Università degli Studi di Napoli Federico II

Corso di Laurea in Informatica

Laboratorio di Programmazione (Gr.3)

Compito del 20/06/2022

2 ore **Dott. Andrea Apicella**

REGOLE

- *Della libreria standard del C, è accettato l'utilizzo **solo** dei seguenti file header:*
  - *stdio.h*
  - *stdlib.h*
  - *math.h*
  - *string.h*
  - *time.h*

  *L'utilizzo di qualsiasi altro file header (e delle relative funzioni) della libreria standard **non sarà accettato**;*
- *Seguire scrupolosamente le direttive nella traccia ed utilizzare eventuali nomi di variabili e/o funzioni dati senza variazioni (anche in termini di maiuscole/minuscole e caratteri _);*
- *Ogni file sorgente dovrà contenere nelle prime righe un commento nel formato:*

```
/* MATRICOLA: ...
COGNOME: ...
NOME: ... */
```

- *E' fortemente consigliato commentare il codice il più possibile;*
- *E' fortemente consigliato modulare il progetto su più file, nello specifico almeno 3:*
  1. *contenente i prototipi e le definizioni di eventuali strutture;*
  2. *contenente la definizione della funzione main(). Nel caso siano richiesti più main(), fare un file diverso per ognuno di essi;*
  3. *contenente le definizioni delle funzioni rimanenti.*

  Se lo si ritiene opportuno, è possibile separare i sorgenti in più file, motivandolo opportunamente con dei commenti;
- ***Non** includere il file eseguibile ed eventuali file oggetto nella consegna.*
- *La consegna finale dovrà quindi contenere (almeno) 4 file:*
  - *i file sorgenti (almeno 3, come specificato sopra);*
  - *il file 'istruzioni.txt', contenente i comandi per la compilazione/linking come richiesti nella traccia.*

TRACCIA DEL 20/06/2022

TIC TAC TOE (meglio conosciuto in Italia come *Tris*) è un gioco a due giocatori che prevede l'inserimento di un simbolo (di solito una X o un O, in base al giocatore) in una griglia 3×3. Vince il giocatore che, per primo, riesce a disporre una sequenza di tre ripetizioni consecutive del suo simbolo lungo una riga, o lungo una colonna, o lungo una diagonale della griglia di gioco. Nel caso in cui la griglia venga riempita totalmente senza ottenere una configurazione di vittoria, la partita finirà in pareggio. Esempi:

configurazioni di vittoria per il giocatore X:

$$\begin{bmatrix} X & O & O \\ X & O & X \\ X & X & O \end{bmatrix}, \begin{bmatrix} X & O & O \\ O & X & \\ & & X \end{bmatrix}, \begin{bmatrix} O & O & X \\ O & O & \\ X & X & X \end{bmatrix}$$

configurazioni di vittoria per il giocatore O :

$$\begin{bmatrix} X & O & O \\ X & O & X \\ O & X & O \end{bmatrix}, \begin{bmatrix} O & O & O \\ X & O & X \\ & & O \end{bmatrix}, \begin{bmatrix} O & O & \\ X & O & X \\ & O & \end{bmatrix}$$

configurazione di pareggio:

$$\begin{bmatrix} X & O & O \\ O & O & X \\ X & X & O \end{bmatrix}, \begin{bmatrix} X & O & O \\ O & X & X \\ O & X & O \end{bmatrix}, \begin{bmatrix} O & X & O \\ O & X & X \\ X & O & X \end{bmatrix}$$

Data una configurazione

$$C = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix},$$

è possibile rappresentare quest'ultima attraverso un'unica sequenza di 9 elementi

$$c_{11}c_{12}c_{13}c_{21}c_{22}c_{23}c_{31}c_{32}c_{33}$$

con $c_{ij} = \begin{cases} X & \textit{se l'elemento in posizione i,j è una } X \\ O & \textit{se l'elemento in posizione i,j è una } O \\ @ & \textit{se la posizione i,j è vuota.} \end{cases}$

Esempio: la sequenza $X@OOOXXXO$ corrisponde alla configurazione:

$$\begin{bmatrix} X & & O \\ O & O & X \\ X & X & O \end{bmatrix}$$

Un file di testo 'configurazioni.txt' contiene, su ogni riga, una possibile configurazione del campo da gioco.

Esempio:

'configurazioni.txt':

```
XOOX@XXXO
XOOXOXOXO
XOOOOXX@O
```

contiene le configurazioni:

$$\begin{bmatrix} X & O & O \\ X & & X \\ X & X & O \end{bmatrix}, \begin{bmatrix} X & O & O \\ X & O & X \\ O & X & O \end{bmatrix}, \begin{bmatrix} X & O & O \\ O & O & X \\ X & & O \end{bmatrix}.$$

Si chiede di scrivere un programma in linguaggio C che effettui le seguenti operazioni:

Punto 1: caricare il contenuto del filein una lista semplicemente concatenata. La funzione di caricamento dovrà chiamarsi *load_matches_from_file(...)*. Ogni nodo della lista dovrà contenere:

- la configurazione della partita, disposta in una matrice *campo* di 3×3 caratteri;
- una variabile *vincitore* di tipo 'char', che dovrà contenere uno dei seguenti valori:
  - 'X', se la configurazione è di vittoria per il giocatore 'X';
  - 'O', se la configurazione è di vittoria per il giocatore 'O';
  - 'P', se la configurazione è di pareggio.

  il vincitore dovrà essere determinato dall'invocazione di un'apposita funzione *the_winner_is(...)* avente come unico parametro un campo da gioco.
- un variabile *punteggio*, di tipo 'int', che dovrà contenere il punteggio della partita calcolato nel seguente modo:

$$\text{punteggio} = \begin{cases} 0 & \text{se la partita è un pareggio} \\ 3 + \text{numero posizioni libere} & \text{altrimenti.} \end{cases}$$

il punteggio dovrà essere calcolato da una apposita funzione *match_score(...)* che dovrà restituire un 'int'. Il numero di posizioni libere dovrà essere calcolato da una seconda funzione *free_positions(...)*. Quest'ultima dovrà implementare una soluzione *preferibilmente ricorsiva*;

Punto 2: stampare a video la lista attraverso un'apposita funzione *print_list(...)* *ricorsiva*. Tale funzione dovrà invocare al suo interno una funzione *print_match(...)* dedicata alla stampa dei dettagli di una singola partita, ossia:

- il vincitore X o O (o l'eventuale pareggio);
- il punteggio fatto;
- il campo da gioco.

Punto 3: rimuovere dalla lista tutte le configurazioni di pareggio attraverso una apposita funzione *remove_tied(...)*. Stampare a video la lista degli elementi residui;

Punto 4: salvare in due file di testo differenti 'vittorie_X.txt' e 'vittorie_O.txt' rispettivamente le configurazioni di vittoria di 'X' e di 'O' ancora presenti nella lista, nello stesso formato del file di input. Il salvataggio dovrà essere effettuato attraverso la *doppia* invocazione di un'*unica* funzione *write_winners_on_file(...)* avente come parametri:

- il nome del file;
- la lista;
- un carattere che indica il giocatore di cui si desiderano salvare le configurazioni vincenti.

la funzione dovrà essere *non* distruttiva della struttura dati;

Punto 5: compilare ed eseguire il programma da linea di comando. Riportare su di un file di testo di nome 'istruzioni.txt' i comandi necessari per effettuare la compilazione con *gcc* dei *singoli* moduli ed il linking. Compilazione e linking dovranno essere effettuati *separatamente*. Il file eseguibile dovrà avere nome **prog**_numerodimatricola**.eseguibile**.

Università degli Studi di Napoli Federico II

Corso di Laurea in Informatica

Laboratorio di Programmazione (Gr.3)

Compito del 19/10/2022

Dott. Andrea Apicella

REGOLE

- *Della libreria standard del C, è accettato l'utilizzo **solo** dei seguenti file header:*
  - *stdio.h*
  - *stdlib.h*
  - *math.h*
  - *string.h*
  - *time.h*

  *L'utilizzo di qualsiasi altro file header e relative funzioni della libreria standard **non sarà accettato**;*
- *Seguire scrupolosamente le direttive nella traccia ed utilizzare eventuali nomi di variabili e/o funzioni dati senza variazioni (anche in termini di maiuscole/minuscole e caratteri _);*
- *Ogni file sorgente dovrà contenere nelle prime righe un commento nel formato:*

```
/* MATRICOLA: ...
COGNOME: ...
NOME: ... */
```

- *E' fortemente consigliato commentare il codice il più possibile;*
- *E' fortemente consigliato modulare il progetto su più file, nello specifico almeno 3:*
  1. *contenente i prototipi e le definizioni di eventuali strutture;*
  2. *contenente la definizione della funzione main(). Nel caso siano richiesti più main(), fare un file diverso per ognuno di essi;*
  3. *contenente le definizioni delle funzioni rimanenti.*

  Se lo si ritiene opportuno, è possibile separare i sorgenti in più di 3 file, motivandolo con dei commenti;
- ***NON** includere il file eseguibile ed eventuali file oggetto nella consegna.*
- *La consegna finale dovrà quindi essere un unico archivio zip contenente (almeno) 4 file:*
  - *i file sorgenti (almeno 3, come specificato sopra);*
  - *il file 'istruzioni.txt', contenente i comandi per la compilazione/linking come richiesti nella traccia.*

  Verificare che tutti i file siano stati correttamente compressi e caricati!

TRACCIA DEL 19/10/2022, TEMPO: 2 ore

Il nascente social network *Fakebook* contiene i *Post* degli utenti in specifiche strutture dati così formate:

- *msg*: messaggio di testo di al più 256 caratteri; per semplicità, si supponga che tale campo non contenga caratteri di a capo al suo interno;
- *n_like*: numero di utenti che apprezzano il post;

Per un singolo utente, le informazioni sono memorizzate in un file di testo nel seguente formato:

```
msg1
n_like1
msg2
n_like2
.
.
.
```

Esempio:

```
01/01/2010: buongiornissimo!1!1!11!!kaffè??
144
11/01/2010: non ho superato l'esame! :(
5
10/02/2010: neanche questa volta ho passato l'esame! >:(
2
16/04/2010: è la terza volta che rifaccio quest'esame!!! >:@
1
20/05/2024: Ho finalmente superato l'esame! :D
14
```

Punto 1:: implementare una struttura dati *Bacheca* con politica di accesso *coda* (*First In First Out*) in grado di memorizzare dei *Post*. Tale struttura dovrà fornire almeno le seguenti funzioni:

- *append(Bacheca, Post)*: inserisce il nuovo post all'interno della bacheca, rispettando la politica di accesso;
- *pop(Bacheca)*: restituisce il collegamento (e rimuove dalla bacheca) il prossimo post, rispettando la politica di accesso;
- *len(Bacheca)*: restituisce il numero di elementi contenuti in bacheca;
- *is_empty(Bacheca)*: restituisce 1 se la bacheca è vuota, 0 altrimenti.

Tale struttura dati dovrà essere implementata attraverso un array di puntatori a Post allocato dinamicamente.

Punto 2: scrivere una funzione *load_user_from_file(Bacheca,nomefile)* che carica tutti i dati contenuti nel file di testo di nome $nomefile$ all'interno della *Bacheca*. Il file di testo è fornito con la traccia, ed ha nome "utente42.txt". L'accesso alla bacheca deve essere effettuato sfruttando soltanto le funzioni definite al punto 1.

Punto 3: nella funzione *main()*, stampare tutti gli elementi della bacheca uno per volta, dando la possibilità all'utente di scegliere se:

- fermarsi;
- andare al prossimo post;
- aggiungere un like al post attuale;
- ”condividere” il post attuale. In questo contesto, ”condividere” il post equivale a salvarlo in coda in un file di testo di nome ”condivisi.txt”.

L'accesso alla bacheca deve essere effettuato sfruttando soltanto le funzioni definite al punto 1. I post dovranno essere ancora presenti all'interno della struttura dati Bacheca al termine dello scorrimento.

E' consigliato implementare una funzione *print(Post)* che stampi il contenuto di *un singolo* Post passato come argomento.

Punto 4: compilare ed eseguire il programma da linea di comando tramite *gcc*. Riportare su di un file di testo di nome 'istruzioni.txt' i comandi necessari per effettuare la generazione del file eseguibile. Le compilazioni dei file contenenti le funzioni diverse dal *main()* dovranno essere effettuate *distintamente*. La compilazione del file contenente il *main()* dovrà essere effettuata assieme al linking con tutti i rimanenti file oggetto. Il file eseguibile dovrà avere nome **utente**_numerodimatricola**.eseguibile**. Consegnare soltanto i file sorgenti, escludendo i file oggetto ed esegubili dalla consegna.

Laboratorio di Programmazione Gr. 3 (N-Z)

Corso di Laurea in Informatica

Università degli Studi di Napoli Federico II

A.A. 2021/22

A. Apicella

In [1]:
```
/* Dati due insiemi A e B, si definisce *prodotto cartesiano*
 l'insieme delle coppie ordinate A \times B = {(a,b) | a in A and b in B}.
rappresentando i due insiemi A e B attraverso due array,
implementare una funzione che restituisca
il prodotto cartesiano A x B all'interno di una matrice,
i.e. ogni riga rappresenta un elemento del prodotto cartesiano.

*/

#include <stdio.h>

#define MAX_ROWS 10
#define MAX_COLS 10



void stampa_vett(int v[],int n)
{
    printf("(");
    for(int i = 0; i < n; i++)
    {
        printf(" %d, ", v[i]);

    }
    printf("\b\b)\n");
}
void stampa_mat(int mat[][MAX_COLS], int n_r, int n_c)
{
    printf("(\n");
    for(int i = 0; i < n_r; i++)
    {
        for(int j = 0; j < n_c; j++)
            printf(" %d, ", mat[i][j]);
        printf("\b\b\n");
    }
    printf(")\n");
}

void cartesian_product(int A[], int n_A,
                       int B[], int n_B,
                       int PROD[][MAX_COLS])
{
    int idx_next = 0;
    for(int i = 0; i < n_A; i++)
    {
        for (int j = 0; j < n_B; j++)
        {
            PROD[idx_next][0] = A[i];
            PROD[idx_next][1] = B[j];
            idx_next++;
        }
    }

}


int main()
{
    int A[] = {1, 2, 3};
    int B[] = {4, 5, 6};
    printf("A = ");
    stampa_vett(A, 3);
    printf("\nB = ");
    stampa_vett(B, 3);

    int PROD[MAX_ROWS][MAX_COLS];
    cartesian_product(A, 3, B, 3, PROD);
    printf("\nAxB = \n");
    stampa_mat(PROD, 3*3, 2);
}
```

```
A = ( 1,  2,  3)

B = ( 4,  5,  6)

AxB =
(
 1,  4
 1,  5
 1,  6
 2,  4
 2,  5
 2,  6
 3,  4
 3,  5
 3,  6
)
```

Si generalizzi l'operazione precedente al caso in cui gli insiemi A e B siano rappresentati da due matrici in cui ogni riga corrisponde ad un elemento dell'insieme rappresentato.

Bozza di una possibile soluzione:

- *IN*: matrici A, B; *OUT*: matrice $A \times B$
  1. per ogni riga r_i di A:
     a. copio la riga r_i in $A \times B$ nella prima riga libera, a partire dalla prima colonna
     b. copio una riga di B non ancora copiata in $A \times B$ nella stessa riga, ma dalla prima colonna libera
     c. torno ad a.

In [4]:
```
/* Si generalizzi l'operazione precedente al caso in cui gli insiemi $A$ e $B$
siano rappresentati da due matrici in cui ogni riga corrisponde ad un elemento
dell'insieme rappresentato.
*/
#include <stdio.h>

#define MAX_ROWS 10
#define MAX_COLS 10

void stampa_mat(int mat[][MAX_COLS], int n_r, int n_c)
{
    printf("(\n");
    for(int i = 0; i < n_r; i++)
    {
        for(int j = 0; j < n_c; j++)
            printf(" %d, ", mat[i][j]);
        printf("\b\b\n");
    }
    printf(")\n");
}

void cartesian_product(int A[][MAX_COLS], int r_A, int c_A,
                       int B[][MAX_COLS], int r_B, int c_B,
                       int PROD[][MAX_COLS])
{
    int idx_next = 0;

    // per ogni riga di A
    for(int i = 0; i < r_A; i++)
    {
        // per ogni riga di B
        for(int q = 0; q < r_B; q++)
        {
            // copia l'i-esima riga di A nella idx-next-esima riga di PROD
            for(int k = 0; k < c_A; k++)
            {
                PROD[idx_next][k] = A[i][k];

            }

            // copia la q-esima riga di B nella idx-next-esima riga di PROD
            // a partire dalla posizione c_A
            for(int k = 0; k < c_B; k++)
            {
                PROD[idx_next][c_A + k] = B[q][k];

            }
            idx_next++;
        }


    }

}
```

```
int main()
{
    int A[][MAX_COLS] = {{1, 2,10},
                         {3, 4,20}};
    int B[][MAX_COLS] = {{5, 6,30},
                         {7, 8,40}};
    printf("A = \n");
    stampa_mat(A, 2, 3);
    printf("\nB = ");
    stampa_mat(B, 2, 3);

    int PROD[MAX_ROWS][MAX_COLS];
    cartesian_product(A, 2,3, B,2,3, PROD);
    printf("\nAxB = \n");
    stampa_mat(PROD, 2*2, 3+3);
}
```

```
A =
(
 1,  2,  10
 3,  4,  20
)

B = (
 5,  6,  30
 7,  8,  40
)

AxB =
(
 1,  2,  10,  5,  6,  30
 1,  2,  10,  7,  8,  40
 3,  4,  20,  5,  6,  30
 3,  4,  20,  7,  8,  40
)
```

In [2]:

```
/* Si generalizzi l'operazione precedente al caso in cui gli insiemi $A$ e $B$
siano rappresentati da due matrici
in cui ogni riga corrisponde ad un elemento dell'insieme rappresentato.
ALTERNATIVA

*/
#include <stdio.h>

#define MAX_ROWS 10
#define MAX_COLS 10

void stampa_mat(int mat[][MAX_COLS], int n_r, int n_c)
{
    printf("(\n");
    for(int i = 0; i < n_r; i++)
    {
        for(int j = 0; j < n_c; j++)
            printf(" %d, ", mat[i][j]);
        printf("\b\b\n");
    }
    printf(")\n");
}

void copia_riga(int SRC[][MAX_COLS], int src_idx_r, int n_src_c,
                int DST[][MAX_COLS], int dst_idx_r, int dst_start_idx_c)
{
    /*copia la src_idx_r-esima riga di SRC nella dst_idx_r-esima riga di DST
      a partire dalla colonna dst_start_idx_c-esima di DST

    */
    for(int i=0; i < n_src_c; i++)
        DST[dst_idx_r][dst_start_idx_c + i] = SRC[src_idx_r][i];

}

void cartesian_product(int A[][MAX_COLS], int r_A, int c_A,
                       int B[][MAX_COLS], int r_B, int c_B,
                       int PROD[][MAX_COLS])
{
    int idx_next = 0;
    // per ogni riga di A
    for(int i = 0; i < r_A; i++)
    {
        // per ogni riga di B
        for(int q = 0; q < r_B; q++)
        {
            // copia l' i-esima riga di A nell'idx-next-esima riga di PROD
            // a partire dalla colonna 0
            copia_riga(A,i,c_A, PROD, idx_next, 0);
            // copia la q-esima riga di B nell'idx-next-esima riga di PROD
            // a partire dalla colonna c_A
            copia_riga(B,q,c_B, PROD, idx_next, c_A);
```

```
            idx_next++;
        }


    }

}


int main()
{
    int A[][MAX_COLS] = {{1, 2},
                         {3, 4}};
    int B[][MAX_COLS] = {{5, 6},
                         {7, 8}};
    printf("A = \n");
    stampa_mat(A, 2, 2);
    printf("\nB = ");
    stampa_mat(B, 2, 2);

    int PROD[MAX_ROWS][MAX_COLS];
    cartesian_product(A,2,2, B,2,2, PROD);
    printf("\nAxB = \n");
    stampa_mat(PROD, 2*2, 2+2);
}
```

```
A =
(
 1,  2
 3,  4
)

B = (
 5,  6
 7,  8
)

AxB =
(
 1,  2,  5,  6
 1,  2,  7,  8
 3,  4,  5,  6
 3,  4,  7,  8
)
```

dato un vettore di strutture `Votante`, ognuna delle quali contenente `citta_residenza`, `id_partito_votato` (numero da 0 da 3 identificante un dato partito), costruire un secondo vettore di strutture contenente, per ogni città, l'istogramma dei partiti in base alla città. In altri termini, ogni struttura contenuta nel vettore dovrà contenere a sua volta, oltre alla città di riferimento, un array così composto:

- l'elemento 0 dovrà contenere quante persone della rispettiva città hanno votato il partito con id 0
- l'elemento 1 dovrà contenere quante persone della rispettiva città hanno votato il partito con id 1
- l'elemento 2 dovrà contenere quante persone della rispettiva città hanno votato il partito con id 2
- l'elemento 3 dovrà contenere quante persone della rispettiva città hanno votato il partito con id 3

stamparne quindi il diagramma a barre con degli `*`

Possibile soluzione:

Definizioni

- struttura *Votante* con campi *citta_residenza* (stringa), *id_partito_votato* (numero intero)
- struttura *Statistica* con campi *citta_residenza* (stringa), *istogramma* (array lungo almeno quanto il numero dei partiti). Ogni posizione corrisponde ad un partito votato (es. indice 0 sarà il numero di votanti del partito 0 in quella città, indice 1 sarà il numero di votanti del partito 1 in quella città, ecc.)

---

bozza algoritmo

IN: vettore *votanti*; *OUT:* vettore *statistiche*

1. per ogni *votante* in *votanti*:
   a. cerca in *statistiche* se esiste già un elemento avente città di residenza del votante
   b. se esiste, incrementa il vettore in posizione del partito votato da *votante*
   c. altrimenti, inserisci la città di *votante* nella prima posizione libera di *statistiche* ed incrementa il vettore in posizione del partito votato

---

In [4]:
```
#include <stdio.h>
```

```c
#define STR_MAXLEN 100
#define PART_MAXN  10
#define CITTA_MAXN 100

struct Votante
{
    char citta_residenza[STR_MAXLEN];
    unsigned int id_partito_votato;
};

struct Statistica
{
    char citta_residenza[STR_MAXLEN]   ;
    unsigned int istogramma[PART_MAXN] ;
};

int strings_equal(char stringa1[], char stringa2[])
{
    int uguali = 1;

    int i = 0;
    while(stringa1[i]!='\0' && stringa2[i]!='\0' && uguali == 1)
    {
        if(stringa1[i] != stringa2[i])
            uguali = 0;
        i++;
    }

    /* Uscito dal while in 3 casi:
    a. stringa1[i] è \0
    b. stringa2[i] è \0
    c. 'uguali' è 0
    se è uscito dal while nei casi a. e b. potrei pensare che le due stringhe sono uguali.
    MA non avrei considerato il caso in cui le due stringhe sono di lunghezza diversa ed
    una è sottostringa dell'altra
    es. stringa1 = "ciao" e stringa2 = "ciaone"
    In questo caso le due stringhe sono diverse, ma 'uguali' sarebbe uguale ad 1.
    Soluzioni possibili:
    1) conto il numero di caratteri nelle due stringhe
    all'inizio della funzione,
    se sono diversi  allora le due stringhe sono diverse;
    2) Oppure...
    */
    // se le due stringhe non sono della stessa lunghezza
    // l'i-esimo carattere (in uscita dal ciclo precedente) sarà diverso
    if(stringa1[i] != stringa2[i])
        uguali = 0;

    return uguali;

}

void strings_copy(char out[],
                  char in[])
{
    int i = 0;
    while(in[i]!='\0')
    {
        out[i] = in[i];
        i++;
    }
    out[i] = '\0';

}

int idx_is_presente(struct Statistica statistiche[], int n_statistiche,
                    char citta_in_esame[STR_MAXLEN])
{
    int trovata = 0;

    int i = 0;
    while(i < n_statistiche && trovata == 0)
    {
        if(strings_equal(citta_in_esame, statistiche[i].citta_residenza) == 1)
            trovata = 1;
        i++;
    }

    if (trovata == 1)
        return i-1;
    else
        return -1;
}


void genera_statistiche(struct Votante votanti[], int n_votanti,
                        struct Statistica statistiche[], int* n_statistiche)
{
    *n_statistiche = 0;
```

```
    for(int i = 0; i < n_votanti; i++)
    {
        int idx_citta = idx_is_presente(statistiche, *n_statistiche,
                                        votanti[i].citta_residenza);
        if( idx_citta == -1)
        {
            strings_copy(statistiche[*n_statistiche].citta_residenza,
                         votanti[i].citta_residenza);
            idx_citta = *n_statistiche;
            (*n_statistiche)++;
        }


        statistiche[idx_citta].istogramma[votanti[i].id_partito_votato]++;

    }


}

void stampa_istogramma(unsigned int v[], unsigned int n)
{
    for(int i = 0; i < n; i++)
    {
        printf("%2d|", i);
        for(int j = 0; j < v[i]; j++)
            printf("*");
        printf("\n");
    }

}

void stampa_statistiche(struct Statistica statistiche[], int n_statistiche, int n_partiti)
{
    for(int i = 0; i < n_statistiche; i++)
    {
        printf("%s: \n", statistiche[i].citta_residenza);
        stampa_istogramma(statistiche[i].istogramma, n_partiti);
    }

}


int main()
{
 struct Votante votanti[] = {
                                {.citta_residenza = "milano", .id_partito_votato=2},
                                {.citta_residenza = "napoli", .id_partito_votato=1},
                                {.citta_residenza = "roma",   .id_partito_votato=3},
                                {.citta_residenza = "napoli", .id_partito_votato=0},
                                {.citta_residenza = "roma",   .id_partito_votato=1},
                                {.citta_residenza = "milano", .id_partito_votato=0},
                                {.citta_residenza = "napoli", .id_partito_votato=2},
                                {.citta_residenza = "roma",   .id_partito_votato=4},
                                {.citta_residenza = "napoli", .id_partito_votato=2},
                                {.citta_residenza = "roma",   .id_partito_votato=1},
                                {.citta_residenza = "roma",   .id_partito_votato=1},
                                {.citta_residenza = "milano", .id_partito_votato=0},
                                {.citta_residenza = "napoli", .id_partito_votato=2},
                                {.citta_residenza = "roma",   .id_partito_votato=3},
                                {.citta_residenza = "napoli", .id_partito_votato=1},
                                {.citta_residenza = "roma",   .id_partito_votato=0},
                            };
    int n_votanti         = 16;

    int n_partiti         = 4;

    struct Statistica statistiche[CITTA_MAXN] = {{.istogramma = {0} }};
    int n_statistiche     = 0;

    genera_statistiche(votanti, n_votanti, statistiche, &n_statistiche);

    stampa_statistiche(statistiche, n_statistiche, n_partiti);


}
```

```
milano:
 0|**
 1|
 2|*
 3|
napoli:
 0|*
 1|**
 2|***
 3|
roma:
 0|*
 1|***
 2|
 3|**
```

Dato in input un vettore **v**, scrivere una funzione che restituisca un vettore **riassunto** di 3 elementi dove:

- il primo elemento *punta* al minimo di **v**. Se tale valore è presente più volte, si desidera che l'elemento punti alla prima occorrenza
- il secondo elemento *punta* al massimo di **v**. Se tale valore è presente più volte, si desidera che l'elemento punti all'ultima occorrenza
- il terzo elemento *punta* al valore più frequente nel vettore **v**. In caso di parità, si punterà all'ultima occorrenza

In [6]:
```
/* Dato in input un vettore $\mathbf{v}$, scrivere una funzione che restituisca un
vettore $\mathbf{riassunto}$ di 3 elementi dove:
- il primo elemento *punta* al minimo di $\mathbf{v}$. Se tale valore è presente più volte,
  si desidera che l'elemento punti alla prima occorrenza
- il secondo elemento *punta* al massimo di $\mathbf{v}$. Se tale valore è presente più volte,
  si desidera che l'elemento punti all'ultima occorrenza
- il terzo elemento *punta* al valore più frequente nel vettore $\mathbf{v}$.
  In caso di parità, si punterà all'ultima occorrenza
*/

#include <stdio.h>

int* get_address_min(int v[], int n)
{
    int idx_min = 0;
    for(int i = 1; i < n; i++)
    {
        if(v[i] < v[idx_min])
            idx_min = i;
    }

    return &v[idx_min];
}

int* get_address_max(int v[], int n)
{
    int idx_max = 0;
    for(int i = 1; i < n; i++)
    {
        if(v[i] >= v[idx_max])
            idx_max = i;
    }

    return &v[idx_max];
}

int* get_address_most_frequent(int v[], int n)
{
    int idx_most_frequent       = 0;
    int frequency_most_frequent = 0;
    for(int i = 0; i < n; i++)
    {
        int counter = 0;
        for(int j = 0; j < n; j++)
        {
            if (v[i] == v[j])
                counter++;
        }

        if (counter >= frequency_most_frequent)
        {
            idx_most_frequent       = i;
            frequency_most_frequent = counter;
        }
    }
    return &v[idx_most_frequent];

}


int main()
{
    int v[] = {1,2,2,2,3,3,3,3,3,3,3,4,5,1,1,6,7,2,2,2,2,0,7};
    int n   = 23;
    int* riassunto[3];
```

```
    riassunto[0] = get_address_min(v, n);
    riassunto[1] = get_address_max(v, n);
    riassunto[2] = get_address_most_frequent(v, n);

    printf("minimo:        %d idx: %d\n", *riassunto[0], riassunto[0]-v);
    printf("massimo:       %d idx: %d\n", *riassunto[1], riassunto[1]-v);
    printf("piu frequente: %d idx: %d\n", *riassunto[2], riassunto[2]-v);

    return 0;

}
```

```
minimo:        0 idx: 21
massimo:       7 idx: 22
piu frequente: 2 idx: 20
```

- Scrivere una funzione che, data una matrice in input, ne restituisca la sua trasposta in una matrice di output.
- Scrivere una funzione che, data una matrice in input, ne restituisca la sua trasposta *in place*, ossia sovrascrivendo la matrice originaria data in input. [ATTENZIONE]

In [5]:
```
/*
Scrivere una funzione che, data una matrice in input,
ne restituisca la sua trasposta in una matrice di output.
*/

/*
Scrivere una funzione che, data una matrice in input,
ne restituisca la sua trasposta *in place*,
ossia sovrascrivendo la matrice originaria data in input.
*/

#include <stdio.h>
#define MAX_COLS 10
#define MAX_ROWS 10

void stampa_mat(int mat[][MAX_COLS], int n_r, int n_c)
{
    printf("(\n");
    for(int i = 0; i < n_r; i++)
    {
        for(int j = 0; j < n_c; j++)
            printf(" %d, ", mat[i][j]);
        printf("\b\b\n");
    }
    printf(")\n");
}

// calcola la trasposta di una matrice M depositandola in una matrice OUT
void T(int M[][MAX_COLS], int n_r, int n_c,
       int OUT[][MAX_COLS], int* out_n_r, int* out_n_c)
{
    for(int i = 0; i < n_r; i++)
        for(int j = 0; j < n_c; j++)
            OUT[j][i] = M[i][j];

    *out_n_r = n_c;
    *out_n_c = n_r;
}

// calcola la trasposta di una matrice M in place
// NB: VALIDO SOLO PER MATRICI QUADRATE!!!!
void T_inplace(int M[][MAX_COLS], int *n_r, int *n_c)
{
    int t;
    for(int i = 0; i < *n_r; i++)
        for(int j = 0; j < i; j++)
        {
            t       = M[j][i];
            M[j][i] = M[i][j];
            M[i][j] = t;
        }
    t = *n_r;
    *n_r = *n_c;
    *n_c = t;
}



int main()
{
    int M[][MAX_COLS] = {
                          {1,2,3},
                          {4,5,6},
                          {7,8,9},
                          {10,11,12},
                          {13,14,15},
```

```
                    };

    int n_r = 5;
    int n_c = 3;
    printf("M:\n");
    stampa_mat(M, n_r, n_c);


    int M_T[MAX_ROWS][MAX_COLS];
    int t_n_r, t_n_c;
    T(M, n_r, n_c, M_T, &t_n_r, &t_n_c);

    printf("M^T:\n");
    stampa_mat(M_T, t_n_r, t_n_c);

    //NB: SE M NON QUADRATA FUNZIONA ??
    T_inplace(M, &n_r, &n_c);

    printf("M^T (inplace):\n");
    stampa_mat(M, n_r, n_c);

    //NB: SE M NON QUADRATA FUNZIONA ??
    T_inplace(M, &n_r, &n_c);

    printf("M^T^T (inplace):\n");
    stampa_mat(M, n_r, n_c);

    return 0;

}
```

```
M:
(
 1,  2,  3
 4,  5,  6
 7,  8,  9
 10,  11,  12
 13,  14,  15
)
M^T:
(
 1,  4,  7,  10,  13
 2,  5,  8,  11,  14
 3,  6,  9,  12,  15
)
M^T (inplace):
(
 1,  4,  7,  10,  13
 2,  5,  8,  11,  14
 3,  6,  9,  12,  15
)
M^T^T (inplace):
(
 1,  2,  3
 4,  5,  6
 7,  8,  9
 0,  0,  0
 0,  0,  0
)
```

Comoe si vede, nel caso di matrici NON quadrate, la funzione implementata per calcolarne la trasposta in-place **NON FUNZIONA**. Il problema di Calcolare la trasposta in place di una matrice NON quadrata è non banale. [non trattato in questo corso].

