Study Questions: Set No. 9
Introduction to C
<u>Thursday November 21, 2013</u>

<u>Covering:</u>

Chapter 14 and 16

1. Write parameterized macros that compute the following values.
   (a) The cube of `x`.
   (b) The remainder when `n` is divided by `4`.
   (c) `1` if the product of `x` and `y` is less than `100`,
   `0` otherwise.
   Do your macros always work? If not, describe what arguments would make them fail.
   Hint: What will happen if the provided argument(s) have side effect?

2. Write a macro `NUMBER.OF.ELEMENS(a)` that computes the number of elements in a one-dimensional array `a`.
   Hint: See the discussion of the `sizeof` operator in Section 8.1.

3. Let `DOUBLE` be the following macro:
   `#define DOUBLE (x) 2*x`
   (a) What is the value of `DOUBLE(1+2)?`
   (b) What is the value of `4/DOUBLE(2)?`
   (c) Fix the definition of `DOUBLE`

4. For each of the following macros, give an example that illustrates a problem with the macro and show how to fix it.
   (a) `#define AVG(x, y) (x+y)/2`
   (b) `#define AREA(x, y) (x)*(y)`

5. Let `TOUPPER` be the following macro:
   `#define TOUPPER(c) ('a' <= (c) && (c) <='z'? (c)-'a'+'A':(c))`
   Let `s` be a string and let `i` be an `int` variable.
   Show the output produced by each of the following program fragments.

```
(a) strcpy(s, "abcde");
    i=0;
    putchar(TOUPPER(s[++i]));
(b) strcpy(s, "01234");
    i=0;
    putchar(TOUPPER(s[++i]));
```

6. Let `TOUPPER` be the following macro:
   `#define TOUPPER(c) ('a' <= (c) && (c) <='z'? (c)-'a'+'A':(c))`
   Let `s` be a string and let `i` be an `int` variable.
   Show the output produced by each of the following program fragments.

```
(a) strcpy(s, "abcde");
    i=0;
    putchar(TOUPPER(s[i++]));
(b) strcpy(s, "01234");
    i=0;
    putchar(TOUPPER(s[i++]));
```

7. Write a macro `DISP(f, x)` that expands into a call of `printf` that displays the value of the function `f` when called with argument `x`. For example,
```
DISP(sqrt, 3.0);
```
should expand into
```
printf("sqrt(%g) = %g\n", 3.0, sqrt(3.0));
```

8. Write the following parameterized macro
`CHECK(x, y, n)` which has the value 1 if both `x` and `y` fall between `0` and `n - 1`, inclusive.

9. Write the following parameterized macro
`MEDIAN(x, y, z)` which finds the median of `x`, `y`, and `z`.

10. Write the following parameterized macro
`POLYNOMIAL(x)` which computes the polynomial $3x^5 + 2x^4 - 5x^3 - x^2 + 7x - 6$

11. C programmers often use the `fprintf` function to write error messages:
```
fprintf(stderr, "Range error: index = %d\n", index);
```
where `stderr` is C's "standard error" stream; the remaining arguments are the same as those for `printf`. Write a macro named `ERROR` that generates the call of `fprintf` shown above when given a format string and the items to be displayed:
```
ERROR("Range error: index = %d\n", index);
```

12. Suppose that the macro `M` has been defined as follows:
```
#define M 10
```
Which of the following tests will fail?
```
(a) #if M
(b) #ifdef M
(c) #ifndef M
(d) #if defined(M)
(e) #if !defined(M)
```

13. Show what the following program will look like after preprocessing. You may ignore any lines added to the program as a result of including the < `stdio.h>` header. What will be the output of this program?
```
#include <stdio.h>
#define N 100
void f(void);
int main(void)
{
  f();
#ifdef N
#undef N
#endif
  return 0;
}

void f(void)
{
#if defined(N)
  printf("N is %d\n", N);
#else
  printf("N is undefined\n");
#endif
}
```

