

## ESERCIZIO 1 [LIV.1]

```
/*
[liv.1] Scrivere una function C char low_up(char ch) che cambia il carattere in input da minuscolo
a maiuscolo e viceversa automaticamente
*/
#include <stdio.h>
#include <stdlib.h>

char low_upp(char ch);

int main()
{
    char X;
    printf("Dammi carattere ");
    scanf("%c",&X);
    printf("Il carattere '%c' e' stato trasformato in '%c'", X, low_upp(X));
    return 0;
}

/*
Input: Carattere da convertire in minuscolo o maiuscolo in base ai casi
Output: Carattere convertito
Descrizione: Cambia il carattere in minuscolo a maiuscolo e viceversa
*/
char low_upp(char ch)
{
    /* Per convertire un carattere da maiuscolo a minuscolo e viceversa, basta fare una
     * x-or col 6° bit (32).
     * ESEMPIO: 'A' = 65 01000001 x-or
     *           00100000 (32)
     * dato che il 6° bit e' diverso, viene commutato e si avra' 01100001 ossia 'a' (97)
     * Invece 'a' = 97 01100001 x-or
     *           00100000 (32)
     * Dato che il 6° bit e' uguale, verra' posto 0 da ottenere 01000001, ossia 'A'
    */
    if(ch >= 65 && ch <= 122)
        return ch^32;
    else
        return ch;
}
```

### OUTPUT:

```
Inserisci un carattere:
A
Il carattere 'A' e' stato trasformato in: 'a'

Inserisci un carattere:
z
Il carattere 'z' e' stato trasformato in: 'Z'

Inserisci un carattere:
$
Il carattere '$' e' stato trasformato in: '$'
```

## ESERCIZIO 2 [LIV.1]

```
/***
[liv.1]
Scrivere una function C char rotate(char ch, char n_bit) per ruotare di n bit (n_bit),
verso sinistra o verso destra (rispettivamente per n_bit<0 e per n_bit>0), il
contenuto di una variabile char mediante gli operatori bitwise
*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
unsigned char rotate(unsigned char ch, char n_bit);
void visualbit(unsigned char ch);

int main()
{
    /* Unsigned char, per non avere il complemento a 2 */
    unsigned char X, X_ruotato;
    short n_bit;
    printf("Inserisci un carattere\n");
    scanf("%c",&X);
    printf("\nCarattere inserito:");fflush(stdin);
    visualbit(X);
    printf("\nInserisci n bit da shiftare\n-n bit<0 shifta sinistra\n-n bit>0 shifta a destra\n\n");
    scanf("%hd",&n_bit);fflush(stdin);
    X_ruotato = rotate(X,n_bit);
    printf("\nIl bit ruotato e':");
    visualbit(X_ruotato);

    return 0;
}
/*----- ROTATE -----
Scopo: Ruotare i bit di un char dato di n bit
Input: ch = char da ruotare, n_bit = bit da shiftare
Output: Bit del char ruotati
Descrizione: La funzione ruota i bit del char mediante 2 shift.
Prima di scoprire se lo shift dovrà essere effettuato a destra o a sinistra,
salviamo ch in una variabile temporanea tmp: in ch faremo lo shift
originale o principale, mentre in tmp shiftiamo in opposto i valori da "salvare"
ad 8-n bit.
Infine uniamo e restituiamo al programma chiamante.
Es. 01000001 e n_bit = -2 (shift a sinistra)
      Allora ch = 00000100 (shiftiamo a sinistra) e tmp = 00000001 (ultimi 2 bit
saranno i primi) .
      return 00000101
*/
unsigned char rotate(unsigned char ch, char n_bit)
{
    unsigned char tmp=ch; /* ch: conterrà lo shift principale
                           tmp: shiftiamo in tal modo da ottenere i bit ruotati o "salvati" */
    //Shift destra
    if(n_bit>0)
    {
        ch=ch>>n_bit; //lo shift principale
        tmp=tmp<<(sizeof(ch)*8-n_bit); //oterremo il "salvataggio" degli ultimi n_bit
                                         //sizeof(ch)=1*8 bit
        return ch|tmp; //Uniamo le 2 porzioni
    }
    //Shift sinistra
    else if(n_bit<0)
    {
        //n_bit negativo indica la direzione, ma il suo modulo n_shift
        n_bit=-n_bit;
        //Processo analogo e inverso
        ch=ch<<n_bit;
        tmp=tmp>>(sizeof(ch)*8-n_bit);
        return ch|tmp;
    }
}
/*
per visualizzare gli 8 bit di una variabile char.
Data una stringa di un byte, la spezzetta in 4 bit: sfrutto
l'esadecimale per risalire ai 4 bit.
*/
void visualbit(unsigned char ch)
```

```

{
    /* 16 stringhe possibili, lunghe 5 (0010+\0) */
    unsigned char bit[16][5] = {"0000", "0001", "0010", "0011",
                                "0100", "0101", "0110", "0111",
                                "1000", "1001", "1010", "1011",
                                "1100", "1101", "1110", "1111"
                            };
    unsigned char C, dx, sx;
    C = ch;
    sx = C >>4;
    dx = C <<4; dx = dx >>4;

    printf("\nchar = %c\t dec = %d\t \n", C, C);
    printf("hex = %x\t bin = %s %s \n", C, bit[sx], bit[dx]); //indirizzo base
}

```

### OUTPUT:

Inserisci un carattere  
A

Carattere inserito:  
char = A dec = 65  
hex = 41 bin = 0100 0001

Inserisci n bit da shiftare  
-n bit<0 shifta sinistra  
-n bit>0 shifta a destra

-2

Il bit ruotato e':  
char = @ dec = 5  
hex = 5 bin = 0000 0101

Inserisci un carattere  
A

Carattere inserito:  
char = A dec = 65  
hex = 41 bin = 0100 0001

Inserisci n bit da shiftare  
-n bit<0 shifta sinistra  
-n bit>0 shifta a destra

3

Il bit ruotato e':  
char = < dec = 40  
hex = 28 bin = 0010 1000

### ESERCIZIO 3 [LIV.1]

```
/***
[liv.1]
Scrivere una function C che, dopo aver estratto i bit da una variabile intera X (tipo
char, short o long), ne calcola il relativo valore dalla formula:
Val_X = b[n-1]2^(n-1)+...+b[2]2^2+b[1]2^1+b[0]2^0
dove b e' l'array dei bit di X. Confrontare il risultato con il valore della variabile
X dichiarata una volta signed ed un'altra unsigned.
***/

#include <stdio.h>
#include <stdlib.h>
#include<math.h>
#define MAX_LEN 32
void estrai_bit(int len, char ch[], char bit[]);
void Visualizza_bit(char bit[], int len);
int main()
{
    /* Potenza shift contiene le potenze di 2 */
    int scelta, i, Potenza_Shift;
    char bit[MAX_LEN]; //array bit
    /* Possibili risultati signed ed unsigned
     Se inserisco un unsigned char, non posso visualizzare ad esempio valori > 255 */
    unsigned char u_ch=0;
    unsigned long u_lo=0;
    unsigned short u_sh=0;

    char ch=0;
    long lo=0;
    short sh=0;
    /* 3 tipi diversi che condividono la stessa area di memoria */
    union word32bit
    {
        long l;
        short s[2];
        char c[4];
    }word;

    puts("Che tipo di dato vuoi inserire?:\n");
    puts("1 - intero char");
    puts("2 - intero short");
    puts("3 - intero long");
    scanf("%d",&scelta);
    switch (scelta)
    {
        /*TIPO CHAR:
        Ha a disposizione 1byte (8bit). E' possibile rappresentare 256 valori differenti.
        Signed char possono rappresentare numeri compresi tra [-128,+127].
        Unsigned char possono rappresentare numeri compresi tra [0,255].
        ATTENZIONE: nella variabile signed, un valore >127 e <255 verrà il complemento a 2 del
        numero su 8 bit.
        In un UNSIGNED inserendo 256 ad esempio, verrà preso come 0, cioè ricomincia
        un nuovo intervallo.
        */

        case 1: //IMMISSIONE DI UN CHAR
            printf("\nInserire un intero di tipo char (1 byte): ");
            fflush(stdin);
            //legge un char
            scanf("%d",&(word.c[0]));
            estrai_bit(sizeof(char),word.c,bit); //estrae i bit dal char inserito

            /* Calcolo del valore intero partendo dalla stringa binaria della variabile
            di input, mediante la formula data.
            Viene eseguito un ciclo per 8 volte effettuando la somma tra le potenze
            di 2 successive*/

            /*inserisce il risultato in una variabile SIGNED char*/
            for (i=0, Potenza_Shift=1; i<sizeof(char)*8; i++)
            {
                ch += bit[i]*Potenza_Shift; //ch += (bit[i]*pow(2,i));
                Potenza_Shift = Potenza_Shift<<1; //Aumenta di 1 l'esponente della potenza
            }
            //Val_X = Val_X + bit[i] * potenza del 2
    }
}
```

```

/*inserisce il risultato in una variabile UNSIGNED char*/
for (i=0, Potenza_Shift=1; i<sizeof(char)*8; i++)
{
    u_ch += bit[i]*Potenza_Shift; //ch += (bit[i]*pow(2,i));
    Potenza_Shift = Potenza_Shift<<1; //Aumenta di 1 l'esponente della potenza
}
printf ("\nStringa binaria: ");Visualizza_bit(bit, sizeof(char));
printf ("\nValore binario con signed char: %d",ch);
printf ("\nValore binario con unsigned char: %d",u_ch);
break;

/*TIPO SHORT:
Ha a disposizione 2byte (16bit). E' possibile rappresentare 65536 valori.
Con i signed short si possono rappresentare numeri compresi tra [+32767,-32768]
Anche qui vale il complemento a 2, se si mette [+65535], ossia 16 '1',
si avra' -32168 (2^n-x=-|y|).
Con gli unsigned short si possono rappresentare i numeri fra [0,65536]. */
case 2: //IMMISSIONE DI UNO SHORT
printf ("\nImmettere un intero short: ");
fflush(stdin);
scanf ("%hd",&(word.s[0]));

estrai_bit(sizeof(short),word.c,bit); //estrae i bit dal char inserito
/* Calcolo del valore intero partendo dalla stringa binaria della variabile
di input, mediante la formula data.
Viene eseguito un ciclo per 16 volte effettuando la somma tra le potenze
di 2 successive*/

/*inserisce il risultato in una variabile SIGNED char*/
for (i=0, Potenza_Shift=1; i<sizeof(short)*8; i++)
{
    sh += bit[i]*Potenza_Shift; //ch += (bit[i]*pow(2,i));
    Potenza_Shift = Potenza_Shift<<1; //Aumenta di 1 l'esponente della potenza
}
//Val_X = Val_X + bit[i] * potenza del 2
/*inserisce il risultato in una variabile UNSIGNED char*/
for (i=0, Potenza_Shift=1; i<sizeof(short)*8; i++)
{
    u_sh += bit[i]*Potenza_Shift; //ch += (bit[i]*pow(2,i));
    Potenza_Shift = Potenza_Shift<<1; //Aumenta di 1 l'esponente della potenza
}

printf ("Stringa binaria: ");Visualizza_bit(bit, sizeof(short));
printf ("\nValore binario con signed short: %d",sh);
printf ("\nValore binario con unsigned short: %d",u_sh);
break;

/*TIPO LONG:
Ha a disposizione 4byte (32bit) quindi puÃ² rappresentare 2^32 (4 294 967 296) valori.
Signed long si possono rappresentare numeri tra [(+2^31)-1,-2^31].
Unsigned short si possono rappresentare tra [0,2^32 - 1] valori */
case 3: //IMMISSIONE DI UN LONG
printf ("\nImmettere un valore intero long: ");
scanf ("%ld",&(word.l)); fflush(stdin);
estrai_bit(sizeof(long),word.c,bit);
/* Calcolo del valore intero partendo dalla stringa binaria della variabile
di input, mediante la formula data.
Viene eseguito un ciclo per 8 volte effettuando la somma tra le potenze
di 2 successive*/

/*inserisce il risultato in una variabile SIGNED char*/
for (i=0, Potenza_Shift=1; i<sizeof(long)*8; i++)
{
    lo += bit[i]*Potenza_Shift; //ch += (bit[i]*pow(2,i));
    Potenza_Shift = Potenza_Shift<<1; //Aumenta di 1 l'esponente della potenza
}
//Val_X = Val_X + bit[i] * potenza del 2
/*inserisce il risultato in una variabile UNSIGNED char*/
for (i=0, Potenza_Shift=1; i<sizeof(long)*8; i++)
{
    u_lo += bit[i]*Potenza_Shift; //ch += (bit[i]*pow(2,i));
    Potenza_Shift = Potenza_Shift<<1; //Aumenta di 1 l'esponente della potenza
}
printf ("Stringa binaria: ");Visualizza_bit(bit, sizeof(long));
printf ("\nValore binario con signed long: %d",lo);
printf ("\nValore binario con unsigned long: %d",u_lo);
break;

```

```

        default: break;
    }
    return 0;
}

/* Dal valore long, short o char si passa ad una stringa di bit */
void estrai_bit(int len, char ch[], char bit[])
{
    short int j,jc;
    char c;

    for(j=0; j<MAX_LEN; j++)
        bit[j]=0;

    //Scorri i byte della parola
    for(jc=0; jc<len; jc++)
    {
        //memorizza l'informazione individuata dal jc-esimo byte
        c=ch[jc]; //Ricava cio' che trovi al byte jc
        //scorri i bit del byte preso in esame
        for(j=0; j<8; j++)
        {
            bit[j+8*jc]=c&1; //individuando byte e bit, inserisce il valore del primo bit
            c=c>>1; //sposta di un posto per controllare il bit successivo
        }
    }
}

/* visualizza di una stringa di bit costruita */
void Visualizza_bit(char bit[], int len)
{
    int j;
    for(j=len*8-1;j>=0;j--)
    {
        printf("%d", bit[j]);
        if(j%8==0)printf(" "); //Arrivato ad ogni 8 bit, prenditi uno spazio
    }
}

```

### OUTPUT:

Che tipo di dato vuoi inserire?:

- 1 - intero char
- 2 - intero short
- 3 - intero long

Immettere un intero di tipo char <1 byte>: 255

Stringa binaria: 11111111  
 Valore binario con signed char: -1  
 Valore binario con unsigned char: 255

Che tipo di dato vuoi inserire?:

- 1 - intero char
- 2 - intero short
- 3 - intero long

Immettere un intero short <2 byte>: 32768

Stringa binaria: 10000000 00000000  
 Valore binario con signed short: -32768  
 Valore binario con unsigned short: 32768

Che tipo di dato vuoi inserire?:

- 1 - intero char
- 2 - intero short
- 3 - intero long

Immettere un valore intero long <4 byte>: 100000000001

Stringa binaria: 01001000 01110110 11101000 00000001  
 Valore binario con signed long: 1215752193  
 Valore binario con unsigned long: 1215752193

NB: 100000000001 e' in decimale!

## ESERCIZIO 4 [LIV.1]

```

/* [liv1] Scrivere una function C per estrarre dalla variabile intera X i k bit piÃ¹ significativi
o meno significativi, dove X e k sono i parametri di input, usando:
1) Una maschera.
2) L'operatore di shift (>> o <<).
3) Il prodotto o la divisione per potenza 2. */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define len 8

unsigned char Maschera(unsigned char x, int scelta, int k);
unsigned char Shift(unsigned char x, int scelta, int k);
unsigned char Prodotto_divisione(unsigned char x, int scelta, int k);
void bit_show(unsigned char n);

int main()
{
    int scelta, k; // k = n_bit maschera
    unsigned char x, Mask_estratta, Shift_estratto, Pro_div_2_estratto;
    //Input numero x
    printf("Inserire un numero: ");
    scanf("%d", &x);
    bit_show(x);
    //Meno o piu' significativi?
    printf("\n\nVuoi estrarre i bit:\n[1] Piu' significativi\n[2] Meno significativi\n");
    scanf("%d", &scelta);
    //Quanti bit?
    printf("Quanti bit estrarre?\n");
    scanf("%d", &k);
    /* estrai i bit in vari modi */
    Mask_estratta = Maschera(x, scelta, k);
    Shift_estratto = Shift(x, scelta, k);
    Pro_div_2_estratto = Prodotto_divisione(x, scelta, k);
    /* Stampa risultati*/
    printf("\nValore iniziale: %d = ", x); bit_show(x);
    printf("\n\nValore Estratto con la maschera : %d = ", Mask_estratta); bit_show(Mask_estratta);
    printf("\n\nValore Estratto con i due shift : %d = ", Shift_estratto); bit_show(Shift_estratto);
    printf("\n\nValore Estratto con i prodotti/rapporto : %d = ", Pro_div_2_estratto);
    bit_show(Pro_div_2_estratto);

    printf("\n");
    return 0;
}

unsigned char Maschera(unsigned char x, int scelta, int k)
{
    /* PREPARIAMO LA MASCHERA DI 32 bit/
    /* unsigned, perche' 2^32-1 mi darebbe un valore anegativo complementato a 2.
       Con unsigned ho il controllo totale dei 32 bit*/
    unsigned char Mask=0, i;
    /* Sfruttando le POTENZE di 2 o lo shift, andiamo a "coprire" i k bit da estrarre */
    for(i=1;i<=k;i++) Mask=Mask<<1|1; //Avrei potuto fare anche Mask =(pow(2,k)-1)
    //----- Estrarre bit -----
    if(scelta==1) /* Per gli MSB SHIFTIAMO la MASK sinistra di 32-k e giungiamo mask con x*/
        return x&(Mask<<(sizeof(unsigned char)*8-k));
    else if(scelta==2) /* Per gli LSB non c'e' bisogno di shiftare a destra, giungiamo la mask ottenuta */
        return x&Mask;
}

unsigned char Shift(unsigned char x, int scelta, int k)
{
    unsigned short Bit_tot= sizeof(unsigned char)*8;
    /* Estrarre i piu' significativi*/
    if(scelta==1)
    {
        x=x>>(Bit_tot-k); //shifta a destra MAX_BIT-k per cancellare i meno significativi
        x=x<<(Bit_tot-k); //shifta a sinistra MAX_BIT-k per l'astrazione
    }
    /* Estrarre i meno significativi*/
    else if(scelta==2)
    {
        x=x<<(Bit_tot-k); //shifta a sinistra MAX_BIT-k per cancellare i meno significativi
        x=x>>(Bit_tot-k); //shifta a destra MAX_BIT-k per l'astrazione
    }
    return x;
}

```

```

unsigned char Prodotto_divisione(unsigned char x, int scelta, int k)
{
    unsigned short Bit_tot= sizeof(unsigned char)*8;
    /* Il ragionamento e' totalmente uguale a Shift()
     Dato che >>1 corrisponde ad una divisione per 2^(1) e <<1 un prodotto per 2^(1),
     Sostituiamo lo shift con le corrispondenti operazioni
     Es. 12>>2 si ha 0011, cioe' 12/2^(2)*/

    /* Estrarre i piu' significativi*/
    if(scelta==1)
    {
        x=x/pow(2,Bit_tot-k); //shifta a destra = divisione per una potenza di 2
        x=x*pow(2,Bit_tot-k); //shifta a sinistra = prodotto per una potenza di 2
    }
    /* Estrarre i meno significativi*/
    else if(scelta==2)
    {
        x=x*pow(2,Bit_tot-k); //shifta a sinistra = prodotto per una potenza di 2
        x=x/pow(2,Bit_tot-k); //shifta a sinistra = prodotto per una potenza di 2
    }
    return x;
}

void bit_show(unsigned char x)
{
    short j;
    int bit[len];
    bit[0]=0;
    j=len-1;
    do
    {
        bit[j] = x&1; //prendi il primo bit e mettilo alla len-1 pos (stampa avverra' al contrario)
        --j;
        x=x>>1; //sposta di 1 a destra per considerare i prossimi bit
    }
    while(n!=0 && j>=0); //Finche' il numero non e' 0 e non ho completato tutti i len bit

    if(j>0)//nel caso j non sia 0 gli altri bit della variabile vengono posti a 0
    {
        do
        {
            bit[j]=0;
            --j;
        }
        while (j>=0);
    }

    for(j=0; j<len; j++) (j%4==0)?printf(" %d",bit[j]):printf("%d",bit[j]); //stampa i bit della
variabile
}

```

### OUTPUT:

Inserire un numero: 123  
 0111 1011

Vuoi estrarre i bit:  
 [1] Più' significativi  
 [2] Meno significativi  
 1  
 Quanti bit estrarre?  
 3

Valore iniziale: 123 = 0111 1011

Valore Estratto con la maschera : 96 = 0110 0000  
 Valore Estratto con i due shift : 96 = 0110 0000  
 Valore Estratto con i prodotti/rapporti : 96 = 0110 0000

Inserire un numero: 123  
 0111 1011

Vuoi estrarre i bit:  
 [1] Più' significativi  
 [2] Meno significativi  
 2  
 Quanti bit estrarre?  
 5

Valore iniziale: 123 = 0111 1011

Valore Estratto con la maschera : 27 = 0001 1011  
 Valore Estratto con i due shift : 27 = 0001 1011  
 Valore Estratto con i prodotti/rapporti : 27 = 0001 1011

## ESERCIZIO 5 [LIV.1]

```
/*
[liv.1] Scrivere due function C di conversione di un intero positivo (int) da base 10 a base 2
mediante l' algoritmo delle divisioni successive realizzato rispettivamente:
    - Usando gli operatori di quoziente e resto della divisione intera;
    - Usando gli operatori bitwise.

*/
#include <stdio.h>
#include <stdlib.h>
#define MAX_LEN 16
void Conv_Quoz(unsigned short X, unsigned short Bit[]);
void Conv_Wise(unsigned short X, unsigned short Bit[]);
void Stampa_Bit(short Bit[]);

int main()
{
    unsigned short X;
    unsigned short Bit[MAX_LEN]; //Per la visualizzazione binaria

    printf("Inserire l'intero da trasformare in binario:\n");
    scanf("%hd", &X);
    /* Conversione effettuate mediante quoziente*/
    printf("Conversione effettuate mediante il quoziente:\n");
    Conv_Quoz(X, Bit);
    Stampa_Bit(Bit);
    /* Conversione effettuate mediante i bitwise*/
    printf("\nConversione effettuate mediante i bitwise:\n");
    Conv_Wise(X, Bit);
    Stampa_Bit(Bit);
    return 0;
}

/*
Il valore di X diventerà il quoziente stesso.
Il vettore Bit[] conterrà bit con ordine MSB al più LSB (Bit[0] = LSB)
*/
void Conv_Quoz(unsigned short X, unsigned short Bit[])
{
    unsigned short R; //Definiamo resto e quoziente
    short i=0;
    /* Il ciclo si interrompe se il quoziente sarà 0 oppure avremo coperto tutti e MAX_LEN bit*/
    do
    {
        Bit[i++] = X%2; //Mediante il resto della divisione, costruiamo la stringa binaria
        X = X/2; //Passiamo al quoziente successivo
    } while(X!=0&&i<MAX_LEN);
    /* Il quoziente ha restituito 0, ma devo coprire il resto dei bit a 0 fino a MAX_LEN */
    while(i<MAX_LEN)
        Bit[i++]=0;
}

void Stampa_Bit(short Bit[])
{
    short i;
    for(i=MAX_LEN-1;i>=0;i--)
    {
        printf("%hd", Bit[i]);
        if(i%4==0)printf(" "); //Arrivato ad ogni 8 bit, prenditi uno spazio
    }
}

/* Effettua le divisioni successive per 2, utilizzando i bit wise*/
void Conv_Wise(unsigned short X, unsigned short Bit[])
{
    unsigned short R; //Definiamo resto e quoziente
    short i=0;
    /* Il ciclo si interrompe se il quoziente sarà 0 oppure avremo coperto tutti e MAX_LEN bit*/
    do
    {
        Bit[i++] = X%2; //Mediante il resto della divisione, costruiamo la stringa binaria
        X=X>>1; //Equivale a X = X/2;
    } while(X!=0||i<MAX_LEN);
    /* Il quoziente ha restituito 0, ma devo coprire il resto dei bit a 0 fino a MAX_LEN */
    while(i<MAX_LEN)
        Bit[i]=0;
}
```

}

**OUTPUT:**

```
Inserire l'intero da trasformare in binario:  
30001  
Conversione effettuate mediante il quoziante:  
0111 0101 0011 0001  
Conversione effettuate mediante i bitwise:  
0111 0101 0011 0001
```

```
Inserire l'intero da trasformare in binario:  
50000  
Conversione effettuate mediante il quoziante:  
1100 0011 0101 0000  
Conversione effettuate mediante i bitwise:  
1100 0011 0101 0000
```

## ESERCIZIO 6 [LIV.1]

```
/*
[liv.1] Scrivere una function C di conversione di un intero positivo da base 2 a base 10
che generi un array di caratteri contenenti le cifre decimali.
*/
#include <stdio.h>
#include <stdlib.h>
#define LEN 8
short int Crea_Array_Decimale(unsigned char x, unsigned char Bit[], unsigned char Cifre_decimali[]);

int main()
{
    unsigned char Bit[8], Cifre_decimali[LEN], x;
    short int i, n_cifre;
    //Input numero x
    printf("Inserire un numero[0...255]: \n");
    scanf("%d", &x);
    //n cifre mi serve per stampare dal piu' significativo al meno significativo
    n_cifre=Crea_Array_Decimale(x, Bit, Cifre_decimali);
    printf("Array di caratteri contenenti le cifre decimali\n");
    for(i=n_cifre-1; i>=0; i--) printf(" [%c]", Cifre_decimali[i]+48);
    printf("\n");
    return 0;
}
short int Crea_Array_Decimale(unsigned char x, unsigned char Bit[], unsigned char Cifre_decimali[])
{
    short int i, q, pot=1;
    unsigned char decimale=0;

    /* ---- Azzero array Bit e delle cifre decimali ---- */
    for(i=0; i<LEN; i++)
    {
        Bit[i]=0; Cifre_decimali[i]=0;
    }
    /* ----- CONVERTO DA BASE 10 A BASE 2 -----*/
    i=0; q=x;
    do
    {
        Bit[i++] = q%2; //Mediante il resto della divisione, costruiamo la stringa binaria
        q = q/2; //Passiamo al quoziente successivo
    }while(q!=0&&i<LEN);
    //L'ordine nell'array e' Bit[0]=lsb perche' e' piu' facile convertirlo in decimale

    /* ---- Base 2 - 10 ----*/
    for(i=0; i<LEN; i++)
        decimale += Bit[i]* (pot<<i); //bit[i]*2^(i)
    /* Se tutto e' andato bene, decimale sara' uguale a x */

    /* --- Dividiamo le cifre decimali ---*/
    i=0; q = decimale;
    do
    {
        Cifre_decimali[i++] = q%10;
        q = q/10;
    }while(q!=0&&i<LEN);
    return i;
}
```

### OUTPUT:

Inserire un numero[0...255]:  
168  
Array di caratteri contenenti le cifre decimali  
[1] [6] [8]

Inserire un numero[0...255]:  
210  
Array di caratteri contenenti le cifre decimali  
[2] [1] [0]

## ESERCIZIO 7 [LIV.2]

```
/**[liv.2]Ripetere l'esercizio precedente nel caso che l'input sia una stringa di caratteri
contenenti i bit del numero.**/
#include <stdio.h>
#include <stdlib.h>
#define LEN 8
short int Crea_Array_Decimale(unsigned char Bit[], unsigned char Cifre_decimali[]);

int main()
{
    unsigned char Bit[LEN], Cifre_decimali[LEN], x;
    short int i=0, n_cifre;
    //Inserisci i bit dentro la stringa
    do
    {
        printf("Inserire il bit [%d]: \n", i);
        scanf("%c", &x);
        if(x=='1'||x=='0') Bit[i++]=x; else printf("\nVALORE NON CONSENTITO, RIPROVA");
        fflush(stdin);
    }while(i<LEN);
    printf("Stringa binaria inserita:\n ");
    for(i=LEN-1; i>=0; i--) printf("%c", Bit[i]);
    //n cifre mi serve per stampare dalla cifra più significativa alla meno significativa
    n_cifre = Crea_Array_Decimale(Bit, Cifre_decimali);
    /* ----- STAMPA CIFRE ----- */
    printf("\nArray di caratteri contenenti le cifre decimali\n");
    for(i=n_cifre-1; i>=0; i--)
        printf(" [%c]", Cifre_decimali[i]+48);
    /* ----- */
    printf("\n");
    return 0;
}
short int Crea_Array_Decimale(unsigned char Bit[], unsigned char Cifre_decimali[])
{
    short int i, q, pot=1;
    unsigned char decimale=0;

    /* ---- Azzero array cifre decimali ---- */
    for(i=0; i<LEN; i++) Cifre_decimali[i]=0;

    /* ---- Base 2 - 10 ----*/
    for(i=0; i<LEN; i++)
        decimale += (Bit[i]-48)*(pot<<i); // (bit[i]-48)*2^(i)...-48 perche' contiene il codice ascii

    /* --- Dividiamo le cifre decimali ---*/
    i=0; q = decimale;
    do
    {
        Cifre_decimali[i++] = q%10;
        q = q/10;
    }while(q!=0&&i<LEN);
    return i;
}
```

### OUTPUT:

```
Inserire il bit [0]:
213
VALORE NON CONSENTITO, RIPROVA
Inserire il bit [0]:
2
VALORE NON CONSENTITO, RIPROVA
Inserire il bit [0]:
0
Inserire il bit [1]:
0
Inserire il bit [2]:
1
Inserire il bit [3]:
1
Inserire il bit [4]:
1
Inserire il bit [5]:
1
Inserire il bit [6]:
0
Inserire il bit [7]:
0
Stringa binaria inserita:
00110110
Array di caratteri contenenti le cifre decimali
[6] [0]
```

```
Inserire il bit [0]:
0
Inserire il bit [1]:
1
Inserire il bit [2]:
1
Inserire il bit [3]:
1
Inserire il bit [4]:
0
Inserire il bit [5]:
1
Inserire il bit [6]:
1
Inserire il bit [7]:
0
Stringa binaria inserita:
01101110
Array di caratteri contenenti le cifre decimali
[1] [1] [0]
```

## ESERCIZIO 8 [LIV.1]

```
/**[l1v1] Scrivere una function C per eseguire l'addizione aritmetica(Versione1) in base 2 mediante gli operatori bitwise.*/
#include <stdio.h>
#include <stdlib.h>
short int Addizione_aritmetica(short int op1, short int op2);
int main()
{
    short int op1, op2;
    /* Input degli operatori*/
    printf("Inserisci il primo addendo:\n");
    scanf("%hd", &op1);
    printf("Inserisci il secondo addendo!\n");
    scanf("%hd", &op2);
    /* Se il valore che esce e' negativo, significa che si sara' verificato un'overflow di intero */
    printf("Il risultato della somma aritmetica mediante bitwise e':\n%hd",Addizione_aritmetica(op1,op2));
    return 0;
}

short int Addizione_aritmetica(short int op1, short int op2)
{
    short int Sum=0, Rip=1;
    /* Cicla fin quando non si ha un riporto uguale a 0 */
    do
    {
        Sum = op1^op2; /* La seguente operazione effettua la somma tra bit, senza riporto*/
        Rip = op1&op2; /* calcola riporto */
        Rip = Rip<<1; /* Porta il riporto a sinistra */
        op1 = Sum; /* operando1 conterrà la somma */
        op2 = Rip; /* operando2 il riporto */
    }while(Rip>0);

    return Sum;
}
```

### OUTPUT:

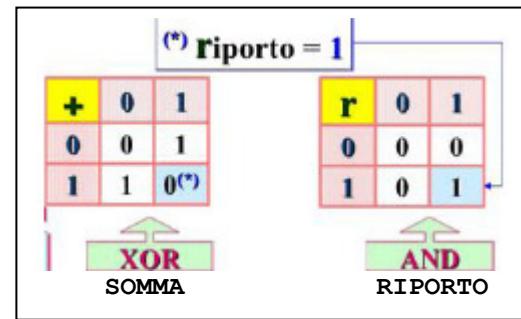
Inserisci il primo addendo:

32700

Inserisci il secondo addendo:

50

Il risultato della somma aritmetica mediante bitwise e':  
32750



### Esempio di OVERFLOW di INTERO.

Inserisci il primo addendo:

32768

Inserisci il secondo addendo:

0

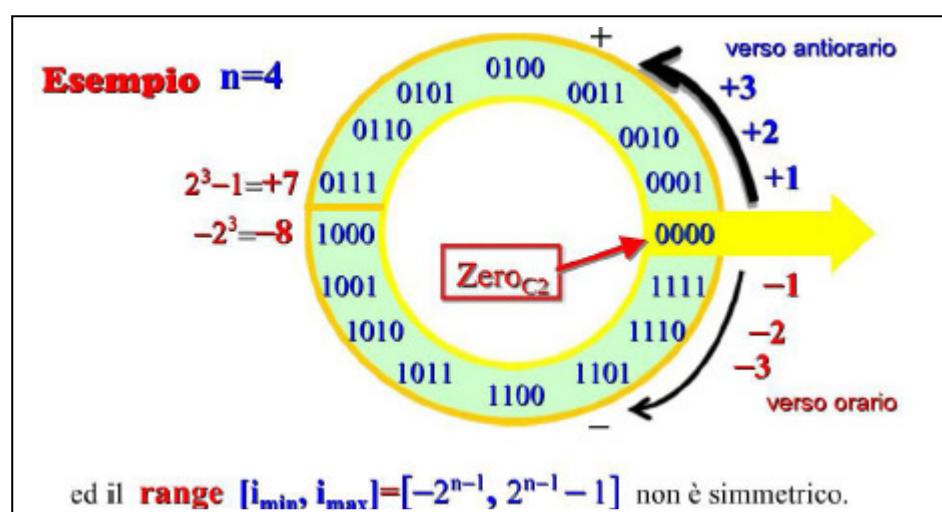
Il risultato della somma aritmetica mediante bitwise e':  
-32768

Con uno short possiamo rappresentare un intervallo del tipo [-32768, +32767].

+32768 oltrepassa questo range, per tanto, ricordando la circonferenza della rappresentazione del complemento a 2, dopo 32767, ossia l'estremo positivo più grande, si avrà -32768, l'estremo negativo più grande. Stesso discorso vale se volessimo rappresentare -32769: se -32768 è il valore minimo rappresentabile, dopo di esso non avremo -32769, ma ripartirebbe da 32767, il valore positivo più grande.