Laboratorio di Programmazione Gr. 3 (N-Z)

Corso di Laurea in Informatica

Università degli Studi di Napoli Federico II

A.A. 2021/22

A. Apicella

Black Jack

Fornire una semplice implementazione di un gioco simile al black jack avente come sfidante il computer.

L'utente potrà scegliere ripetutamente tra due possibilità:

1. estrarre una carta
2. fermarsi

nello specifico:

Estrarre una carta

L'estrazione di una carta comporta la produzione di un numero casuale con valore che può andare da 1 a 13. Ad ogni estrazione, tale valore andrà sommato ai valori ottenuti fino a quel momento. La somma di tali valori sarà il punteggio s_u dell'utente. Se, durante le diverse estrazioni, s_u supera il valore 21, la partita termina automaticamente con messaggio per l'utente "hai perso!".

Fermarsi

Tenendo come riferimento il punteggio s_u ottenuto dall'utente, la macchina dovrà provare a battere tale punteggio producendo un punteggio s_m.

La macchina continuerà ad estrarre numeri casuali (tra 1 e 13) finchè il suo punteggio s_m non supera s_u. In altri termini, scopo della macchina è ottenere un punteggio $s_m > s_u$. Se tale punteggio s_m però supera 21 allora la macchina avrà perso, producendo come messaggio all'utente "hai vinto!". Se invece la macchina riesce ad ottenere un punteggio $s_u < s_m < 21$ la macchina avrà vinto, ed il messaggio per l'utente sarà "hai perso!"

In [1]:
```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

#define P_MAX 21
#define C_MIN 1
#define C_MAX 13

long int random_int_in_range(int a, int b)
{
    return rand() % (b + 1 - a) + a;
}


int main()
{
    srand(time(NULL));
    int scelta = -1;
    int s_u    = 0;
    while(scelta != 2 && s_u < P_MAX)
    {
        printf("Punteggio attuale: %d\n", s_u);
        printf("Scegli:\n 1) estrai una carta\n 2) mi fermo\n");
        scanf("%d", &scelta);
        if (scelta == 1)
        {
            printf("Hai scelto di estrarre una carta.\n");

            int estratta = random_int_in_range(C_MIN, C_MAX);
            printf("Hai estratto un %d.\n", estratta);
            s_u          = s_u + estratta;

        }
        else if (scelta != 2)
        {
            printf("Scelta non valida! ripeti!");
```

```
        }
    }

    printf("Il tuo punteggio finale è di %d!", s_u);
    if (s_u >= P_MAX)
    {
        printf(" Hai perso!\n", s_u);
    }
    else
    {
        // turno del computer
        printf(" Turno della macchina!\n");
        int s_m = 0;
        while( s_m < P_MAX && s_m < s_u )
        {
            int estratta = random_int_in_range(C_MIN, C_MAX);
            printf("Ho estratto un %d.\n", estratta);
            s_m          = s_m + estratta;

        }
        printf("Il mio punteggio finale è %d!", s_m);
        if(s_m >= P_MAX)
            printf(" Hai vinto!\n");
        else
            printf(" Hai perso!\n");

    }


    return 0;
}
```

```
Punteggio attuale: 0
Scegli:
 1) estrai una carta
 2) mi fermo

Hai scelto di estrarre una carta.
Hai estratto un 10.
Punteggio attuale: 10
Scegli:
 1) estrai una carta
 2) mi fermo

Hai scelto di estrarre una carta.
Hai estratto un 5.
Punteggio attuale: 15
Scegli:
 1) estrai una carta
 2) mi fermo

Il tuo punteggio finale è di 15! Turno della macchina!
Ho estratto un 7.
Ho estratto un 13.
Il mio punteggio finale è 20! Hai perso!
```

Black Jack v2

Ripetere l'esercizio precedente, ma simulando un *reale* mazzo di 52 carte, composto da 4 gruppi di 13 carte, ogni gruppo appartenente ad un singolo seme (fiori, quadri, cuori, picche). Ad ogni estrazione, deve essere visualizzato sia il valore della carta sia il seme. Tenere conto che una stessa carta non può essere estratta più di una volta.

Consiglio: utilizzare una `struct` per rappresentare ogni singola Carta.

In [2]:
```
// Una prima soluzione può essere quella di estrarre una carta a caso da un mazzo

#include <stdio.h>
#include <time.h>
#include <string.h>

#define N_CARTE 52
#define P_MAX 21
#define C_MIN 1
#define C_MAX 13
#define N_SEMI 4

struct Carta
{
    int valore;
    char seme[10];
};

char* semi[N_SEMI] = {"picche",
                      "fiori",
                      "quadri",
                      "cuori",
                     };
```

```
void inizializza_mazzo(struct Carta mazzo[])
{
    int idx_carta = 0;
    for(int seme = 0; seme < N_SEMI; seme++)
        for(int valore = C_MIN; valore <= C_MAX; valore++)
        {
            strcpy(mazzo[idx_carta].seme, semi[seme]);
            mazzo[idx_carta].valore = valore;
            idx_carta++;
        }

}

void stampa_carta(struct Carta carta)
{
    printf("%d di %s\n", carta.valore, carta.seme);

}

void stampa_mazzo(struct Carta mazzo[], int n_carte)
{
    printf("===============\n");
    printf("CARTE:\n");
    for(int i = 0; i < n_carte; i++)
    {
        stampa_carta(mazzo[i]);
    }
    printf("===============\n");
}

long int random_int_in_range(int a, int b)
{
    return rand() % (b + 1 - a) + a;
}


struct Carta estrai_carta(struct Carta mazzo[], int* n_carte)
{
    int idx_c             = random_int_in_range(1,*n_carte) - 1;
    struct Carta estratta = mazzo[idx_c];
    for(int i = idx_c; i < (*n_carte)-1; i++)
        mazzo[i] = mazzo[i+1];
    (*n_carte)--;
    return estratta;
}

int main()
{
    srand(time(NULL));

    struct Carta mazzo[N_CARTE];

    int n_carte = N_CARTE;

    inizializza_mazzo(mazzo);
    //stampa_mazzo(mazzo, n_carte);
    int scelta = -1;
    int s_u    = 0;
    while(scelta != 2 && s_u < P_MAX)
    {
        printf("Punteggio attuale: %d\n", s_u);
        printf("Scegli:\n 1) estrai una carta\n 2) mi fermo\n");
        scanf("%d", &scelta);
        if (scelta == 1)
        {
            printf("Hai scelto di estrarre una carta.\n");

            struct Carta estratta = estrai_carta(mazzo, &n_carte);
            printf("Hai estratto un ");
            stampa_carta(estratta);
            s_u          = s_u + estratta.valore;

        }
        else if (scelta != 2)
       {
        printf("Scelta non valida! ripeti!");
       }
    }

    printf("Il tuo punteggio finale è di %d!", s_u);
    if (s_u >= P_MAX)
    {
        printf(" Hai perso!\n", s_u);
    }
    else
    {
        // turno del computer
        printf(" Turno della macchina!\n");
```

```
        int s_m = 0;
        while(s_m < P_MAX && s_m < s_u )
        {
            struct Carta estratta = estrai_carta(mazzo, &n_carte);
            printf("Ho estratto un ");
            stampa_carta(estratta);
            s_m          = s_m + estratta.valore;

        }
        printf("Il mio punteggio finale è %d!", s_m);
        if(s_m >= P_MAX)
            printf(" Hai vinto!\n");
        else
            printf(" Hai perso!\n");

    }


    return 0;
}
```

```
Punteggio attuale: 0
Scegli:
 1) estrai una carta
 2) mi fermo

Hai scelto di estrarre una carta.
Hai estratto un 8 di fiori
Punteggio attuale: 8
Scegli:
 1) estrai una carta
 2) mi fermo

Hai scelto di estrarre una carta.
Hai estratto un 11 di fiori
Punteggio attuale: 19
Scegli:
 1) estrai una carta
 2) mi fermo

Il tuo punteggio finale è di 19! Turno della macchina!
Ho estratto un 2 di cuori
Ho estratto un 10 di quadri
Ho estratto un 12 di quadri
Il mio punteggio finale è 24! Hai vinto!
```

In [ ]:

```
// soluzione alternativa: in questo caso, il mazzo viene mischiato a priori,
// per poi estrarre di volta in volta la carta "in alto" (considerando come carta in alto quella in posizione 0

#include <stdio.h>
#include <time.h>
#include <string.h>

#define N_CARTE 52
#define P_MAX 21
#define C_MIN 1
#define C_MAX 13
#define N_SEMI 4

struct Carta
{
    int valore;
    char seme[10];
};

char* semi[N_SEMI] = {"picche",
                      "fiori",
                      "quadri",
                      "cuori",
                     };

void inizializza_mazzo(struct Carta mazzo[])
{
    int idx_carta = 0;
    for(int seme = 0; seme < N_SEMI; seme++)
        for(int valore = C_MIN; valore <= C_MAX; valore++)
        {
            strcpy(mazzo[idx_carta].seme, semi[seme]);
            mazzo[idx_carta].valore = valore;
            idx_carta++;
        }

}

void stampa_carta(struct Carta carta)
{
    printf("%d di %s\n", carta.valore, carta.seme);

}
```

```
void stampa_mazzo(struct Carta mazzo[], int n_carte)
{
    printf("===============\n");
    printf("CARTE:\n");
    for(int i = 0; i < n_carte; i++)
    {
        stampa_carta(mazzo[i]);
    }
    printf("===============\n");
}

long int random_int_in_range(int a, int b)
{
    return rand() % (b + 1 - a) + a;
}


void mischia_mazzo(struct Carta mazzo[], int n_carte, int n_ripetizioni)
{
    for(int r = 0; r < n_ripetizioni; r++)
    {
        int idx_c1 = random_int_in_range(1,n_carte) - 1;
        int idx_c2 = random_int_in_range(1,n_carte) - 1;
        struct Carta tmp = mazzo[idx_c1];
        mazzo[idx_c1]    = mazzo[idx_c2];
        mazzo[idx_c2]    = tmp;
    }

}

struct Carta estrai_carta(struct Carta mazzo[], int* n_carte)
{
    struct Carta estratta = mazzo[0];
    for(int i = 0; i < (*n_carte)-1; i++)
        mazzo[i] = mazzo[i+1];
    (*n_carte)--;
    return estratta;
}

int main()
{
    srand(time(NULL));

    struct Carta mazzo[N_CARTE];

    int n_carte = N_CARTE;

    inizializza_mazzo(mazzo);
    stampa_mazzo(mazzo, n_carte);
    printf("mischio...\n");
    mischia_mazzo(mazzo, n_carte, 26);
    stampa_mazzo(mazzo, n_carte);
    int scelta = -1;
    int s_u    = 0;
    while(scelta != 2 && s_u < P_MAX)
    {
        printf("Punteggio attuale: %d\n", s_u);
        printf("Scegli:\n 1) estrai una carta\n 2) mi fermo\n");
        scanf("%d", &scelta);
        if (scelta == 1)
        {
            printf("Hai scelto di estrarre una carta.\n");

            struct Carta estratta = estrai_carta(mazzo, &n_carte);
            printf("Hai estratto un ");
            stampa_carta(estratta);
            s_u          = s_u + estratta.valore;

        }
        else if (scelta != 2)
       {
        printf("Scelta non valida! ripeti!");
       }
    }

    printf("Il tuo punteggio finale è di %d!", s_u);
    if (s_u >= P_MAX)
    {
        printf(" Hai perso!\n", s_u);
    }
    else
    {
        // turno del computer
        printf(" Turno della macchina!\n");
        int s_m = 0;
        while(s_m < P_MAX && s_m < s_u )
        {
            struct Carta estratta = estrai_carta(mazzo, &n_carte);
            printf("Ho estratto un ");
```

```
            stampa_carta(estratta);
            s_m          = s_m + estratta.valore;

        }
        printf("Il mio punteggio finale è %d!", s_m);
        if(s_m >= P_MAX)
            printf(" Hai vinto!\n");
        else
            printf(" Hai perso!\n");

    }


    return 0;
}
```

```
===============
CARTE:
1 di picche
2 di picche
3 di picche
4 di picche
5 di picche
6 di picche
7 di picche
8 di picche
9 di picche
10 di picche
11 di picche
12 di picche
13 di picche
1 di fiori
2 di fiori
3 di fiori
4 di fiori
5 di fiori
6 di fiori
7 di fiori
8 di fiori
9 di fiori
10 di fiori
11 di fiori
12 di fiori
13 di fiori
1 di quadri
2 di quadri
3 di quadri
4 di quadri
5 di quadri
6 di quadri
7 di quadri
8 di quadri
9 di quadri
10 di quadri
11 di quadri
12 di quadri
13 di quadri
1 di cuori
2 di cuori
3 di cuori
4 di cuori
5 di cuori
6 di cuori
7 di cuori
8 di cuori
9 di cuori
10 di cuori
11 di cuori
12 di cuori
13 di cuori
===============
mischio...
===============
CARTE:
6 di quadri
2 di picche
3 di quadri
5 di fiori
5 di picche
6 di picche
12 di fiori
2 di cuori
12 di picche
10 di picche
11 di picche
9 di picche
5 di cuori
8 di picche
2 di fiori
```

```
4 di picche
3 di cuori
13 di picche
6 di fiori
7 di fiori
8 di fiori
10 di quadri
10 di fiori
1 di cuori
4 di cuori
13 di fiori
1 di quadri
9 di quadri
3 di picche
7 di picche
5 di quadri
7 di quadri
1 di picche
2 di quadri
8 di quadri
11 di quadri
12 di quadri
7 di cuori
13 di quadri
11 di fiori
1 di fiori
8 di cuori
9 di fiori
4 di fiori
6 di cuori
9 di cuori
3 di fiori
4 di quadri
10 di cuori
11 di cuori
12 di cuori
13 di cuori
===============
Punteggio attuale: 0
Scegli:
 1) estrai una carta
 2) mi fermo
```

In [ ]:

Scrivere un programma che effettui una simulazione del classico gioco dell'*Impiccato* su una frase, così definito:

- viene assegnato all'utente un numero massimo di tentativi con cui può indovinare una frase (e.g., 5)
- viene presentata all'utente una serie di asterischi che indicano la struttura della frase, i.e. da quante lettere ogni parola è composta e quante parole sono e.g. `** *** ******` indica che la frase è costituita da 3 parole rispettivamente di 2, 3 e 6 lettere. Eventuali lettere accentate, apostrofi, segni di punteggiatura e cifre numeriche devono essere visualizzati di default, e.g., la frase "l'orco è vicino" dovrà essere presentata come `*'**** è ******`

L'utente può quindi scegliere se:

- fornire una lettera: scegliere una lettera (maiuscola o minuscola non deve far differenza), se la lettera è contenuta nella frase, tutte le sue corrispondenze vengono svelate (ossia gli asterischi corrispondenti vengono tolti) altrimenti viene consumato un tentativo.
- provare a dare la soluzione: l'utente può provare a digitare la frase, se la soluzione è corretta il giocatore ha vinto, altrimenti viene consumato un tentativo

Quando il numero dei tentativi raggiunge zero, l'utente ha perso ed il gioco termina.

L'insieme delle frasi possibili è contenuto in un file di testo *dizionario.txt* contenente una sequenza di frasi (e null'altro). Ogni frase deve essere lunga al massimo 255 caratteri. All'inizio del gioco, il programma sceglierà casualmente una frase tra tutte le frasi disponibili

In [ ]:

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

#define MAX_LEN 256

void stampa_mat(char M[][MAX_LEN], int n_r)
{
    for(int i = 0; i < n_r; i++)
        printf("%s\n", M[i]);

}

int load_file_in_memory(char filename[], char FRASI[][MAX_LEN], int* n_frasi)
{

    FILE* fp = fopen(filename, "r");
    if (fp == NULL)
    {
        return -1;
    }


    *n_frasi = 0;

    while(fgets(FRASI[*n_frasi], MAX_LEN, fp))
    {
        // pulisci da \n
        int i = 0;
        while(FRASI[*n_frasi][i] !='\0')
        {
            if(FRASI[*n_frasi][i]=='\n')
                FRASI[*n_frasi][i] = '\0';
            i++;
        }
        (*n_frasi)++;
    }

    fclose(fp);
    return 1;
}



int random_int_in_range(int a, int b)
{
    return rand() % (b + 1 - a) + a;
}


void mostra_frase(char frase[MAX_LEN], int flags[MAX_LEN])
{
    int i = 0;
    while(frase[i]!='\0')
    {
      if( flags[i] == 0 &&((frase[i]>='A' && frase[i]<='Z') || (frase[i]>='a' && frase[i]<='z')))
        printf("*");
      else
```

```
      else
        printf("%c",frase[i]);
     i++;
    }
    printf("\n");
}

char to_lower(char c)
{
    if (c >='a' && c<='z')
          return c;
    else
        return c+32;


}

int verifica_se_in_stringa(char stringa[], char carattere, int idxs[])
{
    int n_idxs = 0;
    int i = 0;
    while(stringa[i]!='\0')
    {
        if (to_lower(stringa[i]) == to_lower(carattere))
        {

            idxs[n_idxs] = i;
            n_idxs++;
        }
        i++;
    }
    return n_idxs;
}

int verifica_se_soluzione(char stringa[], char sol[])
{
    int corretta = 1;
    int i = 0;
    //printf("V: %s, %s\n", stringa, sol);
    while(stringa[i]!='\0' && sol[i]!='\0' && to_lower(stringa[i]) == to_lower(sol[i]))
        i++;

    if (to_lower(stringa[i]) != to_lower(sol[i]))
        corretta = 0;
    return corretta;
}

int main()
{
    srand(time(NULL));
    char FRASI[100][MAX_LEN];
    int n_frasi;
    load_file_in_memory("dizionario.txt", FRASI, &n_frasi);
    //stampa_mat(FRASI, n_frasi);
    int idx_frase_scelta = random_int_in_range(0, n_frasi-1);

    int flags[MAX_LEN] = {0};

    int tentativi  = 5;
    int scelta     = -1;
    while( tentativi >0 && scelta!=0)
    {
        mostra_frase(FRASI[idx_frase_scelta], flags);
        printf("tentativi residui: %d\n", tentativi);
        printf("\nscegli:\n1) dò una lettera\n2) provo a dare una soluzione\n0)esci\n");
        scanf("%d", &scelta);
        if(scelta == 1)
        {
            printf("inserisci una lettera\n");
            char l;
            getchar();
            scanf("%c", &l);

            int idxs[256];
            int n_idxs = verifica_se_in_stringa(FRASI[idx_frase_scelta], l, idxs);
            for(int i = 0; i < n_idxs; i++)
                flags[idxs[i]] = 1;
            if(n_idxs == 0)
            {
                printf("sbagliato!\n");
                tentativi --;
            }

        }
```

```
        else if (scelta == 2)

        {
            char soluzione[MAX_LEN];
            printf("prova a darmi la soluzione:\n");
            getchar();
            //fgets(soluzione, MAX_LEN, stdin);
            scanf("%[^\n]s",soluzione);
            printf("hai inserito \"%s\"\n", soluzione);

            int vinto = verifica_se_soluzione(FRASI[idx_frase_scelta], soluzione);
            if(vinto == 1)
            {
                printf("hai vinto!\n");
                scelta = 0;
            }
            else
            {
                printf("la frase non è corretta!\n");
                tentativi --;
            }
        }
        else if (scelta !=0)
        {
            printf("scelta sbagliata!ripeti!\n");
        }
    }
    if(tentativi <=0)
        printf("hai perso!\n");
    printf("la frase è %s\n", FRASI[idx_frase_scelta]);
}
```

```
*'**** è ******
tentativi residui: 5

scegli:
1) dò una lettera
2) provo a dare una soluzione
0)esci

prova a darmi la soluzione:

hai inserito "uno due tre"
la frase non è corretta!
*'**** è ******
tentativi residui: 4

scegli:
1) dò una lettera
2) provo a dare una soluzione
0)esci
prova a darmi la soluzione:
hai inserito " due tre"
la frase non è corretta!
*'**** è ******
tentativi residui: 3

scegli:
1) dò una lettera
2) provo a dare una soluzione
0)esci
prova a darmi la soluzione:
hai inserito "e tre"
la frase non è corretta!
*'**** è ******
tentativi residui: 2

scegli:
1) dò una lettera
2) provo a dare una soluzione
0)esci
prova a darmi la soluzione:
hai inserito "re"
la frase non è corretta!
*'**** è ******
tentativi residui: 1

scegli:
1) dò una lettera
2) provo a dare una soluzione
0)esci
prova a darmi la soluzione:
```

In [2]:

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

#define MAX_LEN 256

void stampa_mat(char M[][MAX_LEN], int n_r)
{
    for(int i = 0; i < n_r; i++)
        printf("%s\n", M[i]);

}

int load_sentence_in_memory(char filename[], char sentence[], int idx)
{

    FILE* fp = fopen(filename, "r");

    if (fp == NULL)
    {
        return -1;
    }


    while(idx >= 0)
    {
        char* ret = fgets(sentence, MAX_LEN, fp);
        if(ret == NULL)
        {
            rewind(fp);
            fgets(sentence, MAX_LEN, fp);

        }
        idx--;
    }

    fclose(fp);
    return 1;
}


int random_int_in_range(int a, int b)
{
    return rand() % (b + 1 - a) + a;
}


void mostra_frase(char frase[MAX_LEN], int flags[MAX_LEN])
{
    int i = 0;
    while(frase[i]!='\0')
    {
      if( flags[i] == 0 &&((frase[i]>='A' && frase[i]<='Z') || (frase[i]>='a' && frase[i]<='z')))
        printf("*");
      else
        printf("%c",frase[i]);
     i++;
    }
    printf("\n");
}

char to_lower(char c)
{
    if (c >='a' && c<='z')
          return c;
    else
        return c+32;


}

int verifica_se_in_stringa(char stringa[], char carattere, int idxs[])
{
    int n_idxs = 0;
    int i = 0;
    while(stringa[i]!='\0')
    {
        if (to_lower(stringa[i]) == to_lower(carattere))
        {

            idxs[n_idxs] = i;
            n_idxs++;
        }
```

```
        i++;

    }
    return n_idxs;
}

int verifica_se_soluzione(char stringa[], char sol[])
{
    int corretta = 1;
    int i = 0;
    //printf("V: %s, %s\n", stringa, sol);
    while(stringa[i]!='\0' && sol[i]!='\0' && to_lower(stringa[i]) == to_lower(sol[i]))
        i++;

    if (to_lower(stringa[i]) != to_lower(sol[i]))
        corretta = 0;
    return corretta;
}

int main()
{
    srand(time(NULL));
    char frase[MAX_LEN];

    int idx_frase_scelta = random_int_in_range(0,1000);
    load_sentence_in_memory("dizionario.txt", frase, idx_frase_scelta);



    int flags[MAX_LEN] = {0};

    int tentativi  = 5;
    int scelta = -1;
    while( tentativi >0 && scelta!=0)
    {
        mostra_frase(frase, flags);
        printf("tentativi residui: %d\n", tentativi);
        printf("\nscegli:\n1) dò una lettera\n2) provo a dare una soluzione\n0)esci\n");
        scanf("%d", &scelta);
        if(scelta == 1)
        {
            printf("inserisci una lettera\n");
            char l;
            getchar();
            scanf("%c", &l);
            int idxs[256];
            int n_idxs = verifica_se_in_stringa(frase, l, idxs);
            for(int i = 0; i < n_idxs; i++)
                flags[idxs[i]] = 1;
            if(n_idxs == 0)
            {
                printf("sbagliato!\n");
                tentativi --;
            }

        }
        else if (scelta == 2)
        {
            char soluzione[MAX_LEN];
            printf("prova a darmi la soluzione:\n");
            getchar();
            //fgets(soluzione, MAX_LEN, stdin);
            scanf("%[^\n]s",soluzione);
            printf("hai inserito \"%s\"\n", soluzione);

            int vinto = verifica_se_soluzione(frase, soluzione);
            if(vinto == 1)
            {
                printf("hai vinto!\n");
                scelta = 0;
            }
            else
            {
                printf("la frase non è corretta!\n");
                tentativi--;
            }
        }
        else if (scelta !=0)
        {
            printf("scelta sbagliata!ripeti!\n");
        }
    }
    if(tentativi <=0)
        printf("hai perso!\n");
```

```
    printf("la frase è %s\n", frase);

}
```

```
******* ** ******** ** *****

tentativi residui: 5

scegli:
1) dò una lettera
2) provo a dare una soluzione
0)esci

la frase è Orlando ha smarrito il senno
```

Si vuole realizzare un programma che simuli una cassafarte i cui dati sono contenuti in un file binario.

Tale cassaforte sarà costituita da:

- una combinazione (numero intero)
- un ammontare di denaro All'utente deve essere presentato un menù in cui può scegliere se:
- creare una nuova cassaforte
- aprire la cassaforte
- inserire denaro
- prelevare denaro
- visualizzare il denaro
- cambiare la combinazione
- chiudere la cassaforte
- uscire dal programma

La creazione di una nuova cassaforte distruggerà dati della precedente e darà la possibilità all'utente di impostare una combinazione.

Tutte le operazioni potranno essere effettuate solo se la cassaforte è aperta. Una volta chiusa la cassaforte, il file binario dovrà essere aggiornato con le nuove informazioni.

In [15]:

```
#include <stdio.h>

#define FILENAME "cassa.bin"
struct Cassaforte{
    int combinazione;
    float ammontare;
    int stato; // 0: chiusa 1: aperta
};


void inizializza_cassaforte(struct Cassaforte* cassaforte)
{
    cassaforte->ammontare = 0;
    cassaforte->stato     = 0;

}
void crea_nuova_cassaforte(struct Cassaforte* cassaforte_nuova)
{

    printf("scegli la combinazione: ");
    scanf("%d", &(cassaforte_nuova->combinazione));

    inizializza_cassaforte(cassaforte_nuova);
    FILE* fp = fopen(FILENAME, "wb");
    if(fp == NULL)
    {
        printf("qualcosa è andato storto durante l'apertura del file in scrittura\n");
        return;
    }
    int n_scritti = fwrite(cassaforte_nuova, sizeof(struct Cassaforte), 1, fp);
    if(n_scritti < 1)
    {
        printf("qualcosa è andato storto durante la scrittura del file\n");
        fclose(fp);
        return;
    }
    fclose(fp);

}


void apri(int combinazione_data, struct Cassaforte* cassaforte)
{
    struct Cassaforte tmp;
```

```
    FILE* fp = fopen(FILENAME, "rb");
    if(fp == NULL)
    {
        printf("qualcosa è andato storto durante l'apertura del file in lettura\n");
        return ;
    }
    int n_letti = fread(&tmp, sizeof(struct Cassaforte), 1, fp);
    if(n_letti < 1)
    {
        printf("qualcosa è andato storto durante la lettura del file\n");
        fclose(fp);
        return ;
    }
    fclose(fp);

    if(combinazione_data == tmp.combinazione)
    {
        *cassaforte        = tmp;
        cassaforte->stato  = 1;
        return ;
    }
    else
    {
        printf("Combinazione errata!\n");

    }

}

void chiudi(struct Cassaforte* cassaforte)
{
    FILE* fp = fopen(FILENAME, "wb");
    if(fp == NULL)
    {
        printf("qualcosa è andato storto durante l'apertura del file in scrittura\n");
        return ;
    }
    int n_scritti = fwrite(cassaforte, sizeof(struct Cassaforte), 1, fp);
    if(n_scritti < 1)
    {
        printf("qualcosa è andato storto durante la scrittura del file\n");
        fclose(fp);
        return;
    }
    fclose(fp);

    cassaforte->stato  = 0;

}

void visualizza_stato(struct Cassaforte cassaforte)
{
    if(cassaforte.stato == 0)
    {
        printf("la cassaforte è chiusa!\n");
    }
    else
    {
        printf("la cassaforte è aperta!\n");
    }

}

void visualizza_ammontare(struct Cassaforte cassaforte)
{
    if(cassaforte.stato == 0)
    {
        printf("la cassaforte è chiusa!\n");
        return;
    }
    printf("il tuo denaro ammonta a %.2f\n", cassaforte.ammontare);
}

void preleva(struct Cassaforte* cassaforte, float somma)
{
    if(cassaforte->stato == 0)
    {
        printf("la cassaforte è chiusa!\n");
        return;
    }
    if(cassaforte->ammontare < somma)
    {
        printf("non hai abbastanza soldi!\n");
```

```
        return;

    }

    cassaforte->ammontare = cassaforte->ammontare - somma;

}

void deposita(struct Cassaforte* cassaforte, float somma)
{
    if(cassaforte->stato == 0)
    {
        printf("la cassaforte è chiusa!");
        return;
    }
    cassaforte->ammontare = cassaforte->ammontare + somma;

}

int main()
{
    int scelta = -1;

    struct Cassaforte cassaforte;
    inizializza_cassaforte(&cassaforte);
    while(scelta != 0)
    {
        printf("sceg1i:\n");
        printf("1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)\n");
        printf("2. apri cassaforte\n");
        printf("3. visualizza lo stato della cassaforte\n");
        printf("4. visualizza ammontare\n");
        printf("5. deposita somma\n");
        printf("6. preleva somma\n");
        printf("7. chiudi cassaforte\n");
        printf("0. esci (eventuali modifiche a cassaforte aperta andranno perse)\n");
        scanf("%d", &scelta);

        if(scelta == 1)
        {
            crea_nuova_cassaforte(&cassaforte);
        }
        else if(scelta == 2)
        {

            int combinazione_data;
            printf("inserisci la combinazione: ");
            scanf("%d", &combinazione_data);
            apri(combinazione_data, &cassaforte);

        }
        else if(scelta == 3)
        {
            visualizza_stato(cassaforte);
        }
        else if(scelta == 4)
        {
            visualizza_ammontare(cassaforte);
        }
        else if(scelta == 5)
        {
            float somma;
            printf("inserisci la somma da inserire: ");
            scanf("%f", &somma);
            deposita(&cassaforte, somma);
        }
        else if(scelta == 6)
        {
            float somma;
            printf("inserisci la somma da prelevare: ");
            scanf("%f", &somma);
            preleva(&cassaforte, somma);

        }
        else if(scelta == 7)
        {
            chiudi(&cassaforte);
        }
        else if(scelta != 0)
        {
            printf("scelta errata!\n");
        }
    }
```

```
}
```

```
sceg1i:
1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)
2. apri cassaforte
3. visualizza lo stato della cassaforte
4. visualizza ammontare
5. deposita somma
6. preleva somma
7. chiudi cassaforte
0. esci (eventuali modifiche a cassaforte aperta andranno perse)

inserisci la combinazione:

qualcosa è andato storto durante l'apertura del file in lettura
sceg1i:
1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)
2. apri cassaforte
3. visualizza lo stato della cassaforte
4. visualizza ammontare
5. deposita somma
6. preleva somma
7. chiudi cassaforte
0. esci (eventuali modifiche a cassaforte aperta andranno perse)
```

