

This course has already ended.

« 7. Multidimensional Arrays

Course materials

2 Multidimensional arrays »

ELEC-A7100 / 7. Multidimensional Arrays / 1 More structured data types

Seventh round tasks as a [ZIP file](#).

This section is mainly for **multidimensional arrays**, but before dealing with multidimensional arrays, a brief introduction is given to **Enumerated data types**, which are used to improve readability of the program.

**Multidimensional arrays** is not a new feature of the C language, but they are arrays, which consist of arrays. Arrays can be built either statically or dynamically. Especially in the latter case, you must be careful with memory allocations, use and releasing memory.

## More structured data types¶

### Enumeration constants¶

Enumeration constants can be used to define variables that have one of the specified constants as a possible value. Here is an example of enumeration *en\_color* definition and its use:

```
1 enum en_color {
2     RED,
3     GREEN,
4     BLUE
5 };
6
7 void printColor(enum en_color col)
8 {
9     switch(col) {
10        case RED:
11            printf("red\n");
12            break;
13        case GREEN:
14            printf("green\n");
15            break;
16        case BLUE:
17            printf("blue\n");
18            break;
19    }
20 }
```

Compiler translates enumeration constants to constant integers, and they can be used in all normal expressions or functions similarly as if a constant integer was used in their place. By default, the enumeration constants are assigned increasing values starting from zero: 'RED' equals to 0, 'GREEN' is 1, and 'BLUE' is 2 in the above example, but typically the integer values should not be directly used.

The integer values for enumerations can also be given explicitly, as shown below:

```
/* Type Duration, some durations in seconds */
typedef enum {
    MINUTE = 60,
    HOUR = 60 * MINUTE,
    DAY = 24 * HOUR
} Duration;
```

Farther down, you can find an example of a spreadsheet computation program with 10 cells to use. Below is an enumeration constant **en\_types** with four possible values that represent what kind of value is present in the cell.

```
enum en_types {
    UNSPEC, // union content is unspecified - empty
    NUMBER, // union stores a number
    LABEL, // union stores a string number
    FORMULA // union stores a formula
};
```

For every cell, we need to store two things: field containing the actual value of the cell and a variable to keep track of the type of the value stored. For these, we will create a structure *struct cell*.

Now, the program first initializes the array to be empty - i.e., to UNSPEC. Then, the program sets a string to the cell 0 and a formula to the cell 1 and finally, prints out the whole array. Printing format depends on the type of the value stored, and for this we can use switch structure since enumerations are practically integers.

```
1 #include <string.h>
2 #include <stdio.h>
3
4 // This structure stores an imaginary formula and two parameters
5 struct formula {
6     char operation[10];
7     char param1[10];
8     char param2[10];
9 };
10
11 union u_types {
12     float num;
13     char label[20];
14     struct formula form;
15 };
16
17 enum en_types {
18     UNSPEC, // union content is unspecified
19     NUMBER, // union stores a number
20     LABEL, // union stores a string number
21     FORMULA // union stores a formula
22 };
23
24 // one cell in spreadsheet: stores the type of the cell (enum)
25 // and the actual content (union, corresponding type indicated by enum)
26 struct cell {
27     enum en_types type;
28     union u_types value;
29 };
30
31 int main(void)
32 {
33     struct cell sheet[10];
34     struct formula avg = { "AVG", "A2", "A3" };
35
36     int i;
37     for (i = 0; i < 10; i++) {
38         // initialization: all fields unspecified
39         sheet[i].type = UNSPEC;
40     }
41
42     // Because label is an array, we need to use strcpy
43     sheet[0].type = LABEL;
44     strcpy(sheet[0].value.label, "Header");
45
46     // Direct assignment works because this is struct
47     sheet[1].type = FORMULA;
48     sheet[1].value.form = avg;
49
50     for (i = 0; i < 10; i++) {
51         switch(sheet[i].type) {
52             case LABEL:
53                 printf("%s\n", sheet[i].value.label);
54                 break;
55             case NUMBER:
56                 printf("%f\n", sheet[i].value.num);
57                 break;
58             case FORMULA:
59                 printf("%s(%s, %s)\n",
60                     sheet[1].value.form.operation,
61                     sheet[1].value.form.param1,
62                     sheet[1].value.form.param2);
63                 break;
64             case UNSPEC:
65                 printf("--\n");
66                 break;
67         }
68     }
69     return 0;
70 }
```

### Task 7.1: Number of seats in a vehicle¶


In the distributed exercise template, you can find the main function that will need a **num\_seats** enumeration to compile. Your task is to implement this enumeration according to the main function and the expected output that you can find below. Using these, you should figure what elements there should be in the enumeration and what are their values.

When the program works correctly, it should print out the following:

We have a variable test of type enum num\_seats  
and it has an integer value of 1!!

Number of seats in a bike is 1  
Number of seats in a motorcycle is 2  
Number of seats in a van is 3  
Number of seats in a car is 5  
Number of seats in a minivan is 7

Points **10 / 10** My submissions **1** ▾  Deadline Friday, 23 July 2021, 19:59  To be submitted alone

 This course has been archived (Saturday, 31 December 2022, 20:00).

### Number of seats in a vehicle

Select your files for grading

 **main.c**

 Choose File No file chosen

Submit

« 7. Multidimensional Arrays

Course materials

2 Multidimensional arrays »