Programmazione I (Tucci/Distasi)

PR1 MT/RD 14/02/2020

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. (a) (**Corso da 9 crediti**) La funzione `diagon_acrostic()` prende come parametro un array di `n` stringhe `strings[]`, unitamente alla sua dimensione `n`. Il risultato è una nuova stringa, allocata dinamicamente, che ha in prima posizione il primo carattere della prima stringa, in seconda posizione il secondo carattere della seconda stringa, e così via: in ultima posizione si troverà l'`n`-esimo carattere dell'`n`-esima stringa.

   Per esempio:

   ```
   stringhe[] = { "abc", "definitivo", "ghiro" }
   diagon_acrostic(stringhe, 3) = "aei"
   ```

   Ognuna delle stringhe in `strings[]` dev'essere lunga almeno `n` caratteri. Se la funzione scopre un elemento di `strings[]` che è lungo meno di `n`, restituisce `NULL`. Scrivere `diagon_acrostic()`.

   (b) (**Corso da 12 crediti**) Consideriamo una lista di nodi definiti come segue.

   ```
   typedef struct nodo
   {
     char *string;
     struct nodo *next;
   } Nodo;
   ```

   La funzione `list_acrostic()` prende come parametro una lista siffatta, unitamente alla sua lunghezza `n`. Il risultato è una nuova stringa, allocata dinamicamente, che ha in prima posizione il primo carattere della prima stringa, in seconda posizione il secondo carattere della seconda stringa, e così via: in ultima posizione si troverà l'`n`-esimo carattere dell'`n`-esima stringa.

   Per esempio:

   ```
   lista = { "abc", "definitivo", "ghiro" }
   list_acrostic(lista, 3) = "aei"
   ```

   Ognuna delle stringhe in `lista` dev'essere lunga almeno `n` caratteri. Se la funzione scopre un elemento di `lista` in cui il campo `string` è lungo meno di `n`, restituisce `NULL`. Scrivere `list_acrostic()`.

2. Un file di testo, già aperto per la lettura, contiene una serie di stringhe prive di spazi, una per riga, ognuna lunga al più `MAX` caratteri (`MAX` è una costante che assumiamo predefinita).

   (a) Scrivere una funzione

   ```
   conta_stringhe_corte(FILE *fp, int limite)
   ```

   che legge il file e restituisce il numero di stringhe di lunghezza inferiore a `limite`.

   (b) Scrivere una funzione

   ```
   stringhe_corte(FILE *fp, int limite, int *nstringhe)
   ```

   che legge il file e salva le sole stringhe di lunghezza inferiore a `limite` in un array allocato dinamicamente. La funzione restituisce l'array così costruito e usa il parametro output `nstringhe` per comunicare il numero di elementi nell'array. Usare la funzione scritta al punto precedente.

Risposte per il modello 1

1. (a) (**Corso da 9 crediti**) La funzione `diagon_acrostic()` prende come parametro un array di `n` stringhe `strings[]`, unitamente alla sua dimensione `n`. Il risultato è una nuova stringa, allocata dinamicamente, che ha in prima posizione il primo carattere della prima stringa, in seconda posizione il secondo carattere della seconda stringa, e così via: in ultima posizione si troverà l'`n`-esimo carattere dell'`n`-esima stringa.

   Per esempio:

   ```
   stringhe[] = { "abc", "definitivo", "ghiro" }
   diagon_acrostic(stringhe, 3) = "aei"
   ```

   Ognuna delle stringhe in `strings[]` dev'essere lunga almeno `n` caratteri. Se la funzione scopre un elemento di `strings[]` che è lungo meno di `n`, restituisce `NULL`. Scrivere `diagon_acrostic()`.

   (b) (**Corso da 12 crediti**) Consideriamo una lista di nodi definiti come segue.

   ```
   typedef struct nodo
   {
     char *string;
     struct nodo *next;
   } Nodo;
   ```

   La funzione `list_acrostic()` prende come parametro una lista siffatta, unitamente alla sua lunghezza `n`. Il risultato è una nuova stringa, allocata dinamicamente, che ha in prima posizione il primo carattere della prima stringa, in seconda posizione il secondo carattere della seconda stringa, e così via: in ultima posizione si troverà l'`n`-esimo carattere dell'`n`-esima stringa.