In [11]:

```
#include <stdio.h>

#define FILENAME "cassa.bin"

#define OK              0
#define ERR_OPEN_WRITE  1
#define ERR_OPEN_READ   2
#define ERR_WRITE       3
#define ERR_READ        4

#define ERR_COMB        5
#define ERR_CHIUSA      6
#define ERR_AMM_INSUFF  7



struct Cassaforte{
    int combinazione;
    float ammontare;
    int stato; // 0: chiusa 1: aperta
};


void error_handler(int err_code)
{
    switch(err_code)
    {
        case ERR_OPEN_WRITE:
            printf("qualcosa è andato storto durante l'apertura del file in scrittura\n");
        break;
        case ERR_OPEN_READ:
            printf("qualcosa è andato storto durante l'apertura del file in lettura\n");
        break;
        case ERR_WRITE:
            printf("qualcosa è andato storto durante la scrittura del file\n");
        break;
        case ERR_READ:
            printf("qualcosa è andato storto durante la lettura del file\n");
        break;
        case ERR_COMB:
            printf("Combinazione errata!\n");
        break;
        case ERR_CHIUSA:
            printf("la cassaforte è chiusa!\n");
        break;
        case ERR_AMM_INSUFF:
            printf("non hai abbastanza soldi!\n");
        break;
        case OK:
        break;
        default:
            printf("ERRORE IMPREVISTO: CODICE %d!\n", err_code);
    }

}


void inizializza_cassaforte(struct Cassaforte* cassaforte)
```

```
void inizializza_cassaforte(struct Cassaforte* cassaforte)
{
    cassaforte->ammontare = 0;
    cassaforte->stato     = 0;

}

int crea_nuova_cassaforte(struct Cassaforte* cassaforte_nuova)
{

    printf("scegli la combinazione: ");
    scanf("%d", &(cassaforte_nuova->combinazione));


    inizializza_cassaforte(cassaforte_nuova);
    FILE* fp = fopen(FILENAME, "wb");
    if(fp == NULL)
        return ERR_OPEN_WRITE;

    // scrivo sul file solo i campi che mi interessano
    int n_scritti = fwrite(&(cassaforte_nuova->combinazione),
                           sizeof(cassaforte_nuova->combinazione), 1, fp);
    if(n_scritti < 1)
    {
        fclose(fp);
        return ERR_WRITE;
    }
    n_scritti = fwrite(&(cassaforte_nuova->ammontare),
                       sizeof(cassaforte_nuova->ammontare), 1, fp);
    if(n_scritti < 1)
    {
        fclose(fp);
        return ERR_WRITE;
    }

    fclose(fp);
    return OK;

}


int apri(int combinazione_data, struct Cassaforte* cassaforte)
{
    struct Cassaforte tmp;
    FILE* fp = fopen(FILENAME, "rb");
    if(fp == NULL)
        return ERR_OPEN_READ;
    int n_letti = fread(&(tmp.combinazione),
                        sizeof(tmp.combinazione), 1, fp);
    if(n_letti < 1)
    {
        fclose(fp);
        return ERR_READ;
    }
    n_letti = fread(&(tmp.ammontare),
                    sizeof(tmp.ammontare), 1, fp);
    if(n_letti < 1)
    {
        fclose(fp);
        return ERR_READ;
    }
    fclose(fp);

    if(combinazione_data == tmp.combinazione)
    {
        *cassaforte        = tmp;
        cassaforte->stato  = 1;
        return OK;
    }
    else
        return ERR_COMB;

}

int chiudi(struct Cassaforte* cassaforte)
{
    FILE* fp = fopen(FILENAME, "wb");
    if(fp == NULL)
        return ERR_OPEN_WRITE;

    int n_scritti = fwrite(&(cassaforte->combinazione),
                           sizeof(cassaforte->combinazione), 1, fp);
    if(n_scritti < 1)
    {
```

```
        fclose(fp);

        return ERR_WRITE;
    }
    n_scritti = fwrite(&(cassaforte->ammontare),
                       sizeof(cassaforte->ammontare), 1, fp);
    if(n_scritti < 1)
    {
        fclose(fp);
        return ERR_WRITE;

    }
    fclose(fp);
    cassaforte->stato  = 0;
    return OK;

}

void visualizza_stato(struct Cassaforte cassaforte)
{
    if(cassaforte.stato == 0)
        printf("la cassaforte è chiusa!\n");
    else
        printf("la cassaforte è aperta!\n");
}

int visualizza_ammontare(struct Cassaforte cassaforte)
{
    if(cassaforte.stato == 0)
        return ERR_CHIUSA;
    printf("il tuo denaro ammonta a %.2f\n", cassaforte.ammontare);
    return OK;
}

int preleva(struct Cassaforte* cassaforte, float somma)
{
    if(cassaforte->stato == 0)
        return ERR_CHIUSA;
    if(cassaforte->ammontare < somma)
        return ERR_AMM_INSUFF;

    cassaforte->ammontare = cassaforte->ammontare - somma;
    return OK;
}

int deposita(struct Cassaforte* cassaforte, float somma)
{
    if(cassaforte->stato == 0)
        return ERR_CHIUSA;
    cassaforte->ammontare = cassaforte->ammontare + somma;
    return OK;
}

int main()
{
    int scelta = -1;

    struct Cassaforte cassaforte;
    inizializza_cassaforte(&cassaforte);
    while(scelta != 0)
    {
        printf("sceg1i:\n");
        printf("1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)\n");
        printf("2. apri cassaforte\n");
        printf("3. visualizza lo stato della cassaforte\n");
        printf("4. visualizza ammontare\n");
        printf("5. deposita somma\n");
        printf("6. preleva somma\n");
        printf("7. chiudi cassaforte\n");
        printf("0. esci (eventuali modifiche a cassaforte aperta andranno perse)\n");
        scanf("%d", &scelta);

        if(scelta == 1)
        {
            int err = crea_nuova_cassaforte(&cassaforte);
            error_handler(err);
        }
        else if(scelta == 2)
        {

            int combinazione_data;
            printf("inserisci la combinazione: ");
            scanf("%d", &combinazione_data);
            int err = apri(combinazione_data, &cassaforte);
```

```
            error_handler(err);

        }
        else if(scelta == 3)
        {
            visualizza_stato(cassaforte);
        }
        else if(scelta == 4)
        {
            int err = visualizza_ammontare(cassaforte);
            error_handler(err);
        }
        else if(scelta == 5)
        {
            float somma;
            printf("inserisci la somma da inserire: ");
            scanf("%f", &somma);
            int err = deposita(&cassaforte, somma);
            error_handler(err);
        }
        else if(scelta == 6)
        {
            float somma;
            printf("inserisci la somma da prelevare: ");
            scanf("%f", &somma);
            int err = preleva(&cassaforte, somma);
            error_handler(err);

        }
        else if(scelta == 7)
        {
            int err = chiudi(&cassaforte);
            error_handler(err);
        }
        else if(scelta != 0)
        {
            printf("scelta errata!\n");
        }
    }

}
```

```
sceg1i:
1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)
2. apri cassaforte
3. visualizza lo stato della cassaforte
4. visualizza ammontare
5. deposita somma
6. preleva somma
7. chiudi cassaforte
0. esci (eventuali modifiche a cassaforte aperta andranno perse)

la cassaforte è chiusa!
sceg1i:
1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)
2. apri cassaforte
3. visualizza lo stato della cassaforte
4. visualizza ammontare
5. deposita somma
6. preleva somma
7. chiudi cassaforte
0. esci (eventuali modifiche a cassaforte aperta andranno perse)

inserisci la combinazione:

sceg1i:
1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)
2. apri cassaforte
3. visualizza lo stato della cassaforte
4. visualizza ammontare
5. deposita somma
6. preleva somma
7. chiudi cassaforte
0. esci (eventuali modifiche a cassaforte aperta andranno perse)

il tuo denaro ammonta a 100.00
sceg1i:
1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)
2. apri cassaforte
3. visualizza lo stato della cassaforte
4. visualizza ammontare
5. deposita somma
6. preleva somma
7. chiudi cassaforte
0. esci (eventuali modifiche a cassaforte aperta andranno perse)


In [ ]:
```

Laboratorio di Programmazione Gr. 3 (N-Z)

Corso di Laurea in Informatica

Università degli Studi di Napoli Federico II

A.A. 2021/22

A. Apicella

- Implementare una struttura dati con politica di accesso Coda con relative funzioni di accesso utilizzando come struttura deposito un array allocato dinamicamente. Tale array dovrà modificare la propria dimensione ad ogni inserimento/rimozione di un elemento.

In [15]:
```
// CODA: Una migliore implementazione

#include <stdio.h>
#define N_CODA 4
#define FULL -1
#define EMPTY -2
#define ERRMEM -3
#define OK 0

struct Coda
{
    int* deposito;
    int size;
};

void init_queue (struct Coda* c)
{
    c->deposito = NULL;
    c->size     = 0; // numero di elementi presenti
}

void print_queue(struct Coda c)
{
    printf("(  ");
    for(int i = 0; i < c.size; i++)
        printf("%d, ", c.deposito[i]);
    printf("\b\b)\n");

}

int is_queue_empty(struct Coda* c)
{
    if(c->size == 0)
    {
        return 1;
    }
    return 0;
}

int is_queue_full(struct Coda* c)
{
    if (c->size >= N_CODA)
    {
        return 1;
    }
    return 0;
}


int insert_in_queue(struct Coda* c, int elemento)
{

    if(is_queue_full(c) == 1)
        return FULL;

    if (c->deposito == NULL)
        c->deposito = malloc(sizeof(int));
    else
    {
        int* tmp    = realloc(c->deposito, (c->size+1)*sizeof(int));
        if (tmp == NULL) // se qualcosa è andato storto
            return ERRMEM;
        // se la riallocazione è andata bene, aggiorna la coda
        c->deposito = tmp;
    }
```

```
    c->deposito[c->size] = elemento;
    c->size++;
    return OK;

}

int read_from_queue(struct Coda* c, int* p_val)
{
    if(is_queue_empty(c) == 1)
        return EMPTY;

    *p_val =  c->deposito[0];
    return OK;
}

int dequeue(struct Coda* c, int* p_val)
{

    if(is_queue_empty(c) == 1)
        return EMPTY;
    read_from_queue(c, p_val);

    if (c->size > 1)
    {
        int* tmp = malloc((c->size-1)*sizeof(int));
        if(tmp == NULL)
            return ERRMEM;
        // copio la coda nel nuovo array (a esclusione dal primo elemento)
        for(int i = 0; i < c->size-1; i++)
            tmp[i] = c->deposito[i+1];

        free(c->deposito);
        c->deposito = tmp;

    }
    else // se nella coda era presente un solo elemento
    {
        free(c->deposito);
        c->deposito = NULL;
    }

    c->size--;
    return OK;

}

void error_handler(int err)
{
    switch(err)
    {
        case FULL:
            printf("coda piena!\n");
            break;
        case EMPTY:
            printf("coda vuota!\n");
            break;
        case ERRMEM:
            printf("errore di memoria!\n");
            break;
        case OK:
            printf("operazione eseguita con successo!\n");
            break;
        default:
            printf("errore sconosciuto!\n");
    }

}

int main()
{
    struct Coda c;
    init_queue(&c);

    int scelta = -1;
    while(scelta != 0)
    {
        printf("digitare:\n");
        printf("1) inserire in coda\n");
        printf("2) leggere dalla coda\n");
        printf("3) leggere e rimuovere dalla coda\n");
        printf("4) visualizzare l'intera coda\n");
        printf("0) uscire\n");
        scanf("%d", &scelta);

        int val, err;
        switch(scelta)
        {
            case 1:
```

```
                printf("Inserisci valore: ");
                scanf("%d", &val);
                err = insert_in_queue(&c, val);
                error_handler(err);
                break;

            case 2:
                err = read_from_queue(&c, &val);
                if (err == OK)
                    printf("il valore in testa era %d\n", val);
                error_handler(err);
                break;

            case 3:
                err = dequeue(&c, &val);
                if (err == OK)
                    printf("il valore in testa era %d\n", val);
                error_handler(err);
                break;
            case 4:
                print_queue(c);
                break;
            case 0:
                printf("ciao!\n");
                break;

            default:
                printf("scelta errata!\n");
        }

    }
}
```

```
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
4) visualizzare l'intera coda
0) uscire

coda vuota!
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
4) visualizzare l'intera coda
0) uscire

coda vuota!
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
4) visualizzare l'intera coda
0) uscire

coda vuota!
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
4) visualizzare l'intera coda
0) uscire

Inserisci valore:

operazione eseguita con successo!
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
4) visualizzare l'intera coda
0) uscire

Inserisci valore:

operazione eseguita con successo!
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
4) visualizzare l'intera coda
0) uscire

(  10, 20)
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
4) visualizzare l'intera coda
0) uscire
```

```
il valore in testa era 10
operazione eseguita con successo!
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
4) visualizzare l'intera coda
0) uscire

il valore in testa era 20
operazione eseguita con successo!
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
4) visualizzare l'intera coda
0) uscire

coda vuota!
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
4) visualizzare l'intera coda
0) uscire

coda vuota!
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
4) visualizzare l'intera coda
0) uscire

coda vuota!
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
4) visualizzare l'intera coda
0) uscire

ciao!
```

- Implementare una struttura dati con politica di accesso Pila con relative funzioni di accesso utilizzando come struttura deposito un array allocato dinamicamente. Tale array dovrà modificare la propria dimensione ad ogni inserimento/rimozione di un elemento

In [17]:
```
// Pila

#include <stdio.h>
#define N_PILA 4
#define FULL -1
#define EMPTY -2
#define ERRMEM -3
#define OK 0

struct Pila
{
    int* deposito;
    int size;

};

void init_stack (struct Pila* c)
{
    c->deposito = NULL;
    c->size     = 0; // numero di elementi presenti
}

void print_stack(struct Pila c)
{
    printf("(  ");
    for(int i = 0; i < c.size; i++)
        printf("%d, ", c.deposito[i]);
    printf("\b\b)\n");

}

int is_stack_empty(struct Pila* c)
{
    if(c->size == 0)
    {
        return 1;
    }
    return 0;
}
```

```
int is_stack_full(struct Pila* c)
{
    if (c->size >= N_PILA)
    {
        return 1;
    }
    return 0;
}


int push(struct Pila* c, int elemento)
{

    if(is_stack_full(c) == 1)
        return FULL;

    if (c->deposito == NULL)
        c->deposito = malloc(sizeof(int));
    else
    {
        int* tmp    = realloc(c->deposito, (c->size+1)*sizeof(int));
        if (tmp == NULL) // se qualcosa è andato storto
            return ERRMEM;
        // se la riallocazione è andata bene, aggiorna la Pila
        c->deposito = tmp;
    }

    c->deposito[c->size] = elemento;
    c->size++;
    return OK;

}

int read_from_stack(struct Pila* c, int* p_val)
{
    if(is_stack_empty(c) == 1)
        return EMPTY;

    *p_val =  c->deposito[c->size-1];
    return OK;
}

int pop(struct Pila* c, int* p_val)
{

    if(is_stack_empty(c) == 1)
        return EMPTY;
    read_from_stack(c, p_val);

    if (c->size > 1)
    {
        int* tmp = realloc(c->deposito, (c->size-1)*sizeof(int));
        if(tmp == NULL)
            return ERRMEM;
        c->deposito = tmp;

    }
    else // se nella Pila era presente un solo elemento
    {
        free(c->deposito);
        c->deposito = NULL;
    }

    c->size--;
    return OK;

}

void error_handler(int err)
{
    switch(err)
    {
        case FULL:
            printf("Pila piena!\n");
            break;
        case EMPTY:
            printf("Pila vuota!\n");
            break;
        case ERRMEM:
            printf("errore di memoria!\n");
            break;
        case OK:
            printf("operazione eseguita con successo!\n");
            break;
        default:
            printf("errore sconosciuto!\n");
    }

}
```

```
int main()
{
    struct Pila c;
    init_stack(&c);

    int scelta = -1;
    while(scelta != 0)
    {
        printf("digitare:\n");
        printf("1) inserire in Pila\n");
        printf("2) leggere dalla Pila\n");
        printf("3) leggere e rimuovere dalla Pila\n");
        printf("4) visualizzare l'intera Pila\n");
        printf("0) uscire\n");
        scanf("%d", &scelta);

        int val, err;
        switch(scelta)
        {
            case 1:
                printf("Inserisci valore: ");
                scanf("%d", &val);
                err = push(&c, val);
                error_handler(err);
                break;

            case 2:
                err = read_from_stack(&c, &val);
                if (err == OK)
                    printf("il valore in testa era %d\n", val);
                error_handler(err);
                break;

            case 3:
                err = pop(&c, &val);
                if (err == OK)
                    printf("il valore in testa era %d\n", val);
                error_handler(err);
                break;
            case 4:
                print_stack(c);
                break;
            case 0:
                printf("ciao!\n");
                break;

            default:
                printf("scelta errata!\n");
        }

    }
}
```

```
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

()
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

Inserisci valore:

operazione eseguita con successo!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

Inserisci valore:

operazione eseguita con successo!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire
```

```
(  10, 20)

digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

il valore in testa era 20
operazione eseguita con successo!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

il valore in testa era 10
operazione eseguita con successo!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

Pila vuota!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

Pila vuota!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

Pila vuota!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

Inserisci valore:

operazione eseguita con successo!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

Inserisci valore:

operazione eseguita con successo!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

Inserisci valore:

operazione eseguita con successo!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

Inserisci valore:

operazione eseguita con successo!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire
```

```
il valore in testa era 60

operazione eseguita con successo!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

il valore in testa era 50
operazione eseguita con successo!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

il valore in testa era 40
operazione eseguita con successo!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

il valore in testa era 30
operazione eseguita con successo!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

Pila vuota!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

Pila vuota!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

Inserisci valore:

operazione eseguita con successo!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

Inserisci valore:

operazione eseguita con successo!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

il valore in testa era 80
operazione eseguita con successo!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

il valore in testa era 70
operazione eseguita con successo!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

Inserisci valore:
```

```
operazione eseguita con successo!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

il valore in testa era 90
operazione eseguita con successo!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

Pila vuota!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

Pila vuota!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

Pila vuota!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
4) visualizzare l'intera Pila
0) uscire

ciao!
```

- Dato un file di testo `nomi.txt` contenente, in ogni riga, una serie di nomi e cognomi nel seguente formato:

  ```
  ASCII
  nome1 cognome1
  nome2 cognome2
  nome3 cognome3
  .
  .
  .
  ```

Produrre un secondo file di testo `nomi_ordinati.txt` contenente i nomi e cognomi *nello stesso formato*, ma ordinati in ordine lessicografico a partire dal cognome.

Esempio:

`nomi.txt`:

```
ASCII
Vittorio Gassman
Michael Douglas
Kirk Douglas
Alessandro Gassman
Gigi Proietti
Carlotta Proietti
```

`nomi_ordinati.txt`:

```
ASCII
Kirk Douglas
Michael Douglas
Alessandro Gassman
Vittorio Gassman
Carlotta Proietti
Gigi Proietti
```

In [2]:
```c
#include <stdio.h>
#include <string.h>

#define MAXLEN 128
```

```
#define FILEIN "nomi.txt"
#define FILEOUT "nomi_ordinati.txt"
#define ERRFOPEN  -1
#define ERRMEM    -2
#define ERRFORMAT -3
#define OK         0
struct Persona
{
    char nome[MAXLEN];
    char cognome[MAXLEN];

};

void error_handler(int err)
{
    switch(err)
    {
        case ERRFOPEN:
            printf("Errore apertura file!\n");
            break;
        case ERRMEM:
            printf("errore di memoria!\n");
            break;
      case ERRFORMAT:
            printf("errore di formato nel file!\n");
            break;
        case OK:
            printf("operazione eseguita con successo!\n");
            break;
        default:
            printf("errore sconosciuto!\n");
    }

}


int get_cleaned_line(char str[], int maxlen, FILE* file_handler)
{
    int len = -1;
    if (fgets(str, maxlen, file_handler) != NULL)
    {
        len = 0;
        while(str[len] != '\0')
            len++;
        if (len > 0 && str[len-1] == '\n')
        {
            str[len-1] = '\0';
            len--;
        }
    }
    return len;
}

void print_v(struct Persona* v, int n)
{
    for(int i = 0; i < n; i++)
        printf("nome: \"%s\"; cognome: \"%s\"\n", v[i].nome, v[i].cognome);
}

int read_file(char filename[], struct Persona** v, int* n)
{
    FILE* fin  = fopen(filename, "r");
    if (fin == NULL)
        return ERRFOPEN;

    char line[MAXLEN];
    int linelen = 1;
    *n          = 0;
    while(linelen > 0)
    {
        linelen = get_cleaned_line(line, MAXLEN, fin);
        if(linelen > 0)
        {

            // inserisco il contenuto della linea in una nuova struct
            struct Persona nuova;
            int i = 0;
            // prendo il nome
            while(i < linelen && line[i]!=' ')
            {
                nuova.nome[i] = line[i];
                i++;
            }

            // se non c'è uno spazio vuol dire che il file è mal formato
            if(i >= linelen)
                return ERRFORMAT;

            // aggiungo il terminatore
```

```
            nuova.nome[i] = '\0';

            // prendo il cognome
            i++; // parto dal carattere successivo allo spazio
            int j = 0;
            while(i < linelen)
            {
                nuova.cognome[j] = line[i];
                j++;
                i++;
            }
            // aggiungo il terminatore
            nuova.cognome[j] = '\0';

            // se l'acquisizione da file è andata a buon fine, rialloco l'array
            struct Persona* tmp = realloc( *v, (*n + 1) * sizeof(struct Persona) );
            if (tmp == NULL)
                return ERRMEM;
            // se la riallocazione è andata a buon fine,
            // sovrascrivo il puntatore v col nuovo array
            *v = tmp;
            // inserisco il nuovo elemento in ultima posizione
            (*v)[*n] = nuova;
            (*n)++;
        }
    }


    fclose(fin);
    return OK;

}


void selection_sort(struct Persona* v, int n)
{

    for(int i = 0; i<n-1; i++)
    {
        int idx_min = i;
        for(int j = i+1; j<n; j++)
        {
            // se il cognome viene alfabeticamente prima
            if(strcmp(v[j].cognome, v[idx_min].cognome) < 0 )
                idx_min = j;
                // se i due cognomi sono ugali
            else if(strcmp(v[j].cognome, v[idx_min].cognome) == 0 )
            {
                // ma il  nome viene alfabetiamente prima
                if(strcmp(v[j].nome, v[idx_min].nome) < 0 )
                    idx_min = j;
            }

        }
        // pongo la persona idx_min nella posizione giusta
        struct Persona tmp = v[i];
        v[i]               = v[idx_min];
        v[idx_min]         = tmp;

    }

}


int write_file(char filename[], struct Persona* v, int n)
{
    FILE* fout = fopen(FILEOUT, "w");
    if(fout == NULL)
        return ERRFOPEN;
    for(int i = 0; i<n; i++)
        fprintf(fout,"%s %s\n", v[i].nome, v[i].cognome);
    fclose(fout);
    return OK;
}

int main()
{
    struct Persona* v = NULL;
    int n             = 0;


    int err =read_file(FILEIN, &v, &n);
    printf("lettura file...");
    error_handler(err);
    print_v(v,n);
    selection_sort(v,n);
    printf("dopo l'ordinamento:\n");
    print_v(v,n);
    err = write_file(FILEOUT, v, n);
```

```
    printf("scrittura file...");
    error_handler(err);

    return 0;
}
```

```
lettura file...operazione eseguita con successo!
nome: "Vittorio"; cognome: "Gassman"
nome: "Michael"; cognome: "Douglas"
nome: "Kirk"; cognome: "Douglas"
nome: "Alessandro"; cognome: "Gassman"
nome: "Gigi"; cognome: "Proietti"
nome: "Carlotta"; cognome: "Proietti"
dopo l'ordinamento:
nome: "Kirk"; cognome: "Douglas"
nome: "Michael"; cognome: "Douglas"
nome: "Alessandro"; cognome: "Gassman"
nome: "Vittorio"; cognome: "Gassman"
nome: "Carlotta"; cognome: "Proietti"
nome: "Gigi"; cognome: "Proietti"
scrittura file...operazione eseguita con successo!
```

In [62]:
```
// soluzione alternativa
#include <stdio.h>
#include <string.h>

#define MAXLEN 128
#define FILEIN "nomi.txt"
#define FILEOUT "nomi_ordinati.txt"
#define ERRFOPEN  -1
#define ERRMEM    -2
#define ERRFORMAT -3
#define OK         0
struct Persona
{
    char nome[MAXLEN];
    char cognome[MAXLEN];

};

void error_handler(int err)
{
    switch(err)
    {
        case ERRFOPEN:
            printf("Errore apertura file!\n");
            break;
        case ERRMEM:
            printf("errore di memoria!\n");
            break;
      case ERRFORMAT:
            printf("errore di formato nel file!\n");
            break;
        case OK:
            printf("operazione eseguita con successo!\n");
            break;
        default:
            printf("errore sconosciuto!\n");
    }

}


void print_v(struct Persona* v, int n)
{
    for(int i = 0; i < n; i++)
        printf("nome: \"%s\"; cognome: \"%s\"\n", v[i].nome, v[i].cognome);
}

int read_file(char filename[], struct Persona** v, int* n)
{
    FILE* fin  = fopen(filename, "r");
    if (fin == NULL)
        return ERRFOPEN;

    struct Persona nuova;
    while(fscanf(fin, "%s %s", nuova.nome, nuova.cognome) == 2)
    {
            struct Persona* tmp = realloc( *v, (*n +1) * sizeof(struct Persona) );
            if (tmp == NULL)
                return ERRMEM;
            // se la riallocazione è andata a buon fine, sovrascrivo il puntatore v col nuovo array
            *v = tmp;

            (*v)[*n] = nuova;
            (*n)++;
    }

    fclose(fin);
```

```
    return OK;

}


void selection_sort(struct Persona* v, int n)
{

    for(int i = 0; i<n-1; i++)
    {
        int idx_min = i;
        for(int j = i+1; j<n; j++)
        {
            if(strcmp(v[j].cognome, v[idx_min].cognome) < 0 )
                idx_min = j;
            else if(strcmp(v[j].cognome, v[idx_min].cognome) == 0 )
            {
                if(strcmp(v[j].nome, v[idx_min].nome) <0 )
                    idx_min = j;
            }

        }
        struct Persona tmp = v[i];
        v[i]               = v[idx_min];
        v[idx_min]         = tmp;

    }

}


int write_file(char filename[], struct Persona* v, int n)
{
    FILE* fout = fopen(FILEOUT, "w");
    if(fout == NULL)
        return ERRFOPEN;
    for(int i = 0; i<n; i++)
        fprintf(fout,"%s %s\n", v[i].nome, v[i].cognome);
    fclose(fout);
    return OK;
}

int main()
{
    struct Persona* v = NULL;
    int n             =0;


    int err =read_file(FILEIN, &v, &n);
    printf("lettura file...");
    error_handler(err);
    print_v(v,n);
    selection_sort(v,n);
    printf("dopo l'ordinamento:\n");
    print_v(v,n);
    err = write_file(FILEOUT, v, n);
    printf("scrittura file...");
    error_handler(err);

    return 0;
}
```

```
lettura file...operazione eseguita con successo!
nome: "Vittorio"; cognome: "Gassman"
nome: "Michael"; cognome: "Douglas"
nome: "Kirk"; cognome: "Douglas"
nome: "Alessandro"; cognome: "Gassman"
nome: "Gigi"; cognome: "Proietti"
nome: "Carlotta"; cognome: "Proietti"
dopo l'ordinamento:
nome: "Kirk"; cognome: "Douglas"
nome: "Michael"; cognome: "Douglas"
nome: "Alessandro"; cognome: "Gassman"
nome: "Vittorio"; cognome: "Gassman"
nome: "Carlotta"; cognome: "Proietti"
nome: "Gigi"; cognome: "Proietti"
scrittura file...operazione eseguita con successo!
```

In [ ]:

Laboratorio di Programmazione Gr. 3 (N-Z)

Corso di Laurea in Informatica

Università degli Studi di Napoli Federico II

A.A. 2021/22

A. Apicella

In [ ]:
```
// REMINDER:
// inserimento di un Nodo in fondo alla lista
void insert(struct Nodo** pp_head, int val)
{
    // 1. genero il nuovo Nodo
    struct Nodo* nuovo;
    nuovo = malloc(sizeof(struct Nodo));
    nuovo->valore          = val;
    nuovo->prossimo        = NULL;

    // 2. scorro la lista fino a raggiungere l'ultimo Nodo
    struct Nodo* in_esame   = *pp_head;
    struct Nodo* last       = NULL;

    // alla fine del ciclo, last punterà all'ultimo Nodo della lista
    // se la lista è vuota, last conterrà NULL
    while(in_esame!= NULL)
    {
        last       = in_esame;
        in_esame   = in_esame->prossimo;
    }

    if( last == NULL )
        *pp_head = nuovo; // se la lista  è vuota,
                          // inserisco il nuovo Nodo come primo Nodo
    else
        last->prossimo = nuovo;
}
```

In [ ]:
```
void insert(struct Nodo** pp_head, int val)
{
    // 1. genero il nuovo Nodo
    struct Nodo* nuovo;
    nuovo = malloc(sizeof(struct Nodo));
    nuovo->valore          = val;
    nuovo->prossimo        = NULL;

    // 2. controllo se la lista è vuota. In tal caso, l'Nodo da inserire è il primo
    if( *pp_head == NULL )
    {
        *pp_head = nuovo;
        return;
    }


    // 3. scorro la lista fino a raggiungere l'ultimo Nodo
    struct Nodo* in_esame   = *pp_head;
    struct Nodo* last       = NULL;

    // alla fine del ciclo, in_esame punterà all'ultimo Nodo della lista
    while(in_esame->prossimo!= NULL)
    {
        in_esame   = in_esame->prossimo;
    }
     in_esame->prossimo = nuovo;
}
```

- Implementare le seguenti funzioni per una lista concatenata di interi:
- `remove_at(...)` : data in input una lista concatenata ed un indice di posizione `idx` , rimuovere dalla lista l'Nodo in posizione `idx` . Si consideri il primo Nodo avente posizione `0` . Tale funzione dovrà restituire:
  - `NULL` , se l'elemento in posizione `idx` non esiste
  - l'indirizzo dell'elemento rimosso dalla lista
- `insert_at(...)` : data in input una lista concatenata, un valore `val` ed un indice di posizione `idx` , inserisce l'Nodo `val` in posizione `idx` . Si consideri il primo Nodo avente posizione `0` . Se `idx` va oltre la lunghezza effettiva della lista, l'Nodo dovrà essere inserito alla fine della lista

---

In [ ]:

```c
#include <stdio.h>

struct Nodo {
    int valore;
    struct Nodo* prossimo;

};

void print(struct Nodo* p_head)
{
    printf("#");
    while(p_head!=NULL)
    {
        printf("->{%d}", p_head->valore);
        p_head = p_head->prossimo;
    }
    printf("\n");
}

void deallocate(struct Nodo* p_head)
{
    while(p_head != NULL)
    {
        struct Nodo* prox = p_head->prossimo;
        free(p_head);
        p_head = prox;
    }
}


/*
 inserisce il valore `val` in posizione `idx`.
 Si consideri il primo Nodo avente posizione `0`.
 Se `idx` va oltre la lunghezza effettiva della lista,
 l'Nodo dovrà essere inserito alla fine della lista

*/

void insert_at(struct Nodo** pp_head, int val, int idx)
{
    // 1. genero il nuovo Nodo
    struct Nodo* nuovo;
    nuovo = malloc(sizeof(struct Nodo));
    nuovo->valore          = val;
    nuovo->prossimo        = NULL;

    // 2. scorro la lista fino a raggiungere la posizione idx
    struct Nodo* in_esame   = *pp_head;
    struct Nodo* pred       = NULL;
    int i                   = 0;

    // se la lista è vuota, pred conterrà NULL
    // altrimenti, pred punterà alla posizione in cui inserire nuovo
    while(in_esame != NULL && i < idx)
    {
        pred       = in_esame;
        in_esame   = in_esame->prossimo;
        i++;
    }

    if( pred == NULL )
    {
        // collego il nuovo Nodo al resto della lista
        nuovo->prossimo = *pp_head;
        *pp_head        = nuovo; // se la lista  è vuota o idx == 0,
                                 // inserisco il nuovo Nodo come primo Nodo
    }
    else
    {
        // collego il nuovo Nodo al resto della lista
        nuovo->prossimo = pred->prossimo;
        pred->prossimo  = nuovo;
    }

}
/*
`remove_at(...)`: data in input una lista concatenata ed un indice di posizione `idx`,
rimuovere dalla lista l'Nodo in posizione `idx`.
Si consideri il primo Nodo avente posizione `0`. Tale funzione dovrà restituire:
   - `NULL`, se l'Nodo in posizione `idx` non esiste
   - l'indirizzo dell'Nodo rimosso dalla lista

*/
struct Nodo* remove_at(struct Nodo** pp_head, int idx)
{
    // 1. scorro la lista fino a raggiungere la posizione idx
    struct Nodo* in_esame   = *pp_head;
    struct Nodo* pred       = NULL;
    int i                   = 0;
```

```
    if (in_esame == NULL) // se la lista è vuota
        return NULL;

    while(in_esame != NULL && i < idx)
    {
        pred       = in_esame;
        in_esame   = in_esame->prossimo;
        i++;
    }

    if( in_esame == NULL ) // se non esiste alcun Nodo in posizione idx
        return NULL;

    struct Nodo* ret;

    if (pred == NULL) // <=> if( idx == 0 ), ossia bisogna rimuovere il primo
    {
        ret = *pp_head; // <=> ret = in_esame;
        *pp_head = (*pp_head)->prossimo;
    }
    else
    {
        ret = in_esame;
        pred->prossimo = in_esame->prossimo;

    }

    return ret;

}


int main()
{
    struct Nodo* p_head = NULL;

    int scelta = -1;
    while(scelta != 0)
    {
        printf("1) inserisci\n2) stampa\n3) cancella tutto\n4)rimuovi\n0) esci\n");
        scanf("%d", &scelta);

        int val;
        switch(scelta)
        {
            case 1:
                printf("valore da inserire: ");
                scanf("%d", &val);
                int idx;
                printf("posizione: ");
                scanf("%d", &idx);
                insert_at(&p_head, val, idx);
                break;
            case 2:
                print(p_head);
                break;
            case 3:
                deallocate(p_head);
                p_head = NULL;
                break;
            case 4:

                printf("posizione: ");
                scanf("%d", &idx);
                struct Nodo* rimosso = remove_at(&p_head, idx);
                if (rimosso == NULL)
                    printf("posizione non valida o lista vuota\n");
                else
                {
                    printf("rimosso Nodo con valore %d.\n",rimosso->valore);
                    free(rimosso);
                }
                break;
            case 0:
                deallocate(p_head);
                p_head = NULL;
                break;
            default:
                printf("scelta sbagliata! Ripetere\n");

        }
    }

    return 0;
}
```

```
1) inserisci
2) stampa
3) cancella tutto
4)rimuovi
0) esci

#
1) inserisci
2) stampa
3) cancella tutto
4)rimuovi
0) esci

valore da inserire:

posizione:

1) inserisci
2) stampa
3) cancella tutto
4)rimuovi
0) esci

#->{10}
1) inserisci
2) stampa
3) cancella tutto
4)rimuovi
0) esci

valore da inserire:

posizione:

1) inserisci
2) stampa
3) cancella tutto
4)rimuovi
0) esci

#->{10}->{20}
1) inserisci
2) stampa
3) cancella tutto
4)rimuovi
0) esci

1) inserisci
2) stampa
3) cancella tutto
4)rimuovi
0) esci

#
1) inserisci
2) stampa
3) cancella tutto
4)rimuovi
0) esci
```

- Implementare una struttura dati con politica di accesso Coda con relative funzioni di accesso utilizzando come struttura una lista concatenata.