14. C compilers usually provide some method of specifying the value of a macro at the time a program is compiled. This ability makes it easy to change the value of a macro without editing any of the program's files. Most compilers (including `gcc`) support the `-D` option, which allows the value of a macro to be specified on the command line, i.e., *defines* a macro as if by using `#define`. Many compilers also support the `-U` option, which "*undefines*" a macro as if by using `#undef`.
Type the following program:

```
#include <stdio.h>

#ifdef DEBUG
#define PRINT_DEBUG(n) printf("Value of " #n ": %d\n", n)
#else
#define PRINT_DEBUG(n)
#endif

int main(void)
{

int i = 1, j = 2, k = 3;

#ifdef DEBUG
  printf("DEBUG is defined:\n");
#else
  printf("DEBUG is not defined:\n");
#endif

PRINT_DEBUG(i);

PRINT_DEBUG(j);

PRINT_DEBUG(k);

PRINT_DEBUG(i + j);

PRINT_DEBUG(2 * i + j - k);

return 0;
}
```

a. Compile and the run this program without using any option during compilation
b. Compile and the run this program using the following options during compilation:
    i. `-DDEBUG=1`
    ii. `-DDEBUG`
    iii. `-Ddebug=1`
    iv. `-Ddebug`
    v. What are the differences between the four runs? Why?

15. Suppose that a program consists of three source files (`main.c`, `f1.c`, and `f2.c`) plus two header files, `f1.h` and `f2.h`. All three source files include `f1.h`. but only `f1.c` and `f2.c` include `f2.h`. Write a makefiie for this program, assuming that the compiler is `gcc` and that the executable file is to be named `demo`
a. If `f1.c` is changed after the program has been built, which files need to be recompile
b. If `f1.h` is changed after the program has been built, which files need to be recompile
c. If `f2.h` is changed after the program has been built, which files need to be recompile

16. Show what the following program will look like after preprocessing. Some lines of the program may cause compilation errors; find all such errors.

```
#define N = 10
#define INC(x)' x+1
#define SUB (x,y) x-y
#define SQR(x) ((x)*(x))
#define CUBE(x) (SQR(x)*(x))
#define M1(x, y) x##y
#define M2(x,y) #x #y
int main(void)
{ int a[N], i, j, k, m;
#ifdef N
  i=j;
#else
  j = i;
#endif
  i = 10 * INC(j);
  i = SUB(j, k);
  i = SQR(SQR(j));
  i = CUBE(i);
  i = M1(j, k);
  puts(M2(i, j));
#undef SQR
  i = SQR(j)
#define SQR
  i = SQR(j);
  Return 0;
}
```

17. Consider the following type `student_t` and the two variables `stu1` and `stu2`.

```
typedef struct
{
  char first_name[20],
       last_name[20];
  int  score;
  char grade;
}student_t;
...
student_t stu1, stu2;
```

Identify the following statements as possibly valid or definitely invalid. If invalid, explain why.

```
(a) student_t stu_list[30];
(b) printf("%s", stu1);
(c) printf("%d %c", stu1.score, stu1.grade);
(d) stu2 = stu1;
(e) if(stu2.score == stu1.score)
      printf("Equal");
(f) if(stu2 == stu1)
      printf("Equal structures");
(g) scan_student(&stu1);
(h) stu2.last_name = "Martin";
```

18. In the following declarations, the `x` and `y` structures have members named `x` and `y`:

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

Are these declarations legal on an individual basis?
Could both declarations appear as shown in a program? Justify your answer.

19. Assume that the `fraction` structure contains two members: `numerator` and `denominator` (both of type `int`).
Write functions that perform the following operations on fractions:
(a) Reduce the fraction `f` to lowest terms.
Hint: To reduce a fraction to lowest terms, first compute the greatest common divisor (GCD) of the numerator and denominator. Then divide both the numerator and denominator by the GCD.
(b) Add the fractions `f1` and `f2`.
(c) Subtract the fraction `f2` from the fraction `f1`.
(d) Multiply the fractions `f1` and `f2`.
(e) Divide the fraction `f1` by the fraction `f2`.
The fractions `f`, `f1`, and `f2` will be arguments of type `struct fraction;` each function will return a value of type `struct fraction.` The fractions returned by the functions in parts (b)-(e) should be reduced to lowest terms. *Hint: You* may use the function from part (a) to help write the functions in parts (b)-(e).

20. Let `color` be the following structure:

```
struct color
{ int red;
  int green;
  int blue;
}
```

Write the following functions:
`(a) struct color make color(int red, int green, int blue);` Returns a color structure containing the specified red, green, and blue values. If any argument is less than zero, the corresponding member of the structure will contain zero instead. If any argument is greater than 255, the corresponding member of the structure will contain 255.
`(b) int getRed(struct color c);`
Returns the value of `c`'s `red` member.
`(c) bool equal_color(struct color color1, struct color color2);`
Returns `true` if the corresponding members of `color1` and `color2` are equal.
`(d) struct color brighter(struct color c);`
Returns a `color` structure that represents a brighter version of the color `c`. The structure is identical to `c`, except that each member has been multiplied by 0.7 (with the result truncated to an integer). However, there are three special cases:
1. If all members of `c` are zero, the function returns a color whose members all have the value 3.
2. If any member of `c` is greater than 0 but less than 3, it is replaced by 3 before the multiplication by 0.7.
3. If multiplying by 0.7 causes a member to exceed 255, it is reduced to 255.
`(e) struct color darker(struct color c);`
Returns a `color` structure that represents a darker version of the color `c`. The structure is identical to `c`, except that each member has been divided by 0.7 (with the result truncated to an integer).

21. Show how to declare a tag named `complex` for a structure with two members, `real` and `imaginary`, of type `double`. Use the `complex` tag to declare variables named `c1, c2,` and `c3`.

22. Declare structure variables named `c1`, `c2`, and `c3`, each having members `real` and `imaginary` of type `double`. Modify this declaration so that `c1`'s members initially have the values 0.0 and 1.0, while `c2`'s members are 1.0 and 0.0 initially. (`c3` is not initialized.)

23. Consider `c1` and `c2` are structure variables of type `complex_t`, which have two members `real` and `imaginary` of type `double`. Copy the members of `c2` into `c1`. Can this be done in one statement, or does it require two?

24. Consider the type `complex_t`, which have two members `real` and `imaginary` of type `double`. Write a function that stores its two parameters (both of type `double`) in a `complex` structure, then returns the structure.

25. Consider the type `complex_t`, which have two members `real` and `imaginary` of type `double`. Write a function that accepts one parameter of type `complex_t` and returns its absolute value, which equals to $\sqrt{real^2 + imaginary^2}$.

26. Consider the type `complex_t`, which have two members `real` and `imaginary` of type `double`. Write a function that accepts two parameters of type `complex_t` and returns the result of their addition.

27. Consider the type `complex_t`, which have two members `real` and `imaginary` of type `double`. Write a function that accepts two parameters of type `complex_t` and returns the result of their multiplication.

28. Consider the type `complex_t`, which have two members `real` and `imaginary` of type `double`. Write a function that accepts two parameters `c1` and `c2` of type `complex_t` and returns the result of their division `c1/c2`.

29. The following structures are designed to store information about objects on a graphics screen:

```
struct point { int x, y; };
struct rectangle { struct point upper_left, lower_right; };
```

A `point` structure stores the `x` and `y` coordinates of a point on the screen. A `rectangle` structure stores the coordinates of the upper left and lower right corners of a rectangle. Write functions that perform the following operations on a `rectangle` structure `r` passed as an argument:

(a) Compute the area of `r`.

(b) Compute the center of `r`, returning it as a `point` value. If either the `x` or `y` coordinate of the center isn't an integer, store its truncated value in the `point` structure.

(c) Move `r` by `x` units in the *x* direction and `y` units in the *y* direction, returning the modified version of `r`, where `x` and `y` are additional arguments to the function.

(d) Determine (by returning `true` or `false`) whether a point `p` lies inside `r` or not, where `p` is an additional argument of type `struct point`.

