Today

- Structured Types

Assignments

- HW7 out (due Thur)
Representing C Struct objects

C/C++ Structs

```c
struct rec {  // 4 fields (x, y, a, p):
    int x;    // -- 2 ints (4 bytes each)
    int y;    // -- 1 2-elem int array (8 bytes)
    int a[2]; // -- 1 pointer (8 bytes)
    int* p;   // 24 total bytes
};
```

Implemented similarly to arrays

- each field stored contiguously in memory
- address of first field is the address of the struct object
A simple example of creating a struct object on the stack

```
// struct type definition
typedef struct rec {
    int x;
    int y;
    int a[2];
    int* p;
} rec;

// create an object value
void main()
{
    rec r;              # assembly encoding:
    ...                # allocate stack space
    r.x = 1;            # 1st 4 bytes
    mov $1, (%rsp)
    r.y = 2;            # 2nd 4 bytes
    mov $2, 4(%rsp)
    r.a[0] = 3;         # 3rd 4 bytes
    mov $3, 8(%rsp)
    r.a[1] = 4;         # 4th 4 bytes
    mov $4, 12(%rsp)
    r.p = &r.x;         # rsp is r.x's address
    ...                # store address in r.p
    mov %rax, 16(%rsp)
}
```
A simple example of creating a struct object dynamically

... 

// create an object value
void main()
{
    // allocate r
    rec* r = (rec*)malloc(sizeof(rec));
    // initialize r
    r->x = 1;
    r->y = 2;
    r->a[0] = 3;
    r->a[1] = 4;
    r->p = &r->x;
    ...
}

• the object is stored contiguously on the heap
Passing struct object by pointer (by reference)

// dynamically create an object value
void main()
{
    // allocate r
    rec* r = (rec*)malloc(sizeof(rec));
    init_rec(r);
}

void init_rec(rec* r)
{
    // 1st arg stores the address
    r->x = 1;
    r->y = 2;
    r->a[0] = 3;
    r->a[1] = 4;
    r->p = &r->x;
}