ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Allocazione Dinamica della Memoria

Un Problema

Il file "pi.txt" contiene un numero imprecisato di cifre di π

- Le cifre sono riportate una per riga

  3

  1

  4

  . . .

- Si desidera realizzare una funzione "leggi_pi"...
- ...Per leggere le cifre e memorizzarle in un vettore di interi

Nel main, vogliamo:

- Usare la funzione per leggere le cifre
- Stamparle a video

Un Problema

Possibilità 1: vettore nel main

```
int leggi_pi(char *nfile, int *v) {
   ...
}
```

numero di elementi letti

vettore da riempire

```
int main() {
   int v[???];
   int n = leggi_pi("pi.txt", v);
}
```

quale dimensione usare?

Un Problema

Possibilità 1: vettore nel main

```
int leggi_pi(char *nfile, int *v) {
   ...
}
```

numero di elementi letti

vettore da riempire

```
int main() {
   int v[???];
   int n = leggi_pi("pi.txt", v);
}
```

quale dimensione usare?

- Non è specificato un numero massimo di elementi!
- Questo approccio non funziona

Un Problema

Possibilità 2: vettore nel record di attivazione di "leggi_pi"

```
int * leggi_pi(char *nfile, int *n) {
   /* apro il file e conto gli elementi */
   *n = /* numero di elementi */
   int v[*n];
   ...
   return v;
}
```

da riempire con il numero di elementi letti

viene restituito il vettore allocato

Un Problema

Possibilità 2: vettore nel record di attivazione di "leggi_pi"

```
int * leggi_pi(char *nfile, int *n) {
   /* apro il file e conto gli elementi */
   *n = /* numero di elementi */
   int v[*n];
   ...
   return v;
}
```

da riempire con il numero di elementi letti

viene restituito il vettore allocato

- L'area di memoria viene deallocata alla fine della funzione
- (e l'allocazione di grandi vettori in stack è sconsigliata)
- Anche questo approccio non funziona

Allocazione Dinamica della Memoria

Cosa ci serve?

- Dobbiamo allocare un'area di memoria (per il vettore)
- La sua dimensione non è nota a priori
- L'area può essere di grandi dimensioni
- L'area deve rimanere allocata alla fine di una funzione

Si può fare sfruttando l'allocazione dinamica della memoria

- Allocazione statica: dimensioni note a tempo di compilazione
- Allocazione dinamica: dimensioni note solo a run-time

Si usano due funzioni dalla libreria "stdlib.h"

Funzione malloc

Per allocare memoria dinamicamente si usa:

```
void * malloc(int num);
```

La funzione "malloc":

- Chiede al SO di allocare un'area di memoria
- La dimensione dell'area è di "num" byte (contigui)
- Restituisce l'indirizzo della prima cella dell'area allocata
- Se l'allocazione fallisce, restituisce NULL

Funzione malloc

Per allocare memoria dinamicamente si usa:

```
void * malloc(int num);
```

La "malloc" alloca memoria nell'area heap:

- È uno dei segmenti di memoria assegnati dal SO al programma
- Può avere grandi dimensioni
- Il suo utilizzo viene in C viene gestito manualmente
  - Occorre richiedere esplicitamente aree di memoria
  - Occorre liberare esplicitamente aree di memoria

Funzione malloc

Per allocare memoria dinamicamente si usa:

```
void * malloc(int num);
```

L'area allocata può essere usata come un array! Infatti:

- È formata da celle contigue
- La "malloc" restituisce un puntatore alla prima cella

Ci sono però due difficoltà:

- La dimensione è in byte (preferimmo ragionare con int, float...)
- La "malloc" restituisce uno strano "puntatore a void"

malloc ed Operatore sizeof

Per specifica la dimensione in termini comprensibili...

...Si usa l'operatore sizeof

```
sizeof(<tipo>)
```

- Applicato ad un tipo, ne restituisce la dimensione in byte
  - Qualche tipo di dato (strutture incluse)
- Esempi:
  - sizeof(int) → dimensione in byte di un intero
  - sizeof(float) → dimensione in byte di un float
- La sintassi ricorda quella di una funzione
  - ...Ma le funzioni si applicano a valori, non a tipi

malloc ed Operatore sizeof

Per allocare memoria per n elementi di un certo tipo:

```
malloc(n * sizeof(<tipo>))
```

