The famous ***Match & Hit*** game is played by a computer and a human player as follows. First, the computer selects a random 4-digit number $N = a \cdot 10^3 + b \cdot 10^2 + c \cdot 10 + d$, where a, b, c, d are *distinct nonzero digits* — that is, a, b, c, d are distinct elements of the set $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Let us call numbers like this ***valid***. The human player then tries to deduce N from a sequence of queries to the computer. Each query consists of a valid 4-digit number $M = x \cdot 10^3 + y \cdot 10^2 + z \cdot 10 + w$. The computer responds to each query with the *number of matches* and the *number of hits*. A ***match*** is a digit of N that appears in M at the same position (thus each of $x = a$, $y = b$, $z = c$, or $w = d$ counts as one match). A ***hit*** is a digit of N that appears in M, but not at the same position. For example, if $N = 5167$, then the queries 2934, 1687, 7165, 5167 will result in the following numbers of matches and hits:

5167	
2934	⟶ no matches and no hits
1687 ○○ ●	⟶ one match and two hits
7165 ○●●○	⟶ two matches and two hits
5167 ●●●●	⟶ four matches and no hits

where matches are denoted by ● and hits are denoted by ○. The play continues until the human player either wins or loses the game, as follows:

> **Human player wins** if he submits a query with 4 matches (that is, $M = N$).
>
> **Human player loses** if he submits 12 queries, none of them with 4 matches.

Write a C program, called `match_and_hit.c`, that implements (the computer part of) the Match & Hit game. In this program, you are required to declare, define, and call the following functions:

- The function **`isvalid(int n)`** that accepts as a parameter an arbitrary integer. The function should return **`1`** if the integer is valid, and **`0`** otherwise. Recall that an integer is valid if it is positive, consists of exactly 4 decimal digits, and all these digits are nonzero and distinct.
- The function **`choose_N(void)`** that returns, as an **`int`**, a uniformly random choice of a valid integer N. The function should call both the **`rand`** library function and the function **`isvalid`**. It should keep calling **`rand`** until the number generated thereby is valid. Recall that before the function **`rand`** is invoked, the random seed should be initialized using **`srand(time(0))`**.

- The function **`matches(int N, int M)`** that accepts as parameters two integers N and M and returns, as an **`int`**, the number of matches. You can assume than both N and M are valid.
- The function **`hits(int N, int M)`** that accepts as parameters two integers N and M, then returns, as an **`int`**, the number of hits. You can assume than both N and M are valid.

Here is a sample run of this program, assuming that the executable file is called `match_and_hit`. This run illustrates a situation where the human player wins the game. User input is underlined.

```
/home/userXYZ/ECE15/Lab3> match_and_hit
 ***Welcome to the MATCH and HIT game***
The computer has selected a 4-digit number.
Try to deduce it in 12 rounds of queries.

Round #1
Please enter your query (4 digits): 5341
-> 2 matches and 1 hit

Round #2
Please enter your query (4 digits): 1235
-> 1 match and 2 hits

Round #3
Please enter your query (4 digits): 2345
-> 4 matches and 0 hits

**********************************
CONGRATULATIONS! You won the game!
**********************************
```

Notice that a singular form (**`match`**, **`hit`**) is used with the number **`1`**. You can easily achieve this functionality with the **`?:`** conditional expression, as shown in class. The next sample run illustrates a situation where the human player loses the game. Again, user input is underlined.

- The function **`matches(int N, int M)`** that accepts as parameters two integers N and M and returns, as an **`int`**, the number of matches. You can assume than both N and M are valid.
- The function **`hits(int N, int M)`** that accepts as parameters two integers N and M, then returns, as an **`int`**, the number of hits. You can assume than both N and M are valid.

Here is a sample run of this program, assuming that the executable file is called `match_and_hit`. This run illustrates a situation where the human player wins the game. User input is underlined.

```
/home/userXYZ/ECE15/Lab3> match_and_hit
 ***Welcome to the MATCH and HIT game***
The computer has selected a 4-digit number.
Try to deduce it in 12 rounds of queries.

Round #1
Please enter your query (4 digits): 5341
-> 2 matches and 1 hit

Round #2
Please enter your query (4 digits): 1235
-> 1 match and 2 hits

Round #3
Please enter your query (4 digits): 2345
-> 4 matches and 0 hits

*********************************
CONGRATULATIONS! You won the game!
*********************************
```

Notice that a singular form (**`match`**, **`hit`**) is used with the number **`1`**. You can easily achieve this functionality with the **`? :`** conditional expression, as shown in class. The next sample run illustrates a situation where the human player loses the game. Again, user input is underlined.

```
/home/userXYZ/ECE15/Lab3> match_and_hit
 ***Welcome to the MATCH and HIT game***
The computer has selected a 4-digit number.
Try to deduce it in 12 rounds of queries.

Round #1
Please enter your query (4 digits): 1234
-> 0 matches and 3 hits
 :    :

Round #12
Please enter your query (4 digits): 2435
-> 2 matches and 2 hits

*********************************
Sorry, out of queries. Game over!
*********************************
```

Notes:

- If the user enters a non-integer, print the message "**`Invalid query. Please try again!`**" Notice that if the user enters an integer *followed* by noninteger characters, these characters should be simply ignored (check the value returned by the **`scanf`** function). If the user enters an integer that is not valid, print the message "**`Invalid number. Please try again!`**" In both of these cases, you should then prompt the user to enter another query, and keep doing so until the user provides a valid input. The following sample run illustrates all this.

```
/home/userXYZ/ECE15/Lab3> match_and_hit
 ***Welcome to the MATCH and HIT game***
The computer has selected a 4-digit number.
Try to deduce it in 12 rounds of queries.

Round #1
Please enter your query (4 digits): ##
Invalid query. Please try again!
Please enter your query (4 digits): 0##
Invalid number. Please try again!
Please enter your query (4 digits): -5341
Invalid number. Please try again!
Please enter your query (4 digits): 123456789
Invalid number. Please try again!
Please enter your query (4 digits): 5340
Invalid number. Please try again!
Please enter your query (4 digits): 5344
Invalid number. Please try again!
Please enter your query (4 digits): 5341#OK?
-> 2 matches and 1 hit

Round #2
Please enter your query (4 digits): 1235
-> 1 match and 2 hits

Round #3
Please enter your query (4 digits): 2345
-> 4 matches and 0 hits

*********************************
CONGRATULATIONS! You won the game!
*********************************
```

- The maximum number of queries (which we have, so far, assumed to be 12) should be introduced as a symbolic constant **`MAX_QUERIES`** in the beginning of the program, using the **`#define`** compiler directive. The number of digits in a valid integer (which we have, so far, assumed to be 4) should be also introduced as a symbolic constant called **`N_DIGITS`**. You can assume that **`N_DIGITS`** is one of $1, 2, 3, 4$ and **`MAX_QUERIES`** is an integer in the range $1, 2, \ldots, 24$. The program should keep working correctly for all such values of **`N_DIGITS`** and **`MAX_QUERIES`**.

- You can always win the game with at most 12 queries, if you query cleverly. Try it, it's fun! Can you guarantee a win with less than 12 queries? If so, what is the minimum number of queries you need? You do not need to answer these questions, but you are welcome to consider them.