Structs

15-411/15-611 Compiler Design

Seth Copen Goldstein

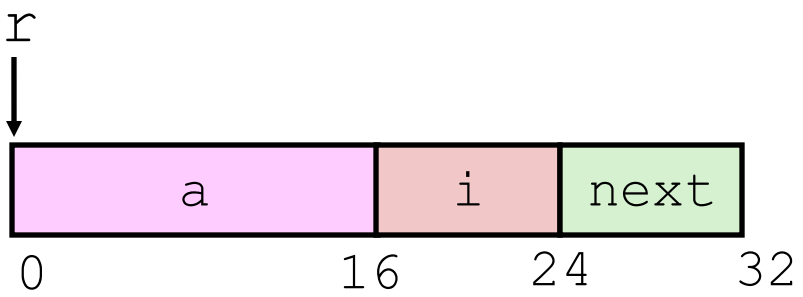
March 17, 2026

Today

- Structures and their machine requirements
- Small and big types
- Language issues
  - restrictions
  - parsing
  - static semantics
- Dynamic Semantics
  - &: pointers, arrays, and structures
  - assignment
- Registers and small type sizes

Structure Representation

```
struct rec {
    int a[4];
    size_t i;
    struct rec *next;
};
```



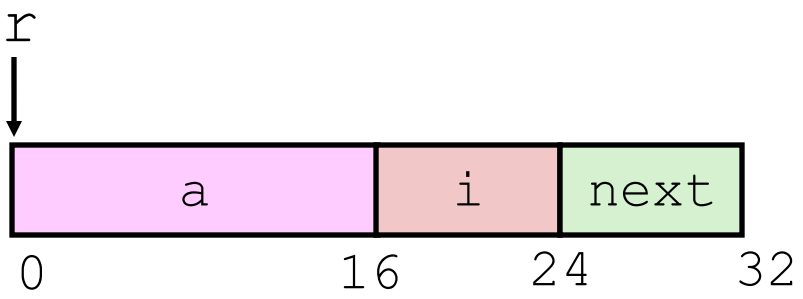
- Structure represented as block of memory
  - **Big enough to hold all of the fields**
- Fields ordered according to declaration
  - **Even if another ordering could yield a more compact representation**

Not important if you **stay in C0**

- Compiler determines overall size + positions of fields
  - **Machine-level program has no understanding of the structures in the source code**

Structure Representation

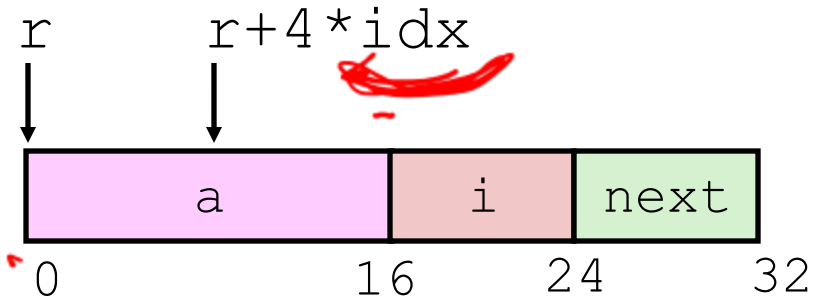
```
struct rec {
    int a[4];
    size_t i;
    struct rec *next;
};
```



- Structure represented as block of memory
  - **Big enough to hold all of the fields**
- Fields ordered according to declaration Why?
  - **Even if another ordering could yield a more compact representation**
- Compiler determines overall size + positions of fields
  - **Machine-level program has no understanding of the structures in the source code**

Generating Pointer to Structure Member

```
struct rec {
    int a[4];
    size_t i;
    struct rec *next;
};
```



- Generating Pointer to Array Element
  - Offset of each structure member determined at compile time
  - Compute as **`r + 4*idx`**

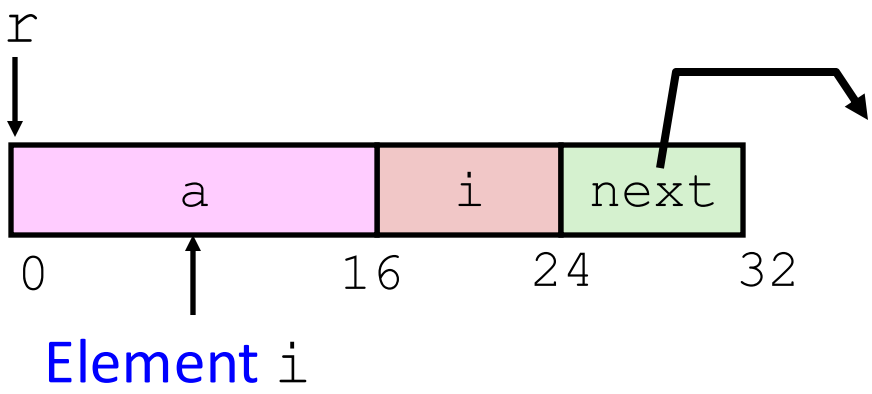
```
int *
get_ap(struct rec *r, size_t idx)
{
  return &r->a[idx];
}
```

```
  # r in %rdi, idx in %rsi
  leaq   (%rdi,%rsi,4), %rax
  ret
```

Following Linked List

```
void
set_val(struct rec *r, int val)
{
  while (r) {
    int i = r->i;
    r->a[i] = val;
    r = r->next;
  }
}
```

```
struct rec {
    int a[4];
    int i;
    struct rec *next;
};
```



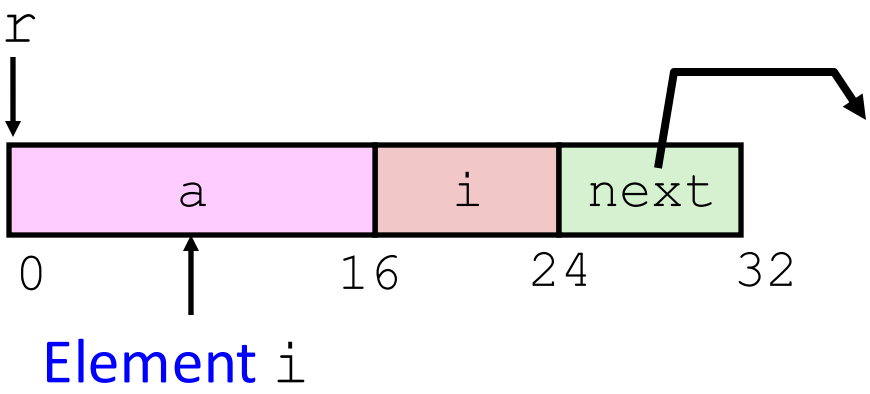
Register	Value
%rdi	r
%rsi	val

```
.L11:                                  # loop:
  movslq  16(%rdi), %rax               #   i = M[r+16]
  movl    %esi, (%rdi,%rax,4)          #   M[r+4*i] = val
  movq    24(%rdi), %rdi               #   r = M[r+24]
  testq   %rdi, %rdi                   #   Test r
  jne     .L11                         #   if !=0 goto loop
```

Which Registers to Use?

```
void
set_val(struct rec *r, int val)
{
  while (r) {
    int i = r->i;
    r->a[i] = val;
    r = r->next;
  }
}
```

```
struct rec {
    int a[4];
    int i;
    struct rec *next;
};
```

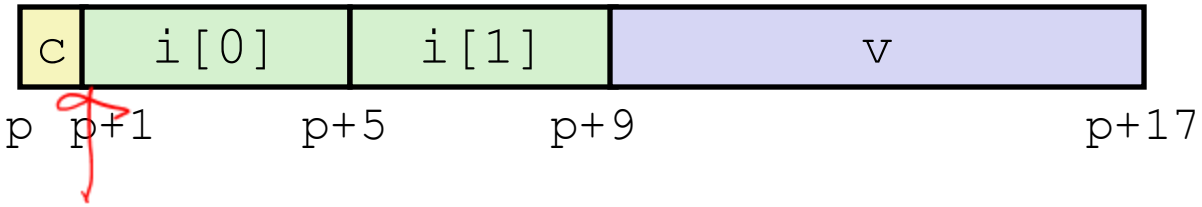


Register	Value
%rdi	r
%esi	val

```
.L11:                                  # loop:
  movslq  16(%rdi), %rax               #   i = M[r+16]
  movl    %esi, (%rdi,%rax,4)          #   M[r+4*i] = val
  movq    24(%rdi), %rdi               #   r = M[r+24]
  testq   %rdi, %rdi                   #   Test r
  jne     .L11                         #   if !=0 goto loop
```

Structures & Alignment

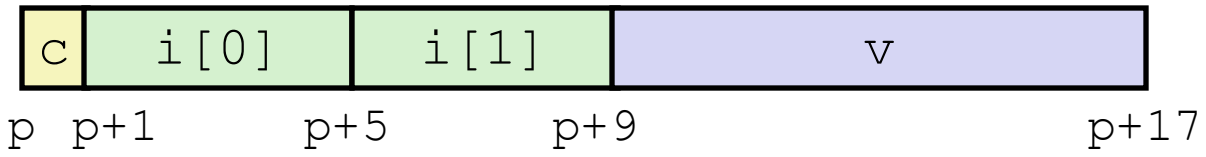
- Unaligned Data




```
struct S1 {
  char c;
  int i[2];
  double v;
} *p;
```

