

# CSC 209 Review 7 Solution

August 27, 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 struct `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  ...
10
11 int compare_dates (struct date d1, struct date d2) {
12     char s1[9], s2[9];
13
14     sprintf(s1,"%4d%2d%2d", d1.year, d1.month, d1.day);
15     sprintf(s2,"%4d%2d%2d", d2.year, d2.month, d2.day);
16
17     // return days
18     return strcmp(s1,s2);
19 }
```

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     struct fraction {
2         int numerator, denominator;
3     };
4
5     ...
6
7     struct fraction reduce (struct fraction f) {
8
9         int gcd;
10
11        struct fraction f2;
12
13        if (f.numerator >= f.denominator) {
14            gcd = f.denominator;
15        } else {
16            gcd = f.numerator;
17        }
18
19        while (gcd > 0) {
20            if ((f.numerator % gcd == 0) &&
21                (f.denominator % gcd == 0)) {
22                break;
23            }
24
25            gcd--;

```

```

26     }
27
28     f2.denominator = f.denominator/gcd;
29     f2.numerator = f.numerator/gcd;
30
31     return f2;
32 }

```

b) I need to write a function that adds two fractions f1 and f2.

The solution to this problem is:

```

1  struct fraction {
2      int numerator, denominator;
3  };
4
5  ...
6
7  struct fraction add (struct fraction *f1, struct fraction *f2) {
8
9      struct fraction f3;
10
11     if (f1->denominator != f2->denominator) {
12         f3.numerator = (f1->numerator * f2->denominator) + (f2->
numerator * f1->denominator);
13         f3.denominator = f1->denominator * f2->denominator;
14     } else {
15         f3.numerator = f2->numerator + f1->numerator;
16     }
17
18     f3 = reduce(f3);
19
20     return f3;
21 }

```

c) I need to create a function that subtracts two fractions f1 and f2.

The solution to this problem is:

```

1  struct fraction {
2      int numerator, denominator;
3  };
4
5  ...
6
7  struct fraction subtract (struct fraction *f1, struct fraction *
f2) {
8
9      struct fraction f3;
10
11     if (f1->denominator != f2->denominator) {
12         f3.numerator = (f2->numerator * f1->denominator) - (f1->
numerator * f2->denominator);
13         f3.denominator = f1->denominator * f2->denominator;

```

```

14     } else {
15         f3.numerator = f2->numerator - f1->denominator;
16     }
17
18     f3 = reduce(f3);
19
20     return f3;
21 }

```

d) I need to create a function that divides fractions `f1` by fraction `f2`.

The solution to this problem is:

```

1     struct fraction {
2         int numerator, denominator;
3     };
4
5     ...
6
7     struct fraction divide (struct fraction *f1, struct fraction *f2)
8     {
9
10        struct fraction f3;
11
12        f3.denominator = f1->denominator * f2->numerator;
13        f3.numerator = f1->numerator * f2->denominator;
14
15        f3 = reduce(f3);
16
17        return f3;
18    }

```

8. a) I need to write a declaration for a `const` variable named `MAGNETA` of type `struct color` that satisfies the following

- red has value 255
- green has value 0
- blue has value 255

The solution to this problem is:

```

1     struct color {
2         int red;
3         int green;
4         int blue;
5     };
6
7     const struct color MAGNETA = {255, 0, 255}

```

I need to rewrite above to use a designated initializer that doesn't specify the value of `green`.

The solution to this problem is:

```
1 struct color {
2     int red;
3     int green;
4     int blue;
5 };
6
7 const struct color MAGNETA = {.red = 255, .blue = 255};
```

### Notes

- b) • `int` variable prints 0, if nothing is in it
9. a) I need to write the function

```
struct color make_color(int red, int green, int blue)
```

that satisfies the following requirements:

- The function `make_color` should return `color` structure
- The function `make_color` should return 0 for a member variable if the corresponding argument has value less than 0
- The function `make_color` should return 255 for a member variable if the corresponding argument has value greater than 255

The solution to this problem is:

```
1 struct color make_color(int red, int green, int blue) {
2     struct color c;
3
4     red = red < 0 ? 0 : red;
5     green = green < 0 ? 0 : green;
6     blue = blue < 0 ? 0 : blue;
7
8     red = red > 255 ? 255 : red;
9     blue = blue > 255 ? 255 : blue;
10    green = green > 255 ? 255 : green;
11
12    c.red = red;
13    c.green = green;
14    c.blue = blue;
15
16    return c;
17 }
```

- b) I need to write the function

```
int getRed(struct color c)
```

that returns the value of `c`'s red member.

