Today

- Structured Types (cont)

Assignments

- HW7 due
- HW8 out (soon-ish)
Passing object by value (copy) ... More complicated

```c
// create an object value
void main() push %rbp
{
    rec r;
    sub $24, %rsp # allocate stack for r
    r.x = 1; mov $1, -24(%rbp) # 1st 4 bytes
    r.y = 2; mov $2, -20(%rbp) # 2nd 4 bytes
    r.a[0] = 3; mov $3, -16(%rbp) # 3rd 4 bytes
    r.a[1] = 4; mov $4, -12(%rbp) # 4th 4 bytes
    r.p = &r.x; lea -24(%rbp), %rax # pointer to r.x
    add_1(r);
    # create copy by pushing fields onto the stack
    push -8(%rbp)
    push -16(%rbp)
    push -24(%rbp)
    call add_1
    add $24, %rsp # remove copy from stack
}

void add_1(rec r)
{
    # overwrites the copy created in caller ...
    r.x += 1; mov 8(%rsp), %eax
    r.y += 1; add $1, %eax
    r.a[0] += 1; mov %eax, 8(%rsp)
    r.a[1] += 1; mov 12(%rsp), %eax
    add $1, %eax
    mov %eax, 12(%rsp)
    mov 16(%rsp), %eax
    add $1, %eax
    mov %eax, 16(%rsp)
    mov 20(%rsp), %eax
    add $1, %eax
    mov %eax, 20(%rsp)
    ret
```

**Exercise:** Trace to and draw the stack contents after the last main instruction
Return a struct value (full example, -Og compile flag)

```c
int main()
{
    rec r1;
    r1.x = 1;
    r1.y = 2;
    r1.a[0] = 3;
    r1.a[1] = 4;
    r1.p = &r1.x;
    rec r2 = add_1(r1);
}
```

```c
rec add_1(rec r)
{
    r.x += 1;
    return r;
}
```

**Exercise:** Trace to and draw the stack contents after the last main instruction.
Note that this results in 3 copies (w/out optimization)

- the original \texttt{r1} object in \texttt{main}
- the copy passed into \texttt{r}
- the copy returned from \texttt{add_1}

Handling structs in the ABI ...

- more complicated than what shown here (and what is in the book)
- e.g., small versus large structs
- the examples give the general ideas