In [9]:
```
#include <stdio.h>
#include <time.h>
#define FULL -1
#define EMPTY -2
#define OK 0

struct Nodo {
    int valore;
    struct Nodo* prossimo;

};


struct Coda
{
    struct Nodo* p_head;
};


// funzioni gestione coda
void init_queue (struct Coda* c)
{
    c->p_head = NULL;
}
```

```
int is_empty(struct Coda* c)
{
    if(c->p_head == NULL)
    {
        return 1;
    }
    return 0;
}

int is_full(struct Coda* c)
{

    return 0;
}


int size(struct Coda* c)
{
    int n_items = 0;
    struct Nodo* cursore = c->p_head;
    while(cursore!= NULL)
    {
        n_items++;
    }
    return n_items;
}


int enqueue(struct Coda* c, int val)
{
    if( is_full(c) == 1 )
        return FULL;
    struct Nodo* nuovo = malloc(sizeof(struct Nodo));
    nuovo->valore   = val;
    nuovo->prossimo = NULL;

    if (c->p_head == NULL)
    {
        c->p_head = nuovo;
    }
    else
    {
        struct Nodo* cursore = c->p_head;
        while(cursore->prossimo != NULL)
            cursore = cursore->prossimo;
        cursore->prossimo = nuovo;
    }

    return OK;

}

// restituisce il valore dell'elemento in testa
int read_from_queue(struct Coda* c, int* p_val)
{
    if(is_empty(c) == 1)
        return EMPTY;
    *p_val    = c->p_head->valore;

    return OK;
}

// elimina Nodo in testa e lo restituisce
int dequeue(struct Coda* c, int* p_val)
{

    if(is_empty(c) == 1)
        return EMPTY;
    read_from_queue(c, p_val);
    struct Nodo* to_free = c->p_head;
    c->p_head = c->p_head->prossimo;
    free(to_free);
    return OK;

}


void print_queue(struct Coda* c)
{
    struct Nodo* cursore = c->p_head;
    while(cursore!= NULL)
    {
        printf("{%d}->", cursore->valore);
        cursore = cursore->prossimo;
    }
    printf("n.items: %d\n", size(c));
    printf("\n");
}
```

```
int main()
{
    struct Coda c;
    init_queue(&c);

    int scelta = -1;
    while(scelta != 0)
    {
        printf("digitare:\n");
        printf("1) inserire in coda\n");
        printf("2) leggere dalla coda\n");
        printf("3) leggere e rimuovere dalla coda\n");
        printf("0) uscire\n");
        scanf("%d", &scelta);

        int val, err;
        switch(scelta)
        {
            case 1:
                printf("Inserisci valore: ");
                scanf("%d", &val);
                err = enqueue(&c, val);
                if(err == OK)
                    printf("valore inserito\n");
                else
                    printf("coda piena!\n");
                break;

            case 2:
                err = read_from_queue(&c, &val);

                if(err == OK)
                    printf("il valore in testa è %d.\n", val);
                else
                    printf("coda vuota!\n");
                break;

            case 3:
                err = dequeue(&c, &val);
                if(err == OK)
                    printf("il valore in testa è %d ; tale valore è stato rimosso.\n", val);
                else
                    printf("coda vuota!\n");
                break;

            case 0:
                printf("ciao!\n");
                break;

            default:
                printf("scelta errata!\n");
        }
    }
}
```

```
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
0) uscire

coda vuota!
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
0) uscire

Inserisci valore:

valore inserito
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
0) uscire

Inserisci valore:

valore inserito
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
0) uscire

Inserisci valore:
```

```
valore inserito
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
0) uscire

il valore in testa è 10.
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
0) uscire

il valore in testa è 10.
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
0) uscire

il valore in testa è 10.
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
0) uscire

il valore in testa è 10.
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
0) uscire

il valore in testa è 10 ; tale valore è stato rimosso.
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
0) uscire

il valore in testa è 20 ; tale valore è stato rimosso.
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
0) uscire

il valore in testa è 30 ; tale valore è stato rimosso.
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
0) uscire

coda vuota!
digitare:
1) inserire in coda
2) leggere dalla coda
3) leggere e rimuovere dalla coda
0) uscire

ciao!
```

- Implementare una struttura dati con politica di accesso Pila con relative funzioni di accesso utilizzando come struttura una lista concatenata.

In [12]:
```
#include <stdio.h>
#include <time.h>
#define FULL -1
#define EMPTY -2
#define OK 0

struct Nodo {
    int valore;
    struct Nodo* prossimo;
};


struct Pila
{
    struct Nodo* p_head;
};


// funzioni gestione Pila
void init_stack (struct Pila* c)
{
```

```
    c->p_head = NULL;
}


int is_empty(struct Pila* c)
{
    if(c->p_head == NULL)
    {
        return 1;
    }
    return 0;
}

int is_full(struct Pila* c)
{

    return 0;
}


int size(struct Pila* c)
{
    int n_items = 0;
    struct Nodo* cursore = c->p_head;
    while(cursore!= NULL)
    {
        n_items++;
    }
    return n_items;
}


int push(struct Pila* c, int Nodo)
{
    if( is_full(c) == 1 )
        return FULL;
    struct Nodo* nuovo = malloc(sizeof(struct Nodo));
    nuovo->valore   = Nodo;
    nuovo->prossimo = NULL;

    if (c->p_head == NULL)
    {
        c->p_head = nuovo;
    }
    else
    {
        nuovo->prossimo = c->p_head;
        c->p_head       = nuovo;
    }

    return OK;

}

// restituisce il valore in testa
int read_from_stack(struct Pila* c, int* p_val)
{
    if(is_empty(c) == 1)
        return EMPTY;
    *p_val    = c->p_head->valore;

    return OK;
}

// elimina il valore in testa e lo restituisce
int pop(struct Pila* c, int* p_val)
{

    if(is_empty(c) == 1)
        return EMPTY;
    read_from_stack(c, p_val);
    struct Nodo* to_free = c->p_head;
    c->p_head = c->p_head->prossimo;
    free(to_free);
    return OK;

}


void print_stack(struct Pila* c)
{
    struct Nodo* cursore = c->p_head;
    while(cursore!= NULL)
    {
        printf("{%d}->", cursore->valore);
        cursore = cursore->prossimo;
    }
    printf("n.items: %d\n", size(c));
    printf("\n");
```

```
}


int main()
{
    struct Pila c;
    init_stack(&c);

    int scelta = -1;
    while(scelta != 0)
    {
        printf("digitare:\n");
        printf("1) inserire in Pila\n");
        printf("2) leggere dalla Pila\n");
        printf("3) leggere e rimuovere dalla Pila\n");
        printf("0) uscire\n");
        scanf("%d", &scelta);

        int val, err;
        switch(scelta)
        {
            case 1:
                printf("Inserisci valore: ");
                scanf("%d", &val);
                err = push(&c, val);
                if(err == OK)
                    printf("valore inserito\n");
                else
                    printf("Pila piena!\n");
                break;

            case 2:
                err = read_from_stack(&c, &val);

                if(err == OK)
                    printf("il valore in testa è %d.\n", val);
                else
                    printf("Pila vuota!\n");
                break;

            case 3:
                err = pop(&c, &val);
                if(err == OK)
                    printf("il valore in testa è %d ; tale valore è stato rimosso.\n", val);
                else
                    printf("Pila vuota!\n");
                break;

            case 0:
                printf("ciao!\n");
                break;

            default:
                printf("scelta errata!\n");
        }
    }
}
```

```
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
0) uscire

Pila vuota!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
0) uscire

Inserisci valore:

valore inserito
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
0) uscire

Inserisci valore:

valore inserito
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
0) uscire

Inserisci valore:
```

```
valore inserito
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
0) uscire

il valore in testa è 30.
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
0) uscire

il valore in testa è 30.
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
0) uscire

il valore in testa è 30.
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
0) uscire

il valore in testa è 30 ; tale valore è stato rimosso.
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
0) uscire

il valore in testa è 20 ; tale valore è stato rimosso.
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
0) uscire

il valore in testa è 10 ; tale valore è stato rimosso.
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
0) uscire

Pila vuota!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
0) uscire

Inserisci valore:

valore inserito
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
0) uscire

Inserisci valore:

valore inserito
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
0) uscire

il valore in testa è 50 ; tale valore è stato rimosso.
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
0) uscire

il valore in testa è 40 ; tale valore è stato rimosso.
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
0) uscire
```

```
Pila vuota!
digitare:
1) inserire in Pila
2) leggere dalla Pila
3) leggere e rimuovere dalla Pila
0) uscire

ciao!
```

- implementare una funzione `insert_sorted(...)` che, data in input una lista concatenata di interi ordinata in maniera crescente ed un valore `val`, inserisca `val` in lista in modo che la lista risultante sia ancora ordinata.

In [2]:
```
#include <stdio.h>

struct Nodo {
    int valore;
    struct Nodo* prossimo;

};

void print(struct Nodo* p_head)
{
    printf("#");
    while(p_head!=NULL)
    {
        printf("->{%d}", p_head->valore);
        p_head = p_head->prossimo;
    }
    printf("\n");
}

void deallocate(struct Nodo* p_head)
{
    while(p_head != NULL)
    {
        struct Nodo* prox = p_head->prossimo;
        free(p_head);
        p_head = prox;
    }
}



void insert_sorted(struct Nodo** pp_head, int val)
{
    // 1. genero il nuovo Nodo
    struct Nodo* nuovo;
    nuovo = malloc(sizeof(struct Nodo));
    nuovo->valore          = val;
    nuovo->prossimo        = NULL;

    // 2. se è il primo valore che sto inserendo
    if(*pp_head == NULL)
    {
        *pp_head = nuovo;
        return;
    }

    // 3. scorro la lista fino a raggiungere la posizione giusta
    struct Nodo* in_esame   = *pp_head;
    struct Nodo* pred       = NULL;

    // se la lista è vuota, pred conterrà NULL
    // altrimenti, pred punterà alla posizione in cui inserire nuovo
    while(in_esame!= NULL && in_esame->valore < val)
    {
        pred       = in_esame;
        in_esame   = in_esame->prossimo;
    }

    if( pred == NULL )
    {
        // collego il nuovo Nodo al resto della lista
        nuovo->prossimo = *pp_head;
        *pp_head        = nuovo; // se la lista  è vuota o idx == 0,
                                 // inserisco il nuovo Nodo come primo Nodo
    }
    else
    {
        // collego il nuovo Nodo al resto della lista
        nuovo->prossimo = pred->prossimo;
        pred->prossimo  = nuovo;
    }

}
```

```
int main()
{
    struct Nodo* p_head = NULL;

    int scelta = -1;
    while(scelta != 0)
    {
        printf("1) inserisci\n2) stampa\n3) cancella tutto\n0) esci\n");
        scanf("%d", &scelta);

        int val;
        switch(scelta)
        {
            case 1:
                printf("valore da inserire: ");
                scanf("%d", &val);
                insert_sorted(&p_head, val);
                break;
            case 2:
                print(p_head);
                break;
            case 3:
                deallocate(p_head);
                p_head = NULL;
                break;
            case 0:
                deallocate(p_head);
                p_head = NULL;
                break;
            default:
                printf("scelta sbagliata! Ripetere\n");

        }
    }

    return 0;
}
```

```
1) inserisci
2) stampa
3) cancella tutto
0) esci

valore da inserire:

1) inserisci
2) stampa
3) cancella tutto
0) esci

valore da inserire:

1) inserisci
2) stampa
3) cancella tutto
0) esci

#->{10}->{15}
1) inserisci
2) stampa
3) cancella tutto
0) esci

valore da inserire:

1) inserisci
2) stampa
3) cancella tutto
0) esci

#->{6}->{10}->{15}
1) inserisci
2) stampa
3) cancella tutto
0) esci

valore da inserire:

1) inserisci
2) stampa
3) cancella tutto
0) esci

#->{6}->{10}->{11}->{15}
1) inserisci
2) stampa
3) cancella tutto
0) esci

valore da inserire:
```

```
1) inserisci
2) stampa
3) cancella tutto
0) esci

#->{6}->{10}->{11}->{15}->{30}
1) inserisci
2) stampa
3) cancella tutto
0) esci

valore da inserire:

1) inserisci
2) stampa
3) cancella tutto
0) esci

#->{6}->{10}->{10}->{11}->{15}->{30}
1) inserisci
2) stampa
3) cancella tutto
0) esci

valore da inserire:

1) inserisci
2) stampa
3) cancella tutto
0) esci

#->{5}->{6}->{10}->{10}->{11}->{15}->{30}
1) inserisci
2) stampa
3) cancella tutto
0) esci

1) inserisci
2) stampa
3) cancella tutto
0) esci

#
1) inserisci
2) stampa
3) cancella tutto
0) esci
```

In [ ]:

Laboratorio di Programmazione Gr. 3 (N-Z)

Corso di Laurea in Informatica

Università degli Studi di Napoli Federico II

A.A. 2021/22

A. Apicella

- Definire una struttura dati contenente interi avente una politica di accesso del tipo *minimum first out*, in cui il valore più piccolo contenuto è sempre il primo ad uscire. Provarla con un apposito `main(...)`.

In [7]:
```
// soluzione 1: inserisco ed ordino

#include <stdio.h>
#define N_CODA 4
#define FULL -1
#define EMPTY -2
#define OK 0

struct Mfo
{
    float deposito[N_CODA];
    int n_occupati;

};

float idx_min_array(float v[], int n, int start)
{
    int idx_min = start;
    float min   = v[start];
    // Loop interno
    for (int i=start+1; i<n; i++)
        if (v[i] < min)
        {
            min     = v[i];
            idx_min = i;
        }
    return idx_min;
}


void selection_sort(float v[], int n)
{
    int idx_min;
    // Loop esterno
    for(int i=0; i<n-1; i++)
    {
        idx_min    = idx_min_array(v, n, i);
        float tmp  = v[i];
        v[i]       = v[idx_min];
        v[idx_min] = tmp;
    }

}


void init_mfo(struct Mfo* c)
{
    c->n_occupati = 0; // numero di elementi presenti
}

int is_mfo_empty(struct Mfo* c)
{
    if(c->n_occupati == 0)
    {
        return EMPTY;
    }
    return OK;
}

int is_mfo_full(struct Mfo* c)
{
    if (c->n_occupati >= N_CODA)
    {
        return FULL;
    }
```

```
    return OK;
}


int insert_in_mfo(struct Mfo* c, int elemento)
{

    if(is_mfo_full(c) == FULL)
        return FULL;
    c->deposito[c->n_occupati] = elemento;
    c->n_occupati++;
    // ordina
    selection_sort(c->deposito, c->n_occupati);
    return OK;

}

int read_from_mfo(struct Mfo* c, int* p_val)
{
    if(is_mfo_empty(c) == EMPTY)
        return EMPTY;

    *p_val =  c->deposito[0];
    return OK;
}

int get_from_mfo(struct Mfo* c, int* p_val)
{

    if(is_mfo_empty(c) == EMPTY)
        return EMPTY;
    read_from_mfo(c, p_val);
    for(int i=0; i < c->n_occupati-1; i++)
        c->deposito[i] = c->deposito[i+1];

    c->n_occupati--;
    return OK;

}

int main()
{
    struct Mfo c;
    init_mfo(&c);

    int scelta = -1;
    while(scelta != 0)
    {
        printf("digitare:\n");
        printf("1) inserire in mfo\n");
        printf("2) leggere da mfo\n");
        printf("3) leggere e rimuovere da mfo\n");
        printf("0) uscire\n");
        scanf("%d", &scelta);

        int val, err;
        switch(scelta)
        {
            case 1:
                printf("Inserisci valore: ");
                scanf("%d", &val);
                err = insert_in_mfo(&c, val);
                if(err == OK)
                    printf("valore inserito\n");
                else
                    printf("coda piena!\n");
                break;

            case 2:
                err = read_from_mfo(&c, &val);

                if(err == OK)
                    printf("il valore in testa è %d.\n", val);
                else
                    printf("coda vuota!\n");
                break;

            case 3:
                err = get_from_mfo(&c, &val);
                if(err == OK)
                    printf("il valore in testa è %d ; tale valore è stato rimosso.\n", val);
                else
                    printf("coda vuota!\n");
                break;

            case 0:
                printf("ciao!\n");
                break;
```

```
            default:
                printf("scelta errata!\n");
        }

    }

    return 0;
}
```

```
digitare:
1) inserire in mfo
2) leggere da mfo
3) leggere e rimuovere da mfo
0) uscire

Inserisci valore:

valore inserito
digitare:
1) inserire in mfo
2) leggere da mfo
3) leggere e rimuovere da mfo
0) uscire

Inserisci valore:

valore inserito
digitare:
1) inserire in mfo
2) leggere da mfo
3) leggere e rimuovere da mfo
0) uscire

Inserisci valore:

valore inserito
digitare:
1) inserire in mfo
2) leggere da mfo
3) leggere e rimuovere da mfo
0) uscire

Inserisci valore:

valore inserito
digitare:
1) inserire in mfo
2) leggere da mfo
3) leggere e rimuovere da mfo
0) uscire

il valore in testa è 1 ; tale valore è stato rimosso.
digitare:
1) inserire in mfo
2) leggere da mfo
3) leggere e rimuovere da mfo
0) uscire

il valore in testa è 2 ; tale valore è stato rimosso.
digitare:
1) inserire in mfo
2) leggere da mfo
3) leggere e rimuovere da mfo
0) uscire

il valore in testa è 4 ; tale valore è stato rimosso.
digitare:
1) inserire in mfo
2) leggere da mfo
3) leggere e rimuovere da mfo
0) uscire

il valore in testa è 5 ; tale valore è stato rimosso.
digitare:
1) inserire in mfo
2) leggere da mfo
3) leggere e rimuovere da mfo
0) uscire

coda vuota!
digitare:
1) inserire in mfo
2) leggere da mfo
3) leggere e rimuovere da mfo
0) uscire

coda vuota!
digitare:
1) inserire in mfo
2) leggere da mfo
3) leggere e rimuovere da mfo
0) uscire
```

```
coda vuota!
digitare:
1) inserire in mfo
2) leggere da mfo
3) leggere e rimuovere da mfo
0) uscire

ciao!
```

In [49]:

```
// soluzione 2: inserisco gli elementi in modo già ordinato

#include <stdio.h>
#define N_CODA 4
#define FULL -1
#define EMPTY -2
#define OK 0

struct Mfo
{
    float deposito[N_CODA];
    int n_occupati;

};


void init_mfo(struct Mfo* c)
{
    c->n_occupati = 0; // numero di elementi presenti
}

int is_mfo_empty(struct Mfo* c)
{
    if(c->n_occupati == 0)
    {
        return EMPTY;
    }
    return OK;
}

int is_mfo_full(struct Mfo* c)
{
    if (c->n_occupati >= N_CODA)
    {
        return FULL;
    }
    return OK;
}


int insert_in_mfo(struct Mfo* c, int elemento)
{

    if(is_mfo_full(c) == FULL)
        return FULL;

    // cerco la posizione corretta
    int i = 0;
    while (i < c->n_occupati && c->deposito[i] < elemento)
        i++;

    for(int j=c-> n_occupati; j >  i; j--)
       c->deposito[j] = c->deposito[j-1];

    c->deposito[i] = elemento;
    c->n_occupati++;
    return OK;

}

int read_from_mfo(struct Mfo* c, int* p_val)
{
    if(is_mfo_empty(c) == EMPTY)
        return EMPTY;

    *p_val =  c->deposito[0];
    return OK;
}

int get_from_mfo(struct Mfo* c, int* p_val)
{

    if(is_mfo_empty(c) == EMPTY)
        return EMPTY;
    read_from_mfo(c, p_val);
    for(int i=0; i < c->n_occupati-1; i++)
        c->deposito[i] = c->deposito[i+1];

    c->n_occupati--;
    return OK;
```

```
}

int main()
{
    struct Mfo c;
    init_mfo(&c);

    int scelta = -1;
    while(scelta != 0)
    {
        printf("digitare:\n");
        printf("1) inserire in mfo\n");
        printf("2) leggere da mfo\n");
        printf("3) leggere e rimuovere da mfo\n");
        printf("0) uscire\n");
        scanf("%d", &scelta);

        int val, err;
        switch(scelta)
        {
            case 1:
                printf("Inserisci valore: ");
                scanf("%d", &val);
                err = insert_in_mfo(&c, val);
                if(err == OK)
                    printf("valore inserito\n");
                else
                    printf("coda piena!\n");
                break;

            case 2:
                err = read_from_mfo(&c, &val);

                if(err == OK)
                    printf("il valore in testa è %d.\n", val);
                else
                    printf("coda vuota!\n");
                break;

            case 3:
                err = get_from_mfo(&c, &val);
                if(err == OK)
                    printf("il valore in testa è %d ; tale valore è stato rimosso.\n", val);
                else
                    printf("coda vuota!\n");
                break;

            case 0:
                printf("ciao!\n");
                break;

            default:
                printf("scelta errata!\n");
        }

    }

    return 0;
}
```

```
digitare:
1) inserire in mfo
2) leggere da mfo
3) leggere e rimuovere da mfo
0) uscire

Inserisci valore:

valore inserito
digitare:
1) inserire in mfo
2) leggere da mfo
3) leggere e rimuovere da mfo
0) uscire

il valore in testa è 5 ; tale valore è stato rimosso.
digitare:
1) inserire in mfo
2) leggere da mfo
3) leggere e rimuovere da mfo
0) uscire

coda vuota!
digitare:
1) inserire in mfo
2) leggere da mfo
3) leggere e rimuovere da mfo
0) uscire
```

```
Inserisci valore:

valore inserito
digitare:
1) inserire in mfo
2) leggere da mfo
3) leggere e rimuovere da mfo
0) uscire

Inserisci valore:

valore inserito
digitare:
1) inserire in mfo
2) leggere da mfo
3) leggere e rimuovere da mfo
0) uscire

Inserisci valore:

valore inserito
digitare:
1) inserire in mfo
2) leggere da mfo
3) leggere e rimuovere da mfo
0) uscire

il valore in testa è 3 ; tale valore è stato rimosso.
digitare:
1) inserire in mfo
2) leggere da mfo
3) leggere e rimuovere da mfo
0) uscire

il valore in testa è 5 ; tale valore è stato rimosso.
digitare:
1) inserire in mfo
2) leggere da mfo
3) leggere e rimuovere da mfo
0) uscire

il valore in testa è 6 ; tale valore è stato rimosso.
digitare:
1) inserire in mfo
2) leggere da mfo
3) leggere e rimuovere da mfo
0) uscire

coda vuota!
digitare:
1) inserire in mfo
2) leggere da mfo
3) leggere e rimuovere da mfo
0) uscire

ciao!
```

- Data una stringa s, verificare se la stringa è palindroma. Gli elementi di s potranno essere letti/scritti solo ed esclusivamente dal primo all'ultimo. Non possono essere utilizzate strutture di supporto diverse da Pile e/o Code.

In [37]:
```
#include <stdio.h>
#define N_PILA 256

#define FULL -1
#define EMPTY -2
#define OK 0
struct Pila
{
    char deposito[N_PILA];
    int n_occupati;

};

void init_stack (struct Pila* p)
{
    p->n_occupati = 0;
}

int is_empty(struct Pila* p)
{
    if(p->n_occupati <= 0)
        return 1;
    return 0;
}


int is_full(struct Pila* pila)
{
    if(pila->n_occupati >= N_PILA)
```

```
        return 1;
    return 0;
}

int pop(struct Pila* p, char* val)
{
    if(is_empty(p)== 1)
        return EMPTY;
    *val = p->deposito[p->n_occupati-1];
    p->n_occupati--;
    return OK;
}


int push(struct Pila* p, char val)
{
    if(is_full(p) == 1)
        return FULL;
    p->deposito[p->n_occupati] = val;
    p->n_occupati++;
    return OK;
}


int main()
{
    struct Pila p;
    init_stack(&p);

    char s[N_PILA];

    printf("stringa: ");
    scanf("%s", s);

    int i = 0;
    while(s[i] !='\0')
    {
        push(&p,s[i]);
        i++;
    }

    i       = 0;
    int pal = 1;
    while(s[i] !='\0' && pal == 1)
    {
        char c;
        pop(&p, &c);
        if(s[i] != c)
            pal = 0;
        i++;
    }

    if(pal == 1)
    {
        printf("la stringa %s è palindroma\n", s);

    }
    else
    {
        printf("la stringa %s non è palindroma\n", s);

    }

    return 0;
}
```

```
stringa:
```

```
la stringa ciao non è palindroma
```

- dato un numero intero positivo, stamparne la codifica binaria. Non possono essere utilizzate strutture di supporto diverse da Pile e/o Code.

In [51]:
```
#include <stdio.h>
#define N_PILA 100

#define FULL -1
#define EMPTY -2
#define OK 0
struct Pila
{
    int deposito[N_PILA];
    int n_occupati;

};

void init_stack (struct Pila* p)
```

```
{
    p->n_occupati = 0;
}

int is_empty(struct Pila* p)
{
    if(p->n_occupati <= 0)
        return 1;
    return 0;
}


int is_full(struct Pila* pila)
{
    if(pila->n_occupati >= N_PILA)
        return 1;
    return 0;
}

int pop(struct Pila* p, int* val)
{
    if(is_empty(p)== 1)
        return EMPTY;
    *val = p->deposito[p->n_occupati-1];
    p->n_occupati--;
    return OK;
}


int push(struct Pila* p, int val)
{
    if(is_full(p) == 1)
        return FULL;
    p->deposito[p->n_occupati] = val;
    p->n_occupati++;
    return OK;
}


int main()
{
    struct Pila p;
    init_stack(&p);

    int numero;
    printf("numero da convertire: ");
    scanf("%d", &numero);


    do
    {
        push(&p, numero%2);
        numero = numero/2;
    }
    while(numero != 0);
    printf("versione binaria: ");
    while( is_empty(&p) != 1)
    {
        pop(&p, &numero);
        printf("%d", numero);
    }
    printf("\n");
    return 0;
}
```

```
numero da convertire:
```

```
versione binaria: 10001110
```

- Si vuole implementare un gioco che simuli il comportamento di un uomo in fila alle casse di un supermercato. Date due code C_1, C_2 ed una persona u, tale persona dovrà scegliere in quale coda mettersi durante lo scorrere della coda. Scopo del gioco è quello di essere servito (ossia di arrivare ad una delle due casse) prima che l'altra coda si svuoti.

Dettagli:

Ogni coda può avere al massimo 20 persone.

Ogni persona porta con sé nel carrello al massimo 20 prodotti.

All'inizio del gioco, l'uomo u vede *quante* persone ci sono nelle due code (ma non quanti prodotti siano presenti nel carrello di ogni persona in coda). Sceglie quindi in quale coda posizionarsi. (La 1 o la 2). Il numero di persone per ogni coda e di prodotti per ogni persona è deciso in maniera casuale.

Il gioco quindi prosegue a turni.

Come all'inizio, ad ogni turno, u è a conoscenza di quante persone sono in attesa in ogni coda, ma non di quanti prodotti hanno nel carrello.

Ad ogni turno, u potrà scegliere se:

1. rimanere nella coda in cui si trova
2. cambiare coda

Contemporaneamente, ogni cassiere vede il numero di prodotti della persona in testa della rispettiva file, "consumando" (ossia facendo pagare) un singolo prodotto ad ogni turno. Al termine del numero di prodotti, la cassa servirà la persona successiva. Il altri termini, ogni coda è "ferma" per un numero di turni uguale al numero di prodotti del cliente in testa.

Esempio:

In C_1 ci sono in attesa 3 persone, in C_2 ci sono in attesa 5 persone. La persona in testa a C_1 ha 3 prodotti, mentre in testa a C_2 c'è una persona con 7 prodotti. C_1 quindi procederà alla persona successiva dopo 3 turni e C_2 dopo 7 turni. In altri termini, dopo 3 turni, C_1 avrà concluso col cliente in testa passando al successivo, avendo così in coda $3 - 1 = 2$ persone, mentre C_2 per passare al cliente successivo (e quindi avere in attesa $5 - 1 = 4$ persone) dovrà aspettare ancora 4 turni per consumare i prodotti rimanenti del cliente in testa.

Il gioco termina quando una delle due code si svuota. Se u si troverà nella coda svuotata, deve apparire un messaggio "hai vinto!" altrimenti "hai perso!"

