

CSC 209 Review 7 Solution

August 26, 2020

1 Exercises

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

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

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

```
1 struct {  
2     int x;  
3     int y;  
4 } 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 its members.

Notes

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

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

members of struct

variables that represent
this 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};
```

values
that initialize
structure variables

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:

```
1  struct {  
2      double real, imaginary;  
3  } 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:

```

1  struct {
2      double real, imaginary;
3  } c1 = {0.0, 1.0},
4      c2 = {1.0, 0.0},
5      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:

```

1  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:

```

1  struct {
2      double real, imaginary;
3  } c1 = {0.0, 1.0},

```

```

4      c2 = {1.0, 0.0},
5      c3;
6
7      ...
8
9      c3 = c1 + c2;

```

Notes

- member variables of struct contains two operators & and . (e.g &part1.number and part1.number)
- & accesses memory address of the member variable, where as . accesses value
- part1 = part2 copies 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:

```

1      struct complex {
2          double real, imaginary;
3      };

```

Notes

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

Structure tag



```

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:

```

1  struct complex {
2      double real, imaginary;
3  } 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:

```

1  struct complex {
2      double real, imaginary;
3  };
4
5  struct complex (double real, double imaginary) {
6      struct complex c1;
7
8      c1.real = real;
9      c1.imaginary = imaginary;
10
11     return c1;
12 }

```

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:

```
1  struct complex {
2      double real, imaginary;
3  };
4
5  struct complex add_complex (struct complex c1, struct complex c2)
6  {
7      struct complex c3;
8
9      c3.real = c1.real + c2.real;
10     c3.imaginary = c1.imaginary + c2.imaginary;
11
12     return c3;
13 }
```

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

```
a) typedef struct {
2     double real, imaginary;
3 } Complex;
```

```
b) typedef struct {
2     double real, imaginary;
3 } Complex;
4
5     ...
6
7     Complex c1, c2, c3;
```

```
c) typedef struct {
2     double real, imaginary;
3 } Complex;
4
5     Complex make_complex(double real, double imaginary) {
6         Complex c;
7
8         c.real = real;
```

```

9      c.imaginary = imaginary;
10
11      return c;
12  }

```

d)

```

1  typedef struct {
2      double real, imaginary;
3  } Complex;
4
5  Complex add_complex(Complex c1, Complex c2) {
6      Complex c3;
7
8      c3.real = c1.real + c2.real;
9      c3.imaginary = c1.imaginary + c2.imaginary;
10
11     return c3;
12 }

```

Notes

• Structure Type

- Is an alternative to declaring a structure tag

```

typedef struct {
    int number;
    char name[NAME_LEN+1];
    int on_hand;
} Part;

```

- Allows us to define a genuine type name

Example

```
Part part1, part2;
```

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

5. a) I need to write the following function

```
int day_of_year (struct date d);
```

satisfying the following requirements:

- The `date` structure should contain three members: `month`, `day`, and `year` all of type `int`
- The function `day_of_year` should return the day of the year (an integer between 1 and 366) that corresponds to the date `d`

The solution to this problem is:

```

1
2     struct date {
3         int month, day, year;
4     };
5
6     int day_of_year (struct date d);
7
8     ...
9
10    int day_of_year (struct date d) {
11        bool leap_year = false;
12        int days = 0, days_in_month[] = {
13            31, 28, 31, 30,
14            31, 30, 31, 31,
15            30, 31, 30, 31};
16
17        // check if it's the leap year
18        if (((d.year % 4 == 0) &&
19            (d.year % 100 != 0)) ||
20            (d.year % 400 == 0)) {
21
22            leap_year = true;
23        }
24
25
26        // add days from months
27        for (int i = 0; i < d.month; i++) {
28            if (i == d.month-1) {
29                days += d.day;
30                continue;
31            }
32
33            days += days_in_month[i];
34        }
35
36
37        // add 1 more day if month > 2 and it's leap year
38        if (leap_year && d.month > 2) {
39            days += 1;
40        }
41
42
43        // return days
44        return days;
45    }

```

b) I need to write the following function

```
int compare_dates (struct date d1, struct date d2);
```

satisfying the following requirements:

- The function `compare_dates` should return
 - -1 if `d1` is an earlier date than `d2`
 - +1 if `d1` is a later date than `d2`
 - 0 if `d1` and `d2` are the same

The solution to this problem is:

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <stdbool.h>
4
5  struct date {
6      int month, day, year;
7  };
8
9  int compare_dates (struct date d1, struct date d2);
10
11 int main(void) {
12     int day;
13     struct date d1 = {3, 1, 2019};
14     struct date d2 = {2, 2, 2019};
15
16     day = compare_dates(d1,d2);
17
18     printf("%d\n", day);
19
20     return 0;
21 }
22
23
24 int compare_dates (struct date d1, struct date d2) {
25     char s1[9], s2[9];
26
27     sprintf(s1,"%4d%2d%2d", d1.year, d1.month, d1.day);
28     sprintf(s2,"%4d%2d%2d", d2.year, d2.month, d2.day);
29
30     // return days
31     return strcmp(s1,s2);
32 }

```

6. I need to write the following function

```
struct time split_time(long total_seconds)
```

satisfying the following requirements:

- The `time` structure has three members `hours`, `minutes`, `seconds`
- The function `split_time` should return `time` struct containing equivalent time in `hours` (0-23), `minutes` (0-59), `seconds` (0-59)

The solution to this problem is:

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdbool.h>
4
5  struct time {
6      int hours, minutes, seconds;
7  };
8
9  struct time split_time (long total_seconds);
10
11  ...
12
13  struct time split_time (long total_seconds) {
14      struct time t;
15      int s = (int)total_seconds;
16
17      printf("%d\n", s);
18
19      t.hours = s / 3600;
20      s = s - (t.hours * 3600);
21
22      t.minutes = s / 60;
23      s = s - (t.minutes * 60);
24
25      t.seconds = s;
26
27      return t;
28  }
```

7. a) I need to write a function that satisfies the following requirements:

- The function should reduce the fraction `f` to lowest terms
- The struct `fraction` contains two members: `numerator` and `denominator`, and both are type `int`

The solution to this problem is:

```
1  #include <stdio.h>
2
3  struct fraction {
4      int numerator, denominator;
5  };
6
7  void reduce (struct fraction *f);
8
9  ...
10
11  void reduce (struct fraction *f) {
12
13      int gcd;
14      struct fraction *fPtr;
```

```
15     fPtr = f;
16
17     if (fPtr->numerator >= fPtr->denominator) {
18         gcd = fPtr->denominator;
19     } else {
20         gcd = fPtr->numerator;
21     }
22
23     while (gcd > 0) {
24         if ((fPtr->numerator % gcd == 0) &&
25             (fPtr->denominator % gcd == 0)) {
26             break;
27         }
28
29         gcd--;
30     }
31
32     fPtr->denominator = fPtr->denominator/gcd;
33     fPtr->numerator = fPtr->numerator/gcd;
34 }
35
```