August 26, 2020

1 Exercises

1. First, I need to justify if the following declarations legal on an individual basis:

```
struct {int x, y;} x;
struct {int x, y;} y;
```

The struct struct {int x, y;} x; is legal. struct {int x, y;} x; is equivalent to

```
struct {
    int x;
    int y;
} x;
```

and 'x' beside struct represents variable of that type. It is used to declare struct and access members of the struct (e.g. x.x., x.y).

The same is true for struct {int x, y;} y;.

Second, I need to answer if both declarations of struct can appear in a program.

The answer is yes. Each structure has a separate name space for it's members.

Notes

- Declaring Structure Variables
 - Struct can have many variables that represent the same struct



• Initializing Structure Variables

- Struct can be initialized with preset values (like python class under __init__)

```
struct {
  int number;
  char name[NAME_LEN+1];
  int on hand;
} part1 = {528, "Disk drive", 10},
  part2 = {914, "Printer cable", 5};
```

2. a) I need to declare structure variables named c1, c2 and c3, each having members real and imaginary of type double.

The solution to this problem is:

```
struct {
          double real, imaginary;
} c1, c2, c3;
```

- b) I need to modify the declaration in part a) so that
 - c1's members initially have the values 0.0 and 1.0
 - c2's members initially have the values 1.0 and 0.0
 - c3 is not initialized

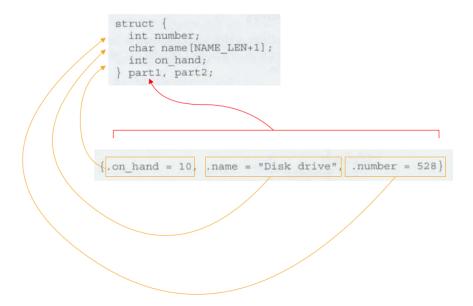
The solution to this problem is:

```
struct {
          double real, imaginary;
} c1 = {0.0, 1.0},
          c2 = {1.0, 0.0},
          c3;
```

Notes

- Designated Initializer
 - Allows specific member variable to be initialized
 - Allows member variables to be initialized in any order

Example



c) I need to write statements that copy the members of c2 to c1.

Copying the members of c2 and c1 can be done in one statement.

Below is the solution to this problem:

```
c2 = c1
```

d) I need to write statements that add the corresponding members of c1 and c2 and store the result in c3.

The solution to this problem is:

```
struct {
          double real, imaginary;
} c1 = {0.0, 1.0},
```

Notes

- member variables of struct contains two operators & and . (e.g &part1.number and part1.number)
- ullet accesses memory address of the member variable, where as . accesses value
- part1 = part2 <u>copies</u> contents in part2 to corresponding member variable in part1

```
struct {
  int number;
  char name[NAME_LEN+1];
  int on_hand;
} part1, part2;
```

3. a) I need to declare a tag named complex for a structure with two members real and imaginary, of type double

The solution to this problem is:

```
struct complex {
          double real, imaginary;
};
```

Notes

- Declaring a Structure Tag
 - allows to use struct in function calls
 - allows to use the same struct in multiple files of a program

```
struct part {
  int number;
  char name[NAME_LEN+1];
  int on_hand;
};
```

b) I need to use the complex tag to declare variables named c1, c2, c3.

The solution to this problem is:

```
struct complex {
    double real, imaginary;
} c1, c2, c3;
```

- c) I need to write a function named make_complex that satisfies the following:
 - The function make_complex should have two parameters (real, imaginary) of type double
 - The function make_complex should store the two arguments in complex struct
 - The function make_complex should return the struct

The solution to this problem is:

```
struct complex {
    double real, imaginary;
};

struct complex (double real, double imaginary) {
    struct complex c1;

c1.real = real;
    c1.imaginary = imaginary;

return c1;
}
```

Notes

- Declaring Variables Cont.
 - Once the struct tag is formed, it can be used to declare variables

Example

```
struct part part1, part2
```

- Structure tag can be combined with the declaration of structure variables.
 - * it's like creating a global variable (if it's outside of main), or local variable (if it's created inside a function) at the instant the struct is formed

```
struct part {
  int number;
  char name[NAME_LEN+1];
  int on_hand;
} part1, part2;
```

- Declared variables can set values the moment it's declared

Example

```
struct part part1 = {528, "Disk Drive", 10}
```

- d) I need to write a function named add_complex that satisfies the following:
 - The function add_complex should have 2 parameters of type struct complex
 - The function add_complex should add the corresponding members of its arguments
 - The function add_complex should return the result of type struct complex

The solution to this problem is:

```
struct complex {
    double real, imaginary;
};

struct complex add_complex (struct complex c1, struct complex c2)
{
    struct complex c3;

    c3.real = c1.real + c2.real;
    c3.imaginary = c1.imaginary + c2.imaginary;

    return c3;
}
```

4. I need to repeat exercise 3 but using a type named complex.

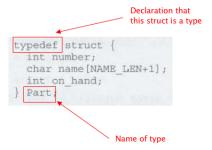
```
a)
       typedef struct {
           double real, imaginary;
       } Complex;
b)
       typedef struct {
           double real, imaginary;
 2
 3
       } Complex;
 4
 5
 6
       Complex c1, c2, c3;
c)
       typedef struct {
           double real, imaginary;
       } Complex;
 3
       Complex make_complex(double real, double imaginary) {
 5
           Complex c;
 6
           c.real = real;
```

```
c.imaginary = imaginary;
 9
10
           return c;
11
12
d
       typedef struct {
           double real, imaginary;
 2
       } Complex;
 3
       Complex add_complex(Complex c1, Complex c2) {
           Complex c3;
 6
           c3.real = c1.real + c2.real;
           c3.imaginary = c1.imaginary + c2.imaginary;
10
           return c3;
11
```

Notes

• Structure Type

- Is an alternative to declaring a structure tag



- Allows us to define a genuine type name

Example

```
Part part1, part2;
```

- Once declared, cannot define a structure tag with the same name