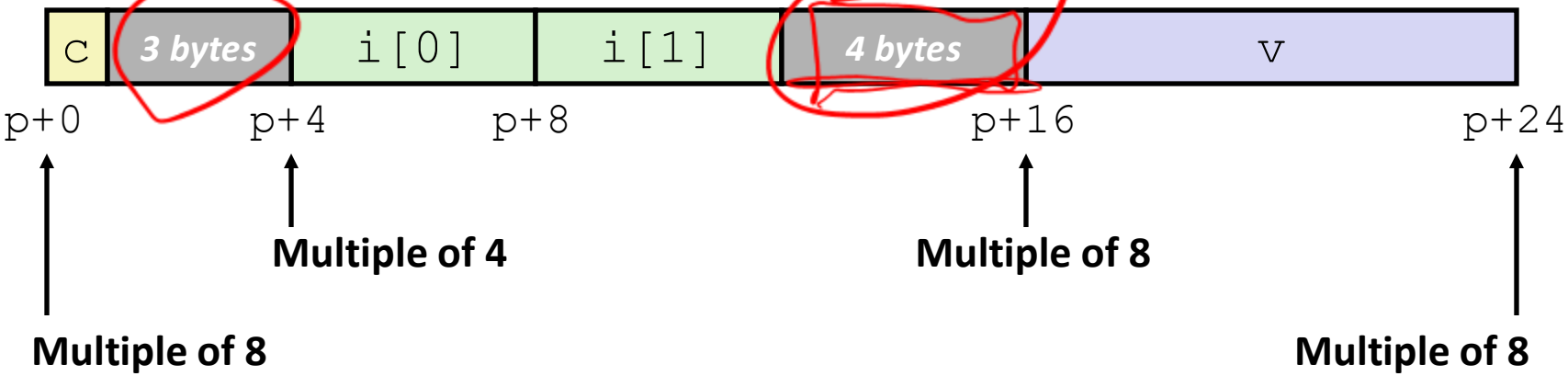
Structures & Alignment

- Unaligned Data




```
struct S1 {
  char c;
  int i[2];
  double v;
} *p;
```

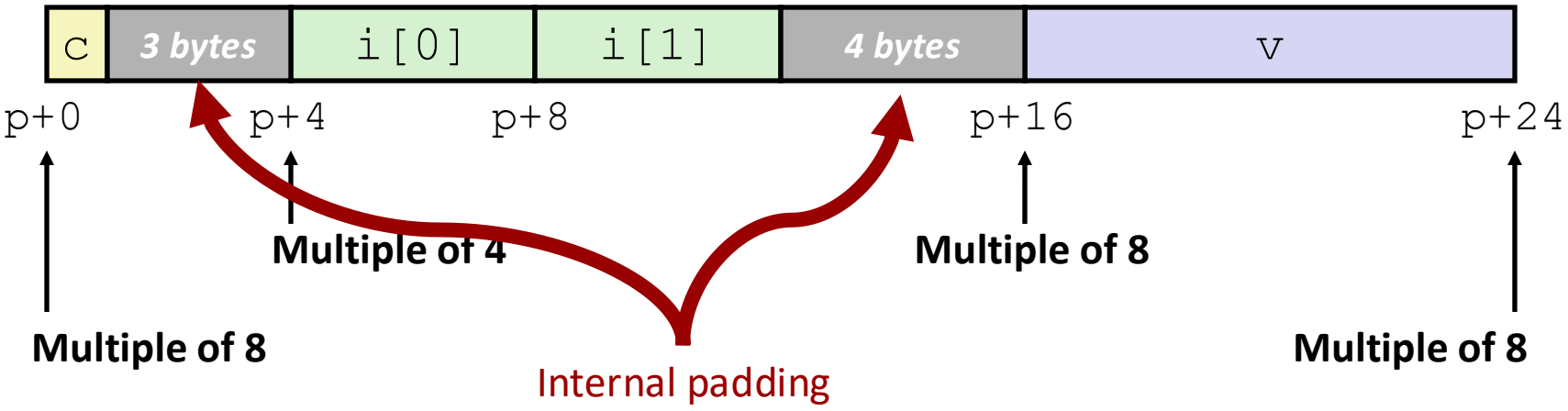
- Aligned Data
  - Primitive data type requires ***K*** bytes
  - Address must be multiple of ***K***



Satisfying Alignment within Structures

- Within structure:
  - Satisfy each element's alignment requirement
- Overall structure placement
  - Each structure has alignment requirement **K**
    - **K** = Largest alignment of any element
  - Initial address & structure length must be multiples of **K**
- Example: K = 8, due to **`double`** element

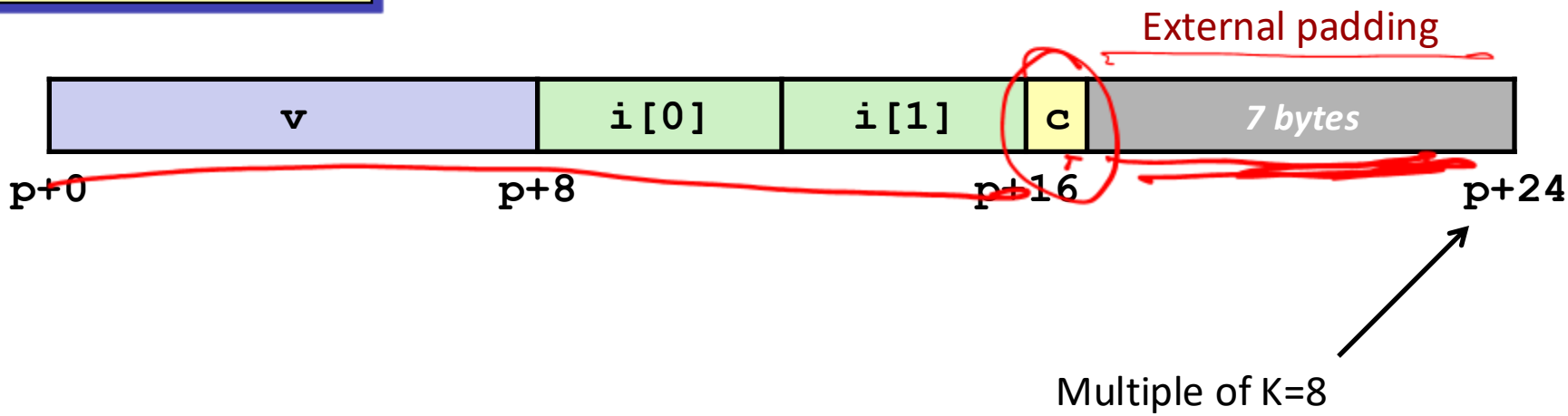
```
struct S1 {
  char c;
  int i[2];
  double v;
} *p;
```



Meeting Overall Alignment Requirement

- For largest alignment requirement K
- Overall structure must be multiple of K

```
struct S2 {
  double v;
  int i[2];
  char c;
} *p;
```



Alignment Principles

- Aligned Data
  - Primitive data type requires ***K*** bytes
  - Address must be multiple of ***K***
  - Required on some machines; advised on x86-64
- Motivation for Aligning Data
  - Memory accessed by (aligned) chunks of 4 or 8 bytes (system dependent)
    - Inefficient to load or store datum that spans cache lines (64 bytes). Intel states should avoid crossing 16 byte boundaries.
    - Virtual memory trickier when datum spans 2 pages (4 KB pages)
- Compiler
  - Inserts gaps in structure to ensure correct alignment of fields

Size & Alignment of C types (x86-64)

- 1 byte: **`char`**, …
  - no restrictions on address
- 2 bytes: **`short`**, …
  - lowest 1 bit of address must be 0_2
- 4 bytes: **`int`**, **`float`**, …
  - lowest 2 bits of address must be 00_2
- 8 bytes: **`double`**, `long`, **`char *`**, …
  - lowest 3 bits of address must be 000_2

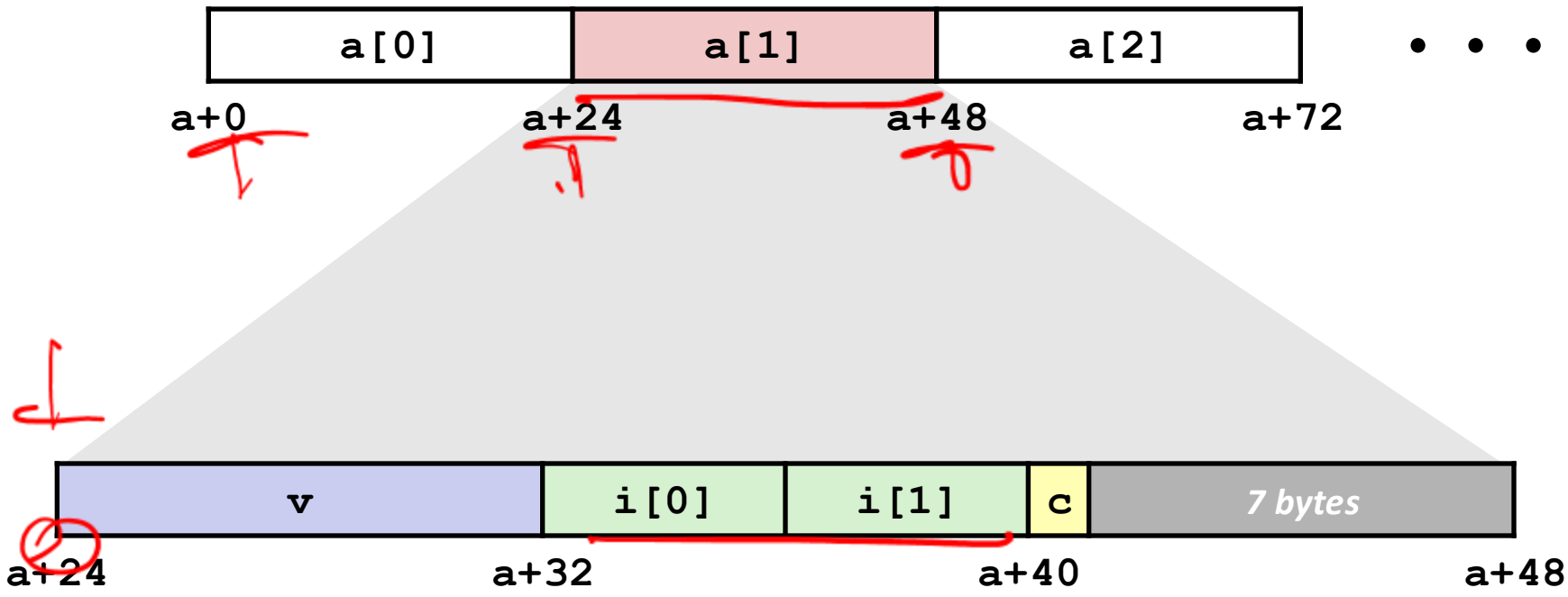
Size & Alignment of C0 types (x86-64)

