# Laboratorio di Algoritmi e Strutture Dati

Esercitazioni del 15 Gennaio 2013

## Esercizio 2: readword

Si tratta di implementare un algoritmo per la lettura e la memorizzazione di *parole* di lunghezza *arbitraria* (vale a dire *non limitata a priori*).

Per *parola* si intende una sequenza di caratteri consecutivi che non siano di spaziatura o di punteggiatura.

Si ricorda di includere il file di intestazione standard `ctype.h` per utilizzare:

`int isspace(int c)` che restituisce un valore $> 0$ se `c` è di spaziatura, 0 altrimenti;
`int ispunct(int c)` che restituisce un valore $> 0$ se `c` è di punteggiatura, 0 altrimenti;

Il programma deve poter essere lanciato come:

`readword` *file*

dove *file* è un arbitrario file di testo (vedi `input.txt` nel materiale didattico predisposto).

Il programma deve:

1. Leggere la prossima parola dal *file* e memorizzarla in un'opportuna stringa allocata dinamicamente. Deve fare questo tramite un'opportuna funzione:

   ```
   char *readword(FILE *fp)
   ```

   che restituisca l'indirizzo del primo carattere della stringa allocata dinamicamente.

2. Stampare la parola sullo schermo secondo il formato seguente:

   {
   *parola*
   }

3. Iterare dal punto 1 fino a raggiungere la fine del file (`EOF`).

### Suggerimenti implementativi: lista di buffer

Per memorizzare una parola si utilizzi una *lista di buffer*, come di seguito descritto:

- Si predisponga una macro che fissi la dimensione scelta per un buffer (un array) di caratteri. Ad esempio:

  ```
  #define DIM 10
  ```

- Si definisca la struttura di un nodo della lista di buffer come:

```
struct bufferlistelement {
    char mem[DIM];
    struct bufferlistelement *next;
};
```

- Utilizzando inserimenti in testa alla lista di buffer si gestisca l'input nel modo qui di seguito esemplificato:

  Supponiamo che la parola da leggere sia `precipitevolissimevolmente`. Allora si proceda come descritto:

  1. Si inserisca un primo elemento nella lista:
     `[?,?,?,?,?,?,?,?,?,?]<--root`
  2. Si leggano `DIM` caratteri dalla parola (nel nostro esempio `DIM` vale 10), e li si immettano nell'array `mem` dell'elemento puntato dalla radice `root` della lista:
     `[p,r,e,c,i,p,i,t,e,v]<--root`
  3. Dal momento che non si è ancora incontrata la fine della parola, si inserisca un nuovo elemento in testa alla lista:
     `[p,r,e,c,i,p,i,t,e,v]<--[?,?,?,?,?,?,?,?,?,?]<--root`
  4. Si leggano `DIM` caratteri dalla parola e li si immettano nell'array `mem` dell'elemento puntato dalla radice `root` della lista:
     `[p,r,e,c,i,p,i,t,e,v]<--[o,l,i,s,s,i,m,e,v,o]<--root`
  5. Dal momento che non si è ancora incontrata la fine della parola, si inserisca un nuovo elemento in testa alla lista:
     `[p,r,e,c,i,p,i,t,e,v]<--[o,l,i,s,s,i,m,e,v,o]<--[?,?,?,?,?,?,?,?,?,?]<--root`
  6. Si leggano i restanti caratteri e li si inseriscano nell'array `mem` dell'elemento puntato dalla radice `root` della lista:
     `[p,r,e,c,i,p,i,t,e,v]<--[o,l,i,s,s,i,m,e,v,o]<--[l,m,e,n,t,e,?,?,?,?]<--root`

- A questo punto è facile calcolare la lunghezza totale della parola.
  Si crei in memoria dinamica uno spazio adeguato (lunghezza +1 per il carattere di terminazione) e vi si vada a copiare la parola, buffer per buffer, *nelle posizioni opportunamente calcolate*.
  A questo scopo può tornare utile la funzione di libreria standard:

  ```
  char *strncpy(char *s1, const char *s2, size_t n)
  ```

  che copia esattamente `n` caratteri da `s2` a `s1` e restituisce l'indirizzo `s1` (NB: `size_t` è un sinonimo di un tipo intero). Il prototipo di `strncpy` è in `string.h`.

- Alla fine si restituisca l'indirizzo dove si è memorizzata la parola, non prima di aver deallocato tutta la lista di buffer.

Per implementare le necessarie funzionalità per la gestione della lista di buffer potete ispirarvi e modificare l'implementazione `intlist.c` vista per liste di interi.

**Supplemento 1**

I caratteri di spaziatura e di punteggiatura possono essere considerati *separatori* di parole. Cosa succede se scegliamo un alto insieme di *separatori*?

- Aggiungere un parametro `separator` alla funzione `readlongword`, dove `separator` è un puntatore a una funzione che ricevuto un carattere come argomento restituisce un valore booleano:

  ```
  char *readword(FILE *fp, char (*separator)(char))
  ```

  in modo tale che `readword` utilizzi la funzione puntata da `separator` per decidere se un carattere è o meno da considerarsi un separatore.

- Scrivere un'opportuna funzione

  ```
  char spazio_o_punto(char c)
  ```

  che restituisca 1 se `c` è di spaziatura o di punteggiatura, 0 altrimenti.

- Scrivere un'opportuna funzione

  ```
  char non_maiuscola(char c)
  ```

  che restituisca 1 se `c` non è una lettera maiuscola, 0 altrimenti.

- Modificare il codice in modo tale che la funzione `readword` venga richiamata passandole un puntatore a una delle due funzioni `spazio_o_punto`, `non_maiuscola`.

**Supplemento 2**

Utilizzare `readword` opportunamente nel programma `sw.c` (visto a lezione tempo fa) che ordina liste di lunghezza arbitraria di parole. Si otterrà un programma per ordinare liste di lunghezza arbitraria di parole di lunghezza arbitraria.

Si confronti il metodo della lista di buffer con il metodo delle riallocazioni crescenti usato in `sw.c`: si rifletta su pregi e difetti dei due approcci.