Fondamenti di Informatica e Laboratorio T-AB
Ingegneria Elettronica e Telecomunicazioni

Lab 11
Allocazione dinamica della memoria

STACK

- ***L'area di memoria stack é quella in cui viene allocato un pacchetto di dati non appena l'esecuzione passa dal programma chiamante a una funzione.***
- ***Questo pacchetto (il quale contiene l'indirizzo di rientro nel programma chiamante, la lista degli argomenti passati alla funzione e tutte le variabili definite nella funzione) viene "impilato" sopra il pacchetto precedente (quello del programma chiamante) e poi automaticamente rimosso dalla memoria appena l'esecuzione della funzione é terminata.***
- ***Le variabili automatiche definite nella funzione hanno lifetime limitato all'esecuzione della funzione stessa proprio perché, quando la funzione termina, il corrispondente pacchetto allocato nell'area stack viene rimosso.***

HEAP

- ***Esiste un'altra area di memoria che il programma può utilizzare. Questa area, detta heap, è soggetta alle seguenti regole:***
- ***non é allocata automaticamente, ma può essere allocata o rimossa solo su esplicita richiesta del programma (allocazione dinamica della memoria);***
- ***l'area allocata non é identificata da un nome, ma é accessibile esclusivamente tramite dereferenziazione di un puntatore;***
- ***il suo scope coincide con quello del puntatore che contiene il suo indirizzo;***
- ***La sua lifetime coincide con l'intera durata del programma, a meno che non venga esplicitamente deallocata; se il puntatore va out of scope, l'area non é più accessibile, ma continua a occupare memoria inutilmente: si verifica l'errore di memory leak.***

Esercizio 0

Un file di testo di nome "`bambini.txt`", contiene informazioni sugli iscritti ad una scuola materna. In particolare, per ogni bambino si memorizza il nome (al più 30 caratteri, senza spazi) e l'età. Il numero di iscritti non è noto a priori.

Si definisca una struttura "Bambino" per memorizzare la informazioni su un singolo iscritto alla scuola.

Esercizio 0

Si definisca poi una funzione:

Bambino* leggi(FILE* bFile, int* n);

Che legge dal file “bFile” le informazioni su una serie di bambini iscritti alla scuola. La funzione deve restituire l’indirizzo della prima cella di un vettore in cui saranno state salvate tali informazioni. Si noti che tale vettore dovrà essere allocato dinamicamente. La funzione dovrà anche inserire in “n” il numero di bambini iscritti alla scuola.

Esercizio 0

Si realizzi poi un programma che legga dal file "bambini.txt" le informazioni sugli iscritti alla scuola utilizzando la funzione "leggi".

Il programma stampi poi a video i nomi dei bambini che hanno raggiunto i 5 anni.

Esercizio 0 – bambini.h

```
#ifndef bambini_h_
#define bambini_h_

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

typedef struct {
    char nome[31];
    int anni;
} Bambino;

Bambino* leggi(FILE* bFile, int* n);

#endif
```

Esercizio 0 – bambini.c

```
#include "bambini.h"

Bambino* leggi(FILE* bFile, int* n)
{
    Bambino* res = NULL;
    Bambino temp;
    int i;

    *n = 0;
    while(fscanf(bFile, "%s%d", temp.nome, &temp.anni) == 2)
       (*n)++;

    res = (Bambino*) malloc((*n) * sizeof(Bambino));
    ...
```

Esercizio 0 – bambini.c

```
    ...

    rewind(bFile);
    for (i = 0; i < *n; i++) {
       fscanf(bFile, "%s%d", temp.nome, &temp.anni);
       res[i] = temp;
    }

    return res;
}
```

Esercizio 0 – main0.c

```
#include "bambini.h"

int main(void) {
    FILE* input_file;
    Bambino* iscritti = NULL;
    int i, n;

    if ((input_file = fopen("bambini.txt", "r")) == NULL) {
       printf("Impossibile aprire il file di input\n");
       exit(1);
    }

    iscritti = leggi(input_file, &n);
    fclose(input_file);
    ...
```

Esercizio 0 – main0.c

```
    ...
    for (i = 0; i < n; i++)
       if (iscritti[i].anni == 5)
             printf("%s\n", iscritti[i].nome);

    free(iscritti);

    return 0;
}
```

Esercizio 1

È dato un file di testo, di nome "`valori.txt`", contenente una sequenza di `int`; non è noto a priori quanti interi siano presenti nel file o in ciascuna linea del file. I valori sono disposti in ordine casuale.