Quindi, un **OVERFLOW di INTERO** si

**verifica quando vogliamo rappresentare un valore oltre agli estremi permessi.**



## ESERCIZIO 9 [LIV.1]

```
/*
[liv1] Scrivere una function C per eseguire la sottrazione aritmetica
in base 2 mediante gli operatori bitwise.
*/
#include <stdio.h>
#include <stdlib.h>
int Sottrazione_aritmetica(int op1, int op2);
int main()
{
    int op1, op2;
    /* Input degli operatori*/
    printf("Inserisci il minuendo:\n");
    scanf("%d", &op1);
    /* Essendo una SOTTRAZIONE ARITMETICA, operando2 deve essere piu' piccolo di operando 1 */
    do
    {
        printf("Inserisci il sottraendo:\n");
        scanf("%d", &op2);
        if(op1<op2)printf("SOTTRAZIONE ARITMETICA!\nMettere un operando minore o uguale al primo
operando\n");
    }while(op1<op2);
    /* ----- */
    printf("Il risultato della sottrazione aritmetica mediante bitwise e':\n%d",
Sottrazione_aritmetica(op1,op2));

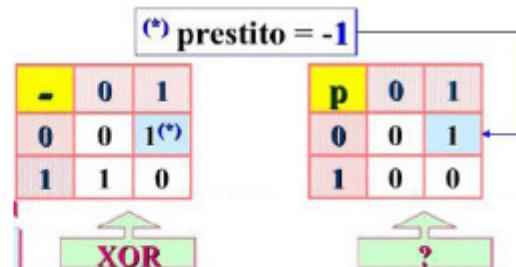
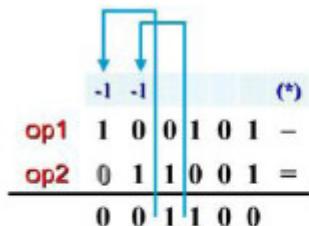
    return 0;
}

int Sottrazione_aritmetica(op1, op2)
{
    int Sub=0, Pre=1;
    /* Cicla fin quando non si ha un riporto nullo*/
    while(Pre>0)
    {
        Sub = op1^op2; /* La seguente operazione effettua la differenza tra bit, senza riporto*/
        Pre = (~op1)&op2; /* calcola il prestito. C'e' un'altra soluzione trovabile con De Morgan*/
        Pre = Pre<<1; /* Porta il prestito a sinistra */
        op1 = Sub; /* operando1 conterrà la somma */
        op2 = Pre; /* operando2 il riporto */
    }
    return Sub;
}
```

### OUTPUT:

```
Inserisci il minuendo:
10
Inserisci il sottraendo:
7
Il risultato della sottrazione aritmetica mediante bitwise e':
3

Inserisci il minuendo:
10
Inserisci il sottraendo:
15
SOTTRAZIONE ARITMETICA!
Mettere un operando minore o uguale al primo operando
Inserisci il sottraendo:
3
Il risultato della sottrazione aritmetica mediante bitwise e':
7
```



**NOT (OR(op1, NOT(op2)))**  
**AND (NOT(op1), op2)**

## ESERCIZIO 10 [LIV.1]

```
/**[liv.1]Scrivere una function C che, fissato il numero n di bit, calcoli
la rappresentazione di un intero:
    -per complemento a 2;
    -per eccesso B (B-biased) **/


#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define MAX_LEN 32
int Complemento_a_2(int n_bit, int x);
int Eccesso_B(int n_bit, int x);
void div_suc(int, int);

int main()
{
    int x, n_bit, Comp_2, Biased;
    /* input */
    printf("Inserire il numero di bit su cui rappresentare:\n");
    scanf("%d", &n_bit);
    printf("Inserire il valore da rappresentare:\n");
    scanf("%d", &x); printf("\n");
    /* Lavoro function */
    Comp_2 = Complemento_a_2(n_bit,x);
    Biased = Eccesso_B(n_bit,x);

    /* output: se i valori superavano i range, si segnala un errore */
    if(Comp_2>=0)
    {
        printf("Rappresentazione in complemento a due: %d\nComplemento in binario: ",Comp_2);
        div_suc(Comp_2, n_bit);
    }
    else printf("Rappresentazione in complemento a due non consentita (oltre al range)\n");
    printf("\n");
    if(Biased>=0)
    {
        printf("Rappresentazione in BIASED(eccesso-B): %d\nBiased in binario: ",Biased);
        div_suc(Biased, n_bit);
    }
    else printf("Rappresentazione in BIASED (eccesso-b) non consentita (oltre al range)");
    printf("\n");
    return 0;
}

/* ----- Complemento_a_2 -----
Esprime la corrispondente rappresentazione in complemento a 2 di quel determinato valore X.
Se x rientra nel range, avremo (Potenza_n_bit+x)%Potenza_n_bit. Nel caso che X sia positivo,
tale operazione restituirà x stesso, dato che il modulo di (Potenza+x)/potenza, restituisce x
come resto. Nel caso che X sia negativo, viene effettuato (Potenza-x)/potenza,
il resto della divisione è lo stesso Potenza-x, cioè il complemento a 2 di un numero negativo.
In conclusione, esprimiamo tutto con C2(k)=[2^n+k]mod 2^n.

ALTERNATIVA: verifica se X è => di 0. Se sì, il complemento sarà X stesso.
Se sarà negativo, fai 2^n-|x|, per conoscere il corrispondente valore negativo
in rappresentazione in complemento a 2 */

int Complemento_a_2(int n_bit, int x)
{
    /* Definiamo i valori max e min per quei n_bit*/
    int Range_max = pow(2, n_bit-1)-1;
    int Range_min = -pow(2, n_bit-1);
    int Potenza_n_bit = pow(2, n_bit); /*Potremo fare ciò con i bitwise e senza sprecare variabili,
                                         ma facciamo così per comodità e leggibilità del codice*/
    /* Se il valore va oltre all'intervallo rappresentabile, segnala errore */
    if(x>=Range_min&&x<=Range_max)
    {
        /*Se x è positivo, restituisce il resto, cioè x stesso. Se negativo, il complemento a 2*/
        return (Potenza_n_bit+x)%Potenza_n_bit;
    }
    else return -1;//oltre ai range, restituisce un errore
}

/* ----- Eccesso_B -----
Esprime la corrispondente rappresentazione biased di quel determinato valore X.
Se x rientra nel range, effettua la somma algebrica tra il BIAS e X.
Se X è negativo, attraverso la somma algebrica, effettuerà una differenza.
Ricordiamo che il BIAS che il valore da cui comincia la rappresentazione*/
int Eccesso_B(int n_bit, int x)
{
    /* Definiamo il bias e gli intervalli estremi */
    int Bias = pow(2, n_bit-1)-1; //Bias indica lo 0, ovvero il valore da cui parte la rappresentazione
    int Range_max = pow(2, n_bit-1);
```

```

int Range_min = -pow(2, n_bit-1)-1;
/* Se il valore va oltre all'intervallo rappresentabile, segnala errore */
if(x>=Range_min&&x<=Range_max)
{
    /* Bias+x=effettivo Valore in biased.Se x<0 effettua una sottrazione, dato che e' una somma
algebrica*/
    return Bias+x; //restituisce la rappresentazione in bias
}
else return -1;
}

/*Questa Function ci servira' a convertire in bit tramite operatori di quoziente e resto della
divisione
intera e stamparli sullo schermo. Ha 2 parametri di ingresso: il numero da volere visualizzare in bit
int numero) e su quanti bit volerlo rappresentare (int n_bit) e nessuno parametro d'uscita.*/
void div_suc(int numero, int n_bit)
{
    int i,quoziente,resto, bit[32];
    for (i=0; i<32; i++)
        bit[i]=0; //Inizializzo a 0 l'array
    i=0;

    quoziente=numero; resto=numero; //Per evitare errori del while inizializzo
                                    //il quoziente e il resto uguali al numero
    while (quoziente>=0 && i<n_bit)
    {
        quoziente=numero/2; //divido il numero per 2
        resto=numero%2; //faccio il modulo tra il numero e 2 quindi mi dara'
                           //il resto della divisione intera
        numero=quoziente; //il numero ora diventerà il quoziente
        bit[(n_bit-1)-i]=resto; //nell'array inseriro il valori dei vari bit
        i++;
    }
    for (i=0; i<n_bit; i++)
        printf("%d",bit[i]); //Stampo sullo schermo i bit meno ai piÃ¹ significativi
    printf("\n\n");
}

```

### OUTPUT:

Inserire il numero di bit su cui rappresentare:

8

Inserire il valore da rappresentare:

-5

Rappresentazione in complemento a due: 251  
Complemento in binario: 11111011

Rappresentazione in BIASED(eccesso-B): 122  
Biased in binario: 01111010

Inserire il numero di bit su cui rappresentare:

16

Inserire il valore da rappresentare:

32768

Rappresentazione in complemento a due non consentita (oltre al range)

Rappresentazione in BIASED(eccesso-B): 65535  
Biased in binario: 1111111111111111

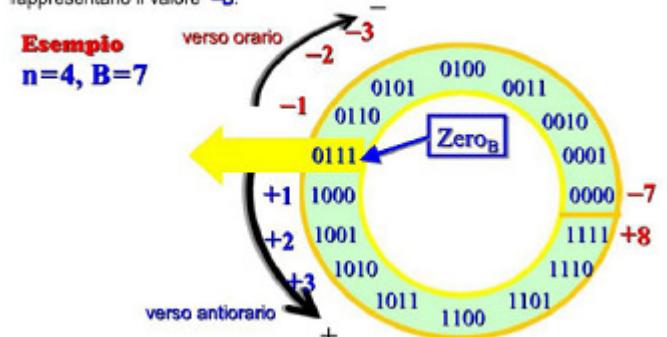
$$C_2(k) = [2^n + k] \% 2^n$$

Se  $k \geq 0 \Rightarrow C_2(k) = k$       Se  $k < 0 \Rightarrow C_2(k) = 2^n - k$

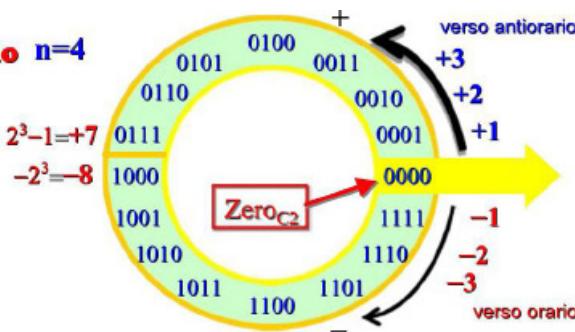
$$\text{Biased}(x) = \text{Bias} + x$$

Nella rappresentazione biased con bias  $B=2^{n-1}-1$  (rappresentazione eccesso B) il bias B rappresenta il valore zero mentre gli n zeri rappresentano il valore  $-B$ .

Esempio  
n=4, B=7



Esempio n=4



ed il range  $[i_{\min}, i_{\max}] = [-2^{n-1}, 2^{n-1}-1]$  non è simmetrico.

ed il range  $[i_{\min}, i_{\max}] = [-(2^{n-1}-1), 2^{n-1}]$  non è simmetrico.

## ESERCIZIO 11 [LIV.1]

```
/***
[liv1] Conoscendo la rappresentazione degli interi in C, riscrivere
la function C per l'addizione binaria di due interi (Z) mediante
gli operatori bitwise (Esercizio 8).
Se l'operazione da implementare deve essere l'addizione algebrica
(cioè deve valere anche per gli interi negativi rappresentati per complemento
a 2), quale accorgimento va usato nella traduzione in C dell'algoritmo...
e perchè...?

RISPOSTA:
Basta cambiare la condizione del while! Controlla la descrizione del perche'...
*/
#include <stdio.h>
#include <stdlib.h>
short int Addizione_aritmetica(short int op1, short int op2);
int main()
{
    short int op1, op2;
    /* Input degli operatori*/
    printf("Inserisci il primo addendo:\n");
    scanf("%hd", &op1);
    printf("Inserisci il secondo addendo!\n");
    scanf("%hd", &op2);
    /* Se il valore che esce e' negativo, significa che si sara' verificato
       un'overflow di intero */
    printf("Il risultato della somma aritmetica mediante bitwise e':\n%hd",
    Addizione_aritmetica(op1,op2));
    return 0;
}

short int Addizione_aritmetica(short int op1, short int op2)
{
    short int Sum=0, Rip=1;

    /* RISPOSTA:
       Diverso e non maggiore a 0, perche' se facessi 2+(-1) e seguendo l'algoritmo
       per l'addizione aritmetica, essendo -1 rappresentato in complemento a due,
       su 4 bit avremo 0010+1111. Si arrivera' ad un passaggio in cui il riporto
       sara' 1000, ma il C lo vedrebbe come un numero negativo e uscirebbe dalla
       condizione del while, senza finire l'algoritmo.
       Fatto cio', utilizzando valori positivi e negativi, cioe' rappresentati
       col C2, si ridurra' tutto ad una somma algebrica.
    */
    /* Cicla fin quando non si ha un riporto uguale a 0 */
    do
    {
        Sum = op1^op2; /* La seguente operazione effettua la somma tra bit, senza riporto */
        Rip = op1&op2; /* calcola riporto */
        Rip = Rip<<1; /* Porta il riporto a sinistra */
        op1 = Sum; /* operando1 conterrà la somma */
        op2 = Rip; /* operando2 il riporto */
    }while(Rip!=0);

    return Sum;
}
```

### OUTPUT:

```
Inserisci il primo addendo:
-102
Inserisci il secondo addendo!
32
Il risultato della somma aritmetica mediante bitwise e':
-70

Inserisci il primo addendo:
10
Inserisci il secondo addendo!
-100
Il risultato della somma aritmetica mediante bitwise e':
-90
```

## ESERCIZIO 12 [LIV.1]

```
/*
[liv.1] Scrivere una function C per visualizzare la rappresentazione binaria (s,e,m) di un numero
float.
Verificare che il valore del numero ottenuto coincida con il dato iniziale*/
#include <stdio.h>
#include <stdlib.h>
#define BIAS 127

float Estrai_sem(int x, char bit_float[]);
void Visualizza_bit_float(char bit_float[]);
int main()
{
    /* Utilizziamo la UNION perche' abbiamo la necessita' di prendere
       da input un valore float, ma all'interno della funzione "ESTRAI_SEM",
       dobbiamo lavorare in valore interi.
       f ed i, sono 2 tipi diversi che condividono la stessa area di memoria
    */
    union fl_int
    {
        float f;
        int i;
    }x;
    /* AVREI POTUTO SCEGLIERE DI LAVORARE ANCHE CON LA STRUCT DI BIT */

    char bit_float[32]; // Bit float avra' il SEM in binario
    float val;
    printf("Inserire un numero float\n");
    scanf("%f", &x.f);
    val = Estrai_sem(x.i, bit_float);
    printf("\nSEGNO ESPONENTE MANTISSA\n");
    Visualizza_bit_float(bit_float);

    printf("\n\nVERIFICA\nApplicando (-1)^s*(1.m)*b^(e-bias), avremo: %.3f \n", val);
    return 0;
}

float Estrai_sem(int x, char bit_float[])
{
    int s, exp, mantissa, tmp, i; /* Segno, Esponente e mantissa*/
    int Mask=0, Estratto;
    float Float_Verifica=0, Mantissa_Hidden;

    /* Estrai segno */
    Mask = 0x80000000;
    Estratto = Mask&x;
    s = (Estratto >> 31);
    /* Estrai Esponente */
    Mask=0x7f800000;
    Estratto = Mask&x;
    exp = (Estratto >> 23);
    /* Estrai Mantissa */
    Mask=0x007fffff;
    mantissa = Mask&x;

    /* CREAIAMO IL BIT FLOAT */
    for(i=31, tmp=x; i>=0; i--)
    {
        bit_float[i]= (char)tmp&1; //estrai i singoli bit e converti a char
        tmp = tmp>>1; //passa al prossimo bit
    }

    /* Ricostruzione del Float iniziale */
    // NB: Per avere il floating point, dobbiamo effettuare
    // x = (-1)^s*[1.m]*b^(e-bias)
    // La nostra base e' 2 e il bias 127. Dato che abbiamo un hidden bit,
    // lo ripristiniamo aggiungendo 1 alla mantissa
    Mantissa_Hidden=((float)mantissa)/(1<<23))+1; /* Divido la mantissa per 2^23, in tal modo da
                                                       spostarmi la virgola e aggiungere l'hidden
                                                       bit */
    Float_Verifica = pow(-1, s)*(Mantissa_Hidden)*pow(2, (exp-BIAS));
    /* */

    printf("\nSegno: %d \n", s);
    printf("Mantissa: %.5f \n", Mantissa_Hidden-1); //Mantissa senza hidden bit
    printf("Esponente: %d \n", exp-BIAS); //Exp-Bias
}
```

```

    return Float_Verifica;
}

void Visualizza_bit_float(char bit_float[])
{
    int i;
    printf("%5d ", bit_float[0]);
    for(i=1;i<9;i++) printf("%d", bit_float[i]);
    printf("   ");
    for(i=9;i<32;i++) printf("%d", bit_float[i]);
}

```

**OUTPUT:**

Inserire un numero float  
-12.75

Segno: -1  
Mantissa: 0.59375  
Esponente: 3

SEGNO ESPONENTE MANTISSA  
1 10000010 10011000000000000000000000000000

**VERIFICA**  
Applicando  $(-1)^s \times (1.m) \times 2^{e-bias}$ , avremo: -12.750

Inserire un numero float  
0.75

Segno: 0  
Mantissa: 0.50000  
Esponente: -1

SEGNO ESPONENTE MANTISSA  
0 01111110 10000000000000000000000000000000

**VERIFICA**  
Applicando  $(-1)^s \times (1.m) \times 2^{e-bias}$ , avremo: 0.750

**PROPRIETA' DA RICORDARE SULLO STANDARD**

Nel **S.A. IEEE Std. F(2, t, E<sub>min</sub>, E<sub>max</sub>)** un numero del formato **Basic** è rappresentato in memoria come



dove segno esponente mantissa

- **s** denota il **segno della mantissa**;
- l'**esponente e** è rappresentato come "*intero biased*";
- la **mantissa m** è rappresentata, con **s per segno e modulo**, su **t** bit a **bit implicito l<sup>†</sup>** (**precisione = t+1**) ed è generata con lo schema del **round to nearest**;

ed il suo **valore** è pertanto  $x = (-1)^s [l.m] \times 2^{e-Bias}$

| Oggetto  | Caratterizzazione       |            |   |
|--|-------------------------|------------|---|
|  | esponente e             | mantissa m | l |
| Numeri normalizzati<br>valore = $(-1)^s [l.m] \times 2^{e-Bias}$     | $E_{min} < e < E_{max}$ | $m \geq 0$ | 1 |
| Infinito con segno   | $e = E_{max}$           | $m = 0$    | - |
| NaN (Not A Number)   | $e = E_{max}$           | $m \neq 0$ | - |
| Zero con segno   | $e = E_{min}$           | $m = 0$    | - |
| Numeri denormalizzati<br>valore = $(-1)^s [l.m] \times 2^{e-Bias+1}$ | $e = E_{min}$           | $m \neq 0$ | 0 |

| Tipo                 | numero bit esponente       | Bias  | t = numero bit mantissa |
|----------------------|----------------------------|-------|-------------------------|
| <b>single</b>        | 8<br>$e \in \{0..255\}$    | 127   | 23                      |
| <b>double</b>        | 11<br>$e \in \{0..2047\}$  | 1023  | 52                      |
| <b>double EXTEND</b> | 15<br>$e \in \{0..32767\}$ | 16383 | 64                      |

### ESERCIZIO 13 [LIV.3]

```
/*
 * [LIV3] Scrivere una function C di conversione di un numero reale da base 10 (l'input è una stringa
 * di char) alla rappresentazione floating-point IEEE Std 754. */
#include <stdio.h>
#include <string.h>
double converti_string_a_float(char s []);

void main()
{
    char s[16];

    /* Inserimento del numero reale come stringa di char */
    printf("Inserire numero reale:\t");
    scanf("%s", &s);

    /* stampa del numero nella rappresentazione floating point IEEE std 754 mediante la
     * alla function */
    printf("\nFloat costruito:\t%.3f", converti_string_a_float(s)); printf("\n\n");
}

/* ----- CONVERTI STRINGA A FLOAT -----
 * Come si svolge l'algoritmo:
 * - Troviamo la posizione del punto frazionare;
 * - Ricaviamo il segno
 * - Calcoliamo la parte intera
 * - Calcoliamo la parte decimale (se e' un numero reale)
 * -----*/
double converti_string_a_float(char s[])
{
    int separatore=strlen(s); //se non c'e' punto, il separatore coincide con il '\0'.
    int i=0, sign, punto=0; //sign sara' il segno e punto indica la presenza del punto.
                           //Fa rendere conto all'algoritmo di non calcolare la parte
                           //decimale
    double val=0.0; //numero double che vogliamo costruire
    double power=10.0; //power e' la potenza di 10

    /** _____ POSIZIONE DEL PUNTO FRAZIONARIO _____ */
    /* si controlla se c'e' il punto frazionario, cioe se il numero inserito e' un float.
     * Se c'e', viene ricordato in punto, per effettuare anche la parte decimale*/
    for (i=0; i<10; i++)
    {
        if (s[i]=='.') ||
        {
            separatore=i;
            punto=1; //C'e' il punto, mi serve per saltare la parte della virgola
            break;
        }
    }
    /** _____ SEGNO _____ */
    /* Se abbiamo '-', conservati la "negativita'", altrimenti dovrà essere per forza
     * positivo*/
    if (s[0]=='-') sign=-1;
    else sign=1;
    /* Se il primo carattere e' un segno di '+' o '-', avanziamo l'indirizzo base dell'array
     * "trascurando il segno". Cio' comporta che avremo un'unita' in meno, quindi la posizione esatta del
     * separatore si trova a separatore-1*/
    if (s[0]=='+') || s[0]=='-')
    {
        s++; //(&flts[0])+1, indirizzo base + 1
        separatore--; //Il separatore sara' una posizione in meno
    }

    /** _____ PARTE PRIMA DELLA VIRGOLA(INTERA) _____ */
    /* Se non e' stato inserito il segno, il ciclo sara' da 0 a separatore -1.
     * Se inseriamo un segno avanziamo l'indirizzo base di 1, ma il separatore
     * sara' una quantita' in meno, perch' il nostro array diminuisce di 1 la sua
     * lunghezza (basta guardare l'operazione sopra) */
    for (i=0; i<separatore; i++)
        val=10.0*val+(s[i]-48); //Evitiamo l'uso di POW
    /* nel codice ASCII il carattere '0' vale 48, quindi togliendo 48 al carattere i-simo
     * si ottiene il numero decimale che si somma al precedente valore di 'val' moltiplicato
     * opportunamente per 10 per tener conto dei pesi delle cifre decimali
     * Si prova a fare una prova e ci si rende conto che va bene questa soluzione */
}
```

```
/** PARTE DOPO LA VIRGOLA(DECIMALE) [se presente] */
/* Se e' un numero intero, salta questa parte, dato che non dobbiamo considerare alcuna cifra decimale */
if(punto)
{
    i = separatore+1; //parto dal primo elemento dopo il punto frazionario
    while (s[i]!='\0')
    {
        /* al valore intero si somma il valore decimale diviso il suo "peso" (posizione)
           Esempio: 12.75 Val=12.00 + 7/10=12.7
                      12.7 + 5/100=12.75 */
        val = val + ((s[i]-'0')/power);
        power=power*10.0; //viene aggiornato power, cioe' il "peso" del valore decimale
        i++;
    }
    return sign*val; /* ritorna il valore finale con segno */
}
```

**OUTPUT:**

**Inserire numero reale:** -12,75  
**Float costruito:** -12.750

**Inserire numero reale:** +21.987  
**Float costruito:** 21.987

**Inserire numero reale:** 5  
**Float costruito:** 5.000

## ESERCIZIO 14 [LIV.1]

```
/* [LIV1] Scrivere delle function C per calcolare rispettivamente l'EPSILON MACCHINA
del tipo float e del tipo double, visualizzando ad ogni passo i singoli bit. */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void Estrai_bit(long long int value, unsigned int bit[], short n);
void Mostra_bit_floating(unsigned int bit[], short n_bit, short exp);
```

$$\mathbf{\epsilon_{mach}} = \min \{ 0 < \epsilon \in F(\beta, t, E_{\min}, E_{\max}) : 1 \oplus \epsilon > 1 \}$$

```
int main()
{
    /* Le union contententi epsilon+1 (mi servono gli int per rapp in binario) */
    union EPS_F //Union per rappresentare epsilon macchina a singola precisione
    {
        float epspl_f;
        int epspl_f_i;
    } Epsilon_f;

    union EPS_D //Union per rappresentare epsilon macchina a doppia precisione
    {
        double epspl_d;
        long long int epspl_d_i;
    } Epsilon_d;

    /* I vari EPSILON */
    double eps_d; float eps_f;
    int scelta; short n =0, i;
    unsigned int bit[64];
    for(i=0;i<64;i++)bit[i]=0; //pulisco vettore

    printf("Vuoi visualizzare singola[1] o doppia[2] precisione? : ");
    fflush(stdin); scanf("%d", &scelta);
    /* Come trovare l'EPSILON macchina? effettuando delle divisioni successive per 2,
       fin quando non trovo il piu' piccolo numero che sommato ad 1, dia un valore maggiore
       di 1. */
    switch(scelta)
    {
        /* CALCOLO DELL'EPSILON MACCHINA A SINGOLA PRECISIONE */
        case 1:
            eps_f = 1;
            Epsilon_f.epspl_f = eps_f +1;
            printf("Risultati singola Precisione\n");
            while (Epsilon_f.epspl_f >1)
            {
                Estrai_bit(Epsilon_f.epspl_f_i, bit, 32); //estraggo i bit
                printf("n= %2d eps =", n); //Stampo l'epsilon macchina
                Mostra_bit_floating(bit, 32, 8);printf("\n");

                eps_f =eps_f/ 2;
                Epsilon_f.epspl_f = eps_f+1;
                n++;
            }
            eps_f*=2.0f; //ho trovato l'epsilon, devo ripristinare l'ultima divisione
            printf("L'epsilon macchina per i FLOAT e' %e", eps_f);
            break;
        /* CALCOLO DELL'EPSILON MACCHINA A DOPPIA PRECISIONE */
        case 2:
            eps_d = 1;
            Epsilon_d.epspl_d = eps_d +1;
            printf("Risultati doppia Precisione\n");
            while (Epsilon_d.epspl_d >1)
            {
                Estrai_bit(Epsilon_d.epspl_d_i, bit, 64); //estraggo i bit
                printf("n= %2d eps =", n); //Stampo l'epsilon macchina
                Mostra_bit_floating(bit, 64, 11);printf("\n");

                eps_d =eps_d/ 2;
                Epsilon_d.epspl_d = eps_d+1;
                n++;
            }
            eps_d*=2.0f; //ho trovato l'epsilon, devo ripristinare l'ultima divisione
    }
}
```

```

        printf("L'epsilon macchina per i DOUBLE e' %e", eps_d);

    break;
}

/*
 * Mostra i bit in base al floating point scelto */
void Mostra_bit_floating(unsigned int bit[], short n_bit, short exp)
{
    short i;
    for (i = 0; i < n_bit; i++)
    {
        if(i == 0 || i == exp)printf("%ld ", bit[i]);
        else printf("%ld", bit[i]);
    }
}
//La function trasforma un intero in un binario
void Estrai_bit(long long int value, unsigned int bit[], short n)
{
    short j = n-1;
    do
    {
        bit[j--]=value&1;
        value =value>> 1;
    }
    while (value!= 0 && j>= 0);
    if(j > 0)
    {
        do
        {
            bit[j--]= 0;
        }
        while (j >=0);
    }
}

```

**rappresentazione (binaria) fl.p.  
a bit隐式 su t bit per la  
mantissa**

$$\epsilon_{\text{mach}} = 2^{-t}$$

**ESEMPIO:** Floating point singola precisione => 23 bit per la mantissa, quindi l'epsilon macchina sara'  $2^{-(23)}$

**OUTPUT:**

Vuoi visualizzare singola[1] o doppia[2] precisione? : 1

Risultati singola Precisione

|       |        |                                   |
|-------|--------|-----------------------------------|
| n= 0  | eps =0 | 10000000 000000000000000000000000 |
| n= 1  | eps =0 | 01111111 100000000000000000000000 |
| n= 2  | eps =0 | 01111111 010000000000000000000000 |
| n= 3  | eps =0 | 01111111 001000000000000000000000 |
| n= 4  | eps =0 | 01111111 000100000000000000000000 |
| n= 5  | eps =0 | 01111111 000010000000000000000000 |
| n= 6  | eps =0 | 01111111 000001000000000000000000 |
| n= 7  | eps =0 | 01111111 000000100000000000000000 |
| n= 8  | eps =0 | 01111111 000000010000000000000000 |
| n= 9  | eps =0 | 01111111 000000001000000000000000 |
| n= 10 | eps =0 | 01111111 000000000100000000000000 |
| n= 11 | eps =0 | 01111111 000000000010000000000000 |
| n= 12 | eps =0 | 01111111 000000000001000000000000 |
| n= 13 | eps =0 | 01111111 000000000000100000000000 |
| n= 14 | eps =0 | 01111111 000000000000010000000000 |
| n= 15 | eps =0 | 01111111 000000000000001000000000 |
| n= 16 | eps =0 | 01111111 000000000000000100000000 |
| n= 17 | eps =0 | 01111111 000000000000000010000000 |
| n= 18 | eps =0 | 01111111 000000000000000001000000 |
| n= 19 | eps =0 | 01111111 000000000000000000100000 |
| n= 20 | eps =0 | 01111111 000000000000000000010000 |
| n= 21 | eps =0 | 01111111 000000000000000000001000 |
| n= 22 | eps =0 | 01111111 000000000000000000000100 |
| n= 23 | eps =0 | 01111111 000000000000000000000010 |

L'epsilon macchina per i FLOAT e' 1.192093e-007



## ESERCIZIO 15 [LIV.3]

```
/*
 [liv3] Scrivere una function C per calcolare dalla definizione l'ULP(x) dove
 x è il parametro reale float di input
*/
#include <stdio.h>
#include <stdlib.h>
#include <float.h>
#define bias 127
int estrae_esponente(int n);
float Calcola_ULP(float x);
```

$$\text{ulp}(a) = \min\{0 < \epsilon \in F(\beta, t, E_{\min}, E_{\max}) : a \oplus \epsilon > a\}$$

**u.l.p.** = Unit in the Last Place

$$\text{ulp}(1) = \text{Epsilon macchina } (\epsilon_{\text{mach}})$$

```
void main()
{
    /* _____ Per estrarre l'esponente, devo lavorare in intero _____ */
    union single
    {
        float f;
        int i;
    } x;

    float ulp;
    int exp;

    printf("Inserisci un numero reale: ");
    scanf("%f", &x.f);
    /*
        _____ CALCOLO ULP
        Dalla teoria sappiamo che per ricavare l'ulp(x), basta fare
        ulp(x) = EpsMach * 2^(exp).
        EpsMach = 2^-t = 2^-23
        Exp deve essere estratto dal floating point, facendo una maschera.
    */

    ulp = epsilon * 2^exp;
    /*
    exp = estrae_esponente(x.i); //estraggo esponente
    ulp = FLT_EPSILON * (1 << exp); //2^(exp)

    printf("\nx = %f\n");
    printf("\nULP1(%f) = %g\nULP2(%f) = %g\n", x.f, ulp, x.f, Calcola_ULP(x.f));
    */

    /**
     * ULP(X) = IL MINIMO VALORE POSITIVO REALE CHE SOMMATO A X, SI HA UN VALORE MAGGIOR DI X
     * ULP(X) = min{epsilon > 0 | (x + epsilon) > x}. Per stessi ordini di grandezza di X
     * (stesso esponente), avremo uno stesso ULP, perche' anche se epsilon non ci da
     * direttamente il successivo a x, per il RN saltiamo direttamente
     * al valore successivo di X */
}

/* In virtu' della definizione, troviamo un valore ulp, tale che Eps > x */
float Calcola_ULP(float x)
{
    float Ulp = x;
    float Eps; //x+Epsilon
    int n=0;

    Eps = Ulp + x;
    while (Eps > x)
    {
        Ulp = Ulp/2;
        Eps = Ulp+x;
        n++;
    }
    printf("\n n = %d \n", n-1); //Escludiamo il ciclo in cui non ci da contributo
    return Ulp *= 2.0f; //ho trovato l'epsilon, devo ripristinare l'ultima divisione
}

int estrae_esponente(int n)
```

```

int mask,int esp;
mask=0x7f800000;
esp = n&mask; //Estraggo l'esponente e lo porto tutto a destra
esp = (esp>>23);
esp = esp-bias; //Trasformo la rapp. bias in quella reale
return esp;
}

```

**OUTPUT:**

\*----- U.L.P. -----\*  
Inserisci un numero reale: 1

x = 1.000000  
n\_cicli = 23

ULP<1.000000> calcolato = 1.19209e-007  
ULP<1.000000> trovato = 1.19209e-007

\*----- U.L.P. -----\*  
Inserisci un numero reale: 4

x = 4.000000  
n\_cicli = 23

ULP<4.000000> calcolato = 4.76837e-007  
ULP<4.000000> trovato = 4.76837e-007

\*----- U.L.P. -----\*  
Inserisci un numero reale: 256

x = 256.000000  
n\_cicli = 23

ULP<256.000000> calcolato = 3.05176e-005  
ULP<256.000000> trovato = 3.05176e-005

## ESERCIZIO 16 [LIV.2]

```
/** [liv.2] Generando in modo random i bit di un numero reale x(doublex), determinare i bit della corrispondente rappresentazione float flx(float flx; flx=(float) x). Se il numero x è rappresentabile nel tipo float, calcolarne l'errore assoluto EA e l'errore relativo ER di rappresentazione (considerando come esatto double x e come approssimante float flx) */
#include <stdio.h>
#include <float.h>
#include <stdlib.h>
#include <time.h>

long long sequenza_randmax();
double errore_rel(double , float);
double errore_ass(double , float);

int main()
{
    /* Trattiamo la variabile come intero per costruire il bit e float per ricavarci gli errori */
    union
    {
        double x_d; /* double */
        long long int x_i;
    }casuale;

    float flx;
    double errore_assoluto=0,errore_relativo=0;
    casuale.x_i=0;

    /* Genero un intero da 8 byte (64bit) che corrisponde ad un floating point.
       Come richiesto dal problema, la parola avra' un double max e min,
       ma noi vogliamo vedere se e' possibile rappresentarlo in float */
    printf("\BIT GENERATO = ");
    casuale.x_i = sequenza_randmax();

    /* Scrambiamo i MAX e MIN valori di rappresentazione float per sapere se si potra'
       rappresentare il double generato nel float*/
    printf("\n\nVALORE GENERATO(DOUBLE) = %e\n\n", casuale.x_d);
    printf("FLT_MAX = %e", FLT_MAX);
    printf("\nFLT_MIN = %e\n", FLT_MIN);

    /* Se il valore assoluto e' compreso tra le 2 costanti,
       possiamo procedere l'algoritmo */
    if(fabs(casuale.x_d)<=FLT_MAX && fabs(casuale.x_d) >=FLT_MIN)
    {
        printf("\nVALORE DOUBLE RIENTRA NEL RANGE FLOAT[FLT_MIN, FLT_MAX]\n\n");
        flx=(float)casuale.x_d; /* Lo trasformo da DOUBLE a FLOAT*/
        printf("VALORE TRASFORMATO(FLOAT) = %e", flx);
        /* Calcolo gli errori per vedere quanta precisione ho perso
           PS il double e' pur sempre finito e discreto, ma avendo
           un range molto piu' gigante dei float, facciamo finta che
           rappresenti il valore reale infinito */
        errore_relativo=errore_rel(casuale.x_d,flx);
        errore_assoluto=errore_ass(casuale.x_d,flx);
        printf("\nL'errore RELATIVO e' %e",errore_relativo);
        printf("\nL'errore ASSOLUTO e' %e\n\n",errore_assoluto);
    }

    else
    {
        printf("\nNON E' POSSIBILE PROCEDERE NEL CALCOLO!\n\nIl valore DOUBLE generato supera
               il range ");
        printf("[FLT_MIN, FLT_MAX] e quindi non e'\npossibile rappresentarlo in FLOAT.\n\n");
    }
    return 0;
}

/* _____ Calcolo Errore Assoluto(Errore sulla cifre DECIMALI) _____ */
double errore_ass(double x, float flx)
{
    double errore_assoluto;
    errore_assoluto= fabs(x-flx);
    return errore_assoluto;
}
```

```

/* _____ Calcolo Errore Relativo(Errore sulla cifre SIGNIFICATIVE) _____ */
double errore_rel(double x, float flx)
{
    double errore_relativo;
    errore_relativo= fabs(x-flx)/fabs(x);
    return errore_relativo;
}
/*
 * _____ GENERA BIT _____
LONG LONG perche' necessito di 8 byte rappresentare un DOUBLE */
long long sequenza_randmax()
{
    short i;
    long long random=0;

    srand(time(NULL)); //Inizializzo il seme
    /* Generando un double, necessito di 64 bit*/
    for(i=0; i<64; i++)
    {
        /* Con Rand()&1 otteniamo 0 o 1, in base al primo bit di Rand(). Random intanto shiftera' a
           Sinistra per far spazio a questo bit che verra' aggiunto con una or. */
        random = random<<1|rand()&1;

        printf("%lld",random&1);
    }
    return random;
}
OUTPUT:
BIT GENERATO = 10111100011111110010010011000101111101111000100010100100101010111

VALORE GENERATO<DOUBLE> = -2.738419e-017

FLT_MAX = 3.402823e+038
FLT_MIN = 1.175494e-038

VALORE DOUBLE RIENTRA NEL RANGE FLOAT[FLT_MIN, FLT_MAX]

VALORE TRASFORMATO<FLOAT> = -2.738419e-017
L'errore RELATIVO e' 7.991731e-009
L'errore ASSOLUTO e' 2.188471e-025
NB: Il controllo e' se il DOUBLE non sia cosi' troppo grande o piccolo da non rientrare nel FLOAT.

BIT GENERATO = 011001011100010111100011001010110000110000101101010101101101101101101101110

VALORE GENERATO<DOUBLE> = 1.816440e+182

FLT_MAX = 3.402823e+038
FLT_MIN = 1.175494e-038

NON E' POSSIBILE PROCEDERE NEL CALCOLO!

Il valore DOUBLE generato supera il range [FLT_MIN, FLT_MAX] e quindi non e'
possibile rappresentarlo in FLOAT.

BIT GENERATO = 0011111011110110110000011110001011100001010011010101011001100101

VALORE GENERATO<DOUBLE> = 2.170312e-005

FLT_MAX = 3.402823e+038
FLT_MIN = 1.175494e-038

VALORE DOUBLE RIENTRA NEL RANGE FLOAT[FLT_MIN, FLT_MAX]

VALORE TRASFORMATO<FLOAT> = 2.170312e-005
L'errore RELATIVO e' 3.410376e-009
L'errore ASSOLUTO e' 7.401579e-014

