WordChecker

- The aim of this year's project is to create a system that, at its heart, checks the correspondence between the letters of 2 words of equal length.
- Words are understood as sequences of symbols that can be lowercase alphabetical characters (a-z) or uppercase (A-Z), numeric digits (0-9), or the symbols - (hyphen) and _ ('underscore')
  - example of a 20-symbol word: djHD1af9fj7g l-ssOP

WordChecker

- The system reads from standard input a sequence of information and instructions, and produces output strings as appropriate.
- More precisely, the system reads:
  - a *k* value, indicating the length of the words
  - a sequence (of arbitrary length) of words, each of length *k*, constituting the set of **admissible words**
    - let it be assumed that the sequence of words does not contain duplicates
- At that point, a sequence of 'matches' is read from standard input, where the start of each new match is marked by the command (also read from input) *+new_match*

WordChecker

- The sequences of input strings for each match (following the command *+new_part*) are made in the following way:
  - reference word (*k* characters long)
    - assume that the reference word belongs to the set of admissible words
  - maximum *n* number of words to compare with the reference word
  - sequence of words (each *k* characters) to be compared with the reference word
- Occasionally, in the sequence of input strings, the command may appear *+print_filtered*, the effect of which is explained below
- In addition, both during a game and between games, the *+insert_start* and *+insert_end* commands may appear, which enclose between them a sequence of new words to be added to the set of eligible words
  - the added words are also of length *k*, and it is always assumed that there are no duplicate words (not even with respect to words already present in the set of admissible words)

WordChecker

- For each word read *p*, to be compared with the reference word *r*, the programme writes to stdout a sequence of *k* characters made as follows:
  - In the following, we denote the characters of the word *p by p[1*], *p[*2], ... *p[k*], with *r[*1], *r*[2],
    ... *r[k*] those of the word *r*, and with *res[*1], *res[*2], ... *res*[*k*] those of the printed sequence
- For each $1 \leq i \leq k$, we have that
  - *res*[*i*] is the '+' character if the *i-th* character of *p* is equal to the *i-th* character of *r*
    - i.e. if it is true that *p[i*] = *r[i*], then *p[i*] is 'in the correct position'.
  - *res*[*i*] is the character '/' if *p[i] does* not appear anywhere in *r*
  - *res*[*i*] is the '|' character if *p[i]* appears in *r*, but not in the *i-th* position; however, if n_i instances of *p[i*] appear in *r*, if c_i is the number of instances of the character *p[i*] that are in the correct position (clearly $c_i \leq n_i$) and if there are before the i-th character in *p* at least $n_i - c_i$ characters equal to *p*[*i*] that are in the incorrect position, then *res*[*i*] must be / instead of |

WordChecker

- For instance, if
  r = `abcabcabc` (so *r*[1] = a, *r*[2] = b, ...)
  and
  p = `bbaabccbcabc`
  we have:

```
            abcabcabcabc
           bbaabcbcbcabc
 res =  /+|++|++/+++++
```

  - note that *res[*1] = / because in *r* there are only 5 b, *p* has 6, and all subsequent bs a *p*[1] are in the correct place
  - similarly, *res*[10] = / because *r* has 5 c, *p* has 6, 4 of which are in the right place, and there is already a c before *p[*10] (in *p*[7]) that is in the wrong place

WordChecker

- Other examples of comparisons (where the first line is the reference word *r*, the second is *p*, and the third is the output *res*)
  - djPDi939-s__e-s
    gioSON-we2_w234
    /|////|/|/+//|/
  - djPDi939-s__e-s
    kiidsa92KFaa94-
    /|/||/|/////|/|
  - djPDi939-s__e-s
    ewi-n4wp-sesr-v
    |/|/////++/|/+/
  - DIk834k249kaoe_
    48kDkkkf-saancd
    ||+||/+////+///

WordChecker

- If a word is read from standard input that does not belong to the set of admissible ones, the programme writes the string *not_exists* to standard output (NB: this is not counted as an attempt)
- If, on the other hand, the word *r* is read (i.e. if *p* = *r*), then the programme writes *ok*
  (without printing the detailed result of the confrontation) and the match ends
- If, after reading *n* admissible words (with *n,* remember, being the maximum number of words to compare with *r*), none of them were equal to *r*, the programme writes *ko* (after printing the result of the comparison of the last word), and the game ends
- After the game is over:
  - There may be no other words to compare (but there may be the inclusion of new eligible words)
  - If the input is the command *+new_game*, a new game starts

WordChecker

- Every comparison between *p* and *r* produces constraints learned from the comparison
- For example, from the following comparison
  ```
  abcabcabc
  bbaabccbcabc
  /+|+++|++/+++++
  ```
  we learn that b is in positions 2, 5, 8, 11, 14, that there are only 5 b's in r (the sixth b gives rise to /), that c is in positions 6, 9, 12, 15, that it is not in positions 7 and 10, that there are only 5 c's (as before, the sixth gives rise to /), that a is in positions 4 and 13, but is not in position 3
- Similarly, from the following comparison
  ```
  djPDi939-s e-s
  gioSON-we2 w234
  /|////|/|/∓//|/
  ```
  we learn that in *r* there is neither g, nor o, nor S, that in *r there* is at least one i and that this is not in position i, that there is at least one - and that this is not in position 7, etc.

WordChecker

- When, during a game, input reads the command *+print_filtered*, the programme must output, in **lexicographic order**, the set of admissible words that are compatible with the constraints learnt so far in the match, written one per line
  - note that the constraints learnt concern, for each symbol:
    1. if the symbol does not belong to *r*
    2. places where that symbol must appear in *r*
    3. places where that symbol cannot appear in *r*
    4. *minimum* number of times the symbol appears in *r*
    5. *exact* number of times the symbol appears in *r*
       - note that constraint 5 is stronger than constraint 4
  - the symbol order (used to establish the lexicographic order of words) is that specified by the ASCII standard
- In addition, after each comparison, the programme must output the number of admissible words still compatible with the constraints learnt

An example execution

Input received

```
5
8adfs
5sjaH
KS06l
Hi23a
laj74
-s9k0
sm_ks
okauE
+new_part
5sjaH
4
KS06l
had7s
okauE
```

Comments and Expected Output

The words are all of length 5

List of admissible words

Start of new game
Reference word
maximum number of words to compare in this match Output (on 2 rows, in column): /////, 5
Output: not_exists
Output: //|/, 3

An example execution

Input received

```
+print_filtered
+insert start
PsjW5
asHdd
paF7s
+insert_end
-s9k0
sghks
+print_filtered
sm_ks
+insert_start
_fah-
0D7dj
+insert_end
```

Comments and Expected Output

Output (in column): `5sjaH`, `8adfs`, `Hi23a`

New words to be added to eligible words (and those that are compatible with the learnt constraints are to be added to the set of words compatible with the learnt constraints)

Output: `/+//`, `2`
Output: `not_exists`
Output (in column): `5sjaH`, `asHdd`
Output: `|////`, `2`
Output immediately after comparison map (max. number of words reached): `ko`

New words to be added to eligible words

An example execution

Input received

```
+new_partite okauE
3
laj74
+print_filtered
sm_ks
okauE
```

Comments and Expected Output

Start new game
Reference word
maximum number of words to compare in this match Output:
`/|/, 4`
Output (in column): `Hi23a , _fah-, asHdd , okauE`
Output: `///|/, 1`
Output: `ok`