- 4 bytes: **`int`**, **`bool`**
  - lowest 2 bits of address must be 00_2
- 8 bytes: τ**`*`**, τ**`[]`**
  - lowest 3 bits of address must be 000_2

Arrays of Structures

- Overall structure length multiple of K
- Satisfy alignment requirement for every element

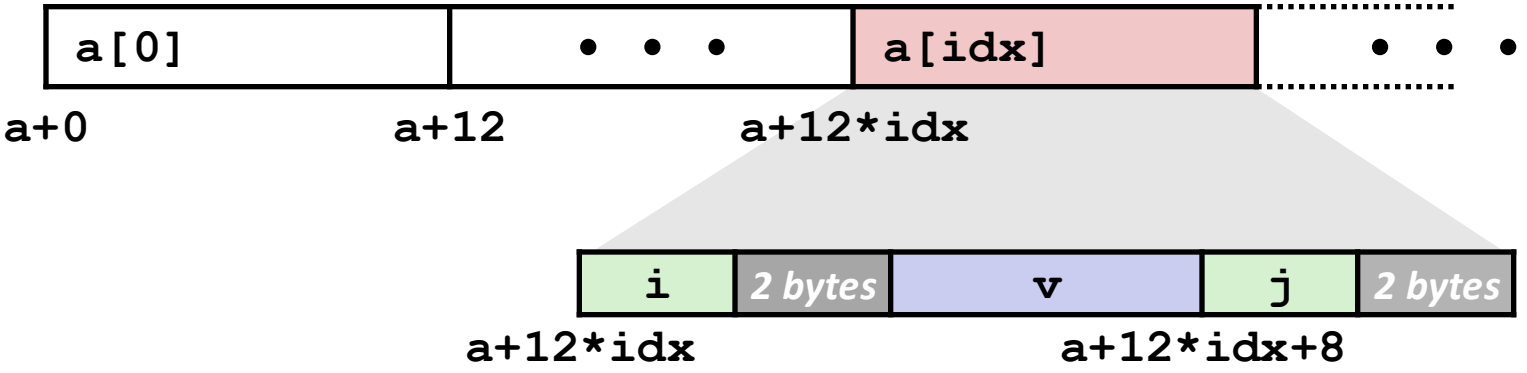
```
struct S2 {
  double v;
  int i[2];
  char c;
} a[10];
```



Accessing Array Elements

- Compute array offset 12*idx
  - **`sizeof(S3)`**, including alignment spacers
- Element **`j`** is at offset 8 within structure
- Assembler gives offset **`a+8`**
  - Resolved during linking

```
struct S3 {
  short i;
  float v;
  short j;
} a[10];
```

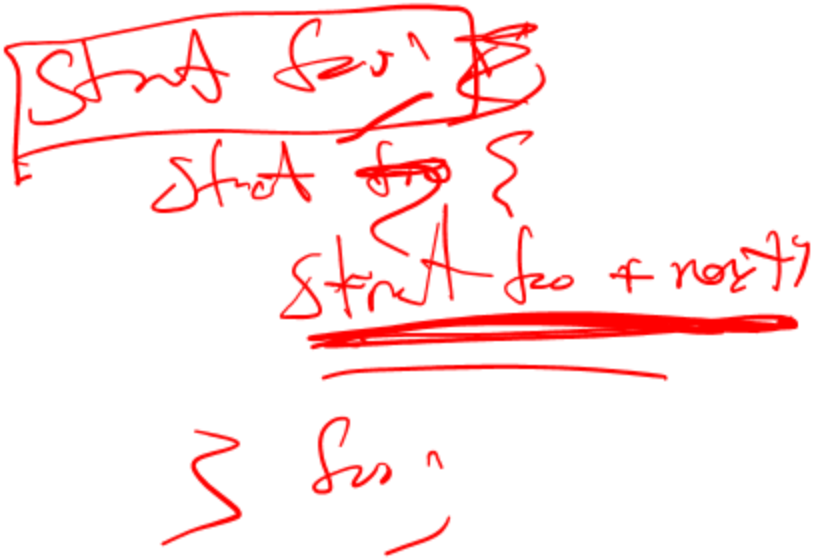



```
short get_j(int idx)
{
  return a[idx].j;
}
```

```
# %rdi = idx
leaq (%rdi,%rdi,2),%rax # 3*idx
movzwl a+8(,%rax,4),%eax
```

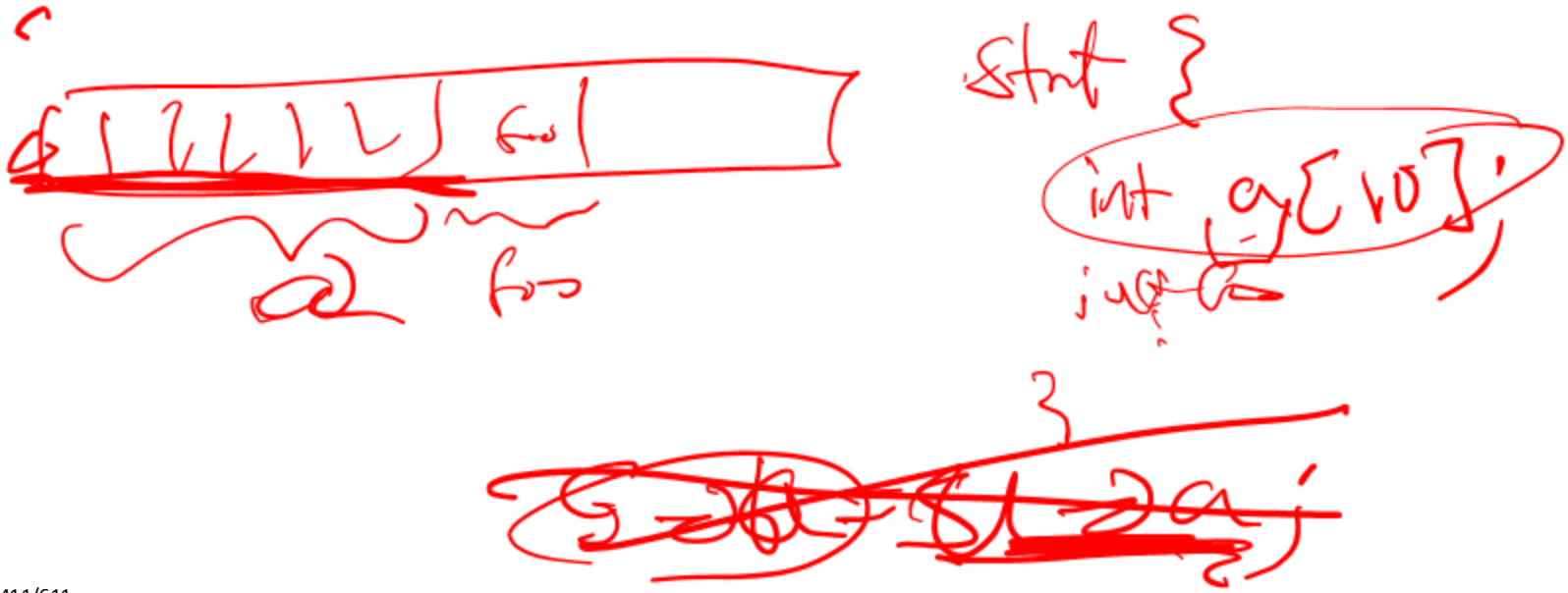
L4 structs

- Must be allocated on the heap.
- Field names are in their own namespace
- In each **`struct`**, field names must be unique
- Can only be defined once.
- Can be used (in special cases) before declared!



