

Compound Data

```
#include <stdio.h>

struct posn {
    int x;
    int y;
}; /* <- don't forget the semicolon! */

int main() {
    struct posn p = { 10, 20 };

    printf("Posn is at (%d, %d)\n", p.x, p.y);

    return 0;
}
```

[Copy](#)

Functions on Structures

```
struct posn flip_posn(struct posn p) {
    struct posn p2;
    p2.x = p.y;
    p2.y = p.x;
    return p2;
}

int main() {
    struct posn p = { 10, 20 };

    p = flip_posn(p);
    printf("Posn is at (%d, %d)\n", p.x, p.y);

    return 0;
}
```

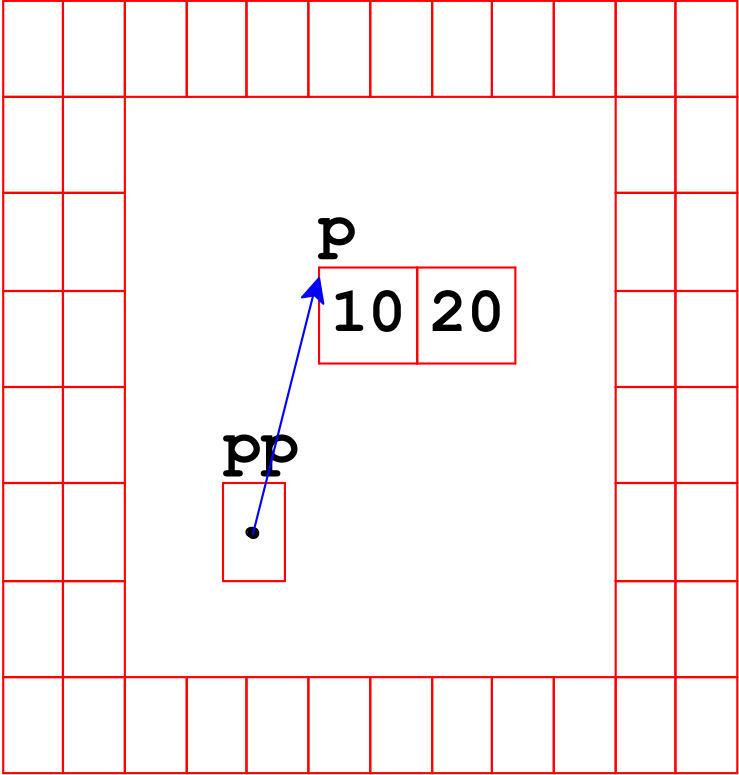
Structure Pointers

```
void flip_this_posn(struct posn * pp) {
    int z;
    z = (*pp).x;
    (*pp).x = (*pp).y;
    (*pp).y = z;
}

int main() {
    struct posn p = { 10, 20 };

    flip_this_posn(&p);
    printf("Posn is at (%d, %d)\n", p.x, p.y);

    return 0;
}
```



Arrow Notation

```
void flip_this_posn(struct posn * pp) {  
    int z;  
    z = pp->x; /* same as z = (*pp).z */  
    pp->x = pp->y;  
    pp->y = z;  
}
```

[Copy](#)

-> abbreviates a combination of * and .

Tests

```
void flip_this_posn(struct posn * p) { .... }

void check_int(int a, int b) {
    if (a != b) {
        printf("failed\n");
        exit(1);
    }
}

void tests() {
    struct posn p1 = { 7, 3 };

    check_int(p1.x, 7);
    check_int(p1.y, 3);
    flip_this_posn(&p1);
    check_int(p1.x, 3);
    check_int(p1.y, 7);
}

int main() {
    struct posn p = { 10, 20 };
    tests();
    flip_this_posn(&p);
    ...
}
```

[Copy](#)

Another Structure Example

```
struct snake {  
    char* name;  
    double weight;  
    char* food;  
};
```

[Copy](#)

Implement

```
struct snake feed_snake(struct snake s);  
void feed_this_snake(struct snake * s);
```

Nested Structures

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

[Copy](#)