- Di fatto, allochiamo un array di "n" elementi
- Qualche esempio:

```
malloc(10 * sizeof(int)); // 10 interi
malloc(1000 * sizeof(float)); // 1000 float

typedef struct {int x, y;} punto;
malloc(10 * sizeof(punto)); // 10 strutture "punto"
```

Puntatori generici

Un "puntatore a void" è semplicemente un puntatore generico

- Tutti i puntatori sono indirizzi ed hanno la stessa dimensione
- Il tipo del puntatore serve solo durante il dereferenziamento
  - I.e. quando scriviamo "*p", occorre sapere il tipo puntato...
  - ...per poter ottenere il valore

Quindi "void*" in:

```
void * malloc(int num);
```

... Vuol dire solo che la malloc restituisce un indirizzo

Puntatori generici... e cast

Per poter utilizzare un puntatore a void occorre un cast

Dobbiamo specificare noi il tipo. Qualche esempio:

```
int *v = (int*) malloc(10 * sizeof(int));
```

- Allocazione dinamica di un vettore di 10 interi

```
float *v = (float*) malloc(1000 * sizeof(float));
```

- Allocazione dinamica di un vettore di 10 interi

```
punto *v = (punto*) malloc(10 * sizeof(punto));
```

- Allocazione dinamica di un vettore di 10 punti

Funzione "free"

L'area di memoria ottenuta va deallocata esplicitamente

Si usa la funzione:

```
void free(void *ptr);
```

- Basta passare il puntatore ottenuto con la "malloc"
- "free" rende l'area di memoria disponibile per altri usi

Se ci si dimentica di eseguire la "free":

- L'area di memoria rimane allocata, anche se non serve più
- Si parla di memory leak
- Se compare in un ciclo, può essere un problema serio

Torniamo al Nostro Prblema

Possiamo allocare il vettore dinamicamente nella funzione

```
int * leggi_pi(char *nfile, int *n) {
    FILE *fp = fopen(nfile, "r");
    if(fp == NULL) {
        fprintf(stderr, "Impossibile aprire %s\n", nfile);
        exit(1);
    }
    *n = 0;
    int v;
    while(fscanf(fp, "%d", &v) == 1) (*n)++;;
    int *res = (int*) malloc((*n) * sizeof(int));
    rewind(fp);
    for(int i = 0; i < *n; ++i)
        fscanf(fp, "%d", &res[i]);
    fclose(fp);
    return res;
}
```

Torniamo al Nostro Prblema

Possiamo allocare il vettore dinamicamente nella funzione

```
int * leggi_pi(char *nfile, int *n) {
    FILE *fp = fopen(nfile, "r");
    if(fp == NULL) {
        fprintf(stderr, "Impossibile aprire %s\n", nfile);
        exit(1);
    }
    *n = 0;
    int v;
    while(fscanf(fp, "%d", &v) == 1) (*n)++;
    int *res = (int*) malloc((*n) * sizeof(int));
    rewind(fp);
    for(int i = 0; i < *n; ++i)
        fscanf(fp, "%d", &res[i]);
    fclose(fp);
    return res;
}
```

apriamo e chiudiamo il file normalmente

Torniamo al Nostro Prblema

Possiamo allocare il vettore dinamicamente nella funzione

```
int * leggi_pi(char *nfile, int *n) {
    FILE *fp = fopen(nfile, "r");
    if(fp == NULL) {
        fprintf(stderr, "Impossibile aprire %s\n", nfile);
        exit(1);
    }
    *n = 0;
    int v;
    while(fscanf(fp, "%d", &v) == 1) (*n)++;
    int *res = (int*) malloc((*n) * sizeof(int));
    rewind(fp);
    for(int i = 0; i < *n; ++i)
        fscanf(fp, "%d", &res[i]);
    fclose(fp);
    return res;
}
```

Lettura del file, al solo scopo di contarne gli elementi

Torniamo al Nostro Prblema

Possiamo allocare il vettore dinamicamente nella funzione

```
int * leggi_pi(char *nfile, int *n) {
    FILE *fp = fopen(nfile, "r");
    if(fp == NULL) {
        fprintf(stderr, "Impossibile aprire %s\n", nfile);
        exit(1);
    }
    *n = 0;
    int v;
    while(fscanf(fp, "%d", &v) == 1) (*n)++;
    int *res = (int*) malloc((*n) * sizeof(int));
    rewind(fp);
    for(int i = 0; i < *n; ++i)
        fscanf(fp, "%d", &res[i]);
    fclose(fp);
    return res;
}
```