Suggerimento:

soltanto per vedere se tutto procede correttamenet, implementare una ulteriore funzione di stampa della coda che visualizzi anche il numero di prodotti di ogni utente

In [73]:
```
#include <stdio.h>
#include <time.h>
#define N_CODA 20
#define FULL -1
#define EMPTY -2
#define OK 0

#define MAX_PERSONE 5
#define MIN_PERSONE 1


#define MAX_PRODOTTI 10
#define MIN_PRODOTTI 1

#define DEBUG_MODE 0

struct Coda
{
    int deposito[N_CODA];
    int idx_prima_libera;
    int idx_prima_occupata;
    int n_occupati;

};

void inizializza_coda (struct Coda* c)
{
    c->idx_prima_libera   = 0;
    c->idx_prima_occupata = 0;
    c->n_occupati         = 0;
}

int is_empty(struct Coda* c)
{
    if(c->n_occupati == 0)
    {
        return 1;
    }
    return 0;
}

int is_full(struct Coda* c)
{
    if (c->n_occupati >= N_CODA)
    {
        return 1;
    }
    return 0;
}


int enqueue(struct Coda* c, int elemento)
{

    if(is_full(c) == 1)
```

```
        return FULL;
    c->deposito[c->idx_prima_libera] = elemento;
    c->idx_prima_libera = (c->idx_prima_libera+1) % N_CODA;

    c->n_occupati++;
    return OK;

}

// restituisce l'elemento in testa
int get_from_queue(struct Coda* c, int* p_val)
{
    if(is_empty(c) == 1)
        return EMPTY;

    *p_val =  c->deposito[c->idx_prima_occupata];
    return OK;
}

// elimina l'elemento in testa e lo restituisce
int dequeue(struct Coda* c, int* p_val)
{

    if(is_empty(c) == 1)
        return EMPTY;
    get_from_queue(c, p_val);
    c->idx_prima_occupata =  (c->idx_prima_occupata+1) % N_CODA;
    c->n_occupati--;
    return OK;

}

// modifica l'elemento in testa alla coda (non molto elegante...)
int update_head_queue(struct Coda* c, int nuovo_valore)
{
    if(is_empty(c) == 1)
        return EMPTY;
    c->deposito[c->idx_prima_occupata] = nuovo_valore;
    return OK;
}

void print_queue(struct Coda* c)
{
    printf("n. persone: %d\n", c->n_occupati);
    int idx = c->idx_prima_occupata;
    for(int i =0; i < c->n_occupati; i++)
    {
        printf("%d ", c->deposito[idx]);
        idx = (idx+1) % N_CODA;
    }
    printf("\n");
}

int size(struct Coda* c)
{
    return c->n_occupati;
}

long int random_int_in_range(int a, int b)
{
    return rand() % (b + 1 - a) + a;
}

int riempi_a_caso(struct Coda* c)
{
    int n_persone = random_int_in_range(MIN_PERSONE, MAX_PERSONE);

    int i = 0;

    int err = OK;
    while( i<n_persone && err == OK)
    {

        int n_prodotti = random_int_in_range(MIN_PRODOTTI, MAX_PRODOTTI);

        err            = enqueue(c, n_prodotti);

        i++;
    }

    return err;

}



void visualizza_cassa(struct Coda* c,int debug_mode)
{
    int persone_in_coda = size(c);
```

```
    if(debug_mode == 0)
        for(int i = 0; i < persone_in_coda; i++)
        {
            printf("O ");
            if (i == 0)
                printf("|");
        }
    else
        print_queue(c);
    printf("\n");

}

int consuma_prodotto(struct Coda* c)
{
    int n_prodotti_testa;
    int err = get_from_queue(c, &n_prodotti_testa);
    if(err != OK)
        return err;
    if(n_prodotti_testa > 0)
    {
        n_prodotti_testa--;
        err = update_head_queue(c, n_prodotti_testa);
    }
    else
    {
        dequeue(c, &n_prodotti_testa);
    }
    return err;

}


int main()
{
    srand(time(NULL));

    struct Coda C1;
    struct Coda C2;
    inizializza_coda(&C1);
    inizializza_coda(&C2);

    if( riempi_a_caso(&C1)!=OK )
    {
        perror("errore! coda piena!\n");
        return -1;
    }
    if( riempi_a_caso(&C2)!=OK )
    {
        perror("errore! coda piena!\n");
        return -1;
    }

    printf("SITUAZIONE CASSA 1:\n");
    visualizza_cassa(&C1,DEBUG_MODE);
    printf("SITUAZIONE CASSA 2:\n");
    visualizza_cassa(&C2,DEBUG_MODE);

    int cassa_scelta = -1;
    while(cassa_scelta != 1 && cassa_scelta != 2)
    {
        printf("In quale cassa vuoi andare inizialmente (1 o 2)? ");
        scanf("%d", &cassa_scelta);
        if(cassa_scelta != 1 && cassa_scelta != 2)
            printf("scelta non valida! Ripeti!\n");
    }

    int scelta = -1;


    while(scelta != 0 && (is_empty(&C1) == 0 && is_empty(&C2) == 0) )
    {



        printf("_____________________________________\n", cassa_scelta);
        printf("TI TROVI ATTUALMENTE ALLA CASSA %d\n", cassa_scelta);
        printf("_____________________________________\n", cassa_scelta);

        printf("SITUAZIONE CASSA 1:\n");
        visualizza_cassa(&C1,DEBUG_MODE);
        printf("SITUAZIONE CASSA 2:\n");
        visualizza_cassa(&C2,DEBUG_MODE);


        printf("scegli:\n");
        printf("1)cambio cassa\n");
        printf("2)resto\n");
        printf("0)esci\n");
```

```
        scanf("%d", &scelta);

        if(scelta == 1)
        {
            if(cassa_scelta == 1)
                cassa_scelta = 2;
            else
                cassa_scelta = 1;

            consuma_prodotto(&C1);
            consuma_prodotto(&C2);

        }
        else if(scelta == 2)
        {
            consuma_prodotto(&C1);
            consuma_prodotto(&C2);
        }
        else if(scelta != 0)
        {
            printf("scelta non valida! Ripeti!\n");
        }

    }

     if(is_empty(&C1) == 1)
         printf("si è svuotata la CASSA 1\n");

     if(is_empty(&C2) == 1)
         printf("si è svuotata la CASSA 2\n");

    if(is_empty(&C1) == 1 && cassa_scelta == 1)
       printf("Hai vinto!\n");
    else if(is_empty(&C2) == 1 && cassa_scelta == 2)
        printf("Hai vinto!\n");
    else
        printf("Hai perso!\n");

    return 0;

}
```

```
SITUAZIONE CASSA 1:
O |O O O
SITUAZIONE CASSA 2:
O |O
In quale cassa vuoi andare inizialmente (1 o 2)?

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O O O
SITUAZIONE CASSA 2:
O |O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O O O
SITUAZIONE CASSA 2:
O |O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O O O
SITUAZIONE CASSA 2:
O |O
scegli:
1)cambio cassa
2)resto
0)esci
```

```
_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O O O
SITUAZIONE CASSA 2:
O |O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O O O
SITUAZIONE CASSA 2:
O |O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O O O
SITUAZIONE CASSA 2:
O |O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O O O
SITUAZIONE CASSA 2:
O |O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O O O
SITUAZIONE CASSA 2:
O |O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O O O
SITUAZIONE CASSA 2:
O |O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O O
SITUAZIONE CASSA 2:
O |O
scegli:
1)cambio cassa
2)resto
0)esci
```

```
_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O O
SITUAZIONE CASSA 2:
O |O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O O
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O O
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci
```

```
_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O
SITUAZIONE CASSA 2:

O |
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci

si è svuotata la CASSA 2
Hai vinto!
```

In [ ]:

```
 // alternativa più elegante

#include <stdio.h>
#include <time.h>
#define N_CODA 20
#define FULL -1
#define EMPTY -2
#define OK 0

#define MAX_PERSONE 5
#define MIN_PERSONE 1


#define MAX_PRODOTTI 10
#define MIN_PRODOTTI 1

#define DEBUG_MODE 0

struct Coda
{
    int deposito[N_CODA];
    int idx_prima_libera;
    int idx_prima_occupata;
    int n_occupati;

};


struct Cassa
{
    int prodotti_da_passare;
    struct Coda coda;

};


long int random_int_in_range(int a, int b)
{
```

```
    return rand() % (b + 1 - a) + a;
}



// funzioni gestione coda
void inizializza_coda (struct Coda* c)
{
    c->idx_prima_libera   = 0;
    c->idx_prima_occupata = 0;
    c->n_occupati         = 0;
}


int is_empty(struct Coda* c)
{
    if(c->n_occupati == 0)
    {
        return 1;
    }
    return 0;
}

int is_full(struct Coda* c)
{
    if (c->n_occupati >= N_CODA)
    {
        return 1;
    }
    return 0;
}


int enqueue(struct Coda* c, int elemento)
{

    if(is_full(c) == 1)
        return FULL;
    c->deposito[c->idx_prima_libera] = elemento;
    c->idx_prima_libera = (c->idx_prima_libera+1) % N_CODA;

    c->n_occupati++;
    return OK;

}

// restituisce l'elemento in testa
int get_from_queue(struct Coda* c, int* p_val)
{
    if(is_empty(c) == 1)
        return EMPTY;

    *p_val =  c->deposito[c->idx_prima_occupata];
    return OK;
}

// elimina l'elemento in testa e lo restituisce
int dequeue(struct Coda* c, int* p_val)
{

    if(is_empty(c) == 1)
        return EMPTY;
    get_from_queue(c, p_val);
    c->idx_prima_occupata =  (c->idx_prima_occupata+1) % N_CODA;
    c->n_occupati--;
    return OK;

}

void print_queue(struct Coda* c)
{
    printf("n.items: %d\n", c->n_occupati);
    int idx = c->idx_prima_occupata;
    printf("items: ");
    for(int i =0; i < c->n_occupati; i++)
    {
        printf("%d ", c->deposito[idx]);
        idx = (idx+1) % N_CODA;
    }
    printf("\n");
}

int size(struct Coda* c)
{
    return c->n_occupati;
}


int riempi_a_caso(struct Coda* c)
```

```
{
    int n_persone = random_int_in_range(MIN_PERSONE, MAX_PERSONE);

    int i   = 0;

    int err = OK;
    while( i<n_persone && err == OK)
    {

        int n_prodotti = random_int_in_range(MIN_PRODOTTI, MAX_PRODOTTI);

        err            = enqueue(c, n_prodotti);

        i++;
    }

    return err;

}


// funzioni gestione cassa

void inizializza_cassa(struct Cassa* c)
{
    c->prodotti_da_passare = 0;
    inizializza_coda(&(c->coda));
    riempi_a_caso(&(c->coda));

}


int is_cassa_vuota(struct Cassa* c)
{
    if (c->prodotti_da_passare > 0)
        return 0;
    if (is_empty(&(c->coda))==1)
        return 1;
    return 0;

}

void visualizza_cassa(struct Cassa* cassa,int debug_mode)
{
    int persone_in_coda = size(&(cassa->coda));
    if (cassa->prodotti_da_passare > 0)
        persone_in_coda++;

    if(debug_mode == 0)
        for(int i = 0; i < persone_in_coda; i++)
        {
            printf("O ");
            if (i == 0)
                printf("|");
        }
    else
    {
        printf("prodotti da passare: %d\n", cassa->prodotti_da_passare);
        printf("stato coda:\n");
        print_queue(&cassa->coda);
    }

    printf("\n");

}

int consuma_prodotto(struct Cassa* cassa)
{

    if(cassa->prodotti_da_passare > 0)
    {
        (cassa->prodotti_da_passare)--;

    }
    else
    {
        int err = dequeue(&cassa->coda, &(cassa->prodotti_da_passare));
        if(err != OK)
            return err;
    }
    return OK;

}




int main()
```

```
{
    srand(time(NULL));

    struct Cassa C1;
    struct Cassa C2;
    inizializza_cassa(&C1);
    inizializza_cassa(&C2);


    printf("SITUAZIONE CASSA 1:\n");
    visualizza_cassa(&C1, DEBUG_MODE);
    printf("SITUAZIONE CASSA 2:\n");
    visualizza_cassa(&C2, DEBUG_MODE);

    int cassa_scelta = -1;
    while(cassa_scelta != 1 && cassa_scelta != 2)
    {
        printf("In quale cassa vuoi andare inizialmente (1 o 2)? ");
        scanf("%d", &cassa_scelta);
        if(cassa_scelta != 1 && cassa_scelta != 2)
            printf("scelta non valida! Ripeti!\n");
    }

    int scelta = -1;


    while(scelta != 0 && (is_cassa_vuota(&C1) == 0 && is_cassa_vuota(&C2) == 0) )
    {



        printf("_____________________________________\n", cassa_scelta);
        printf("TI TROVI ATTUALMENTE ALLA CASSA %d\n", cassa_scelta);
        printf("_____________________________________\n", cassa_scelta);

        printf("SITUAZIONE CASSA 1:\n");
        visualizza_cassa(&C1,DEBUG_MODE);
        printf("SITUAZIONE CASSA 2:\n");
        visualizza_cassa(&C2,DEBUG_MODE);


        printf("scegli:\n");
        printf("1)cambio cassa\n");
        printf("2)resto\n");
        printf("0)esci\n");
        scanf("%d", &scelta);

        if(scelta == 1)
        {
            if(cassa_scelta == 1)
                cassa_scelta = 2;
            else
                cassa_scelta = 1;

            consuma_prodotto(&C1);
            consuma_prodotto(&C2);

        }
        else if(scelta == 2)
        {
            consuma_prodotto(&C1);
            consuma_prodotto(&C2);
        }
        else if(scelta != 0)
        {
            printf("scelta non valida! Ripeti!\n");
        }

    }

     if(is_cassa_vuota(&C1) == 1)
         printf("si è svuotata la CASSA 1\n");

     if(is_cassa_vuota(&C2) == 1)
         printf("si è svuotata la CASSA 2\n");

    if(is_cassa_vuota(&C1) == 1 && cassa_scelta == 1)
       printf("Hai vinto!\n");
    else if(is_cassa_vuota(&C2) == 1 && cassa_scelta == 2)
        printf("Hai vinto!\n");
    else
        printf("Hai perso!\n");

    return 0;

}
```

```
SITUAZIONE CASSA 1:
O |O O O
SITUAZIONE CASSA 2:
O |O
In quale cassa vuoi andare inizialmente (1 o 2)?

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O O O
SITUAZIONE CASSA 2:
O |O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O O O
SITUAZIONE CASSA 2:
O |O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O O O
SITUAZIONE CASSA 2:
O |O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O O O
SITUAZIONE CASSA 2:
O |O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O O O
SITUAZIONE CASSA 2:
O |O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O O O
SITUAZIONE CASSA 2:
O |O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O O
SITUAZIONE CASSA 2:
O |O
scegli:
1)cambio cassa
2)resto
0)esci
```

TI TROVI ATTUALMENTE ALLA CASSA 2

SITUAZIONE CASSA 1:
O |O O
SITUAZIONE CASSA 2:
O |O
scegli:
1)cambio cassa
2)resto
0)esci

TI TROVI ATTUALMENTE ALLA CASSA 2

SITUAZIONE CASSA 1:
O |O O
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci

TI TROVI ATTUALMENTE ALLA CASSA 2

SITUAZIONE CASSA 1:
O |O O
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci

TI TROVI ATTUALMENTE ALLA CASSA 2

SITUAZIONE CASSA 1:
O |O O
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci

TI TROVI ATTUALMENTE ALLA CASSA 2

SITUAZIONE CASSA 1:
O |O
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci

TI TROVI ATTUALMENTE ALLA CASSA 2

SITUAZIONE CASSA 1:
O |O
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci

TI TROVI ATTUALMENTE ALLA CASSA 2

SITUAZIONE CASSA 1:
O |O
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci

```
_____________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_____________________________________
SITUAZIONE CASSA 1:
O |O
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci


_____________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_____________________________________
SITUAZIONE CASSA 1:
O |O
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci


_____________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_____________________________________
SITUAZIONE CASSA 1:
O |
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci


_____________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_____________________________________
SITUAZIONE CASSA 1:
O |
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci

si è svuotata la CASSA 1
Hai perso!
```

In [ ]:

dato un vettore $\mathbf{v} = (v_1, v_2, \ldots, v_n)$, fornire una soluzione ricorsiva per stamparne i valori. Si consideri come caso base la stampa di un singolo elemento

$$stampa\_vett(\mathbf{v}): \begin{cases} nothing & \text{se } \#\mathbf{v} = 0 \\ stampa(v_1) & \text{se } \#\mathbf{v} = 1 \\ stampa(v_1), stampa\_vett(\mathbf{v}_{2,\ldots,\#\mathbf{v}}) & \text{altrimenti.} \end{cases}$$

In [6]:
```
#include <stdio.h>
void stampa_vettore(int *v, int n)
{
    if( n == 0 )
    {
        printf("\n");
        return;
    }
    stampa_vettore(&v[1], n-1); // <=> stampa_vettore(V+1, n-1);
    printf("%2d, ", v[0]);       // <=> printf("%d ", *V);

}



int main()
{
 int v[] = {1,5,11,9};
 stampa_vettore(v, 4);

 return 0;
}
```

```
 9, 11,  5,  1,
```

Stampare una matrice di dimensioni $n_r \times n_c$ di interi utilizzando una soluzione ricorsiva. Si consideri come caso base la stampa di una singola riga della matrice.

- scompongo in sottoproblemi
  - Posso vedere la stampa di una matrice come la stampa di R vettori ognuno di lunghezza C.
- determino i casi base
  - se la matrice ha 0 righe, non stampo nulla
  - se la matrice ha una riga, stampo la riga
- determino il caso ricorsivo
  - se la matrice ha più di una riga, stampa la prima riga e richiamo la procedura sulle righe rimanenti

$$stampa\_matrice: \begin{cases} nothing & \text{se } \#righe(M) = 0 \\ stampa\_riga(M_1), & \text{se } \#righe(M) = 1 \\ stampa\_riga(M_1), stampa\_matrice(M_{2,\ldots,\#righe(M)}), & \text{se } \#righe(M) > 1 \end{cases}$$

o più semplicemente

$$stampa\_matrice: \begin{cases} nothing & \text{se } \#righe(M) = 0 \\ stampa\_riga(M_1), stampa\_matrice(M_{2,\ldots,\#righe(M)}), & altrimenti \end{cases}$$

Inoltre, posso vedere la stampa di una riga come l'acquisizione di C vettori di lunghezza 1:

$$stampa\_riga(\mathbf{v}): \begin{cases} nothing & \text{se } \#\mathbf{v} = 0 \\ stampa\_elemento(v_1), stampa\_riga(\mathbf{v}_{2,\ldots,\#\mathbf{v}}) & altrimenti \end{cases}$$

In [3]:
```
// visualizzazione di una matrice. Possibile soluzione ricorsiva
#include <stdio.h>
#define R 3
#define C 5


void stampa_vettore(int *v, int n)
{
    if( n == 0 )
    {
        printf("\n");
        return;
    }
    printf("%2d, ", v[0]);       // <=> printf("%d ", *V);
    stampa_vettore(&v[1], n-1); // <=> stampa_vettore(V+1, n-1);
```

```
}

void stampa_matrice(int A[][C],int r, int c)
{

    if(r == 0)
        return;
    stampa_vettore(*A, c);
    stampa_matrice(A+1, r-1, c);
    return;

}

void stampa_iter(int A[][C], int r, int c)
{
    for(int i = 0; i<r; i=i+1)
    {
        for(int j = 0; j<c; j=j+1)
        {
            printf("[%d,%d] => %2d ", i,j, A[i][j]);
        }
        printf("\n");

    }
}

int main()
{
    int A[R][C] = {{1,2,3,4,5},
                   {6,7,8,9,10},
                   {11,12,13,14,15}};

    stampa_matrice(A,R,C);
    //stampa_iter(A,R,C);
}
```

```
 1,  2,  3,  4,  5,
 6,  7,  8,  9, 10,
11, 12, 13, 14, 15,
```

In [ ]:

```
// Acquisizione e visualizzazione di una matrice.
// Soluzione NON ricorsiva con utilizzo di funzioni ricorsive
#include <stdio.h>
#define R 2
#define C 3
void acquisisci(int A[][C], int r, int c, int i, int j)
{
    if(i == r-1 && j == c)
        return;
    if(j == c)
    {
        j = 0; // idx colonne a 0
        ++i;   // prox riga
    }

    printf("inserisci l'elemento in posizione %d,%d\n", i, j);
    scanf("%d", &A[i][j]);

    acquisisci(A, r,c,i,++j);
}

void stampa(int A[][C], int r, int c, int i, int j)
{
    if(j == c)
    {
        ++i; // prox riga
        j = 0; // idx colonne a 0
        printf("\n");
    }
    if(i == r)
        return;
    printf("%5d ", A[i][j]);

    stampa(A, r, c, i, ++j);
}

int main()
{
    int A[R][C];
    acquisisci(A,R,C,0,0);
    stampa(A,R,C,0,0);
}
```

Dati due vettori $\mathbf{v}, \mathbf{w} \in \mathrm{R}^n$, fornire una soluzione ricorsiva per il calcolo del loro prodotto scalare

$$\mathbf{v} \cdot \mathbf{w} = \begin{cases} v_1 \cdot w_1 & \text{se } \#\mathbf{v} = 1 \\ v_1 \cdot w_1 + \mathbf{v}_{2,\ldots,n} \cdot \mathbf{w}_{2,\ldots,n} & \text{se } \#\mathbf{v} > 1 \end{cases}$$

In [14]:
```
#include <stdio.h>
#include <string.h>

float vetmul(float v[],float w[], int n)
{

    if(n == 1)
        return v[0]*w[0];
    return v[0]*w[0] + vetmul(v+1, w+1, n-1);

}

int main()
{
    float v[]   = {1,2,3,4};
    float w[]   = {1,2,3,4};

    printf("prodotto vw: %.2f\n", vetmul(v,w, 4));

    return 0;
}
```

```
prodotto vw: 30.00
```

data una stringa $\mathbf{s}$, fornire una soluzione ricorsiva per la verifica che s sia palindroma

$$palindroma(\mathbf{s}) = \begin{cases} True & \text{se } \#\mathbf{s} \leq 1 \\ (s_1 = s_{\#\mathbf{s}}) \wedge palindroma(\mathbf{s}_{2,\ldots,\#\mathbf{s}-1}) & \text{se } \#\mathbf{s} > 1 \end{cases}$$

Nota: il caso *false* è implicito nel secondo caso se $s_1 \neq s_{\#\mathbf{s}}$.

O anche

$$palindroma(\mathbf{s}) = \begin{cases} True & \text{se } \#\mathbf{s} \leq 1 \\ False & \text{se } \#\mathbf{s} > 1 \wedge s_1 \neq s_{\#\mathbf{s}} \\ palindroma(\mathbf{s}_{2,\ldots,\#\mathbf{s}-1}) & \text{se } \#\mathbf{s} > 1 \wedge s_1 = s_{\#\mathbf{s}} \end{cases}$$

In [33]:
```
#include <stdio.h>
#include <string.h>

int palindroma(char s_start[],char s_end[])
{

    if(s_start >= s_end)
        return 1;

    if(*s_start != *s_end)
        return 0;
    return palindroma(s_start+1,s_end-1);


}

int main()
{
    char* s = "lanssnal";
    int len = strlen(s);

    int is_pal = palindroma(s, s+len-1);

    printf("La stringa %s ", s);
    if(is_pal == 0)
        printf("non ");
    printf("è palindroma\n");
    return 0;
}
```

```
La stringa lanssnal è palindroma
```

Data una matrice quadrata, proporre una soluzione ricorsiva per stamparne la diagonale principale

$$stampa\_diag(M): \begin{cases} None & \text{se } \#righe(M) = 0 \\ m_{1,1}, stampa\_diag(M_{2\ldots\#righe(M),2\ldots\#colonne(M)}) & \text{altrimenti} \end{cases}$$

In [10]:
```
// matrice allocata nell'heap (strategia Matrix as array of pointers to subarrays)
// metodo 1: allocazioni di nuove submatrici
#include <stdio.h>
```

```
#include <string.h>

int** allocate_mat(int n)
{
    int** m    = calloc(n, sizeof(int*));

    m[0] = calloc(n*n, sizeof(int));

    for(int i = 1; i < n; i++)
        m[i] = m[0] + i*n;  // sfrutto l'aritmetica dei puntatori
    return m;

}

void stampa_mat(int** M, int n)
{
    printf("(\n");
    for (int i =0; i< n; i++)
    {
        for(int j = 0; j< n; j++)
            printf("%2d, ", M[i][j]);
       printf("\n");
    }
    printf(")");

}



int** make_submat(int** M, int n)
{
    int new_n       = n-1;
    int** tmp       = allocate_mat(new_n);
    for(int i = 1; i < n; i++)
    {
        for(int j = 1; j < n; j++)
        {
            tmp[i-1][j-1] = M[i][j];
        }
    }
    return tmp;
}

void stampa_diag(int** M, int n)
{
    printf("%d, ", **M);
    if(n>1)
    {
        int** new_M = make_submat(M, n);
        stampa_diag(new_M,n-1);
        free(new_M);
    }

}


int main()
{
    int n  = 4;

    int **M = allocate_mat(n);
    for(int i = 0; i < n*n; i++)
        M[0][i] = i+1;
    stampa_mat(M, n);

    printf("\nDiagonale: ");
    stampa_diag(M, n);
    printf("\n");
    free(M);
    return 0;

}
```

```
(
 1,  2,  3,  4,
 5,  6,  7,  8,
 9, 10, 11, 12,
13, 14, 15, 16,
)
Diagonale: 1, 6, 11, 16,
```

In [20]:

```
// matrice allocata nell'heap (strategia Matrix as Array)
// metodo 1: allocazioni di nuove submatrici
#include <stdio.h>
#include <string.h>

void stampa_mat(int* M, int n)
{
    printf("(\n");
```

```
    for (int i =0; i< n; i++)
    {
        for(int j = 0; j< n; j++)
            printf("%2d, ", M[i*n+j]);
       printf("\n");
    }
    printf(")");

}

int* make_submat(int* M, int n)
{
    int new_n       = n-1;
    int* tmp        = malloc(new_n*new_n*sizeof(int));
    int  cursor_tmp = 0;
    for(int i = 1; i < n; i++)
    {
        for(int j = 1; j < n; j++)
        {
            tmp[cursor_tmp] = M[i*n + j];
            cursor_tmp++;
        }
    }
    return tmp;
}

void stampa_diag(int* M, int n)
{
    printf("%d, ", *M);
    if(n>1)
    {
        int* new_M = make_submat(M, n);
        stampa_diag(new_M,n-1);
        free(new_M);
    }

}


int main()
{
    int n  = 4;

    int *M = malloc(n*n*sizeof(int));
    for(int i = 0; i < n*n; i++)
        M[i] = i+1;
    stampa_mat(M, n);

    printf("\nDiagonale: ");
    stampa_diag(M, n);
    printf("\n");
    free(M);
    return 0;

}
```

```
(
 1,  2,  3,  4,
 5,  6,  7,  8,
 9, 10, 11, 12,
13, 14, 15, 16,
)
Diagonale: 1, 6, 11, 16,
```

In [19]:

```
// matrice allocata nell'heap (strategia Matrix as Array)
// metodo 2: sfruttando l'aritmetica dei puntatori
#include <stdio.h>
#include <string.h>

void stampa_mat(int *M, int n)
{
    printf("(\n");
    for (int i =0; i< n; i++)
    {
        for(int j = 0; j< n; j++)
            printf("%2d, ", M[i*n+j]);
       printf("\n");
    }
    printf(")");

}

void stampa_diag(int* M, int n, int offset_righe)
{
    printf("%d, ",*M);
    if(n>1)
        stampa_diag((M+offset_righe)+1,n-1, offset_righe);
```

```
}


int main()
{
    int n  = 4;

    int *M = malloc(n*n*sizeof(int));
    for(int i = 0; i < n*n; i++)
        M[i] = i+1;
    stampa_mat(M, n);

    printf("\nDiagonale: ");
    stampa_diag(M, n, n);
    printf("\n");
    free(M);
    return 0;

}
```

```
(
 1,  2,  3,  4,
 5,  6,  7,  8,
 9, 10, 11, 12,
13, 14, 15, 16,
)
Diagonale: 1, 6, 11, 16,
```

In [3]:

```
// matrice allocata sullo stack

#include <stdio.h>
#include <string.h>
#define MAX_N 10

void stampa_mat(int M[][MAX_N], int n)
{
    printf("(\n");
    for (int i =0; i< n; i++)
    {
        for(int j = 0; j< n; j++)
            printf("%d, ", M[i][j]);
       printf("\n");
    }
    printf(")");

}

void stampa_diag(int* M, int n)
{
    printf("%d, ",*M);
    if(n>1)
        stampa_diag((M+MAX_N)+1,n-1); // l'offset è sempre MAX_N

}


int main()
{

    int M[MAX_N][MAX_N] = {{1,2,3},
                           {4,5,6},
                           {7,8,9}};

    stampa_mat(M, 3);

    printf("\nDiagonale: ");
    stampa_diag(*M, 3);
    printf("\n");
    return 0;

}
```

```
(
1, 2, 3,
4, 5, 6,
7, 8, 9,
)
Diagonale: 1, 5, 9,
```

Dato un numero intero ≥ 0, fornire una funzione ricorsiva che ne stampi la sua codifica binaria

$$bin(x): \begin{cases} print(x) & \text{se } x = 0 \vee x = 1 \\ bin(\lfloor x/2 \rfloor), print(mod(x, 2)) & \text{altrimenti} \end{cases}$$

oppure

$$bin(x): \begin{cases} print(x) & \text{se } x = 0 \vee x = 1 \\ bin(\lfloor x/2 \rfloor), bin(mod(x,2)) & \text{altrimenti} \end{cases}$$

In [4]:
```
#include <stdio.h>

void bin(int x)
{
    if(x == 0 || x == 1)
    {
        printf("%d", x);
        return;
    }
    bin(x/2);
    printf("%d",x%2);
}



int main()
{

    bin(12);

    return 0;

}
```

```
1100
```

Dato un numero intero ≥ 0, fornire una funzione ricorsiva che ne *restituisca* la sua codifica binaria

$$bin(x) = \begin{cases} x & \text{se } x = 0 \vee x = 1 \\ 10 \cdot bin(\lfloor x/2 \rfloor) + mod(x,2) & \text{altrimenti} \end{cases}$$

In [2]:
```
#include <stdio.h>

int bin(int x)
{
    if(x == 0 || x==1)
        return x ;
    return 10*bin(x/2) + x%2;

}

int main()
{
    int cod = bin(16);
    printf("%d\n", cod);
    return 0;
}
```

```
10000
```

Data una lista concatenata contenente valori interi, scrivere una funzione ricorsiva che ne elimini tutti i nodi avente valore pari a k, con k fornito dall'utente

Possibile ragionamento:

- se la lista è vuota, restitutisci la lista vuota
- altrimenti
  - se il primo elemento della lista è k, restituisci la lista ottenuta cancellando tutte le occorrenze di k a partire dall'elemento successivo
  - altrimenti, restituisci la lista il cui primo elemento coincide con il primo elemento della lista ed il resto è ottenuto cancellando tutte le occorrenze di k a partire dall'elemento successivo

In [12]:
```
#include <stdio.h>
#include <time.h>
#define OK 0
struct Nodo {
    int valore;
    struct Nodo* prossimo;

};



// funzioni gestione lista
void init_list(struct Nodo** p_head)
{
    *p_head = NULL;
}
```

```
int insert(struct Nodo** p_head, int val)
{

    struct Nodo* nuovo = malloc(sizeof(struct Nodo));
    nuovo->valore   = val;
    nuovo->prossimo = NULL;

    if (*p_head == NULL)
    {
       *p_head = nuovo;
    }
    else
    {
        struct Nodo* cursore = *p_head;
        while(cursore->prossimo != NULL)
            cursore = cursore->prossimo;
        cursore->prossimo = nuovo;
    }

    return OK;

}


// stampa ricorsiva di una lista
void print_list(struct Nodo* p_head)
{

    if(p_head == NULL)
    {
        printf("\n");
        return;
    }
    printf("{%d}->", p_head->valore);
    print_list(p_head->prossimo);
}


long int random_int_in_range(int a, int b)
{
    return rand() % (b + 1 - a) + a;
}

void fill_random(struct Nodo** p_head, int n)
{
    for(int i=0; i < n; i++)
        insert(p_head, random_int_in_range(1,3));

}

/*
struct Nodo* remove_key(struct Nodo* p_head, int k)
{
    if (p_head == NULL)
        return NULL;
    p_head->prossimo = remove_key(p_head->prossimo, k);
    if(p_head->valore == k)
    {
        struct Nodo* to_free = p_head;
        p_head = p_head->prossimo;
        free(to_free);
    }

    return p_head;
}
*/

struct Nodo* remove_key(struct Nodo* p_head, int k)
{
    if (p_head == NULL)
        return NULL;
    if(p_head->valore == k)
    {
        struct Nodo* prox = p_head->prossimo;
        free(p_head);
        return remove_key(prox,k);
    }
    p_head->prossimo = remove_key(p_head->prossimo, k);
    return p_head;

}



int main()
{
    struct Nodo* p_head;
```

```
    init_list(&p_head);

    fill_random(&p_head, 10);
    printf("situazione iniziale:\n");
    print_list(p_head);


    int k;
    printf("quale valore desideri eliminare?");
    scanf("%d", &k);

    p_head = remove_key(p_head, k);

    printf("situazione finale:\n");
    print_list(p_head);


}
```

```
situazione iniziale:
{2}->{2}->{1}->{2}->{3}->{2}->{2}->{1}->{1}->{2}->
quale valore desideri eliminare?

situazione finale:
{1}->{3}->{1}->{1}->
```

In [ ]:

Si realizzi un programma in linguaggio C che,

- legga un file di testo dove per ogni riga si ha `NomeAzienda Dipendenti Dipartimenti Sedi`. I dati vanno immagazzinati in una opportuna lista a singolo link; i dati nella lista devono contenere tutti i dati inclusi nel file di input nello stesso ordine del file di input, o al più un ordine inverso; stampare la lista a schermo dopo la lettura.
- successivamente si eliminino i record relativi alle aziende che hanno meno di 300 dipendenti; stampare la lista a schermo al termine dell'operazione;
- riscriva i record su un altro file, seguendo lo stesso formato del file di input, in maniera che le aziende siano ordinate i maniera decrescente secondo il numero di dipedenti (N.B., Non è necessario ordinare la lista).

Per semplicità considerate che i nomi delle aziende, come nell'esempio, siano composte da una sola parola senza spazi.