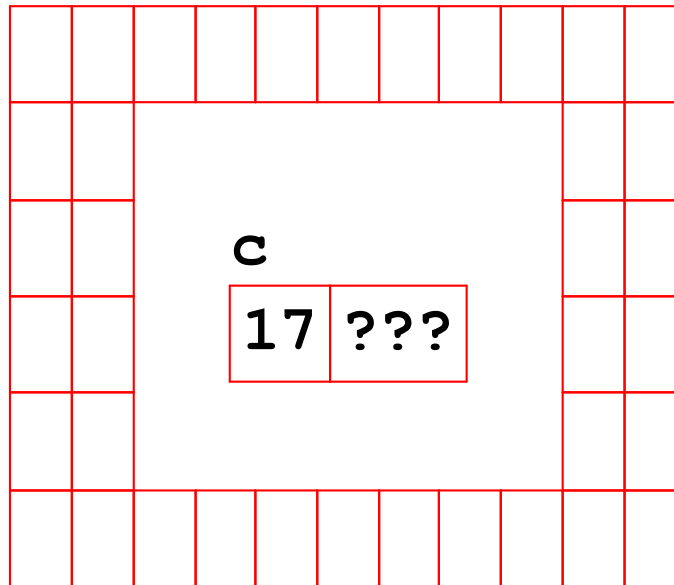
Implement

```
struct ant send_ant_home(struct ant a);
```


Cons Cells

```
struct int_cons {  
    int first;  
    struct list rest; /* ??? */  
};
```

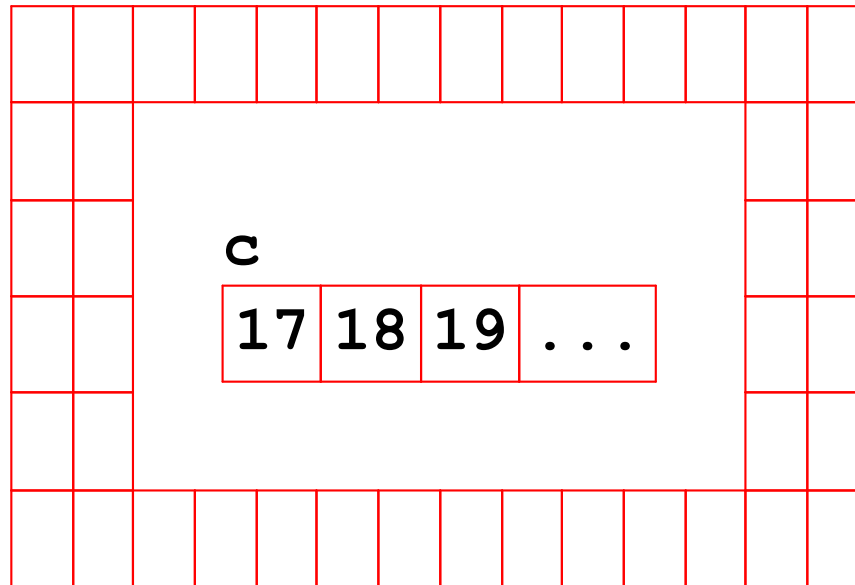
```
struct int_cons c = { 17, ??? };
```



Cons Cells

```
struct int_cons {  
    int first;  
    struct int_cons rest; /* ??? */  
};
```

```
struct int_cons c = { 17, { 18, { 19, ... } } };
```



Lists

```
struct int_cons {  
    int first;  
    struct int_cons * rest; /* NULL => empty */  
};
```