```

## ESERCIZIO 17 [LIV.2]

```

/*
[liv.2] Scrivere una function C per valutare un polinomio mediante l'algoritmo
di Horner. Applicare l'algoritmo ai dati dell'esempio 3 nelle dispense
calcolando l'errore relativo. Usare una versione dell'algoritmo a
precisione estesa per ottenere il valore "esatto".
*/
#include <stdio.h>
#include <stdlib.h>
#include <float.h>
#include <math.h>

double alg_di_Horner(double [],double, short);
double cal_val_rel(double , float );

void main()
{
    short grado,i;
    double coef[10]; //Array contenente i coefficienti del polinomio
    double Err_relativo=0.0, x, risultato_Horner=0.0;
    float float_risultato_Horner=0.0;

    printf("Inserire il grado del polinomio: ");
    scanf("%hd",&grado);
    fflush(stdin);

    /* _____ Inserimento dei coefficienti(Dal piu' alto al piu' piccolo) _____ */
    for(i=grado;i>=0;i--)
    {
        printf("\nInserisci il coefficiente di x^%hd: ",i);
        scanf("%lf",&coef[i]);
        fflush(stdin);
    }

    /* _____ INSERIRE IL PUNTO DI DOVE SI VUOLE CALCOLARE IL POLINOMIO _____ */
    printf("\nAdesso inserisci la x di P(x) per cui vuoi calcolare il polinomio: ");
    scanf("%lf",&x);

    risultato_Horner = alg_di_Horner(coef,x,grado); //Risultato preso come "REALE" = X
    printf("\nP(%lf) = %lf\n",x,risultato_Horner);

    float_risultato_Horner = (float)risultato_Horner; //Risultato in FLOAT =~ X

    /* _____ CALCOLO DELL'ERRORE RELATIVO _____ */
    Err_relativo = cal_val_rel(risultato_Horner, float_risultato_Horner);

    /* _____ OUTPUT _____ */
    printf("\nP(x) DOUBLE = %22.16e\nP(x) FLOAT = "
    "%22.16e\n",risultato_Horner,float_risultato_Horner);
    printf("\nErrore Relativo = %g\n1/2Eps = %g\n",Err_relativo, 0.5*FLT_EPSILON);
    /* _____ CALCOLO DELL'ACCURATEZZA MASSIMA STATICIA _____ */
    if(Err_relativo>(1/2)*FLT_EPSILON)
        printf("\nL'errore non e' di massima accuratezza Er>1/2*EpsMach\n");
    else
    printf("\nL'errore e' di massima accuratezza Er<=1/2*EpsMach\n");

    system("pause");
}

/*
HORNER ITERATIVO: : effettuando un ciclo for per 'grado' volte e partendo dal
coefficiente di grado più basso, la funzione calcola il valore
del polinomio in un punto prefissato
P(x)= c[N]+x(c[N-1]+x(( ... c[3]+x( c[2]+x ( c[1]+c[0]*x) )))...)
*/
double alg_di_Horner(double c[],double x, short grado)
{
    short i=grado;
    long double ris=c[0];

    for(i=1;i<=grado;i++)
        ris = c[i]+x*ris;

    return ris;
}

```

```
/* CALCOLO ERRORE RELATIVO */
double cal_val_rel(double x, float flx)
{
    double err=0.0;
    err=fabs((x-flx))/fabs(x);
    return err;
}
```

**OUTPUT:**

Inserire il grado del polinomio: 3  
Inserisci il coefficiente di  $x^3$ : 2  
Inserisci il coefficiente di  $x^2$ : 0  
Inserisci il coefficiente di  $x^1$ : 1  
Inserisci il coefficiente di  $x^0$ : 2  
Adesso inserisci la x di  $P(x)$  per cui vuoi calcolare il polinomio: 4  
 $P(4.000000) = 146.000000$   
 $P(x)$  DOUBLE = 1.460000000000000e+002  
 $P(x)$  FLOAT = 1.460000000000000e+002  
Errore Relativo = 0  
1/2Eps = 5.96046e-008

L'errore e' di massima accuratezza Errore relativo<=1/2\*EpsMach

Inserire il grado del polinomio: 8  
Inserisci il coefficiente di  $x^8$ : 2  
Inserisci il coefficiente di  $x^7$ : 1  
Inserisci il coefficiente di  $x^6$ : -3  
Inserisci il coefficiente di  $x^5$ : 4  
Inserisci il coefficiente di  $x^4$ : 2  
Inserisci il coefficiente di  $x^3$ : 0  
Inserisci il coefficiente di  $x^2$ : 9  
Inserisci il coefficiente di  $x^1$ : 4  
Inserisci il coefficiente di  $x^0$ : 2  
Adesso inserisci la x di  $P(x)$  per cui vuoi calcolare il polinomio: 20  
 $P(20.000000) = 56896350822.000000$   
 $P(x)$  DOUBLE = 5.689635082200000e+010  
 $P(x)$  FLOAT = 5.689635225600000e+010  
Errore Relativo = 2.52037e-008  
1/2Eps = 5.96046e-008

L'errore non e' di massima accuratezza Errore relativo>1/2\*EpsMach

|   |   |   |
|---|---|---|
| $f_l(x)$ risultato di<br>massima accuratezza                        |  | $f_l(x) = x(1 + \delta)$ ,<br>$ \delta  \leq \frac{1}{2} \epsilon_{mach}$ |
| dove $\delta = \frac{ x - f_l(x) }{ x }$ , $\delta$ errore relativo |   |   |

## ESERCIZIO 18 [LIV.1]

/\* [LIV1] Scrivere function C per calcolare una somma di molti addendi dello stesso ordine di grandezza. A scelta la versione dell'algoritmo di somma a gruppi. (Algoritmo del raddoppiamento)  
Risolvere il seguente problema: (soluzione tra gli output)

$N = 10^8, a_k = 10^{-6}, \forall k = 1, \dots, N \Rightarrow \sum_{k=1}^N a_k = \sum_{k=1}^{10^8} 10^{-6} = 100$

```
#include <stdio.h>
#include <stdlib.h>
float somma_blocchi(unsigned int, float []);

void main()
{
    float somma, *addendi; unsigned int n, k;
    float num; //numero reale da sommare
    printf("Inserisci il numero di somme da effettuare: ");
    scanf("%u", &n);
    addendi = (float*)malloc(n*sizeof(float)); //Allociamo dinamicamente: e' richiesto molto spazio
    fflush(stdin);

    printf("inserisci il numero reale da sommare: ");
    scanf("%f", &num);

    for(k=0; k<n; k++)
        addendi[k] = num; //Addendi dello stesso ordine

    somma=somma_blocchi(n, addendi);
    printf("\n\nSomma \nesp = %e\ndec = %4.2f", somma, somma);
}

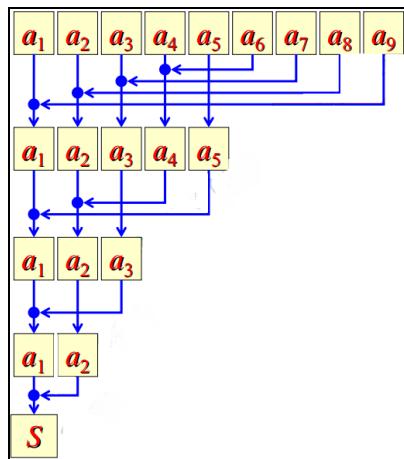
/* RADDOPPIAMENTO */
float somma_blocchi(unsigned int N, float A[])
{
    unsigned int k, i=1, Nmez;

    while(N>1)
    {
        /* Dividi N e fai NMEZ somme. Il primo con l'ultimo, secondo col penultimo... */
        Nmez=N/2;
        for(k=0; k<Nmez; k++)
            A[k]=A[k]+A[N-1-k];
        /* Se N e' dispari allora N=Nmez+1 altrimenti l'elemento di mezzo non considerato tra le N Dispari somme, si perde.*/
        if(N%2 == 0)
            N=Nmez;
        else
            N=Nmez+1;
    }
    /* La somma lavora in place, conservando le somme sui K cicli. L'ultimo elemento sara' la somma totale (Algoritmo del raddoppiamento)*/
    return A[0];
}
```

### OUTPUT:

Inserisci il numero di somme da effettuare: 100000000  
inserisci il numero reale da sommare: 0.000001

Somma  
esp = 1.000000e+002  
dec = 100.00



## ESERCIZIO 19 [LIV.1]

```

/*
 [liv1] Scrivere una function C di somma iterativa con criterio di arresto naturale.
*/
#include <stdio.h>
#include <stdlib.h>
#include <float.h>

float Somma_criterio_naturale(float x, float *somma, int *n);

int main()
{
    float x,somma=0.0, last_a;
    int n=0; //contatore dei cicli

    printf("Inserire valore X: "); scanf("%f",&x);
    last_a = Somma_criterio_naturale(x, &somma,&n);

    printf("\nSomma con criterio di arresto naturale: %e", somma);
    printf("\nNumero di cicli: %d", n);
    printf("\nUltimo valore di X che ha dato contributo alla somma ULP(S): %e\n\n", last_a);
    return 0;
}

/*
Il criterio di arresto naturale serve per evitare somme inutili di addendi non
contributivi rispetto a S.
Possiamo avere l'approssimazione di  $ULP(S)=S*Emach*0.5$ 
*/
float Somma_criterio_naturale(float x, float *somma, int *n)
{
    /* while( $x \geq ULP(S)$ )
       Cicla fin quando x non diventa piu' piccolo di ULP(S) e , quindi, non dara' piu' contributo
       alla somma.
       NB: Emach diviso 2 per il round to nearest */
    while (x>(*somma)*FLT_EPSILON*0.5)
    {
        *somma = *somma + x; //Somma ad x
        x = x/2;
        *n+=1; //Conta ciclo
    }
    x = x*2; //riprisina l'ultimo valore di X che ha dato contributo
    *n=*n-1; //Con l'ultima moltiplicazione, ho riprisinato il vecchio n
    return x;
}

OUTPUT:
Inserire valore X: 1

Somma con criterio di arresto naturale: 2.000000e+000
Numero di cicli: 23
Ultimo valore di X che ha dato contributo alla somma ULP(S): 1.192093e-007
NB: Con questo algoritmo calcoliamo l'ULP(X). Sopra e' presente l'ULP(1), ossia Epsilon
Macchina.

Da notare anche il numero di cicli '23' perche' effettua 23 shift a destra (partendo dal
bit隐式) per avere un 1 sull'ultimo bit della mantissa e coai avere un valore che dia
ancora un contributo alla somma.

Inserire valore X: 128

Somma con criterio di arresto naturale: 2.560000e+002
Numero di cicli: 23
Ultimo valore di X che ha dato contributo alla somma ULP(S): 1.525879e-005

```

## ESERCIZIO 20 [LIV.1]

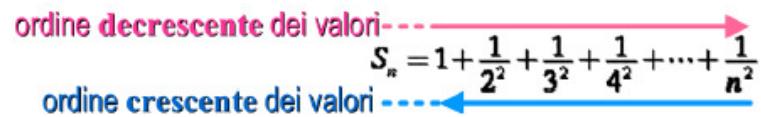
/\*\* [liv.1] Scrivere una function C per calcolare una somma di addendi ordinati, rispettivamente in ordine crescente ed in ordine decrescente \*\*/

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define MAX_DIM 50
```

/\* Calcolo contemporaneamente la somma in ordine crescente e decrescente  
Sommando in ordine crescente, avremo un RISULTATO DI MASSIMA ACCURATEZZA \*/

$$S_n = \sum_{k=1}^n \frac{1}{k^2} \approx \frac{\pi^2}{6}$$

$\left\{ \frac{1}{k^2} \right\}$  decrescente



```
float somma_crescente(int);
float somma_decrescente(int);
double errore_relativo(double, float);
int main()
{
    float Sum_Cre, Sum_Dec;
    double x_esatto = (M_PI*M_PI)/6; //La successione da calcolare si approssima a pi^2/6
    double Er_rel_cre, Er_rel_dec;
    int n;

    printf("Inserisci il numero di somme:\n");
    scanf("%d", &n);
    /* CALCOLO SOMMA CRESCENTE E DECRESCENTE CON RELATIVI ERRORI RELATIVI */
    Sum_Cre = somma_crescente(n);
    Sum_Dec = somma_decrescente(n);
    Er_rel_cre = errore_relativo(x_esatto, Sum_Cre);
    Er_rel_dec = errore_relativo(x_esatto, Sum_Dec);

    printf("Sum Cre %e!\nSum Dec %e!\nErrore relativo cre %e\nErrore relativo dec %e", x_esatto,
    Sum_Cre, Sum_Dec, Er_rel_cre, Er_rel_dec);

    return 0;
}
/* Somma crescente somma dal 1/n^2 a 1 */
float somma_crescente(int n)
{
    int i;
    float sum=0.0f;
    for (i=n; i>0; i--)
        sum=sum+(1.0/(i*i));
    return sum;
}
/* Somma decrescente somma dal 1 a 1/n^2 a */
float somma_decrescente(int n)
{
    int i;
    float sum=0.0f;
    for (i=1; i<=n; i++)
        sum=sum+(1.0/(i*i));
    return sum;
}
double errore_relativo(double x_esatto, float flx)
{
    return fabs(x_esatto-flx)/(fabs(x_esatto));
}
```

### OUTPUT:

Inserisci il numero di somme:  
40000  
Sum Cre 1.644934e+000!  
Sum Dec 1.644909e+000!  
Errore relativo cre 1.644725e+000  
Errore relativo dec 1.522408e-005

/\* PERCHE' ERRORE RELATIVO CRESCENTE E' MOLTO PIU' ACCURATO? \*/  
Si vede che quando sommiamo i termini in ordine decrescente l'errore e' sempre piu grande di quello ottenuto sommando i termini in ordine crescente, addirittura risulta costante. Il problema e' dovuto al fatto che, da una certa iterazione in poi, l'n-esimo termine della successione avra avuto valore inferiore all'ULP dell'attuale somma parziale, non portando dunque alcun contributo al risultato che invece avrebbe dovuto essere diverso. \*/

Inserisci il numero di somme:  
300  
Sum Cre 1.644934e+000!  
Sum Dec 1.641606e+000!  
Errore relativo cre 1.641606e+000  
Errore relativo dec 2.023021e-003

## ESERCIZIO 21 [LIV.2]

```
/** [liv.2] Scrivere una function C per calcolare la somma di addendi di segno alternato. **/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define DIM 80

float somma_normale(float Vet[], int n);
float somma_controllata(float Vet[], int n);
double somma_reale(float Vet[], int n); //Riferimento REALE
int main()
{
    float a[DIM]; //Array di input
    double sum_reale=0.0;
    int i; //Contatore per i cicli
    int n; //Max numero di elementi da sommare: Genera vettore della successione (-2)^k di n elementi

    printf ("Quanti elementi della successione generare?[MAX 200]: ");
    scanf ("%d", &n);

    /* Crea vettore di elementi a segno alternato */
    for (i=0; i<n; i++) a[i]=(float)pow(-2,i);

    /* Ricavo il valore reale della somma */
    sum_reale = somma_normale(a, n);
    printf("Somma normale\t : %f\n", somma_normale(a, n));
    printf("Somma controllata: %f\n", somma_controllata(a, n));
    printf("Errore relativo Somma normale\t : %g\n",
           fabs(sum_reale-somma_normale(a, n))/ fabs(sum_reale));
    printf("Errore relativo Somma controllata: %g ",
           fabs(sum_reale-somma_controllata(a, n))/fabs(sum_reale));
    return 0;
}

/* La function Somma in modo separato gli addendi positivi e negativi */
float somma_controllata(float Vet[], int n)
{
    float sum_pos=0, sum_neg=0;
    int i;
    //Somma tutti gli elementi positivi della successione
    for (i=0; i<n; i=i+2) sum_pos=sum_pos+Vet[i];
    //Somma tutti gli elementi negativi della successione
    for (i=1; i<n; i=i+2) sum_neg=sum_neg+Vet[i];
    return sum_pos+sum_neg;//Ritorna la somma delle due sommatorie(+/-)
}

/* La function Somma in modo condiviso sia gli addendi positivi e che negativi */
float somma_normale(float Vet[], int n)
{
    float sum=0;
    int i;
    for (i=0; i<n; i++) sum=sum+Vet[i];
    return sum;
}

/* La function Somma in modo separato gli addendi positivi e negativi DOUBLE */
double somma_reale(float Vet[], int n)
{
    double sum_pos=0.0, sum_neg=0.0;
    int i;
    //Somma tutti gli elementi positivi della successione
    for (i=0; i<n; i=i+2) sum_pos=sum_pos+Vet[i];
    //Somma tutti gli elementi negativi della successione
    for (i=1; i<n; i=i+2) sum_neg=sum_neg+Vet[i];
    return sum_pos+sum_neg;//Ritorna la somma delle due sommatorie(+/-)
}
```

### OUTPUT:

```
Quanti elementi della successione generare?[MAX 200]: 100
Somma normale : 2112750874450609500000000000000.000000
Somma controllata: 2112751063345268800000000000000.000000
Errore relativo Somma normale : 5.96046e-008
Errore relativo Somma controllata: 2.98023e-008
```

```
/* CONCLUSIONE: se faccio insieme la somma di addendi a segno alternato,
   ho un valore meno accurato rispetto ad una somma in cui separo
   gli addendi positivi e negativi. */
```

## ESERCIZIO 22 [Quiz]

# QUIZ

**Quiz:** che differenza c'è tra i due codici C che seguono:

```
void main()
{
    char stringa[ ] = "ciao";
    puts(stringa);
    *stringa = 'm';
    puts(stringa);
}
```

```
void main()
{
    char *stringa = "ciao";
    puts(stringa);
    *stringa = 'm';
    puts(stringa);
}
```

Tra i 2 codici in C la differenza e' grande:

nel primo codice si dichiara staticamente un array per contenere i caratteri della stringa costante, mentre nel secondo caso viene dichiarato un puntatore che punta all'indirizzo di una stringa costante. Il primo programma **modifica il primo valore puntato dall'array** stringa[] (ossia il puntato dall'indirizzo base) e viene inserito un altro carattere.

Nel secondo codice **stiamo tentando di modificare il carattere di una stringa costante** puntato da \*stringa: ciò e' impossibile dato che il puntatore punta ad una stringa costante e l'unica operazione che si può effettuare e' modificare l'indirizzo a cui punta il puntatore.

Provando i codici il **primo funziona**, mentre il **secondo crasha** perché stiamo facendo una "violazione di accesso", appunto, perche' non possiamo modificare una costante.

## ESERCIZIO 23 [LIV.1]

```
/*
[liv.1] Confrontando i risultati con quelli delle relative funzioni del C ed utilizzando per le
stringhe
    o l'allocazione statica
    o l'allocazione dinamica
scrivere una function C che accetta in input il numero n e legge da tastiera n caratteri (uno
alla volta)
costruendo la stringa che li contiene (output) senza usare strcat(...).
*/
#include <stdio.h>
#include <stdlib.h>
void Allocazione_Statica(int n, char *TestoStat);
void Allocazione_Dinamica(int n, char **TestoDin);

int main()
{
    char TestoStat[16], *TestoDin;
    int n=0;
    char Scelta;

    printf("Scegliere il tipo di allocazione\n[1] Statica\n[2] Dinamica\n");
    scanf("%d", &Scelta);
    switch(Scelta)
    {
        case 1:   printf("Inserisci la dimensione il numero di carattere da inserire:\n");
                   scanf("%d", &n);
                   Allocazione_Statica(n, TestoStat); /*Passo direttamente l'indirizzo base */
                   printf("I caratteri messi sono: %s", TestoStat);
                   break;

        case 2:   printf("Inserisci la dimensione il numero di carattere da inserire:\n");
                   scanf("%d", &n);
                   Allocazione_Dinamica(n, &TestoDin); /* Passo l'indirizzo del puntatore!!!
                                                 Cio' perche' voglio modificare il
                                                 contenuto del puntatore,
                                                 assegnandogli l'indirizzo
                                                 allocato dinamicamente, lavorando
                                                 per riferimento (o indirizzo).
                                                 */
                   printf("I caratteri messi sono: %s", TestoDin);
                   break;
        default:  printf("Scelta non valida");
                   break;
    }
    free(TestoDin);
    return 0;
}

/*
* _____ ALLOCAZIONE STATICIA _____
Semplicemente passiamo l'indirizzo base del vettore e andiamo a lavorare
sul puntato di questi indirizzi. Ritornando al programma chiamante,
le modifiche ai valori puntati in memoria restano.
Se passo Vet[] alla function e faccio Vet[0]=Qualcosa, vado ad inserire
all'indirizzo puntato da Vet+0 il valore.
Se modiflico l'indirizzo, e quindi modificando la variabile puntatore,
quando torna al programma chiamante, lo stack ripristina il valore del puntatore
perche' il valore da riprisintare e' stato passato come parametro (Base Pointer).
*/
void Allocazione_Statica(int n, char TestoStat[])
{
    int i;
    for(i=0; i<n; i++) // -1 perche' l'ultimo elemento e' \0
    {
        fflush(stdin);
        printf("Inserisci il carattere %d:\n", i);
        TestoStat[i] = getchar();
    }
    TestoStat[i]='\0';
}
```

```

/*
    _____ ALLOCAZIONE DINAMICA _____
La situazione cambia: ora passo un puntatore doppio.
Ragione: Io ho la necessita' di modificare il puntatore in modo "permanente",
dato che gli dovrò assegnare l'indirizzo allocato a cui punterà'.
Per fare ciò, devo passare l'indirizzo del puntatore.
Avendo l'indirizzo del puntatore, accedo al puntatore (*) e di conseguenza
al valore puntato del puntatore (**).
Indirizzo Del Puntatore -> (*) Puntatore(indirizzo del valore)-> (**) Puntato dal
puntatore.
*/
void Allocazione_Dinamica(int n, char **TestoDin)
{
    int i;
    *TestoDin = (char*)malloc(sizeof(char)*n+1); /* Allora n char e converti il puntatore a
                                                void* a char*. +1 e' per il '\0'*/
    for(i=0; i<n; i++) // -1 perche' l'ultimo elemento e' \0
    {
        fflush(stdin);
        printf("Inserisci il carattere %d:\n", i);
        (*TestoDin+i) = getchar(); /* Il puntato dell'indirizzo base e' un puntatore. Il
                                    puntato del puntato Indica il puntato del puntatore
                                    (il valore)*/
    }
    (*TestoDin+n) = '\0'; /* (*TestoDin)[i] */
}

```

**OUTPUT:**

Scegliere il tipo di allocazione  
[1] Statica  
[2] Dinamica

1  
Inserisci la dimensione il numero di carattere da inserire:  
3  
Inserisci il carattere 0:  
Z  
Inserisci il carattere 1:  
I  
Inserisci il carattere 2:  
O  
I caratteri messi sono: ZIO

Scegliere il tipo di allocazione  
[1] Statica  
[2] Dinamica

2  
Inserisci la dimensione il numero di carattere da inserire:  
5  
Inserisci il carattere 0:  
P  
Inserisci il carattere 1:  
E  
Inserisci il carattere 2:  
P  
Inserisci il carattere 3:  
P  
Inserisci il carattere 4:  
E  
I caratteri messi sono: PEPPE

## ESERCIZIO 24 [LIV.1]

```
/*
[liv.1] Confrontando i risultati con quelli delle relative funzioni del C ed utilizzando per le stringhe
    o l'allocazione statica
    o l'allocazione dinamica
scrivere una function C che restituiscia la concatenazione di due stringhe date in input senza usare strcat(...).
E' a scelta restituire la concatenazione delle due stringhe in una terza variabile (parametro di output o function stessa) oppure nella prima delle due variabili di input.
**/


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

void Concatenazione(char *S1, char *S2, char *Testo, int N1, int N2);

int main()
{
    /* Variabili usate per l'allocazione statica */
    char TestoStat[32], Str1[16], Str2[16];

    /* Variabili usate per l'allocazione dinamica */
    char *Str1_Din, *Str2_Din, *TestoDin;
    int n, m;

    /* _____ Input stringhe _____ */
    printf("Inserisci la prima stringa\n");
    gets(Str1);
    printf("Inserisci la seconda stringa\n");
    gets(Str2);

    n=strlen(Str1); m=strlen(Str2); //Mi conservo le dim per le successive allocazioni

    /* _____ CREO LE STRINGHE DINAMICAMENTE _____ */
    Str1_Din = (char *)malloc(strlen(Str1)+1); //Allocata strlen(Str1) spazio. +1 per il '\0'
    if (Str1_Din==NULL) //controlla se ti restituisce una locazione di memoria
    {printf("Memoria insufficiente"); exit(1);}
    Str2_Din = (char *)malloc(strlen(Str2)+1);
    if (Str2_Din==NULL)
    {printf("Memoria insufficiente"); exit(1);}

    /* Strcpy, mi copia anche il '\0' */
    strcpy(Str1_Din, Str1);
    strcpy(Str2_Din, Str2);

    /* _____ CONCATENAZIONE STATICÀ _____ */
    /* Passo le stringhe Statiche*/
    Concatenazione(Str1, Str2, TestoStat, n, m); /*Passo l'indirizzo base */
    printf("Concatenazione con ALLOCAZIONE STATICÀ: %s\n", TestoStat);

    /* _____ CONCATENAZIONE DINAMICA _____ */
    /* Allocata spazio per il risultato e passo le stringhe dinamiche */
    TestoDin = (char *)malloc(n+m+1); /* Allocata a TestoDin n+m+1(+1 e' per il '\0') */

    Concatenazione(Str1_Din, Str2_Din, TestoDin,n,m); /* Passo L'indirizzo base */
    printf("Concatenazione con ALLOCAZIONE DINAMICA: %s", TestoDin);

    return 0;
}

/*
_____ CONCATENAZIONE _____
Passiamo le stringhe, la stringa risultato e le dimensioni delle stringhe.
Effettuiamo il semplice algoritmo di concatenazione.

```

PS Passando un ARRAY STATICO o DINAMICO, li posso trattare tutti e 2 alla stessa maniera, nel senso che a posta della notazione a puntatore, potevo usare quella classica (\*Vet+2 <=> Vet[2])

```
/*
void Concatenazione(char *S1, char *S2, char *Testo, int N1, int N2)
{
    int i, j;

    /* Copia la prima stringa */
    for(i=0; i<N1; i++)
        *(Testo+i) = *(S1+i);
    /* Copia la seconda stringa a partire da i (da dove si era fermato precedentemente) */
    for(j=0; j<=N2; j++, i++) // j<=N2, cosi' copia alla fine '\0'
        *(Testo+i) = *(S2+j);
}
```

#### OUTPUT:

```
Inserisci la prima stringa
Piano
Inserisci la seconda stringa
forte
Concatenazione con ALLOCAZIONE STATICÀ: Pianoforte
Concatenazione con ALLOCAZIONE DINAMICA: Pianoforte
```

```
Inserisci la prima stringa
pan
Inserisci la seconda stringa
carre
Concatenazione con ALLOCAZIONE STATICÀ: pancarrè
Concatenazione con ALLOCAZIONE DINAMICA: pancarrè
```

## ESERCIZIO 25 [LIV.1]

```
/**[liv.1]Confrontando i risultati con quelli delle relative funzioni del C, scrivere una
function C che restituisca la prima occorrenza di una sottostringa in una stringa senza usare
strstr(...). */
#include <stdio.h>
#include <stdlib.h>
#define TROVATO 1
#define NON_TROVATO -1
int TrovaParole(char Testo[], char Chiave[], int n, int m);
int UgualeStringa(char Testo[], char Chiave[], int m);

int main()
{
    char Testo[]="ananas", Chiave[]="nana";
    int m = strlen(Chiave), n = strlen(Testo), k=0, i;

    printf("Stringa: %s\nPattern: %s\n\n", Testo, Chiave);
    k = TrovaParole(Testo, Chiave, n, m);
    if(k>=0)
    {
        printf("Il pattern e' stato trovato alla posizione %d: ", k);
        for(i=k; k<i+m; k++)
            printf("%c", Testo[k]);
    }
    else printf("Pattern non trovato!\n\n");
    printf("\n\n");

    return 0;
}

/*
    _____ CERCA LA PRIMA OCCORRENZA
    Non appena trova la prima sottostringa coincidente con la chiave,
    restituisci un segnale di TROVATO, altrimenti alla fine restituira'
    NONTROVATO.
*/
int TrovaParole(char Testo[], char Chiave[], int n, int m)
{
    int i=0;
    /* Cicla la stringa, fino a n-m+1 perche' oltre non avrebbe senso (se vado oltre,
       e' ovvio che non ci sara' il pattern)*/
    do
    {
        if(UgualeStringa( &(Testo[i]), Chiave, m ) !=NON_TROVATO)
            return i; /* Sto ritornando semplicemente i, ma si vuole dimostrare che e' la stessa
                       cosa se sottraggo l'indirizzo base a indirizzo corrente */
        i++;
    }while(i<n-m+1);

    return NON_TROVATO;
}
/* CONTROLLA SE LA PORZIONE DI STRINGA E LA SOTTO STRINGA COINCIDONO */
int UgualeStringa(char Testo[], char Chiave[], int m)
{
    int i;
    /* Se nel ciclo non entrera' mai nell'if, significa che la parola sara'
       esatta e restituiamo TROVATO per dire che coincide il pattern */
    for(i=0; i<m; i++)
    {
        if(Testo[i] !=Chiave[i]) //Se uno dei m caratteri sono diversi, restituisce 0
            return NON_TROVATO; //Sto restituendo 0 come avviso
    }
    return TROVATO; //Arrivato qui, mi dice che ho trovato il primo pattern
}
```

### OUTPUT:

Stringa: ananas  
 Pattern: nana

Il pattern e' stato trovato alla posizione 1: nana

Stringa: ananas  
 Pattern: asna

Pattern non trovato!

## ESERCIZIO 26 [LIV.2]

```
/*
Usando l'allocazione dinamica e le funzioni C per manipolare le stringhe, scrivere una
function C che restituisca la posizione di tutte le occorrenze di una sottostringa
in una stringa ed il loro numero totale.
    o [liv.1]ne visualizzi la posizione di tutte le occorrenze trovate.
    o [liv.2]restituisca in un array la posizione di tutte le occorrenze trovate.
**/


/* Il programma del sottoscritto rispetto entrambi i criteri di livello */

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

int *StrStr_Pos(char Testo[], char Chiave[], int n, int m, int *n_pos);

int main()
{
    char Testo[]="ananZZanZZan", Chiave[]="an";
    int m = strlen(Chiave), n = strlen(Testo), i;
    int *Conta_Pos, n_pos=0; //Conta_Pos=Array delle posizioni e n_pos=Dimensione dell'array

    printf("Testo: %s\nPattern: %s\n\n", Testo, Chiave);

    Conta_Pos = StrStr_Pos(Testo, Chiave, n, m, &n_pos);

    if(Conta_Pos)
    {
        printf("Le %d posizioni trovate sono negli indici: \n", n_pos);
        for(i=0;i<n_pos; i++)
            printf("Posizione indice: %d\n", Conta_Pos[i]);
    }
    else printf("Non e' stata trovata nessuna parola\n");

    free(Conta_Pos);
    return 0;
}

int *StrStr_Pos(char Testo[], char Chiave[], int n, int m, int *n_pos)
{
    int *ch; //contiene i risultati di strstr()
    int Pos_trovata, *array_n;
    int i=0; //Indice controllo. Se trova un'occorrenza, salta di m

    /* _____ TROVA PRIMA OCCORRENZA _____ */
    /* In questa prima parte, cerco solo per la prima volta, il primo pattern.
       Se c'e', alloco con malloc ad un indirizzo dove successivamente(se esisteranno
       altri pattern) riallocherà'(REALLOC) sullo stesso indirizzo, tenendo conto
       dei numero delle posizioni. In tal modo avro' un vettore di n_pos
       riducendo al minimo lo spazio da occupare. Se la prima volta non trova
       nulla, restituisce un indirizzo a NULL, per segnalare il mancato matching
       di almeno una sottostringa*/
    ch = strstr(&(Testo[i]), Chiave); /* Cerca il pattern. Passo indirizzo base di Testo*/

    if(ch) /* Ho trovato qualcosa? */
    {
        *n_pos+=1; /* Ho almeno una posizione. Sarebbe n_pos = 1*/
        array_n=(int *)malloc(*n_pos*sizeof(int)); /* Richiedi indirizzo per allocare 1 byte
                                                       PS dovremo effettuare dei controlli,
                                                       ma il codice diventerebbe pesante per
                                                       il problema interessato */
        if(array_n==NULL) {printf("ERRORE DI ALLOCAZIONE");exit(EXIT_FAILURE);}

        Pos_trovata = (int)ch-(int)Testo; //Ricavo l'indice trovato tramite gli indirizzi
        array_n[0]= Pos_trovata; // *array_n = indice posizione trovata.

        i = Pos_trovata+m; /* passa alle prossime sotto stringhe dopo il primo pattern trovato
                           saltando il pattern gia' calcolato */
    }
    else return NULL; /* Ritorna un indirizzo a NULL (non trovo niente->non alloco niente!) */

    /* _____ Trova altri matching riallocando lo stesso array _____ */
    do
    {

```

```

ch = strstr(&(Testo[i]), Chiave); /* Cerca il pattern. Se lo trova, ricicla, e passa a
                                    m posizioni dopo */
if(ch)
{
    *n_pos+=1; // (*n_pos)++;Avanzo perche' ho trovato un pattern
    array_n=(int *)realloc(array_n, (*n_pos)*sizeof(int)); /* Rialloco per un ulteriore
                                                               valore della posizione */
    if(array_n==NULL) {printf("ERRORE DI ALLOCAZIONE");exit(EXIT_FAILURE);}

    Pos_trovata = (int)ch-(int)Testo; //Ricavo l'indice trovato
    array_n[*n_pos-1]= Pos_trovata; /* Inserisco la pos_trovata a n_pos-1
                                         perche' la prima posizione dell'array
                                         in C e' 0 */
    i = Pos_trovata+m; /* dovrà ripartire dall'indirizzo da dove ho trovato il pattern,
                         piu' m per saltare la sottostringa appena trovata e passare
                         ai successivi*/
}
}while(i<n-m+1&&ch!=NULL); /* Cicla fin quando strstr fornisce una ricerca, ossia un indirizzo
                                oppure i si trova prima di n-m+1 */

return array_n; //Ritorno l'indirizzo base su cui ho lavorato dinamicamente
}

```

#### OUTPUT:

Testo: ananZZanZZan  
Pattern: an

Le 4 posizioni trovate sono negli indici:  
Posizione indice: 0  
Posizione indice: 2  
Posizione indice: 6  
Posizione indice: 10

Testo: ananZZanZZan  
Pattern: ZZ

Le 2 posizioni trovate sono negli indici:  
Posizione indice: 4  
Posizione indice: 8

Testo: ananZZanZZan  
Pattern: AAA

Non è stata trovata nessuna parola

L'algoritmo di ricerca diretta richiede, nel caso peggiore, **NM** confronti tra caratteri, dove **N** è la lunghezza del **testo** ed **M** quella del **pattern**.

In problemi di *text editing* (elaborazione di testi), dove **N >> M**, l'algoritmo richiederà mediamente un numero di confronti **≈N**, in quanto difficilmente il ciclo interno verrà eseguito.

Tuttavia quando l'**alfabeto è binario** oppure il **testo è altamente ripetitivo** la ricerca diretta risulta estremamente lenta: è conveniente pertanto ricorrere ad algoritmi più veloci.

## ESERCIZIO 27 [LIV.2]

```

/** liv.2] Utilizzando per le stringhe
   o l'allocazione statica
   o l'allocazione dinamica
scrivere una function C che elimini tutte le occorrenze di una data sottostringa
in una stringa. **/


#include <stdio.h>
#include <stdlib.h>
#include<string.h>
void Elimina_Stringa_stat(char *Testo, char *Chiave, char Testo_Nuovo_stat[]);
void Elimina_Stringa_din(char *Testo, char *Chiave, char **Testo_Nuovo_din);
/* Alternativa: creare un array posizioni! */
int main()
{
    char Testo []="anZanZanAan", Chiave []="an";

    char Testo_Nuovo_stat[32]; /* Risultato in un array statico */
    char *Testo_Nuovo_din; /* In uno dinamico */

    printf("Testo Iniziale = %s\n", Testo);
    printf("Pattern da eliminare = %s\n\n", Chiave);