Esempio:

Supponendo che il file di input contenga

```
ascii
Ubuntu 230 27 4
Fedora 580 22 16
Manjaro 410 24 12
Mint 320 17 8
Antergos 470 28 13
Arch 290 21 7
CentOS 440 33 12
Kali 240 35 5

ascii
Fedora 580 30 16
Antergos 470 30 13
CentOS 440 30 12
Manjaro 410 30 12
Mint 320 30 8
```

In [17]:
```
// strategia 1
// - inserimento in lista ricorsivo
// - distruttiva della lista durante l'inserimento ordinato all'interno del file
#include <stdio.h>
#define ERRFILE -1
#define OK 0
struct Azienda
{
    char NomeAzienda[64];
    int Dipendenti;
    int Dipartimenti;
    int Sedi;

    struct Azienda* successivo;

};


void print_list(struct Azienda* p_head)
{
    if (p_head == NULL)
        return;
    printf("NOME: %s, DIPENDENTI: %d, DIPARTIMENTI: %d, SEDI: %d\n",
           p_head->NomeAzienda, p_head->Dipendenti,
           p_head->Dipartimenti, p_head->Sedi);
    print_list(p_head->successivo);

}

int length_list(struct  Azienda* p_head)
{
    int n = 0;
    while(p_head != NULL)
    {
        p_head = p_head -> successivo;
        n++;
    }
    return n;
}

void free_list(struct  Azienda* p_head)
{
    struct  Azienda* tmp;
    while(p_head != NULL)
    {
        tmp = p_head->successivo;
        free(p_head);
        p_head = tmp;
```

```
    }

}

void append_in_list(struct  Azienda** pp_head, struct Azienda az)
{
    if(*pp_head == NULL)
    {
      struct Azienda* p_new = malloc(sizeof(struct Azienda));
      *p_new                = az; // copio i valori presi dal file
                                  // nella nuova struct

      /*p_new->Dipendenti = az.Dipendenti;
      p_new->Dipartimenti = az.Dipartimenti;
      p_new->Sedi = az.Sedi;
      strcpy(p_new->NomeAzienda, az.NomeAzienda);
      */

      p_new->successivo     = NULL;
      *pp_head              = p_new;
      return;

    }
    append_in_list( &( (*pp_head)->successivo ), az);

}


int read_from_file(char filename[64], struct  Azienda** pp_head)
{
  FILE* fp = fopen(filename, "r");
  if (fp == NULL)
      return ERRFILE;

  // creo struttura temporanea
  struct Azienda tmp;
  // leggo una tupla di valori dal file
  while(fscanf(fp,"%s %d %d %d",
               tmp.NomeAzienda, &(tmp.Dipendenti),
               &(tmp.Dipartimenti), &(tmp.Sedi)) == 4)
  {
      // inserisco il nodo nella lista
      append_in_list(pp_head, tmp);
  }
  // 3. chiudo il file
  fclose(fp);

  return OK;
}

void remove_less_than(struct  Azienda** pp_head, int soglia_dip)
{
    struct Azienda* cursore = *pp_head;
    struct Azienda* prec    = NULL;
    while(cursore != NULL)
    {
        if(cursore->Dipendenti < soglia_dip)
        {
            struct Azienda* to_del = cursore;
            if (prec == NULL)
                *pp_head = to_del->successivo;
            else
                prec->successivo = to_del->successivo;

            cursore = to_del->successivo;
            free(to_del);
        }
        else
        {
            prec   = cursore;
            cursore = cursore->successivo;

        }

    }

}


int write_sorted_on_file(char filename[64], struct  Azienda** pp_head)
{
    FILE* fp = fopen(filename, "w");
    if(fp == NULL)
        return ERRFILE;

    while(*pp_head != NULL)
    {
```

```
        // cerco l'azienda con il maggior numero di dipendenti
        // non ancora inserita nel file
        struct Azienda* cursore    =  *pp_head;
        struct Azienda* p_max      =  *pp_head;
        struct Azienda* prec       = NULL;
        struct Azienda* p_prec_max = NULL;
        while( cursore !=  NULL )
        {

            if(cursore->Dipendenti > p_max->Dipendenti)
            {
                p_prec_max = prec;
                p_max      = cursore;
            }
            prec    = cursore;
            cursore = cursore->successivo;
        }
        // scrivo l'azienda nel file
        fprintf(fp,"%s %d %d %d\n",
               p_max->NomeAzienda, p_max->Dipendenti,
               p_max->Dipartimenti, p_max->Sedi);

        // rimuovo dalla lista
        if(p_prec_max == NULL)
            *pp_head = p_max->successivo;
        else
            p_prec_max->successivo = p_max->successivo;
        free(p_max);
    }

    fclose(fp);
    return OK;

}

void err_handler(int err)
{
    switch(err)
    {
        case OK:
            printf("OK!\n");
        break;
        case ERRFILE:
            printf("errore nella gestione del file!\n");
            exit(1);
        break;
        default:
            printf("errore non gestito!\n");
            exit(1);
    }

}


int main()
{
    struct Azienda* p_head = NULL;

    int err;
    printf("leggo dal file...");
    err = read_from_file("aziende.txt", &p_head);
    err_handler(err);
    printf("prima della rimozione:\n");
    print_list(p_head);
    remove_less_than(&p_head, 300);
    printf("dopo la rimozione:\n");
    print_list(p_head);
    printf("salvo...");
    err = write_sorted_on_file("aziende_new.txt", &p_head);
    err_handler(err);
    printf("lista dopo il salvataggio:\n");
    print_list(p_head);
    free_list(p_head);
    p_head = NULL;


    return 0;
}
```

```
leggo dal file...OK!
prima della rimozione:
NOME: Antergos2, DIPENDENTI: 470, DIPARTIMENTI: 1, SEDI: 4
NOME: Ubuntu, DIPENDENTI: 230, DIPARTIMENTI: 27, SEDI: 4
NOME: Fedora, DIPENDENTI: 580, DIPARTIMENTI: 22, SEDI: 16
NOME: Fedora2, DIPENDENTI: 580, DIPARTIMENTI: 10, SEDI: 40
NOME: Manjaro, DIPENDENTI: 410, DIPARTIMENTI: 24, SEDI: 12
NOME: Mint, DIPENDENTI: 320, DIPARTIMENTI: 17, SEDI: 8
NOME: Antergos, DIPENDENTI: 470, DIPARTIMENTI: 28, SEDI: 13
NOME: Arch, DIPENDENTI: 290, DIPARTIMENTI: 21, SEDI: 7
NOME: CentOS, DIPENDENTI: 440, DIPARTIMENTI: 33, SEDI: 12
NOME: Kali, DIPENDENTI: 240, DIPARTIMENTI: 35, SEDI: 5
NOME: Manjaro2, DIPENDENTI: 410, DIPARTIMENTI: 1, SEDI: 2
dopo la rimozione:
NOME: Antergos2, DIPENDENTI: 470, DIPARTIMENTI: 1, SEDI: 4
NOME: Fedora, DIPENDENTI: 580, DIPARTIMENTI: 22, SEDI: 16
NOME: Fedora2, DIPENDENTI: 580, DIPARTIMENTI: 10, SEDI: 40
NOME: Manjaro, DIPENDENTI: 410, DIPARTIMENTI: 24, SEDI: 12
NOME: Mint, DIPENDENTI: 320, DIPARTIMENTI: 17, SEDI: 8
NOME: Antergos, DIPENDENTI: 470, DIPARTIMENTI: 28, SEDI: 13
NOME: CentOS, DIPENDENTI: 440, DIPARTIMENTI: 33, SEDI: 12
NOME: Manjaro2, DIPENDENTI: 410, DIPARTIMENTI: 1, SEDI: 2
salvo...OK!
lista dopo il salvataggio...
```

In [34]:

```
// strategia 2
// - inserimento in lista ricorsivo
// - non distruttiva della struttura dati
// - utilizzo di un puntatore di supporto
#include <stdio.h>

#define ERRFILE -1
#define OK 0
struct Azienda
{
    char NomeAzienda[64];
    int Dipendenti;
    int Dipartimenti;
    int Sedi;

    struct Azienda* successivo;

};


void print_list(struct Azienda* p_head)
{
    if (p_head == NULL)
        return;
    printf("NOME: %s, DIPENDENTI: %d, DIPARTIMENTI: %d, SEDI: %d\n",
           p_head->NomeAzienda, p_head->Dipendenti,
           p_head->Dipartimenti, p_head->Sedi);
    print_list(p_head->successivo);

}

int length_list(struct  Azienda* p_head)
{
    int n = 0;
    while(p_head != NULL)
    {
        p_head = p_head -> successivo;
        n++;
    }
    return n;
}

void free_list(struct  Azienda* p_head)
{
    struct  Azienda* tmp;
    while(p_head != NULL)
    {
        tmp = p_head->successivo;
        free(p_head);
        p_head = tmp;

    }

}

void append_in_list(struct  Azienda** pp_head, struct Azienda az)
{
    if(*pp_head == NULL)
    {
      struct Azienda* p_new = malloc(sizeof(struct Azienda));
      *p_new                = az; // copio i valori presi dal file
                                  // nella nuova struct
      p_new->successivo     = NULL;
      *pp_head              = p_new;
```

```
      return;

    }
    append_in_list( &( (*pp_head)->successivo ), az);
}


int read_from_file(char filename[64],struct  Azienda** pp_head)
{
  FILE* fp = fopen(filename, "r");
  if (fp == NULL)
      return ERRFILE;

  // creo struttura temporanea
  struct Azienda tmp;
  // leggo una tupla di valori dal file
  while(fscanf(fp,"%s %d %d %d",
               tmp.NomeAzienda, &(tmp.Dipendenti),
               &(tmp.Dipartimenti), &(tmp.Sedi)) == 4)
  {
      // inserisco il nodo nella lista
      append_in_list(pp_head, tmp);
  }
  // chiudo il file
  fclose(fp);

  return OK;
}

void remove_less_than(struct  Azienda** pp_head, int soglia_dip)
{
    struct Azienda* cursore = *pp_head;
    struct Azienda* prec    = NULL;
    while(cursore != NULL)
    {
        if(cursore->Dipendenti < soglia_dip)
        {
            struct Azienda* to_del = cursore;
            if (prec == NULL)
                *pp_head = to_del->successivo;
            else
                prec->successivo = to_del->successivo;

            cursore = to_del->successivo;
            free(to_del);
        }
        else
        {
             prec   = cursore;
            cursore = cursore->successivo;

        }

    }

}


int write_sorted_on_file(char filename[64], struct  Azienda** pp_head)
{
    FILE* fp = fopen(filename, "w");
    if(fp == NULL)
        return ERRFILE;

    struct Azienda* p_prec_max = NULL; // puntatore al massimo trovato nell'iterazione precedente
    int max_found              = 1; // 1: se è stato trovato un massimo che rispetta le condizioni
                                    // 0: altrimenti (ossia tutti gli elementi sono stati inseriti)
    // mentre ci sono ancora valori da inserire nel file
    while(max_found == 1)
    {

        // cerco l'azienda con il maggior numero di dipendenti
        struct Azienda* cursore  =  *pp_head;
        struct Azienda* p_max    =  NULL;

        max_found                = 0;
        while( cursore !=  NULL )
        {
            // se è la prima ricerca del massimo oppure
            // se il valore in esame è minore del massimo trovato in precedenza
            if (p_prec_max == NULL || cursore->Dipendenti < p_prec_max->Dipendenti)
            {
                // se è il primo valore oppure se è più grande del massimo trovato fino ad ora
                if(p_max == NULL || cursore->Dipendenti > p_max->Dipendenti )
                {
                    p_max      = cursore;
                    max_found  = 1;
                }
```

```
            }
            cursore = cursore->successivo;
        }
        // scrivo l'azienda nel file
        if(p_max != NULL)
        {
            fprintf(fp,"%s %d %d %d\n",
                   p_max->NomeAzienda, p_max->Dipendenti,
                   p_max->Dipartimenti, p_max->Sedi);
            // cerco per eventuali valori uguali al massimo da scrivere nel file
            cursore = p_max->successivo;
            while(cursore != NULL)
            {
                if (cursore->Dipendenti == p_max->Dipendenti)
                    fprintf(fp,"%s %d %d %d\n",
                       cursore->NomeAzienda, cursore->Dipendenti,
                       cursore->Dipartimenti, cursore->Sedi);
                cursore = cursore->successivo;
            }
        }
        p_prec_max = p_max;

    }

    fclose(fp);
    return OK;

}

void err_handler(int err)
{
    switch(err)
    {
        case OK:
            printf("OK!\n");
        break;
        case ERRFILE:
            printf("errore nella gestione del file!\n");
            exit(1);
        break;
        default:
            printf("errore non gestito!\n");
            exit(1);
    }

}


int main()
{
    struct Azienda* p_head = NULL;

    int err;
    printf("leggo dal file...");
    err = read_from_file("aziende.txt", &p_head);
    err_handler(err);
    printf("prima della rimozione:\n");
    print_list(p_head);
    remove_less_than(&p_head, 300);
    printf("dopo la rimozione:\n");
    print_list(p_head);
    printf("salvo...");
    err = write_sorted_on_file("aziende_new.txt", &p_head);
    err_handler(err);
    printf("lista dopo il salvataggio:\n");
    print_list(p_head);
    free_list(p_head);
    p_head = NULL;

    return 0;
}
```

```
leggo dal file...OK!
prima della rimozione:
NOME: Antergos2, DIPENDENTI: 470, DIPARTIMENTI: 1, SEDI: 4
NOME: Ubuntu, DIPENDENTI: 230, DIPARTIMENTI: 27, SEDI: 4
NOME: Fedora, DIPENDENTI: 580, DIPARTIMENTI: 22, SEDI: 16
NOME: Fedora2, DIPENDENTI: 580, DIPARTIMENTI: 10, SEDI: 40
NOME: Manjaro, DIPENDENTI: 410, DIPARTIMENTI: 24, SEDI: 12
NOME: Mint, DIPENDENTI: 320, DIPARTIMENTI: 17, SEDI: 8
NOME: Antergos, DIPENDENTI: 470, DIPARTIMENTI: 28, SEDI: 13
NOME: Arch, DIPENDENTI: 290, DIPARTIMENTI: 21, SEDI: 7
NOME: CentOS, DIPENDENTI: 440, DIPARTIMENTI: 33, SEDI: 12
NOME: Kali, DIPENDENTI: 240, DIPARTIMENTI: 35, SEDI: 5
NOME: Manjaro2, DIPENDENTI: 410, DIPARTIMENTI: 1, SEDI: 2
dopo la rimozione:
NOME: Antergos2, DIPENDENTI: 470, DIPARTIMENTI: 1, SEDI: 4
NOME: Fedora, DIPENDENTI: 580, DIPARTIMENTI: 22, SEDI: 16
NOME: Fedora2, DIPENDENTI: 580, DIPARTIMENTI: 10, SEDI: 40
NOME: Manjaro, DIPENDENTI: 410, DIPARTIMENTI: 24, SEDI: 12
NOME: Mint, DIPENDENTI: 320, DIPARTIMENTI: 17, SEDI: 8
NOME: Antergos, DIPENDENTI: 470, DIPARTIMENTI: 28, SEDI: 13
NOME: CentOS, DIPENDENTI: 440, DIPARTIMENTI: 33, SEDI: 12
NOME: Manjaro2, DIPENDENTI: 410, DIPARTIMENTI: 1, SEDI: 2
salvo...OK!
lista dopo il salvataggio:
NOME: Antergos2, DIPENDENTI: 470, DIPARTIMENTI: 1, SEDI: 4
NOME: Fedora, DIPENDENTI: 580, DIPARTIMENTI: 22, SEDI: 16
NOME: Fedora2, DIPENDENTI: 580, DIPARTIMENTI: 10, SEDI: 40
NOME: Manjaro, DIPENDENTI: 410, DIPARTIMENTI: 24, SEDI: 12
NOME: Mint, DIPENDENTI: 320, DIPARTIMENTI: 17, SEDI: 8
NOME: Antergos, DIPENDENTI: 470, DIPARTIMENTI: 28, SEDI: 13
NOME: CentOS, DIPENDENTI: 440, DIPARTIMENTI: 33, SEDI: 12
NOME: Manjaro2, DIPENDENTI: 410, DIPARTIMENTI: 1, SEDI: 2
```

In [5]:

```
// strategia 3
// - inserimento in lista iterativo
// - non distruttiva della struttura dati durante l'inserimento ordinato
// - utilizzo di un flag di supporto nella struttura

#include <stdio.h>

#define ERRFILE -1
#define OK 0
struct Azienda
{
    char NomeAzienda[64];
    int Dipendenti;
    int Dipartimenti;
    int Sedi;

    struct Azienda* successivo;

    // definisco un flag ad-hoc per il file in cui scrivere
    // i valori possibili saranno:
    // 1: se l'elemento è già stato inserito nel file
    // 0: se l'elemento non è ancora stato inserito nel file
    int flag;

};

void init_azienda(struct Azienda* p_az)
{
    p_az->flag = 0;
}

void print_list(struct Azienda* p_head)
{
    if (p_head == NULL)
        return;
    printf("NOME: %s, DIPENDENTI: %d, DIPARTIMENTI: %d, SEDI: %d\n",
           p_head->NomeAzienda, p_head->Dipendenti,
           p_head->Dipartimenti, p_head->Sedi);
    print_list(p_head->successivo);

}

int length_list(struct  Azienda* p_head)
{
    int n = 0;
    while(p_head != NULL)
    {
        p_head = p_head -> successivo;
        n++;
    }
    return n;
}

void free_list(struct  Azienda* p_head)
{
    struct  Azienda* tmp;
```

```
    while(p_head != NULL)
    {
        tmp = p_head->successivo;
        free(p_head);
        p_head = tmp;

    }

}

int read_from_file(char filename[64],struct  Azienda** pp_head)
{
  FILE* fp = fopen(filename, "r");
  if (fp == NULL)
      return ERRFILE;

  // creo struttura temporanea
  struct Azienda tmp;
  init_azienda(&tmp);
  // leggo una tupla di valori dal file
  while(fscanf(fp,"%s %d %d %d",
               tmp.NomeAzienda, &(tmp.Dipendenti),
               &(tmp.Dipartimenti), &(tmp.Sedi)) == 4)
  {
      // 1. creo nuovo nodo
      struct Azienda* p_new = malloc(sizeof(struct Azienda));
      *p_new                = tmp; // copio i valori presi dal file nella nuova struct
      p_new->successivo     = NULL;

      // 2. inserisco il nodo nella lista
      if(*pp_head == NULL)
          *pp_head = p_new;
      else
      {
          struct Azienda* cursore = *pp_head;
          while(cursore->successivo != NULL)
              cursore = cursore->successivo;
          cursore->successivo  = p_new;
      }
  }
  // 3. chiudo il file
  fclose(fp);

  return OK;
}

void remove_less_than(struct  Azienda** pp_head, int soglia_dip)
{
    struct Azienda* cursore = *pp_head;
    struct Azienda* prec    = NULL;
    while(cursore != NULL)
    {
        if(cursore->Dipendenti < soglia_dip)
        {
            struct Azienda* to_del = cursore;
            if (prec == NULL)
                *pp_head = to_del->successivo;
            else
                prec->successivo = to_del->successivo;

            cursore = to_del->successivo;
            free(to_del);
        }
        else
        {
             prec   = cursore;
            cursore = cursore->successivo;
        }
    }
}


int write_sorted_on_file(char filename[64], struct  Azienda* p_head)
{
    FILE* fp = fopen(filename, "w");
    if(fp == NULL)
        return ERRFILE;

    int n      = length_list(p_head);

    for(int i = 0; i < n; i++)
    {

        // cerco l'azienda con il maggior numero di dipendenti
        // non ancora inserita nel file
        struct Azienda* cursore =  p_head;
        struct Azienda* p_max   =  NULL;
        while( cursore !=  NULL )
        {
```

```
            if(cursore->flag == 0)
            {
                if( p_max == NULL || cursore->Dipendenti > p_max->Dipendenti)
                {
                    p_max = cursore;
                }
            }
            cursore = cursore->successivo;
        }
        if(p_max != NULL)
        {
            fprintf(fp,"%s %d %d %d\n",
               p_max->NomeAzienda, p_max->Dipendenti,
               p_max->Dipartimenti, p_max->Sedi);
            p_max->flag = 1;
        }

    }

    fclose(fp);
    return OK;

}

void err_handler(int err)
{
    switch(err)
    {
        case OK:
            printf("OK!\n");
        break;
        case ERRFILE:
            printf("errore nella gestione del file!\n");
            exit(1);
        break;
        default:
            printf("errore non gestito!\n");
            exit(1);
    }

}


int main()
{
    struct Azienda* p_head = NULL;

    int err;
    printf("leggo dal file...");
    err = read_from_file("aziende.txt", &p_head);
    err_handler(err);
    printf("prima della rimozione:\n");
    print_list(p_head);
    remove_less_than(&p_head, 300);
    printf("dopo la rimozione:\n");
    print_list(p_head);
    printf("salvo...");
    err = write_sorted_on_file("aziende_new.txt", p_head);
    err_handler(err);
    printf("lista dopo il salvataggio:\n");
    print_list(p_head);
    free_list(p_head);
    p_head = NULL;


    return 0;
}
```

```
leggo dal file...OK!
prima della rimozione:
NOME: Antergos2, DIPENDENTI: 470, DIPARTIMENTI: 1, SEDI: 4
NOME: Ubuntu, DIPENDENTI: 230, DIPARTIMENTI: 27, SEDI: 4
NOME: Fedora, DIPENDENTI: 580, DIPARTIMENTI: 22, SEDI: 16
NOME: Fedora2, DIPENDENTI: 580, DIPARTIMENTI: 10, SEDI: 40
NOME: Manjaro, DIPENDENTI: 410, DIPARTIMENTI: 24, SEDI: 12
NOME: Mint, DIPENDENTI: 320, DIPARTIMENTI: 17, SEDI: 8
NOME: Antergos, DIPENDENTI: 470, DIPARTIMENTI: 28, SEDI: 13
NOME: Arch, DIPENDENTI: 290, DIPARTIMENTI: 21, SEDI: 7
NOME: CentOS, DIPENDENTI: 440, DIPARTIMENTI: 33, SEDI: 12
NOME: Kali, DIPENDENTI: 240, DIPARTIMENTI: 35, SEDI: 5
NOME: Manjaro2, DIPENDENTI: 410, DIPARTIMENTI: 1, SEDI: 2
dopo la rimozione:
NOME: Antergos2, DIPENDENTI: 470, DIPARTIMENTI: 1, SEDI: 4
NOME: Fedora, DIPENDENTI: 580, DIPARTIMENTI: 22, SEDI: 16
NOME: Fedora2, DIPENDENTI: 580, DIPARTIMENTI: 10, SEDI: 40
NOME: Manjaro, DIPENDENTI: 410, DIPARTIMENTI: 24, SEDI: 12
NOME: Mint, DIPENDENTI: 320, DIPARTIMENTI: 17, SEDI: 8
NOME: Antergos, DIPENDENTI: 470, DIPARTIMENTI: 28, SEDI: 13
NOME: CentOS, DIPENDENTI: 440, DIPARTIMENTI: 33, SEDI: 12
NOME: Manjaro2, DIPENDENTI: 410, DIPARTIMENTI: 1, SEDI: 2
salvo...OK!
lista dopo il salvataggio:
NOME: Antergos2, DIPENDENTI: 470, DIPARTIMENTI: 1, SEDI: 4
NOME: Fedora, DIPENDENTI: 580, DIPARTIMENTI: 22, SEDI: 16
NOME: Fedora2, DIPENDENTI: 580, DIPARTIMENTI: 10, SEDI: 40
NOME: Manjaro, DIPENDENTI: 410, DIPARTIMENTI: 24, SEDI: 12
NOME: Mint, DIPENDENTI: 320, DIPARTIMENTI: 17, SEDI: 8
NOME: Antergos, DIPENDENTI: 470, DIPARTIMENTI: 28, SEDI: 13
NOME: CentOS, DIPENDENTI: 440, DIPARTIMENTI: 33, SEDI: 12
NOME: Manjaro2, DIPENDENTI: 410, DIPARTIMENTI: 1, SEDI: 2
```

In [8]:

```
// strategia 4
// - inserimento in lista iterativo
// - non distruttiva della struttura dati durante l'inserimento ordinato
// - utilizzo di un vettore di flag di appoggio

#include <stdio.h>

#define ERRFILE -1
#define OK 0
struct Azienda
{
    char NomeAzienda[64];
    int Dipendenti;
    int Dipartimenti;
    int Sedi;

    struct Azienda* successivo;

};


void print_list(struct Azienda* p_head)
{
    if (p_head == NULL)
        return;
    printf("NOME: %s, DIPENDENTI: %d, DIPARTIMENTI: %d, SEDI: %d\n",
           p_head->NomeAzienda, p_head->Dipendenti,
           p_head->Dipartimenti, p_head->Sedi);
    print_list(p_head->successivo);

}

int length_list(struct  Azienda* p_head)
{
    int n = 0;
    while(p_head != NULL)
    {
        p_head = p_head -> successivo;
        n++;
    }
    return n;
}

void free_list(struct  Azienda* p_head)
{
    struct  Azienda* tmp;
    while(p_head != NULL)
    {
        tmp = p_head->successivo;
        free(p_head);
        p_head = tmp;

    }

}
```

```
int read_from_file(char filename[64],struct  Azienda** pp_head)
{
  FILE* fp = fopen(filename, "r");
  if (fp == NULL)
      return ERRFILE;

  // creo struttura temporanea
  struct Azienda tmp;
  // leggo una tupla di valori dal file
  while(fscanf(fp,"%s %d %d %d",
               tmp.NomeAzienda, &(tmp.Dipendenti),
               &(tmp.Dipartimenti), &(tmp.Sedi)) == 4)
  {
      // 1. creo nuovo nodo
      struct Azienda* p_new = malloc(sizeof(struct Azienda));
      *p_new                = tmp; // copio i valori presi dal file nella nuova struct
      p_new->successivo     = NULL;

      // 2. inserisco il nodo nella lista
      if(*pp_head == NULL)
          *pp_head = p_new;
      else
      {
          struct Azienda* cursore = *pp_head;
          while(cursore->successivo != NULL)
              cursore = cursore->successivo;
          cursore->successivo  = p_new;
      }
  }
  // 3. chiudo il file
  fclose(fp);

  return OK;
}

void remove_less_than(struct  Azienda** pp_head, int soglia_dip)
{
    struct Azienda* cursore = *pp_head;
    struct Azienda* prec    = NULL;
    while(cursore != NULL)
    {
        if(cursore->Dipendenti < soglia_dip)
        {
            struct Azienda* to_del = cursore;
            if (prec == NULL)
                *pp_head = to_del->successivo;
            else
                prec->successivo = to_del->successivo;

            cursore = to_del->successivo;
            free(to_del);
        }
        else
        {
             prec   = cursore;
            cursore = cursore->successivo;

        }

    }

}


int write_sorted_on_file(char filename[64], struct  Azienda* p_head)
{
    FILE* fp = fopen(filename, "w");
    if(fp == NULL)
        return ERRFILE;
    // definisco un vettore di flag lungo quanto la lista
    // i valori possibili di ogni posizione i-esima saranno:
    // 1: se l'elemento i-esimo della lista è già stato inserito nel file
    // 0: se l'elemento i-esimo della lista non è ancora stato inserito nel file
    int n      = length_list(p_head);
    int* flags = calloc(n,sizeof(int)); //calloc mi garantisce che siano inizializzati a 0


    for(int i = 0; i < n; i++)
    {

        // cerco l'azienda con il maggior numero di dipendenti
        // non ancora inserita nel file
        struct Azienda* cursore =  p_head;
        struct Azienda* p_max   =  NULL;
        int idx_list            = 0;
        int idx_max             = -1;
        while( cursore !=  NULL )
        {
            if(flags[idx_list] == 0)
```

```
            {
                if(p_max == NULL || cursore->Dipendenti > p_max->Dipendenti)
                {
                    p_max = cursore;
                    idx_max = idx_list;
                }
            }
            idx_list++;
            cursore = cursore->successivo;
        }

        if(p_max != NULL)
        {
            fprintf(fp,"%s %d %d %d\n",
              p_max->NomeAzienda, p_max->Dipendenti,
              p_max->Dipartimenti, p_max->Sedi);
            flags[idx_max] = 1;
        }

    }

    free(flags);
    fclose(fp);

    return OK;

}

void err_handler(int err)
{
    switch(err)
    {
        case OK:
            printf("OK!\n");
        break;
        case ERRFILE:
            printf("errore nella gestione del file!\n");
            exit(1);
        break;
        default:
            printf("errore non gestito!\n");
            exit(1);
    }

}


int main()
{
    struct Azienda* p_head = NULL;

    int err;
    printf("leggo dal file...");
    err = read_from_file("aziende.txt", &p_head);
    err_handler(err);
    printf("prima della rimozione:\n");
    print_list(p_head);
    remove_less_than(&p_head, 300);
    printf("dopo la rimozione:\n");
    print_list(p_head);
    printf("salvo...");
    err = write_sorted_on_file("aziende_new.txt", p_head);
    err_handler(err);
    printf("lista dopo il salvataggio:\n");
    print_list(p_head);

    free_list(p_head);
    p_head = NULL;

    return 0;
}
```

```
leggo dal file...OK!
prima della rimozione:
NOME: Antergos2, DIPENDENTI: 470, DIPARTIMENTI: 1, SEDI: 4
NOME: Ubuntu, DIPENDENTI: 230, DIPARTIMENTI: 27, SEDI: 4
NOME: Fedora, DIPENDENTI: 580, DIPARTIMENTI: 22, SEDI: 16
NOME: Fedora2, DIPENDENTI: 580, DIPARTIMENTI: 10, SEDI: 40
NOME: Manjaro, DIPENDENTI: 410, DIPARTIMENTI: 24, SEDI: 12
NOME: Mint, DIPENDENTI: 320, DIPARTIMENTI: 17, SEDI: 8
NOME: Antergos, DIPENDENTI: 470, DIPARTIMENTI: 28, SEDI: 13
NOME: Arch, DIPENDENTI: 290, DIPARTIMENTI: 21, SEDI: 7
NOME: CentOS, DIPENDENTI: 440, DIPARTIMENTI: 33, SEDI: 12
NOME: Kali, DIPENDENTI: 240, DIPARTIMENTI: 35, SEDI: 5
NOME: Manjaro2, DIPENDENTI: 410, DIPARTIMENTI: 1, SEDI: 2
dopo la rimozione:
NOME: Antergos2, DIPENDENTI: 470, DIPARTIMENTI: 1, SEDI: 4
NOME: Fedora, DIPENDENTI: 580, DIPARTIMENTI: 22, SEDI: 16
NOME: Fedora2, DIPENDENTI: 580, DIPARTIMENTI: 10, SEDI: 40
NOME: Manjaro, DIPENDENTI: 410, DIPARTIMENTI: 24, SEDI: 12
NOME: Mint, DIPENDENTI: 320, DIPARTIMENTI: 17, SEDI: 8
NOME: Antergos, DIPENDENTI: 470, DIPARTIMENTI: 28, SEDI: 13
NOME: CentOS, DIPENDENTI: 440, DIPARTIMENTI: 33, SEDI: 12
NOME: Manjaro2, DIPENDENTI: 410, DIPARTIMENTI: 1, SEDI: 2
salvo...OK!
lista dopo il salvataggio:
NOME: Antergos2, DIPENDENTI: 470, DIPARTIMENTI: 1, SEDI: 4
NOME: Fedora, DIPENDENTI: 580, DIPARTIMENTI: 22, SEDI: 16
NOME: Fedora2, DIPENDENTI: 580, DIPARTIMENTI: 10, SEDI: 40
NOME: Manjaro, DIPENDENTI: 410, DIPARTIMENTI: 24, SEDI: 12
NOME: Mint, DIPENDENTI: 320, DIPARTIMENTI: 17, SEDI: 8
NOME: Antergos, DIPENDENTI: 470, DIPARTIMENTI: 28, SEDI: 13
NOME: CentOS, DIPENDENTI: 440, DIPARTIMENTI: 33, SEDI: 12
NOME: Manjaro2, DIPENDENTI: 410, DIPARTIMENTI: 1, SEDI: 2
```

In [ ]:

Laboratorio di Programmazione Gr. 3 (N-Z)

Corso di Laurea in Informatica

Università degli Studi di Napoli Federico II

A.A. 2022/23

A. Apicella

ESERCITAZIONE 1

VARIABILI FLOAT/DOUBLE

1. date base e altezza da input, scrivere un programma che calcoli e stampi l'area ed il perimetro di un rettangolo
2. date base e altezza da input, scrivere un programma che calcoli e stampi l'area di un triangolo
3. dato il raggio da input, scrivere un programma che calcoli e stampi area e perimetro di un cerchio
4. fare un programma che calcoli e stampi la media di 4 numeri interi dati dall'utente
5. fare un programma che calcoli e stampi la media pesata di 4 numeri interi e 4 pesi reali dati dall'utente
6. fare un programma che calcoli e stampi il risultato scambio del contenuto di due variabili, stamparle prima e dopo lo scambio (ossia, date due variabili `a` e `b`, "spostare" in `a` il contenuto della variabile `b` ed in `b` il contenuto della variabile `a`)
7. date le coordinate $P_1 = (x_1, y_1)$ e $P_2 = (x_2, y_2)$ di due punti, fare un programma che ne calcoli la distanza euclidea (si ricorda che la distanza euclidea tra due punti è pari a $d(P_1, P_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
8. dati in input i coefficienti m e q ed un valore y, fare un programma che risolva la corrispondente equazione di primo grado in forma normale (ossia, considerando l'equazione $y = mx + q$, dati i valori di m,q ed y, calcolare stampare il valore di x)
9. dati i coefficienti in tre variabili a,b,c in input ed un valore x, risolvere un'equazione di secondo grado in forma normale (ossia, considerando l'equazione $ax^2 + bx + c = 0$, dati i valori di a, b, c in input, calcolare e stampare i corrispondenti valori di x)

SELEZIONI

1. fare un programa che stampi se un numero dato in input è pari o dispari (se è dispari stampare `DISPARI`, se è pari stampare `PARI`)
2. fare un programa che, dati in input 3 valori a,b ed x, stampi se x è compreso tra a e b o meno (estremi inclusi)
3. fare un programa che, dati da tastiera 3 valori dall'utente a,b ed x, stampi se x è compreso tra a e b o meno (estremi esclusi)
4. fare un programa che, dato da tastiera un numero reale, ne calcoli e ne stampi il valore assoluto
5. fare un programa che calcoli e stampi il massimo tra 3 valori dati dall'utente in 3 variabili distinte
6. fare un programa che, dato un voto di esame da tastiera, stampi `promosso` se maggiore o uguale di 18, `bocciato` in caso contrario
7. dati da tastiera 3 valori a,b e op, stampare:
   - la somma di a e b se op è 1;
   - la differenza tra a e b se op è 2;
   - il prodotto tra a e b se op è 3;
   - il quoziente tra a e b se op è 4.

