}
```