unimportant details and focus on the essential ones. For example, the entire opening paragraphs except the last sentence in UVa 579 - ClockHands are about the history of the clock and is completely unrelated to the actual problem. However, harder problems are usually written as succinctly as possible—they are already difficult enough without additional embellishment.

- **Input and Output description**. In this section, you will be given details on how the input is formatted and on how you should format your output. This part is usually written in a formal manner. A good problem should have clear input constraints as the same problem might be solvable with different algorithms for different input constraints (see Table 1.4).
- **Sample Input and Sample Output**. Problem authors usually only provide trivial test cases to contestants. The sample input/output is intended for contestants to check their basic understanding of the problem and to verify if their code can parse the given input using the given input format and produce the correct output using the given output format. Do not submit your code to the judge if it does not even pass the given sample input/output. See Section 1.2.5 about testing your code before submission.
- **Hints or Footnotes**. In some cases, the problem authors may drop hints or add footnotes to further describe the problem.

1.3.2 Typical Input/Output Routines

Multiple Test Cases

In a programming contest problem, the correctness of your code is usually determined by running your code against *several* test cases. Rather than using many individual test case files, modern programming contest problems usually use *one* test case file with multiple test cases included. In this section, we use a very simple problem as an example of a multiple-test-cases problem: Given two integers in one line, output their sum in one line. We will illustrate three possible input/output formats:

- The number of test cases is given in the first line of the input.
- The multiple test cases are terminated by special values (usually zeroes).
- The multiple test cases are terminated by the EOF (end-of-file) signal.

```
C/C++ Source Code                                | Sample Input | Sample Output
-------------------------------------------------------------------------------
int TC, a, b;                                    | 3            | 3
scanf("%d", &TC); // number of test cases        | 1 2          | 12
while (TC--) { // shortcut to repeat until 0     | 5 7          | 9
  scanf("%d %d", &a, &b); // compute answer      | 6 3          |--------------
  printf("%d\n", a + b); // on the fly           |--------------|
}                                                |              |
-------------------------------------------------------------------------------
int a, b;                                        | 1 2          | 3
// stop when both integers are 0                 | 5 7          | 12
while (scanf("%d %d", &a, &b), (a || b))         | 6 3          | 9
  printf("%d\n", a + b);                         | 0 0          |--------------
```

```
-------------------------------------------------------------------------
int a, b;                                   | 1 2          | 3
// scanf returns the number of items read   | 5 7          | 12
while (scanf("%d %d", &a, &b) == 2)         | 6 3          | 9
// or you can check for EOF, i.e.           |--------------|--------------
// while (scanf("%d %d", &a, &b) != EOF)    |              |
  printf("%d\n", a + b);                    |              |
```

Case Numbers and Blank Lines

Some problems with multiple test cases require the output of each test case to be numbered sequentially. Some also require a blank line *after* each test case. Let's modify the simple problem above to include the case number in the output (starting from one) with this output format: "`Case [NUMBER]: [ANSWER]`" followed by a blank line for each test case. Assuming that the input is terminated by the EOF signal, we can use the following code:

```
C/C++ Source Code                           | Sample Input | Sample Output
-------------------------------------------------------------------------
int a, b, c = 1;                            | 1 2          | Case 1: 3
while (scanf("%d %d", &a, &b) != EOF)       | 5 7          |
  // notice the two '\n'                    | 6 3          | Case 2: 12
  printf("Case %d: %d\n\n", c++, a + b);    |--------------|
                                            |              | Case 3: 9
                                            |              |
                                            |              |--------------
```

Some other problems require us to output blank lines only *between* test cases. If we use the approach above, we will end up with an extra new line at the end of our output, producing unnecessary 'Presentation Error' (PE) verdict. We should use the following code instead:

```
C/C++ Source Code                           | Sample Input | Sample Output
-------------------------------------------------------------------------
int a, b, c = 1;                            | 1 2          | Case 1: 3
while (scanf("%d %d", &a, &b) != EOF) {     | 5 7          |
  if (c > 1) printf("\n"); // 2nd/more cases | 6 3          | Case 2: 12
  printf("Case %d: %d\n", c++, a + b);      |--------------|
}                                           |              | Case 3: 9
                                            |              |--------------
```

Variable Number of Inputs

Let's change the simple problem above slightly. For each test case (each input line), we are now given an integer k ($k \geq 1$), followed by k integers. Our task is now to output the sum of these k integers. Assuming that the input is terminated by the EOF signal and we do not require case numbering, we can use the following code:

```
C/C++ Source Code                           | Sample Input | Sample Output
-------------------------------------------------------------------------
```

```
int k, ans, v;                                  | 1 1          | 1
while (scanf("%d", &k) != EOF) {                | 2 3 4        | 7
  ans = 0;                                      | 3 8 1 1      | 10
  while (k--) { scanf("%d", &v); ans += v; }    | 4 7 2 9 3    | 21
  printf("%d\n", ans);                          | 5 1 1 1 1 1  | 5
}                                               |--------------|--------------
```

Exercise 1.3.1*: What if the problem author decides to make the input *a little more* problematic? Instead of an integer k at the beginning of each test case, you are now required to sum all integers in each test case (each line). Hint: See Section 6.2.

Exercise 1.3.2*: Rewrite all C/C++ source code in this Section 1.3.2 in Java!

1.3.3 Time to Start the Journey

There is no better way to begin your journey in competitive programming than to solve a few programming problems. To help you pick problems to start with among the ≈ 4097 problems in UVa online judge [47], we have listed some of the easiest Ad Hoc problems below. More details about Ad Hoc problems will be presented in the next Section 1.4.

- **Super Easy**
  You should get these problems AC[9] in under 7 minutes[10] each! If you are new to competitive programming, we strongly recommend that you start your journey by solving some problems from this category after completing the previous Section 1.3.2. Note: Since each category contains numerous problems for you to try, we have *highlighted* a maximum of three (3) **must try *** problems in each category. These are the problems that, we think, are more interesting or are of higher quality.

- **Easy**
  We have broken up the 'Easy' category into two smaller ones. The problems in this category are still easy, but just 'a bit' harder than the 'Super Easy' ones.

- **Medium: One Notch Above Easy**
  Here, we list some other Ad Hoc problems that may be slightly trickier (or longer) than those in the 'Easy' category.

- Super Easy Problems in the UVa Online Judge (solvable in under 7 minutes)
  1. UVa 00272 - TEX Quotes (replace all double quotes to TEX() style quotes)
  2. *UVa 01124 - Celebrity Jeopardy* (LA 2681, just echo/re-print the input again)
  3. UVa 10550 - Combination Lock (simple, do as asked)
  4. UVa 11044 - Searching for Nessy (one liner code/formula exists)
  5. **UVa 11172 - Relational Operators *** (ad hoc, very easy, one liner)
  6. UVa 11364 - Parking (linear scan to get l & r, answer $= 2 * (r - l)$)
  7. **UVa 11498 - Division of Nlogonia *** (just use if-else statements)

[9] Do not feel bad if you are unable to do so. There can be many reasons why a code may not get AC.
[10] Seven minutes is just a rough estimate. Some of these problems can be solved with one-liners.