    //Restituisco in un vettore statico
    Elimina_Stringa_stat(Testo, Chiave, Testo_Nuovo_stat);
    //Restituisco in un puntatore doppio
    Elimina_Stringa_din(Testo, Chiave, &Testo_Nuovo_din);

    printf("Testo senza pattern (STATICO) = %s\n\n", Testo_Nuovo_stat);
    printf("Testo senza pattern (DINAMICO) = %s\n\n", Testo_Nuovo_din);

    return 0;
}
/* _____ ELIMINARE UN PATTERN DA UNA STRINGA (LAVORANDO STATICAMENTE) _____
Procedimento: Trovo la posizione del pattern nella stringa; Sarà l'indirizzo
fino a dove sposterò tutti gli elementi alla sua destra, "sovrascrivendogli"
a partire da src, cioè a partire dall'elemento diverso dal pattern.
Cicla fin quando non trova altri pattern.
Esempio: anZanZan. Primo ciclo trova "an" ad indirizzo 1000=(Dest) e lo possiamo eliminare
spostando gli elementi a partire dall'indirizzo 1000+2=(SRC). Trattando di un char,
se ho 8 elementi e voglio eliminare il primo an, farò 8-(SRC-INDIRIZZOBASE)+1 perché
copio sposto anche il '\0'.
Tutti questi dati vengono dati a MEMMCOPY(Dest, Src, Num_Byte);
*/
void Elimina_Stringa_stat(char *Testo, char *Chiave, char Testo_Nuovo_stat[])
{
    int Num_byte; /* Quanti elementi spostare?*/
    char *Dest, *Src; /* Dest=Fino a che indirizzo sposto?
                      Src=A partire da quale indirizzo devo spostare?
                      Sono parametri che devo dare a MEMMCOPY*/
    int N_Chiave = strlen(Chiave);
    int N_Testo = strlen(Testo);

    strcpy(Testo_Nuovo_stat, Testo); //Non voglio sporcare testo

    do
    {
        Dest = strstr(Testo_Nuovo_stat, Chiave); //Indirizzo della occorrenza
        if(Dest) //se c'e'
        {
            Src = Dest+N_Chiave; //Da quale elemento spostare i valori a destra verso sinistra
            Num_byte = N_Testo-(int)(Src-Testo_Nuovo_stat)+1; /* Quanti byte spostare?
                                                               N_Testo-(Indice di src).
                                                               +1 e' per il '\0' */
            memmove(Dest, Src, Num_byte); //Se lo vede memmove
            N_Testo -= N_Chiave; /* Con memmove la cancellazione avviene sovrascrivendo
                                  dei successivi caratteri; La lunghezza totale diminuirà
                                  di N_Chiave */
        }
        /*Cicla fin quando non trova piu' nessun pattern o e' l'ultimo */
    }while(Dest!=NULL && Dest<Testo_Nuovo_stat+N_Testo-N_Chiave);
}

```

```

/* _____ ELIMINARE UN PATTERN DA UNA STRINGA (LAVORANDO DINAMICAMENTE) _____
Procedimento: Trovo la posizione iniziale del pattern p[0] nella stringa; Sarà l'indirizzo
fino a dove sposterò tutti gli elementi alla sua destra, "sovrascrivendogli"
a partire da src, cioè a partire dall'elemento diverso dal pattern (p[N_Chiave]). 
Cicla fin quando non trova altri pattern.
Esempio: anZanZan. Primo ciclo trova "an" ad indirizzo 1000=(Dest) e lo possiamo eliminare
spostando gli elementi a partire dall'indirizzo 1000+2=(SRC). Trattando di un char,
se ho 8 elementi e voglio eliminare il primo an, farò N_TESTO-(SRC-INDIRIZZOBASE)+1 perché
sposto anche il '\0'.
Tutti questi dati vengono dati a MEMMCOPY(Dest, Src, Num_Byte);
PS DINAMICO perché il testo risultato viene creato dinamicamente e rialloco sempre minor
spazio ogni volta che cancello un pattern, così da sfruttare tutte le N_TESTO+1 spazi di memoria.
Il puntatore è doppio perché la modifica che faccio al puntatore (quindi modificando l'indirizzo
a cui punterà), voglio che sia effettiva al programma chiamante. Se avessi fatto il puntatore
singolo, le modifiche ai valori puntati sarebbero state effettive, ma non le modifiche
all'indirizzo
a cui punta (Discorso di ciò che PUSH e POP lo stack)
*/
void Elimina_Stringa_din(char *Testo, char *Chiave, char **Testo_Nuovo_din)
{
    int Num_byte; /* Quanti elementi spostare? */
    char *Dest, *Src; /* Dest=Fino a che indirizzo sposto?
                        Src=A partire da quale indirizzo devo spostare?
                        Sono parametri che devo dare a MEMMCOPY*/
    int N_Chiave = strlen(Chiave);
    int N_Testo = strlen(Testo);

    /* RISERVO UNO SPAZIO PER IL TESTO INIZIALE SU CUI CONTROLLERÒ E RIALLOCHERÒ EVENTUALMENTE */
    *Testo_Nuovo_din = (char *)malloc(N_Testo+1); // +1 per il '\0'
    strcpy(*Testo_Nuovo_din, Testo); // Non voglio sporcare testo

    do
    {
        Dest = strstr(*Testo_Nuovo_din, Chiave); // Indirizzo della occorrenza
        if(Dest) // se c'è
        {
            Src = Dest+N_Chiave; // Da quale elemento spostare (Pattern+len_pattern...)
            Num_byte = N_Testo-(int)(Src-*Testo_Nuovo_din)+1; /* Quanti byte spostare?
                                                               N_Testo-(Indice di src).
                                                               +1 è per il '\0' */
            memmove(Dest, Src, Num_byte); // Se lo vede memmove con i dati forniti
            N_Testo -= N_Chiave; /* Con memmove la cancellazione avviene sovrascrivendo
                                   i caratteri relativi al pattern;
                                   La lunghezza totale diminuirà di N_Chiave */
        }
        /* Cicla fin quando non trova più nessun pattern o è l'ultimo */
        while(Dest!=NULL && Dest<(*Testo_Nuovo_din+N_Testo)-N_Chiave); // i < n-m
    }
    // Ho eliminato un pattern e ora rialloco lo stesso ind. per avere uno spazio minore
    *Testo_Nuovo_din = (char *)realloc(*Testo_Nuovo_din, N_Testo*(sizeof(char))+1); // +1 per lo '\0'
}

```

### OUTPUT:

Testo Iniziale = anTrovaaanLaanParolaan  
 Pattern de eliminare = an

Testo senza pattern (STATICO) = TrovaLaParola  
 Testo senza pattern (DINAMICO) = TrovaLaParola

Testo Iniziale = 321 123C123I123A1230 321  
 Pattern de eliminare = 123

Testo senza pattern (STATICO) = 321 CIAO 321  
 Testo senza pattern (DINAMICO) = 321 CIAO 321

Testo Iniziale = Non mi elimini...  
 Pattern de eliminare = elimina

Testo senza pattern (STATICO) = Non mi elimini...  
 Testo senza pattern (DINAMICO) = Non mi elimini...

## ESERCIZIO 28 [LIV.2]

```

/* [liv2] Scrivere una function C che sostituisca in un testo tutte le occorrenze di una data
sottostringa S1
con un'altra S2 (le duesottostringhe possono avere anche lunghezze diverse). */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* prototipi*/
int *StrStr_Pos(char Testo[], char Chiave[], int n, int m, int *n_pos);
void sostituisci( char **testo, char *S1, char *S2, short *J_occ, short num_occ);

int main()
{
    char *testo, *S1, *S2;
    unsigned int len_t, len_S1, len_S2;
    short *J_occ, num_occ=0; /* j_occ punterà alle posizioni di inizio del pattern */

    puts("---- SOSTITUISCI UNA SOTTOSTRINGA IN UNA STRINGA ----");
    /* Definiamo le lunghezze dei vari parametri*/
    printf("Lunghezza del testo: ");
    scanf("%d",&len_t);
    printf("Lunghezza della stringa S1: ");
    scanf("%d",&len_S1);
    printf("Lunghezza della stringa S2: ");
    scanf("%d",&len_S2);
    /* Alloca le varie stringhe */
    testo = (char *)malloc(++len_t);
    if (testo == NULL)
    {fprintf(stderr, "\nAllocazione del testo non riuscita!\n");exit(EXIT_FAILURE);}
    S1 = (char *)malloc(++len_S1);
    if (S1 == NULL){fprintf(stderr, "\nAllocazione di S1 non riuscita!\n");exit(EXIT_FAILURE);}
    S2 = (char *)malloc(++len_S2);
    if (S2 == NULL){fprintf(stderr, "\nAllocazione di S2 non riuscita!\n");exit(EXIT_FAILURE);}

    /* Inserisci Testo, pattern, stringa da trovare...*/
    printf("\nDigitare il testo: ");fflush(stdin);
    gets(testo);
    printf("\nTesto: %s \n",testo);
    printf("\nInserisci la sottostringa da sostituire: ");fflush(stdin);
    gets(S1);
    printf("\nInserisci sottostringa con la quale sostituire: ");fflush(stdin);
    gets(S2);

    /* Restituisce array occorrenze con relative posizioni */
    J_occ = StrStr_Pos(testo, S1, len_t, len_S1, &num_occ);

    if (num_occ>0) //Sostituisco se ho almeno un'occorrenza
    {
        /* Ora puoi sostituire*/
        sostituisci(&testo, S1, S2, J_occ, num_occ);
        free(S1); free(S2); //dealloca memoria
        printf("\nNUOVO TESTO: %s \n", testo);
    }
    else printf("\ntesto non modificato!\n\nTESTO: %s \n", testo);
    free(testo);

    return 0;
}

/*
   function che cerca le occorrenze del pattern nel testo restituisce array occorrenze
*/
short*StrStr_Pos(char Testo[], char Chiave[], int n, int m, short*n_pos)
{
    short*ch; //contiene i risultati di strstr()
    shortPos_trovata, *array_n;
    shorti=0; //Indice controllo. Se trova un'occorrenza, salta di m
    /* _____ TROVA PRIMA OCCORRENZA _____ */
    /* In questa prima parte, cerco solo per la prima volta, il primo pattern.
       Se c'è, alloco con malloc ad un indirizzo dove successivamente(se esisteranno
       altri pattern) riallocherà'(REALLOC) sullo stesso indirizzo, tenendo conto
       dei numero delle posizioni. In tal modo avrò un vettore di n_pos
       riducendo al minimo lo spazio da occupare. Se la prima volta non trova
       nulla, restituisce un indirizzo a NULL, per segnalare il mancato matching
       di almeno una sottostringa*/
}

```

```

ch = strstr(&(Testo[i]), Chiave); /* Cerca il pattern. Passo indirizzo base di Testo*/
if(ch) /* Ho trovato qualcosa? */
{
    *n_pos+=1; /* Ho almeno una posizione. Sarebbe n_pos = 1*/
    array_n=(short*)malloc(*n_pos*sizeof(short)); /* Richiedi indirizzo per allocare 1 byte
                                                PS dovremo effettuare dei controlli,
                                                ma il codice diventerebbe pesante per
                                                il problema interessato */
    if(array_n==NULL) {printf("ERRORE DI ALLOCAZIONE");exit(EXIT_FAILURE);}

    Pos_trovata = (short)ch-(short)Testo; //Ricavo l'indice trovato tramite gli indirizzi
    array_n[0]= Pos_trovata; // *array_n = indice posizione trovato.

    i = Pos_trovata+m; /* passa alle prossime sotto stringhe dopo il primo pattern trovato
                           saltando il pattern gia' calcolato */
}
else return NULL; /* Ritorna un indirizzo a NULL (non trovo niente->non alloco niente!) */

/* _____ Trova altri matching riallocando lo stesso array _____ */
do
{
    ch = strstr(&(Testo[i]), Chiave); /* Cerca il pattern. Se lo trova, ricicla, e passa a
                                         m posizioni dopo */

    if(ch)
    {
        *n_pos+=1; // (*n_pos)++;Avanzo perche' ho trovato un pattern
        array_n=(short*)realloc(array_n, (*n_pos)*sizeof(short)); /* Rialloco per un ulteriore
                                                                     valore della posizione */
        if(array_n==NULL) {printf("ERRORE DI ALLOCAZIONE");exit(EXIT_FAILURE);}

        Pos_trovata = (short)ch-(short)Testo; //Ricavo l'indice trovato
        array_n[*n_pos-1]= Pos_trovata; /* Inserisco la pos_trovata a n_pos-1
                                         perche' la prima posizione dell'array
                                         in C e' 0 */
        i = Pos_trovata+m; /* dovrà ripartire dall'indirizzo da dove ho trovato il pattern,
                           piu' m per saltare la sottostringa appena trovata e passare
                           ai successivi*/
    }
while(i<n-m+1&&ch!=NULL); /* Cicla fin quando strstr fornisce una ricerca, ossia un indirizzo
                                oppure i si trova prima di n-m+1 */

return array_n; //Ritorno l'indirizzo base su cui ho lavorato dinamicamente
}

/*
Sostituire in testo S1 con S2.
Se S1 = S2, sostituisco direttamente. Le posizioni in cui si trovano le occorrenze
e' dato da *J_occ.
Se S1>S2 significa che dovrò accorciare il testo e cercare di portare S1=S2 mediante
degli spostamenti con memmove. Quando S1=S2 sostituisco.
Se S1<S2 significa che dovrò allungare il testo e cercare di portare S1=S2 mediante
degli spostamenti con memmove. Quando S1=S2 sostituisco.

*/
void sostituisci( char **testo, char *S1, char *S2, short *J_occ, short num_occ)
{
/*
    S1 = Stringa da sostituire
    S2 = Stringa con cui sostituire
*/
char *src, *dest; int num_byte;
unsigned short i, j;
int len_t = strlen(*testo);
int len_S1 = strlen(S1);
int len_S2 = strlen(S2);
int d = len_S1 - len_S2;
int len_new;

/* S1 > S2 */
if (d == 0)
{
    /* Sovrascrivi S2 su S1 individuata da indice j_occ[i] */
    for (i = 0; i < num_occ; i++)
        memcpy( *testo + J_occ[i], S2, strlen(S2));
}

```

```

/* S1 > S2
   La stringa da sostituire e' piu' grande di quella con cui sostituire, quindi il testo
   si accorcerà */
else if (d > 0)
{
    short shift = 0; //Indica di quanto già ho accorciato il testo
    for (i = 0; i < num_occ; i++)
    {

        /* accorcia il testo un carattere alla volta per avere i 2 pattern della stessa lunghezza */
        for (j = 0; j < d; j++)
        {
            src = *testo + J_occ[i] + shift + len_S1 - 1 - j;
            dest = src - 1;
            num_byte = *testo + len_t - src + 1;
            memmove(dest, src, num_byte);
        }
        /* i 2 pattern hanno la stessa lunghezza il nuovo sostuisce il vecchio */
        memcpy(*testo + J_occ[i] + shift, S2, len_S2);
        shift -= d;
    }
    *testo = realloc(*testo, len_t + shift); //riallocazione del testo
    if (testo == NULL)
        {fprintf(stderr, "\nRiallocazione del testo impossibile\n"); exit(EXIT_FAILURE);}
}
/* S1 < S2
   La stringa da sostituire e' piu' piccola di quella con cui sostituire, quindi il testo
   si allungaerà */
else
{
    d = -d;
    len_new = len_t + 1 + num_occ * d; //calcola la nuova lunghezza del testo
    *testo = realloc(*testo, len_new); //il testo viene allungato
    if (testo)
    {
        /* si procede dalla fine all'inizio: porto le stringhe da sostituire alla stessa
           lunghezza del pattern con cui sostituire (memmove) e sostituisco (memcpy) */
        for (i = num_occ; i > 0; i--)
        {
            src = *testo + J_occ[i-1] + len_S1;
            dest = src + d;
            num_byte = strlen(src) + 1;
            memmove(dest, src, num_byte); //spostamento
            memcpy(*testo + J_occ[i-1], S2, len_S2); //sostituzione
        }
    }
    else{printf(stderr, "\nRiallocazione del testo impossibile\n"); exit(EXIT_FAILURE);}
}
}

```

### OUTPUT:

S1=S2  
 ---- SOSTITUISCI UNA SOTTOSTRINGA IN UNA STRINGA ----\*  
 Lunghezza del testo: 6  
 Lunghezza della stringa S1: 2  
 Lunghezza della stringa S2: 2  
 Digitare il testo: AN!!AN  
 Testo: AN!!AN  
 Inserisci la sottostringa da sostituire: AN  
 Inserisci sottostringa con la quale sostituire: xx  
 NUOVO TESTO: xx!!xx

### S1<S2

---- SOSTITUISCI UNA SOTTOSTRINGA IN UNA STRINGA ----\*  
 Lunghezza del testo: 8  
 Lunghezza della stringa S1: 2  
 Lunghezza della stringa S2: 4  
 Digitare il testo: AN!!AN!!  
 Testo: AN!!AN!!  
 Inserisci la sottostringa da sostituire: AN  
 Inserisci sottostringa con la quale sostituire: \$\$\$  
 NUOVO TESTO: \$\$\$!!\$\$\$\$!!

S1>S2  
 ---- SOSTITUISCI UNA SOTTOSTRINGA IN UNA STRINGA ----\*  
 Lunghezza del testo: 8  
 Lunghezza della stringa S1: 3  
 Lunghezza della stringa S2: 2  
 Digitare il testo: xNANxNAN  
 Testo: xNANxNAN  
 Inserisci la sottostringa da sostituire: NAN  
 Inserisci sottostringa con la quale sostituire: ^  
 NUOVO TESTO: x^x^

### Pattern non trovato

---- SOSTITUISCI UNA SOTTOSTRINGA IN UNA STRINGA ----\*  
 Lunghezza del testo: 5  
 Lunghezza della stringa S1: 1  
 Lunghezza della stringa S2: 3  
 Digitare il testo: ABCDE  
 Testo: ABCDE  
 Inserisci la sottostringa da sostituire: Z  
 Inserisci sottostringa con la quale sostituire: HFG  
 testo non modificato!  
 TESTO: ABCDE

## ESERCIZIO 29 [LIV.1]

\*\*  
 [liv.1] A partire dalla matrice A( $m \times n$ ), del tipo sotto indicato, allocata staticamente e dinamicamente per righe, visualizzarne gli elementi per colonne.

$$A_{4x6} = \begin{bmatrix} 11 & 12 & 13 & 14 & 15 & 16 \\ 21 & 22 & 23 & 24 & 25 & 26 \\ 31 & 32 & 33 & 34 & 35 & 36 \\ 41 & 42 & 43 & 44 & 45 & 46 \end{bmatrix}$$

Gli elementi  $a_{ij}$  della matrice sono tali che le unità indicano la colonna e le decine indicano la riga cui l'elemento appartiene.

```
*/
#include <stdio.h>
#include <stdlib.h>
#define M 4
#define N 6

int main()
{
    /* Alloca staticamente la matrice */
    int A[M][N]={{11,12,13,14,15,16},
                  {21,22,23,24,25,26},
                  {31,32,33,34,35,36},
                  {41,42,43,44,45,46}};
    int *Pa, i, j;
    /* Allocò dinamicamente la matrice */
    Pa = (int *)malloc(N*M*sizeof(int));
    if(Pa==NULL) {printf("Errore di allocazione!"); return 0;};
    /* Copia la matrice A in matrice Pa */
    for(i=0;i<M;i++)
        for(j=0; j<N; j++)
            *(Pa+N*i+j)=A[i][j];
    /* VISUALIZZO PER COLONNE SIA LA MATRICE STATICÀ CHE DINAMICA */
    printf("\nAllocazione statica della matrice A:\n"); //Allocazione statica
    for(i=0;i<N;i++)
    {
        for(j=0;j<M;j++) /*Gli indici sono stati spostati per visualizzare per colonne*/
            printf("%d\t",A[j][i]); //Inverto solo indici perche' la riga deve andare
                                      //più veloce della colonna
        printf("\n");
    }
    printf("\nAllocazione dinamica della matrice A:\n"); //Allocazione dinamica
    for(i=0; i<N; i++)
    {
        for(j=0;j<M;j++) /*Gli indici sono stati spostati per visualizzare per colonne*/
            printf("%d\t",*(Pa+j*N+i)); //Inverto solo indici perche' la riga deve andare
                                      //più veloce della colonna
        printf("\n");
    }
    return 0;
}
```

### OUTPUT:

Allocazione statica della matrice A:

|    |    |    |    |
|----|----|----|----|
| 11 | 21 | 31 | 41 |
| 12 | 22 | 32 | 42 |
| 13 | 23 | 33 | 43 |
| 14 | 24 | 34 | 44 |
| 15 | 25 | 35 | 45 |
| 16 | 26 | 36 | 46 |

Allocazione dinamica della matrice A:

|    |    |    |    |
|----|----|----|----|
| 11 | 21 | 31 | 41 |
| 12 | 22 | 32 | 42 |
| 13 | 23 | 33 | 43 |
| 14 | 24 | 34 | 44 |
| 15 | 25 | 35 | 45 |
| 16 | 26 | 36 | 46 |

## ESERCIZIO 30 [LIV.1]

```
/*
[liv.1] Scrivere una function C che restituisca la matrice C prodotto righeXcolonne
di due matrici rettangolari A e B le cui dimensioni sono stabilite in input
(usare per tutte le matrici l'allocazione dinamica e generarle come numeri reali random).
C'è qualche preferenza nell'usare malloc() o calloc() rispettivamente per A, B o C?
Verificare se i tempi di esecuzione, per la sola allocazione e totali, sono gli stessi.
**/


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

int *Righe_X_Colonne(int *A, int *B, int n, int m, int p); //restituisce un puntatore ad int
void Stampa_Mat(int *X, int m, int n);
int main()
{
    int m,n, p, *A, *B, *C, i , j;
    LARGE_INTEGER ticksPerSecond, TICKS1,TICKS2;
    double Time_Malloc, Time_Calloc;

    QueryPerformanceFrequency(&ticksPerSecond); // processor clock frequency
    /* _____ INPUT _____ */
    //N.B. numero colonne A = numero righe B per poter fare il prodotto rxC tra matrici
    printf("Inserire numero righe di A: "); scanf("%d",&m);
    printf("Inserire numero colonne di A, ossia numero righe di B): "); scanf("%d",&p);
    printf("Inserire numero colonne di B: "); scanf("%d",&n);

    /* _____ ALLOCCHIAMO LE MATRICI _____ */
    /** ALLOCAZIONE CON CALLOC **/
    QueryPerformanceCounter(&TICKS1);
    A = (int *)calloc(m*p, sizeof(int));if(A==NULL) {printf("Errore di
                                                allocazione!");exit(1);}
    B = (int *)calloc(p*n, sizeof(int));if(B==NULL) {printf("Errore di
                                                allocazione!");exit(1);}
    QueryPerformanceCounter(&TICKS2);
    Time_Calloc =(double) (TICKS2.QuadPart-TICKS1.QuadPart) / (double)ticksPerSecond.QuadPart;
    free(A);free(B); //Dealloc, perche' dopo realloco

    /** ALLOCAZIONE CON MALLOC **/
    QueryPerformanceCounter(&TICKS1);
    A = (int *)malloc(m*p*sizeof(int));if(A==NULL) {printf("Errore di
                                                allocazione!");exit(1);}
    B = (int *)malloc(p*n*sizeof(int));if(B==NULL) {printf("Errore di
                                                allocazione!");exit(1);}

    QueryPerformanceCounter(&TICKS2);
    Time_Malloc=(double) (TICKS2.QuadPart TICKS1.QuadPart) / (double)ticksPerSecond.QuadPart;

    /*____ ASSEGNA MO DEI VALORI RANDOM ALLE MATRICI CREATE ____*/
    srand(time(NULL)); //Inizializzo seed
    //Matrice A
    for (i=0; i<m; i++)
        for (j=0; j<p; j++)
            *(A+i*p+j) = (float) (rand()%6); //memorizzazione per righe
    //Matrice B
    for (i=0; i<p; i++)
        for (j=0; j<n; j++)
            *(B+i*p+j) = (float) (rand()%6); //memorizzazione per righe

    //Effettua l'operazione
    C=Righe_X_Colonne(A,B, n, m, p);

    /* Stampe delle matrici */
    printf("\n***** MATRICE A *****\n");
    Stampa_Mat(A, m, p);
    printf("\n***** MATRICE B *****\n");
    Stampa_Mat(B, p, n);
    printf("\n\n***** MATRICE C = A x B *****\n");
    Stampa_Mat(C, m, n);
    free(A);free(B);free(C);
    printf("\n\n Time_Malloc: %g", Time_Malloc);
    printf("\n\n Time_Calloc: %g", Time_Calloc);

    return 0;
}
```

```

/* C [MxN] = A [MxP] X B [PxN] */
int *Righe_X_Colonne(int *A, int *B, int n, int m, int p)
{
    int *C, i, j, k;
    //Essendo una sommatoria, devo inizializzare tutto a 0
    C = (int *)calloc(m*n, sizeof(int)); if(C==NULL) {printf("Errore di allocazione!"); exit(1);}

    for (i=0; i<m; i++)           //Scorre le righe di A
        for (j=0; j<n; j++)       //Scorre le colonne B
            for (k=0; k<p; k++)   //Scorre per p volte la colonna A/raga B ed per la sommatoria
            {
                /* A i,j-simo della matrice C assegna quello che c'era piu' il prodotto
                   dell'elemento K-esimo dell'i-esima riga di 'A' per l'elemento K-esimo della
                   j-esima colonna di 'B'
                   C[i][j]= A[i][k] * B[k][j]*/
                C[i][j] = A[i][k] * B[k][j];
                *(C+i*n+j) = *(C+i*n+j) + (*((A+i*p+k)) * (*(B+k*n+j)));
            }
        return C;
}
void Stampa_Mat(int *X, int m, int n)
{
    int i, j;
    for (i=0; i<m; i++)
    {
        for (j=0; j<n; j++) printf("%d\t", *(X+i*n+j));
        printf("\n");
    }
}

```

### OUTPUT:

C'è qualche preferenza nell'usare malloc() o calloc() rispettivamente per A, B o C?

Certo, per A e B, non mi interessa che inizialmente ci siano dei valori sporchi, tanto verranno inseriti valori random. Per C, dato che è una sommatoria, inizialmente ogni valore della matrice viene posto uguale a 0 e ciò ha un suo tempo. Quindi la **Calloc** è più lenta della **Malloc** (siamo certi di dirlo per valori grandi).

```

Inserire numero righe di A: 2
Inserire numero colonne di A, ossia numero righe di B): 2
Inserire numero colonne di B: 4

```

```
***** MATRICE A *****
3      0
0      5
```

```
***** MATRICE B *****
1      3      2      4
1      2      4      1
```

```
***** MATRICE C = A x B *****
3      9      6      12
5      10     20     5
```

Time\_Malloc: 8.19194e-007

Time\_Calloc: 3.27677e-006

```

Inserire numero righe di A: 6
Inserire numero colonne di A, ossia numero righe di B): 2
Inserire numero colonne di B: 5

```

```
***** MATRICE A *****
1      5
4      3
0      5
1      0
0      1
2      2
```

```
***** MATRICE B *****
5      0      4      5      3
0      5      4      2      0
```

```
***** MATRICE C = A x B *****
5      25     24     15     3
20     15     28     26     12
0      25     20     10     0
5      0      4      5      3
0      5      4      2      0
10     10     16     14     6
```

Time\_Malloc: 8.19194e-007

Time\_Calloc: 3.27677e-006

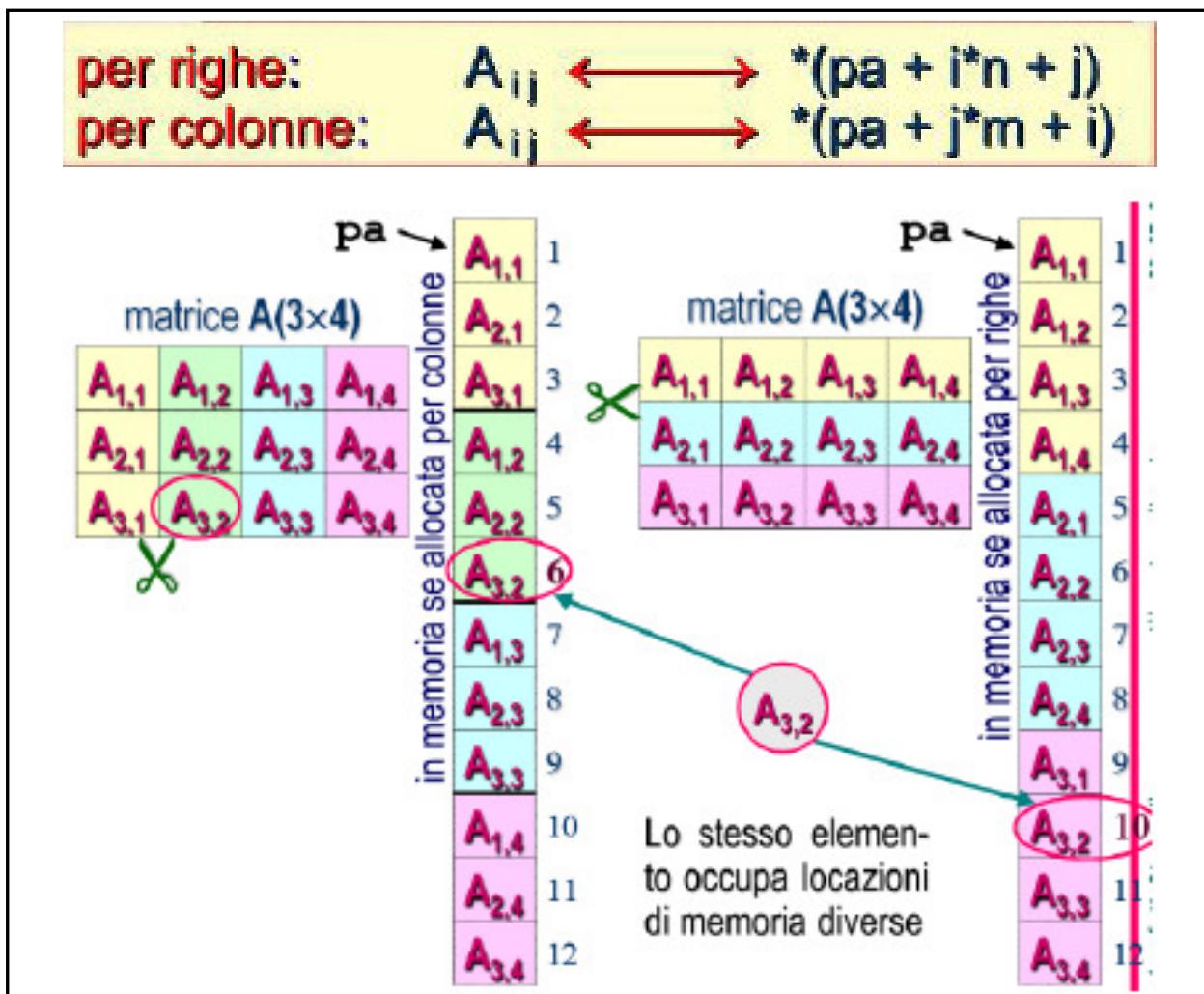
## MA COSA SIGNIFICA ALLOCARE PER RIGHE O COLONNE?

Allocare per righe significa organizzare la memoria per righe, quindi avere per ogni riga, tutte le colonne per n volte di quella riga in modo consecutivo.

La formula da utilizzare e'  $*(\text{pa} + i * \text{n} + j)$ . (Matrici (MxN)

Allocare per colonne significa organizzare la memoria per colonne, quindi avere per ogni colonna, tutte le righe per m volte di quella colonna in modo consecutivo.

La formula da utilizzare e'  $*(\text{pa} + j * \text{m} + i)$ .

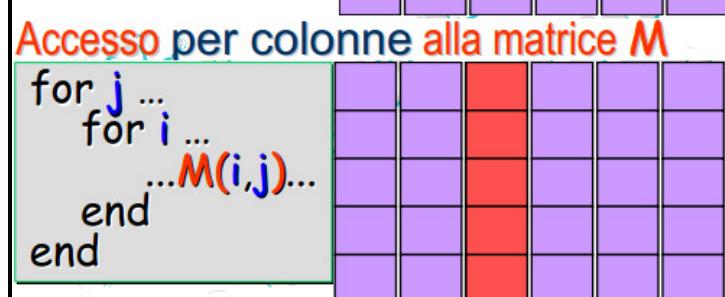
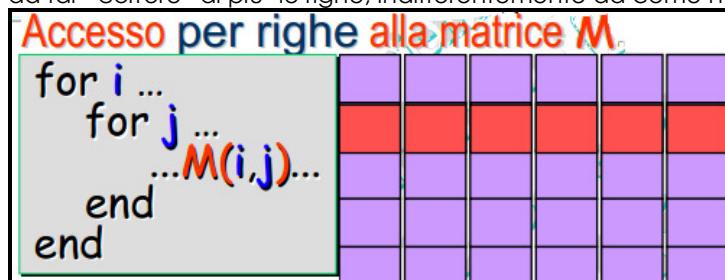


## MA COSA SIGNIFICA ACCEDERE PER RIGHE O COLONNE?

la differenza sta a come si accede nella memoria e come calibrare i "salti", ossia nell'accesso per righe l'indice j delle colonne va più velocemente dell'indice i delle righe, invece nell'accesso per colonne l'indice i delle righe va più velocemente dell'indice j.

**ricorda:** L'accesso è indipendente da come è allocata la matrice in memoria!

Conclusione: se alloco per righe o colonne e voglio accedere per colonne, devo invertire i loro indici o pongo i cicli in modo da far "correre" di più le righe, indifferentemente da come ho allocato.



### ESERCIZIO 31 [LIV.3]

/\*\* [liv.3] Ripetere l'esercizio precedente sul prodotto righeXcolonne di matrici, una prima volta, allocando tutte le matrici in memoria per colonne ed, una seconda volta, per righe. Per ciascun tipo di allocazione in memoria, scrivere due function C per il prodotto righeXcolonne:  
una che acceda a tutte le matrici per colonne e l'altra per righe. Confrontare i tempi dell'esecuzione delle due modalita' di accesso alle matrici rispetto alla loro allocazione in memoria, deducendo quindi il tipo di accesso piu' efficiente rispetto al criterio di memorizzazione.

\*\*/

PER LA TEORIA, VEDERE LA PAGINA PRECEDENTE

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <time.h>
/* Prototipi */
//Restituiscono l'indirizzo che punta alla matrice C
int *rxc_Alloc_Riga_Access_Riga(int *A, int n, int m, int p);
int *rxc_Alloc_Colonna_Access_Colonna(int *A, int *B, int n, int m, int p);
int *rxc_Alloc_Riga_Access_Colonna(int *A, int *B, int n, int m, int p);
int *rxc_Alloc_Colonna_Access_Riga(int *A, int *B, int n, int m, int p);

void Stampa_Mat_rig(int *X, int m, int n);
void Stampa_Mat_col(int *X, int m, int n);

int main()
{
    int m, n, p, *A, *B, *C, i, j;
    LARGE_INTEGER ticksPerSecond, TICKS1, TICKS2;
    double Time_r_r, Time_r_c, Time_c_r, Time_c_c;
    QueryPerformanceFrequency(&ticksPerSecond); // processor clock frequency
    srand(time(NULL)); //Inizializzo seed
    /*           INPUT           */
    //N.B. numero colonne A = numero righe B per poter fare il prodotto rxC tra matrici
    printf("Inserire numero righe di A: "); scanf("%d", &m);
    printf("Inserire numero colonne di A, ossia il numero righe di B: "); scanf("%d", &p);
    printf("Inserire numero colonne di B: "); scanf("%d", &n);

    /* _____ ALLOCZO DINAMICAMENTE IN MEMORIA _____ */
    A = (int *)malloc(m*p*sizeof(int));
    if(A==NULL) {printf("Errore di allocazione!"); exit(1);}
    B = (int *)malloc(p*n*sizeof(int));
    if(B==NULL) {printf("Errore di allocazione!"); exit(1);}

    /** NB Allocazione per colonne o per righe, indica semplicemente il modo in cui sono memorizzati in memoria.**/

    /*----- ALLOCAZIONE(MEMORIZZAZIONE) PER RIGHE -----*/
    /* _____ Genero valori random, memorizzandoli in memoria per RIGHE _____ */
    //Matrice A(M, P)
    for (i=0; i<m; i++)
        for (j=0; j<p; j++)
            *(A+i*p+j) = (float) (rand()%6); //memorizzazione per RIGHE
    //Matrice B(P, N)
    for (i=0; i<p; i++)
        for (j=0; j<n; j++)
            *(B+i*n+j) = (float) (rand()%6); //memorizzazione per RIGHE

    /* _____ Accedi sia per righe che per colonne alle matrici allocate per RIGHE. _____ */
    //Accedi per righe e calcola tempo
    QueryPerformanceCounter(&TICKS1); //inizio tempo
    C = rxc_Alloc_Riga_Access_Riga(A, B, n, m, p);
    QueryPerformanceCounter(&TICKS2); //fine tempo
    Time_r_r = (double) (TICKS2.QuadPart-TICKS1.QuadPart) / (double) ticksPerSecond.QuadPart;
    printf("\n_____ ALLOCAZIONE PER RIGHE E ACCESSO PER RIGHE _____\n");
    printf("\n***** MATRICE A *****\n"); Stampa_Mat_rig(A, m, p);
    printf("\n***** MATRICE B *****\n"); Stampa_Mat_rig(B, p, n);
    printf("\n\n***** MATRICE C = A x B *****\n"); Stampa_Mat_rig(C, m, n);
    //accedi per colonne e calcola tempo
    QueryPerformanceCounter(&TICKS1); //inizio tempo
    C = rxc_Alloc_Riga_Access_Colonna(A, B, n, m, p);
```

```

QueryPerformanceCounter(&TICKS2); //fine tempo
Time_r_c = (double)(TICKS2.QuadPart-TICKS1.QuadPart)/(double)ticksPerSecond.QuadPart;
printf("\n____ ALLOCAZIONE PER RIGHE E ACCESSO PER COLONNE ____\n");
//anche se e' allocata per righe, stampiamo per colonne per vedere che operazione fa
printf("\n***** MATRICE A *****\n");Stampa_Mat_rig(A, m, p);
printf("\n***** MATRICE B *****\n");Stampa_Mat_rig(B, p, n);
printf("\n\n***** MATRICE C = A x B *****\n");Stampa_Mat_rig(C, m, n);

/*----- ALLOCAZIONE (MEMORIZZAZIONE) PER COLONNE -----*/
/* _____ Genero valori random, memorizzandoli in memoria per COLONNE _____ */
//Matrice A(M, P)
for (i=0; i<m; i++)
    for (j=0; j<p; j++)
        *(A+j*m+i) = (float)(rand()%6); //memorizzazione per COLONNE
//Matrice B(P, N)
for (i=0; i<p; i++)
    for (j=0; j<n; j++)
        *(B+j*p+i) = (float)(rand()%6); //memorizzazione per COLONNE
/* _____ Accedi sia per righe che per colonne alle matrici allocate per COLONNE. _____ */
//Accedi per righe
QueryPerformanceCounter(&TICKS1); //inizio tempo
C = rxc_Alloc_Colonna_Access_Riga(A,B,n,m,p);
QueryPerformanceCounter(&TICKS2); //fine tempo
Time_c_r = (double)(TICKS2.QuadPart-TICKS1.QuadPart)/(double)ticksPerSecond.QuadPart;
printf("\n____ ALLOCAZIONE PER COLONNE E ACCESSO PER RIGHE ____\n");
printf("\n***** MATRICE A *****\n");Stampa_Mat_col(A, m, p);
printf("\n***** MATRICE B *****\n");Stampa_Mat_col(B, p, n);
printf("\n\n***** MATRICE C = A x B *****\n");Stampa_Mat_col(C, m, n);

//Accedi per colonne
QueryPerformanceCounter(&TICKS1); //inizio tempo
C = rxc_Alloc_Colonna_Access_Colonna(A,B,n,m,p);
QueryPerformanceCounter(&TICKS2); //fine tempo
Time_c_c = (double)(TICKS2.QuadPart-TICKS1.QuadPart)/(double)ticksPerSecond.QuadPart;
printf("\n____ ALLOCAZIONE PER COLONNE E ACCESSO PER COLONNE ____\n");
printf("\n***** MATRICE A *****\n");Stampa_Mat_col(A, m, p);
printf("\n***** MATRICE B *****\n");Stampa_Mat_col(B, p, n);
printf("\n\n***** MATRICE C = A x B *****\n");Stampa_Mat_col(C, m, n);

/*-----*/
printf("\n\n");
free(A);free(B);free(C);
printf("Tempo con l'allocazione per righe e l'accesso per righe: %g\n", Time_r_r);
printf("Tempo con l'allocazione per righe e l'accesso per colonne: %g\n", Time_r_c);
printf("Tempo con l'allocazione per colonne e l'accesso per righe: %g\n", Time_c_r);
printf("Tempo con l'allocazione per colonne e l'accesso per colonne: %g\n", Time_c_c);
printf("\n");
return 0;
}

/* Prodotto righe per colonne accedendo per RIGHE e allocando per RIGHE
C[MxN]= A[MxP] X B[PxN] */
int *rxc_Alloc_Riga_Access_Riga(int *A,int *B, int n, int m, int p)
{
    int *C, i, j, k;
    //Essendo una sommatoria, devo inizializzare tutto a 0
    C = (int *)calloc(m*n,sizeof(int));if(C==NULL){printf("Errore di allocazione!");exit(1);}

    for (i=0; i<m; i++)           //Scorre le righe
        for (j=0; j<n; j++)       //Scorre le colonne
            for (k=0; k<p; k++) //Scorre per p volte la colonna A/riga B ed per la sommatoria
            {
                /* A i,j-simo della matrice C assegna quello che c'era piu' il prodotto
                   dell'elemento K-esimo dell'i-esima riga di 'A' per l'elemento K-esimo della
                   1-esima colonna di 'B'
                   C[i][j] = A[i][k] * B[k][j]*/
                C[i][j] = A[i][k] * B[k][j];
                *(C+i*n+j) = *(C+i*n+j) + (*((A+i*p+k)) * (*(B+k*n+j)));
            }
    return C;
}

/* Prodotto righe per colonne accedendo per COLONNE e allocando per RIGHE
C[MxN]= A[MxP] X B[PxN] */
int *rxc_Alloc_Colonna_Access_Riga(int *A,int *B, int n, int m, int p)
{
    int *C, i, j, k;
    //Essendo una sommatoria, devo inizializzare tutto a 0
    C = (int *)calloc(m*n,sizeof(int));if(C==NULL){printf("Errore di allocazione!");exit(1);}

    for (j=0; j<n; j++)           //Scorre le righe
        for (i=0; i<m; i++)       //Scorre le colonne
            for (k=0; k<p; k++) //Scorre per p volte la colonna A/riga B ed per la sommatoria
            {
                /* A i,j-simo della matrice C assegna quello che c'era piu' il prodotto
                   dell'elemento K-esimo dell'i-esima riga di 'A' per l'elemento K-esimo della
                   1-esima colonna di 'B'
                   C[i][j] = A[i][k] * B[k][j]*/
                C[j][i] = A[i][k] * B[k][j];
                *(C+j*m+i) = *(C+j*m+i) + ((A[i]*p+k) * (B+k*n+j));
            }
    return C;
}

