Section 5: Arrays & Other Data Structures

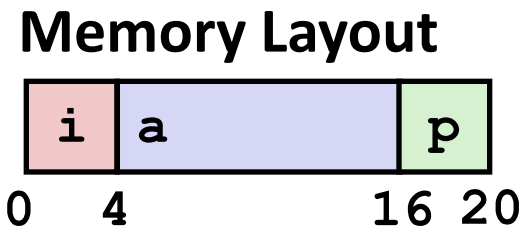
- Array allocation and access in memory
- Multi-dimensional or nested arrays
- Multi-level arrays
- Other structures in memory
- Data structures and alignment

Structures

```
struct rec {
  int i;
  int a[3];
  int *p;
};
```

Structures

```
struct rec {
  int i;
  int a[3];
  int *p;
};
```



- **Characteristics**
  - Contiguously-allocated region of memory
  - Refer to members within structure by names
  - Members may be of different types

Structures

```
struct rec {
  int i;
  int a[3];
  int *p;
};
```

- **Accessing Structure Member**
  - Given an instance of the struct, we can use the `.` operator, just like Java:
    - `struct rec r1;  r1.i = val;`
  - What if we have a *pointer* to a struct: `struct rec *r = &r1;`
    - Using `*` and `.` operators: `(*r).i = val;`
    - Or, use `->` operator for short: `r->i = val;`
  - Pointer indicates first byte of structure; access members with offsets

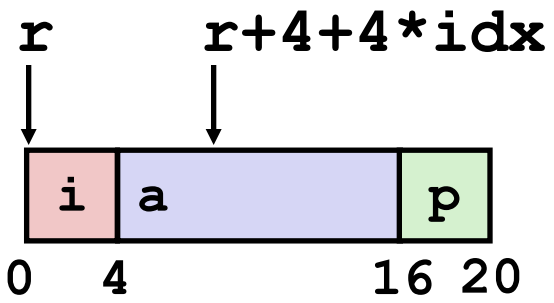
```
void
set_i(struct rec *r,
      int val)
{
    r->i = val;
}
```

IA32 Assembly

```
# %eax = val
# %edx = r
movl %eax,(%edx)    # Mem[r] = val
```

Generating Pointer to Structure Member

```
struct rec {
  int i;
  int a[3];
  int *p;
};
```



- **Generating Pointer to Array Element**
  - Offset of each structure member determined at compile time

```
int *find_a
 (struct rec *r, int idx)
{
   return &r->a[idx];
}
```

```
# %ecx = idx
# %edx = r
leal 0(,%ecx,4),%eax    # 4*idx
leal 4(%eax,%edx),%eax # r+4*idx+4
```