Programmazione I (Tucci/Distasi)

PR1 MT/RD 17/02/2021

Modello: 1

Cognome: _____

Nome: _____

Matricola: _____

Email: _____

Regole del gioco: Compilare i dati personali prima d'incominciare. Alla fine della prova, inviare un singolo file pdf (non immagini separate, non un link web) all'indirizzo email

`prog1unisa.aula1@gmail.com`

Buon lavoro!

1. Scrivere una funzione

   ```
   double * load_doubles (FILE *fp, int *n)
   ```

   che legge un file binario e crea un array di numeri `double` allocato dinamicamente. Il file in questione contiene come primo dato un numero intero che indica il numero di numeri `float` presenti, poi i numeri stessi. Un esempio:

   6, 8.8, 6.1, 4.12, 2.1, -0.5, -2.1.

   L'array creato in questo caso avrà dimensione 6. Esso conterrà i numeri in virgola mobile elencati nell'esempio, da 8.8 a -2.1. La funzione restituisce l'array così ottenuto, e usa il parametro output `n` per comunicarne la dimensione.

   In caso di problemi di qualsiasi tipo (per esempio: file di formato errato o fallita allocazione memoria), la funzione restituisce `NULL`, stampando un messaggio di errore che specifica l'errore incontrato.

2. Scrivere una funzione

   ```
   double media_negativi (FILE *fin)
   ```

   che usa la funzione scritta al punto precedente per leggere un file binario e restituisce la media degli elementi in virgola mobile minori di zero. In caso di problemi di qualsiasi tipo, la funzione restituisce un valore qualsiasi, ma maggiore di zero.

   Nel caso fornito, in assenza di errori il valore restituito sarà -1.3, che è la media fra -0.5 e -2.1.

Risposte per il modello 1

1. Scrivere una funzione

   ```
   double * load_doubles (FILE *fp, int *n)
   ```

   che legge un file binario e crea un array di numeri `double` allocato dinamicamente. Il file in questione contiene come primo dato un numero intero che indica il numero di numeri `float` presenti, poi i numeri stessi. Un esempio:

   6, 8.8, 6.1, 4.12, 2.1, -0.5, -2.1.

   L'array creato in questo caso avrà dimensione 6. Esso conterrà i numeri in virgola mobile elencati nell'esempio, da 8.8 a -2.1. La funzione restituisce l'array così ottenuto, e usa il parametro output `n` per comunicarne la dimensione.

   In caso di problemi di qualsiasi tipo (per esempio: file di formato errato o fallita allocazione memoria), la funzione restituisce `NULL`, stampando un messaggio di errore che specifica l'errore incontrato.

   Risposta
   Ecco una possibile soluzione.

```
#include <stdio.h>
#include <stdlib.h>

void simple_error(char *function_name, char *err_msg)
{
  fprintf(stderr, "error: %s in function %s()\n", err_msg, function_name);
}

double *load_doubles(FILE * fp, int *n)
{
  int size;                     // size of the new array
  double *array;
  int status;
  char *thisfunct = "load_doubles";     // for error messages

  status = fread(&size, sizeof(int), 1, fp);
  if (status != 1)
    {
      simple_error(thisfunct, "can't read size from file");
      return NULL;
    }
  array = malloc(sizeof(double) * size);
  if (array == NULL)
    {
      simple_error(thisfunct, "malloc() failed");
      return NULL;
    }
  status = fread(array, sizeof(double), size, fp);
  if (status != size)
    {
      simple_error(thisfunct, "not enough data in file");
      return NULL;
    }
  // else, everything is OK!
  *n = size;
  return array;
}
```

2. Scrivere una funzione

```
double media_negativi (FILE *fin)
```

che usa la funzione scritta al punto precedente per leggere un file binario e restituisce la media degli elementi in virgola mobile minori di zero. In caso di problemi di qualsiasi tipo, la funzione restituisce un valore qualsiasi, ma maggiore di zero.

Nel caso fornito, in assenza di errori il valore restituito sarà -1.3, che è la media fra -0.5 e -2.1.

Risposta Ecco una possibile soluzione.

```
#include <stdio.h>
#include <stdlib.h>

double *load_doubles(FILE * fp, int *n);

double media_negativi(FILE * fin)
{
  double *array;
  int size;
  int i, n_negatives;
  double sum;

  array = load_doubles(fin, &size);
  if (array == NULL)            // any problems?
    {
      return 1.0;               // signal problems: return something > 0.0
    }
  // else
  sum = 0.0;
  n_negatives = 0;
  for (i = 0; i < size; i++)
    {
      if (array[i] < 0.0)       // only negative numbers matter
        {
          sum += array[i];
          n_negatives++;
        }
    }
  free(array);                  // used only here, don't leak memory
  if (n_negatives == 0)
    {
      return 0.0;               // signal: no negative numbers
    }
  return sum / n_negatives;     // "normal" case: average
}
```