```

/\* Prodotto righe per colonne accedendo per COLONNE e allocando per RIGHE

```

C[MxN]= A[MxP] X B[PxN] */
int *rxc_Alloc_Riga_Access_Colonna(int *A, int *B, int n, int m, int p)
{
    int *C, i, j, k;
    //Essendo una sommatoria, devo inizializzare tutto a 0
    C = (int *)calloc(m*n,sizeof(int));if(C==NULL){printf("Errore di allocazione!");exit(1);}

    for (j=0; j<n; j++)           //Scorre le righe
        for (i=0; i<m; i++)       //Scorre le colonne
            for (k=0; k<p; k++)   //Scorre per p volte la colonna A/rima B ed per la sommatoria
            {
                /* A i,j-simo della matrice C assegna quello che c'era piu' il prodotto
                   dell'elemento K-esimo dell'i-esima riga di 'A' per l'elemento K-esimo della
                   l-esima colonna di 'B'
                C[i][j] = A[i][k] * B[k][j]*/
                C[i][j] = A[i][k] * B[k][j];
                *(C+i*n+j) = *(C+i*n+j) + (* (A+i*p+k)) * (* (B+k*n+j));
            }
    return C;
}

/* Prodotto righe per colonne accedendo per RIGHE e allocando per COLONNE
C[MxN]= A[MxP] X B[PxN] */
int *rxc_Alloc_Colonna_Access_Riga(int *A,int *B, int n, int m, int p)
{
    int *C, i, j, k;
    //Essendo una sommatoria, devo inizializzare tutto a 0
    C = (int *)calloc(m*n,sizeof(int));if(C==NULL){printf("Errore di allocazione!");exit(1);}

    for (i=0; i<m; i++)           //Scorre le righe
        for (j=0; j<n; j++)       //Scorre le colonne
            for (k=0; k<p; k++)   //Scorre per p volte la colonna A/rima B ed per la sommatoria
            {
                /* A i,j-simo della matrice C assegna quello che c'era piu' il prodotto
                   dell'elemento K-esimo dell'i-esima riga di 'A' per l'elemento K-esimo della
                   l-esima colonna di 'B'
                C[i][j] = A[i][k] * B[k][j]*/
                C[i][j] = A[i][k] * B[k][j];
                *(C+j*m+i) = *(C+j*m+i) + (* (A+k*m+i)) * (* (B+j*p+k));
            }
    return C;
}

/* Prodotto righe per colonne accedendo per COLONNE e allocando per COLONNE
C[MxN]= A[MxP] X B[PxN] */
int *rxc_Alloc_Colonna_Access_Colonna(int *A,int *B, int n, int m, int p)
{
    int *C, i, j, k;
    //Essendo una sommatoria, devo inizializzare tutto a 0
    C = (int *)calloc(m*n,sizeof(int));if(C==NULL){printf("Errore di allocazione!");exit(1);}

    for (j=0; j<n; j++)           //Scorre le righe
        for (i=0; i<m; i++)       //Scorre le colonne
            for (k=0; k<p; k++)   //Scorre per p volte la colonna A/rima B ed per la sommatoria
            {
                /* A i,j-simo della matrice C assegna quello che c'era piu' il prodotto
                   dell'elemento K-esimo dell'i-esima riga di 'A' per l'elemento K-esimo della
                   l-esima colonna di 'B'
                C[i][j] = A[i][k] * B[k][j]*/
                C[j][i] = A[i][k] * B[k][j];
                *(C+j*m+i) = *(C+j*m+i) + (* (A+k*m+i)) * (* (B+j*p+k));
            }
    return C;
}

/* Le stampo sono in base all'allocazione */
void Stampa_Mat_rig(int *X, int m, int n)
{
    int i,j;
    for (i=0; i<m; i++)
    {
        for (j=0; j<n; j++)
            printf("%d\t",*(X+i*n+j));
        printf("\n");
    }
}

```

```

void Stampa_Mat_col(int *X, int m, int n)
{
    int i,j;
    for (i=0; i<m; i++)
    {
        for (j=0; j<n; j++)
            printf("%d\t", *(X+j*m+i));
        printf("\n");
    }
}

```

### OUTPUT:

Inserire numero righe di A: 2  
Inserire numero colonne di A, ossia il numero righe di B: 3  
Inserire numero colonne di B: 2

#### ALLOCAZIONE PER RIGHE E ACCESSO PER RIGHE

\*\*\*\*\* MATRICE A \*\*\*\*\*  
2 4 0  
5 4 2

\*\*\*\*\* MATRICE B \*\*\*\*\*  
5 0  
1 4  
3 4

\*\*\*\*\* MATRICE C = A x B \*\*\*\*\*  
14 16  
35 24

#### ALLOCAZIONE PER RIGHE E ACCESSO PER COLONNE

\*\*\*\*\* MATRICE A \*\*\*\*\*  
2 4 0  
5 4 2

\*\*\*\*\* MATRICE B \*\*\*\*\*  
5 0  
1 4  
3 4

\*\*\*\*\* MATRICE C = A x B \*\*\*\*\*  
14 16  
35 24

#### ALLOCAZIONE PER COLONNE E ACCESSO PER RIGHE

\*\*\*\*\* MATRICE A \*\*\*\*\*  
3 2 1  
5 1 3

\*\*\*\*\* MATRICE B \*\*\*\*\*  
4 2  
0 2  
0 1

\*\*\*\*\* MATRICE C = A x B \*\*\*\*\*  
12 11  
20 15

#### ALLOCAZIONE PER COLONNE E ACCESSO PER COLONNE

\*\*\*\*\* MATRICE A \*\*\*\*\*  
3 2 1  
5 1 3

\*\*\*\*\* MATRICE B \*\*\*\*\*  
4 2  
0 2  
0 1

\*\*\*\*\* MATRICE C = A x B \*\*\*\*\*  
12 11  
20 15

Tempo con l'allocazione per righe e l'accesso per righe: 8.19194e-007  
Tempo con l'allocazione per righe e l'accesso per colonne: 1.22879e-006  
Tempo con l'allocazione per colonne e l'accesso per righe: 1.63839e-006  
Tempo con l'allocazione per colonne e l'accesso per colonne: 3.27677e-007

**CONCLUSIONE:** dopo vari test, si evince che se una matrice e' allocata per righe o colonne, l'accesso piu' veloce sara' quello uguale al criterio usato per allocare. In questo caso vince l'allocazione per righe e accesso per colonne, ma i reali valori da confrontare si ottengono trattando matrici dell'ordine di 500x500.

## ESERCIZIO 32 [LIV.3]

```
/** [liv.3]Determinare il numero totale di occorrenze di un pattern in un testo, usando per la ricerca l'algoritmo KMP. */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void preprocessing(char *, int *); //Prototipi delle function
int scansione(char *,char *,int *);

int main()
{
    char Patt[100], Testo[100];
    int i, m, *table;
    char scelta;
    printf("\t\t***** ALGORITMO DI RICERCA KMP *****\n");
    fflush(stdin);
    printf("Inserire un testo: ");
    gets(Testo);
    printf("Inserire un pattern: ");
    gets(Patt);
    m=strlen(Patt);
    table=(int*)malloc(m*sizeof(int));
    preprocessing(Patt,table);
    printf("\n\nTabella di preprocessing del pattern\n\n");
    printf("J : ");
    for(i=0;i<m;i++) printf("%d ",i); printf("\nShift : ");
    for(i=0;i<m;i++) printf("%d ",table[i]);
    printf("\n\nNumero di occorrenze del pattern nel testo:%d\n",scansione(Testo, Patt,table));
    printf("");
    return 0;
}
/*
____ PREPROCESSING ____
In questa fase dobbiamo costruire una tabella che indica DI QUANTO MI DEVO SPOSTARE NEL PATTERN quando avviene un insuccesso (mismatch), ma conservando la posizione dei match "buoni". All'interno del pattern, quindi, dobbiamo considerare i prefissi e suffissi uguali di sottostringhe del pattern.
Esempio: se ho il pattern 'ATATEL', la funzione di insuccesso sara':
F[] = 001200, dove F[2] indica che per quella sottostringa del pattern ho un prefisso e un suffisso uguali di lunghezza 1, F[3] e' lo stesso ma lungo di 2...
RICORDA: Prefisso e suffisso NON possono avere una lunghezza totale quanto la sottostringa considerata! */

void preprocessing(char Patt[], int table[])
{
    int i, j;
    int m=strlen(Patt);
    table[0]=0; /* table[0]=0 per definizione (suffisso e prefisso 0) */
    i=1; /* (non puo certo esiste una sottostringa di una stringa di lunghezza 1) */
    j=0;
    while(i<m)
    {
        printf("Confronto tra P[%d] = %c e P[%d] = %c\n",j,Patt[j],i,Patt[i]);

        /* SE un carattere della stringa prefisso in j è uguale a quello della stringa suffisso in i... */
        if(P[j]==P[i])
        {
            table[i]=j+1; /*incrementa entita' dei prefissi e suffissi*/
            i++; /*si passa ai prossimi caratteri della stringa*/
            j++; /*prefisso in j e della stringa suffisso in i*/
        }
        /* SE si verifica un mismatch e non si sta considerando il primo carattere della stringa prefisso in j*/
        else if(j>0)
        {
            j=table[j-1]; /* si fa ripartire il confronto da table[j-1], ossia 'salta' il prefisso = suffisso di P[0..i] trovato alla precedente iterazione*/
        }
        /* SE si verifica un mismatch e si sta considerando il primo ( j=0 )*/
        else
        {
            table[i]=0; /* Se ho un mismatch e indice del prefisso a 0, devo mettere 0 */
            i++; /*carattere della stringa prefisso in j, avanzare nella tabella*/
        }
    }
}
```

```
/*
SCANSIONE VELOCE DEL TESTO
Da qui parte la ricerca vera e propria: Confronta ogni elemento j-esimo del pattern con l'i-esimo
del testo:
Se trovo sempre dei caratteri uguali e conto un'occorrenza, si comporta come il naive.
Se dopo j-1 successi e al j-esimo tentativo ho un insuccesso (mismatch di un carattere)
e quindi j>0, T[i] sara' confrontato con P[Table[j-1]], quindi dando per buoni dei
matching precedenti, possiamo eventualmente risparmiare dei controlli inutili sul
pattern, dato che la tabella ci da la possibilita' di RICORDARE se determinati
caratteri si ripetono. Se ho subito un insuccesso o j=0, si comporta come il naive, cioe' passa
al carattere successivo del testo. */
int scansione(char Testo[], char Patt[], int table[])
{
    int i, j, occ=0;
    int n=strlen(Testo);
    int m=strlen(Patt);
    i=j=0; /* j=indice sul pattern i=indice sul testo */

    /*Fin quando non finisce il testo*/
    while(i<n)
    {
        if(Pat[j]==Testo[i]) /*match positivo*/
        { /* terminato lo scorrimento del pattern, è individuata una nuova occorrenza*/
            if(j==(m-1))
            {
                occ++; //conto occorrenza della stringa
                j=-1; //In realta' dovrebbe ripartire da 0, ma -1 perche' incremento dopo
            }
            i++; j++; /*si passa al confronto successivo*/
        }
        /* Se Si verifica un MISMATCH*/
        else if(j>0) j=table[j-1]; /* se j>0 si aggiorna j per risparmiare posizioni, prendendo
                                      le relative informazioni nella tabella costruita precedent.
                                      In pratica, j si fa dare l'informazione su "QUANTI CARATTERI
                                      SONO BUONI E QUINDI DA QUALE CARATTERE DEL PATTERN RIPARTIRE
                                      EVITANDO INUTILI CALCOLI */
        else if(!j) i++; /*se j=0 si passa al prossimo carattere del testo*/
    }
    return occ;
}
```

**OUTPUT:**

\*\*\*\*\* ALGORITMO DI RICERCA KMP \*\*\*\*\*

Inserire un testo : atalataalaatatalatalatalq  
Inserire un pattern: atatal

Confronto tra P[0] = a e P[1] = t  
Confronto tra P[0] = a e P[2] = a  
Confronto tra P[1] = t e P[3] = t  
Confronto tra P[2] = a e P[4] = a  
Confronto tra P[3] = t e P[5] = l  
Confronto tra P[1] = t e P[5] = l  
Confronto tra P[0] = a e P[5] = l

Tabella di preprocessing del pattern

J : 0 1 2 3 4 5  
Table[j] : 0 0 1 2 3 0

\*\*\*\*\* ALGORITMO DI RICERCA KMP \*\*\*\*\*

Inserire un testo : abbabalabbalababbaba  
Inserire un pattern: abbaba

Tabella di preprocessing del pattern

J : 0 1 2 3 4 5  
Table[j] : 0 0 0 1 2 1

Numero di occorrenze del pattern nel testo:2

❖ Complessità computazionale nel caso peggiore:  $O(MN)$ .

|   |   |   |   |   |   |   |   |   |   |     |
|---|---|---|---|---|---|---|---|---|---|-----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... |
| a | b | a | b | b | a | b | a | a |   |     |
| a | b | a | b | a | c |   |   |   |   |     |
| a | b | a | b | a | c |   |   |   |   |     |
| a | b | a | b | a | c |   |   |   |   |     |
| a | b | a | b | a | c |   |   |   |   |     |
| a | b | a | b | a | c |   |   |   |   |     |
| a | b | a | b | a | c |   |   |   |   |     |

❖ Complessità computazionale nel caso medio:  $O(M+N)$ .

**Vantaggi**

- Indipendenza dall'alfabeto e dalla distribuzione dei suoi caratteri.
- Velocità di esecuzione della ricerca.

**Svantaggi**

- Degrada all'algoritmo naïve nel caso peggiore.

## ESERCIZIO 33 [LIV.1]

```
/*
[liv.1] Scrivere una function C che legga, mediante una variabile "buffer" di 200char,
un file testo e lo visualizzi sullo schermo 40 char per riga e 25 righe per ogni schermata,
fermandosi finchè non viene premuto un tasto per continuare.
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define BUFSIZE 200
#define BUF_RIGA 25
#define BUF_COLONNA 40

void Visualizza_File_Testo(char *Nome_File, char buffer[]);

int main()
{
    char Nome_File[32], buffer[200];
    printf("Inserire il nome di estensione .txt: ");
    gets(Nome_File); strcat(Nome_File, ".txt");
    printf("nomefile: %s\n\n", Nome_File);
    Visualizza_File_Testo(Nome_File, buffer);
    printf("\n\n");
    return 0;
}

/* Come funziona: Nell'array vengono caricate 40 caratteri e vengono visualizzati.
dopo aver caricato i 40 caratteri (o se fin quando il file pointer punta a
qualcosa di diverso a EOF), viene contata una riga. Arrivato a 25 righe,
viene chiesto di premere un tasto per continuare e visualizzare le restanti
righe che verranno caricate successivamente. Se fp punta a EOF alla 25esima
riga, il programma non conta la riga in piu', perche' significhera' che
quella sara' stata l'ultima riga.*/
void Visualizza_File_Testo(char *Nome_File, char buffer[])
{
    FILE *fp; //Puntatore al file
    int i_riga=0; //Conta le righe
    //se si verifica un errore nell'apertura del file
    if ((fp=fopen(Nome_File, "r")) == NULL) {puts("Errore apertura file"); exit(1);}

    //Cicla fin quando il file pointer non punta alla fine
    while(!feof(fp))
    {
        i_riga=0;

        /* STAMPA AL PIU' 25 RIGHE */
        /* Scorre il file fin quando l'indice non è uguale a max righe o non e' finito
        il file */
        while(i_riga<BUF_RIGA && !feof(fp))
        {
            fgets(buffer,BUF_COLONNA,fp); //Leggi i primi 40 char
            printf("%s\n",buffer);
            i_riga++;
        }

        //Se il file pointer punta ancora a qualcosa
        if(!feof(fp))
        {
            printf("\n");
            fflush(stdin);
            printf("* Premi un tasto per visualizzare la prossima schermata *\n\n");
            getch();
        }
        else puts("\nFile finito!");
    }
}
```

**OUTPUT:**

Inserire il nome di estensione .txt: file  
nomefile: file.txt

Sting e' nato a Wallsend, nella periferia a nord di Newcastle-upon-Tyne in Inghilterra il 2 ottobre 1951, da Audrey Cowell, una parrucchiera, ed Eric, un ingegnere civile. Fin da bambino aspirava alla carriera di musicista. Frequento' la St. Cuthbert Roman Catholic School a Newcastle (la stessa scuola frequentata da Neil Tennant), e successivamente l'università di Warwick a Coventry, ma non si laureò. Dal 1971 al 1974 frequentò un corso di preparazione per insegnanti. Prima di suonare musica professionalmente, Sumner lavorò come scavatore e insegnante di inglese e, per un solo anno, anche di disegno. Tenne i suoi primi concerti ovunque potesse ottenere un lavoro. Suonò con gruppi jazz, in vari locali dell'epoca, come i Phoenix Jazzmen e Last Exit. Si pensa che abbia ricevuto il suo soprannome mentre suonava con i Jazzmen, perché portava spesso una maglietta a strisce nere e gialle; uno dei componenti della band, Gordon Solloway, notò che sembrava un'ape, e così di

\* Premi un tasto per visualizzare la prossima schermata \*

venne Stinger (colui che punge), abbreviato poi in Sting (pungiglione). Si fa abitualmente chiamare Sting da tutti, compresi i figli: compare come Gordon Matthew Sumner solo sui documenti ufficiali. Sting sposò l'attrice irlandese Frances Tomelty nel 1976. La coppia ebbe due figli prima del loro divorzio nel 1982. Poco dopo, Sting cominciò a vivere con l'attrice (e più tardi produttrice) Trudie Styler ma non si sposò fino al 20 agosto 1992. Sting e Trudie hanno avuto quattro figli. Anche se Sting possiede molti terreni nel Regno Unito e negli Stati Uniti, attualmente si considera a casa a Figline Valdarno in Toscana, dove ha comprato la villa del "Palagio". Nel 2002 acquistò i beni confinanti della ex fattoria Serristori, ovvero 200 ettari tra vigne, uliveti, pascoli e boschi, diventando così proprietario di un'intera collina di oltre 300 ettari tra il Chianti e il Valdarno. L'azienda agricola 'Tenuta Il Palagio' dà lavoro a 15 dipendenti fissi (il più giovane ha

\* Premi un tasto per visualizzare la prossima schermata \*

posti in classifiche di vario rango, nel corso degli anni, lo hanno visto al 77º posto tra i 100 personaggi più influenti al mondo (2011 - rivista americana TIME), al 9º posto tra i 10 personaggi musicali inglesi più ricchi del Regno Unito (2011 - Sunday Times) e al 63º posto tra i 100 più grandi artisti di sempre del Rock'n'Roll (1998 - emittente televisiva VH1).

Sting è un appassionato di calcio e tifoso del Newcastle, mentre ha sviluppato negli ultimi tempi una simpatia verso la Fiorentina.

File finito!

## ESERCIZIO 34 [LIV.2]

```
/** [liv.2] Scrivere un programma C che crei un file binario studenti1.dat contenente le seguenti informazioni:
o cognome e nome (30c) c = char
o matricola (ccc/ccccc)
o numero degli esami superati (short)
o media pesata degli esami (float)
o crediti acquisiti (short).

Il file contiene le informazioni già ordinate per matricola. Scrivere una function C che, a partire da un file di aggiornamento relativo ad un certo esame (per esempio, "esameProg2.dat") contenente gli studenti che l'hanno superato ed i relativi voti, crei il file "studenti2.dat" aggiornato.**/

#include <stdio.h>
#include <stdlib.h>
#define FILE_ARCHIVIO "studenti1.dat"
#define FILE_AGGIORNAMENTO "esameProg2.dat"
#define FILE_ARCHIVIO_PROMOSSI "studenti2.dat"
#define CREDITI_PROG2 9

/* Tale struct contiene gli studenti del corso */
typedef struct Studenti //definisce la struct che contiene i dati generici
{
    char Nome_Cognome[30];
    char Matricola[9];
    short N_esami;
    float Media;
    short Crediti;
    //short voto_esame;
} STUDENTI;
/* Tale struct contiene solo gli studenti che dovranno essere aggiornati: solo i promossi */
typedef struct Aggiornamento //definisce la struct che contiene i dati generici
{
    char Matricola[9];
    short Voto_Nuovo;
    //short voto_esame;
} AGGIORNAMENTO;

void Inserisci_Studente();
void Ordina_Mat();
void Visualizza_Archivio(char *Nome_File);
void Crea_File_Di_Aggiorramento();
void Aggiorna_Archivio();
int Appartiene(char *Key, STUDENTI *Studente, int n_stud);

int main()
{
    short Scelta;

    do
    {
        printf("***** MENU *****\n");
        printf("[1] Inserire nuovo studente");
        printf("\n[2] Aggiornare dati dei studenti promossi");
        printf("\n[3] Visualizzare la lista degli studenti del corso");
        printf("\n[4] Uscire\n");

        printf("\nInserire scelta: "); scanf ("%hd", &Scelta);
        fflush (stdin);
        switch (Scelta)
        {
            case 1: Inserisci_Studente(); Ordina_Mat(); break;
            case 2: Crea_File_Di_Aggiorramento(); Aggiorna_Archivio();
                      Visualizza_Archivio(FILE_ARCHIVIO_PROMOSSI); break;
            case 3: Visualizza_Archivio(FILE_ARCHIVIO); break;
        }
        printf ("\n\n");
        fflush (stdin);
    }
    while (Scelta != 4);

    return 0;
}

/* Appartiene e' una semplice ricerca binaria tra la matricola da ricercare(key) e
```

```

la lista su cui cercare (Studente).
PERCHE' SCEGLIERE LA RICERCA BINARIA? PERCHE' OGNI VOLTA CHE AGGIUNGIAMO
UNO STUDENTE, LO ORDINIAMO, COSA DA OTTENERE SEMPRE UN ARCHIVIO ORDINATO E
SFRUTTARE LA POTENZA DELLA RICERCA BINARIA O(LOG2N) */
int Appartiene(char *Key, STUDENTI *Studente, int N)
{
    int Mediano, Inizio=0, Fine=N;
    while (Inizio<=Fine)
    {
        Mediano=(Inizio+Fine)/2; //ottieni mediano
        if(strcmp(Key, Studente[Mediano].Matricola)==0)
            return Mediano; //Mediano risulterà essere la posizione
        /* Se la chiave è maggiore, prendi la porzione di destra*/
        else if(strcmp(Key, Studente[Mediano].Matricola)>0)
            Inizio=Mediano+1;
        /* Se la chiave è minore di mediano, prendi la porzione di sinistra*/
        else
            Fine=Mediano-1;
    }
    //Se non trova nulla
    return -1;
}

/* IN ALTERNATIVA, MA MENO VANTAGGIOSA, LA RICERCA SEQUENZIALE
int i;
for(i=0;i<n_stud; i++)
    if(strcmp(Key, Studente[i].Matricola)==0)
        return i; /* Matricola univoca->Appena trovi, restituisci indice di posizione
*/
//finiti i cicli, significa che non avrà trovato nulla
/*return -1;*/
}

/* Aggiorna archivio ha tutto per procedere. A questo punto,
per semplificare le ricerche e non accedere troppo al file (ACCESSO AL FILE
è molto lento), memorizziamo in un Array la lista di tutti i studenti con
relative informazioni e in un altro array memorizziamo solo gli studenti
promossi con i nuovi voti. Da qui si procede alla ricerca della matricola
da aggiornare (se non viene trovata perché si sbaglia ad inserire, viene
segnalato dal programma) e, trovato lo studente promosso, si aggiorna il
totale dei crediti con la nuova media. Ora non resta che andare a scrivere
nel file "Studenti2.dat" */
void Aggiorna_Archivio()
{
    FILE *fp;
    AGGIORNAMENTO Studente_da_Agg[200]; //Conterrà studenti da aggiornare
    STUDENTI Studente[200]; //lista studenti corso
    int i=0, n_stud_da_Agg=0, n_stud, Pos=0;

    /* MEMORIZZA AGGIORNAMENTO */
    fp=fopen(FILE_AGGIORNAMENTO,"rb"); //apre il file binario in lettura
    /* Memorizzo tutto in un vettore per semplificare */
    while (!feof(fp)) //finché non è finito il file degli studenti
    {
        /*legge uno alla volta gli studenti dal file*/
        fread (&Studente_da_Agg[n_stud_da_Agg], sizeof(AGGIORNAMENTO), 1, fp);
        n_stud_da_Agg++; /*incrementa il contatore del numero degli studenti presenti*/
    }
    n_stud_da_Agg--; //Decrementa per aver contato automaticamente
    fclose (fp); //Chiudi file dell'aggiornamento

    /* MEMORIZZA ARCHIVIO */
    /* Memorizzo tutto in un vettore per non avere stream su file diversi */
    fp=fopen(FILE_ARCHIVIO,"rb"); //apre il file binario in lettura
    while (!feof(fp)) //finché non è finito il file degli studenti
    {
        /*legge uno alla volta gli studenti dal file*/
        fread (&Studente[n_stud], sizeof(STUDENTI), 1, fp);
        n_stud++; /*incrementa il contatore del numero degli studenti presenti*/
    }
    n_stud--; //Decrementa per aver contato automaticamente
    fclose (fp); //Chiudi file dell'aggiornamento

    /* RICERCA */
    /* Se la matricola da aggiornare è stata immessa bene, verrà trovata
       nell'archivio e salvata nel nuovo file studenti2.dat, come da traccia */
    STUDENTI Scheda;
    fp=fopen(FILE_ARCHIVIO_PROMOSSI,"wb"); //apre il file binario in scrittura (dall'ultimo eof)

```

```

for(i=0; i<n_stud_da_Agg; i++)
{
    /* Appartiene restituisce la pos nel vettore studenti. Se non trova nulla,
     * restituisce -1 come errore*/
    Pos = Appartiene(Studente_da_Agg[i].Matricola, Studente, n_stud);
    if(Pos===-1)
    {
        printf("\nLa matricola %s non e' stata trovata", Studente_da_Agg[i].Matricola);
    }
    else
    {
        /* Non voglio sporcare l'archivio, salvo una scheda alla volta */
        Scheda = Studente[Pos];
        /* Aggiorno voti e crediti
         * MediaNuova=(OldMedia*oldcfu+cfuNuovo*VotoNuovo) / (Oldcfu+cfuNew) */
        Scheda.Media = ((Scheda.Media*Scheda.Crediti+
                          CREDITI_PROG2*Studente_da_Agg[i].Voto_Nuovo) /
                         (Scheda.Crediti+CREDITI_PROG2));
        Scheda.Crediti+=CREDITI_PROG2; //anche il tot dei cfu e' aggiornato
        fwrite(&Scheda, sizeof(STUDENTI), 1, fp);
    }
}
fclose (fp); // Chiudi file dell'aggiornamento
}

/* La seguente function ha il seguente compito di costruire un file contenente
 * le matricole promosse con i relativi voti;
 * RAGIONAMENTO: Dato che il numero degli studenti promossi e' molto variabile,
 * creiamo un vettore di tipo AGGIORNAMENTO, che contiene gli n studenti
 * promossi e che successivamente dobbiamo aggiornare. Inseriti quanti ne sono
 * e chi sono (per matricola), vengono scritti nel file.
 * ATTENZIONE: Non e' assolutamente la miglior scelta, ma per motivi didattici
 * e richiesta della traccia del problema, andiamo a salvare tutto su file */
void Crea_File_Di_Aggiorramento()
{
    int n_da_aggiornare, i;
    AGGIORNAMENTO *Update_Studenti;
    FILE *fp; //dichiarazione puntatore a file

    printf("Inserire Numero di studenti promossi");
    scanf("%d", &n_da_aggiornare);

    /* Contenitore di studenti promossi */
    Update_Studenti=(AGGIORNAMENTO *)malloc(sizeof(AGGIORNAMENTO)*n_da_aggiornare);
    if(Update_Studenti==NULL) {printf("ERRORE DI ALLOCAZIONE"); exit(EXIT_FAILURE);}

    fp=fopen (FILE_AGGIORNAMENTO,"wb"); //apre il file binario in scrittura

    /* scrivi su file i studenti promossi con relativo voto */
    for(i=0; i<n_da_aggiornare; i++)
    {
        fflush(stdin);
        printf("Inserire la matricola da aggiornare: "); gets(Update_Studenti[i].Matricola);
        fflush(stdin);
        printf("Inserire il voto: "); scanf("%hd", &Update_Studenti[i].Voto_Nuovo);
        fwrite(&Update_Studenti[i], sizeof(AGGIORNAMENTO), 1, fp); //scrivi su file
        printf("\n\n");
    }
    fclose (fp); //Chiudi file
}

/* Inserire i studenti nell'archivio principale; dati i nominativi, vengono
 * scritti nel FILE_ARCHIVIO */
void Inserisci_Studente()
{
    STUDENTI Persona;
    FILE *fp;

    fp=fopen (FILE_ARCHIVIO,"ab"); //apre il file binario in scrittura (dall'ultimo eof)
    /* ASSEGNA I CAMPI */
    printf("\nInserisci nome e cognome dello studente: "); gets(Persona.Nome_Cognome);
    printf("\nInserisci matricola: "); gets(Persona.Matricola);
    printf("\nInserisci il numero di esami fatti: "); scanf("%hd", &Persona.N_esami);
    printf("\nInserisci la media: "); scanf("%f", &Persona.Media);
    printf("\nInserisci il numero di crediti: "); scanf("%hd", &Persona.Crediti);
    //Scrivi su file
    fwrite(&Persona, sizeof(Persona), 1, fp);
    fclose(fp); //Chiudi file
}

```

```

/* Visualizza cio' che c'Ã" nei FILE */
void Visualizza_Archivio(char *Nome_File)
{
    FILE *fp;
    STUDENTI Scheda;
    int N, i=0;
    fp=fopen(Nome_File,"rb"); //apre il file binario in scrittura (dall'ultimo eof)
    /* Ricava il numero di Studenti presenti nel file */
    fseek(fp, 0, SEEK_END); //posizione alla fine
    N=ftell(fp)/sizeof(Scheda); /* Posizione finale diviso lo spazio che occupa
                                    ogni struct, ci da' il numero effettivi di studenti */
    fseek(fp, 0, SEEK_SET); //Riposiziona dall'inizio
    printf("\n\n");
    while(i<N) //Dovrebbe andare while(!feof(fp)), ma stampa 2 volte l'ultima struct
    {
        fread(&Scheda, sizeof(Scheda),1,fp); //Leggi la riga e vai avanti
        printf("Nome e Cognome: %s\n", Scheda.Nome_Cognome);
        printf("Matricola: %s\n", Scheda.Matricola);
        printf("Numero esami: %d\n", Scheda.N_esami);
        printf("Media: %.2f\n", Scheda.Media);
        printf("Crediti: %d\n", Scheda.Crediti);
        printf("\n\n");
        i++;
    }
    fclose (fp); //Chiudi file
}