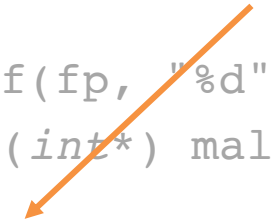
Adesso che la dimensione è nota, possiamo allocare dinamicamente un vettore nell'area heap

Torniamo al Nostro Prblema

Possiamo allocare il vettore dinamicamente nella funzione

```
int * leggi_pi(char *nfile, int *n) {
    FILE *fp = fopen(nfile, "r");
    if(fp == NULL) {
        fprintf(stderr, "Impossibile aprire %s\n", nfile);
        exit(1);
    }
    *n = 0;
    int v;
    while(fscanf(fp, "%d", &v) == 1) (*n)++;;
    int *res = (int*) malloc((*n) * sizeof(int));
    rewind(fp);
    for(int i = 0; i < *n; ++i)
        fscanf(fp, "%d", &res[i]);
    fclose(fp);
    return res;
}
```

Riportiamo il cursore all'inizio del file



Torniamo al Nostro Prblema

Possiamo allocare il vettore dinamicamente nella funzione

```
int * leggi_pi(char *nfile, int *n) {
    FILE *fp = fopen(nfile, "r");
    if(fp == NULL) {
        fprintf(stderr, "Impossibile aprire %s\n", nfile);
        exit(1);
    }
    *n = 0;
    int v;
    while(fscanf(fp, "%d", &v) == 1) (*n)++;;
    int *res = (int*) malloc((*n) * sizeof(int));
    rewind(fp);
    for(int i = 0; i < *n; ++i)
        fscanf(fp, "%d", &res[i]);
    fclose(fp);
    return res;
}
```

Seconda lettura (questa volta nel vettore)

Torniamo al Nostro Prblema

Possiamo allocare il vettore dinamicamente nella funzione

```
int * leggi_pi(char *nfile, int *n) {
    FILE *fp = fopen(nfile, "r");
    if(fp == NULL) {
        fprintf(stderr, "Impossibile aprire %s\n", nfile);
        exit(1);
    }
    *n = 0;
    int v;
    while(fscanf(fp, "%d", &v) == 1) (*n)++;;
    int *res = (int*) malloc((*n) * sizeof(int));
    rewind(fp);
    for(int i = 0; i < *n; ++i)
        fscanf(fp, "%d", &res[i]);
    fclose(fp);
    return res;
}
```

Restituiamo il puntatore fornito da "malloc"

Torniamo al Nostro Prblema

Nel main dobbiamo ricordarci di eseguire "free"

```
int main() {
    /* leggo le prime cifre di pi greco */
    int n;
    int *v = leggi_pi("pi.txt", &n);

    /* stampo */
    printf("%d.", v[0]);
    for(int i = 1; i < n; ++i)
        printf("%d", v[i]);
    printf("\n");

    /* dealloco il vettore */
    free(v);
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Lettura di Voti

Lettura di Voti

Il file "voti.txt" contiene i risultati di una prova d'esame

Ogni riga del file è nella forma

```
matricola voto
```

- Non è noto a priori il numero massimo di voti nel file
- "matricola" è una stringa senza spazi di 10 caratteri
- Il voto è un valore in 30mi, con alcune convenzioni speciali:
  - 0: respinto
  - -1: ritirato/assente
  - 31: 30 e lode

Nel file "voti.h" si definisca una struttura dati "voto" atta a contenere le informazioni su un risultato

Lettura di Voti

Nei file "voti.h/voti.c" si definisca la funzione:

```
voto * leggi_voti(char *nfile, int *n);
```

Che:

- Legga da un file di testo di nome "nfile" le informazioni sui voti
- Le scriva in un vettore allocato dinamicamente
- Restituisca un puntatore all'area di memoria allocata
- Restituisca in "n" il numero di elementi

Lettura di Voti

Nel file "main.c" si definisca un main di collaudo:

- Il main deve usare la funzione "leggi_voti"
- Quindi stampare le matricole ed il voto
- Per i valori speciali, occorre stampare una stringa (e.g. "respinto, 30L"), anziché il valore numerico

Il codice della soluzione è disponibile in un archivio zip

Soluzione (anche su archivio zip)

Nel file "voti.h":

```
#ifndef VOTI_H_
#define VOTI_H_

#include <stdio.h>
#include <stdlib.h>
#define NMAT 11

typedef struct {
    char matricola[NMAT];
    int voto;
} voto;

voto * leggi_voti(char *nfile, int *n);

