struct pointers

The -> Operator

- A struct pointer is declared like any other pointer
  ```
  struct person *p1;
  Person *p2;
  ```
- The arrow operator -> is used when accessing struct members with a pointer

```
#include <stdlib.h>
typedef struct {
   char name;
   int id;
} Person;

int main() {
    Person person1, person2;  // memory allocated for 2 Person structs
    Person *p = NULL;  // Always initialize pointers! NULL is defined in stdlib.h
    p = &person1;     // Now p points to person1
    // The following 3 statements are equivalent:
    person1.id = 25;
    p->id = 25;
    (*p).id = 25;
}
```

Array Traversal

```
#include <stdio.h>
#include <stdlib.h>
#include "person.h"    // Assume Person and MAX_DATA are defined here

int main() {
    Person data[MAX_DATA];
    Person *p = NULL;

    // Assume array gets filled with data
    int i = 0;
    for (p = data; i < MAX_DATA; i++, p++)
       printf("name: %c, id: %d\n", p->name, p->id);
}
```

Function Parameter

```
#include <stdio.h>
#include <stdlib.h>
#include "person.h"   // Assume Person is defined here

Person slowUpdate(Person);
void fastUpdate(Person *);

int main() {
    Person person1 = {'a', 25};
    person1 = slowUpdate(person1);  // parameter and return structs copied to stack
    fastUpdate(&person1);    // just address of struct copied to stack as parameter
}

Person slowUpdate(Person person){
    person.id = 500;
    return person;
}

void fastUpdate(Person * p){
    p->id = 500;
}
```

Function Return Type

- A function may return a struct pointer

```
struct person *foo();
Person *foo();
```

- As always, make sure the pointer returned does not point to a local variable!