WordChecker

- L'obiettivo del progetto di quest'anno è di realizzare un sistema che, al suo cuore, controlla la corrispondenza tra le lettere di 2 parole di ugual lunghezza.
- Le parole sono intese come sequenze di simboli che possono essere caratteri alfabetici minuscoli (a-z) o maiuscoli (A-Z), cifre numeriche (0-9), oppure i simboli - (trattino) e _ ("underscore")
  - esempio di parola di 20 simboli: djHD1af9fj7g__l-ssOP

WordChecker

- Il sistema legge da standard input una sequenza di informazioni e istruzioni, e produce delle stringhe in output a seconda dei casi.
- Più precisamente, il sistema legge:
  - un valore *k*, che indica la lunghezza delle parole
  - una sequenza (di lunghezza arbitraria) di parole, ognuna di lunghezza *k*, che costituisce l'insieme delle parole ammissibili
    - si dia pure per scontato che la sequenza di parole non contenga duplicati
- A quel punto, viene letta da standard input una sequenza di "partite", in cui l'inizio di ogni nuova partita è marcato dal comando (letto sempre da input) *+nuova_partita*

WordChecker

- Le sequenze di stringhe in input per ogni partita (successive al comando *+nuova_partita*) sono fatte nel seguente modo:
  - parola di riferimento (di lunghezza *k* caratteri)
    - si assuma che la parola di riferimento appartenga all'insieme di parole ammissibili
  - numero *n* massimo di parole da confrontare con la parola di riferimento
  - sequenza di parole (ognuna di *k* caratteri) da confrontare con la parola di riferimento
- Ogni tanto, nella sequenza di stringhe in input, può comparire il comando *+stampa_filtrate*, il cui effetto è spiegato in seguito
- Inoltre, sia durante una partita, che tra una partita e l'altra, possono comparire i comandi *+inserisci_inizio* e *+inserisci_fine* che racchiudono tra di loro una sequenza di nuove parole da aggiungere all'insieme delle parole ammissibili
  - le parole aggiunte sono anch'esse di lunghezza *k*, e si dà sempre per scontato che non ci siano parole duplicate (neanche rispetto alle parole già presenti nell'insieme di quelle ammissibili)

WordChecker

- Per ogni parola letta (che nel seguito indichiamo con *p*), da confrontare con la parola di riferimento (che nel seguito indichiamo con *r*), il programma scrive su standard output una sequenza di *k* caratteri fatta nella seguente maniera.
  - nel seguito, indichiamo con *p*[1], *p*[2], … *p*[*k*] i caratteri della parola *p*, con *r*[1], *r*[2], … *r*[*k*] quelli della parola *r*, e con *res*[1], *res*[2], … *res*[*k*] quelli della sequenza stampata
- Per ogni $1 \leq i \leq k$, si ha che
  - *res*[*i*] è il carattere + se l'*i*-esimo carattere di *p* è uguale all'*i*-esimo carattere di *r*
    - cioè se vale che *p*[*i*] = *r*[*i*], quindi *p*[*i*] è "in posizione corretta"
  - *res*[*i*] è il carattere / se l'*i*-esimo carattere di *p* non compare da nessuna parte in *r*
  - *res*[*i*] è il carattere | se l'*i*-esimo carattere di *p* (indicato nel seguito come *p*[*i*]) compare in *r*, ma non in posizione *i*-esima; tuttavia, se in *r* compaiono n_i istanze di *p*[*i*], se c_i è il numero di istanze del carattere *p*[*i*] che sono in posizione corretta (chiaramente deve valere che $c_i \leq n_i$) e se ci sono prima del carattere i-esimo in *p* almeno n_i-c_i caratteri uguali a *p*[*i*] che sono in posizione scorretta, allora *res*[*i*] deve essere / invece che |

WordChecker

- Per esempio, se
  r = a b c a b c a b c a b c a b c
  (quindi *r*[1] = a, *r*[2] = b, ecc.) e
  p = b b a a b c c b c c b c a b c
  (con *p*[1] = b, *p*[2] = b, …), abbiamo che res risulta la seguente sequenza:

  a b c a b c a b c a b c a b c
  b b a a b c c b c c b c a b c
  / + | + + + | + + / + + + + +

  - si noti che *res*[1] = / perché in *r* ci sono solo 5 b, *p* ne ha 6, e tutte le b successive a *p*[1] sono nel posto corretto
  - similmente, *res*[10] = / perché *r* ha 5 c, *p* ne ha 6, di cui 4 al posto giusto, e c'è già una c prima di *p*[10] (in *p*[7]) che è al posto sbagliato

WordChecker

- Altri esempi di confronti (dove la prima riga è la parola di riferimento *r*, la seconda è *p*, e la terza è l'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

- Se da standard input viene letta una parola che non appartiene all'insieme di quelle ammissibili, il programma scrive su standard output la stringa *not_exists*
- Se invece viene letta la parola *r* (cioè se *p* = *r*), allora il programma scrive *ok* (senza stampare il risultato dettagliato del confronto) e la partita termina
- Se, dopo avere letto *n* parole ammissibili (con *n*, si ricordi, numero massimo di parole da confrontare con *r*), nessuna di queste era uguale a *r*, il programma scrive *ko* (dopo avere stampato il risultato del confronto dell'ultima parola), e la partita termina
- Dopo che la partita è finita:
  - Non ci possono essere altre parole da confrontare (ma ci potrebbe essere l'inserimento di nuove parole ammissibili)
  - Se in input c'è il comando *+nuova_partita*, ha inizio una nuova partita

WordChecker

- Ogni confronto tra *p* e *r* produce dei vincoli appresi dal confronto
- Per esempio, dal seguente confronto
  ```
  abcabcabcabcabc
  bbaabccbccbcabc
  /+|+++|++/+++++
  ```
  si apprende che b è in posizioni 2, 5, 8, 11, 14, che ci sono solo 5 b in r (la sesta b dà luogo a /), che c è in posizioni 6, 9, 12, 15, che non è in posizioni 7 e 10, che ci sono solo 5 c (come prima, la sesta dà luogo a /), che a è in posizioni 4 e 13, ma non è in posizione 3
- In maniera analoga, dal seguente confronto
  ```
  djPDi939-s__e-s
  gioSON-we2_w234
  /|////|/|/∓//|/
  ```
  si apprende che in *r* non ci sono g, né o, né S, che in *r* c'è almeno una i e che questa non è in posizione i, che c'è almeno un - e che non è in posizione 7, ecc.

WordChecker

- Quando, durante una partita, da input si legge il comando *+stampa_filtrate*, il programma deve produrre in output, in **ordine lessicografico**, l'insieme delle parole ammissibili che sono compatibili con i vincoli appresi fino a quel momento nella partita, scritte una per riga
  - si noti che i vincoli appresi riguardano, per ogni simbolo:
    1. se il simbolo non appartiene a *r*
    2. posti in cui quel simbolo deve comparire in *r*
    3. posti in cui quel simbolo non può comparire in *r*
    4. numero *minimo* di volte che il simbolo compare in *r*
    5. numero *esatto* di volte che il simbolo compare in *r*
       - si noti che il vincolo 5 è più forte del vincolo 4
  - l'ordine dei simboli (usato per stabilire l'ordine lessicografico delle parole) è quello specificato dallo standard ASCII
- Inoltre, dopo ogni confronto, il programma deve stampare in output il numero di parole ammissibili ancora compatibili con i vincoli appresi

Un'esecuzione d'esempio

Input ricevuto	**Commenti e Output Atteso**
`5`	Le parole sono tutte di lunghezza 5 simboli
`8adfs`	
`5sjaH`	
`KS06l`	
`Hi23a`	Elenco di parole ammissibili
`laj74`	
`-s9k0`	
`sm_ks`	
`okauE`	
`+nuova_partita`	Inizio nuova partita
`5sjaH`	Parola di riferimento
`4`	numero massimo di parole da confrontare in questa partita
`KS06l`	Output (su 2 righe, in colonna): `/////, 5`
`had7s`	Output: `not_exists`
`okauE`	Output: `//|//, 3`

Un'esecuzione d'esempio

Input ricevuto	**Commenti e Output Atteso**

`+stampa_filtrate`
`+inserisci_inizio`
`PsjW5`
`asHdd`
`paF7s`
`+inserisci_fine`
`-s9k0`
`sghks`
`+stampa_filtrate`
`sm_ks`
`+inserisci_inizio`
`_fah-`
`0D7dj`
`+inserisci_fine`
	Output (in colonna): `5sjaH, 8adfs, Hi23a`
	
	Nuove parole da aggiungere a quelle ammissibili (e quelle che sono compatibili con i vincoli appresi sono da aggiungere all'insieme delle parole compatibili con i vincoli appresi)
	
	Output: `/+///, 2`
	Output: `not_exists`
	Output (in colonna): `5sjaH, asHdd`
	Output: `|////, 2`
	Output subito dopo mappa confronto (raggiunto numero massimo parole): `ko`
	
	Nuove parole da aggiungere a quelle ammissibili

Un'esecuzione d'esempio

Input ricevuto	**Commenti e Output Atteso**
`+nuova_partita`	Inizio nuova partita
`okauE`	Parola di riferimento
`3`	numero massimo di parole da confrontare in questa partita
`laj74`	Output: `/|///, 4`
`+stampa_filtrate`	Output (in colonna): `Hi23a, _fah-, asHdd, okauE`
`sm_ks`	Output: `///|/, 1`
`okauE`	Output: `ok`