```
enum gray_values {
  BLACK = 0,
  DARK_GRAY = 64,
  GRAY = 128,
  LIGHT_GRAY = 192,
};
```

**Is this practice legal?**

A: This practice is indeed legal in C99 (and is supported by some pre-C99 compilers as well). Allowing a "trailing comma" makes enumerations easier to modify, because we can add a constant to the end of an enumeration without changing existing lines of code. For example, we might want to add `WHITE` to our enumeration:

C99

```
enum gray_values {
  BLACK = 0,
  DARK_GRAY = 64,
  GRAY = 128,
  LIGHT_GRAY = 192,
  WHITE = 255,
};
```

The comma after the definition of `LIGHT_GRAY` makes it easy to add `WHITE` to the end of the list.

One reason for this change is that C89 allows trailing commas in initializers, so it seemed inconsistent not to allow the same flexibility in enumerations. Incidentally, C99 also allows trailing commas in compound literals.

C99

**Q: Can the values of an enumerated type be used as subscripts?**

A: Yes, indeed. They are integers and have—by default—values that start at 0 and count upward, so they make great subscripts. In C99, moreover, enumeration constants can be used as subscripts in designated initializers. Here's an example:

C99

```
enum weekdays {MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY};
const char *daily_specials[] = {
  [MONDAY] = "Beef ravioli",
  [TUESDAY] = "BLTs",
  [WEDNESDAY] = "Pizza",
  [THURSDAY] = "Chicken fajitas",
  [FRIDAY] = "Macaroni and cheese"
};
```

## Exercises

**Section 16.1**

1. 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.

W 2. (a) Declare structure variables named `c1`, `c2`, and `c3`, each having members `real` and `imaginary` of type `double`.

(b) Modify the declaration in part (a) 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.)

(c) Write statements that copy the members of `c2` into `c1`. Can this be done in one statement, or does it require two?

(d) Write statements that add the corresponding members of `c1` and `c2`, storing the result in `c3`.

**Section 16.2**

3. (a) Show how to declare a tag named `complex` for a structure with two members, `real` and `imaginary`, of type `double`.

(b) Use the `complex` tag to declare variables named `c1`, `c2`, and `c3`.

(c) Write a function named `make_complex` that stores its two arguments (both of type `double`) in a `complex` structure, then returns the structure.

(d) Write a function named `add_complex` that adds the corresponding members of its arguments (both `complex` structures), then returns the result (another `complex` structure).

W 4. Repeat Exercise 3, but this time using a *type* named `Complex`.

5. Write the following functions, assuming that the `date` structure contains three members: `month`, `day`, and `year` (all of type `int`).

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

Returns the day of the year (an integer between 1 and 366) that corresponds to the date `d`.

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

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.

6. 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).

7. 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).

8. Let `color` be the following structure:

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

(a) Write a declaration for a `const` variable named `MAGENTA` of type `struct color` whose members have the values 255, 0, and 255, respectively.

(b) (C99) Repeat part (a), but use a designated initializer that doesn't specify the value of `green`, allowing it to default to 0.

9. Write the following functions. (The `color` structure is defined in Exercise 8.)

(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 divided 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 division by 0.7. (3) If dividing 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 multiplied by 0.7 (with the result truncated to an integer).

**Section 16.3**

10. 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`. (`x` and `y` are additional arguments to the function.)
(d) Determine whether a point `p` lies within `r`, returning `true` or `false`. (`p` is an additional argument of type `struct point`.)

**Section 16.4**

W 11. 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.)

12. 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.)

13. Suppose that `s` is the following structure (`point` is a structure tag declared in Exercise 10):

```
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;`

W 14. Let `shape` be the structure tag declared in Exercise 13. 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.)

**Section 16.5**

W 15. (a) Declare a tag for an enumeration whose values represent the seven days of the week.

(b) Use `typedef` to define a name for the enumeration of part (a).

16. 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 exactly the same properties as constants created using `#define`.

(c) Enumeration constants have the values 0, 1, 2, … by default.

(d) All constants in an enumeration must have different values.

(e) Enumeration constants may be used as integers in expressions.

W 17. 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" (always yield a meaningful result)?

```
(a) b = FALSE;
(b) b = i;
(c) b++;
(d) i = b;
(e) i = 2 * b + 1;
```

18. (a) Each square of a chessboard can hold one piece—a pawn, knight, bishop, rook, queen, or king—or it may be empty. Each piece is either black or white. Define two enumerated types: `Piece`, which has seven possible values (one of which is "empty"), and `Color`, which has two.

(b) Using the types from part (a), define a structure type named `Square` that can store both the type of a piece and its color.

(c) Using the `Square` type from part (b), declare an $8 \times 8$ array named `board` that can store the entire contents of a chessboard.

(d) Add an initializer to the declaration in part (c) so that `board`'s initial value corresponds to the usual arrangement of pieces at the start of a chess game. A square that's not occupied by a piece should have an "empty" piece value and the color black.

19. 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)

20. 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`.

21. What are the integer values of the enumeration constants in each of the following declarations?

(a) `enum {NUL, 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};`

22. Let `chess_pieces` be the following enumeration:

```
enum chess_pieces {KING, QUEEN, ROOK, BISHOP, KNIGHT, PAWN};
```

(a) Write a declaration (including an initializer) for a constant array of integers named `piece_value` that stores the numbers 200, 9, 5, 3, 3, and 1, representing the value of each chess piece, from king to pawn. (The king's value is actually infinite, since "capturing" the king (checkmate) ends the game, but some chess-playing software assigns the king a large value such as 200.)

(b) (C99) Repeat part (a), but use a designated initializer to initialize the array. Use the enumeration constants in `chess_pieces` as subscripts in the designators. (*Hint:* See the last question in Q&A for an example.)

## Programming Projects

1. Write a program that asks the user to enter an international dialing code and then looks it up in the `country_codes` array (see Section 16.3). If it finds the code, the program should display the name of the corresponding country; if not, the program should print an error message.

2. Modify the `inventory.c` program of Section 16.3 so that the `p` (print) operation displays the parts sorted by part number.

3. Modify the `inventory.c` program of Section 16.3 by making `inventory` and `num_parts` local to the `main` function.

4. Modify the `inventory.c` program of Section 16.3 by adding a `price` member to the `part` structure. The `insert` function should ask the user for the price of a new item. The `search` and `print` functions should display the price. Add a new command that allows the user to change the price of a part.

5. Modify Programming Project 8 from Chapter 5 so that the times are stored in a single array. The elements of the array will be structures, each containing a departure time and the corresponding arrival time. (Each time will be an integer, representing the number of minutes since midnight.) The program will use a loop to search the array for the departure time closest to the time entered by the user.

6. Modify Programming Project 9 from Chapter 5 so that each date entered by the user is stored in a `date` structure (see Exercise 5). Incorporate the `compare_dates` function of Exercise 5 into your program.