Si realizzi un programma che, letti dal file tali valori interi, li stampi a video ponendo prima i numeri pari e poi i numeri dispari

A tal scopo si definisca:

Esercizio 1

1. Una funzione

```
int readLength(FILE *f, int *even, int *odd)
```

che determini quanti valori sono presenti nel file

In particolare, la funzione deve restituire il numero totale di valori presenti nel file, e tramite i parametri `even` e `odd` deve restituire il numero di valori pari e di valori dispari rispettivamente (la somma di `even` + `odd` deve ovviamente essere uguale al numero totale di valori presenti nel file)

Esercizio 1

2. Un programma **`main`** che, aperto opportunamente il file "**`valori.txt`**", determini quanti valori sono presenti sul file tramite la funzione **`readLength(…)`**

Il programma deve allocare dinamicamente memoria sufficiente per leggere tutti i valori, e deve poi procedere a leggere i valori dal file e a disporli nel vettore allocato, prima i pari e poi i dispari.

Ad esempio, se nel file ci sono 13 valori pari e 16 valori dispari, nelle prime 13 posizioni del vettore ci dovranno essere i valori pari, e nelle seguenti 16 i valori dispari

Si ricorda l'esistenza della procedura di libreria **`void rewind(FILE *f)`** che riporta la testina di lettura a inizio file. Il programma stampi infine a video tale vettore

Esercizio 2

Una società di telefonia cellulare gestisce un programma di premiazione per “utenti fedeli”. In particolare, per ogni cliente viene salvato su un file di testo “**punti.txt**” il nome del cliente (al massimo 31 caratteri) e un numero intero che rappresenta i punti accumulati. Tali informazioni potranno essere memorizzate in una struttura **user**:

```
#define DIM 32

typedef struct {
   char name[DIM];
   int points;
} user;
```

Esercizio 2

1) Si scriva una funzione:

int readPoints (char usersFile[], user results[], int maxDim, int minPoints)

che, ricevuto in ingresso il nome di un file **usersFile**, un array **results** di strutture **user**, la dimensione massima dell'array **maxDim**, e un limite inferiore di punti **minPoints**, copi nell'array **results** i dati dei clienti che hanno almeno i punti specificati da **minPoints**

La funzione deve restituire come risultato il numero di utenti con almeno **minPoints**; si noti che tale risultato rappresenta anche la dimensione logica dell'array **results**. Qualora il file non sia accessibile, la funzione deve restituire il valore -1

Esercizio 2

2) Si scriva poi un programma **main()** che chieda all'utente un numero massimo di clienti da esaminare ed allochi dinamicamente un vettore **V** di **user** sufficientemente grande per poter contenere i dati di tutti gli utenti salvati. Il programma dovrà poi chiedere all'utente un punteggio minimo e, utilizzando la funzione **readPoints()**, leggere da file e memorizzare in **V** i dati degli utenti che hanno almeno il punteggio minimo specificato. Il programma infine deve stampare a video il nome ed il punteggio degli utenti contenuti in **V** se e solo se il nome comincia per "Me"

Il file contiene una quantità indefinita di informazioni:
non è possibile contenerle tutte in un array di dimensione fissata a priori
→ *malloc*

Esercizio 3

Un negozio di noleggio CD registra, tramite un PC collegato al registratore di cassa, i dati relativi al noleggio dei Compact Disc. Per ogni utente che restituisce un disco, su un file di testo di nome "**RentedLog.txt**" viene scritto su ogni riga, in ordine:

- un intero **cd_code**, identificativo univoco di un cd
- una stringa, contenente il nome del cliente (al più 64 caratteri, senza spazi)
- un intero **days**, che indica la durata in giorni del noleggio

