Enumerated Types | Coursera

coursera.org/learn/programming-fundamentals/supplement/a7zEW/enumerated-types

Enumerated Types

Enumerated Types

The last form of custom type that a programmer can create is called an enumerated type. Enumerated types are named constants that can increase the readability and the correctness of your code. They are most useful when you have a type of data with a set of values that you would like to label by their conceptual name (rather than using a raw number) and either the particular numerical values do not matter (as long as they are distinct), or they occur naturally in a sequential progression. For example, until 2011 the United States' Homeland Security maintained a color-coded terrorism threat advisory scale that it used to maintain heightened or more relaxed security in various locations including major airports. There were five threat levels from green to red in ascending order of severity.

These five threat levels could be recorded in an enumerated type which we can create ourselves as shown in the following code.

```
1
2
3
4
5
6
7
enum threat_level_t {
  LOW,
  GUARDED,
  ELEVATED,
  HIGH,
  SEVERE
};
```

We begin with the keyword **enum**, followed by the name of the new enumerated type, in this case threat_level_t. The various threat levels are placed in curly braces, as shown. Each level is assigned a constant value, starting with 0. The enumerated names are constant—they are not assignable variables. Their values cannot change throughout the program. The convention for indicating that a name denotes a constant is to write the name in all uppercase. However, variables of the enumerated type can be created, and assigned to normally.

Because enumerated types have integer values, they can be used in constructs such as simple value comparisons, switch statements, and for loops. the following code shows an example of the first two. The Enumerated Types lecture illustrates the execution of this code.

```
3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20
```

```
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
1
2
    case LOW:
      printf("Green/Low.\n");
      break;
    case GUARDED:
      printf("Blue/Guarded.\n");
      break;
    case ELEVATED:
      printf("Yellow/Elevated.\n");
      break;
    case HIGH:
```

```
      printf("Orange/High.\n");
      break;
    case SEVERE:
      printf("Red/Severe.\n");
      break;
  }
}

void printShoes(enum threat_level_t currThreat) {
  if (currThreat >= ELEVATED) {
     printf("Please take off your shoes.\n”);
  }
  else {
     printf("Please leave your shoes on.\n”);
  }
}

int main(void) {
  enum threat_level_t myThreat = HIGH;
  printf("Current threat level is:\n");
  printThreat(myThreat);
  printShoes(myThreat);
  return 0;
}
void printThreat(enum threat_level_t threat){
  switch(threat) {
```

Another example of enumerated types would be if we wanted to make a program that regularly refers to a small set of fruits: grapes, apples, oranges, bananas, and pears. Suppose we want to represent each of these as a number (because we regularly use constructs like switch statements on the fruits themselves), but we do not really care which number each is represented as. We can make a enumerated type, **enum fruit_t {GRAPE, APPLE,…};** and then use these constants throughout our code.