

Pointer of Structure

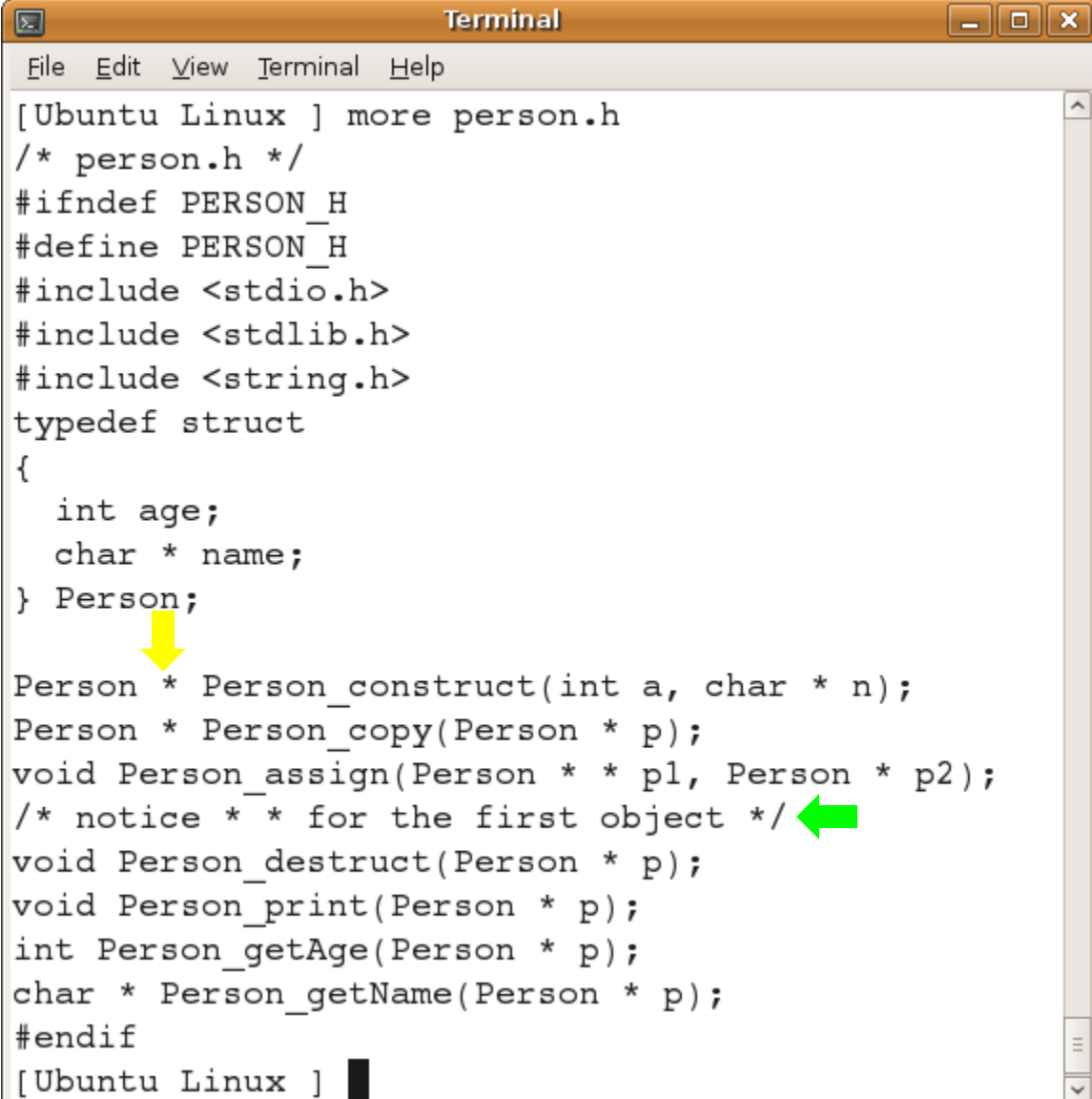
Yung-Hsiang Lu

Review Pointers

```
int * iptr;  
double * dptr;  
char * cptr;
```

How about

```
Person * pptr;  
⇒ no problem
```



```
Terminal  
File Edit View Terminal Help  
[Ubuntu Linux ] more person.h  
/* person.h */  
#ifndef PERSON_H  
#define PERSON_H  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
typedef struct  
{  
    int age;  
    char * name;  
} Person;  
  
Person * Person_construct(int a, char * n);  
Person * Person_copy(Person * p);  
void Person_assign(Person * * p1, Person * p2);  
/* notice * * for the first object */  
void Person_destruct(Person * p);  
void Person_print(Person * p);  
int Person_getAge(Person * p);  
char * Person_getName(Person * p);  
#endif  
[Ubuntu Linux ]
```

A terminal window titled "Terminal" with a menu bar (File, Edit, View, Terminal, Help). The window shows the output of the command "more person.h". The code displayed is the header file "person.h", which defines a "Person" struct with "age" (int) and "name" (char*) fields. It also declares several functions: "Person_construct", "Person_copy", "Person_assign", "Person_destruct", "Person_print", "Person_getAge", and "Person_getName". A yellow arrow points to the closing brace of the "Person" struct definition, and a green arrow points to the double asterisks in the comment "/* notice * * for the first object */". The terminal prompt "[Ubuntu Linux]" is visible at the bottom.

[Ubuntu Linux] more person.c

```
#include "person.h"
```

```
Person * Person_construct(int a, char * n) {
```

```
    Person * p = malloc(sizeof(Person));
```

two-step allocation

```
    p -> age = a;
```

```
    p -> name = malloc((strlen(n) + 1) * sizeof(char));
```

```
    strcpy(p -> name, n);
```

```
    return p;
```

use -> instead of .

```
}
```

```
Person * Person_copy(Person * p) {
```

```
    return Person_construct(p -> age, p -> name);
```

```
}
```

```
void Person_assign(Person * * p1, Person * p2) {
```

```
    /* notice * * for the first object */
```

```
    if ((*p1) == p2) { return; } /* must check first */
```

```
    Person_destruct (* p1);
```

```
    * p1 = Person_copy(p2);
```

```
}
```

```
void Person_destruct(Person * p) {
```

```
    free (p -> name);
```

```
    free (p);
```

**two-step release
symmetric to allocation**

```
}
```

```
void Person_print(Person * p) {
```

```
    printf("age= %d, name= %s\n", p -> age, p -> name);
```

```
}
```

```
int Person_getAge(Person * p) {
```

```
    return p -> age;
```

```
}
```

```
char * Person_getName(Person * p) {
```

```
    return p -> name;
```


```
}
```

Why * * for assign Function?

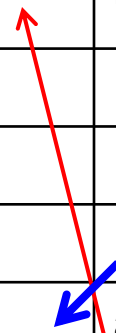
```
void Person_assign(Person * * p1, Person * p2) {  
    if ((*p1) == p2) { return; }  
    Person_destruct (* p1);  
    * p1 = Person_copy(p2);  
}
```

change where p1 points to

Address	Data
	'J'
	age
	name
& p1	



Address	Data
	'S'
& p1	
	age
	name



Will this work?

```
void Person_assign(Person * p1, Person * p2) {  
    if (p1 == p2) { return; }  
    p1 -> age = p2 -> age;  
    strcpy(p1 -> name, p2 -> name);  
}
```

**This will not work if
strlen(p1 -> name) < strlen(p2 -> name)**