Data

- "Atomic" data: numbers, booleans, strings...

Data

- "Atomic" data: numbers, booleans, strings...
- Compound data: posns, structures, arrays, ...

Data

- "Atomic" data: numbers, booleans, strings...
- Compound data: posns, structures, arrays, ...
- Variants:

```
; A list is either...

; A grade is either...

; An animal is either...
```

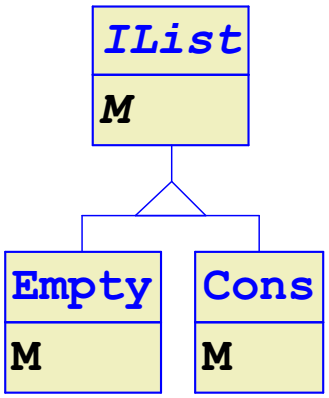
Variants

Racket:

```
; A list is either
;  - empty
;  - (cons num list)
```

```
; F : list -> ...
(define (F l)
  (cond
    [(empty? l) ...]
    [(cons? l) ...]))
```

Java:



C:

```
struct cons {
  int val;
  struct cons * next;
};

... f(struct cons * l) {
  if (l == NULL)
    ... /* empty */
  else
    ... /* cons */
}
```

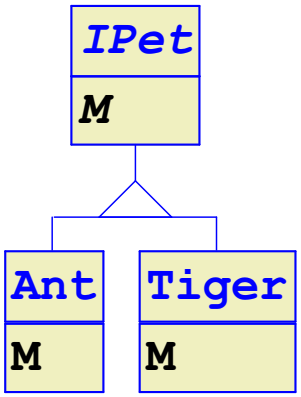
Variants

Racket:

```
; A pet is either
;  - (make-ant num posn)
;  - (make-tiger num sym)
(define-struct ant (weight loc))
(define-struct tiger (weight name))
```

```
; F : pet -> ...
(define (F p)
  (cond
    [(ant? p) ...]
    [(tiger? p) ...]))
```

Java:

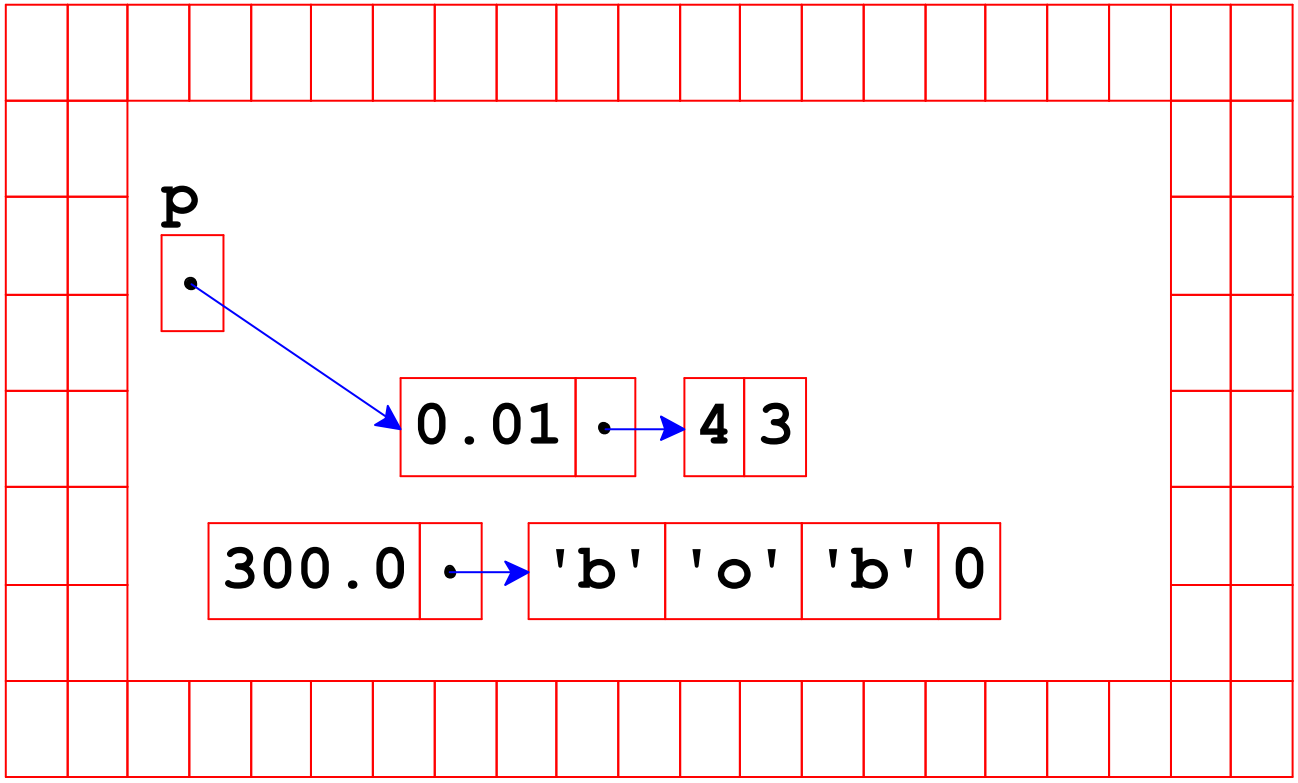


C:

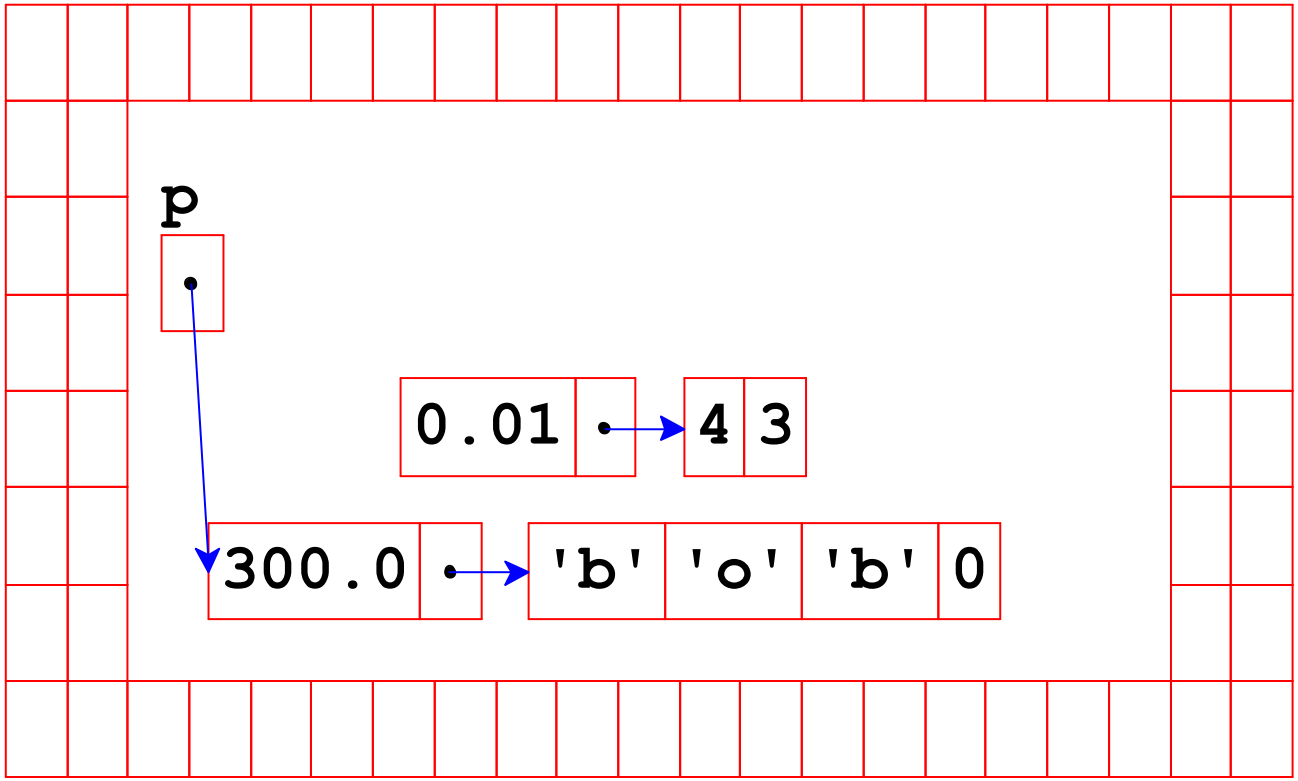
```
struct ant {
  double weight;
  posn* loc;
};

struct tiger {
  double weight;
  char* name;
};
```

```
... f(??? p) {
  if (???)
    ... /* ant */
  else
    ... /* tiger */
}
```