Big and Small Types

- Small types fit in registers
  - int, bool, τ*, τ[]
  - 4 or 8 bytes in L4
- Large types are allocated on the heap
  - **`struct`** *s*



Restrictions vis a vis Small Types

All of the following must be small types:

- Local variables
- function parameters
- return values
- lval and rval in assignments
- Expressions
  - conditional expressions
  - `==` and `!=`
  - simple expressions (i.e., expressions as statements)

Namespaces

- Each struct definition creates its own namespace, so
  - fieldnames never conflict with other variables, function names, type names, field names in other **`struct`**s
- Field names in a structure must be unique

Declare v. Define

- Declaration: `struct s;`
- Definition: `struct s {` $\tau_1 f_1$`;` … $\tau_n f_n$`; };`
- Only 1 definition allowed
- If size is irrelevant:
  - Can be used before defined
  - Can be used without prior declaration!
- Size is relevant in
  - `alloc(struct` *s*`)` and `alloc_array(struct` *s*`,` *e*`)`

Static Semantics

- Extend types

$$\tau ::= \text{int} \mid \text{bool} \mid \tau* \mid \tau[] \mid \text{struct } s$$

- Extend expressions

$$d ::= \ldots \mid d.f$$
$$e ::= \ldots \mid e.f \mid e \Rightarrow f$$

- Elaboration

$$e \rightarrow f \equiv (*e).f$$

- Typing

$$\frac{\Gamma \vdash e : \text{struct } s \quad s.f : \tau}{\Gamma \vdash e.f : \tau}$$

Note: **`struct`** *s* must be defined

Static Semantics

- Extend types

$$\tau \ ::= \text{int} \mid \text{bool} \mid \tau* \mid \tau[] \mid \text{struct } s$$

- Extend expressions

$$d \ ::= \ \ldots \mid d.f$$
$$e \ ::= \ \ldots \mid e.f \mid e \rightarrow f$$

- Elabora

Because we defined *d* to be an expression no other typing rules are needed!

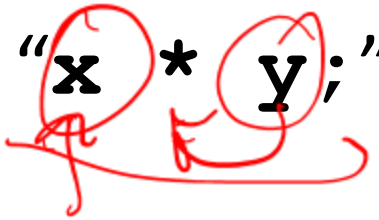
(Note: restrictions to small types on all previous rules.)

- Typing

$$\frac{\Gamma \vdash e : \text{struct } s \quad s.f : \tau}{\Gamma \vdash e.f : \tau}$$

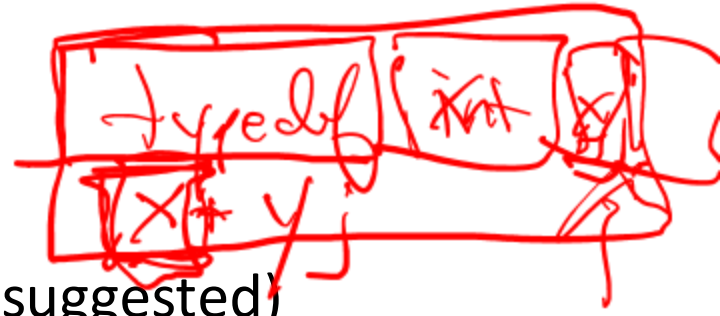
Parsing L4

- What is meaning of "`x * y;`"



Parsing L4

- What is meaning of "`x * y;`"
  - Is it variable x times variable y?
  - Is it variable y is a pointer to type x?
- How to resolve this context sensitive Issue?
  - top-down parser will require backtracking
  - bottom-parser:
    - Solve after parse. How?
    - Get lexer involved.
      (beware lexer lookahead - but suggested)



Dynamic Semantics - Approach

```
struct Point {
   int x;
   int y;
};
…
struct Point* p = alloc_struct(point);
```

- What is value of **`p->x`**?

Dynamic Semantics - Approach

```
struct Point {
   int x;
   int y;
};
…
struct Point* p = alloc_struct(point);
```

- What is value of `(*p).x`?
- What is value of `*p`?

Dynamic Semantics - Approach

```
struct Point {
   int x;
   int y;
};
…
struct Point* p = alloc_struct(point);
```

- What is value of `(*p).x`?
- What is value of `*p`?
- Two approaches

Dynamic Semantics - Approach

```
struct Point {
   int x;
   int y;
};
…
struct Point* p = alloc_struct(point);
```

- What is value of `(*p).x`?
- What is value of `*p`?
- Approach one: `*p` is entire structure
  - read in entire structure
  - select field

$$H\,;\,S\,;\,\eta \vdash e.f \triangleright K \longrightarrow H\,;\,S\,;\,\eta \vdash e \triangleright (\_.y\,,\,K)$$
$$H\,;\,S\,;\,\eta \vdash \{x = v_1, y = v_2\} \triangleright (\_.y\,,\,K) \longrightarrow H\,;\,S\,;\,\eta \vdash v_2 \triangleright K$$

Dynamic Semantics - Approach

```
struct Point {
   int x;
   int y;
};
…
struct Point* p = alloc_struct(point);
```

- What is value of `(*p).x`?
- What is value of `*p`?
- Approach one: `*p` is entire structure
- Approach two: `*p` has no meaning in and of itself, rather p is an address and `(*p).x` is an address calculation followed by a load

Address of operator: &

- Introduce, into the dynamic semantics, the address-of operation, **`&`**
- So, **`(*p).f`** becomes:
  - get address of **`p`**
  - get offset from start of **`p`** to **`f`**
  - calculate a = sum of above
  - load proper number of bytes from a
- I.e., **`(*p).f`** ⇒ **`*(&((*p).f))`**
- Notice similarity to *d used as an lval (from last lecture)

From last lecture

Writing to the heap

- left to right evaluation of address and rval

$$H\ ;\ S\ ;\ \eta \vdash \mathsf{assign}(*d, e) \blacktriangleright K \longrightarrow H\ ;\ S\ ;\ \eta \vdash d \triangleright (\mathsf{assign}(*\_, e)\ ,\ K)$$

$$H\ ;\ S\ ;\ \eta \vdash a \triangleright (\mathsf{assign}(*\_, e)\ ,\ K) \longrightarrow H\ ;\ S\ ;\ \eta \vdash e \triangleright (\mathsf{assign}(*a, \_)\ ,\ K)$$

- Then making assignment (if $a \neq 0$)

$$H\ ;\ S\ ;\ \eta \vdash c \triangleright (\mathsf{assign}(*a, \_)\ ,\ K) \longrightarrow H[a \mapsto c]\ ;\ S\ ;\ \eta \vdash \mathsf{nop} \blacktriangleright K \quad (a \neq 0)$$

$$H\ ;\ S\ ;\ \eta \vdash c \triangleright (\mathsf{assign}(*a, \_)\ ,\ K) \longrightarrow \mathsf{exception}(\mathsf{mem}) \quad (a = 0)$$

Address of operator: &

- Introduce, into the dynamic semantics, the address-of operation, **`&`**
- So, **`(*p).f`** becomes:
  - get address of **`p`**
  - get offset from start of **`p`** to **`f`**
  - calculate a = sum of above
  - load proper number of bytes from a
- I.e., **`(*p).f`** $\Rightarrow$ **`*(&((*p).f))`**
- Notice similarity to *d used as an lval (from last lecture)

Field access

- We evaluate e.f as *(&(e.f))

$$H\ ;\ S\ ;\ \eta \vdash e.f \rhd K \quad \longrightarrow \quad H\ ;\ S\ ;\ \eta \vdash *(\&(e.f)) \rhd K$$

Addresses and Large Types

$$H\ ;\ S\ ;\ \eta \vdash \&(*e) \triangleright K \longrightarrow H\ ;\ S\ ;\ \eta \vdash e \triangleright K$$

Addresses and Large Types

$$H\ ;\ S\ ;\ \eta \vdash \&(*e) \triangleright K \longrightarrow H\ ;\ S\ ;\ \eta \vdash e \triangleright K$$

$$H\ ;\ S\ ;\ \eta \vdash \&(e.f) \triangleright K \longrightarrow$$

- get address of **`p`**
- get offset from start of **`p`** to **`f`**
- calculate a = sum of above
- load proper number of bytes from a

Addresses and Large Types

$$H\ ;\ S\ ;\ \eta \vdash \&(*e) \triangleright K \longrightarrow H\ ;\ S\ ;\ \eta \vdash e \triangleright K$$

$$H\ ;\ S\ ;\ \eta \vdash \&(e.f) \triangleright K \longrightarrow H\ ;\ S\ ;\ \eta \vdash \&e \triangleright (\&(\_.f)\ ,\ K)$$

$$H\ ;\ S\ ;\ \eta \vdash a \triangleright (\&(\_.f),\ K) \longrightarrow$$

– get address of **p**

– get offset from start of **p** to **f**

– calculate a = sum of above

– load proper number of bytes from a

Addresses and Large Types

$$H\ ;\ S\ ;\ \eta \vdash \&(*e) \triangleright K \longrightarrow H\ ;\ S\ ;\ \eta \vdash e \triangleright K$$

$$H\ ;\ S\ ;\ \eta \vdash \&(e.f) \triangleright K \longrightarrow H\ ;\ S\ ;\ \eta \vdash \&e \triangleright (\&(\_.f)\ ,\ K)$$

$$H\ ;\ S\ ;\ \eta \vdash a \triangleright (\&(\_.f)\ ,\ K) \longrightarrow H\ ;\ S\ ;\ \eta \vdash a + \mathsf{offset}(s, f) \triangleright K \quad (a \neq 0, a : \mathsf{struct}\ s)$$

- get address of **p**
- get offset from start of **p** to **f**
- calculate a = sum of above
- load proper number of bytes from a

Addresses and Large Types

$$H\ ;\ S\ ;\ \eta \vdash \&(*e) \triangleright K \longrightarrow H\ ;\ S\ ;\ \eta \vdash e \triangleright K$$

$$H\ ;\ S\ ;\ \eta \vdash \&(e.f) \triangleright K \longrightarrow H\ ;\ S\ ;\ \eta \vdash \&e \triangleright (\&(\_.f)\ ,\ K)$$

$$H\ ;\ S\ ;\ \eta \vdash a \triangleright (\&(\_.f)\ ,\ K) \longrightarrow H\ ;\ S\ ;\ \eta \vdash a + \mathsf{offset}(s, f) \triangleright K \quad (a \neq 0, a : \mathsf{struct}\ s)$$

$$H\ ;\ S\ ;\ \eta \vdash a \triangleright (\&(\_.f)\ ,\ K) \longrightarrow \mathsf{exception}(\mathsf{mem}) \quad (a = 0)$$

- get address of **p**
- get offset from start of **p** to **f**
- calculate a = sum of above
- load proper number of bytes from a

Addresses and Large Types

$$H\ ;\ S\ ;\ \eta \vdash \&(e_1[e_2]) \triangleright K \quad \longrightarrow \quad H\ ;\ S\ ;\ \eta \vdash e_1 \triangleright (\&(\_[e_2])\ ,\ K)$$

$$H\ ;\ S\ ;\ \eta \vdash a \triangleright (\&(\_[e_2])\ ,\ K) \quad \longrightarrow \quad H\ ;\ S\ ;\ \eta \vdash e_2 \triangleright (\&(a[\_]\ ,\ K)$$

Addresses and Large Types

$$H\ ;\ S\ ;\ \eta \vdash \&(e_1[e_2]) \triangleright K \longrightarrow H\ ;\ S\ ;\ \eta \vdash e_1 \triangleright (\&(\_[e_2])\ ,\ K)$$

$$H\ ;\ S\ ;\ \eta \vdash a \triangleright (\&(\_[e_2])\ ,\ K) \longrightarrow H\ ;\ S\ ;\ \eta \vdash e_2 \triangleright (\&(a[\_]\ ,\ K)$$

$$H\ ;\ S\ ;\ \eta \vdash i \triangleright (\&(a[\_]\ ,\ K) \longrightarrow$$

Addresses and Large Types

$$H\ ;\ S\ ;\ \eta \vdash \&(*e) \triangleright K \longrightarrow H\ ;\ S\ ;\ \eta \vdash e \triangleright K$$

$$H\ ;\ S\ ;\ \eta \vdash \&(e.f) \triangleright K \longrightarrow H\ ;\ S\ ;\ \eta \vdash \&e \triangleright (\&(\_.f)\ ,\ K)$$

$$H\ ;\ S\ ;\ \eta \vdash a \triangleright (\&(\_.f)\ ,\ K) \longrightarrow H\ ;\ S\ ;\ \eta \vdash a + \mathsf{offset}(s, f) \triangleright K \quad (a \neq 0, a : \mathsf{struct}\ s)$$

$$H\ ;\ S\ ;\ \eta \vdash a \triangleright (\&(\_.f)\ ,\ K) \longrightarrow \mathsf{exception}(\mathsf{mem}) \quad (a = 0)$$

$$H\ ;\ S\ ;\ \eta \vdash \&(e_1[e_2]) \triangleright K \longrightarrow H\ ;\ S\ ;\ \eta \vdash e_1 \triangleright (\&(\_[e_2])\ ,\ K)$$

$$H\ ;\ S\ ;\ \eta \vdash a \triangleright (\&(\_[e_2])\ ,\ K) \longrightarrow H\ ;\ S\ ;\ \eta \vdash e_2 \triangleright (\&(a[\_]\ ,\ K)$$

$$H\ ;\ S\ ;\ \eta \vdash i \triangleright (\&(a[\_]\ ,\ K) \longrightarrow H\ ;\ S\ ;\ \eta \vdash a + i|\tau| \triangleright K \quad a \neq 0, 0 \leq i < \mathsf{length}(a), a : \tau[\,]$$

Addresses and Large Types

$$H\ ;\ S\ ;\ \eta \vdash \&(*e) \triangleright K \longrightarrow H\ ;\ S\ ;\ \eta \vdash e \triangleright K$$

$$H\ ;\ S\ ;\ \eta \vdash \&(e.f) \triangleright K \longrightarrow H\ ;\ S\ ;\ \eta \vdash \&e \triangleright (\&(\_.f)\ ,\ K)$$

$$H\ ;\ S\ ;\ \eta \vdash a \triangleright (\&(\_.f)\ ,\ K) \longrightarrow H\ ;\ S\ ;\ \eta \vdash a + \mathsf{offset}(s, f) \triangleright K \quad (a \neq 0, a : \mathsf{struct}\ s)$$

$$H\ ;\ S\ ;\ \eta \vdash a \triangleright (\&(\_.f)\ ,\ K) \longrightarrow \mathsf{exception}(\mathsf{mem}) \quad (a = 0)$$

$$H\ ;\ S\ ;\ \eta \vdash \&(e_1[e_2]) \triangleright K \longrightarrow H\ ;\ S\ ;\ \eta \vdash e_1 \triangleright (\&(\_[e_2])\ ,\ K)$$

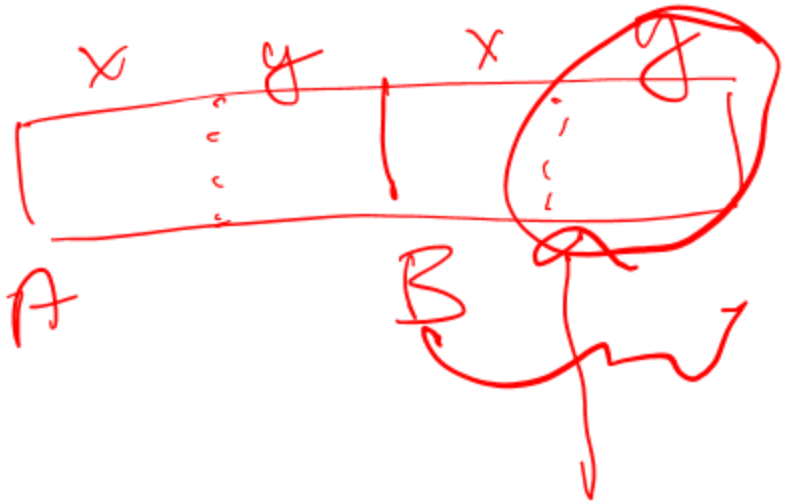
$$H\ ;\ S\ ;\ \eta \vdash a \triangleright (\&(\_[e_2])\ ,\ K) \longrightarrow H\ ;\ S\ ;\ \eta \vdash e_2 \triangleright (\&(a[\_]\ ,\ K)$$

$$H\ ;\ S\ ;\ \eta \vdash i \triangleright (\&(a[\_]\ ,\ K) \longrightarrow H\ ;\ S\ ;\ \eta \vdash a + i|\tau| \triangleright K \quad a \neq 0, 0 \leq i < \mathsf{length}(a), a : \tau[\,]$$

$$H\ ;\ S\ ;\ \eta \vdash i \triangleright (\&(a[\_]\ ,\ K) \longrightarrow \mathsf{exception}(\mathsf{mem}) \quad a = 0 \text{ or } i < 0 \text{ or } i \geq \mathsf{length}(a)$$

Example

```
struct Point {
  int x;
  int y;
};
struct Line {
  struct Point A;
  struct Point B;
};
…
struct Line* L = alloc(struct Line);
…
int x = L->B.y;
```



After elaboration =>

Example

```
struct Point {
  int x;
  int y;
};
struct Line {
  struct Point A;
  struct Point B;
};
…
struct Line* L = alloc(struct Line);
…
int x = (*L).B.y;
```

x = (*L).B.y;

$$H\ ;\ S\ ;\ \eta \vdash \mathsf{assign}(x, (*L).B.y) \ \blacktriangleright\ K$$

$H\ ;\ S\ ;\ \eta \vdash e.f \rhd K$	$\longrightarrow$	$H\ ;\ S\ ;\ \eta \vdash *(\&(e.f)) \rhd K$
$H\ ;\ S\ ;\ \eta \vdash \&(*e) \rhd K$	$\longrightarrow$	$H\ ;\ S\ ;\ \eta \vdash e \rhd K$
$H\ ;\ S\ ;\ \eta \vdash \&(e.f) \rhd K$	$\longrightarrow$	$H\ ;\ S\ ;\ \eta \vdash \&e \rhd (\&(\_.f)\ ,\ K)$
$H\ ;\ S\ ;\ \eta \vdash a \rhd (\&(\_.f)\ ,\ K)$	$\longrightarrow$	$H\ ;\ S\ ;\ \eta \vdash a + \mathsf{offset}(s, f) \rhd K$ $(a \neq 0, a : \mathsf{struct}\ s)$
$H\ ;\ S\ ;\ \eta \vdash a \rhd (\&(\_.f)\ ,\ K)$	$\longrightarrow$	$\mathsf{exception}(\mathsf{mem})$ $(a = 0)$
$H\ ;\ S\ ;\ \eta \vdash \&(e_1[e_2]) \rhd K$	$\longrightarrow$	$H\ ;\ S\ ;\ \eta \vdash e_1 \rhd (\&(\_[e_2])\ ,\ K)$
$H\ ;\ S\ ;\ \eta \vdash a \rhd (\&(\_[e_2])\ ,\ K)$	$\longrightarrow$	$H\ ;\ S\ ;\ \eta \vdash e_2 \rhd (\&(a[\_]\ ,\ K)$
$H\ ;\ S\ ;\ \eta \vdash i \rhd (\&(a[\_]\ ,\ K)$	$\longrightarrow$	$H\ ;\ S\ ;\ \eta \vdash a + i\lvert\tau\rvert \rhd K$ $a \neq 0, 0 \leq i < \mathsf{length}(a), a : \tau[\,]$
$H\ ;\ S\ ;\ \eta \vdash i \rhd (\&(a[\_]\ ,\ K)$	$\longrightarrow$	$\mathsf{exception}(\mathsf{mem})$ $a = 0$ or $i < 0$ or $i \geq \mathsf{length}(a)$

x = (*L).B.y;

$$H\ ;\ S\ ;\ \eta \vdash \mathsf{assign}(x, (*L).B.y) \blacktriangleright K$$
$$\longrightarrow\ H\ ;\ S\ ;\ \eta \vdash ((*L).B.y) \quad \triangleright(\mathsf{assign}(x, \_)\ ,\ K)$$