#endif
```

Soluzione (anche su archivio zip)

Nel file "voti.c":

```c
#include "voti.h"
voto * leggi_voti(char *nfile, int *n) {
    /* apro il file */
    FILE *fp = fopen(nfile, "r");
    if(fp == NULL) {
        fprintf(stderr, "Impossibile aprire %s\n", nfile);
        exit(1);
    }
    /* conto il numero di elementi da leggere */
    *n = 0;
    voto t;
    while(fscanf(fp, "%s%d", t.matricola, &t.voto) == 2) (*n)++;;
    /* leggo i dati */
    voto *res = (voto*) malloc((*n) * sizeof(voto));
    rewind(fp);
    for(int i = 0; i < *n; ++i)
        fscanf(fp, "%s%d", res[i].matricola, &res[i].voto);
    /* chiudo il file */
    fclose(fp);
    return res;
}
```

Soluzione (anche su archivio zip)

Nel file "main.c":

```c
#include "voti.h"
int main() {
    int n;
    voto *v = leggi_voti("voti.txt", &n);
    for(int i = 0; i < n; ++i) {
        printf("%s ", v[i].matricola);
        switch(v[i].voto) {
            case 0:
                printf("respinto\n");  break;
            case -1:
                printf("assente/ritirato\n"); break;
            case 31:
                printf("30L\n"); break;
            default:
                printf("%d\n", v[i].voto);
        }
    }
    free(v);
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Lettura di Persone

Lettura di Persone

Il file "persone.txt" contiene nomi e cognomi di persone

- La prima riga riporta il numero di coppie nome/cognome
- Ogni riga successiva è nella forma

  ```
  nome cognome
  ```

- "nome" e "cognome" sono stringhe senza spazi di al più 255 caratteri

Nel file "persone.h" si definisca una struttura dati "persona" atta a contenere le informazioni su una persona

Lettura di Persone

Nei file "persone.h/persone.c" si definisca la funzione:

```
persona * leggi_persone(char *nfile, int *n);
```

Che:

- Legga da un file di testo di nome "nfile" le informazioni sulle persone
- Le scriva in un vettore allocato dinamicamente
- Restituisca un puntatore all'area di memoria allocata
- Restituisca in "n" il numero di elementi

Lettura di Persone

Nel file "main.c" si definisca un main di collaudo:

- Il main deve usare la funzione "leggi_persone"
- Quindi stampare i nomi e cognomi letti

Il codice della soluzione è disponibile in un archivio zip

Soluzione (anche su archivio zip)

Nel file "persone.h":

```
#ifndef PERSONE_H_
#define PERSONE_H_

#include <stdio.h>
#include <stdlib.h>
#define NNOME 256
#define NCOGNOME 256

typedef struct {
    char nome[NNOME];
    char cognome[NCOGNOME];
} persona;

persona * leggi_persone(char *nfile, int *n);

#endif
```

Soluzione (anche su archivio zip)

Nel file "persone.c":

```
#include "persone.h"
persona * leggi_persone(char *nfile, int *n) {
    /* apro il file */
    FILE *fp = fopen(nfile, "r");
    if(fp == NULL) {
        fprintf(stderr, "Impossibile aprire %s\n", nfile);
        exit(1);
    }
    /* leggo il numero di elementi */
    if (fscanf(fp, "%d", n) != 1) {
        fprintf(stderr, "Impossibile leggere il numero di elementi\n");
        exit(1);
    }
    /* leggo i dati */
    persona *res = (persona*) malloc((*n) * sizeof(persona));
    for(int i = 0; i < *n; ++i)
        fscanf(fp, "%s%s", res[i].nome, res[i].cognome);
    /* restituisco un puntatore all'array */
    fclose(fp);
    return res;
}
```

Soluzione (anche su archivio zip)

Nel file "main.c":

```c
#include "persone.h"

int main() {
    int n;
    persona *v = leggi_persone("persone.txt", &n);

    for(int i = 0; i < n; ++i)
        printf("%s %s\n", v[i].nome, v[i].cognome);

    free(v);
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Andrea Acquaviva < andrea.acquaviva@unibo.it>
Michele Lombardi <michele.lombardi2@unibo.it
Andrea Borghesi <andrea.borghesi3@unibo.it>
Giuseppe Tagliavini <giuseppe.tagliavini@unibo.it>
Allegra De Filippo < allegra.defilippo@unibo.it>