VARIABILI+CICLI (NB: NON USARE VETTORI)

1. Stampare con un ciclo i numeri da 1 a 10
2. Stampare i numeri da 1 a 10 separati da uno spazio
3. Stampare i numeri pari da 1 a 10 in senso crescente
4. Stampare i numeri pari da 1 a 10 in modo decrescente
5. stampare i numeri da 1 a 50 separati da uno spazio e andando a capo ogni 10
6. usando un solo ciclo, stampare i numeri pari da 1 a 50 andando a capo ogni 5
7. dato un input n, stampa di una striscia di n "#" (es. se l'utente immette 5, vorrò veder stampato `#####`)
8. calcolo potenza di un numero (dati due numeri a e b, calcolare a^b)
9. calcolo fattoriale di un numero (dato un numero a, calcolare $a!$)
10. Inserendo 10 numeri interi, calcolare e stampare quanti numeri pari sono stati inseriti
11. calcolo media di n valori inseriti da tastiera (con n inserito da input, NB: non usare array)
12. calcolo media di n valori inseriti da tastiera ma solo dei pari (con n inserito da input, NB: non usare array)
13. Senza utilizzare array, date le altezze di N individui (con N inserito da tastiera), calcolare:
    A. l'altezza media,

   B. il numero di persone più alte di m. 1,80,
   C. il numero di persone più basse di m. 1,65.
14. calcolare la media di una serie di valori inseriti da tastiera (serie terminante con 0)
15. usando un solo ciclo, stampare 5 righe di 10 asterischi
16. ripetere l'esercizio precedente usando due cicli invece di uno
17. dati due input b e h, stampa di un rettangolo fatto di `*` di base b e altezza h
18. dato un input n, stampa di un triangolo Fatto di `*` iniziante con un `*` e terminante con n `*`
19. dato un input n, stampa di un quadrato di `*` di base n avente sulla diagonale principale il carattere `#`
20. dato un input n, stampa di un quadrato di `*` di base n avente sulla diagonale secondaria il carattere `#`
21. calcolo media di serie di valori inseriti da tastiera ma solo dei pari(serie terminante con 0)
22. data una sequenza di 10 numeri, dire se la sequenza inserita è crescente o no (NB: l'output deve essere prodotto alla fine)
23. data una sequenza di 10 numeri, dire se la sequenza inserita è crescente o no (NB: l'output deve essere prodotto alla fine)
24. stampare la media dei numeri interi da 1 a 10
25. stampare la media dei numeri interi tra a e b, con a e b dati dall'utente
26. dato da tastiera un valore n, stampare n `A` seguite da n `B` (es. se l'utente immette 3, stampare `AAABBB`)
27. dato da tastiera due valori n ed m, stampare n `A` seguite da m `B` (es. se l'utente immette 3 e 5, stampare `AAABBBBB`)
28. dato da tastiera un valore n, stampare n `A` alternate ad n `B` (es. se l'utente immette 3, stampare `ABABABA`)
29. dati da tastiera due valori a e b, sommare tutti i numeri pari compresi, eventuali estremi inclusi, tra a e b (es. se l'utente immette 2 e 7, il risultato dovrà essere $2 + 4 + 6 = 12$)

VARIABILI CHAR

1. dato un carattere da tastiera, determinare se è maiuscolo o minuscolo e convertirlo nel rispettivo minuscolo/maiuscolo
2. determinare se un carattere inserito da tastiera è una vocale
3. Stampare tutto l'alfabeto delle maiuscole
4. Stampare l'alfabeto delle minuscole invertito, ossia dalla `z` alla `a`
5. far acquisire all'utente due numeri in due variabili di tipo `float` ed un simbolo in una variabile di tipo `char`; se la variabile di tipo char ha al suo interno il simbolo '+', stampare la somma delle due variabili, se invece ha al suo interno il simbolo '-', stampare la differenza delle due variabili, in tutti gli altri casi stampare "non posso farlo"

In [ ]:

Laboratorio di Programmazione Gr. 3 (N-Z)

Corso di Laurea in Informatica

Università degli Studi di Napoli Federico II

A.A. 2022/23

A. Apicella

ESERCITAZIONE 2

ARRAY MONODIMENSIONALI

1. determinare il minimo (massimo) presente in un vettore e stamparne a video sia valore che la posizione in cui si trova. Esempio: dato `A = 1,3,5,-2,4,0,6` allora stampa `il minimo è -2 e la posizione in cui si trova è 3`
2. stampare la somma di tutti gli elementi in un vettore
3. dato un vettore ed un valore k, ricerca di k all'interno di un vettore (versione NON ottimizzata, usare un `for` (o `while`) che scorre il vettore fino alla fine)
4. dato un vettore stampare `UGUALI` se tutti gli elementi contenuti al suo interno sono tra loro uguali, `NO` se ne esiste anche soltanto uno diverso (versione NON ottimizzata, ossia attraverso un ciclo for (o while) che scorre il vettore fino alla fine)
5. dato un vettore, calcolare e stampare i valori assoluti di tutti gli elementi contenuti al suo interno
6. dato un vettore, calcolare i valori assoluti di tutti gli elementi contenuti al suo interno e inserirli in un altro vettore nelle posizioni corrispondenti. Esempio: dato `A = -1,3,-5,-2,9` allora `RIS = 1,3,5,2,9`
7. dati due vettori di uguale lunghezza, effettuarne la somma elemento per elemento e memorizzazione in un terzo vettore. Esempio: dati `A = 1,2,5,3, B = 2,4,1,2` allora `RIS = 3,6,6,5`
8. dato un vettore, verificare se è ordinato in modo crescente.
9. concatenare due vettori in un terzo vettore. Esempio: dati `A = 1,2,5,3, B = 2,4,1,2` allora `RIS = 1,2,5,3,2,4,1,2`
10. copiare gli elementi pari e dispari di un vettore in 2 vettori differenti
11. copiare gli elementi nelle posizioni pari e dispari di un vettore in 2 vettori differenti
12. dato un vettore e date due variabili contenenti due indici di posizione,scambiare gli elementi nelle posizioni date
13. Dato un vettore, costruire un nuovo vettore uguale al primo rovesciato
14. Dato un vettore, rovesciarlo *in loco*
15. dato un vettore A di numeri interi, creare un secondo vettore B contenente:
    - 1, se il valore nella rispettiva posizione è multiplo di 3 o maggiore di 100,
    - 2, altrimenti
16. dato un vettore A di numeri interi, creare un secondo vettore B contenente solo i valori di A che siano compresi tra 10 e 20
17. dato due vettori di numeri A e B, inserire in una variabile T il valore:

- 1, se i due vettori sono uguali (es. A= [1 4 5] e B= [1 4 5])
- 2, se sono uno il "reverse" dell'altro (es. A= [1 4 5] e B= [5 4 1])
- 3, tutti gli altri casi.

17. dato un vettore `X` , calcolare la somma S di ogni numero elevato alla rispettiva posizione. Esempio: `X = [3 5 2 7]` allora `S=3^1+5^2+2^3+7^4`
18. calcolare le occorrenze di ogni valore all'interno di un vettore (ossia contare quante volte appare ogni elemento di un vettore). Esempio:
    - Vettore: `1 2 2 5 1 5 5 1 2 2 3 1`
    - Risultato: `4 4 4 3 4 3 3 4 4 4 1 4` in quanto: ci sono 4 numeri 1, ci sono 4 numeri 2, ci sono 4 numeri 2, ci sono 3 numeri 5, ci sono 4 numeri 1, ci sono 3 numeri 5 e così via; ogni posizione i-esima del vettore `Risultato` deve contenere il conteggio di quante volte appare in `Vettore` l'elemento in posizione i-esima
19. costruire un vettore che sia l'unione di 2 vettori dati (ossia mettere gli elementi dei due vettori in un terzo vettore senza ripetizioni)
20. costruire un vettore che sia l'intersezione di 2 vettori dati (ossia mettere i soli elementi comuni tra i due vettori in un terzo vettore senza ripetizioni)

ARRAY MONODIMENSIONALI - HARD

1. dato un vettore, stamparne un diagramma a barre fatto di `*` . esempio: vettore: `[ 3 6 4]` la stampa sarà:
   ***
   ******
   ****
2. dato un vettore A ed un vettore B di lunghezza
   $\leq$
   alla lunghezza di A, determinare se il vettore B è un sottovettore di A (ossia se esiste una sequenza di elementi in A uguale all'*intera* sequenza di elementi in B)
3. dato un vettore ed un indice di posizione `idx` , eliminazione dell'elemento in posizione `idx` tramite shift a sinistra
4. dato un vettore ed un indice di posizione `idx` , eliminazione dell'elemento in posizione `idx` tramite shift a destra
5. dato un vettore, eliminarne i duplicati tramite shift
6. dati due vettori `X` ed `Y` contenenti le coordinate di una serie di punti, calcolare una matrice `D` contente le distanze tra questi punti;
7. dato un vettore ed un valore
   k
   , ricerca di
   k
   all'interno di un vettore (ottimizzato, ossia il ciclo si deve fermare quando l'elemento è stato trovato, USARE UN WHILE NON UN FOR)

ARRAY BIDIMENSIONALI

1. effettuare somma di tutti gli elementi di una matrice
2. memorizzare e stamapre le somme di ogni riga di una matrice in un vettore
3. ricerca di un elemento
   k
   dato dall'utente all'interno di una matrice. Se presente, visualizzare l'indice di riga e di colonna in cui si trova
4. copiare la diagonale principale di una matrice in un vettore
5. copiare la diagonale secondaria di una matrice in un vettore
6. sommare due matrici elemento per elemento memorizzandone i valori in una terza matrice
7. data una matrice, scrivere un programma che determini qual è la colonna col maggior numero di elementi pari. Stamparne quindi l'indice
8. data una matrice e due interi `pos1` e `pos2` , scambiare le due righe nelle posizioni corrispondenti
9. copiare il perimetro di una matrice in un vettore
10. data una matrice, inserire gli indici di riga e di colonna di tutti gli elementi negativi in due vettori distinti, il primo contenente le posizioni di riga ed il secondo le posizioni di colonna
11. data una matrice quadrata, dire se la diagonale principale è uguale alla diagonale secondaria

ARRAY BIDIMENSIONALI - HARD

12. verificare se un dato vettore è uguale ad una o più righe di una matrice; se sì, stampare in quali posizioni si trovano tali righe
13. date due matrici `M1` e `M2` , dire, per ogni riga di `M2` , se ne esiste *almeno* una uguale in `M1`
14. date due matrici `M1` e `M2` , dire, per ogni riga di `M2` , se ne esiste *soltanto* una uguale in `M1`
15. Data una matrice di caratteri composta solo di `O` , `X` , `_` (posizione vuota), determinate se tale matrice corrisponde ad una configurazione vincente per il gioco del Tris

Stringhe

1. calcolare la lunghezza di una stringa senza utilizzare la funzione string length (strlen)
2. effettuare la copia di una stringa in un'altra senza utilizzare funzione string copy (strcpy)
3. implementare funzione string compare (strcmp)
4. verificare se una stringa è composta da soli caratteri maiuscoli
5. trasformazione stringa minuscola in maiuscola
6. contare quanti caratteri maiuscoli sono presenti in una stringa
7. dire in che posizione si trova il primo carattere maiuscolo
8. contare quanti vocali e consonanti sono presenti in una stringa
9. verificare se una stringa è palindroma
10. data una stringa e due variabili carattere C1 e C2, sostituire nella stringa tutte le occorrenze di C1 con C2
11. data una stringa S ed un vettore di interi V, inserire in V il numero di occorrenze di ogni carattere di S esempio: `S= [ p i p p o \0 ]` allora `V= [ 3 1 3 3 1 ]`
12. scrivere un programma che stampi video la domanda `cos'è un file .h?` e, se l'utente risponde con una stringa contenente la parola `libreria`, il programma stampa `bocciato`, altrimenti `Ok. Andiamo avanti.`
13. Data una matrice contenente, in ogni riga, una stringa, copiare in un vettore la prima lettera presente in ogni riga dopo averla convertita in maiuscolo
14. Data una stringa s ed un'altra stringa q, verificare se q è una sottostringa di s

Vettori

1. il vettore somma $\mathbf{s} \in \mathrm{R}^N$ tra due vettori $\mathbf{v} \in \mathrm{R}^N$ e $\mathbf{w} \in \mathrm{R}^N$ è definito formalmente come:

   $$s_i = v_i + w_i, \forall_{1 \leq i \leq R}$$

   in altri termini:

   $$\mathbf{s} = (v_1 + w_1, v_2 + w_2, \ldots, v_N + w_n)$$

   scrivere un programma che, dati due vettori, calcoli e visualizzi il vettore somma risultante

2. il prodotto scalare $p \in \mathrm{R}$ tra due vettori $\mathbf{v} \in \mathrm{R}^N$ e $\mathbf{w} \in \mathrm{R}^N$ è definito formalmente come:

   $$p = \sum_{i=1}^{N} v_i \cdot w_i$$

   scrivere un programma che, dati due vettori, calcoli e visualizzi il loro prodotto scalare

Matrici

1. la matrice somma $S \in \mathrm{R}^{R \times C}$ tra due matrici $M, W \in \mathrm{R}^{R \times C}$ è la matrice avente, in ogni posizione, la somma degli elementi nelle posizioni corrispondenti, ossia, $\forall_{1 \leq i \leq R}, \forall_{1 \leq j \leq C}$:

   $$s_{i,j} = m_{i,j} + m_{i,j}$$

   scrivere un programma che, date due matrici, calcoli e visualizzi la loro somma vettoriale
2. Il prodotto tra una matrice $A \in \mathrm{R}^{R \times C}$ ed un vettore $\mathbf{x} \in \mathrm{R}^C$ è definito come il vettore $\mathbf{q} \in \mathrm{R}^R$ tale che, $\forall_{1 \leq i \leq R}$, vale:

   $$q_i = \sum_{j=1}^{C} a_{i,j} \cdot x_j$$

   In altri termini:

   $$\mathbf{q} = \begin{pmatrix} a_{1,1} \cdot x_1 + \cdots + a_{1,C} \cdot x_C \\ a_{2,1} \cdot x_1 + \cdots + a_{2,C} \cdot x_C \\ \vdots \\ a_{R,1} \cdot x_1 + \cdots + a_{R,C} \cdot x_C \end{pmatrix}$$

   Scrivere un programma che, data una matrice ed un vettore, ne calcoli il prodotto

3. Il prodotto tra due matrici $M \in \mathrm{R}^{R \times Q}$ e $W \in \mathrm{R}^{Q \times C}$ è definito come la matrice $P \in \mathrm{R}^{R \times C}$ tale che, $\forall_{1 \leq i \leq R}, \forall_{1 \leq j \leq C}$:

   $$p_{i,j} = \sum_{k=1}^{Q} m_{i,k} \cdot w_{k,j} =$$
   $$= m_{i,1} \cdot w_{1,j} + m_{i,2} \cdot w_{2,j} + m_{i,3} \cdot w_{3,j} + \cdots + m_{i,Q} \cdot w_{Q,j}$$

1. data una matrice $M \in \mathrm{R}^{N \times N}$, copiare in un vettore la sua diagonale principale ed in un altro vettore la sua diagonale secondaria. Si ricorda che:
   - la diagonale principale di una matrice $M \in \mathrm{R}^{N \times N}$ è la sequenza degli elementi $m_{i,j}$ tali che $i = j$. Esempio:

$$M = \begin{pmatrix} m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,1} & m_{3,2} & m_{3,3} \end{pmatrix}$$

   - la diagonale secondaria di una matrice $M \in \mathrm{R}^{N \times N}$ è la sequenza degli elementi $m_{i,j}$ tali che $i + j = N + 1$. Esempio:

$$M = \begin{pmatrix} m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,1} & m_{3,2} & m_{3,3} \end{pmatrix}$$

In [ ]:

Laboratorio di Programmazione Gr. 3 (N-Z)

Corso di Laurea in Informatica

Università degli Studi di Napoli Federico II

A.A. 2021/22

A. Apicella

Matrici & vettori

1. Dati due insiemi A e B, si definisce *prodotto cartesiano* l'insieme delle coppie ordinate $A \times B = \{(a, b) \mid a \in A \wedge b \in B\}$.

rappresentando i due insiemi A e B attraverso due array, implementare una funzione che restituisca il prodotto cartesiano $A \times B$ all'interno di una matrice, i.e. ogni riga rappresenta un elemento del prodotto cartesiano. Esempio:

$$A = (1, 2, 3),\ B = (4, 5, 6) \Rightarrow A \times B = \begin{pmatrix} 1\,4 \\ 1\,5 \\ 1\,6 \\ 2\,4 \\ 2\,5 \\ 2\,6 \\ 3\,4 \\ 3\,5 \\ 3\,6 \end{pmatrix}$$

1. Si generalizzi l'operazione precedente al caso in cui gli insiemi A e B siano rappresentati da due matrici in cui ogni riga corrisponde ad un elemento dell'insieme rappresentato. Esempio:

$$A = \begin{pmatrix} 1\,2 \\ 3\,4 \end{pmatrix} B = \begin{pmatrix} 5\,6 \\ 7\,8 \end{pmatrix} \Rightarrow A \times B = \begin{pmatrix} 1\,2\,5\,6 \\ 1\,2\,7\,8 \\ 3\,4\,5\,6 \\ 3\,4\,7\,8 \end{pmatrix}$$

2. Scrivere una funzione che, data una matrice in input, ne restituisca la sua trasposta in una matrice di output. Si ricorda che, data una matrice M, la sua trasposta M^T è una matrice tale che

$$M_{i,j} = M^T_{j,i},\ \forall_{i,j}$$

3. A. Scrivere una funzione che, data una matrice in input, ne restituisca la sua trasposta *in place*, ossia sovrascrivendo la matrice originaria data in input. NB: non utilizzare matrici "temporanee".

Struct

1. dato un vettore di strutture contenente cognome e età, calcolare l'età media di tutte e sole le persone con più di 30 anni e il cui cognome inizi con la lettera 'B'
2. dato un vettore di strutture contenente `id_esame` (int), `esame` (stringa), `voto` (int) stampare il nome dell'esame con voto minimo ed il nome dell'esame con voto massimo
3. dato un vettore di strutture `Votante` , ognuna delle quali contenente `citta_residenza` , `id_partito_votato` (numero da 0 da 3 identificante un dato partito), costruire un secondo vettore di strutture contenente, per ogni città, l'istogramma dei partiti in base alla città. In altri termini, ogni struttura contenuta nel vettore dovrà contenere a sua volta, oltre alla città di riferimento, un array così composto:

    - l'elemento 0 dovrà contenere quante persone della rispettiva città hanno votato il partito con id 0
    - l'elemento 1 dovrà contenere quante persone della rispettiva città hanno votato il partito con id 1
    - l'elemento 2 dovrà contenere quante persone della rispettiva città hanno votato il partito con id 2
    - l'elemento 3 dovrà contenere quante persone della rispettiva città hanno votato il partito con id 3

stamparne quindi il diagramma a barre con degli *

Puntatori

1. implementare una funzione che accetta un vettore, restituire due variabili, la prima contenente il numero di elementi pari presenti nel vettore e la seconda il numero di elementi dispari (utilizzare il passaggio di parametri tramite puntatore per restituire i risultati)
2. implementare una funzione che, data in input una stringa, restituisca due variabili, la prima contenente la prima consonante presente nell stringa, la seconda variabile contenente l'ultima vocale.
3. implementare due funzioni:

    - `lock(...)` : accetta come parametro una variabile di tipo int; se questa variabile è $\neq 0$,la mette a 0 e restituisce 1 (ossia operazione andata a buon fine); se è ==0 la si lascia a zero e si restituisce 0 (ossia operazione non effettuata perchè, appunto, inutile in quanto la variabile già è a 0)
    - `unlock(...)` : accetta una variabile di tipo int; se questa variabile è ==0, la mette a 1 e si restituisce 1 ; se la variabile è $\neq 0$ la si lascia al valore che ha e si restituisce 0. Esempio di `main()` :

    ```
    int a = 1;
    int r;
    r = lock(&a);
    printf("esito operazione: %d\n", r); // stamperà 1
    r = lock(&a);
    printf("esito operazione: %d\n", r); // stamperà 0
    r = unlock(&a);
    printf("esito operazione: %d\n", r); // stamperà 1
    r = lock(&a);
    printf("esito operazione: %d\n", r); // stamperà 1
    ```

4. Dato in input un vettore **v**, scrivere una funzione che restituisca un vettore **riassunto** di 3 elementi dove:

    - il primo elemento *punta* al minimo di **v**. Se tale valore è presente più volte, si desidera che l'elemento punti alla prima occorrenza
        - il secondo elemento *punta* al massimo di **v**. Se tale valore è presente più volte, si desidera che l'elemento punti all'ultima occorrenza
        - il terzo elemento *punta* al valore più frequente nel vettore **v**. In caso di parità, si punterà all'ultima occorrenza

In [ ]:

Laboratorio di Programmazione Gr. 3 (N-Z)

Corso di Laurea in Informatica

Università degli Studi di Napoli Federico II

A.A. 2021/22

A. Apicella

Black Jack

Fornire una semplice implementazione di un gioco simile al black jack avente come sfidante il computer.

L'utente potrà scegliere ripetutamente tra due possibilità:

1. estrarre una carta
2. fermarsi

nello specifico:

Estrarre una carta

L'estrazione di una carta comporta la produzione di un numero casuale con valore che può andare da 1 a 13. Ad ogni estrazione, tale valore andrà sommato ai valori ottenuti fino a quel momento. La somma di tali valori sarà il punteggio s_u dell'utente. Se, durante le diverse estrazioni, s_u supera il valore 21, la partita termina automaticamente con messaggio per l'utente "hai perso!".

Fermarsi

Tenendo come riferimento il punteggio s_u ottenuto dall'utente, la macchina dovrà provare a battere tale punteggio producendo un punteggio s_m.

La macchina continuerà ad estrarre numeri casuali (tra 1 e 13) finchè il suo punteggio s_m non supera s_u. In altri termini, scopo della macchina è ottenere un punteggio $s_m > s_u$. Se tale punteggio s_m però supera 21 allora la macchina avrà perso, producendo come messaggio all'utente "hai vinto!". Se invece la macchina riesce ad ottenere un punteggio $s_u < s_m < 21$ la macchina avrà vinto, ed il messaggio per l'utente sarà "hai perso!"

Esempio di esecuzione:

```
Punteggio attuale: 0
Scegli:
 1) estrai una carta
 2) mi fermo

 1

Hai scelto di estrarre una carta.
Hai estratto un 2.
Punteggio attuale: 2
Scegli:
 1) estrai una carta
 2) mi fermo

 1

Hai scelto di estrarre una carta.
Hai estratto un 2.
Punteggio attuale: 4
Scegli:
 1) estrai una carta
 2) mi fermo

 1

Hai scelto di estrarre una carta.
Hai estratto un 12.
Punteggio attuale: 16
Scegli:
 1) estrai una carta
```

```
 2) mi fermo

 2

Il tuo punteggio finale è di 16! Turno della macchina!
Ho estratto un 3.
Ho estratto un 2.
Ho estratto un 8.
Ho estratto un 9.
Il mio punteggio finale è 22! Hai vinto!
```

Black Jack v2

Ripetere l'esercizio precedente, ma simulando un *reale* mazzo di 52 carte, composto da 4 gruppi di 13 carte, ogni gruppo appartenente ad un singolo seme (fiori, quadri, cuori, picche). Ad ogni estrazione, deve essere visualizzato sia il valore della carta sia il seme. Tenere conto che una stessa carta non può essere estratta più di una volta.

Consiglio: utilizzare una `struct` per rappresentare ogni singola Carta.

Laboratorio di Programmazione Gr. 3 (N-Z)

Corso di Laurea in Informatica

Università degli Studi di Napoli Federico II

A.A. 2021/22

A. Apicella

L'impiccato

Scrivere un programma che effettui una simulazione del classico gioco dell'*Impiccato* su una frase, così definito:

- viene assegnato all'utente un numero massimo di tentativi con cui può indovinare una frase (e.g., 5)
- viene presentata all'utente una serie di asterischi che indicano la struttura della frase, i.e. da quante lettere ogni parola è composta e quante parole sono e.g. `** *** ******` indica che la frase è costituita da 3 parole rispettivamente di 2, 3 e 6 lettere. Eventuali lettere accentate, apostrofi, segni di punteggiatura e cifre numeriche devono essere visualizzati di default, e.g., la frase "l'anno è bisestile" dovrà essere presentata come `*'**** è *********`

L'utente può quindi scegliere se:

- fornire una lettera: scegliere una lettera (maiuscola o minuscola non deve far differenza), se la lettera è contenuta nella frase, tutte le sue corrispondenze vengono svelate (ossia gli asterischi corrispondenti vengono tolti) altrimenti viene consumato un tentativo.
- provare a dare la soluzione: l'utente può provare a digitare la frase, se la soluzione è corretta il giocatore ha vinto, altrimenti viene consumato un tentativo

Quando il numero dei tentativi raggiunge zero, l'utente ha perso ed il gioco termina.

L'insieme delle frasi possibili è contenuto in un file di testo *dizionario.txt* contenente una sequenza di frasi (e null'altro). Ogni frase deve essere lunga al massimo 255 caratteri. All'inizio del gioco, il programma sceglierà casualmente una frase tra tutte le frasi disponibili.

Esempio di esecuzione:

In [24]:

```
** ***** ** ***********
tentativi residui: 5

scegli:
1) dò una lettera
2) provo a dare una soluzione
0)esci

inserisci una lettera

** *o*** ** *o********o
tentativi residui: 5

scegli:
1) dò una lettera
2) provo a dare una soluzione
0)esci

inserisci una lettera

** *o**e ** *o**e*****o
tentativi residui: 5

scegli:
1) dò una lettera
2) provo a dare una soluzione
0)esci

prova a darmi la soluzione:

hai inserito "il conte di montecristo"
hai vinto!
la frase è Il conte di Montecristo
```

La cassaforte

Si vuole realizzare un programma che simuli una cassaforte i cui dati sono contenuti in un file **binario**

Si vuole realizzare un programma che simuli una cassaforte i cui dati sono contenuti in un file **binario**.

Tale cassaforte sarà costituita da:

- una combinazione (numero intero)
- un ammontare di denaro

All'utente deve essere presentato un menù in cui può scegliere se:

- creare una nuova cassaforte
- aprire la cassaforte
- inserire denaro
- prelevare denaro
- visualizzare il denaro
- cambiare la combinazione
- chiudere la cassaforte
- uscire dal programma

La creazione di una nuova cassaforte distruggerà i dati della precedente e darà la possibilità all'utente di impostare una combinazione.

Tutte le operazioni potranno essere effettuate solo se la cassaforte è aperta. Una volta chiusa la cassaforte, il file binario dovrà essere aggiornato con le nuove informazioni.

Esempio di esecuzione:

In [1]:

```
sceg1i:
1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)
2. apri cassaforte
3. visualizza lo stato della cassaforte
4. visualizza ammontare
5. deposita somma
6. preleva somma
7. chiudi cassaforte
0. esci (eventuali modifiche a cassaforte aperta andranno perse)

scegli la combinazione:

sceg1i:
1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)
2. apri cassaforte
3. visualizza lo stato della cassaforte
4. visualizza ammontare
5. deposita somma
6. preleva somma
7. chiudi cassaforte
0. esci (eventuali modifiche a cassaforte aperta andranno perse)

la cassaforte è chiusa!
sceg1i:
1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)
2. apri cassaforte
3. visualizza lo stato della cassaforte
4. visualizza ammontare
5. deposita somma
6. preleva somma
7. chiudi cassaforte
0. esci (eventuali modifiche a cassaforte aperta andranno perse)

la cassaforte è chiusa!
sceg1i:
1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)
2. apri cassaforte
3. visualizza lo stato della cassaforte
4. visualizza ammontare
5. deposita somma
6. preleva somma
7. chiudi cassaforte
0. esci (eventuali modifiche a cassaforte aperta andranno perse)

inserisci la combinazione:

sceg1i:
1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)
2. apri cassaforte
3. visualizza lo stato della cassaforte
4. visualizza ammontare
5. deposita somma
6. preleva somma
7. chiudi cassaforte
0. esci (eventuali modifiche a cassaforte aperta andranno perse)
```

```
la cassaforte è aperta!
sceg1i:

1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)
2. apri cassaforte
3. visualizza lo stato della cassaforte
4. visualizza ammontare
5. deposita somma
6. preleva somma
7. chiudi cassaforte
0. esci (eventuali modifiche a cassaforte aperta andranno perse)

il tuo denaro ammonta a 0.00
sceg1i:
1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)
2. apri cassaforte
3. visualizza lo stato della cassaforte
4. visualizza ammontare
5. deposita somma
6. preleva somma
7. chiudi cassaforte
0. esci (eventuali modifiche a cassaforte aperta andranno perse)

inserisci la somma da inserire:

sceg1i:
1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)
2. apri cassaforte
3. visualizza lo stato della cassaforte
4. visualizza ammontare
5. deposita somma
6. preleva somma
7. chiudi cassaforte
0. esci (eventuali modifiche a cassaforte aperta andranno perse)

il tuo denaro ammonta a 100.00
sceg1i:
1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)
2. apri cassaforte
3. visualizza lo stato della cassaforte
4. visualizza ammontare
5. deposita somma
6. preleva somma
7. chiudi cassaforte
0. esci (eventuali modifiche a cassaforte aperta andranno perse)

sceg1i:
1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)
2. apri cassaforte
3. visualizza lo stato della cassaforte
4. visualizza ammontare
5. deposita somma
6. preleva somma
7. chiudi cassaforte
0. esci (eventuali modifiche a cassaforte aperta andranno perse)

la cassaforte è chiusa!
sceg1i:
1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)
2. apri cassaforte
3. visualizza lo stato della cassaforte
4. visualizza ammontare
5. deposita somma
6. preleva somma
7. chiudi cassaforte
0. esci (eventuali modifiche a cassaforte aperta andranno perse)

la cassaforte è chiusa!
sceg1i:
1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)
2. apri cassaforte
3. visualizza lo stato della cassaforte
4. visualizza ammontare
5. deposita somma
6. preleva somma
7. chiudi cassaforte
0. esci (eventuali modifiche a cassaforte aperta andranno perse)

inserisci la somma da prelevare:

la cassaforte è chiusa!
sceg1i:
1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)
2. apri cassaforte
3. visualizza lo stato della cassaforte
4. visualizza ammontare
5. deposita somma
6. preleva somma
7. chiudi cassaforte
0. esci (eventuali modifiche a cassaforte aperta andranno perse)
```

```
inserisci la combinazione:

sceg1i:
1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)
2. apri cassaforte
3. visualizza lo stato della cassaforte
4. visualizza ammontare
5. deposita somma
6. preleva somma
7. chiudi cassaforte
0. esci (eventuali modifiche a cassaforte aperta andranno perse)

il tuo denaro ammonta a 100.00
sceg1i:
1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)
2. apri cassaforte
3. visualizza lo stato della cassaforte
4. visualizza ammontare
5. deposita somma
6. preleva somma
7. chiudi cassaforte
0. esci (eventuali modifiche a cassaforte aperta andranno perse)

inserisci la somma da prelevare:

sceg1i:
1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)
2. apri cassaforte
3. visualizza lo stato della cassaforte
4. visualizza ammontare
5. deposita somma
6. preleva somma
7. chiudi cassaforte
0. esci (eventuali modifiche a cassaforte aperta andranno perse)

il tuo denaro ammonta a 75.00
sceg1i:
1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)
2. apri cassaforte
3. visualizza lo stato della cassaforte
4. visualizza ammontare
5. deposita somma
6. preleva somma
7. chiudi cassaforte
0. esci (eventuali modifiche a cassaforte aperta andranno perse)

sceg1i:
1. crea nuova cassaforte (eventuale cassaforte già esistente sarà distrutta)
2. apri cassaforte
3. visualizza lo stato della cassaforte
4. visualizza ammontare
5. deposita somma
6. preleva somma
7. chiudi cassaforte
0. esci (eventuali modifiche a cassaforte aperta andranno perse)
```

Università degli Studi di Napoli Federico II

Corso di Laurea in Informatica

Laboratorio di Programmazione (Gr.3)

Compito del 13/02/2023

Dott. Andrea Apicella

REGOLE

Per questo esame non potete usare il vostro solito account. Per accedere al pc dovete:

1. verificate il nome della macchina, che è la voce in alto a destra, ad esempio *ilc3-44*
2. usate il nome della macchina come nome utente, come password usate *infolab*

Una volta entrati, create una cartella con il vostro nome, cognome e numero di matricola all'interno della cartella *utente* che trovate sul desktop. La cartella *utente* monta sul pc locale la vostra home sul server remoto linux.
Durante la prova, salvate tutti i vostri file all'interno di questa cartella. **Non salvate direttamente sul desktop perché se si spegne la macchina perdete tutto.**
Prima di iniziare materialmente lo svolgimento della traccia, è fortemente consigliato verificare che i file sorgente vengano salvati correttamente dall'editor che scegliete di usare.
Il compito andrà svolto utilizzando il sistema operativo Linux, un editor di testo generico di vostra scelta e il compilatore gcc da linea di comando in una shell.
Una volta che avete finito, la consegna avviene creando un archivio della cartella avente come nome *nome_cognome_numero di matricola* e copiandolo nella cartella *consegna elaborati*. State attenti che **questa operazione può essere fatta solo una volta**.
Non potete ne leggere, ne sovrascrivere, ne rinominare nulla di ciò che c'è nella cartella *consegna elaborati*, quindi non preoccupatevi se non potete accedere al compito una volta che avete consegnato.

