Programmazione I (Tucci/Distasi)

PR1 PRE MT/RD 12/01/2023

Modello: 1

Cognome: _____

Nome: _____

Matricola: _____

Email: _____

Regole del gioco: Compilare i dati personali prima d'incominciare. Una volta iniziata la prova, non è consentito lasciare l'aula. Usare questi stessi fogli (compreso il retro, dove necessario) per rispondere. *Buon lavoro!*

1. Un file di testo contiene nella prima riga un numero intero n, e di seguito n linee di testo, ognuna lunga al più `MAXLINE` caratteri.

   Scriviamo una funzione `read_text()` che legge il file (già aperto in lettura) e restituisce un array di n stringhe: un elemento per ogni linea di testo. L'array e i suoi elementi devono essere allocati dinamicamente. Inoltre, la funzione `read_text()` usa un parametro output per salvare la dimensione dell'array restituito.

2. Immaginiamo di avere due file di testo già aperti, rispettivamente in lettura e in scrittura: `fin` e `fout`. Il file puntato da `fin` contiene una serie di righe di testo, ognuna lunga al più `MAXLINE` caratteri. Scriviamo una funzione

   ```
   int copy_v(FILE *fin, FILE *fout)
   ```

   che copia nel file puntato da `fout` le righe che contengono più vocali che consonanti. La funzione restituisce il numero di righe copiate. Le righe potrebbero contenere spazi, quindi la lettura è più agevole con `fgets()` che con `fscanf()`.

   Scriviamo una funzione separata per valutare se una certa riga è da copiare o no. Abbiamo a disposizione due funzioni "di libreria"

   ```
   int is_vowel(char c)    int is_consonant(char c),
   ```

   che restituiscono nonzero (vero) se il carattere `c` è rispettivamente una vocale/una consonante, zero altrimenti.

   Facoltativo Scriviamo anche `is_vowel()` e `is_consonant()`. Ci potrebbero essere utili alcune funzioni dichiarate in `ctype.h`: `isalpha(c)` (`c` è un carattere alfabetico?) e `tolower(c)`/`toupper(c)` (trasforma `c` in minuscola/maiuscola).

Risposte per il modello 1

1. Un file di testo contiene nella prima riga un numero intero n, e di seguito n linee di testo, ognuna lunga al più `MAXLINE` caratteri.

   Scriviamo una funzione `read_text()` che legge il file (già aperto in lettura) e restituisce un array di n stringhe: un elemento per ogni linea di testo. L'array e i suoi elementi devono essere allocati dinamicamente. Inoltre, la funzione `read_text()` usa un parametro output per salvare la dimensione dell'array restituito.

   Risposta Ecco una possibile soluzione.

```
char **read_text(FILE * f, int *arraysize)
{
  int i, size;
  char **lines;
  char buffer[MAXLINE + 1], *s;

  fscanf(f, "%d", &size);
  lines = malloc(size * sizeof(char **));
  if (lines == NULL)
    {
      fprintf(stderr, "malloc() failed. Exiting.\n");
      *arraysize = 0;
      return NULL;
    }
  for (i = 0; i < size; i++)
    {
      s = fgets(buffer, MAXLINE + 1, f);
      if (s == NULL)            // if fgets() is OK, s == buffer
        {
          fprintf(stderr, "gets() failed, i=%d\n", i);
      } else
        {
          lines[i] = strdup(buffer);
          /* or:
              lines[i] = malloc(strlen(buffer)+1);
              strcpy(lines[i], buffer);
          */
          if (lines[i] == NULL) // malloc() in strdup() failed
            {
              fprintf(stderr, "strdup() failed, i=%d\n", i);
              *arraysize = i;
              return lines;    // return what we can
            }
        }
    }
  *arraysize = size;
  return lines;
}
```

2. Immaginiamo di avere due file di testo già aperti, rispettivamente in lettura e in scrittura: `fin` e `fout`. Il file puntato da `fin` contiene una serie di righe di testo, ognuna lunga al più `MAXLINE` caratteri. Scriviamo una funzione

   ```
   int copy_v(FILE *fin, FILE *fout)
   ```

   che copia nel file puntato da `fout` le righe che contengono più vocali che consonanti. La funzione restituisce il numero di righe copiate. Le righe potrebbero contenere spazi, quindi la lettura è più agevole con `fgets()` che con `fscanf()`.

   Scriviamo una funzione separata per valutare se una certa riga è da copiare o no. Abbiamo a disposizione due funzioni "di libreria"

   ```
   int is_vowel(char c)    int is_consonant(char c),
   ```

   che restituiscono nonzero (vero) se il carattere `c` è rispettivamente una vocale/una consonante, zero altrimenti.

   Facoltativo Scriviamo anche `is_vowel()` e `is_consonant()`. Ci potrebbero essere utili alcune funzioni dichiarate in `ctype.h`: `isalpha(c)` (`c` è un carattere alfabetico?) e `tolower(c)`/`toupper(c)` (trasforma `c` in minuscola/maiuscola).

   Risposta Ecco una possibile soluzione.

```
/* "library" prototypes */
int is_vowel(char c);           // is c a vowel?
int is_consonant(char c);       // is c a consonant?

int is_vowelish(char *s)        //does s have more vowels than consonants?
{
  int i, vowels, consonants;

  vowels = consonants = 0;
  for (i = 0; s[i] != '\0'; i++)
    {
      if (is_vowel(s[i]))
        {
          vowels++;
        }
      else if (is_consonant(s[i]))
        {
          consonants++;
        }
    }
  return (vowels > consonants);
}

int copy_v(FILE * fin, FILE * fout)
{
  char buffer[MAXLINE + 1];     // we read our lines here
  char *s;                      // when OK, fgets() return value is == buffer
  int copied = 0;               // count copied lines

  while ((s = fgets(buffer, MAXLINE + 1, fin)) != NULL) // NULL: EOF
    {
      if (is_vowelish(buffer))
        {
          fprintf(fout, "%s", buffer);
          copied++;
        }
    }
  return copied;
}
```

```
int is_vowel(char c)
{
  switch (toupper(c))
    {
    case 'A':
    case 'E':
    case 'I':
    case 'O':
    case 'U':
    case 'Y':
      return 1;
    }
  return 0;
}

int is_consonant(char c)
{
  return (
      isalpha(c)                // it needs to be a letter
      && !is_vowel(c)           // but not a vowel
      );
}
```

Notiamo che questa versione di `copy_v()` non controlla il successo della scrittura. Ricordando che `fprintf()` restituisce il numero di caratteri scritti, come potremmo realizzare questo controllo?

Come potremmo scrivere `is_vowel()` senza usare `toupper()` o `tolower()`? E senza usare uno `switch`?

Quanto si complica la scrittura di `is_vowelish()` se non possiamo usare funzioni ausiliarie come `is_vowel()` per dividerla in sottoproblemi?