Dopo aver definito opportunamente una struttura **rent** per contenere tali informazioni, il candidato realizzi un programma che chieda all'utente il **nome di un cliente e il numero massimo di record che si vogliono ottenere**, e stampi a video la lista dei CD noleggiati dal cliente, subito seguito dalla durata media di un noleggio per tale cliente

```
#define DIM 65
typedef struct {
 int cd_code;
 char renter[DIM];
 int days;
} rent;
```

Esercizio 3

1) Il candidato scriva una funzione **readRented(…)** che riceve in ingresso il nome di un file di testo, il nome di un utente, un puntatore a strutture **rent** (che punta ad un'area di memoria opportunamente allocata in precedenza) e la dimensione massima di tale area di memoria (in termini di numero di strutture di tipo **rent**). La funzione apra il file e salvi in memoria (tramite il puntatore ricevuto come parametro) i **record relativi all'utente specificato** (per controllare se un record è relativo al cliente specificato, si utilizzi la funzione **strcmp(…)**). La funzione restituisca il **numero di record effettivamente letti**, che deve risultare minore o uguale alla dimensione massima specificata. Qualora si raggiunga la dimensione massima di record letti prima di aver terminato il file, si ignorino i record rimanenti

Esercizio 3

2) Il candidato realizzi poi un programma C che chieda inizialmente all'utente il nome di un cliente e il numero massimo di elementi su cui si vuole effettuare la statistica. Dopo aver allocato dinamicamente memoria sufficiente secondo le istruzioni ricevute dall'utente, il programma utilizzi la funzione **readRented(…)** per ottenere i dati relativi al determinato cliente. **Si stampi a video poi, in ordine, per ogni CD noleggiato, il nome del cliente, il codice del CD e la durata del noleggio**. Si stampi infine la durata media del noleggio

Esercizio 4

Un negoziante tiene traccia del prezzo degli articoli in vendita e dell'elenco degli articoli già venduti in due file di testo distinti. In particolare,

- il file **listino.txt** specifica in ogni riga, separati tra loro da uno spazio, la tipologia di articolo in vendita (al più dieci caratteri senza spazi), la sua marca (al più 10 caratteri senza spazi) e il suo prezzo in euro (float).
- Il file **venduti.txt** elenca gli articoli già venduti, con una riga con tipologia e marca per ogni articolo venduto.

listino.txt
acqua fiuggi 7.0 acqua recoaro 6.0 pasta barilla 0.3 pasta dececco 0.5

venduti.txt
acqua recoaro acqua recoaro pasta barilla pasta barilla acqua recoaro pasta dececco

Esercizio 4

- Ovviamente possono esserci più occorrenze di uno stesso articolo in **venduti.txt** e
- non è detto che ogni articolo presente in listino.txt sia presente anche in **venduti.txt**
- invece se un articolo è presente in **venduti.txt** allora è sicuramente presente anche in **listino.txt**

Dopo aver realizzato una struttura dati item in cui sia possibile specificare la tipologia di un articolo, la sua marca, il prezzo in euro e la quantità di articoli già venduti tramite un intero, il candidato realizzi una funzione:

```
item* articoli(FILE* listino,FILE* venduti,
               char* tipologia, int* len)
```

Esercizio 4

Tale funzione:

- ricevuti in ingresso due puntatori a file ed una tipologia di articolo (ad esempio **`pasta`**)
- legga il file **`listino`** per calcolare quanti articoli sono presenti del tipo **`tipologia`** e sfrutti tale valore per allocare dinamicamente memoria sufficiente per contenere tutti gli articoli di quel tipo presenti nel file **listino.txt**
- Per ogni marca in vendita della tipologia di articolo richiesta, la funzione inserisca nello spazio di memoria allocata dinamicamente un **`item`**
  - Per ogni **`item`** specificare, oltre a tipologia, marca e prezzo, anche il numero di articoli venduti (ovvero il numero di occorrenze in **`venduti`**).

Esercizio 4

La funzione **`articoli(...)`** restituisca alla funzione chiamante un puntatore all'area di memoria che contiene gli **`item`** e il numero di elementi restituiti tramite **`len`**

Si ricorda l'esistenza della funzione **`void rewind(*FILE)`** che riporta la testina di lettura a inizio file e della funzione **`int strcmp(char* st, char * ct)`** per il confronto tra stringhe

Infine si scriva un main di esempio dove viene invocata la funzione articoli(...)