CSC 209 Review 7 Solution

August 27, 2020

1   Exercises

1. First, I need to justify if the folllowing 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__)

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` <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:

```
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       struct complex c3;
7
8       c3.real = c1.real + c2.real;
9       c3.imaginary = c1.imaginary + c2.imaginary;
10
11      return c3;
12  }
```

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

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

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

c)
```
1   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

  - 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:

```
struct date {
    int month, day, year;
};

int day_of_year (struct date d);


...

int day_of_year (struct date d) {
    bool leap_year = false;
    int days = 0, days_in_month[] = {
        31, 28, 31, 30,
        31, 30, 31, 31,
        30, 31, 30, 31};

    // check if it's the leap year
    if (((d.year % 4 == 0) &&
        (d.year % 100 != 0)) ||
        (d.year % 400 == 0)) {

            leap_year = true;
        }


    // add days from months
    for (int i = 0; i < d.month; i++) {
        if (i == d.month-1) {
            days += d.day;
            continue;
        }

        days += days_in_month[i];
    }


    // add 1 more day if month > 2 and it's leap year
    if (leap_year && d.month > 2) {
        days += 1;
    }


    // return days
    return days;
}
```

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:

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

struct date {
    int month, day, year;
};

...

int compare_dates (struct date d1, struct date d2) {
    char s1[9], s2[9];

    sprintf(s1,"%4d%2d%2d", d1.year, d1.month, d1.day);
    sprintf(s2,"%4d%2d%2d", d2.year, d2.month, d2.day);

    // return days
    return strcmp(s1,s2);
}
```

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:

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

struct time {
    int hours, minutes, seconds;
};

struct time split_time (long total_seconds);
```

```
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: numberator 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          struct fraction f3;
10
11         f3.denominator = f1->denominator * f2->numerator;
12         f3.numerator = f1->numerator * f2->denominator;
13
14         f3 = reduce(f3);
15
16         return f3;
17     }
```

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, than, 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
```

14

```
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) {
    struct point pt;

    pt.x = (int)((r.upper_left.y - r.lower_right.y)/2);
    pt.y = (int)((r.lower_right.x - r.upper_left.x)/2);

    return pt;
}
```

c)
```
struct rectangle translate(struct rectangle r, int x, int y) {

    r.upper_left.x += x;
    r.upper_left.y += y;

    r.lower_right.x += x;
    r.lower_right.y += y;

    return r;
}
```

d)
```
bool inside_rectangle(struct rectangle r, struct point p) {
    if ((p.x > r.upper_left.x && p.x < r.lower_right.x) &&
        (p.y > r.lower_right.y && p.y < r.upper_left.y)) {
            return true;
        }

    return false;
}
```

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

- `a` - 8 bytes
- `e` - 8 bytes
- `f` - 4 bytes

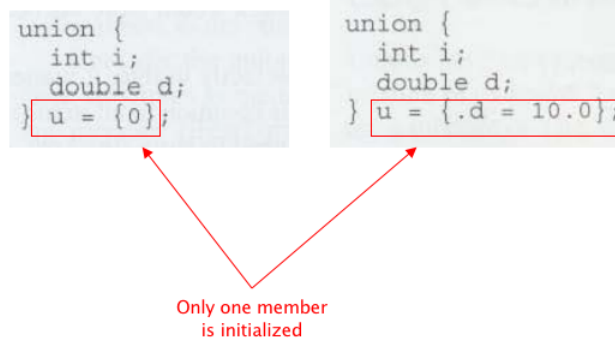**Notes**

- **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

Only one member
is initialized

Example

Use union to allocate
space for struct with largest
memory

Only one of book, mug, shirt
is needed

Call the shared space 'item'

- 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)
```
1   #include <math.h>
2   #define M_PI 3.14159265358979323846
3
4   double area (struct shape s) {
5       if (s.shape_kind == RECTANGLE) {
6           return s.u.rectangle.height * s.u.rectangle.width;
7       } else {
8           return M_PI * pow(s.u.circle.radius, 2);
9       }
10  }
```

b)
```
1   #include <math.h>
2   #define M_PI 3.14159265358979323846
3
4   double area (struct shape s) {
5       if (s.shape_kind == RECTANGLE) {
6           return s.u.rectangle.height * s.u.rectangle.width;
7       } else {
8           return M_PI * pow(s.u.circle.radius, 2);
9       }
10  }
```

c)
```
1   struct shape translate (struct shape s, int x, int y) {
2       s.center.x += x;
3       s.center.y += y;
4
5       return s;
6   }
```

d)
```
1   struct shape translate (struct shape s, int x, int y) {
2       s.center.x += x;
3       s.center.y += y;
4
5       return s;
6   }
```

e)
```
1   struct shape scale (struct shape s, double c) {
2       if (s.shape_kind == RECTANGLE) {
3           s.u.rectangle.height *=  c;
4           s.u.rectangle.width *= c;
5       } else {
6           s.u.circle.radius *= c;
7       }
8
9       return s;
10  }
```

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

Notes

19

- **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 <u>integers</u>
    - Has value 0 for the first constant by default (similar to how array is structured)

    Example

    - Is perfect for struct with unions

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

16. The statements a), c), and e) are true.
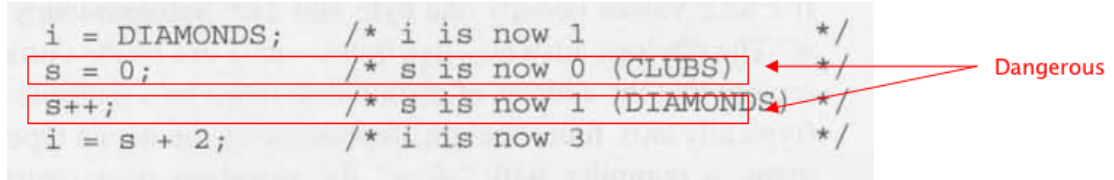
17. All are legal, but only a), c), d), e) are safe

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

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

   typedef enum {BLACK, WHITE} Color;

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

c)
```
Square board[8][8] =
{
    { {ROOK, WHITE},{KNIGHT, WHITE},{BISHOP, WHITE},{KING, WHITE
},{QUEEN, WHITE},{BISHOP, WHITE},{KNIGHT, WHITE},{ROOK, WHITE} },
    { {PAWN, WHITE},{PAWN, WHITE},{PAWN, WHITE},{PAWN, WHITE},{
PAWN, WHITE},{PAWN, WHITE},{PAWN, WHITE},{PAWN, WHITE} },
    { {},{},{},{},{},{},{},{} },
    { {},{},{},{},{},{},{},{} },
    { {},{},{},{},{},{},{},{} },
    { {},{},{},{},{},{},{},{} },
    { {PAWN, BLACK},{PAWN, BLACK},{PAWN, BLACK},{PAWN, BLACK},{
PAWN, BLACK},{PAWN, BLACK},{PAWN, BLACK},{PAWN, BLACK} },
    { {ROOK, BLACK},{KNIGHT, BLACK},{BISHOP, BLACK},{QUEEN, BLACK
},{KING, BLACK},{BISHOP, BLACK},{KNIGHT, BLACK},{ROOK, BLACK} },
};
```