```

```

/* ORDINA GLI STUDENTI NEI FILE IN BASE ALLA MATRICOLA.
   (IMPLEMENTAZIONE DEL BUBBLE SORT) */
void Ordina_Mat()
{
    int i=0, //Inizio da ordinare
        f, // indice corrente sulle coppie
        s; //Ultimo scambio effettuato

    int N;

    FILE *Pf; //Stream
    STUDENTI Scheda1, Scheda2; //Schede che conterranno i record da confrontare

    Pf = fopen (FILE_ARCHIVIO, "r+b"); //Apriamo il file dei record
    fseek(Pf, 0, SEEK_END);
    N = ftell(Pf)/sizeof(STUDENTI); /* Ricavo il numero di studenti*/

    /* BUBBLE SORT
       NB Non e' l'algoritmo piu' efficiente, ma per il problema considerato e' efficiente, dato
       che avremo sempre N-1 elementi gia' ordinati, quindi si avvia il caso migliore del bubble
       o(N)*/
    while(i < N-1)
    {
        s = f = N - 1;
        while(f>i)
        {
            //Prendi i record da confrontare e salvalo nelle 2 schede
            fseek(Pf, sizeof(STUDENTI)*f, SEEK_SET);
            fread(&Scheda2, sizeof(STUDENTI),1,Pf);
            fseek(Pf, sizeof(STUDENTI)*(f-1), SEEK_SET);
            fread(&Scheda1, sizeof(STUDENTI),1,Pf);
            //Confronta i dati
            if (strcmp (Scheda1.Matricola, Scheda2.Matricola) > 0)
            {
                //Scambio(senza variabile di app, scrivo direttamente)
                fseek(Pf, sizeof(STUDENTI)*f, SEEK_SET);
                fwrite(&Scheda1, sizeof(STUDENTI),1,Pf);
                fseek(Pf, sizeof(STUDENTI)*(f-1), SEEK_SET);
                fwrite(&Scheda2, sizeof(STUDENTI),1,Pf);
                s = f; //memorizziamo la zona in cui Ã" avvenuto lo scambio
            }
            f = f - 1;
        }
        i = s; //scavalca la zona gia` ordinata
    }
    fclose (Pf); //Chiudi file
}

```

}

**OUTPUT:**

```
***** MENU *****  
[1] Inserire nuovo studente  
[2] Aggiornare dati dei studenti promossi  
[3] Visualizzare la lista degli studenti del corso  
[4] Uscire
```

Inserire scelta: 1

Inserisci nome e cognome dello studente: Giuseppe Accardo

Inserisci matricola: 0124000879

Inserisci il numero di esami fatti: 7

Inserisci la media: 29

Inserisci il numero di crediti: 40

```
***** MENU *****  
[1] Inserire nuovo studente  
[2] Aggiornare dati dei studenti promossi  
[3] Visualizzare la lista degli studenti del corso  
[4] Uscire
```

Inserire scelta: 2

Inserire Numero di studenti promossi

Inserire la matricola da aggiornare: 124000879

Inserire il voto: 30

Nome e Cognome: Giuseppe Accardo

Matricola: 124000879

Numero esami: 7

Media: 29.16

Crediti: 58

```
***** MENU *****
```

```
[1] Inserire nuovo studente  
[2] Aggiornare dati dei studenti promossi  
[3] Visualizzare la lista degli studenti del corso  
[4] Uscire
```

Inserire scelta: 1

Inserisci nome e cognome dello studente: Pierluigi Lipardi

Inserisci matricola: 124000789

Inserisci il numero di esami fatti: 4

Inserisci la media: 24

Inserisci il numero di crediti: 31

```
***** MENU *****
```

```
[1] Inserire nuovo studente  
[2] Aggiornare dati dei studenti promossi  
[3] Visualizzare la lista degli studenti del corso  
[4] Uscire
```

Inserire scelta: 3

Nome e Cognome: Pierluigi Lipardi

Matricola: 124000789

Numero esami: 4

Media: 24.00

Crediti: 31

Nome e Cognome: Giuseppe Accardo

Matricola: 124000879

Numero esami: 7

Media: 29.00

Crediti: 49

## ESERCIZIO 35 [LIV.1]

```
/** [liv.] Scrivere una function C per la ricerca diretta di tutte le occorrenze di un pattern in un testo dove:
    o il testo è memorizzato in un file;
    o la lettura del file avviene a pezzi mediante un array-buffer;
    o il pattern è definito in input (e costruito dinamicamente). **/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define BUFSIZE 30
short cerca_patt_in_buff(int, char*, FILE *); //Prototipo della function
int main()
{
    char scelta,*p_patt, *p_buff, nomefile[60];
    FILE *fp;
    int len_patt;
    short quanti; //Conta le occorrenze
    printf("Introduci la lunghezza del pattern : ");
    scanf("%d",&len_patt);len_patt++; // len++ per '\0'
    p_patt=(char*)malloc(len_patt*sizeof(char));
    if(p_patt==NULL)
    {printf("Memoria insufficiente\n");exit(1);}
    else
    {
        printf("\nIntroduci il pattern di ricerca poi [INVIO] : ");
        fflush(stdin); gets(p_patt);
        printf("\nSpecificare il nome del file su cui condurre la ricerca: ");
        gets(nomefile);
        fp=fopen(nomefile,"r");
        if(fp==NULL)
        { printf("Impossibile aprire il file\n"); exit(1);}
        else
        {
            quanti=cerca_patt_in_buff(len_patt-1,p_patt,fp);
            fclose(fp);
            printf("\nNumero delle occorrenze del pattern nel testo : %d\n",quanti);
        }
    }
    puts("");
    return 0;
}

/* Si noti che la function "cerca_patt_in_buff" e' in grado di individuare
le occorrenze di un pattern in un testo anche quando il pattern e' "diviso"
tra più letture */
short cerca_patt_in_buff(int len_patt,char *p_patt,FILE *fp)
{
    char *ch; //Puntatore al pattern trovato
    char *ultimi_jc; // Punta al suffisso del testo di grandezza a partire da Len_pat-1,
                     // attraverso j so' quanti eventuali caratteri mi mancano
    char buffer[BUFSIZE]; // Buffer contenente testo estratto
    char p_null='\0'; //NULL
    short num_trovati, j;
    num_trovati=j=0;
    /* Fin quando trova qualcosa nel file */
    while(!feof(fp))
    {
        *buffer=p_null; //Buffer[0] = '\0', stringa "vuota"
        fgets(buffer,BUFSIZE,fp); //Ottieni dal file un testo di BUFSIZE caratteri in buffer

        ch=strstr(buffer,p_patt+j); /* Dammi l'indirizzo della prima occorrenza.
                                       Se ci sono 2 stringhe a cavallo e riconosco alla prima stringa
                                       un suffisso uguale a metà' del prefisso del pattern, ricordo
                                       di quanti posti dovrò spostare il pattern per cercare
                                       l'eventuale resto nella seconda stringa, partendo da patt+j
                                       */
        while(ch!=NULL) /* Se ne ha trovato uno, conta e trova altri*/
        {
            num_trovati++;
            ch=strstr(ch+len_patt-j,p_patt);/* Cerca altro, a partire da ch+len_patt-j.
                                              "Selva" e alla seconda stringa trovavo "lva",
                                              Devo avanzare di ch+len_pat, ma considerare
                                              quel j considerato nella stringa precedente*/
            j=0;
        }
    }
}
```

```

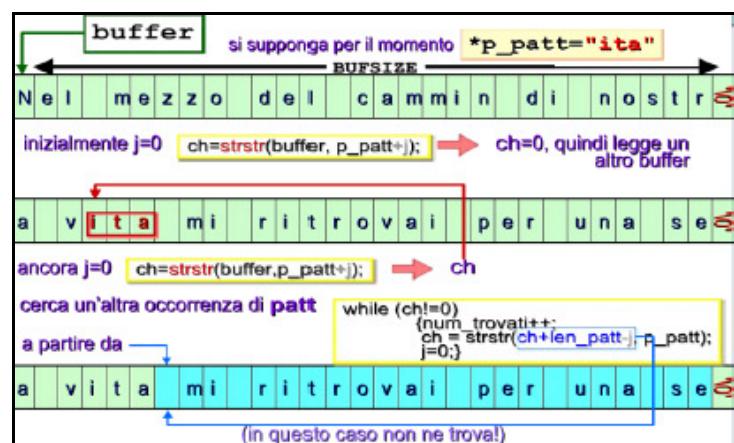
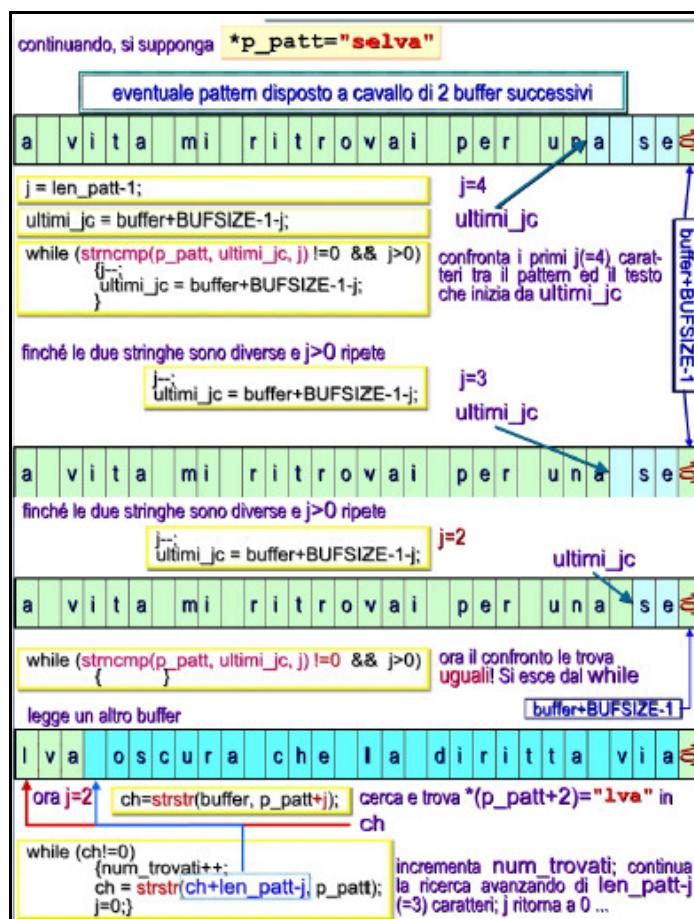
/* eventuale pattern disposto su due buffer successivi
   calcola l'eventuale numero j di caratteri finali del buffer
   che sono uguali a quelli iniziali del pattern
   (PREFISSO E SUFFISSO)
*/
j=len_patt-1; /* non dobbiamo considerare tutto il pattern, perche' ne troviamo una parte*/
ultimi_jc=buffer+BUFSIZE-1-j; /* Cio' puntato dal suffisso*/
while(strncmp(p_patt,ultimi_jc,j)!=0 && j>0) //Fin quando non trova o j si esaurisce
{
    j--;
    ultimi_jc=buffer+BUFSIZE-1-j; /* Considera porzioni piu' ristrette */
}
return num_trovati;
}

```

**OUTPUT:****Introduci la lunghezza del pattern : 5****Introduci il pattern di ricerca poi [INVIO] : selva****Specificare il nome del file su cui condurre la ricerca: File\_Testo.txt****Numero delle occorrenze del pattern nel testo : 1**

File\_Testo.txt

1 Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura ché la diritta via era smarrita.

**Introduci la lunghezza del pattern : 3****Introduci il pattern di ricerca poi [INVIO] : ita****Specificare il nome del file su cui condurre la ricerca: File\_Testo.txt****Numero delle occorrenze del pattern nel testo : 2**

## ESERCIZIO 36 [LIV.1]

```
/*
[liv.1] Simulare in C la gestione di una pila (stack) tramite array statico creando
le funzioni di manipolazione push() [inserimento] e pop() [eliminazione].
Il programma deve prevedere un menù che consenta di scegliere l'operazione da eseguire.
*/
#include <stdio.h>
#include <stdlib.h>
#define LEN_STACK 5

void Push(char Vet_Stack[], char Elemento, short *Testa);
void Pop(short *Testa);
/* COSA E' UNA PILA: e' una struttura lineare in cui l'inserimento e l'eliminazione
avvengono in un solo estremo detto TESTA
L.I.F.O.
*/

int main()
{
    char Scelta, i;
    char Vet_Stack[LEN_STACK]; /* Simuliamo uno stack*/
    short Testa = -1; /* individua uno stack pointer.Punta sempre all'ultimo elemento
inserito */
    char Elemento;
    do
    {
        printf("***** MENU *****\n");
        printf("[1] Push[inserisci un char]\n[2] Pop[Elimina un char]\n[3] Esci\n\n");
        scanf("%c", &Scelta); fflush(stdin);

        switch (Scelta) //si richiamano le funzioni corrispondenti alla scelta inserita
        {
            case '1': printf("Inserisci un carattere\n"); Elemento = getch();
                        Push(Vet_Stack, Elemento, &Testa); break;
            case '2': Pop(&Testa); break;

            }
        /* _____ Visualizza Stack a partire dalla testa _____ */
        printf("*----- Visualizzazione stack -----* \n");
        if(Testa>-1) for(i=Testa; i>=0; i--) printf("\t\t [%d] %c\n", i ,Vet_Stack[i]);
        else printf("\t\t Stack Vuoto\n");
        /* ----- */
        printf ("\n\n"); fflush(stdin);
    }
    while (Scelta != '3');
    return 0;
}

/* Push: Inserire in testa dell'array, appunto, come lo stack. */
void Push(char Vet_Stack[], char Elemento, short *Testa)
{
    if(*Testa<LEN_STACK) //Se non ho superato il massimo dello stack
    {
        /* *Testa=*Testa+1 -> Vet_Stack=Elemento
        La mia testa punterà sempre all'ultimo elemento inserito, quindi incrementando
        metterò l'elemento in testa o PUSHO */
        Vet_Stack[++(*Testa)] = Elemento;
    }
    else printf ("\n----- NON INSERISCO NULLA, STACK PIENO! -----*\n");
}

/* Pop: Eliminare dalla testa dell'array. */
void Pop(short *Testa)
{
    if(*Testa>-1) //Se c'e' qualcosa
    {
        /* Decrementando l'indice dell'ultimo elemento inserito, appunto, non verrà
        più "visto" l'ultimo elemento inserito, nascondendolo*/
        (*Testa)--;
    }
}
```

```
//altrimenti "Stack vuoto" viene gestito nel main
}
PS: il carattere viene inserito con getch()
```

**OUTPUT:**

```
***** MENU *****
[1] Push[inserisci un char]
[2] Pop[Elimina un char]
[3] Esci
```

```
1
Inserisci un carattere
 [0] A
```

```
***** MENU *****
[1] Push[inserisci un char]
[2] Pop[Elimina un char]
[3] Esci
```

```
1
Inserisci un carattere
 [1] B
 [0] A
```

```
***** MENU *****
[1] Push[inserisci un char]
[2] Pop[Elimina un char]
[3] Esci
```

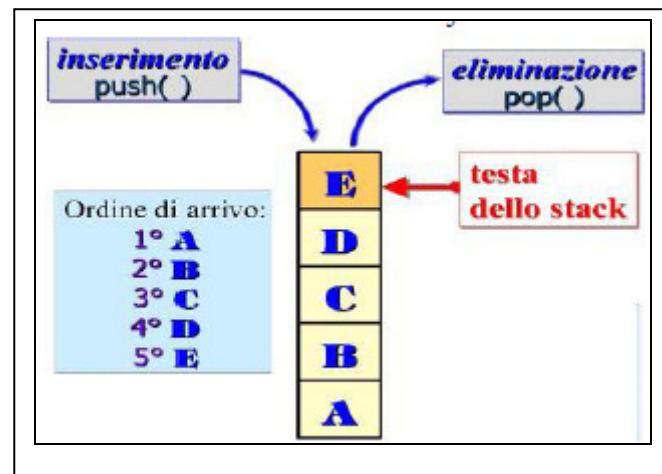
```
1
Inserisci un carattere
 [2] C
 [1] B
 [0] A
```

```
***** MENU *****
[1] Push[inserisci un char]
[2] Pop[Elimina un char]
[3] Esci
```

```
2
 [1] B
 [0] A
```

```
***** MENU *****
[1] Push[inserisci un char]
[2] Pop[Elimina un char]
[3] Esci
```

```
2
 [0] A
```



## ESERCIZIO 37 [LIV.1]

```
/*
[liv.1] Simulare in C la gestione di una coda(queue) tramite array statico creando le funzioni di
manipolazione enqueue() [inserimento] e dequeue() [eliminazione]. Il programma deve
prevedere un menu che consente di scegliere l'operazione da eseguire.
Le informazioni NON vanno spostate!
**/


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

#define LEN_QUEUE 5
void Enqueue(char Vet_Coda[], char Elemento, short *Fondo);
void Dequeue(short *Testa);

/* COSA E' UNA CODA: e' una struttura lineare in cui l'inserimento avviene in FONDO
e l'eliminazione in TESTA. Quindi l'accesso e' su due fronti.
F.I.F.O
NB non e' il miglior modo, dato che non posso sfruttare le eliminazioni lasciate da TESTA
(SOLUZIONE?ARRAY CIRCOLARE!) */
int main()
{
    char Scelta, i;
    char Vet_Coda[LEN_QUEUE]; /* Simuliamo una coda */
    short Testa=0; /* individua l'indice del primo elemento da estrarre[TESTA della coda]*/
    short Fondo=0; /* individua l'indice dell'ultimo elemento dove inserire in coda[FONDO]*/
    char Elemento;

    do
    {
        printf("***** MENU *****\n");
        printf("[1] Enqueue[inserisci un char]\n[2] Dequeue[Elimina un char]\n[3]
Esci\n\n");
        scanf("%c", &Scelta);
        fflush(stdin);

        switch (Scelta) //si richiamano le funzioni corrispondenti alla scelta inserita
        {
            case '1': printf("Inserisci un carattere\n"); Elemento = getch();
                        Enqueue(Vet_Coda, Elemento, &Fondo); break;
            case '2': /* Cancello se ho almeno un valore in coda */
                        if(Fondo!=Testa) Dequeue(&Testa);
                        else printf("Coda vuota, non elimino\n");
                        break;
        }

        /* _____ Visualizza Coda a partire dalla fine _____ */
        /* Se Fondo==Testa, la coda e' vuota.*/
        if(Testa<LEN_QUEUE&&Fondo!=Testa)
        {
            printf("----- Visualizzazione Coda ----- \n\n");
            for(i=Fondo-1; i>=Testa; i--) printf("\t\t [%d] %c\n", i ,Vet_Coda[i]);
        }
        /* ----- */
        printf ("\n\n");
        fflush(stdin);
    }
    while (Scelta != '3');

    return 0;
}

/* Testa <-A[0,1.....N-1]-> Fondo */
/* Enqueue: Inserire al fondo dell'array l'elemento. */
void Enqueue(char Vet_Coda[], char Elemento, short *Fondo)
{
    /* Vet_Coda[*Fondo]=Elemento -> (*Fondo)++;
       Inserisco in coda e aumento Fondo, che mi indica
       la il posto della fila successiva */
    if(*Fondo<LEN_QUEUE) Vet_Coda[( *Fondo )+1]=Elemento;
    else printf("\n----- NON INSERISCO NULLA, FONDO FINITO! ----- \n");
}

/* Dequeue: Eliminare dalla testa dell'array. */

```

```

void Dequeue (short *Testa)
{
    /* Vet_Coda[*Fondo]=Elemento -> (*Fondo)++; */
    /* Testa arrivata a LEN_QUEUE, mi indichera' che non potro' ne inserire un elemento ne
     * eliminarlo.
     * Avanzando testa, elimino il primo elemento in testa. */
    if (*Testa<LEN_QUEUE) (*Testa)++;
}

```

#### OUTPUT:

\*\*\*\*\* MENU \*\*\*\*\*  
[1] Enqueue[inserisci un char]  
[2] Dequeue[Elimina un char]  
[3] Esci

1  
Inserisci un carattere  
[0] A

\*\*\*\*\* MENU \*\*\*\*\*  
[1] Enqueue[inserisci un char]  
[2] Dequeue[Elimina un char]  
[3] Esci

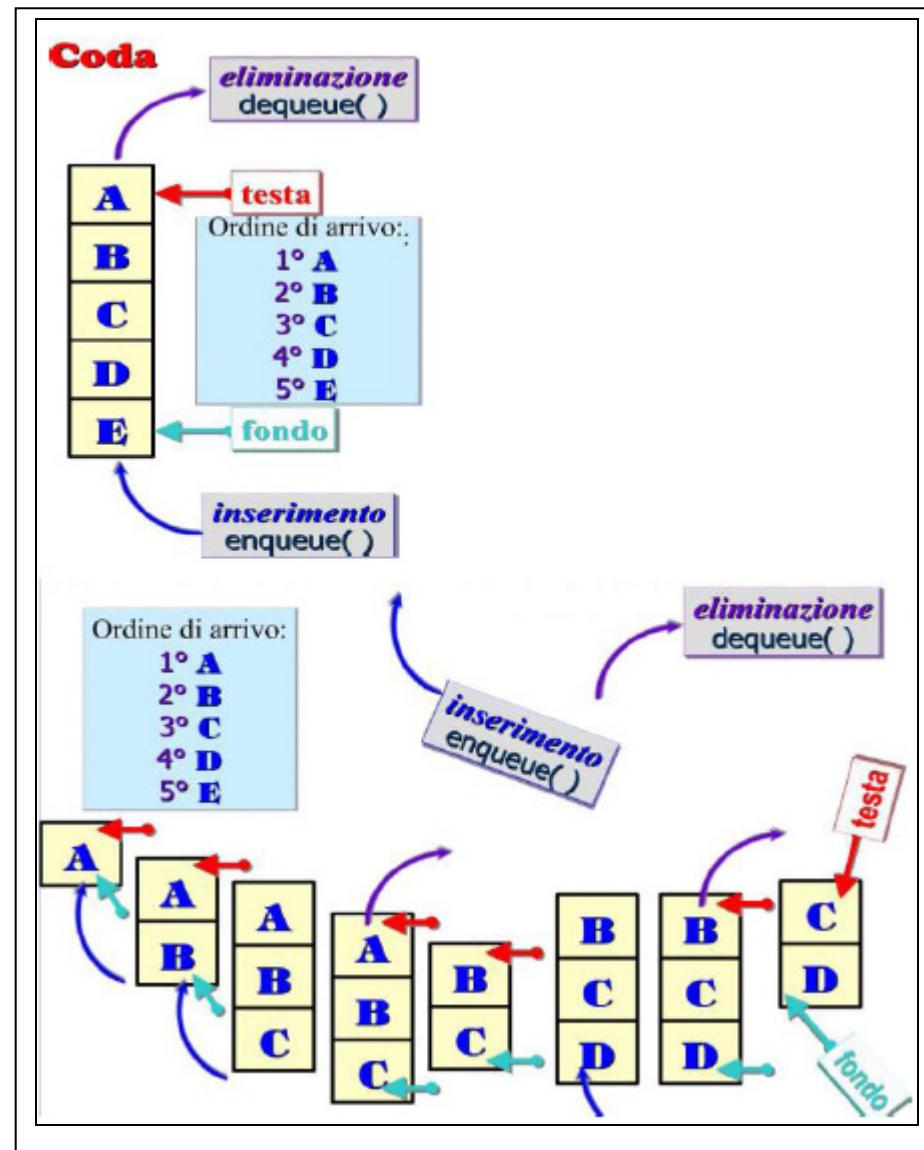
1  
Inserisci un carattere  
[1] B  
[0] A

\*\*\*\*\* MENU \*\*\*\*\*  
[1] Enqueue[inserisci un char]  
[2] Dequeue[Elimina un char]  
[3] Esci

1  
Inserisci un carattere  
[2] C  
[1] B  
[0] A

\*\*\*\*\* MENU \*\*\*\*\*  
[1] Enqueue[inserisci un char]  
[2] Dequeue[Elimina un char]  
[3] Esci

2  
[2] C  
[1] B



\*\*\*\*\* MENU \*\*\*\*\*  
[1] Enqueue[inserisci un char]  
[2] Dequeue[Elimina un char]  
[3] Esci

2  
[2] C

## ESERCIZIO 38 [LIV.3]

```
/*
[liv.3] Simulare in C la gestione di una coda(queue) tramite array statico
circolare creando le funzioni di manipolazione enqueue() [inserimento] e
dequeue() [eliminazione]. Il programma deve prevedere un menu che consenta
di scegliere l'operazione da eseguire.
*/
#include <stdio.h>
#include <stdlib.h>

#define LEN_QUEUE 5
void Enqueue(char Vet_Coda[], char Elemento, short Testa, short *N_Elem);
void Dequeue(char Vet_Coda[], char Elemento, short *Testa, short *N_Elem);

/* COSA E' UNA CODA: e' una struttura lineare in cui l'inserimento avviene in FONDO
e l'eliminazione in TESTA. Quindi l'accesso e' su due fronti.
F.I.F.O
*/
/*
ARRAY CIRCOLARE: E' appunto un array in cui gli elementi "Girano".(Struttura chiusa)
ESEMPIO: se testa sta ad una posizione di 1 e fondo ad N-1 del vettore,
possiamo inserire un altro elemento che non va ad N del
vettore (andrebbe oltre), ma MOD[TESTA+N_ELEM]LEN_ARRAY, che ci dà il corrispondente
indice dell'inizio dell'array, rimasto vuoto da Testa. Se ho inizialmente N_ELEM=N-1,
TESTA+N_ELEM mi individua N, ma effettuando il Modulo con LEN_ARRAY, ottengo l'indice 0,
quindi una retroazione in ingresso.
*/
int main()
{
    char Scelta, i;
    char Vet_Coda_Circle[LEN_QUEUE]; /* Simuliamo una coda */
    short Testa = 0; /* Mi individua l'indice del primo elemento da estrarre[dalla TESTA della coda]*/
    short Fondo = 0; /* individua l'indice dell'ultimo elemento dove inserire in coda[dal FONDO]*/
    short N_Elem = 0; /* Quanti elementi ci sono in coda */
    char Elemento;
    do
    {
        printf("***** MENU *****\n");
        printf("[1] Enqueue[inserisci un char]\n[2] Dequeue[Elimina un char]\n[3] Esci\n\n");
        scanf("%c", &Scelta);
        fflush(stdin);

        switch (Scelta) // si richiamano le funzioni corrispondenti alla scelta inserita
        {
            case '1': printf("Inserisci un carattere\n"); Elemento = getch();
                if(N_Elem<LEN_QUEUE) Enqueue(Vet_Coda_Circle, Elemento, Testa, &N_Elem);
                else printf("____ Pila piena!! ____\n");
                break;
            case '2': /* Cancello se ho almeno un valore in coda */
                if(N_Elem) Dequeue(Vet_Coda_Circle, Elemento, &Testa, &N_Elem);
                else printf("Coda vuota, non elimino\n");
                break;

            }
        /* _____ Visualizza Pila a partire da FONDO a TESTA _____ */
        Fondo=N_Elem+Testa; //Ricavo il fondo
        if(N_Elem<=LEN_QUEUE&&N_Elem>0)
        {
            printf("----- Visualizzazione pila -----* \n\n");
            for(i=N_Elem-1; i>=0; i--)
                printf("\t\t [%d] %c\n", (i+Testa)%LEN_QUEUE , Vet_Coda_Circle[(i+Testa)%LEN_QUEUE]);
        }
        /* ----- */
        printf ("\n\n");
        fflush(stdin);
    }
    while (Scelta != '3');

    return 0;
}

/* _____ Testa <-A[0,1,2.....N-1]-> Fondo _____ */

/* Enqueue: Inserire al fondo dell'array l'elemento.
NB Il controllo se posso inserire altri elementi
confrontando N_Elem con LEN_PILA e' stato fatto nel main*/
void Enqueue(char Vet_Coda[], char Elemento, short Testa, short *N_Elem)
```

```

{
    short Fondo;
    Fondo= Testa+*N_Elem; //Ricavo fondo, che indica dove posso inserire
    Fondo= Fondo%LEN_QUEUE; /*Ricavo l'indice per "la circolazione"(Es. Se fondo>N, circola dietro)*/
    Vet_Coda[Fondo]=Elemento; //Inserisco in fondo
    (*N_Elem)++; // conta elementi
}

/* Dequeue: Eliminare dalla testa dell'array.
   NB Il controllo della coda vuota e' fatto nel main */
void Dequeue(char Vet_Coda[], char Elemento, short *Testa, short *N_Elem)
{
    *Testa= *Testa+1; //Avanza la testa, per non considerare piu' l'elemento corrente in testa
    *Testa = *Testa%LEN_QUEUE; //Anche qui puo' capitare la circolazione
    (*N_Elem)--; //avro' meno elementi
}

```

### OUTPUT:

\*\*\*\*\* MENU \*\*\*\*\*  
 [1] Enqueue[inserisci un char]  
 [2] Dequeue[Elimina un char]  
 [3] Esci

RIEMPI LA PILA (Visualizza da 'Fondo' a 'Testa')

[0] A

[1] B  
 [0] A

[2] C  
 [1] B  
 [0] A

[3] D  
 [2] C  
 [1] B  
 [0] A

[4] E  
 [3] D  
 [2] C  
 [1] B  
 [0] A

Pila piena!!

SVUOTO UN PO' LA PILA (Visualizza da 'Fondo' a 'Testa')

[4] E  
 [3] D  
 [2] C  
 [1] B

[4] E  
 [3] D  
 [2] C

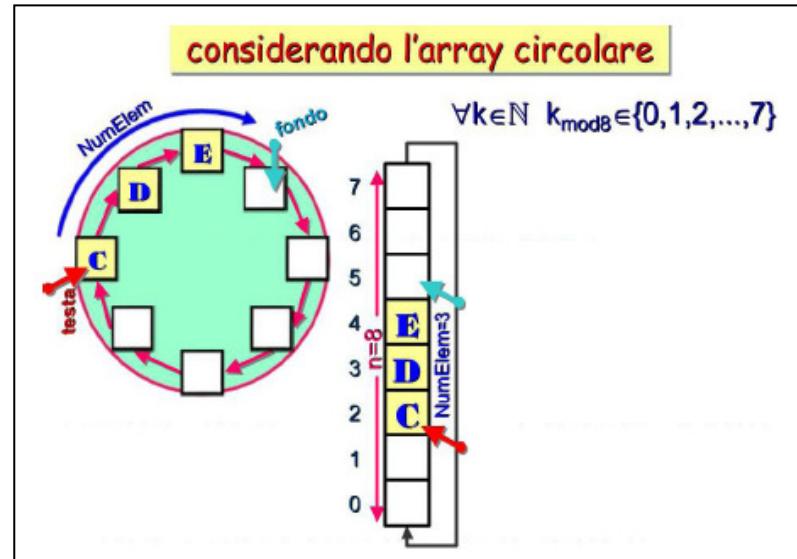
[4] E  
 [3] D

INSERISCO DAL FONDO : fondo e'  
 All'ultima posizione, ma sfruttando il modulo,  
 fondo circola andando alla testa della coda.  
 (Visualizza da 'Fondo' a 'Testa')

[0] h  
 [4] E  
 [3] D

[1] i  
 [0] h  
 [4] E  
 [3] D

[2] Z  
 [1] i  
 [0] h  
 [4] E  
 [3] D



## ESERCIZIO 39 [LIV.1]

```
/**
[liv.1]Simulare in C l'algoritmo di visita di una lista lineare già memorizzata mediante un array
statico di struct in cui il primo campo contiene l'informazione ed il secondo contiene il link al
nodo successivo (in questo caso il link è l'indice di una componente dell'array).
Memorizzando nell'array i dati come mostrato nella figura che segue, l'output del programma
consiste nell'elenco di nomi ordinato alfabeticamente.
*/
#include <stdio.h>
#include <stdlib.h>
typedef struct Lista
{
    char Nome[15];
    short Next;
} LISTA;
int main()
{
    short p_testa=3;
    LISTA Vett[]={{"Anna", 5}, {"Mario", 8}, {"Giuseppe", 6}, {"Angela", 0}, {"Valeria", -1}, {"Fabrizio", 7},
                  {"Marianna", 1}, {"Giovanni", 2}, {"Patrizia", 10}, {"Valentina", 4}, {"Sara", 9}
                  };
    printf("--- VISITA DI UN ALGORITMO DI UNA LISTA LINEARE IN UN ARRAY STATICO ---\n\n");
    /* Visita, ossia accedo alle celle dei vettori e avanzo testa in base all'indirizzo del
       nodo prossimo.
       A se un nodo punta a -1, e' l'ultimo*/
    while(p_testa!=-1)
    {
        printf("%s\n", Vett[p_testa].Nome);
        p_testa=Vett[p_testa].Next;
    }
    printf("\n\n");
    return 0;
}
```

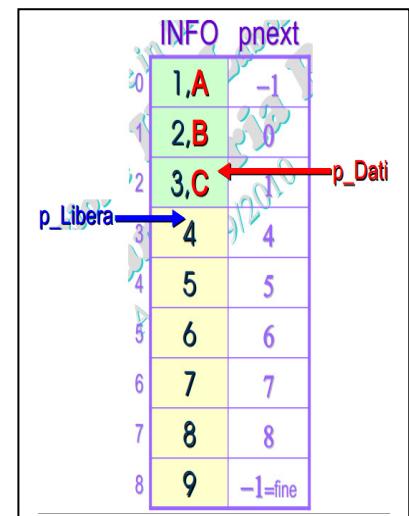
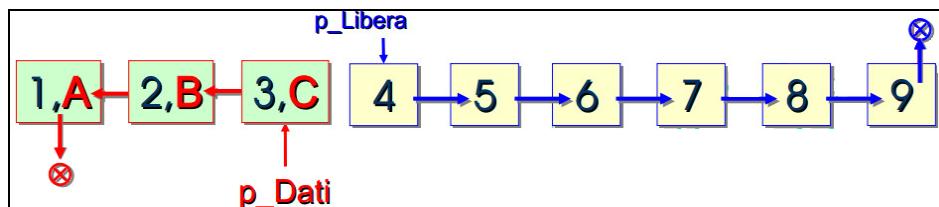
### OUTPUT:

--- VISITA DI UN ALGORITMO DI UNA LISTA LINEARE IN UN ARRAY STATICO ---

Angela  
 Anna  
 Fabrizio  
 Giovanni  
 Giuseppe  
 Marianna  
 Mario  
 Patrizia  
 Sara  
 Valentina  
 Valeria

## ESERCIZIO 40 [LIV.3]

/\*\* [liv.3] Simulare in C la gestione delle camere di un albergo mediante liste lineari rappresentate su un array distract: i principali campi sono le informazioni (numero di camera, cliente, etc.) ed i link (puntatori ai nodi della lista) \*\*/



```
/* COME FUNZIONA ALGORITMO: Abbiamo 2 liste che lavorano su un solo array di struct:
   - p_dati e' l'indice che punta alla testa della lista dei dati
   - p_libera e' l'indice che punta alla testa della lista delle camere libere.
   L'inserimento e' sempre in testa a ListaDati mentre l'eliminazione ha 2
   casi.*/
#include <stdio.h>
#include <stdlib.h>
/* Identificazione camera */
typedef struct Camera_struct
{
    char Cliente[16];
    short Id_Camera;
    short Next; //In questo caso gli indici fungono da puntatori
}CAMERA_STRUCT;

void Inserisci_Prenotazione(CAMERA_STRUCT, short *, short *, char *);
short Elimina_Prenotazione(CAMERA_STRUCT, short *, short *, char *);
void Visualizza_Prenotati(CAMERA_STRUCT , short);

int main()
{
    /* Definiamo 9 camere vuote con relativi collegamenti iniziali */
    CAMERA_STRUCT Camera[9]={{ "",1,1}, {"",2,2}, {"",3,3}, {"",4,4}, {"",5,5}, {"",6,6}, {"",7,7},
                           {"",8,8}, {"",9,-1}};
    short p_libera=0, p_dati=-1; /* Definiamo la testa della ListaLibera e lista piena.
                                   p_libera punta alla testa della ListaLibera, ossia la prima camera
                                   libera.
                                   p_dati punta alla testa della ListaPiena, ossia la prima camera
                                   Piena.
                                   Inizialmente non ho camere piene, solo tutte vuote. La prima camera
                                   vuota
                                   e' all'indice 0.*/
    char Scelta, Nome_Cliente[16];
    short Esito;
    do
    {
        printf("----- GESTIONE CAMERE ALBERGO -----*\n");
        printf("*[1] - Inserisci Prenotazione      *\n");
        printf("*[2] - Elimina Prenotazione       *\n");
        printf("*[3] - Visualizza clienti prenotati*\n");
        printf("*[4] - Esci                      *\n");
        printf("*-----*\n");
        printf("\nScelta: ");
        scanf("%c", &Scelta);fflush(stdin);
        switch(Scelta)
        {
            case '1': if(p_libera!=-1)
            {
                printf("Inserisci cliente da inserire\n");gets(Nome_Cliente);
                Inserisci_Prenotazione(Camera, &p_libera, &p_dati, Nome_Cliente);
            }
        }
    } while(Scelta != '4');
```

```

        }
        else printf("Non posso aggiungere: Tutte le camere sono piene!! \n");
        break;
    case '2': if(p_dati!=-1)
    {
        printf("Inserisci cliente da Eliminare\n"); gets(Nome_Cliente);
        Esito = Elimina_Prenotazione(Camera, &p_libera, &p_dati, Nome_Cliente);
        (Esito!=1)?printf("Non ho trovato nessun cliente\n") : ("Cliente
                           Eliminato\n");
    }
    else printf("Non posso eliminare: tutte le camere sono vuote!!\n");
    break;
    case '3': if(p_dati!=-1)
    {
        printf("Le camere prenotate dai clienti [Numero stanza, Nome
                           Cliente]:\n");
        Visualizza_Prenotati(Camera, p_dati);
    }
    else printf("\nNessun cliente\n");
    break;
}
printf("\n");
fflush(stdin);
}while(Scelta!='4');

return 0;
}
/*
 * Inserisci_Prenotazione _____
Viene inserito il cliente in ListaDati.
Quando si deve inserire un cliente:
- salviamo la testa di ListaDati
- facciamo puntare p_dati alla nuova testa presa da p_libera
- inseriamo informazioni al nuovo nodo
- p_libera punterà al suo successivo
- facciamo il link del nuovo nodo aggiunto a p_dati iniziale
*/
void Inserisci_Prenotazione(CAMERA_STRUCT Camera[], short *p_libera, short *p_dati, char
*Nome_Cliente)
{
    short app_p_dati; /* Inizialmente conterrà la testa di ListaDati .
                        Quando avrò la nuova testa p_dati(rubato da p_libera), mi servirà
                        per agganciare la nuova testa a quella vecchia, cioè app_p_dati*/
    /* Salva la testa di ListaDati, che diventerà poi vecchia */
    app_p_dati = *p_dati;
    /* Aggiungi in testa a ListaDati (p_dati punterà a p_libera, precedentemente modificato */
    *p_dati = *p_libera; //p_dati punterà alla nuova testa di ListaLibera
    strcpy(Camera[*p_dati].Cliente, Nome_Cliente); //Inserisci cliente
    /* Eliminare in testa di ListaLibera*/
    *p_libera= Camera[*p_libera].Next; //p_libera punta al successivo
    /* Aggancia nuovo nodo a quello vecchio */
    Camera[*p_dati].Next = app_p_dati; //agganciala nuova testa a quella vecchia salvata
}
/*
 * Elimina_Prenotazione _____
Viene eliminato il cliente in ListaDati.
Si hanno 2 casi: eliminazione in testa e eliminazione in mezzo.
Se l'eliminazione è in testa, ossia l'elemento da eliminare è la
testa della ListaDati.
PRIMO CASO: - salvo l'indice della testa
              - Punta al successivo della testa
              - aggancia il nodo scartato da p_libera
              - egli di vento la nuova p_libera
SECONDO CASO: - Salviamo l'indice della camera successiva a p_dati
              - Scavalca app_p_dati, agganciando p_dati al successivo di app_p_dati
              - Aggancia il nodo eliminato alla testa di ListaLibera
              - Fai diventare egli la nuova testa di lista libera
*/
short Elimina_Prenotazione(CAMERA_STRUCT Camera[], short *p_libera, short *p_dati, char
*Nome_Cliente)
{
    short app_p_dati; /* Contiene la testa di ListaPiena. Quando avrò la nuova testa,
                       essendo diventata la precedente, mi servirà per agganciare la
                       nuova testa a quella vecchia */
    short p_dati1=*p_dati; //utilizzo p_dati1 per la visita e non perdere p_dati
    short p_liberal=*p_libera;
    short Flag=0;

```

```

/* PRIMO CASO: Provo in prima testa della ListaDati*/
if(strcmp(Camera[*p_dati].Cliente, Nome_Cliente)==0)
{
    /* Salva testa ListaDati*/
    app_p_dati=*p_dati;
    /* Punta al successivo della testa */
    *p_dati=Camera[*p_dati].Next;
    /* Agganciati alla testa di ListaLibera */
    Camera[app_p_dati].Next=*p_libera;
    /* Diventa la nuova testa di ListaLibera*/
    *p_libera=app_p_dati;
    return 1;//ho trovato e subito esco, FLAG
}

/* SECONDO CASO: In mezzo o in coda*/
else
{
    /* Visito la lista per cercare cliente (Scavalco la testa p_Dati)*/
    while(Camera[p_dati].Next!=-1) //
    {
        /* Se il cliente e' quello della camera successiva */
        // RICORDA, app_p_dati e pdati sono indici!
        if(strcmp(Camera[ Camera[p_dati].Next ].Cliente, Nome_Cliente)==0)
        {
            /* Salviamo l'indice della camera successiva a p_dati*/
            app_p_dati = Camera[p_dati].Next;
            /* Scavalca app_p_dati, agganciando p_dati al successivo di app_p_dati */
            Camera[p_dati].Next= Camera[app_p_dati].Next; //Camera[Camera[p_dati].Next];
            /* Aggancia il nodo eliminato alla testa di ListaLibera */
            Camera[app_p_dati].Next=*p_libera;
            /* Fai diventare egli la nuova testa di lista libera*/
            *p_libera=app_p_dati;
            return 1;//ho trovato e subito esco, FLAG
        }
        p_dati=Camera[p_dati].Next; //Passa al nodo successivo
    }
}
return 0; //Arrivato qui, non ho trovato niente
}

```

```
void Visualizza_Prenotati(CAMERA_STRUCT Camera[], short p_dati)
```

```
{
    printf("\n");
    printf("\t");
    while(p_dati!=-1)
    {
        printf("[%hd, %s] ", Camera[p_dati].Id_Camera, Camera[p_dati].Cliente);
        p_dati=Camera[p_dati].Next;
    }
}
```

OUTPUT:

```
***** GESTIONE CAMERE ALBERGO *****
*[1] - Inserisci Prenotazione      *
*[2] - Elimina Prenotazione       *
*[3] - Visualizza clienti prenotati*
*[4] - Esci                         *
*****
```

Scelta: 1  
Inserisci cliente da inserire  
A

Scelta: 1  
Inserisci cliente da inserire  
B

Scelta: 3  
Le camere prenotate dai clienti  
[Numero stanza, Nome Cliente]:  
[2, B]  
[1, A]

Scelta: 2  
Inserisci cliente da Eliminare  
B

Scelta: 3  
Le camere prenotate dai clienti  
Numero stanza, Nome Cliente:  
[1, A]

## ESERCIZIO 41 [LIV.1]

```
/*
[liv1]Realizzare la gestione di una lista lineare mediante menù: visualizzazione mediante visita,
inserimento in testa,inserimento in mezzo, e eliminazione in testa, eliminazione in mezzo.
Implementare la lista lineare con una struttura autoriferente dinamica.
*/
#include <stdio.h>
#include <stdlib.h>

/* Semplice struct di dati */
typedef struct
{
    char nome[20];
    short eta;
}INFO_FIELD;

/* Dichiarazione della struttura del singolo nodo "struct PERSONA" che contiene i due campi:
 - informazione che si chiama 'info' ed e' di tipo INFO_FIELD dichiarato tramite una typedef
 - p_next ed e' un puntatore autoriferente */
struct PERSONA
{
    INFO_FIELD info;
    struct PERSONA *p_next;
};

//Prototipi delle function
struct PERSONA *crea_lista();
void insl_testa (INFO_FIELD Dati, struct PERSONA **head);
void insl_nodo (INFO_FIELD Dati, struct PERSONA **punt);
void visualizza (struct PERSONA *head);
void elim_nodo (struct PERSONA *punt);
void elim_testa (struct PERSONA **head);
int cerca_nodo (struct PERSONA **punt, char key[]); //ricerca per nome

int main()
{
    struct PERSONA *head, *punt; /* Head=Punta solo la testa, Punt=un nodo qualunque */
    INFO_FIELD New_Dato;

    char Scelta, testo_key[20];
    int Esito;

    /* Crea lista. Head inizialmente punta a Head */
    head = crea_lista();
    do
    {
        printf("----- ESEMPIO DI GESTIONE DI UNA LISTA LIENARE -----*\n");
        printf("[1] - Inserisci in testa alla lista *\n");
        printf("[2] - Inserisci un elemento nel nodo successivo *\n");
        printf("[3] - Elimina un elemento dalla testa *\n");
        printf("[4] - Elimina un elemento dal nodo successivo *\n");
        printf("[5] - Visualizza la lista *\n");
        printf("[6] - Esci dal programma *\n");
        printf("*-----*\n");
        printf("\nScelta: ");
        scanf("%c", &Scelta);fflush(stdin);printf("\n");
        fflush(stdin);
        switch(Scelta)
        {
            case '1': printf("---- INSERIMENTO IN TESTA ---*\n");
                        printf("Inserisci nome: ");gets(New_Dato.nome);
                        printf("Inserisci eta': ");scanf("%hd", &New_Dato.eta);
                        insl_testa (New_Dato, &head); //Inserisci in testa
                        break;

            case '2': if(head!=NULL) //Ho almeno un elemento nella lista
            {
                printf("---- INSERIMENTO NEL NODO SUCCESSIVO ---*\n");
                printf("Inserisci nome del nodo a cui agganciare successivamente il nuovo
                      nodo: \n");
                gets(testo_key);
                punt=head; /* Punt punterà all'indirizzo di base inizialmente
                           per non perdere head*/
                Esito = cerca_nodo(&punt, testo_key); //cerca nodo
            }
        }
    } while(Scelta != '6');
}
```

```

        if(Esito)//se nodo e' stato trovato, aggiungi
    {
        printf("INSERISCI DATI DEL NUOVO NODO\n");
        printf("Inserire nome: ");gets(New_Dato.nome);
        printf("Inserire eta': ");scanf("%hd", &New_Dato.eta);
        insl_nodo(&New_Dato, &punt); /* Inserisci dopo nodo trovato.
                                         Passiamo per indirizzo perche' punt
                                         dovrà puntere al nuovo nodo */
    }
    else printf("Il nodo non e' stato trovato! \n");
}
else printf("Lista Vuota!!\n");

break;

case '3': if(head!=NULL) //Ho almeno un elemento nella lista
{
    printf("---- ELIMINAZIONE IN TESTA ---*\n");
    elim_testa(&head); //Chiamata alla function eliminazione_in_testa
}
else printf("Lista Vuota!!\n");
break;

case '4': if(head!=NULL) //Ho almeno un elemento nella lista
{
    printf("---- ELIMINAZIONE NEL NODO SUCCESSIVO ---*\n");
    printf("Inserisci nome dell nodo dove verra' eliminato il successivo:\n");
    gets(testo_key);
    punt=head; /* Punt puntera' all'indirizzo di base inizialmente
                  per non perdere head*/
    Esito = cerca_nodo(&punt, testo_key); //cerca nodo
    /* se nodo e' stato trovato e non e' l'ultimo, elimina
       successivo.
       RICORDA: l'ultimo nodo punta a null: Quindi non posso eliminare
       l'elemento successivo puntato dall'ultimo nodo (sarebbe NULL) */
    if(Esito && punt->p_next)
    {
        /* Se il nodo trovato non e' la testa*/
        elim_nodo(punt);
    }
    else if(punt->p_next)printf("Il successivo del nodo cercato e' NULL!\n");
    else printf("Il nodo non e' stato trovato! \n");
}
else printf("Lista Vuota!!\n");
break;

case '5': if(head!=NULL) //Ho almeno un elemento nella lista
{
    printf("---- VISUALIZZA LISTA ---*\n\n");
    visualizza(head);
}
else printf("Lista vuota!!\n");
break;
}

printf("\n");
fflush(stdin);
}while(Scelta!='6');
return 0;
}

/*
Cerca Nodo: In base ad un nome dato, tale funzione ricerca la lista contenente il medesimo
nome. La ricerca avviene tramite una visita e la restituzione dei parametri e' per indirizzo.
*/
int cerca_nodo (struct PERSONA **punt, char key[])
{
/* visita della lista*/
while(*punt!=NULL)
{ /* Se il nome del nodo e' uguale alla stringa cercata*/
    if(strcmp( (*punt)->info.nome, key)==0) // (**punt).info.testa.
        return 1; //Ok, nodo trovato; *punt puntera' al nodo trovato
    *punt=(*punt)->p_next;
}
return 0; //se non trovi niente, segnala errore
}

```

```

/*
 Inserisci in testa: Date le informazioni nuove, viene allocato un nuovo nodo e dopo
 aver inserito le info, viene agganciato al primo nodo (indicato da Head) e per poi far puntare
 head a tale nodo nuovo .
 HEAD e' puntatore doppio, perche' dovremo modificare l'indirizzo a cui punta.
*/
void insl_testa (INFO_FIELD Dati, struct PERSONA **head)
{
    struct PERSONA *ptr; //Puntatore al NUOVO NODO
    /* Crea nodo e inserisci dati */
    ptr=(struct PERSONA *)calloc(1,sizeof (struct PERSONA)); //Richiediamo di allocare un nodo di
                                                               grandeza persona
    ptr->info=Dati; //Inserisci dati; (*ptr).info
    /* Aggancia il new nodo al nodo in testa (me lo dice head dove sta)*/
    ptr->p_next = *head; //Head contiene l'indirizzo del nodo in testa
    /* Aggancia testa al new nodo */
    *head=ptr;
}

/* Inserisci in mezzo: Date le informazioni nuove, viene allocato un nuovo nodo e verra' aggiunto
 dopo il nodo puntato da *punt. Quindi il nuovo nodo punterà al prossimo di punt e punt punterà
 proprio al nuovo nodo.
 Dopo aver aggiunto, PUNT punterà al nuovo nodo.
*/
void insl_nodo(INFO_FIELD Dati, struct PERSONA **punt)
{
    struct PERSONA *ptr; //Puntatore al NUOVO NODO
    /* Crea nodo e inserisci dati */
    ptr=(struct PERSONA *)calloc(1,sizeof (struct PERSONA)); //Richiediamo di allocare un nodo di
                                                               grandeza persona
    ptr->info=Dati; //Inserisci dati; (*ptr).info
    /* Aggancia il new nodo al successivo di punt */
    ptr->p_next= (*punt)->p_next;
    /* Aggancia il nodo considerato a punt al nuovo nodo */
    (*punt)->p_next = ptr;
    /* Punt ora lo faremo puntare al nuovo nodo, perche' sara' il corrente (facoltativo) */
    *punt=ptr;
}

/*
 Elimina in testa: Elimina il nodo puntato da testa. Head successivamente punterà al prossimo del
 nodo eliminato.
*/
void elim_testa(struct PERSONA **head)
{
    struct PERSONA *Libera; Libera=*head; //Libero dalla memoria successivamente quel nodo
    /* Istruzione fondamentale*/
    *head = (*head)->p_next;

    free(Libera); //Libera dalla memoria quel nodo eliminato
}

/*
 Elimina in mezzo: Elimina il nodo successivo cercato (puntato da punt). Per eliminare,
 basta far scavalcare a tale nodo il successivo a quello da eliminare.
*/
void elim_nodo(struct PERSONA *punt)
{
    struct PERSONA *Libera; Libera=(punt)->p_next; //Libero dalla memoria successivamente quel nodo

    /* Scavalca facendo agganciare al nodo puntato da punt, al nodo successore del nodo da eliminare.
       ((*punt)->Next)->p_next indica che dopo essere andato al nodo da eliminare, ricavo l'indirizzo
       del suo successore (indirizzo del nodo da eliminare->Next) e quindi lo inseriro' al nodo punt
    */
    punt->p_next= (punt->p_next)->p_next;

    free(Libera); //Libera dalla memoria quel nodo eliminato
}

/*
 Crea lista: Restituisce un punto iniziale della testa.
*/
struct PERSONA *crea_lista()
{
    struct PERSONA *head;
    head =NULL;
    return head;
}

```

```

}

/*
Visualizza: semplicemente effettua la visita. Head e' un puntatore passato per valore
perche' la modifica del valore a cui punta, non mi interessa nel main e inoltre,
passando da head a head->next, nel main non perdero' head iniziale.
*/
void visualizza(struct PERSONA *head)
{
    /* Visita della lista */
    /* Facciamo un do while perche' head==NULL la prima volta, gia' e' stato controllato */
    do
    {
        printf(" Nome: %2s % \n Eta : %2hd\n\n", head->info.nome, head->info.eta);
        head=head->p_next;
    }while(head!=NULL);
}

```

#### OUTPUT:

```

*----- ESEMPIO DI GESTIONE DI UNA LISTA LIENARE -----*
*[1] - Inserisci in testa alla lista
*[2] - Inserisci un elemento nel nodo successivo
*[3] - Elimina un elemento dalla testa
*[4] - Elimina un elemento dal nodo successivo
*[5] - Visualizza la lista
*[6] - Esci dal programma
*-----*
Scelta: 1

---- INSERIMENTO IN TESTA ----*
Inserisci nome: A
Inserisci eta': 18

Scelta: 2

---- INSERIMENTO NEL NODO SUCCESSIVO ----*
Inserisci nome del nodo a cui agganciare successivamente il nuovo nodo:
A
INSERISCI DATI DEL NUOVO NODO
Inserire nome: B
Inserire eta': 20

Scelta: 5

---- VISUALIZZA LISTA ----*
Nome: A
Eta : 18

Nome: B
Eta : 20

Scelta: 4

---- ELIMINAZIONE NEL NODO SUCCESSIVO ----*
Inserisci nome dell nodo dove verrà eliminato il successivo:
A

Scelta: 5

---- VISUALIZZA LISTA ----*
Nome: A
Eta : 18

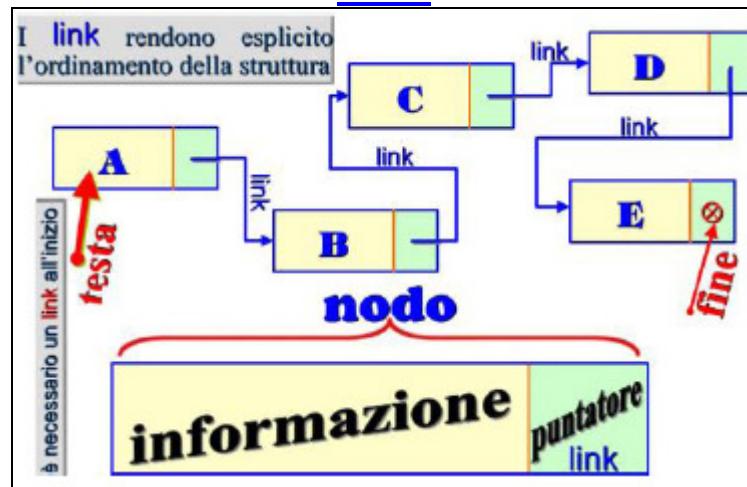
Scelta: 3

---- ELIMINAZIONE IN TESTA ----*
Scelta: 5

Lista vuota!!

```

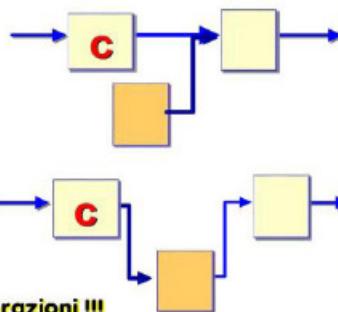
## LISTE



## OPERAZIONI SU LISTE

### INSERIMENTO DI UN ELEMENTO DOPO QUELLO CORRENTE

1) inserire il link che va dall'elemento nuovo al successore.

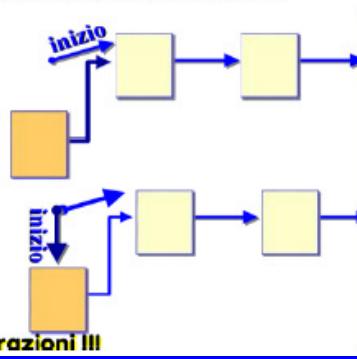


2) modificare il link dell'elemento corrente affinché punti al nuovo elemento.

Attenzione all'ordine delle operazioni !!!

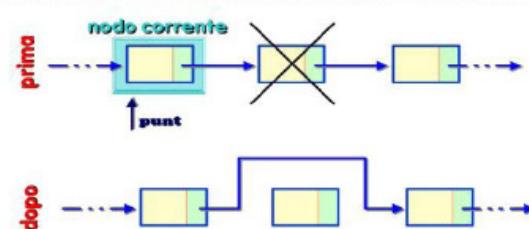
### INSERIMENTO DI UN ELEMENTO IN TESTA

1) inserire il link che va dall'elemento nuovo alla testa della struttura.

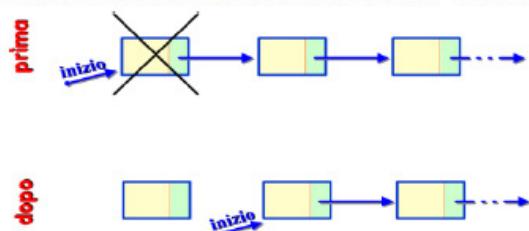


Attenzione all'ordine delle operazioni !!!

### ELIMINAZIONE ELEMENTO CORRENTE



### ELIMINAZIONE ELEMENTO IN TESTA



## ESERCIZIO 42 [LIV.2]

```

/*[LIV. 2] A partire dalla versione ricorsiva di una lista in C, scrivere la versione iterativa. */
#include <stdio.h>
#include <stdlib.h>

typedef char DATA; //definiamo di un nuovo tipo DATA che e' un char
/* Definiamo i nodi */
struct linked_list
{
    DATA d; //informazioni di tipo DATA
    struct linked_list *next; // puntatore autoriferente
};
/* definiamo struct linked_list element*/
typedef struct linked_list ELEMENT;
/* definizione di un nuovo tipo puntatore a element, quindi puntatore ad una struttura
linked_list. Apposta di fare struct link *testa, faccio LINK testa.
*/
typedef ELEMENT *LINK;
/* Prototipi function */
LINK array_to_list(DATA s[]);
LINK array_to_list_iterativa(DATA s[]);

int main()
{
    DATA c_arr[]="La vita e' regolata dal karma: tutto alla fine torna!"; //array da inserire
    /* dichiarazione dei 2 puntatori alla struttura ELEMENT */
    LINK head_list, p_list;
    printf("Lista Costruita: \n");
    /* COSTRUisci LISTA */
    /*costruzione della lista che avviene mediante la chiamata che definisce il puntatore
    head_list uguale al valore che ritorna la funzione (un puntatore)*/
    head_list = array_to_list_iterativa(c_arr);
    /* VISITA DELLA LISTA */
    p_list=head_list;
    while (p_list != NULL)
    {
        putchar(p_list->d);
        p_list = p_list->next;
    }
    puts("");
    return 0;
}

/*
Versione Iterativa: L'algoritmo ha una complessita' o(n), poiche' inserisce sempre in testa,
evitando di effettuare la visita.
COME FUNZIONA: e' molto semplice, basta prendere i caratteri a partire da len-1 e salvo
il carattere nei nodi che creo. Dato che e' un inserimento in testa, inserito l'ultimo carattere
al primo nodo, alla fine dei cicli mi ritrovero' l'ultimo nodo che ho inserito inizialmente,
mentre l'elemento della testa della lista conterra' l'ultimo elemento messo dal ciclo: il primo
carattere.
*/
LINK array_to_list_iterativa(DATA s[])
{
    LINK head, Nodo; //RICORDA, e' come se facessi struct LISTA *head
    int len=strlen(s);

    head=NULL; //Puntera' inizialmente a NULL
    /* Facciamo un ciclo per len-1 volte */
    while(len>0)
    {
        /* Crea Nodo e inserisci il carattere */
        Nodo=(ELEMENT *)calloc(1,sizeof (ELEMENT)); //Richiediamo di allocare un nodo di grandezza
        persona
        Nodo->d=s[len-1]; //Inserisci dati; (*ptr).info
        /* Aggancia il nodo al puntato da head */
        Nodo->next=head;
        /* head punterà al nuovo nodo*/
        head=Nodo;
        /* Considero dall'ultimo carattere al primo, poiché inserisco per testa*/
        len--;
    }
    return head;
}
/*

```

```
    Versione Ricorsiva
*/
LINK array_to_list(DATA s[])
{
    LINK head;
    if (s[0]=='\0') return NULL; //caso base. Il fondo della lista che punta a null
    else
    {
        head = malloc(sizeof(ELEMENT)); //alloca nodo
        head->d = s[0]; //inserisci valore a S[0]
        head->next = array_to_list(s+1); /* Autoattivazione: s+1 per passare all'indirizzo
                                         successivo. Ritornato NULL, si combinano a ritroso i
                                         nodi, agganciandosi ad ogni ritorno della chiamata */
        return head;
    }
}
```

**OUTPUT:**

**Lista Costruita:**  
La vita e' regolata dal karma: tutto alla fine torna!

## ESERCIZIO 43 [LIV.2]

```
/**
[liv.2] Scrivere una function C per costruire una lista ordinata in ordine alfabetico
a partire da un elenco di nomi in ordine casuale, come nel seguente.
**/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define len 11

/* Definiamo i nodi */
struct LISTA
{
    char Nome[16]; //informazioni di tipo DATA
    struct LISTA *p_next; // puntatore autoriferente
};

/* PROTOTIPI */
void ins_in_ordine(struct LISTA **head, char *elenco[]); //Indirizzo base dell'array
void insl_testa (char *Testo, struct LISTA **head);
void insl_nodo (char *Testo, struct LISTA **punt);
void visualizza(struct LISTA *head);
struct LISTA *crea_lista();

#include "header.h"

int main()
{
    //ARRAY FRASTAGLIATO = array di stringhe costanti, ossia indirizzi base delle stringhe
    char *elenco[]={
        "Anna", "Mario", "Giuseppe", "Angela", "Valeria", "Fabrizio",
        "Marianna", "Giovanni", "Patrizia", "Valentina", "Sara"
    };

    /* Definiamo i puntatori alla lista dell'elenco*/
    struct LISTA *head;
    /* Inizializzo la lista, ponendo head a NULL */
    head = crea_lista();
    printf("---- LISTA ORDINATA, A PARTIRE DA UN'ARRAY FRASTAGLIATO ---");
    /* Inserisci in ordine */
    ins_in_ordine(&head, elenco); //Indirizzo base dell'array
    /* Stampa Lista con visita */
    visualizza(head);
    printf("\n");
    free(head);
    return 0;
}

#include "header.h"

/*
ins_in_ordine: - Aggiungiamo il primo nome dell'elenco in testa;
               - Entra nel ciclo in cui scorrono gli elenchi(dal secondo in poi);
               - posizioniamo punt alla testa per visitare la lista;
               - Controlla in testa se elenco[i]<(*head)->Nome). Se si, aggiungi in testa
                 altrimenti ricordati che head sara' prec, passa al secondo nodo e visita la
                 lista;
               - durante la visita troviamo il nodo in cui e' presente il piu' piccolo:
                 dobbiamo quindi agganciarlo al nodo precedente (insl_nodo passiamo prec,
                 perche' inserisce dopo prec);
               - Se non trova il piu' piccolo, aggancia alla fine della lista;
*/
void ins_in_ordine(struct LISTA **head, char *elenco[]) //Indirizzo base dell'array
{
    struct LISTA *punt;
    struct LISTA *prec;
    short i=0, Esito;
```

```

/* INSERISCO PRIMO NODO */
/*__ La prima stringa in assoluto la metto in testa __*/
insl_testa(elenco[i], head); //head lo passo cosi perche' e' gia' l'indirizzo del puntatore
                             e sara' modificato
i++; //Considera la prossima stringa dell'elenco

/* CICLO ESTERNO: Ciclo per scorrere l'elenco a partire dal secondo.
   Per ogni stringa dell'elenco, controllo in quale ordine della lista che si sta costruendo
   deve
   andare. APPENA TROVATO UN NODO dove inserire, il ciclo che fa la visita (INTERNO)
   della lista si conclude poiche' sara' il piu' piccolo rispetto ai suoi successivi.
*/
for(i=1;i<len;i++) //elenco
{
    punt=*head; // agganciati in testa e non perdere head
    /* CONTROLLO IN TESTA DELLA LISTA */
    /* Dimmi l'esito tra elenco[i] e la stringa in testa. Se essa e' minore, inserisci in
       testa*/
    Esito = strcmp(elenco[i], (*head->Nome)); //punt->Nome
    if(Esito== -1)
        insl_testa(elenco[i], head); //passo l'indirizzo del puntatore. Head cambiera'
    /* CONTROLLA IN MEZZO ALLA LISTA, OVVERO FAI UNA VISITA */
    else
    {
        prec = *head; /* salvo il precedente (mi serve per l'inserimento in mezzo) */
        punt=punt->p_next; //avanzo punt perche' gia' ho considerato con l'if il primo nodo
        /* Se punt->next puntasse a NULL, significa che verra' messo in coda.
           (Caso in cui abbiamo solo un nodo e vogliamo aggiungere il secondo) */
        if(punt==NULL) insl_nodo(elenco[i], &prec); /* prec indicherebbe l'ultimo nodo e punt
                                                       il successivo (NULL)*/
        while(punt!=NULL && Esito!= -1) //Esito=-1->HA INSERITO
        {
            Esito = strcmp(elenco[i], punt->Nome); //controlla esito
            /* Se e' piu' piccolo del nodo corrente, aggiungi dopo prec */
            if(Esito== -1)
                insl_nodo(elenco[i], &prec);
            /* Oppure se e' l'ultimo nodo, aggiungi alla fine della lista */
            else if(punt->p_next==NULL) //se il prossimo nodo a punt e' NULL, inserisco in
                                         coda
                insl_nodo(elenco[i], &punt);
            /* Vado avanti (se ha inserito nella lista, sara' controllato da esito e
               uscira') */
            prec=punt; //salvo il precedente
            punt=punt->p_next; //prossimo nodo
        }
    }
}
/*
   Crea lista: Restituisce un punto iniziale della testa.
*/
struct LISTA *crea_lista()
{
    struct LISTA *head;
    head =NULL;
    return head;
}
/*
   Visualizza: semplicemente effettua la visita. Head e' un puntatore passato per valore
   perche' la modifica del valore a cui punta, non mi interessa nel main e inoltre,
   passando da head a head->next, nel main non perdero' head iniziale.
*/
void visualizza(struct LISTA *head)
{
    /* Visita della lista */
    /* Facciamo un do while perche' head==NULL la prima volta, gia' e' stato controllato */
    do
    {
        printf(" Nome: %s \n", head->Nome);
        head=head->p_next;
    }while(head!=NULL);
}

```

```

/*
    Inserisci in testa: Date le informazioni nuove, viene allocato un nuovo nodo e dopo
    aver inserito le info, viene agganciato al primo nodo (indicato da Head) e per poi far
    puntare
        head a tale nodo nuovo .
    HEAD e' puntatore doppio, perche' dovremo modificare l'indirizzo a cui punta.
*/
void insl_testa (char *Testo, struct LISTA **head)
{
    struct LISTA *ptr; //Puntatore al NUOVO NODO
    /* Crea nodo e inserisci dati */
    ptr=(struct LISTA *)calloc(1,sizeof (struct LISTA)); //Richiediamo di allocare un nodo di
grandezza LISTA
    strcpy(ptr->Nome, Testo); //Inserisci dati; (*ptr).info
    /* Aggancia il new nodo al nodo in testa (me lo dice head dove sta)*/
    ptr->p_next = *head; //Head contiene l'indirizzo del nodo in testa
    /* Aggancia testa al new nodo */
    *head=ptr;
}

/* Inserisci in mezzo: Date le informazioni nuove, viene allocato un nuovo nodo e verrà'
aggiunto
    dopo il nodo puntato da *punt. Quindi il nuovo nodo punterà al prossimo di punt e punt
punterà'
    proprio al nuovo nodo.
    Dopo aver aggiunto, PUNT punterà al nuovo nodo.
*/
void insl_nodo (char *Testo, struct LISTA **punt)
{
    struct LISTA *ptr; //Puntatore al NUOVO NODO
    /* Crea nodo e inserisci dati */
    ptr=(struct LISTA *)calloc(1,sizeof (struct LISTA)); //Richiediamo di allocare un nodo di
grandezza LISTA
    strcpy(ptr->Nome, Testo); //Inserisci dati; (*ptr).info
    /* Aggancia il new nodo al successivo di punt */
    ptr->p_next= (*punt)->p_next;
    /* Aggancia il nodo considerato a punt al nuovo nodo */
    (*punt)->p_next = ptr;
    /* Punt ora lo faremo puntare al nuovo nodo, perche' sarà il corrente (facoltativo) */
    *punt=ptr;
}

```

#### OUTPUT:

```

/** ELENCO INIZIALE**/
char *elenco[]={"Anna",
                  "Mario",
                  "Giuseppe",
                  "Angela",
                  "Valeria",
                  "Fabrizio",
                  "Marianna",
                  "Giovanni",
                  "Patrizia",
                  "Valentina",
                  "Sara"};

```

**---- LISTA ORDINATA, A PARTIRE DA UN'ARRAY FRASTAGLIATO ----\***

```

Nome: Angela
Nome: Anna
Nome: Fabrizio
Nome: Giovanni
Nome: Giuseppe
Nome: Marianna
Nome: Mario
Nome: Patrizia
Nome: Sara
Nome: Valentina
Nome: Valeria

```

## ESERCIZIO 44A [LIV.1]

```
/*
[LIV1] Realizzare in C le funzioni per la gestione della struttura dati PILA mediante lista lineare
dinamica e generica con nodo sentinella.
**/

/* Il nodo sentinella serve per evitare di avere due function per l'inserimento e due function
per l'eliminazione. Infatti è possibile avere un'unica sequenza di operazioni sia per gli
inserimenti/eliminazioni sulla testa che per quelli fatti sul nodo corrente.
Occorre semplicemente aggiugere un nodo fittizio che funga da testa ma che punti alla testa
reale della lista: un nodo sentinella.
Ogni modifica sulla testa comporterà in questo modo la sola alterazione del campo puntatore del
nodo sentinella.
*/

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

/* dichiarazione del tipo INFO_FIELD, serve per le informazioni del nodo */
typedef struct{
    char nome[20];
}INFO_FIELD;

/* PROTOTIPI (Ricorda: se vogliamo modificare l'indirizzo a cui punta) */
void *creaLista_sentinella(); //crea una lista vuota
void visualizza(void *p_punt); //stampa la lista ad ogni operazione sulle liste
void insl_nodo(short len_info, INFO_FIELD *p_dato, void *p_punt); //inserisce in testa un nodo
void elim_nodo(void *p_punt); //elimina la testa della pila

int main()
{
    /* dichiarazione della lista (nel main perche' usiamo delle function standardizzate) */
    struct PERSONA
    {
        INFO_FIELD info;
        struct PERSONA *p_next; //puntatore autoriferente
    };

    struct PERSONA *head; //Head e' propriola testa sui cui lavoro
    short scelta,len_info;

    INFO_FIELD nuovodato; //campo informazioni che verrà inserito nel nuovo nodo da creare
    len_info = sizeof(INFO_FIELD);

    /*----- INSERISCI NODO SENTINELLA -----*/
    /* viene inserito in testa come nodo vuoto.
       Quando inserisco o elimino, lo farò da questo (funzionerà come ins_testa)
       Head, punta a questo nodo sentinella.*/
    head = (struct PERSONA *) creaLista_sentinella(); //verrà restituito un *void, quindi
                                                       effettuiamo cast
    /*-----*/
    do{
        puts("----- PILA -----*\n\n");
        /* stampa se dopo il nodo sentinella c'e' qualcosa */
        /* Inizia a stampare dal nodo successivo sentinella */
        if(head->p_next) visualizza((void *) (head)->p_next); //stampa la lista dopo ogni operazione
        puts("[1] Inserisci sulla pila *");
        puts("[2] Elimina dalla pila *");
        puts("[3] Esci *");
        fflush(stdin);
        scanf("%hd",&scelta);
        switch(scelta)
        {
            case 1: puts("\nInserisci nome: "); fflush(stdin); gets(nuovodato.nome);
                      /* Inserisce dopo nodo sentinella, senza utilizzare "insl_testa" */
                      insl_nodo(len_info,&nuovodato,(void *)head);
                      break;

            case 2: if(head->p_next != NULL)/* Se dopo il nodo sentinella c'e' qualcosa */
                      /* Elimina dopo nodo sentinella, senza utilizzare "elim_testa" */
                      elim_nodo((void *)head);
                      else puts("Impossibile eliminare, la pila e' vuota");
                      break;
        }
    }
}
```

```

} while (scelta!=3);

return 0;
}

/*
   Crea lista: Restituisce un punto iniziale della testa.
      La testa punterà al nodo sentinella.
*/
void *creaLista_sentinella()
{
    struct lista
    {
        INFO_FIELD info;
        struct lista *p_next;
    };
    struct lista *testa; /* il tipo non importa, nel main ho il cast */
    testa = (struct lista *) calloc(1,sizeof(struct lista));
    //testa->p_next=NULL; con calloc posso evitare tale istruzione
    return testa;
}

void visualizza(void *p_punt)
{
    /* definisce una struttura generica che condivide con il main solo il tipo infotipo e
       dichiara un puntatore alla struttura */
    struct PERSONA {
        INFO_FIELD info;
        struct PERSONA *p_next;
    } *ptr; /*puntatore alla struttura*/

    short i=0;
    ptr = ((struct PERSONA *)p_punt);

    //Visita di una lista
    while (ptr != NULL)
    {
        printf("[%hd]\t%s\n",i,ptr->info.nome);
        ptr = ptr->p_next;
        i++;
    }
    printf("\n");
}

/* Inserisci: Date le informazioni nuove, viene allocato un nuovo nodo e verrà aggiunto
   dopo il nodo puntato da *p_punt. Quindi il nuovo nodo punterà al prossimo di p_punt e p_punt punterà
   proprio al nuovo nodo.
   Dopo aver aggiunto, PUNT punterà al nuovo nodo.
   ps **p_punt potrebbe essere anche *p_punt perché non modifichia l'indirizzo a cui punta
*/
void insl_nodo(short len_info,INFO_FIELD *p_dato, void *p_punt)
{
    /* definisce una struttura generica che condivide con il main solo il tipo infotipo e
       dichiara un puntatore alla struttura */
    struct LISTA {
        INFO_FIELD info;
        struct LISTA *p_next;
    } *ptr; /*puntatore alla struttura*/
    /* alloco nodo e trasferisco informazioni copiando i byte */
    ptr = malloc (1, sizeof (struct LISTA));
    memcpy (ptr, p_dato, len_info); //copia len_info byte da *p_dato a *ptr, senza interessarsi della
struttura
    /* aggancia nodo nuovo al prossimo di p_punt*/
    ptr->p_next=((struct LISTA *)p_punt)->p_next; // colleghiamo il nuovo nodo con la vecchia testa
    /* nodo p_punt è agganciato a ptr*/
    ((struct LISTA *)p_punt)->p_next=ptr;
}

/*
   Elimina: Elimina il nodo successivo cercato (puntato da p_punt). Per eliminare,
   basta far scavalcare a tale nodo il successivo a quello da eliminare.
*/
void elim_nodo(void *p_punt)
{
    /* definisce una struttura generica che condivide con il main solo il tipo infotipo e
       dichiara un puntatore alla struttura */
    struct LISTA {
        INFO_FIELD info;
        struct LISTA *p_next;
    }
}