   Per esempio:

   ```
   lista = { "abc", "definitivo", "ghiro" }
   list_acrostic(lista, 3) = "aei"
   ```

   Ognuna delle stringhe in `lista` dev'essere lunga almeno `n` caratteri. Se la funzione scopre un elemento di `lista` in cui il campo `string` è lungo meno di `n`, restituisce `NULL`. Scrivere `list_acrostic()`.

Risposta
Ecco una possibile soluzione.

```
void *xmalloc(size_t nbytes)    // malloc() con controllo errore
{
  void *result;

  if ((result = malloc(nbytes)) == NULL)
    {
      fprintf(stderr, "malloc(%lu) failed. Exiting.\n", nbytes);
      exit(-1);
    }
  return result;
}
```

```
char *diagon_acrostic(char *strings[], int n)
{
  char *result;
  int i;

  result = xmalloc(n + 1);      // n caratteri piu' terminatore
  for (i = 0; i < n; i++)
    {
      if (strlen(strings[i]) < n)
        {
          fprintf(stderr, "strings[%d] troppo corta\n", i);
          free(result);
          return NULL;
        }
      // else
      result[i] = strings[i][i];
    }
  result[i] = '\0';
  return result;
}

char *list_acrostic(Nodo * p, int n)
{
  char *result;
  int i;

  result = xmalloc(n + 1);      // n caratteri piu' terminatore
  i = 0;
  while (p != NULL)
    {
      if (strlen(p->string) < n)
        {
          fprintf(stderr, "strings[%d] troppo corta\n", i);
          free(result);
          return NULL;
        }
      // else
      result[i] = p->string[i];
      p = p->next;
      i++;
    }
  result[i] = '\0';
  return result;
}
```

2. Un file di testo, già aperto per la lettura, contiene una serie di stringhe prive di spazi, una per riga, ognuna lunga al più `MAX` caratteri (`MAX` è una costante che assumiamo predefinita).

   (a) Scrivere una funzione

   ```
   conta_stringhe_corte(FILE *fp, int limite)
   ```

   che legge il file e restituisce il numero di stringhe di lunghezza inferiore a `limite`.

   (b) Scrivere una funzione

   ```
   stringhe_corte(FILE *fp, int limite, int *nstringhe)
   ```

   che legge il file e salva le sole stringhe di lunghezza inferiore a `limite` in un array allocato dinamicamente. La funzione restituisce l'array così costruito e usa il parametro output `nstringhe` per comunicare il numero di elementi nell'array. Usare la funzione scritta al punto precedente.

Risposta Ecco una possibile soluzione.

```
int conta_stringhe_corte(FILE * fp, int limite)
{
  char buf[MAX];                // leggiamo l'input qui
  int fscanf_ok = 1;            // diventa 0 alla fine del file
  int conta_corte = 0;          // risultato: numero di stringhe corte

  while (1)                     // contiamo le stringhe corte
    {
      fscanf_ok = fscanf(fp, "%s", &buf[0]); // 1 se letto OK, else 0
      if (fscanf_ok != 1)       // file finito?
        {
          break;
        }
      // else
      if (strlen(buf) < limite) // questa stringa e' corta?
        {
          conta_corte++;
        }
    }
  return conta_corte;
}
```

```
char **stringhe_corte(FILE * fp, int limite, int *nrighe)
{
  char buf[MAX];                // leggiamo l'input qui
  char **result;                // sara' array di stringhe restituito
  int fscanf_ok = 1;            // diventa 0 alla fine del file
  size_t len;
  int i, ncorte;

  ncorte = conta_stringhe_corte(fp, limite);
  *nrighe = ncorte;             // sistemiamo parametro output
  // crea spazio per array di ncorte stringhe (puntatori)
  result = xmalloc(ncorte * sizeof(char *));
  rewind(fp);                   // leggiamo file da capo
  fscanf_ok = 1;
  i = 0;
  while (1)                     // copiamo stringhe corte in result[]
    {
      fscanf_ok = fscanf(fp, "%s", &buf[0]); // 1 se letto OK, else 0
      if (fscanf_ok != 1)       // file finito?
        {
          break;
        }
      // else
      if ((len = strlen(buf)) < limite) // questa stringa e' corta?
        {
          result[i] = xmalloc(len + 1); // spazio per caratteri+tappo
          strcpy(result[i], buf);
          i++;
        }
    }
  return result;
}
```