$H\ ;\ S\ ;\ \eta \vdash e.f \triangleright K$	$\longrightarrow$	$H\ ;\ S\ ;\ \eta \vdash *(\&(e.f)) \triangleright K$
$H\ ;\ S\ ;\ \eta \vdash \&(*e) \triangleright K$	$\longrightarrow$	$H\ ;\ S\ ;\ \eta \vdash e \triangleright K$
$H\ ;\ S\ ;\ \eta \vdash \&(e.f) \triangleright K$	$\longrightarrow$	$H\ ;\ S\ ;\ \eta \vdash \&e \triangleright (\&(\_.f)\ ,\ K)$
$H\ ;\ S\ ;\ \eta \vdash a \triangleright (\&(\_.f)\ ,\ K)$	$\longrightarrow$	$H\ ;\ S\ ;\ \eta \vdash a + \mathsf{offset}(s, f) \triangleright K$ $(a \neq 0, a : \mathsf{struct}\ s)$
$H\ ;\ S\ ;\ \eta \vdash a \triangleright (\&(\_.f)\ ,\ K)$	$\longrightarrow$	$\mathsf{exception}(\mathsf{mem})$ $(a = 0)$
$H\ ;\ S\ ;\ \eta \vdash \&(e_1[e_2]) \triangleright K$	$\longrightarrow$	$H\ ;\ S\ ;\ \eta \vdash e_1 \triangleright (\&(\_[e_2])\ ,\ K)$
$H\ ;\ S\ ;\ \eta \vdash a \triangleright (\&(\_[e_2])\ ,\ K)$	$\longrightarrow$	$H\ ;\ S\ ;\ \eta \vdash e_2 \triangleright (\&(a[\_]\ ,\ K)$
$H\ ;\ S\ ;\ \eta \vdash i \triangleright (\&(a[\_]\ ,\ K)$	$\longrightarrow$	$H\ ;\ S\ ;\ \eta \vdash a + i\lvert\tau\rvert \triangleright K$ $a \neq 0, 0 \leq i < \mathsf{length}(a), a : \tau[\,]$
$H\ ;\ S\ ;\ \eta \vdash i \triangleright (\&(a[\_]\ ,\ K)$	$\longrightarrow$	$\mathsf{exception}(\mathsf{mem})$ $a = 0$ or $i < 0$ or $i \geq \mathsf{length}(a)$

x = (*L).B.y;

$$
\begin{array}{ll}
 & H\ ;\ S\ ;\ \eta \vdash \mathsf{assign}(x, (*L).B.y) \blacktriangleright K \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash ((*L).B.y) \rhd (\mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash *(\&((*L).B.y)) \rhd (\mathsf{assign}(x, \_)\ ,\ K)
\end{array}
$$

$H\ ;\ S\ ;\ \eta \vdash e.f \rhd K$	$\longrightarrow$	$H\ ;\ S\ ;\ \eta \vdash *(\&(e.f)) \rhd K$
$H\ ;\ S\ ;\ \eta \vdash \&(*e) \rhd K$	$\longrightarrow$	$H\ ;\ S\ ;\ \eta \vdash e \rhd K$
$H\ ;\ S\ ;\ \eta \vdash \&(e.f) \rhd K$	$\longrightarrow$	$H\ ;\ S\ ;\ \eta \vdash \&e \rhd (\&(\_.f)\ ,\ K)$
$H\ ;\ S\ ;\ \eta \vdash a \rhd (\&(\_.f)\ ,\ K)$	$\longrightarrow$	$H\ ;\ S\ ;\ \eta \vdash a + \mathsf{offset}(s, f) \rhd K$ $(a \neq 0, a : \mathsf{struct}\ s)$
$H\ ;\ S\ ;\ \eta \vdash a \rhd (\&(\_.f)\ ,\ K)$	$\longrightarrow$	$\mathsf{exception}(\mathsf{mem})$ $(a = 0)$
$H\ ;\ S\ ;\ \eta \vdash \&(e_1[e_2]) \rhd K$	$\longrightarrow$	$H\ ;\ S\ ;\ \eta \vdash e_1 \rhd (\&(\_[e_2])\ ,\ K)$
$H\ ;\ S\ ;\ \eta \vdash a \rhd (\&(\_[e_2])\ ,\ K)$	$\longrightarrow$	$H\ ;\ S\ ;\ \eta \vdash e_2 \rhd (\&(a[\_]\ ,\ K)$
$H\ ;\ S\ ;\ \eta \vdash i \rhd (\&(a[\_]\ ,\ K)$	$\longrightarrow$	$H\ ;\ S\ ;\ \eta \vdash a + i\lvert\tau\rvert \rhd K$ $a \neq 0, 0 \leq i < \mathsf{length}(a), a : \tau[\,]$
$H\ ;\ S\ ;\ \eta \vdash i \rhd (\&(a[\_]\ ,\ K)$	$\longrightarrow$	$\mathsf{exception}(\mathsf{mem})$ $a = 0$ or $i < 0$ or $i \geq \mathsf{length}(a)$

x = (*L).B.y;

$$
\begin{array}{lll}
 & H\ ;\ S\ ;\ \eta \vdash \mathsf{assign}(x, (*L).B.y) & \blacktriangleright\ K \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash ((*L).B.y) & \rhd(\mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash *(\&((*L).B.y)) & \rhd(\mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&((*L).B.y) & \rhd(*(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K)
\end{array}
$$

$$
\begin{array}{rcll}
H\ ;\ S\ ;\ \eta \vdash e.f \rhd K & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash *(\&(e.f)) \rhd K & \\
H\ ;\ S\ ;\ \eta \vdash \&(*e) \rhd K & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash e \rhd K & \\
H\ ;\ S\ ;\ \eta \vdash \&(e.f) \rhd K & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&e \rhd (\&(\_.f)\ ,\ K) & \\
H\ ;\ S\ ;\ \eta \vdash a \rhd (\&(\_.f)\ ,\ K) & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash a + \mathsf{offset}(s, f) \rhd K & \\
 & & & (a \neq 0, a : \mathsf{struct}\ s) \\
H\ ;\ S\ ;\ \eta \vdash a \rhd (\&(\_.f)\ ,\ K) & \longrightarrow & \mathsf{exception}(\mathsf{mem}) & (a = 0) \\
H\ ;\ S\ ;\ \eta \vdash \&(e_1[e_2]) \rhd K & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash e_1 \rhd (\&(\_[e_2])\ ,\ K) & \\
H\ ;\ S\ ;\ \eta \vdash a \rhd (\&(\_[e_2])\ ,\ K) & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash e_2 \rhd (\&(a[\_]\ ,\ K) & \\
H\ ;\ S\ ;\ \eta \vdash i \rhd (\&(a[\_]\ ,\ K) & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash a + i|\tau| \rhd K & \\
 & & & a \neq 0, 0 \leq i < \mathsf{length}(a), a : \tau[\,] \\
H\ ;\ S\ ;\ \eta \vdash i \rhd (\&(a[\_]\ ,\ K) & \longrightarrow & \mathsf{exception}(\mathsf{mem}) & \\
 & & & a = 0 \text{ or } i < 0 \text{ or } i \geq \mathsf{length}(a)
\end{array}
$$

x = (*L).B.y;

$$
\begin{array}{lll}
 & H\ ;\ S\ ;\ \eta \vdash \mathsf{assign}(x, (*L).B.y) & \blacktriangleright\ K \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash ((*L).B.y) & \rhd(\mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash *(\&((*L).B.y)) & \rhd(\mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&((*L).B.y) & \rhd(*(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&((*L).B) & \rhd(\&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K)
\end{array}
$$

$$
\begin{array}{lcll}
H\ ;\ S\ ;\ \eta \vdash e.f \rhd K & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash *(\&(e.f)) \rhd K & \\
H\ ;\ S\ ;\ \eta \vdash \&(*e) \rhd K & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash e \rhd K & \\
H\ ;\ S\ ;\ \eta \vdash \&(e.f) \rhd K & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&e \rhd (\&(\_.f)\ ,\ K) & \\
H\ ;\ S\ ;\ \eta \vdash a \rhd (\&(\_.f)\ ,\ K) & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash a + \mathsf{offset}(s, f) \rhd K & \\
 & & & (a \neq 0, a : \mathsf{struct}\ s) \\
H\ ;\ S\ ;\ \eta \vdash a \rhd (\&(\_.f)\ ,\ K) & \longrightarrow & \mathsf{exception}(\mathsf{mem}) & (a = 0) \\
H\ ;\ S\ ;\ \eta \vdash \&(e_1[e_2]) \rhd K & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash e_1 \rhd (\&(\_[e_2])\ ,\ K) & \\
H\ ;\ S\ ;\ \eta \vdash a \rhd (\&(\_[e_2])\ ,\ K) & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash e_2 \rhd (\&(a[\_]\ ,\ K) & \\
H\ ;\ S\ ;\ \eta \vdash i \rhd (\&(a[\_]\ ,\ K) & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash a + i|\tau| \rhd K & \\
 & & & a \neq 0, 0 \leq i < \mathsf{length}(a), a : \tau[\,] \\
H\ ;\ S\ ;\ \eta \vdash i \rhd (\&(a[\_]\ ,\ K) & \longrightarrow & \mathsf{exception}(\mathsf{mem}) & \\
 & & & a = 0 \text{ or } i < 0 \text{ or } i \geq \mathsf{length}(a)
\end{array}
$$

x = (*L).B.y;

$$
\begin{array}{lll}
 & H\ ;\ S\ ;\ \eta \vdash \mathsf{assign}(x, (*L).B.y) & \blacktriangleright\ K \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash ((*L).B.y) & \rhd(\mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash *(\&((*L).B.y)) & \rhd(\mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&((*L).B.y) & \rhd(*(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&((*L).B) & \rhd(\&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&(*L) & \rhd(\&(\_.B)\ ,\ \&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K)
\end{array}
$$

$$
\begin{array}{lcll}
H\ ;\ S\ ;\ \eta \vdash e.f \rhd K & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash *(\&(e.f)) \rhd K & \\
H\ ;\ S\ ;\ \eta \vdash \&(*e) \rhd K & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash e \rhd K & \\
H\ ;\ S\ ;\ \eta \vdash \&(e.f) \rhd K & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&e \rhd (\&(\_.f)\ ,\ K) & \\
H\ ;\ S\ ;\ \eta \vdash a \rhd (\&(\_.f)\ ,\ K) & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash a + \mathsf{offset}(s, f) \rhd K & \\
 & & & (a \neq 0, a : \mathsf{struct}\ s) \\
H\ ;\ S\ ;\ \eta \vdash a \rhd (\&(\_.f)\ ,\ K) & \longrightarrow & \mathsf{exception}(\mathsf{mem}) & (a = 0) \\
H\ ;\ S\ ;\ \eta \vdash \&(e_1[e_2]) \rhd K & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash e_1 \rhd (\&(\_[e_2])\ ,\ K) & \\
H\ ;\ S\ ;\ \eta \vdash a \rhd (\&(\_[e_2])\ ,\ K) & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash e_2 \rhd (\&(a[\_]\ ,\ K) & \\
H\ ;\ S\ ;\ \eta \vdash i \rhd (\&(a[\_]\ ,\ K) & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash a + i|\tau| \rhd K & \\
 & & a \neq 0, 0 \leq i < \mathsf{length}(a), a : \tau[\,] & \\
H\ ;\ S\ ;\ \eta \vdash i \rhd (\&(a[\_]\ ,\ K) & \longrightarrow & \mathsf{exception}(\mathsf{mem}) & \\
 & & a = 0 \text{ or } i < 0 \text{ or } i \geq \mathsf{length}(a) &
\end{array}
$$

x = (*L).B.y;

$$
\begin{array}{ll}
 & H\ ;\ S\ ;\ \eta \vdash \mathsf{assign}(x, (*L).B.y) \quad \blacktriangleright\ K \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash ((*L).B.y) \quad \rhd(\mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash *(\&((*L).B.y)) \quad \rhd(\mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&((*L).B.y) \quad \rhd(*(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&((*L).B) \quad \rhd(\&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&(*L) \quad \rhd(\&(\_.B)\ ,\ \&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash L \quad \rhd(\&(\_.B)\ ,\ \&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K)
\end{array}
$$

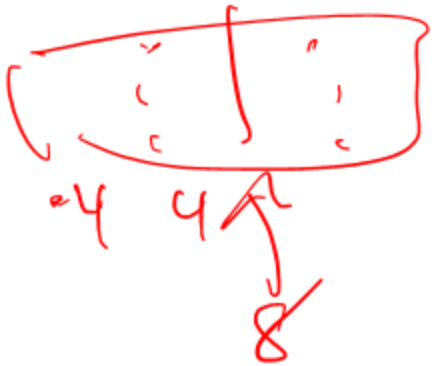
$$
\begin{array}{lcl}
H\ ;\ S\ ;\ \eta \vdash e.f \rhd K & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash *(\&(e.f)) \rhd K \\
H\ ;\ S\ ;\ \eta \vdash \&(*e) \rhd K & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash e \rhd K \\
H\ ;\ S\ ;\ \eta \vdash \&(e.f) \rhd K & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&e \rhd (\&(\_.f)\ ,\ K) \\
H\ ;\ S\ ;\ \eta \vdash a \rhd (\&(\_.f)\ ,\ K) & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash a + \mathsf{offset}(s, f) \rhd K \\
 & & (a \neq 0, a : \mathsf{struct}\ s) \\
H\ ;\ S\ ;\ \eta \vdash a \rhd (\&(\_.f)\ ,\ K) & \longrightarrow & \mathsf{exception}(\mathsf{mem}) \qquad (a = 0) \\
H\ ;\ S\ ;\ \eta \vdash \&(e_1[e_2]) \rhd K & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash e_1 \rhd (\&(\_[e_2])\ ,\ K) \\
H\ ;\ S\ ;\ \eta \vdash a \rhd (\&(\_[e_2])\ ,\ K) & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash e_2 \rhd (\&(a[\_]\ ,\ K) \\
H\ ;\ S\ ;\ \eta \vdash i \rhd (\&(a[\_]\ ,\ K) & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash a + i|\tau| \rhd K \\
 & & a \neq 0, 0 \leq i < \mathsf{length}(a), a : \tau[\,] \\
H\ ;\ S\ ;\ \eta \vdash i \rhd (\&(a[\_]\ ,\ K) & \longrightarrow & \mathsf{exception}(\mathsf{mem}) \\
 & & a = 0 \text{ or } i < 0 \text{ or } i \geq \mathsf{length}(a)
\end{array}
$$

x = (*L).B.y;

$$
\begin{array}{ll}
 & H\ ;\ S\ ;\ \eta \vdash \mathsf{assign}(x, (*L).B.y) \quad \blacktriangleright\ K \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash ((*L).B.y) \quad \rhd(\mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash *(\&((*L).B.y)) \quad \rhd(\mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&((*L).B.y) \quad \rhd(*(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&((*L).B) \quad \rhd(\&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&(*L) \quad \rhd(\&(\_.B)\ ,\ \&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash L \quad \rhd(\&(\_.B)\ ,\ \&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash a \quad \rhd(\&(\_.B)\ ,\ \&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
 & \qquad (\textit{given that } H\ ;\ S\ ;\ \eta(L) = a, a \neq 0)
\end{array}
$$

$H\ ;\ S\ ;\ \eta \vdash a \rhd (\&(\_.f)\ ,\ K)$	$\longrightarrow$	$H\ ;\ S\ ;\ \eta \vdash a + \mathsf{offset}(s, f) \rhd K$	$(a \neq 0, a : \mathsf{struct}\ s)$
$H\ ;\ S\ ;\ \eta \vdash a \rhd (\&(\_.f)\ ,\ K)$	$\longrightarrow$	$\mathsf{exception}(\mathsf{mem})$	$(a = 0)$

x = (*L).B.y;



$$
\begin{array}{lll}
 & H\ ;\ S\ ;\ \eta \vdash \mathsf{assign}(x, (*L).B.y) & \blacktriangleright\ K \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash ((*L).B.y) & \triangleright(\mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash *(\&((*L).B.y)) & \triangleright(\mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&((*L).B.y) & \triangleright(*(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&((*L).B) & \triangleright(\&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&(*L) & \triangleright(\&(\_.B)\ ,\ \&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash L & \triangleright(\&(\_.B)\ ,\ \&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash a & \triangleright(\&(\_.B)\ ,\ \&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
 & & (\textit{given that } H\ ;\ S\ ;\ \eta(L) = a, a \neq 0) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash a + 8 & \triangleright(\&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
 & & (\textit{given that } \mathsf{offset}(\mathsf{line}, B) = 8)
\end{array}
$$

$$
\begin{array}{lll}
H\ ;\ S\ ;\ \eta \vdash a \triangleright (\&(\_.f)\ ,\ K) & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash a + \mathsf{offset}(s, f) \triangleright K \\
 & & (a \neq 0, a : \mathsf{struct}\ s) \\
H\ ;\ S\ ;\ \eta \vdash a \triangleright (\&(\_.f)\ ,\ K) & \longrightarrow & \mathsf{exception}(\mathsf{mem}) \qquad (a = 0)
\end{array}
$$

x = (*L).B.y;

$$
\begin{array}{lll}
 & H\ ;\ S\ ;\ \eta \vdash \mathsf{assign}(x, (*L).B.y) & \blacktriangleright\ K \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash ((*L).B.y) & \triangleright(\mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash *(\&((*L).B.y)) & \triangleright(\mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&((*L).B.y) & \triangleright(*(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&((*L).B) & \triangleright(\&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&(*L) & \triangleright(\&(\_.B)\ ,\ \&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash L & \triangleright(\&(\_.B)\ ,\ \&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash a & \triangleright(\&(\_.B)\ ,\ \&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
 & & (\textit{given that}\ H\ ;\ S\ ;\ \eta(L) = a, a \neq 0) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash a + 8 & \triangleright(\&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
 & & (\textit{given that}\ \mathsf{offset}(\mathsf{line}, B) = 8) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash a + 12 & \triangleright *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K \\
 & & (\textit{given that}\ \mathsf{offset}(\mathsf{point}, y) = 4)
\end{array}
$$

$$
\begin{array}{lll}
H\ ;\ S\ ;\ \eta \vdash a \triangleright (\&(\_.f)\ ,\ K) & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash a + \mathsf{offset}(s, f) \triangleright K \\
 & & (a \neq 0, a : \mathsf{struct}\ s) \\
H\ ;\ S\ ;\ \eta \vdash a \triangleright (\&(\_.f)\ ,\ K) & \longrightarrow & \mathsf{exception}(\mathsf{mem}) \qquad (a = 0)
\end{array}
$$

x = (*L).B.y;

$$
\begin{array}{ll}
 & H\ ;\ S\ ;\ \eta \vdash \mathsf{assign}(x, (*L).B.y) \quad \blacktriangleright\ K \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash ((*L).B.y) \quad \triangleright(\mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash *(\&((*L).B.y)) \quad \triangleright(\mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&((*L).B.y) \quad \triangleright(*(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&((*L).B) \quad \triangleright(\&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&(*L) \quad \triangleright(\&(\_.B)\ ,\ \&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash L \quad \triangleright(\&(\_.B)\ ,\ \&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash a \quad \triangleright(\&(\_.B)\ ,\ \&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
 & \qquad\qquad (\textit{given that}\ H\ ;\ S\ ;\ \eta(L) = a, a \neq 0) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash a + 8 \quad \triangleright(\&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
 & \qquad\qquad (\textit{given that}\ \mathsf{offset}(\mathsf{line}, B) = 8) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash a + 12 \quad \triangleright *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K \\
 & \qquad\qquad (\textit{given that}\ \mathsf{offset}(\mathsf{point}, y) = 4) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash c \quad \triangleright(\mathsf{assign}(x, \_)\ ,\ K) \\
 & \qquad\qquad (\textit{given that}\ H(a + 12) = c)
\end{array}
$$

x = (*L).B.y;

$$
\begin{array}{lll}
 & H\ ;\ S\ ;\ \eta \vdash \mathsf{assign}(x, (*L).B.y) & \blacktriangleright\ K \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash ((*L).B.y) & \triangleright(\mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash *(\&((*L).B.y)) & \triangleright(\mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&((*L).B.y) & \triangleright(*(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&((*L).B) & \triangleright(\&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&(*L) & \triangleright(\&(\_.B)\ ,\ \&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash L & \triangleright(\&(\_.B)\ ,\ \&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash a & \triangleright(\&(\_.B)\ ,\ \&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
 & & (\textit{given that}\ H\ ;\ S\ ;\ \eta(L) = a, a \neq 0) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash a + 8 & \triangleright(\&(\_.y)\ ,\ *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K) \\
 & & (\textit{given that}\ \mathsf{offset}(\mathsf{line}, B) = 8) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash a + 12 & \triangleright *(\_)\ ,\ \mathsf{assign}(x, \_)\ ,\ K \\
 & & (\textit{given that}\ \mathsf{offset}(\mathsf{point}, y) = 4) \\
\longrightarrow & H\ ;\ S\ ;\ \eta \vdash c & \triangleright(\mathsf{assign}(x, \_)\ ,\ K) \\
 & & (\textit{given that}\ H(a + 12) = c) \\
\longrightarrow & H\ ;\ S\ ;\ \eta[x \mapsto c] \vdash \mathsf{nop} & \blacktriangleright\ K
\end{array}
$$

Allocation

- Similar to regular alloc, but size is defined by the struct, as per alignment rules.
- Initialization (for L4) is to set all to 0.

$$H\ ;\ S\ ;\ \eta \vdash \mathsf{alloc\_struct}(\mathsf{struct}\ s) \triangleright K \quad \longrightarrow \quad H'\ ;\ S\ ;\ \eta \vdash a \triangleright K$$

$$\begin{array}{rcl} a & = & H(\mathsf{next}) \\ z & = & \mathsf{sizeof}(\mathsf{struct}\ s) \\ H' & = & H[a \mapsto 0, \ldots, a + (z-1) \mapsto 0, \mathsf{next} \mapsto a + z] \end{array}$$

From last lecture

Assignment to Array

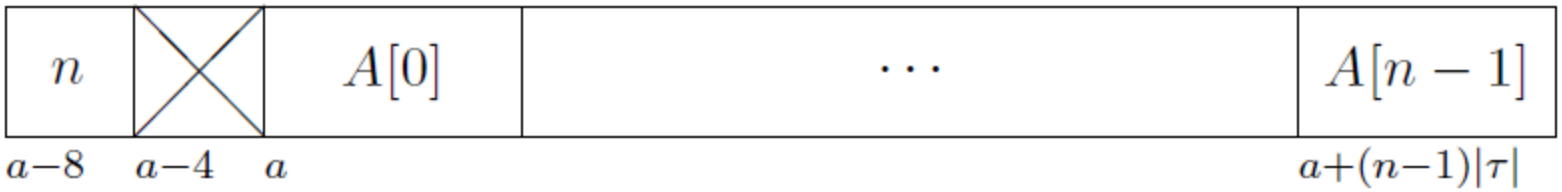
$$H\ ;\ S\ ;\ \eta \vdash \mathsf{assign}(d[e_2], e_3) \blacktriangleright K \longrightarrow H\ ;\ S\ ;\ \eta \vdash d \triangleright (\mathsf{assign}(\_[e_2], e_3)\ ,\ K)$$

$$H\ ;\ S\ ;\ \eta \vdash a \triangleright (\mathsf{assign}(\_[e_2], e_3)\ ,\ K) \longrightarrow H\ ;\ S\ ;\ \eta \vdash e_2 \triangleright (\mathsf{assign}(a[\_], e_3)\ ,\ K)$$

$$H\ ;\ S\ ;\ \eta \vdash i \triangleright (\mathsf{assign}(a[\_], e_3)\ ,\ K) \longrightarrow H\ ;\ S\ ;\ \eta \vdash e_3 \triangleright (\mathsf{assign}(a + i|\tau|, \_)\ ,\ K)$$
$$a \neq 0, 0 \leq i < \mathsf{length}(a), a : \tau[\,]$$

$$\longrightarrow \mathsf{exception}(\mathsf{mem})$$
$$a = 0 \text{ or } i < 0 \text{ or } i \geq \mathsf{length}(a)$$

$$H\ ;\ S\ ;\ \eta \vdash c \triangleright (\mathsf{assign}(b, \_)\ ,\ K) \longrightarrow H[b \mapsto c]\ ;\ S\ ;\ \eta \vdash \mathsf{nop} \blacktriangleright K$$



Assignment

- Can simplify all rules for assignment to heap allocated locations

$$
\begin{array}{lcll}
H\ ;\ S\ ;\ \eta \vdash \mathsf{assign}(d, e) \blacktriangleright K & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash \&d \rhd (\mathsf{assign}(\_, e)\ ,\ K) & (d \neq x) \\
H\ ;\ S\ ;\ \eta \vdash a \rhd (\mathsf{assign}(\_, e)\ ,\ K) & \longrightarrow & H\ ;\ S\ ;\ \eta \vdash e \rhd (\mathsf{assign}(a, \_)\ ,\ K) & \\
H\ ;\ S\ ;\ \eta \vdash v \rhd (\mathsf{assign}(a, \_)\ ,\ K) & \longrightarrow & H[a \mapsto v]\ ;\ S\ ;\ \eta \vdash \mathsf{nop} \blacktriangleright K & (a \neq 0) \\
H\ ;\ S\ ;\ \eta \vdash v \rhd (\mathsf{assign}(a, \_)\ ,\ K) & \longrightarrow & \mathsf{exception}(\mathsf{mem}) & (a = 0)
\end{array}
$$

- Likewise, can simplify assignment ops

$d \odot = e$

- When d is a small type, e.g., a variable *x*, elaborate to *assign(x,x⊙e)*
- When d is an address on the heap elaborate to *asnop(d,⊙,e)* and we have:

$$H\ ;\ S\ ;\ \eta \vdash \mathsf{asnop}(d, \odot, e) \blacktriangleright K \longrightarrow H\ ;\ S\ ;\ \eta \vdash \&d \rhd (\mathsf{asnop}(\_, \odot, e)\ ,\ K)$$
$$H\ ;\ S\ ;\ \eta \vdash a \rhd (\mathsf{asnop}(\_, \odot, e)\ ,\ K) \longrightarrow H\ ;\ S\ ;\ \eta \vdash e \rhd (\mathsf{asnop}(a, \odot, \_)\ ,\ K)$$
$$H\ ;\ S\ ;\ \eta \vdash v \rhd (\mathsf{asnop}(a, \odot, \_)\ ,\ K) \longrightarrow H\ ;\ S\ ;\ \eta \vdash \mathsf{assign}(a, *a \odot v) \blacktriangleright K$$

  - evaluation of address
  - evaluation of value on right-hand side
  - assignment (which continues as before)

Registers and small types

- two type sizes held in registers
  - 32-bit: int & bool
  - 64-bit: pointer
- Be careful moving these around
  - **`movl`**, **`cmpl`**, **`addl`** vs. **`movq`**, **`cmpq`**, **`addq`**
- Track stack space, heap space, etc. required
- Suggestions:
  - track sizes for your temps, vars, etc.
  - use explicit extend ops in your IR

32/64-bit implementation

- Use explicit extend ops in IR, e.g.,

  dest64 <- zeroextend src32

  dest64 <- signextend src32

- Remember, zeroextend comes for free:
  - **`movl %eax, %eax`**

    sets high order 32 bits of **`%rax`** to 0!
  - similarly for other instructions

Starting Thursday analysis and optimization