30. Suppose that `s` is the following structure (`point` is a structure tag declared in the previous question):

```
struct shape
{ int shape kind;          /* RECTANGLE or CIRCLE */
  struct point center;     /* coordinates of center */
  union
  { struct
    { int height, width;
    } rectangle;
    struct
    {
      int radius;
    } circle;
  } u;
} s;
```

If the value of `shape_kind` is `RECTANGLE`, the `height` and `width` members store the dimensions of a rectangle. If the value of `shape_kind` is `CIRCLE`, the `radius` member stores the radius of a circle. Indicate which of the following statements are legal, and show how to repair the ones that aren't:

```
(a) s.shape_kind = RECTANGLE;
```

```
(b) s.center.x = 10;
(c) s.height = 25;
(d) s.u.rectangle.width = 8;
(e) s.u.circle = 5;
(f) s.u.radius = 5;
```

31. Let `shape` be the structure tag declared in the previous question. Write functions that perform the following operations on a shape structure `s` passed as an argument:
(a) Compute the area of `s`.
(b) Move `s` by `x` units in the `x` direction and `y` units in the `y` direction, returning the modified version of `s`. (`x` and `y` are additional arguments to the function.)
(c) Scale `s` by a factor of `c` (a `double` value), returning the modified version of `s`. (`c` is an additional argument to the function.)

32. Consider the `date` structure contains three members: `month, day,` and `year` (all of type `int).`
Write a functions `int day_of_year(struct date d)` that returns the day of the year (an integer between 1 and 366) that corresponds to the date `d`.

33. Consider the `date` structure contains three members: `month, day,` and `year` (all of type `int).`
Write a functions `int compare_dates(struct date d1, struct date d2)` that returns `-1` if `d1` is an earlier date than `d2`, `+1` if `d1` is a later date than `d2`, and `0` if `d1` and `d2` are the same.

34. Write the following function, assuming that the `time` structure contains three members: `hours, minutes,` and `seconds` (all of type `int).`
`struct time split_time(long total_seconds);`
`total_seconds` is a time represented as the number of seconds since midnight. The function returns a structure containing the equivalent time in hours (0-23), minutes (0-59), and seconds (0-59).

35. Suppose that `s` is the following structure:
```
struct
{ double a;
  union
  { char b[4];
    double c;
    int d;
  } e;
  char f[4];
} s;
```
If `char` values occupy one byte, `int` values occupy four bytes, and `double` values occupy eight bytes, how much space will a C compiler allocate for `s`? (Assume that the compiler leaves no "holes" between members.)

36. Suppose that `u` is the following union:
```
union
{ double a;
  struct
  { char b[4];
    double c;
    int d;
  } e;
  char f[4];
} u;
```
If `char` values occupy one byte, `int` values occupy four bytes, and `double` values occupy eight bytes, how much space will a C compiler allocate for `u`? (Assume that the compiler leaves no "holes" between members.)

37. Which of the following statements about enumeration constants are true?
(a) An enumeration constant may represent any integer specified by the programmer.
(b) Enumeration constants have the values 0, 1, 2, ... by default.
(c) All constants in an enumeration must have different values.
(d) Enumeration constants may be used as integers in expressions.

38. Suppose that `b` and `i` are declared as follows:
```
enum {FALSE, TRUE} b;
int i;
```
Which of the following statements are legal? Which ones are "safe" (i.e., always yield a meaningful result)?
```
(a) b = FALSE;
(b) b = i;
(c) b++;
(e) i = b;
(f) i=2 * b + 1;
```

39. Declare a tag for an enumeration whose values represent the seven days of the week

40. Declare a structure with the following members whose tag is `pinball_machine`:
`name` - a string of up to 40 characters
`year` - an integer (representing the year of manufacture)
`type` - an enumeration with the values EM (electromechanical) and SS (solid state)
`players` - an integer (representing the maximum number of players)

41. Suppose that the `direction` variable is declared in the following way:
```
enum {NORTH, SOUTH, EAST, WEST} direction;
```
Let `x` and `y` be `int` variables. Write a `switch` statement that tests the value of `direction`, incrementing `x` if `direction is EAST`, decrementing `x` if `direction is WEST`, incrementing `y` if `direction is SOUTH`, and decrementing `y` if `direction is NORTH`.

42. Consider the following enumeration:
```
enum {frosh, soph, jr, sr};
```
What is displayed by this code fragment
```
for(class = frosh; class <=jr; ++class)
   printf("%d\n", class);
```

43. What are the integer values of the enumeration constants in each of the following declarations?
```
(a) enum {UL, SOH, STX, ETX};
(b) enum {VT = 11, FF, CR} ;
(c) enum {SO = 14, SI, DLE, CAN = 24, EM};
(d) enum {ENQ = 45, ACK, BEL, LF = 37, ETB, ESC};
```