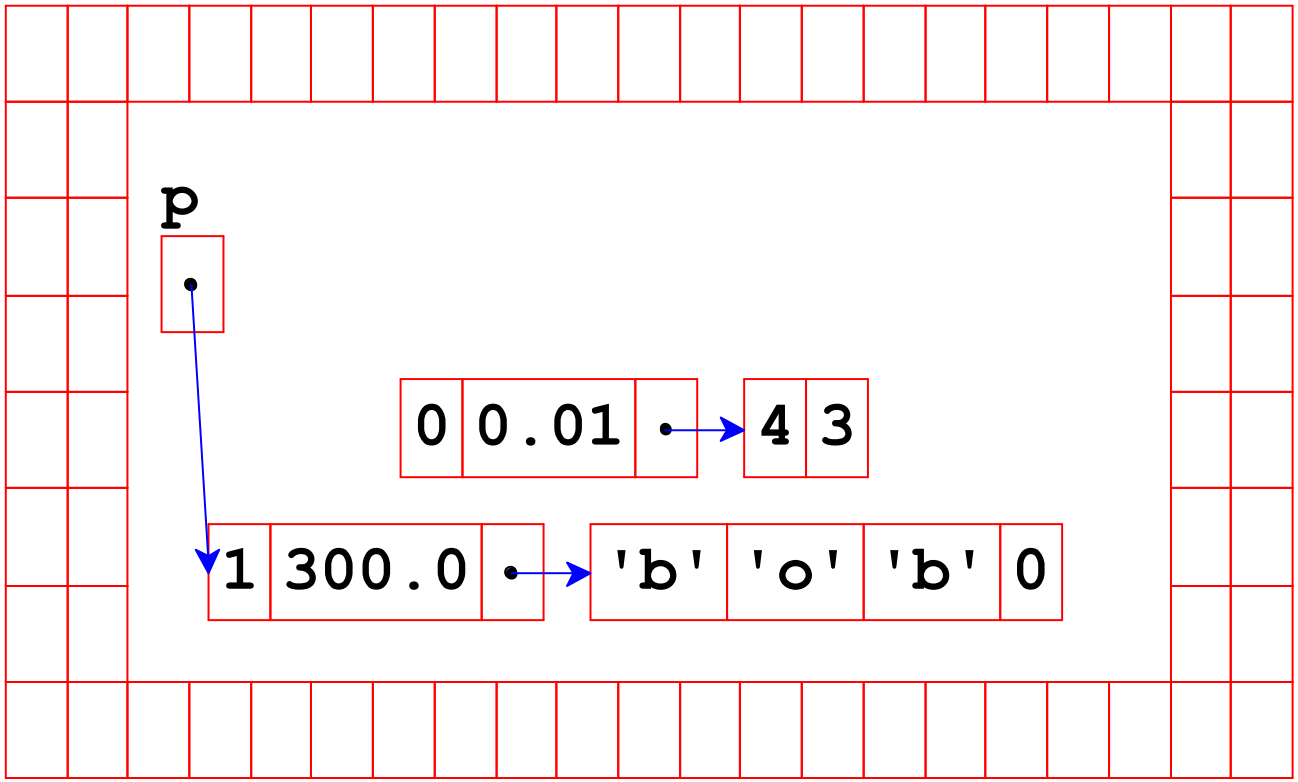
Variants



Variants



Variants



Tagging Variants

```
enum {
  ant_type,
  tiger_type
};

typedef struct ant {
  int tag; /* = ant_type */
  double weight;
  posn* loc;
} ant;

typedef struct tiger {
  int tag; /* = tiger_type */
  double weight;
  char* name;
} tiger;
```

```
... f(void* p) {
  if (*(int *)p == ant_type) {
    ant* a = (ant*)p;
    ...
  } else {
    tiger* t = (tiger*)p;
    ...
  }
}
```

Shared Variant Header

```
enum {
  ant_type,
  tiger_type
};

typedef struct pet {
  int tag;
  double weight;
} pet;

typedef struct ant {
  pet p;
  posn* loc;
} ant;

typedef struct tiger {
  pet p;
  char* name;
} tiger;
```

```
... f(pet* p) {
  if (p->tag == ant_type) {
    ant* a = (ant*)p;
    ...
  } else {
    tiger* t = (tiger*)p;
    ...
  }
}
```

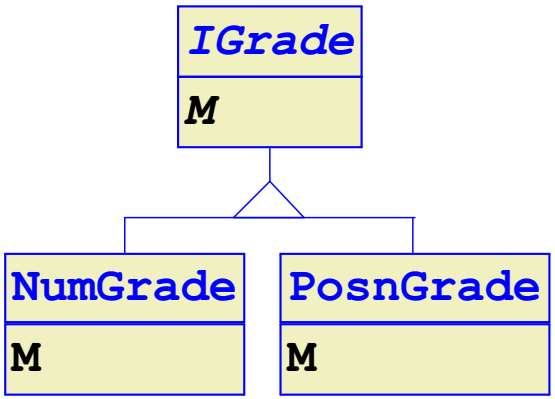
Translating Non-Structure Variants

Racket:

```
; A grade is either
;  - num
;  - (make-posn num num)
```

```
; F : grade -> ...
(define (F g)
  (cond
    [(number? g) ...]
    [(posn? g) ...]))
```

Java:



C:

```
typedef struct grade {
  int tag;
} grade;

struct num_grade {
  grade g;
  double n;
};

struct posn_grade {
  grade g;
  posn* p;
};
```

Example

See **`shape.c`**

Function Pointers as Methods

See **`shape_obj.c`**