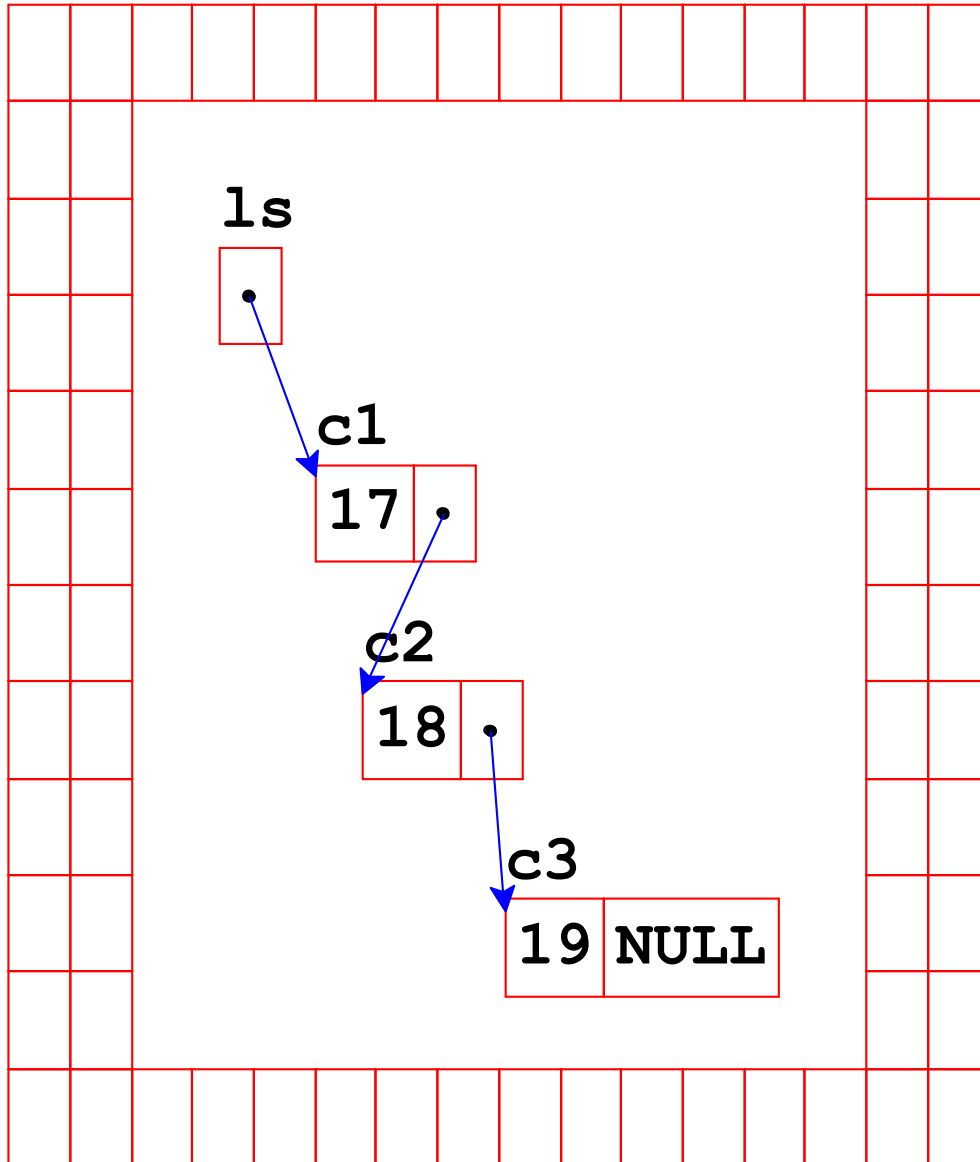
[Copy](#)

Lists

```
void print_list(struct int_cons * ls) {  
    ...  
}
```

```
int main() {  
    struct int_cons c3 = { 19, NULL };  
    struct int_cons c2 = { 18, &c3 };  
    struct int_cons c1 = { 17, &c2 };  
    struct int_cons * ls = &c1;  
  
    print_list(ls);  
  
    return 0;  
}
```

[Copy](#)



Lists

```
void print_list(struct int_cons * ls) {  
    if (ls == NULL) {  
    } else {  
        printf("%d\n", ls->first);  
        print_list(ls->rest);  
    }  
}
```

```
int main() {  
    struct int_cons c3 = { 19, NULL };  
    struct int_cons c2 = { 18, &c3 };  
    struct int_cons c1 = { 17, &c2 };  
    struct int_cons * ls = &c1;  
  
    print_list(ls);  
  
    return 0;  
}
```

Lists

```
void print_list(struct int_cons * ls) {  
    for (; ls != NULL; ls = ls->rest) {  
        printf("%d\n", ls->first);  
    }  
}
```

```
int main() {  
    struct int_cons c3 = { 19, NULL };  
    struct int_cons c2 = { 18, &c3 };  
    struct int_cons c1 = { 17, &c2 };  
    struct int_cons * ls = &c1;  
  
    print_list(ls);  
  
    return 0;  
}
```

[Copy](#)

Building Lists

```
int main(int argc, char** argv) {
    struct int_cons * ls = NULL;
    int i;

    for (i = argc - 1; i > 0; i--) {
        ls = cons(atoi(argv[i]), ls);
    }

    print_list(ls);

    return 0;
}
```

[Copy](#)

A Bad Allocator

```
struct int_cons * cons(int i, struct int_cons * ls) {  
    struct int_cons c;  
  
    c.first = i;  
    c.rest = ls;  
  
    return &c;  
}
```

[Copy](#)

Memory is reserved for **c** only until **cons** returns

A Good Allocator

```
struct int_cons * cons(int i, struct int_cons * ls) {  
    struct int_cons * ls2;  
  
    ls2 = (struct int_cons *)malloc(sizeof(struct int_cons));  
  
    ls2->first = i;  
    ls2->rest = ls;  
  
    return ls2;  
}
```

[Copy](#)

Typedefs

```
typedef struct int_cons * list;

list cons(int i, list ls) {
    list ls2;

    ls2 = (list)malloc(sizeof(struct int_cons));

    ls2->first = i;
    ls2->rest = ls;

    return ls2;
}
```

[Copy](#)

Beware: `malloc(sizeof(list))` would be wrong