The solution to this problem is

```
1  int getRed(struct color c) {
2      return c.red;
3  }
```

c) I need to write the function

```
bool equal_color(struct color color1, struct color color2)
```

satisfying the following requirement

- The function `equal_color` returns true if the corresponding members of `color1` and `color2` are equal

The solution to this problem is:

```
1  bool equal_color(struct color color1, struct color color2) {
2      if ((color1.red == color2.red) &&
3          (color1.green == color2.green) &&
4          (color1.blue == color2.blue)) {
5          return true;
6      }
7
8      return false;
9  }
```

d) I need to write the function

```
struct color brighter(struct color c)
```

satisfying the following requirement

- if all members of `c` are zero, then return color whose all members have value 3
- If any member of `c` is greater than 0 but less than 3, it is replaced by 3 before the division by 0.7
- if any member of `c` is greater than equal to 3, then, divide by 0.7
- if diving by 0.7 exceeds 255, then replace it with 255

The solution to this problem is:

```
1  struct color brighter(struct color c) {
2
3      if((c.red == 0 ) &&
4          (c.green == 0) &&
5          (c.blue == 0)) {
6
7          c.red = 3;
8          c.green = 3;
9          c.blue = 3;
10 }
```

```

11         return c;
12     }
13
14     c.red = c.red < 3 && c.red > 0 ? 3 : c.red;
15     c.green = c.green < 3 && c.green > 0 ? 3 : c.green;
16     c.blue = c.blue < 3 && c.blue > 0 ? 3 : c.blue;
17
18     c.red = (int)(c.red/0.7);
19     c.green = (int)(c.green/0.7);
20     c.blue = (int)(c.blue/0.7);
21
22     c.red = c.red > 255 ? 255 : c.red;
23     c.green = c.green > 255 ? 255 : c.green;
24     c.blue = c.blue > 255 ? 255 : c.blue;
25
26     return c;
27 }

```

e)

```

1 struct color darker(struct color c) {
2
3     if((c.red == 0 ) &&
4         (c.green == 0) &&
5         (c.blue == 0)) {
6
7         c.red = 3;
8         c.green = 3;
9         c.blue = 3;
10
11         return c;
12     }
13
14     c.red = c.red < 3 && c.red > 0 ? 3 : c.red;
15     c.green = c.green < 3 && c.green > 0 ? 3 : c.green;
16     c.blue = c.blue < 3 && c.blue > 0 ? 3 : c.blue;
17
18     c.red = (int)(c.red*0.7);
19     c.green = (int)(c.green*0.7);
20     c.blue = (int)(c.blue*0.7);
21
22     c.red = c.red > 255 ? 255 : c.red;
23     c.green = c.green > 255 ? 255 : c.green;
24     c.blue = c.blue > 255 ? 255 : c.blue;
25
26     return c;
27 }

```

10. a)

```

1 int area(struct rectangle r) {
2     return ((r.upper_left.y - r.lower_right.y) *
3             (r.lower_right.x - r.upper_left.x));
4 }

```

## Notes

- **Nested Structure**

- Works just like a nested class

```
struct person_name {
    char first[FIRST_NAME_LEN+1];
    char middle_initial;
    char last[LAST_NAME_LEN+1];
};
```

```
struct student {
    struct person_name name;
    int id, age;
    char sex;
} student1, student2;
```

```
strcpy(student1.name.first, "Fred");
```

```
b) struct point center(struct rectangle r) {
2     struct point pt;
3
4     pt.x = (int)((r.upper_left.y - r.lower_right.y)/2);
5     pt.y = (int)((r.lower_right.x - r.upper_left.x)/2);
6
7     return pt;
8 }
```

```
c) struct rectangle translate(struct rectangle r, int x, int y) {
2
3     r.upper_left.x += x;
4     r.upper_left.y += y;
5
6     r.lower_right.x += x;
7     r.lower_right.y += y;
8
9     return r;
10 }
```

```
d) bool inside_rectangle(struct rectangle r, struct point p) {
2     if ((p.x > r.upper_left.x && p.x < r.lower_right.x) &&
3         (p.y > r.lower_right.y && p.y < r.upper_left.y)) {
4         return true;
5     }
6
7     return false;
8 }
```



11. Total of 20 bytes are allocated for **s**.

- **a** - 8 bytes
- **e** - 8 bytes
- **f** - 4 bytes

### Notes

- **Union**

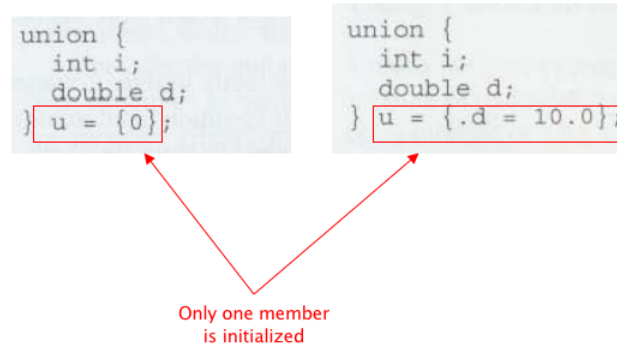
- is very similar to structure (with one difference)
- is accessed the same way as structure

### Example

- shares memory among all members
  - \* Memory is allocated for the largest of members



- Is used to save space in structures
- Only one member can be given an initial value



### Example



- feels as if union is used as an efficient version of switch (could python's type-independent variable also use union?)

12. Total of 16 bytes are allocated for u.

- a - 8 bytes
- e - 16 bytes [is largest]
- f - 4 bytes

- 13.
- a) legal
  - b) legal
  - c) correction: `s.u.rectangle.height = 25;`
  - d) legal

- e) correction: `s.u.circle.radius = 5;`
- f) correction: `s.u.circle.radius = 5;`

```

14. a)  #include <math.h>
        #define M_PI 3.14159265358979323846

        double area (struct shape s) {
            if (s.shape_kind == RECTANGLE) {
                return s.u.rectangle.height * s.u.rectangle.width;
            } else {
                return M_PI * pow(s.u.circle.radius, 2);
            }
        }

```

```

b)      #include <math.h>
        #define M_PI 3.14159265358979323846

        double area (struct shape s) {
            if (s.shape_kind == RECTANGLE) {
                return s.u.rectangle.height * s.u.rectangle.width;
            } else {
                return M_PI * pow(s.u.circle.radius, 2);
            }
        }

```

```

c)      struct shape translate (struct shape s, int x, int y) {
        s.center.x += x;
        s.center.y += y;

        return s;
    }

```

```

d)      struct shape translate (struct shape s, int x, int y) {
        s.center.x += x;
        s.center.y += y;

        return s;
    }

```

```

e)      struct shape scale (struct shape s, double c) {
        if (s.shape_kind == RECTANGLE) {
            s.u.rectangle.height *= c;
            s.u.rectangle.width *= c;
        } else {
            s.u.circle.radius *= c;
        }

        return s;
    }

```

15. a) `enum days {SUN, MON, TUES, WED, THURS, FRI, SAT}`

### Notes

- **Enumeration**

- Is a set of a small number of possible values
- Is declared the similar way as union and structure

**Example:**

```
enum suit { CLUBS = 0, DIAMONDS = 1, HEARTS = 2, SPADES = 3 }
```

enum declared by

```
enum suit s1, s2;
```

- C treats enumeration variables and constants as integers
- Has value 0 for the first constant by default (similar to how array is structured)

**Example**

```
int i;
enum { CLUBS, DIAMONDS, HEARTS, SPADES } s;

i = DIAMONDS; /* i is now 1 */
s = 0;        /* s is now 0 (CLUBS) */
s++;          /* s is now 1 (DIAMONDS) */
i = s + 2;    /* i is now 3 */
```

- Is perfect for struct with unions

```
typedef struct {
    enum { INT_KIND, DOUBLE_KIND } kind;
    union {
        int i;
        double d;
    } u;
} Number;

void print_number(Number n)
{
    if (n.kind == INT_KIND)
        printf("%d", n.u.i);
    else
        printf("%g", n.u.d);
}
```

b) `typedef enum {SUN, MON, TUES, WED, THURS, FRI, SAT} Days`

16. The statements a), c), and e) are true.

17. All are legal, but only a), c), d), e) are safe

**Correct Solution:**

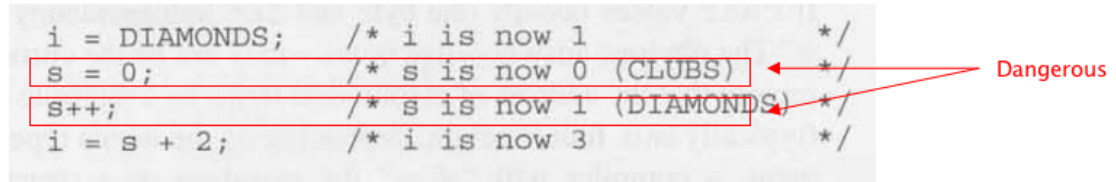
All are legal, but only a), d) and e) are safe

**Notes**

- It's dangerous to use integer as an enumeration value

### Example

```
i = DIAMONDS; /* i is now 1 */
s = 0; /* s is now 0 (CLUBS) */
s++; /* s is now 1 (DIAMONDS) */
i = s + 2; /* i is now 3 */
```



Dangerous

18. a) `typedef enum {PAWN, KNIGHT, BISHOP, ROOK, QUEEN, KING, EMPTY} Piece;`

`typedef enum {BLACK, WHITE} Color;`

b)

```
typedef struct {
    Piece piece;
    Color color;
} Square;
```

c) `Square board[8][8]`