TRACCIA DEL 13/02/2023, TEMPO: 2.5 ore

La società *VaCuoncCuonc* gestisce autovelox sparsi sul territorio nazionale e, periodicamente, fornisce alle forze dell'ordine i dati dei veicoli che devono essere sanzionati. I dati di ogni veicolo sono raccolti in strutture dati di nome *Veicolo*, ognuna delle quali è così composta:

- *targa*: campo di testo di al più 10 caratteri alfanumerici;
- *velocita*: campo numerico contenente la velocità rilevata (in km/h) nel tratto di strada controllato;
- *limite*: campo numerico contenente la velocità limite (in km/h) nel tratto di strada controllato.

I dati raccolti sono memorizzati in un unico file di testo così formato:

```
targa1 velocita1 limite1
targa2 velocita2 limite2
.
.
.
```

Esempio di file di input:

```
DYD666 110.0 100.0
S99821 80.0 90.0
BAT1 120.0 50.0
KN1GHT 160.0 130.0
PAP313 95.0 90.0
```

Punto 1: implementare una struttura dati *Report* con politica di accesso *pila* (*Last In First Out*) in grado di memorizzare le informazioni di tutti i veicoli. Lo studente può scegliere se implementare tale coda tramite un array dinamico o una lista concatenata.

Punto 2: scrivere una funzione *load_vehicles_from_file(Report,nomefile)* che carica tutti i dati contenuti nel file di testo di nome $nomefile$ all'interno del *Report*. Il file di testo è fornito con la traccia.

Punto 3: scrivere una funzione *is_crime(Veicolo)* che, dato un veicolo, restituisca 1 se è sanzionabile, 0 altrimenti. Un veicolo è sanzionabile se ha superato il limite tenendo conto di una tolleranza del 5%. Esempi:

velocità rilevata: 103 km/h; limite: 100 km/h; limite con tolleranza = $100 + 5\%(100) = 105$. Non è sanzionabile in quanto $103 \leq 105$.

velocità rilevata: 95 km/h; limite: 90 km/h; limite con tolleranza = $90 + 5\%(90) = 94.5$. E' sanzionabile in quanto $95 > 94.5$.

Punto 4: scrivere una funzione *write_crimes_on_file(Report, nomefile)* che scriva all'interno di *nomefile* le informazioni dei soli veicoli da sanzionare, selezionati attraverso la funzione definita al punto precedente. Tali informazioni saranno:

- la targa;
- di quanto è stato superato il limite;
- di quanto ammonta la sanzione (in euro).

L'ammontare della sanzione deve essere così calcolato:

$$\begin{cases} 173.0 \text{ euro} & \text{se si è superato il limite di al più 10 km/h;} \\ 695.0 \text{ euro} & \text{oltre 10 e fino a 40 km/h in più del limite;} \\ 2170.0 \text{ euro} & \text{oltre 40 e fino a 60 km/h in più del limite;} \\ 3382.0 \text{ euro} & \text{oltre 60 km/h in più del limite.} \end{cases}$$

Esempio di file di output:

```
DYD666 10.0 173.0
BAT1 70.0 3382.0
KN1GHT 30.0 695.0
PAP313 5.0 173.0
```

Punto 5: Provare il tutto in una funzione *main()*. Compilare ed eseguire il programma da linea di comando tramite *gcc*. Riportare su di un file di testo di nome 'istruzioni.txt' i comandi necessari per effettuare la generazione del file eseguibile. La strategia di generazione del file eseguibile è a libera scelta dello studente. Consegnare soltanto i file sorgenti, escludendo i file oggetto ed esegubili dalla consegna.

Università degli Studi di Napoli Federico II

Corso di Laurea in Informatica

Laboratorio di Programmazione (Gr.3)

Compito del 14/03/2023

Dott. Andrea Apicella

REGOLE

Per questo esame non potete usare il vostro solito account. Per accedere al pc dovete:

1. verificate il nome della macchina, che è la voce in alto a destra, ad esempio *ilc3-44*
2. usate il nome della macchina come nome utente, come password usate *infolab*

Una volta entrati, create una cartella con il vostro nome, cognome e numero di matricola all'interno della cartella *utente (Z:)* che trovate sul desktop. La cartella *utente* monta sul pc locale la vostra home sul server remoto linux.
Durante la prova, salvate tutti i vostri file all'interno di questa cartella. **Non salvate direttamente sul desktop perché se si spegne la macchina perdete tutto.**
Prima di iniziare materialmente lo svolgimento della traccia, è fortemente consigliato verificare che i file sorgente vengano salvati correttamente dall'editor che scegliete di usare.
Il compito andrà svolto utilizzando il sistema operativo Linux, un editor di testo generico di vostra scelta e il compilatore gcc da linea di comando in una shell.
Una volta che avete finito, la consegna avviene creando un archivio della cartella avente come nome *nome_cognome_numero di matricola* e copiandolo nella cartella *consegna elaborati*. State attenti che **questa operazione può essere fatta solo una volta.**
Non potete ne leggere, ne sovrascrivere, ne rinominare nulla di ciò che c'è nella cartella *consegna elaborati*, quindi non preoccupatevi se non potete accedere al compito una volta che avete consegnato.

TRACCIA DEL 14/03/2023, TEMPO: 2 ore

Un file di testo contiene tutti i brani musicali ascoltati da un utente nel seguente formato:

```
titolo1
autore1
durata nel formato mm:ss 1
titolo2
autore2
duranta nel formato mm:ss 2
.
.
.
```

Esempio:

```
we will rock you
queen
2:01
it's my life
```

```
bon jovi
3:46
we will rock you
queen
2:01
the show must go on
queen
4:36
```

Da notare che una stessa canzone *può apparire più volte* nel file.

L'utente vuole creare un nuovo file che sia una versione "compressa" del precedente, ma senza perdere informazioni.

Punto 1: Implementare una struttura dati dinamica *Elenco* in cui ogni elemento sia una struttura *Brano* composta dai seguenti campi:

- *titolo*: stringa di massimo 30 caratteri;
- *autore*: stringa di massimo 30 caratteri con iniziale maiuscola;
- *durata_in_sec*: intero contenente la durata del brano espressa in secondi;
- *rips*: numero di ascolti del brano.

Implementare una funzione *load_music_from_file(nome_file, Elenco)* che riempia la struttura dati rispettando i seguenti vincoli:

- ogni canzone deve apparire *una sola volta* nella struttura dati, avente nel campo *rips* il numero di volte effettive in cui è stato ascoltato il brano. Esempio: il brano "we will rock you" avrà *rips* pari a 2 in quanto dal file risulta che è stato ascoltato 2 volte, mentre i brani "it's my life" e "the show must go on" avranno *rips* pari a 1.
- la durata del brano deve essere espressa in secondi complessivi. Esempio: il brano "we will rock you" avrà durata in secondi 121 avendo una durata di 2:01 minuti.

Punto 2: Implementare una funzione *write_music_on_file(nome_file, Elenco)* che scriva all'interno di un file di testo l'elenco generato nel formato:

```
titolo1
autore1
duratainsec1
rips1
titolo2
autore2
duratainsec2
rips2
.
.
.
```

E' preferibile che la funzione sia implementata in maniera ricorsiva.

Punto 3: generare l'eseguibile del programma da linea di comando tramite *gcc*. Riportare su di un file di testo di nome 'istruzioni.txt' i comandi necessari per effettuare la generazione del file eseguibile. Le compilazioni dei file contenenti le funzioni diverse dal *main()* dovranno essere effettuate *separatamente*. La compilazione del file contenente il *main()* dovrà essere effettuata assieme al linking con tutti i rimanenti file oggetto. Il file eseguibile dovrà avere nome **music_numerodimatricola.eseguibile**. Consegnare soltanto i file sorgenti, escludendo i file oggetto ed esegubili dalla consegna.

Università degli Studi di Napoli Federico II

Corso di Laurea in Informatica

Laboratorio di Programmazione (Gr.3)

Compito del 20/06/2023

Dott. Andrea Apicella

REGOLE

Per questo esame non potete usare il vostro solito account. Per accedere al pc dovete:

1. verificate il nome della macchina, che è la voce in alto a destra, ad esempio *ilc3-44*
2. usate il nome della macchina come nome utente, come password usate *infolab*

Una volta entrati, create una cartella con il vostro nome, cognome e numero di matricola all'interno della cartella *utente (Z:)* che trovate sul desktop. La cartella *utente* monta sul pc locale la vostra home sul server remoto linux. Durante la prova, salvate tutti i vostri file all'interno di questa cartella. **Non salvate direttamente sul desktop perché se si spegne la macchina perdete tutto.**
Prima di iniziare materialmente lo svolgimento della traccia, è fortemente consigliato verificare che i file sorgente vengano salvati correttamente dall'editor che scegliete di usare.
Il compito andrà svolto utilizzando preferibilmente il sistema operativo Linux, un editor di testo generico di vostra scelta ed il compilatore *gcc* da linea di comando aperto in una shell.
Una volta che avete finito, la consegna avviene creando un archivio zip della cartella avente come nome *nome_cognome_numero di matricola.zip* e copiandolo nella cartella *consegna elaborati*. State attenti che **questa operazione può essere fatta solo una volta.**
Non potete ne leggere, ne sovrascrivere, ne rinominare nulla di ciò che c'è nella cartella *consegna elaborati*, quindi non preoccupatevi se non potete accedere al compito una volta che avete consegnato.

TRACCIA DEL 20/06/2023, TEMPO: 2.5 ore

Un gioco di ruolo prevede una serie di personaggi con le seguenti caratteristiche:

- *name*: il nome del personaggio
- *hp*: health point, ossia punti salute residui;
- *dp*: defense point, ossia punti difesa;
- *ap*: attack point, ossia punti attacco.

Le caratteristiche di una serie di personaggi sono contenuti all'interno del file di testo "personaggi.txt" nel seguente formato:

```
name1
hp1
dp1
ap1
name2
hp2
dp2
ap2
.
.
.
```

Esempio:

```
elfo
100
50
20
troll
70
30
70
guerriero
80
50
80
```

Punto 1: implementare una struttura dati dinamica in grado di contenere le caratteristiche di una serie personaggi in apposite struct, considerando che ogni personaggio può avere un nome composto da un'unica parola di al più 32 caratteri e che $hp, dp, ap \in \mathbb{N}$. Lo studente può scegliere se implementare tale struttura dati attraverso un array dinamico o una lista concatenata. Per tale struttura dati dovranno essere fornite le seguenti funzioni:

- *insert(Elenco L, Personaggio p)*: inserisce il personaggio p nell'elenco L;
- *print(Elenco L)*: stampa a video tutti nomi dei personaggi dell'elenco L. Fornire una soluzione possibilmente ricorsiva a tale funzione;
- *remove(Elenco L, nomePersonaggio n)*: elimina dall' elenco L il personaggio di nome n. Se l'eliminazione va a buon fine, tale funzione dovrà restituire 1, altrimenti 0.

Implementare quindi una funzione *load(Elenco L, Filename f)* che carichi all'interno di L tutti i dati contenuti nel file di testo f.

Punto 2: Implementare una funzione *colpisci(Personaggio p1, Personaggio p2)* che simuli un colpo da parte di $p1$ verso $p2$. Tale funzione dovrà:

- stampare il nome $p1$ seguito da *colpisce* seguito dal nome di $p2$, assieme ai rispettivi hp residui;
- aggiornare gli hp di $p2$ secondo la seguente formula:

$$hp_{p2} \leftarrow hp_{p2} - \lfloor \frac{ap_{p1}}{dp_{p2}} \rfloor \cdot hp_{p2}$$

Tenere in considerazione che gli hp di un personaggio non possono diventare negativi.

Punto 3: implementare una funzione *sfida(Personaggio p1, Personaggio p2)* che faccia colpire $p2$ da $p1$ e $p1$ da $p2$ in maniera alternata, finchè uno dei due personaggi non finisca i propri hp. Al termine, deve essere stampato il nome del vincitore della sfida.

Punto 4: nella *main()*, visualizzare l'elenco dei personaggi contenuti nel file "elenco.txt" e dare all'utente la possibilità di sceglierne due digitandone i nomi, e quindi far partire una sfida tra i due personaggi. Al termine della sfida, il perdente dovrà essere eliminato dall'elenco.

Punto 5: Compilare ed eseguire il programma da linea di comando tramite *gcc*. Riportare su di un file di testo di nome 'istruzioni.txt' i comandi necessari per effettuare la generazione del file eseguibile. La strategia di generazione del file eseguibile è a libera scelta dello studente. Consegnare soltanto i file sorgenti, escludendo i file oggetto ed eseguibili dalla consegna.

Punto Facoltativo A: Implementare una funzione *write(Elenco L, Filename f)* che salvi all'interno di f tutti i dati contenuti nell'elenco L nello stesso formato del file di input. Invocare tale funzione nel *main()* al termine della sfida in modo da avere nel file 'elenco_aggiornato.txt' i dati aggiornati dei personaggi.

Punto Facoltativo B: Scrivere una funzione che scelga i personaggi coinvolti nello scontro in maniera casuale. Provare ad invocarla all'interno di un *main()* scritto su di un nuovo file.

Università degli Studi di Napoli Federico II

Corso di Laurea in Informatica

Laboratorio di Programmazione (Gr.3)

Compito del 20/07/2023

Dott. Andrea Apicella

REGOLE

Per questo esame non potete usare il vostro solito account. Per accedere al pc dovete:

1. verificate il nome della macchina, che è la voce in alto a destra, ad esempio *ilc3-44*
2. usate il nome della macchina come nome utente, come password usate *infolab*

Una volta entrati, create una cartella con il vostro nome, cognome e numero di matricola all'interno della cartella *utente (Z:)* che trovate sul desktop. La cartella *utente* monta sul pc locale la vostra home sul server remoto linux. Durante la prova, salvate tutti i vostri file all'interno di questa cartella. **Non salvate direttamente sul desktop perché se si spegne la macchina perdete tutto.**
Prima di iniziare materialmente lo svolgimento della traccia, è fortemente consigliato verificare che i file sorgente vengano salvati correttamente dall'editor che scegliete di usare.
Il compito andrà svolto utilizzando preferibilmente il sistema operativo Linux, un editor di testo generico di vostra scelta ed il compilatore *gcc* da linea di comando aperto in una shell.
Una volta che avete finito, la consegna avviene creando un archivio zip della cartella avente come nome *nome_cognome_numero di matricola.zip* e copiandolo nella cartella *consegna elaborati*. State attenti che **questa operazione può essere fatta solo una volta.**
Non potete ne leggere, ne sovrascrivere, ne rinominare nulla di ciò che c'è nella cartella *consegna elaborati*, quindi non preoccupatevi se non potete accedere al compito una volta che avete consegnato.

TRACCIA DEL 20/07/2023, TEMPO: 2.5 ore

Si ricorda che un path assoluto di un file, in linux, è una stringa nel seguente formato:

/dir1/dir2/.../dirn/nomefile.est

esempi di path validi:
/documenti/fileprivati/tracciatrafugata.txt
/miafoto.png
/db/vegeta.jpg
Un path assoluto di un file è quindi ben formato se:

- contiene almeno un carattere '/' e tale carattere è il primo;
- l'ultimo carattere è una lettera.

Il nascente social network *Pezzottangram* contiene le *Foto* degli utenti in specifiche strutture dati così formate:

- *img_path*: path assoluto dell'immagine sul supporto di memorizzazione;
- *didascalia*: messaggio di testo di al più 31 caratteri; tale campo non contiene caratteri di a capo al suo interno;
- *n_like*: numero di utenti che apprezzano la foto;

Un file è un immagine se il suo path assoluto termina con '.bmp', '.jpg', oppure '.png'. Per un singolo utente, le informazioni sono memorizzate in un file di testo nel seguente formato:

```
path1
didascalia1
n_like1
path2
didascalia2
n_like2
.
.
.
```

Esempio:

```
/data/miefoto/fototramonto.png
che bel tramonto!
4
/data/lenottibianche.jpg
ho finalmente finito di leggere questo capolavoro della letteratura russa
1
/data/miefoto/paninomangiatoapranzo.bmp
gnam!
1546
```

Punto 1:: implementare una struttura dati *Bacheca* con politica di accesso *Pila* (*Last In First Out*) in grado di memorizzare delle *Foto*. Tale struttura dovrà fornire almeno le seguenti funzioni:

- *push(Bacheca, Foto)*: inserisce la foto passata come parametro all'interno della bacheca, rispettando la politica di accesso;
- *pop(Bacheca)*: restituisce il collegamento (e rimuove dalla bacheca) la prossima foto;
- *len(Bacheca)*: restituisce il numero di elementi contenuti in bacheca;
- *is_empty(Bacheca)*: restituisce 1 se la bacheca è vuota, 0 altrimenti.

E' a scelta dello studente decidere se tale struttura dati dovrà essere implementata attraverso un array di puntatori a Post allocato dinamicamente o attraverso un array di Post allocato dinamicamente.

Punto 2: scrivere una funzione *is_valid_path(path)* che, dato un path assoluto in input, restituisca 1 se il path è ben formato, 0 altrimenti

Punto 3: scrivere una funzione *load_user_from_file(Bacheca,nomefile)* che carica tutti i dati contenuti nel file di testo di nome *nomefile* all'interno della *Bacheca*. Il file di testo è fornito con la traccia, ed ha nome "utente42.txt". L'accesso alla bacheca deve essere effettuato sfruttando soltanto le funzioni definite al punto 1.

Punto 4: nella funzione *main()*, caricare nella Bacheca solo le foto che hanno path assoluto valido, quindi stampare tutti gli elementi della bacheca uno per volta, dando la possibilità all'utente di scegliere se:

- fermarsi;
- andare alla prossima Foto;
- aggiungere un like alla foto attuale.

L'accesso alla bacheca deve essere effettuato sfruttando soltanto le funzioni definite al punto 1. I post dovranno essere nuovamente presenti all'interno della struttura dati Bacheca al termine dello scorrimento (NB: ricordate che è una pila...). E' consigliato implementare una funzione di supporto *print(Foto)* che stampi il contenuto di *una singola* Foto passata come argomento.

Punto 5: compilare ed eseguire il programma da linea di comando tramite *gcc*. Riportare su di un file di testo di nome 'istruzioni.txt' i comandi necessari per effettuare la generazione del file eseguibile. Le compilazioni dei file contenenti le funzioni diverse dal *main()* dovranno essere effettuate *distintamente*. La compilazione del file contenente il *main()* dovrà essere effettuata assieme al linking con tutti i rimanenti file oggetto. Il file eseguibile dovrà avere nome **utente**_numerodimatricola.**eseguibile**. Consegnare soltanto i file sorgenti, escludendo i file oggetto ed eseguibili dalla consegna.

Punto Facoltativo:: scrivere una funzione *stats_on_file(nomefile, Bacheca)* che stampi su un file di testo 'statistica.txt' un diagramma a barre fatto di asterischi dei like ricevuti da ogni foto. Esempio: se ci sono 3 foto che hanno rispettivamente 5, 4, e 7 like, il diagramma a barre sarà

```
5   *****
4   ****
7   *******
```

Laboratorio di Programmazione Gr. 3 (N-Z)

Corso di Laurea in Informatica

Università degli Studi di Napoli Federico II

A.A. 2021/22

A. Apicella

- Implementare una struttura dati con politica di accesso Coda con relative funzioni di accesso utilizzando come struttura deposito un array allocato dinamicamente. Tale array dovrà modificare la propria dimensione ad ogni inserimento/rimozione di un elemento.

- Implementare una struttura dati con politica di accesso Pila con relative funzioni di accesso utilizzando come struttura deposito un array allocato dinamicamente. Tale array dovrà modificare la propria dimensione ad ogni inserimento/rimozione di un elemento

- Dato un file di testo `nomi.txt` contenente, in ogni riga, una serie di nomi e cognomi nel seguente formato:

  ```
  ASCII
  nome1 cognome1
  nome2 cognome2
  nome3 cognome3
  .
  .
  .
  ```

Produrre un secondo file di testo `nomi_ordinati.txt` contenente i nomi e cognomi *nello stesso formato*, ma ordinati in ordine lessicografico a partire dal cognome.

Esempio:

`nomi.txt`:

```
ASCII
Vittorio Gassman
Michael Douglas
Kirk Douglas
Alessandro Gassman
Gigi Proietti
Carlotta Proietti
```

`nomi_ordinati.txt`:

```
ASCII
Kirk Douglas
Michael Douglas
Alessandro Gassman
Vittorio Gassman
Carlotta Proietti
Gigi Proietti
```

Laboratorio di Programmazione Gr. 3 (N-Z)

Corso di Laurea in Informatica

Università degli Studi di Napoli Federico II

A.A. 2021/22

A. Apicella

- Implementare le seguenti funzioni per una lista concatenata di interi:
- `remove_at(...)`: data in input una lista concatenata ed un indice di posizione `idx`, rimuovere dalla lista l'elemento in posizione `idx`. Si consideri il primo elemento avente posizione `0`. Tale funzione dovrà restituire:
  - `NULL`, se l'elemento in posizione `idx` non esiste
  - l'indirizzo dell'elemento rimosso dalla lista
- `insert_at(...)`: data in input una lista concatenata, un valore `val` ed un indice di posizione `idx`, inserisce l'elemento `val` in posizione `idx`. Si consideri il primo elemento avente posizione `0`. Se `idx` va oltre la lunghezza effettiva della lista, l'elemento dovrà essere inserito alla fine della lista

- Implementare una struttura dati con politica di accesso Coda con relative funzioni di accesso utilizzando come struttura una lista concatenata.

- Implementare una struttura dati con politica di accesso Pila con relative funzioni di accesso utilizzando come struttura una lista concatenata.

- implementare una funzione `insert_sorted(...)` che, data in input una lista di interi ordinata in maniera crescente ed un valore `val`, inserisca `val` in lista in modo che la lista risultante sia ancora ordinata.

In [ ]:

Laboratorio di Programmazione Gr. 3 (N-Z)

Corso di Laurea in Informatica

Università degli Studi di Napoli Federico II

A.A. 2021/22

A. Apicella

- Definire una struttura dati contenente interi avente una politica di accesso del tipo *minimum first out*, in cui il valore più piccolo contenuto è sempre il primo ad uscire. Provarla con un apposito `main(...)` .
- Data una stringa s, verificare se la stringa è palindroma. Gli elementi di s potranno essere letti/scritti solo ed esclusivamente dal primo all'ultimo. Non possono essere utilizzate strutture di supporto diverse da Pile e/o Code.
- dato un numero, stamparne la codifica binaria. Non possono essere utilizzate strutture di supporto diverse da Pile e/o Code.

- Si vuole implementare un gioco che simuli il comportamento di un uomo in fila alle casse di un supermercato. Date due code C_1, C_2 ed una persona u, tale persona dovrà scegliere in quale coda mettersi durante lo scorrere della coda. Scopo del gioco è quello di essere servito (ossia di arrivare ad una delle due casse) prima che l'altra coda si svuoti.
  - Ogni coda può avere al massimo 20 persone.
  - Ogni persona porta con sé nel carrello al massimo 20 prodotti.

Regolamento:

- All'inizio del gioco, l'uomo u vede *quante* persone ci sono nelle due code (ma non quanti prodotti siano presenti nel carrello di ogni persona in coda). Sceglie quindi in quale coda posizionarsi. (La 1 o la 2). Il numero di persone per ogni coda e di prodotti per ogni persona è deciso in maniera casuale.
- Il gioco quindi prosegue a turni.
- Come all'inizio, ad ogni turno, u è a conoscenza di quante persone sono in attesa in ogni coda, ma non di quanti prodotti hanno nel carrello. Ad ogni turno, u potrà scegliere se:

  1. rimanere nella coda in cui si trova
  2. cambiare coda
- Contemporaneamente, ogni cassiere vede il numero di prodotti della persona in testa della rispettiva file, "consumando" (ossia facendo pagare) un singolo prodotto ad ogni turno. Al termine del numero di prodotti, la cassa servirà la persona successiva. Il altri termini, ogni coda è "ferma" per un numero di turni uguale al numero di prodotti del cliente in testa.

Esempio:

In C_1 ci sono in attesa 3 persone, in C_2 ci sono in attesa 5 persone. La persona in testa a C_1 ha 3 prodotti, mentre in testa a C_2 c'è una persona con 7 prodotti. C_1 quindi procederà alla persona successiva dopo 3 turni e C_2 dopo 7 turni. In altri termini, dopo 3 turni, C_1 avrà concluso col cliente in testa passando al successivo, avendo così in coda $3 - 1 = 2$ persone, mentre C_2 per passare al cliente successivo (e quindi avere in attesa $5 - 1 = 4$ persone) dovrà aspettare ancora 4 turni per consumare i prodotti rimanenti del cliente in testa.

Il gioco termina quando una delle due code si svuota. Se u si troverà nella coda svuotata, apparirà un messaggio "hai vinto!", altrimenti "hai perso!"

Suggerimento:

soltanto per vedere se tutto procede correttamente, implementare una ulteriore funzione di stampa della coda che visualizzi anche il numero di prodotti di ogni utente in coda

In [13]:

```
SITUAZIONE CASSA 1:
0 |0 0 0
SITUAZIONE CASSA 2:
0 |0 0 0 0
In quale cassa vuoi andare inizialmente (1 o 2)?
```

```
_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 1
_________________________________
SITUAZIONE CASSA 1:
O |O O O
SITUAZIONE CASSA 2:
O |O O O O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 1
_________________________________
SITUAZIONE CASSA 1:
O |O O O
SITUAZIONE CASSA 2:
O |O O O O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 1
_________________________________
SITUAZIONE CASSA 1:
O |O O O
SITUAZIONE CASSA 2:
O |O O O O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 1
_________________________________
SITUAZIONE CASSA 1:
O |O O O
SITUAZIONE CASSA 2:
O |O O O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 1
_________________________________
SITUAZIONE CASSA 1:
O |O O O
SITUAZIONE CASSA 2:
O |O O O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 1
_________________________________
SITUAZIONE CASSA 1:
O |O O
SITUAZIONE CASSA 2:
O |O O O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 1
_________________________________
SITUAZIONE CASSA 1:
O |O O
SITUAZIONE CASSA 2:
O |O O O
scegli:
1)cambio cassa
2)resto
0)esci
```

```
_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 1
_________________________________
SITUAZIONE CASSA 1:
0 |0 0
SITUAZIONE CASSA 2:
0 |0 0
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 1
_________________________________
SITUAZIONE CASSA 1:
0 |0 0
SITUAZIONE CASSA 2:
0 |0 0
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 1
_________________________________
SITUAZIONE CASSA 1:
0 |0 0
SITUAZIONE CASSA 2:
0 |0 0
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 1
_________________________________
SITUAZIONE CASSA 1:
0 |0 0
SITUAZIONE CASSA 2:
0 |0 0
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 1
_________________________________
SITUAZIONE CASSA 1:
0 |0 0
SITUAZIONE CASSA 2:
0 |0 0
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 1
_________________________________
SITUAZIONE CASSA 1:
0 |0 0
SITUAZIONE CASSA 2:
0 |0 0
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 1
_________________________________
SITUAZIONE CASSA 1:
0 |0 0
SITUAZIONE CASSA 2:
0 |0 0
scegli:
1)cambio cassa
2)resto
0)esci
```

```
_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 1
_________________________________
SITUAZIONE CASSA 1:
O |O O
SITUAZIONE CASSA 2:
O |O O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 1
_________________________________
SITUAZIONE CASSA 1:
O |O
SITUAZIONE CASSA 2:
O |O O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 1
_________________________________
SITUAZIONE CASSA 1:
O |O
SITUAZIONE CASSA 2:
O |O O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 1
_________________________________
SITUAZIONE CASSA 1:
O |O
SITUAZIONE CASSA 2:
O |O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 1
_________________________________
SITUAZIONE CASSA 1:
O |O
SITUAZIONE CASSA 2:
O |O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 1
_________________________________
SITUAZIONE CASSA 1:
O |O
SITUAZIONE CASSA 2:
O |O
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 1
_________________________________
SITUAZIONE CASSA 1:
O |O
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci
```

```
_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |O
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci

_________________________________
TI TROVI ATTUALMENTE ALLA CASSA 2
_________________________________
SITUAZIONE CASSA 1:
O |
SITUAZIONE CASSA 2:
O |
scegli:
1)cambio cassa
2)resto
0)esci

si è svuotata la CASSA 2
Hai vinto!
```

In [ ]:

dato un vettore $\mathbf{v} = (v_1, v_2, \ldots, v_n)$, fornire una soluzione ricorsiva per stamparne i valori. Si consideri come caso base la stampa di un singolo elemento

In [ ]:

```
```

Stampare una matrice di dimensioni $n_r \times n_c$ di interi utilizzando una soluzione ricorsiva. Si consideri come caso base la stampa di una singola riga della matrice.

In [ ]:

```
```

In [ ]:

```
```

Dati due vettori $\mathbf{v}, \mathbf{w} \in \mathrm{R}^n$, fornire una soluzione ricorsiva per il calcolo del loro prodotto scalare

In [14]:

```
```

```
prodotto vw: 30.00
```

data una stringa $\mathbf{s}$, fornire una soluzione ricorsiva per la verifica che $\mathbf{s}$ sia palindroma

In [33]:

```
```

```
La stringa lanssnal è palindroma
```

Dato un intero,stamparne la sua codifica binaria attraverso una funzione ricorsiva che adotta una soluzione ricorsiva

Dato un intero, implementare una funzione ricorsiva che adotta una soluzione ricorsiva che ne restituisca la codifica binaria

Data una matrice quadrata, proporre una soluzione ricorsiva per stamparne la diagonale principale

In [1]:

```
```

```
1, 2, 3,
4, 5, 6,
7, 8, 9,
diagonale: 1, 5, 9,
```

Dato un numero intero ≥ 0, fornire una funzione ricorsiva che ne stampi la sua codifica binaria

Dato un numero intero ≥ 0, fornire una funzione ricorsiva che ne *restituisca* la sua codifica binaria

Ricorsione e liste

Data una lista concatenata L, proporre una soluzione ricorsiva che ne stampi tutti i valori

Data una lista concatenata L, proporre una soluzione ricorsiva che ne stampi tutti i valori dall'ultimo al primo

Data una lista concatenata contenente valori interi, scrivere una funzione ricorsiva che ne elimini tutti i nodi avente valore pari a k, con k fornito dall'utente

Data una lista concatenata in cui ogni nodo contiene un numero di telefono codificato in forma di vettore di interi, eliminare dalla lista tutti i numeri di telefono che non iniziano con prefisso `081` .

Stampare la lista risultante.

In [48]:

```
situazione iniziale:
{2}->{2}->{1}->{2}->{3}->{2}->{2}->{1}->{1}->{2}->
quale valore desideri eliminare?

situazione finale:
{2}->{2}->{1}->{2}->{2}->{2}->{1}->{1}->{2}->
```

In [ ]:

Laboratorio di Programmazione Gr. 3 (N-Z)

Corso di Laurea in Informatica

Università degli Studi di Napoli Federico II

A.A. 2021/22

A. Apicella

Si realizzi un programma in linguaggio C che,

- legga un file di testo dove per ogni riga si ha `NomeAzienda Dipendenti Dipartimenti Sedi`. I dati vanno immagazzinati in una opportuna lista a singolo link; i dati nella lista devono contenere tutti i dati inclusi nel file di input nello stesso ordine del file di input, o al più un ordine inverso; stampare la lista a schermo dopo la lettura.
- successivamente si eliminino i record relativi alle aziende che hanno meno di 300 dipendenti; stampare la lista a schermo al termine dell'operazione;
- riscriva i record su un altro file, seguendo lo stesso formato del file di input, in maniera che le aziende siano ordinate in maniera decrescente secondo il numero di dipedenti (N.B., Non è necessario ordinare la lista).

Per semplicità considerate che i nomi delle aziende, come nell'esempio, siano composte da una sola parola senza spazi.

Esempio:

Supponendo che il file di input contenga

```
ascii
Ubuntu 230 27 4
Fedora 580 22 16
Manjaro 410 24 12
Mint 320 17 8
Antergos 470 28 13
Arch 290 21 7
CentOS 440 33 12
Kali 240 35 5

ascii
Fedora 580 30 16
Antergos 470 30 13
CentOS 440 30 12
Manjaro 410 30 12
Mint 320 30 8
```

In [ ]: