# Programmazione I (Tucci/Distasi)

PR1 MT/RD 22/01/2019

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. Scrivere una funzione

   ```
   int * negatives_last(int *x, int size, int *n_negatives)
   ```

   che riceva come parametri un array `x[]` di interi e la sua taglia `size`, restituendo un vettore di pari taglia in cui gli elementi negativi sono stati spostati alla fine. La funzione usa il parametro output `n_negatives` per comunicare il numero di elementi negativi presenti alla fine dell'array restituito. L'ordine preciso degli elementi non è importante, purché tutti gli elementi positivi precedano tutti i negativi. L'array restituito deve essere allocato dinamicamente.

2. - A [Esame da 9 crediti] Nella directory di lavoro attuale c'è un file di testo che contiene una serie di numeri interi, uno per riga. Scrivere una funzione

     ```
     int * make_int_array(char *fname, int *size)
     ```

     che prende come parametro il nome del file, inserisce gli interi in un array allocato dinamicamente e lo restituisce, usando il parametro output `size` per comunicarne la dimensione al chiamante.

     *Suggerimento* Sarà necessario leggere il file due volte: la prima per contare i numeri e dimensionare correttamente l'array, la seconda per scrivere i numeri nell'array così creato. Per rileggere il file dall'inizio, si può chiuderlo e riaprirlo.

   - B [Esame da 12 crediti] Nella directory di lavoro attuale c'è un file di testo che contiene una serie di parole di lunghezza arbitraria (non limitata), separate da spazi, newline o tab. Scrivere una funzione

     ```
     Nodo * make_word_list(char *fname)
     ```

     che prende come parametro il nome del file, crea una lista di nodi, uno per ogni parola, e restituisce la lista così ottenuta. Definire il tipo di dato `Nodo` in modo opportuno.

     *Suggerimento* Bisogna allocare dinamicamente non solo ogni nodo, ma anche lo spazio per le stringhe puntate: esse possono essere di lunghezza arbitraria.

     *Advanced* Per evitare buffer overflow durante la lettura, si può usare `fscanf()` con l'opzione `%`⟨*lunghezza*⟩`s` nella specifica di conversione. In questo modo, ogni parola sarà troncata e i caratteri letti non supereranno i limiti del buffer. Per esempio, `"%16383s"` troncherebbe ogni parola in modo da rientrare in un buffer di 16 kB ($16383 = 16 \times 1024 - 1$; l'ultimo carattere rimane disponibile per il terminatore `'\0'`).

# Risposte per il modello 1

1. Scrivere una funzione

   ```
   int * negatives_last(int *x, int size, int *n_negatives)
   ```

   che riceva come parametri un array `x[]` di interi e la sua taglia `size`, restituendo un vettore di pari taglia in cui gli elementi negativi sono stati spostati alla fine. La funzione usa il parametro output `n_negatives` per comunicare il numero di elementi negativi presenti alla fine dell'array restituito. L'ordine preciso degli elementi non è importante, purché tutti gli elementi positivi precedano tutti i negativi. L'array restituito deve essere allocato dinamicamente.

   **Risposta** Ecco una possibile soluzione.

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

/*
 * move negative numbers at the end,
 * count them and report in n_negatives
 */
int *negatives_last(int *x, int size, int *n_negatives)
{
  int i;                        // index in source array
  int ni, nj;                   // indices in dest. array, nonnegative/negative
  int *newx;                    // the destination array

  newx = malloc(size * sizeof(int));
  if (newx == NULL)
    {
      fprintf(stderr, "malloc(%ld) failed. Bye!\n", size * sizeof(int));
      exit(-1);
    }
  ni = 0;                       // nonnegative numbers found in x[]
  nj = 0;                       // negative numbers found in x[]
  for (i = 0; i < size; i++)
    {
      if (x[i] < 0)             // negative, goes at the end
        {
          newx[size - 1 - nj] = x[i];
          nj++;                 // count one more negative number
      } else
        {
          newx[ni] = x[i];
          ni++;                 // count one more nonnegative number
        }
    }
  *n_negatives = nj;
  return newx;
}
```

2. - A [Esame da 9 crediti] Nella directory di lavoro attuale c'è un file di testo che contiene una serie di numeri interi, uno per riga. Scrivere una funzione

     ```
     int * make_int_array(char *fname, int *size)
     ```

     che prende come parametro il nome del file, inserisce gli interi in un array allocato dinamicamente e lo restituisce, usando il parametro output `size` per comunicarne la dimensione al chiamante.

     *Suggerimento* Sarà necessario leggere il file due volte: la prima per contare i numeri e dimensionare correttamente l'array, la seconda per scrivere i numeri nell'array così creato. Per rileggere il file dall'inizio, si può chiuderlo e riaprirlo.

   - B [Esame da 12 crediti] Nella directory di lavoro attuale c'è un file di testo che contiene una serie di parole di lunghezza arbitraria (non limitata), separate da spazi, newline o tab. Scrivere una funzione

     ```
     Nodo * make_word_list(char *fname)
     ```

     che prende come parametro il nome del file, crea una lista di nodi, uno per ogni parola, e restituisce la lista così ottenuta. Definire il tipo di dato `Nodo` in modo opportuno.

     *Suggerimento* Bisogna allocare dinamicamente non solo ogni nodo, ma anche lo spazio per le stringhe puntate: esse possono essere di lunghezza arbitraria.

     *Advanced* Per evitare buffer overflow durante la lettura, si può usare `fscanf()` con l'opzione `%`⟨*lunghezza*⟩`s` nella specifica di conversione. In questo modo, ogni parola sarà troncata e i caratteri letti non supereranno i limiti del buffer. Per esempio, `"%16383s"` troncherebbe ogni parola in modo da rientrare in un buffer di 16 kB ($16383 = 16 \times 1024 - 1$; l'ultimo carattere rimane disponibile per il terminatore `'\0'`).

**Risposta**
Funzione di utilità comune, incorpora `malloc()` e il controllo d'errore.

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

/*
 * malloc(), error check, exit if error
 */
void *xmalloc(size_t nbytes)
{
  void *result;

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

Ecco una possibile soluzione per il quesito A **(esame da 9 crediti).**

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

/*
 * make an array from ints in the named text file
 * return array size in output parameter
 */
int *make_int_array(char *fname, int *size)
{
  int *result;                  // the returned array
  int nints;                    // the size of the returned array
  int thisint;                  // buffer for reading current number
  FILE *fin;
  int i;

  fin = fopen(fname, "r");
  if (fin == NULL)
    {
      fprintf(stderr, "Can't open %s for reading. Bye!\n", fname);
      return NULL;
    }
  nints = 0;                    // let's count the integers in the file
  while (fscanf(fin, "%d", &thisint) == 1)      // while a number is read OK
    {
      nints++;
    }
  result = xmalloc(nints * sizeof(int));
  rewind(fin);                  // back to start of file (same as close/reopen)
  i = 0;
  while (fscanf(fin, "%d", &thisint) == 1)      // while a number is read OK
    {
      result[i++] = thisint;    // number goes into array
    }
  if (i != nints)
    {
      fprintf(stderr, "Warning: expecting %d numbers, but got %d.\n",
              nints, i);
    }
  fclose(fin);
  *size = nints;                // fill return parameter with array size
  return result;
}
```

Ecco una possibile soluzione per il quesito B **(esame da 12 crediti).**

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

typedef struct nodo
{
  char *word;
  struct nodo *next;
} Nodo;


/*
 * Make a new node with a given word
 */
Nodo *make_new_node(char *word)
{
  Nodo *new_node;

  new_node = xmalloc(sizeof(Nodo));
  new_node->word = xmalloc(strlen(word) + 1);
  strcpy(new_node->word, word);
  new_node->next = NULL;
  return new_node;
}

/*
 * make a list from words in the named file
 */
Nodo *make_word_list(char *fname)
{
  Nodo *result;                 // the returned list
  Nodo *new;
  Nodo *last;                   // we append new nodes here
  Nodo dummy;                   // fake list head, makes it easier
  char buf[16 * 1024 + 1];      // input buffer for a word: 16 kB + 1 byte
  FILE *fin;

  fin = fopen(fname, "r");
  if (fin == NULL)
    {
      fprintf(stderr, "Can't open %s for reading. Bye!\n", fname);
      return NULL;
    }
  dummy.word = "not-a-real-word";
  dummy.next = NULL;
  last = &dummy;
  // don't read more than 16 * 1024 bytes with this call to scanf()
  while (fscanf(fin, "%16384s", buf) == 1)      // while a word is read OK
    {
      new = make_new_node(buf);
      last->next = new;         // word goes into list
      last = new;               // next node will be appended after new
    }
  fclose(fin);
  return dummy.next;
}
```

# Programmazione I (Tucci/Distasi)

PR1 MT/RD 22/01/2019

Modello: 2

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. Scrivere una funzione

   ```
   int * negatives_first(int *x, int size, int *n_nonnegatives)
   ```

   che riceva come parametri un array `x[]` di interi e la sua taglia `size`, restituendo un vettore di pari taglia in cui gli elementi negativi sono stati spostati all'inizio. La funzione usa il parametro output `n_nonnegatives` per comunicare il numero di elementi non negativi presenti alla fine dell'array restituito. L'ordine preciso degli elementi non è importante, purché tutti gli elementi negativi precedano tutti i non negativi. L'array restituito deve essere allocato dinamicamente.

2. - A [Esame da 9 crediti] Nella directory di lavoro attuale c'è un file di testo che contiene una serie di numeri interi, uno per riga. Scrivere una funzione

    ```
    int * make_int_array(char *fname, int *size)
    ```

    che prende come parametro il nome del file, inserisce gli interi in un array allocato dinamicamente e lo restituisce, usando il parametro output `size` per comunicarne la dimensione al chiamante.

    *Suggerimento* Sarà necessario leggere il file due volte: la prima per contare i numeri e dimensionare correttamente l'array, la seconda per scrivere i numeri nell'array così creato. Per rileggere il file dall'inizio, si può chiuderlo e riaprirlo.

  - B [Esame da 12 crediti] Nella directory di lavoro attuale c'è un file di testo che contiene una serie di parole di lunghezza arbitraria (non limitata), separate da spazi, newline o tab. Scrivere una funzione

    ```
    Nodo * make_word_list(char *fname)
    ```

    che prende come parametro il nome del file, crea una lista di nodi, uno per ogni parola, e restituisce la lista così ottenuta. Definire il tipo di dato `Nodo` in modo opportuno.

    *Suggerimento* Bisogna allocare dinamicamente non solo ogni nodo, ma anche lo spazio per le stringhe puntate: esse possono essere di lunghezza arbitraria.

    *Advanced* Per evitare buffer overflow durante la lettura, si può usare `fscanf()` con l'opzione `%`⟨*lunghezza*⟩`s` nella specifica di conversione. In questo modo, ogni parola sarà troncata e i caratteri letti non supereranno i limiti del buffer. Per esempio, `"%16383s"` troncherebbe ogni parola in modo da rientrare in un buffer di 16 kB ($16383 = 16 \times 1024 - 1$; l'ultimo carattere rimane disponibile per il terminatore `'\0'`).

# Risposte per il modello 2

1. Scrivere una funzione

   ```
   int * negatives_first(int *x, int size, int *n_nonnegatives)
   ```

   che riceva come parametri un array `x[]` di interi e la sua taglia `size`, restituendo un vettore di pari taglia in cui gli elementi negativi sono stati spostati all'inizio. La funzione usa il parametro output `n_nonnegatives` per comunicare il numero di elementi non negativi presenti alla fine dell'array restituito. L'ordine preciso degli elementi non è importante, purché tutti gli elementi negativi precedano tutti i non negativi. L'array restituito deve essere allocato dinamicamente.

   **Risposta** Ecco una possibile soluzione.

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

/*
 * move negative numbers at the beginning,
 * count the nonnegatives and report in n_nonnegatives
 */
int *negatives_first(int *x, int size, int *n_nonnegatives)
{
  int i;                        // index in source array
  int ni, nj;                   // indices in dest. array, negative/nonnegative
  int *newx;                    // the destination array

  newx = malloc(size * sizeof(int));
  if (newx == NULL)
    {
      fprintf(stderr, "malloc(%ld) failed. Bye!\n", size * sizeof(int));
      exit(-1);
    }
  ni = 0;                       // negative numbers found in x[]
  nj = 0;                       // nonnegative numbers found in x[]
  for (i = 0; i < size; i++)
    {
      if (x[i] < 0)             // negative, goes at the beginning
        {
          newx[ni] = x[i];
          ni++;                 // count one more negative number
      } else
        {
          newx[size - 1 - nj] = x[i];
          nj++;                 // count one more nonnegative number
        }
    }
  *n_nonnegatives = nj;
  return newx;
}
```

2. - A [Esame da 9 crediti] Nella directory di lavoro attuale c'è un file di testo che contiene una serie di numeri interi, uno per riga. Scrivere una funzione

     ```
     int * make_int_array(char *fname, int *size)
     ```

     che prende come parametro il nome del file, inserisce gli interi in un array allocato dinamicamente e lo restituisce, usando il parametro output `size` per comunicarne la dimensione al chiamante.

     *Suggerimento* Sarà necessario leggere il file due volte: la prima per contare i numeri e dimensionare correttamente l'array, la seconda per scrivere i numeri nell'array così creato. Per rileggere il file dall'inizio, si può chiuderlo e riaprirlo.

   - B [Esame da 12 crediti] Nella directory di lavoro attuale c'è un file di testo che contiene una serie di parole di lunghezza arbitraria (non limitata), separate da spazi, newline o tab. Scrivere una funzione

     ```
     Nodo * make_word_list(char *fname)
     ```

     che prende come parametro il nome del file, crea una lista di nodi, uno per ogni parola, e restituisce la lista così ottenuta. Definire il tipo di dato `Nodo` in modo opportuno.

     *Suggerimento* Bisogna allocare dinamicamente non solo ogni nodo, ma anche lo spazio per le stringhe puntate: esse possono essere di lunghezza arbitraria.

     *Advanced* Per evitare buffer overflow durante la lettura, si può usare `fscanf()` con l'opzione `%`⟨*lunghezza*⟩`s` nella specifica di conversione. In questo modo, ogni parola sarà troncata e i caratteri letti non supereranno i limiti del buffer. Per esempio, `"%16383s"` troncherebbe ogni parola in modo da rientrare in un buffer di 16 kB ($16383 = 16 \times 1024 - 1$; l'ultimo carattere rimane disponibile per il terminatore `'\0'`).

**Risposta**
Funzione di utilità comune, incorpora `malloc()` e il controllo d'errore.

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

/*
 * malloc(), error check, exit if error
 */
void *xmalloc(size_t nbytes)
{
  void *result;

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

Ecco una possibile soluzione per il quesito A **(esame da 9 crediti).**

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

/*
 * make an array from ints in the named text file
 * return array size in output parameter
 */
int *make_int_array(char *fname, int *size)
{
  int *result;                  // the returned array
  int nints;                    // the size of the returned array
  int thisint;                  // buffer for reading current number
  FILE *fin;
  int i;

  fin = fopen(fname, "r");
  if (fin == NULL)
    {
      fprintf(stderr, "Can't open %s for reading. Bye!\n", fname);
      return NULL;
    }
  nints = 0;                    // let's count the integers in the file
  while (fscanf(fin, "%d", &thisint) == 1)      // while a number is read OK
    {
      nints++;
    }
  result = xmalloc(nints * sizeof(int));
  rewind(fin);                  // back to start of file (same as close/reopen)
  i = 0;
  while (fscanf(fin, "%d", &thisint) == 1)      // while a number is read OK
    {
      result[i++] = thisint;    // number goes into array
    }
  if (i != nints)
    {
      fprintf(stderr, "Warning: expecting %d numbers, but got %d.\n",
              nints, i);
    }
  fclose(fin);
  *size = nints;                // fill return parameter with array size
  return result;
}
```

Ecco una possibile soluzione per il quesito B **(esame da 12 crediti).**

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

typedef struct nodo
{
  char *word;
  struct nodo *next;
} Nodo;


/*
 * Make a new node with a given word
 */
Nodo *make_new_node(char *word)
{
  Nodo *new_node;

  new_node = xmalloc(sizeof(Nodo));
  new_node->word = xmalloc(strlen(word) + 1);
  strcpy(new_node->word, word);
  new_node->next = NULL;
  return new_node;
}

/*
 * make a list from words in the named file
 */
Nodo *make_word_list(char *fname)
{
  Nodo *result;                 // the returned list
  Nodo *new;
  Nodo *last;                   // we append new nodes here
  Nodo dummy;                   // fake list head, makes it easier
  char buf[16 * 1024 + 1];      // input buffer for a word: 16 kB + 1 byte
  FILE *fin;

  fin = fopen(fname, "r");
  if (fin == NULL)
    {
      fprintf(stderr, "Can't open %s for reading. Bye!\n", fname);
      return NULL;
    }
  dummy.word = "not-a-real-word";
  dummy.next = NULL;
  last = &dummy;
  // don't read more than 16 * 1024 bytes with this call to scanf()
  while (fscanf(fin, "%16384s", buf) == 1)      // while a word is read OK
    {
      new = make_new_node(buf);
      last->next = new;         // word goes into list
      last = new;               // next node will be appended after new
    }
  fclose(fin);
  return dummy.next;
}
```