```

```

} *ptr; /*puntatore alla struttura*/
/* prendi nodo successivo al nodo sentinella (o puntato da p_punt) */
ptr = ((struct LISTA *)p_punt)->p_next;
/* Aggiorna nodo p_punt al successivo del successivo (guarda ptr chi e') */
((struct LISTA *) p_punt)->p_next = ptr->p_next;
/* libera ptr, ossia il successivo a sentinella eliminato */
free (ptr); // elimina la testa della lista
}

```

**OUTPUT:**

\*[1] Inserisci sulla pila \*
\*[2] Elimina dalla pila \*
\*[3] Esci \*

1

Inserisci nome:

A

\*----- PILA -----\*

[0] A

\*[1] Inserisci sulla pila \*
\*[2] Elimina dalla pila \*
\*[3] Esci \*

1

Inserisci nome:

B

\*----- PILA -----\*

[0] B
[1] A

\*[1] Inserisci sulla pila \*
\*[2] Elimina dalla pila \*
\*[3] Esci \*

1

Inserisci nome:

C

\*----- PILA -----\*

[0] C
[1] B
[2] A

\*[1] Inserisci sulla pila \*
\*[2] Elimina dalla pila \*
\*[3] Esci \*

2

\*----- PILA -----\*

[0] B
[1] A

\*[1] Inserisci sulla pila \*
\*[2] Elimina dalla pila \*
\*[3] Esci \*

2

\*----- PILA -----\*

[0] A

\*[1] Inserisci sulla pila \*
\*[2] Elimina dalla pila \*
\*[3] Esci \*

2

\*----- PILA -----\*

## ESERCIZIO 44B [LIV.1]

```
/**
[LIV1]Realizzare in C le funzioni per la gestione della struttura dati CODA mediante lista lineare
dinamica e generica con nodo sentinella.
**/

/*Il nodo sentinella serve per evitare di avere due function per l'inserimento e due function
per l'eliminazione. Infatti è possibile avere un'unica sequenza di operazioni sia per gli
inserimenti/eliminazioni sulla testa che per quelli fatti sul nodo corrente.
Occorre semplicemente aggiugere un nodo fittizio che funga da testa ma che punti alla testa
reale della lista: un nodo sentinella.
Ogni modifica sulla testa comporterà in questo modo la sola alterazione del campo puntatore del
nodo sentinella.
*/

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

/* dichiarazione del tipo INFO_FIELD, serve per le informazioni del nodo */
typedef struct{
    char nome[20];
}INFO_FIELD;

/* PROTOTIPI (Ricorda:se ** vogliamo modificare l'indirizzo a cui punta) */
void *creaLista_sentinella(); //crea una lista vuota
void visualizza(void *p_punt); //stampa la lista ad ogni operazione sulle liste
void insl_nodo(short len_info, INFO_FIELD *p_dato, void **p_punt); //inserisce in testa un nodo
void elim_nodo(void *p_punt); //elimina la testa della pila

int main()
{ /* dichiarazione della lista (nel main perche' usiamo delle function standardizzate) */
    struct PERSONA
    {
        INFO_FIELD info;
        struct PERSONA *p_next; //puntatore autoriferente
    };

    struct PERSONA *head, *fondo;
    short scelta, len_info;

    INFO_FIELD nuvodato; //campo informazioni che verrà inserito nel nuovo nodo da creare
    len_info = sizeof(INFO_FIELD);

    /*----- INSERISCI NODO SENTINELLA -----*/
    /*viene inserito in testa un nodo vuoto.
     Quando inserisco o elimino, lo farò da questo(funzionera' come ins_testa)
     Head, punta a questo nodo sentinella.*/
    head = (struct PERSONA *) creaLista_sentinella(); //verrà restituito un *void, quindi
                                                       effettuiamo cast
    fondo=head; //Inizialmente, fondo e head puntano al nodo sentinella
    /*-----*/
    do{
        puts("----- CODA -----*\n\n");
        /* stampa se dopo il nodo sentinella c'e' qualcosa */
        /* Inizia a stampare dal nodo successivo sentinella */
        if(head->p_next)visualizza((void *) (head)->p_next); //stampa la lista dopo ogni operazione
        puts("*[1] Inserisci sulla CODA *");
        puts("*[2] Elimina dalla CODA *");
        puts("*[3] Esci *");
        fflush(stdin);
        scanf("%hd",&scelta);
        switch(scelta)
        {
            /*Ripasso: CODA inserisce in fondo, ma estrae dalla testa */
            case 1: puts("\nInserisci nome: ");fflush(stdin);gets(nuvodato.nome);
                      /* Inserisce dopo nodo sentinella, senza utilizzare "insl_testa" */
                      /** RICAVO FONDO = PUNTA ALL'ULTIMO VALORE INSERITO**/
                      /* Quando inserisco un elemento dopo fondo, fondo si aggiorna all'indirizzo
                         del nuovo nodo: quindi quando rienserò a partire da quell'indirizzo
                         che indica il nodo alla fine, inserirà alla fine e fondo si riaggiornerà */
                      insl_nodo(len_info,&nuvodato,(void **)&fondo);
                      break;
            /* Se dopo il nodo sentinella c'e' qualcosa */
            case 2: if(head->p_next != NULL)
```

```

        elim_nodo((void *)head); /* head punta a SENTINELLA, quindi elimina elemento
                                   dopo sentinella (cosÌ funziona elim_nodo).
                                   Head punterà SEMPRE al nodo sentinella, quindi
                                   non necessito di modificarlo */
    else puts("Impossibile eliminare, la pila e' vuota");
    break;
}
}while(scelta!=3);

return 0;
}

/*
   Crea lista: Restituisce un punto iniziale della testa.
   La testa punterà al nodo sentinella.
*/
void *creaLista_sentinella()
{
    struct lista
    {
        INFO_FIELD info;
        struct lista *p_next;
    };
    struct lista *testa; /* il tipo non importa, nel main ho il cast */
    testa = (struct lista *) calloc(1,sizeof(struct lista));
    //testa->p_next=NULL; con calloc posso evitare tale istruzione
    return testa;
}

void visualizza(void *p_punt)
{
    /* definisce una struttura generica che condivide con il main solo il tipo infofield e
       dichiara un puntatore alla struttura */
    struct PERSONA {
        INFO_FIELD info;
        struct PERSONA *p_next;
    } *ptr; /*puntatore alla struttura*/

    /* Inserisci: Date le informazioni nuove, viene allocato un nuovo nodo e verrà aggiunto
       dopo il nodo puntato da *p_punt. Quindi il nuovo nodo punterà al prossimo di p_punt e p_punt punterà
       proprio al nuovo nodo.
       Dopo aver aggiunto, PUNT punterà al nuovo nodo.
       ps **p_punt potrebbe essere anche *p_punt perché non modifichia l'indirizzo a cui punta
    */
    void insl_nodo(short len_info,INFO_FIELD *p_dato, void **p_punt)
    {
        /* definisce una struttura generica che condivide con il main solo il tipo infofield e
           dichiara un puntatore alla struttura */
        struct LISTA
        {
            INFO_FIELD info;
            struct LISTA *p_next;
        } *ptr; /*puntatore alla struttura*/

        /* alloco nodo e trasferisco informazioni copiando i byte */
        ptr = calloc (1, sizeof (struct LISTA));
        memcpy (ptr, p_dato, len_info); //copia len_info byte da *p_dato a *ptr, senza interessarsi della
        struttura
        /* aggancia nodo nuovo al prossimo di p_punt*/
        ptr->p_next=((struct LISTA *)*p_punt)->p_next; // colleghiamo il nuovo nodo con la vecchia testa
        /* nodo p_punt è agganciato a ptr*/
        ((struct LISTA *)*p_punt)->p_next=ptr;
        /* Aggiorna il FONDO */
        *p_punt=ptr;
    }

    /*
       Elimina: Elimina il nodo successivo cercato (puntato da p_punt). Per eliminare,
       basta far scavalcare a tale nodo il successivo a quello da eliminare.
    */
    void elim_nodo(void *p_punt)
    {
        /* definisce una struttura generica che condivide con il main solo il tipo infofield e
           dichiara un puntatore alla struttura */
        struct LISTA {
            INFO_FIELD info;
            struct LISTA *p_next;
        }
    }
}

```

```

} *ptr; /*puntatore alla struttura*/
/* prendi nodo successivo al nodo sentinella (o puntato da p_punt) */
ptr = ((struct LISTA *)p_punt)->p_next;
/* Aggiorna nodo p_punt al successivo del successivo (guarda ptr chi e') */
((struct LISTA *) p_punt)->p_next = ptr->p_next;
/* libera ptr, ossia il successivo a sentinella eliminato */
free (ptr); // elimina la testa della lista
}

```

### OUTPUT:

\*[1] Inserisci sulla CODA \*  
\*[2] Elimina dalla CODA \*  
\*[3] Esci \*

1

Inserisci nome:

A

\*----- CODA -----\*

[0] A

\*[1] Inserisci sulla CODA \*  
\*[2] Elimina dalla CODA \*  
\*[3] Esci \*

1

Inserisci nome:

B

\*----- CODA -----\*

[0] A  
[1] B

\*[1] Inserisci sulla CODA \*  
\*[2] Elimina dalla CODA \*  
\*[3] Esci \*

1

Inserisci nome:

C

\*----- CODA -----\*

[0] A  
[1] B  
[2] C

\*[1] Inserisci sulla CODA \*  
\*[2] Elimina dalla CODA \*  
\*[3] Esci \*

2

\*----- CODA -----\*

[0] B  
[1] C  
\*[1] Inserisci sulla CODA \*  
\*[2] Elimina dalla CODA \*  
\*[3] Esci \*

2

\*----- CODA -----\*

[0] C

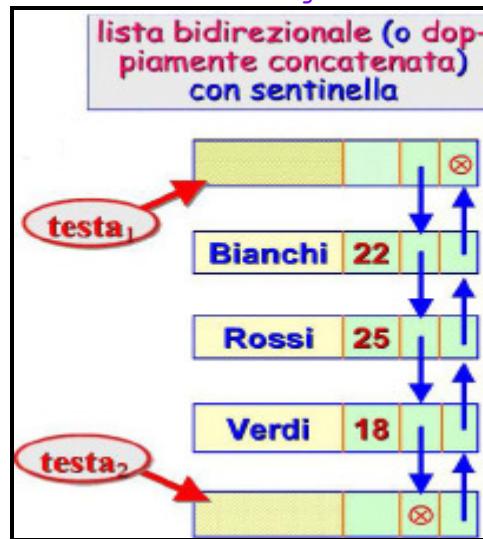
\*[1] Inserisci sulla CODA \*  
\*[2] Elimina dalla CODA \*  
\*[3] Esci \*

2

\*----- CODA -----\*

## ESERCIZIO 45A [LIV.2]

/\*\*  
 [liv.2] Realizzare in C le funzioni per la gestione, mediante menù, delle strutture dati LISTA BIDIREZIONALE mediante LISTA lineare dinamica e generica con nodo sentinella.



```
**/  

/* Il programma era stato realizzato con i puntatori a void, ma la calloc dava problemi in esecuzioni,  

   ma non in debug!*/  

#include <stdio.h>  

#include <stdlib.h>  

#include <string.h>  

/* dichiarazione del tipo INFO_FIELD, serve per le informazioni del nodo */  

typedef struct  

{  

    char nome[20];  

}INFO_FIELD;  

/* dichiarazione della LISTA */  

struct LISTA  

{  

    INFO_FIELD info;  

    struct LISTA *next;  

    struct LISTA *before;  

};  

/* PROTOTIPI (Ricorda:se ** vogliamo modificare l'indirizzo a cui punta) */  

struct LISTA *crealistica_sentinella(); //Prototipi delle Function  

void elimina_nodo(struct LISTA *prima, struct LISTA *dopo);  

void inserisci_nodo (struct LISTA *prima, struct LISTA *dopo, INFO_FIELD Dati);  

void stampaLISTA (struct LISTA *head1, struct LISTA *head2);  

void stampaLISTA2 (struct LISTA *head1, struct LISTA *head2);  

struct LISTA *cerca_nodo(struct LISTA *p_head1, struct LISTA *p_head2, char *buffer);  

int main()  

{  

    char scelta;  

    short esito;  

    struct LISTA *head1,*head2, *punt;  

    INFO_FIELD nuovodato; //campo informazioni che verrà inserito nel nuovo nodo da creare  

    /* _____ INSERISCI NODO SENTINELLA _____ */  

    /* viene inserito in testa un nodo vuoto.  

       Quando inserisco o elimino, lo farò da questo(funzionerà come ins_testa)  

       Per la lista BIDIREZIONALE, necessitiamo di 2 sentinelle*/  

    head1= (struct LISTA *)crealistica_sentinella(); //Creazione nodo sentinella di testa  

    head2= (struct LISTA *)crealistica_sentinella(); //Creazione nodo sentinella di coda  

    // agganciamo la sentinella 1 con la sentinella 2 e viceversa  

    head1->next=head2;  

    head2->before=head1;  

    /* ----- */  

    do{ printf("----- LISTA BIDIREZIONALE -----*\n");  

        printf("*[1] - Inserisci in testa della LISTA *\n");  

        printf("*[2] - Inserisci in mezzo alla LISTA *\n");  

        printf("*[3] - Elimina in testa della LISTA *\n");  

        printf("*[4] - Elimina in mezzo alla LISTA *\n");  

    }while(scelta!=5);  

    if(esito==1){  

        if(prima==NULL){  

            printf("La lista è vuota, non puoi inserire!\n");  

        }else{  

            if(prima==head1){  

                inserisci_nodo(prima,head1,nuovodato);  

            }else{  

                inserisci_nodo(head1,prima,nuovodato);  

            }  

        }  

    }else if(esito==2){  

        if(prima==NULL){  

            printf("La lista è vuota, non puoi inserire!\n");  

        }else{  

            if(prima==head1){  

                inserisci_nodo(prima,head1,nuovodato);  

            }else{  

                inserisci_nodo(head1,prima,nuovodato);  

            }  

        }  

    }else if(esito==3){  

        if(prima==NULL){  

            printf("La lista è vuota, non puoi eliminare!\n");  

        }else{  

            if(prima==head1){  

                elimina_nodo(prima,head1);  

            }else{  

                elimina_nodo(head1,prima);  

            }  

        }  

    }else if(esito==4){  

        if(prima==NULL){  

            printf("La lista è vuota, non puoi eliminare!\n");  

        }else{  

            if(prima==head1){  

                elimina_nodo(prima,head1);  

            }else{  

                elimina_nodo(head1,prima);  

            }  

        }  

    }  

}
```

```

printf("*[5] - Visualizza la LISTA dalla testa                                *\n");
printf("*[6] - Per visualizzare la LISTA dal fondo                         *\n");
printf("*[7] - Uscire dal programma                                         *\n");
printf("*-----*\n");
scanf("%d", &scelta);
switch(scelta)
{
    case 1: printf("Inserire il nome: ");fflush(stdin);gets(nuovodato.nome);
              /* Inserisci il nuovo nodo tra Sentinella e il successivo di sentinella */
              inserisci_nodo(head1,head1->next, nuovodato); //head1 punta a sentinella 1
              break;

    case 2: printf("Inserire nome del nodo a cui aggiungere il successivo: ");
              fflush(stdin); gets(nuovodato.nome);
              /* Cerca nodo a partire dal successivo di sentinella1 fino a sentinella
                 2 */
              punt = cerca_nodo(head1->next, head2,nuovodato.nome);
              printf("Inserire nome del nodo da agganciare: ");
              fflush(stdin); gets(nuovodato.nome);
              /*Inserisci tra il trovato punt e il successivo di punt */
              inserisci_nodo(punt, punt->next, nuovodato);
              break;

    case 3: if(head1->next==head2 && head2->before==head1) //Se la lista e' vuota
              puts("LISTA vuota, non posso eliminare! ");
              else
              {
                  /* Elimina funziona passando i nodi che hanno in mezzo il nodo da
                     eliminare.
                     Passando il nodo sentinella e il successivo del successivo
                     del nodo sentinella, passiamo i nodi in cui e' compresa la testa
                     della lista da eliminare.
                     [Sentinella]->[Nodo in testa da elim]->[Nodo succ] quindi
                     sentinella deve puntare al successivo del nodo da eliminare
                  */
                  elimina_nodo(head1, (head1->next)->next);
              }
              break;

    case 4:/* Controllo se la LISTA è vuota.Bastava una delle 2 espressioni, se
              vale una per forza vale l'altra */
              if(head1->next==head2 && head2->before==head1)
                  puts("LISTA vuota");
              else
              {
                  printf("Inserire il nome della persona da eliminare dalla LISTA:");
                  fflush(stdin); gets(nuovodato.nome);
                  punt = cerca_nodo(head1->next, head2, nuovodato.nome);
                  //Se il nodo non e' stato trovato, restituisce null
                  if(punt==NULL)puts("Errore, non e' stato eliminato nulla!");
                  else
                  {
                      /*Rispetto al nodo trovato, gli passiamo i "vicini" in mezzo */
                      elimina_nodo(punt->before, punt->next);
                      puts("Nodo trovato ed eliminato");
                  }
              }
              break;

    case 5: printf("---- VISUALIZZAZIONE LISTA DALLA TESTA ---*\n\n");
              /* Controllo se la LISTA è vuota.
                 Bastava una delle 2 espressioni, se vale una per forza vale l'altra */
              if(head1->next==head2 && head2->before==head1)
                  puts("LISTA vuota");
              else
                  stampaLISTA (head1->next,head2);//visualizzare dalla testa
              break;
    case 6: printf("---- VISUALIZZAZIONE LISTA DAL FONDO ---*\n\n");
              /* Controllo se la LISTA è vuota.
                 Bastava una delle 2 espressioni, se vale una per forza vale l'altra */
              if(head1->next==head2 && head2->before==head1)
                  puts("LISTA vuota");
              else
                  stampaLISTA2 (head1,head2->before);//visualizzare dal fondo
              break;
}
}while(scelta!=7);

```

```

    return 0;
}

/*
    Crea lista: Restituisce un punto iniziale della testa.
        La testa punterà al nodo sentinella.
*/
struct LISTA *crealistica_sentinella()
{
    struct LISTA *testa;
    testa=(struct LISTA *) calloc (1,sizeof(struct LISTA));
    return testa;
}
/*
    inserisci_nodo: Questa funzione adatta per le liste bidirezionali, ha il compito di inserire
    un nuovo nodo tra 2 nodi individuati da 2 puntatori: prima e dopo.
    Agganciato il nuovo nodo tra prima e dopo, i nodi prima e dopo dovranno essere agganciati a loro
    Volta al nuovo nodo creato.
*/
void inserisci_nodo (struct LISTA *prima, struct LISTA *dopo, INFO_FIELD Dati)
{
    struct LISTA *ptr;
    /* Crea nuovo nodo */
    ptr= (struct LISTA *)calloc (1,sizeof(struct LISTA)); //Creazione del nodo
    strcpy(ptr->info.nome, Dati.nome); //Copia dei dati
    /* Aggancia il nuovo nodo tra prima e dopo*/
    ptr->next=dopo; //Creazione dei link
    ptr->before=prima;
    /* aggancia prima e dopo in next*/
    (prima)->next=ptr;
    (dopo)->before=ptr;
}

/*
    elimina_nodo: Dato 2 puntatori, dove 'prima' punta al precedente del nodo da eliminare
    e 'dopo' al nodo successivo del nodo da eliminare .
    Semplicemente faremo scavalcare prima a dopo e viceversa.
    Prima sarà del tipo (punt->befor) e dopo del tipo (punt->next).
*/
void elimina_nodo(struct LISTA *prima,struct LISTA *dopo)
{
    struct LISTA *Libera;
    Libera = prima->next;

    /* _____ ELIMINAZIONE _____ */
    /* Aggancia il nodo precedente a dopo, ossia quello successivo trovato*/
    (prima)->next = dopo; //dopo è sempre quello successivo al nodo corrente
    /* aggancia nodo a prima*/
    (dopo)->before= prima;
    //Deallociamo ciò che stava tra prima e dopo: il nodo da eliminato
    free(Libera);
}

/* StampaLista: Stampa la lista a partire dalla testa */
void stampaLISTA (struct LISTA *head1, struct LISTA *head2)
{
    struct LISTA *punt;
    int i=1;
    punt=head1;
    /* Se punt punta alla sentinella2, la visita è finita*/
    while(punt!=head2)
    {
        printf("\n%d NOME = %s\t \n",i, punt->info.nome);
        i++;
        punt=punt->next;
    }
}

/* StampaLista: Stampa la lista a partire dal fondo */
void stampaLISTA2 (struct LISTA *head1, struct LISTA *head2)
{
    struct LISTA *punt;
    int i=1;
    punt=head2;
    /* Se punt punta alla sentinella1, la visita è finita*/
    while(punt!=head1)
    {
}

```

```

        printf("\n%d NOME = %s\t \n", i, punt->info.nome);
        i++;
        punt=punt->before; /* precedente */
    }
}

/* cerca_nodo: Tale function si occupa di restituire il puntatore al nodo che possiede
   il campo info uguale a quello della chiave buffer (stringa da cercare ad esempio) */
struct LISTA *cerca_nodo (struct LISTA *p_head1, struct LISTA *p_head2, char *buffer)
{
    struct LISTA *punt;
    punt = p_head1;
    while (punt->next != p_head2)
    {
        /* se strcmp restituisce 0, e' ok*/
        if ( !strcmp((*punt).info.nome, buffer) )
            return punt;
        punt = punt->next;
    }
    /* Il ciclo esce ad un nodo prima, per&ograve punt sara' avanzato e controlliamo per l'ultima volta*/
    if ( !strcmp((*punt).info.nome, buffer) )
        return punt; //Eccolo: Era l'ultimo!
    else
        return NULL; //NON HO TROVATO NIENTE
}

```

**OUTPUT:**

```

----- LISTA BIDIREZIONALE -----
*[1] - Inserisci in testa della LISTA
*[2] - Inserisci in mezzo alla LISTA
*[3] - Elimina in testa della LISTA
*[4] - Elimina in mezzo alla LISTA
*[5] - Visualizza la LISTA dalla testa
*[6] - Per visualizzare la LISTA dal fondo
*[7] - Uscire dal programma
-----
```

**Scelta:**  
1

Inserire il nome: A

**Scelta:**  
2

Inserire nome del nodo a cui aggiungere il successivo: A  
Inserire nome del nodo da agganciare: B

**Scelta:**  
2

Inserire nome del nodo a cui aggiungere il successivo: B  
Inserire nome del nodo da agganciare: C

--- VISUALIZZAZIONE LISTA DALLA TESTA ---\*      --- VISUALIZZAZIONE LISTA DAL FONDO ---\*

1 NOME = A  
2 NOME = B  
3 NOME = C

1 NOME = C  
2 NOME = B  
3 NOME = A

**Scelta:**  
4

Inserire il nome della persona da eliminare dalla LISTA:B  
Nodo trovato ed eliminato

--- VISUALIZZAZIONE LISTA DALLA TESTA ---\*

1 NOME = A  
2 NOME = C

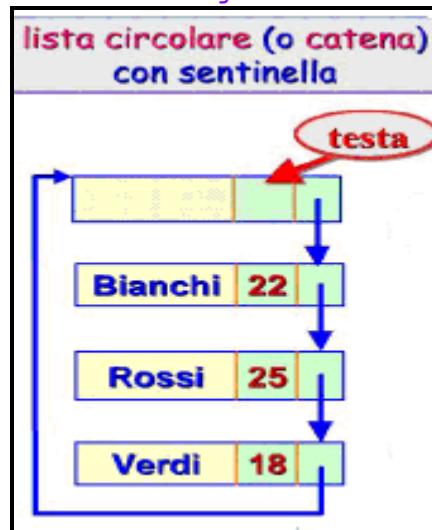
**Scelta:**  
3

--- VISUALIZZAZIONE LISTA DALLA TESTA ---\*

1 NOME = C

## ESERCIZIO 45B [LIV.2]

/\*\*  
[liv.2]Realizzare in C le funzioni per la gestione, mediante menù, delle strutture dati LISTA CIRCOLARE mediante LISTA lineare dinamica e generica con nodo sentinella.



```
**/  
  
#include <stdio.h>  
#include <stdlib.h>  
#define SIZE 20  
/* dichiarazione del tipo INFO_FIELD, serve per le informazioni del nodo */  
typedef struct  
{  
    char nome[20];  
} INFO_FIELD;  
  
/* PROTOTIPI */  
void *crealista_sentinella();  
void *cerca_lista(void *p_head, char *buffer);  
void *cerca_lista_p(void *head, char *buffer);  
short is_testa(void *p_head, char *buffer);  
void elim_nodo(void *p_punt);  
void visita(void *punt, void *head);  
void insl_nodo(short len_info, INFO_FIELD *p_dato, void *p_punt);  
  
int main()  
{  
    /* dichiarazione della LISTA */  
    struct LISTA  
    {  
        INFO_FIELD info;  
        struct LISTA *next;  
        struct LISTA *before;  
    };  
  
    char scelta, buffer[SIZE];  
    short esito;  
    struct LISTA *head, *punt, *old_punt;  
    INFO_FIELD nuovodato; //campo informazioni che verrÀ inserito nel nuovo nodo da creare  
    /* _____ INSERISCI NODO SENTINELLA _____ */  
    /* viene inserito in testa un nodo vuoto.  
       Quando inserisco o elimino, lo faro' da questo(funzionera' come ins_testa)  
       Per la lista CIRCOLARE, necessitiamo di 1 sentinella in testa*/  
    head = (struct LISTA *)crealista_sentinella(); //Creazione nodo sentinella di testa  
    // agganciamo la sentinella la se stessa  
    head->next=head;  
    /* ----- */  
  
    do{  
        printf("----- LISTA CIRCOLARE -----*\n");  
        printf("*[1] - Inserisci un elemento in testa alla LISTA *\n");  
        printf("*[2] - Inserisci un elemento in mezzo alla LISTA *\n");  
        printf("*[3] - Elimina un elemento in testa alla LISTA *\n");  
        printf("*[4] - Elimina un elemento in mezzo alla LISTA *\n");  
        printf("*[5] - Visualizza la LISTA dalla testa *\n");  
        printf("*[6] - Uscire dal programma *\n");  
    }while(scelta != 6);  
}
```

```

printf("-----*\n");

scelta= getch();
switch(scelta)
{
    case '1':
        printf("inserisci il nome ");
        fflush(stdin); gets(nuovodato.nome);
        /* Dato che vuole inserire in testa, gli passo heado cosÃ¬ inserisce dopo il nodo
           sentinella */
        insl_nodo(sizeof(nuovodato), &nuovodato, (void *)head);
        printf("\ninserimento in testa effettuato\n");
        break;

    case '2':
        if (head == NULL)
            printf("\nLISTA VUOTA - IMPOSSIBILE INSERIRE ELEMENTI NEL MEZZO\n");
        else
        {
            printf("Dopo quale nodo della lista lo vuoi inserire: \n");
            fflush(stdin); gets(buffer);
            punt = cerca_lista((void *)head, buffer);
            if (punt == NULL)
                printf("\nData non trovato nella lista\n");
            else
            {
                printf("inserisci il nome ");
                fflush(stdin); gets(nuovodato.nome);
                insl_nodo(sizeof(nuovodato), &nuovodato, (void *)punt);
                printf("\nData inserito dopo il nodo corrente!\n");
            }
        }
        break;

    case '3':
        if (head == NULL)
            printf("\nLISTA VUOTA - IMPOSSIBILE ELIMINARE ELEMENTI\n");
        else
        {
            printf("Eliminazione in testa...\n");
            strcpy(buffer, (*head).info.nome);
            elim_nodo((void *)head);
            printf("eliminazione dalla testa della lista effettuata !\n");
        }
        break;

    case '4':
        if (head == NULL)
            printf("\nLISTA VUOTA - IMPOSSIBILE ELIMINARE ELEMENTI\n");
        else
        {
            printf("Quale nodo della lista vuoi eliminare? \nDigitare uno dei nomi presenti
                   in lista ");
            fflush(stdin);
            gets(buffer);
            /* Controlla se l'elemento da eliminare e' in testa. In tal caso attiviamo
               l'altra function.
               PS head punta al nodo sentinella, head->next al primo nodo info */
            esito = is_testa((void *)head->next, buffer);
            if (esito)
                printf("\nIl nodo e' il primo della lista - usare l'opzione 3\n\n");
            else
            {
                /* Se non e' in testa, ricavati il precedente. Da lui ti elimini il
                   successivo, ossia il nodo interessato */
                old_punt = cerca_lista_p((void *)head, buffer);
                if (old_punt == NULL)
                    printf("\nData non trovato nella lista\n");
                else
                {
                    printf("Eliminazione nodo corrente...\n");
                    elim_nodo((void *)old_punt);
                    printf("%s eliminato dalla lista\n", buffer);
                }
            }
        }
        break;
}

```

```

    case '5':
        /* Se il nodo sentinella ripunta a se stesso */
        if (head->next == head)
            printf("\nLISTA VUOTA\n");
        else
        {
            printf("\nVisita della lista\n\n");
            /* head->next indica l'eventuale nodo successivo al nodo sentinella e
               head lo utilizziamo se ha riciclato in testa*/
            visita((void *)head->next, (void *)head);
        }
        break;
    }
}while(scelta != '6');

return 0;
}

/*
Crea lista: Restituisce un punto iniziale della testa.
           La testa punterà al nodo sentinella.
*/
void *crealista_sentinella()
{
    struct lista
    {
        INFO_FIELD info;
        struct lista *p_next;
    };
    struct lista *testa; /* il tipo non importa, nel main ho il cast */
    testa = (struct lista *) calloc(1, sizeof(struct lista));
    //testa->p_next=NULL; con calloc posso evitare tale istruzione
    return testa;
}

/*
insl_nodo: inserisce dato dopo nodo corrente. Crea il nodo nuovo e lo inserisce
dopo il nodo puntato da p_punt o head (quindi inserirebbe in testa)
*/
void insl_nodo(short len_info, INFO_FIELD *p_dato, void *p_punt)
{
    struct lista
    {
        INFO_FIELD info;
        struct lista *p_next;
    } *ptr;

    /* Crea nodo e inserisci le info con la memcpy*/
    ptr = calloc(1, sizeof(struct lista));

    if (ptr == NULL)
    {fprintf(stderr, "\nAllocazione del nodo non riuscita\n"); exit(EXIT_FAILURE);}
    else
    {
        memcpy(ptr, p_dato, len_info);
        /* Aggancia ptr al successivo di p_punt o phead */
        ptr->p_next = ((struct lista *)p_punt)->p_next;
        /* Aggancia p_punt al nuovo nodo*/
        ((struct lista *)p_punt)->p_next = ptr;
        /*p_punt = ptr; Non mi serve a niente ricordare dove sia avvenuto l'inserimento
    }
}

/*
visita: visita la lista a partire dalla testa. Quando un nodo ripunterà alla testa, si ferma.
*/
void visita(void *punt, void *head)
{
    struct lista
    {
        INFO_FIELD info;
        struct lista *p_next;
    } *ptr;
    ptr = punt;
    int i=1;
    /* Essendo una lista circolare, se il prossimo riparte dal testa si ferma*/
}

```

```

while (ptr != head)
{
    printf("%d NOME = %s \t\n", i++, (*ptr).info.nome);
    ptr = ptr->p_next;
}
}

/*
    Eliminazione nodo: Viene effettuata una visita sulla lista e appena viene trovato il nodo
    da eliminare
*/
void elim_nodo(void *p_punt)
{
    struct lista
    {
        INFO_FIELD info;
        struct lista *p_next;
    } *ptr;
    /* ptr la usiamo per scrivere meno codice per risalire */
    ptr = ((struct lista*)p_punt)->p_next;
    /* Punt->next, sara' il successivo del suo successivo */
    ((struct lista*)p_punt)->p_next = ptr->p_next;
    free(ptr); //libera heap
}
/*
is_testa: passato il nodo dopo il nodo sentinella (la testa reale), controlla
se cio' che cerchiamo e' in testa.
*/
short is_testa(void *p_head, char *buffer)
{
    struct lista
    {
        INFO_FIELD info;
        struct lista *p_next;
    } *punt;
    punt = p_head;

    /* se strcmp restituisce 0, e' ok*/
    if (!strcmp((*punt).info.nome, buffer) )
        return 1;
    else
        return 0; //NON HO TROVATO NIENTE
}

/*
cerca_lista_p: Tale function si occupa di restituire il puntatore PRECEDENTE al nodo che possiede
il campo info uguale a quello della chiave buffer (stringa da cercare ad esempio), tale che
possiamo effettuare l'eliminazione in mezzo. La particolarita' di questa function e' quello di
SALVARE il puntatore "VECCHIO", prima che venga aggiornato.
PS La function e' adatta per cercare in mezzo, ma non in testa perche' non avrebbe l'opportunita'
di salvare il precedente (nel main abbiamo un dovuto controllo)
*/
void *cerca_lista_p(void *head, char *buffer)
{
    struct lista
    {
        INFO_FIELD info;
        struct lista *p_next;
    } *punt, *previous;
    punt = head;

    while (punt->p_next != head)
    {
        if ( !strcmp((*punt).info.nome, buffer) )
            return previous; //restituisce il precedente
        previous = punt; /* PRIMA DI AGGIORNARE, SALVA PUNTATORE CHE SARA? PRECEDENTE*/
        punt = punt->p_next;
    }
    if ( !strcmp((*punt).info.nome, buffer) )
        return previous;
    else
        return NULL;
}

/*
cerca_lista: Tale function si occupa di restituire il puntatore al nodo che possiede
il campo info uguale a quello della chiave buffer (stringa da cercare ad esempio)

```

```
/*
void *cerca_lista(void *p_head, char *buffer)
{
    struct lista
    {
        INFO_FIELD info;
        struct lista *p_next;
    } *punt;
    punt = p_head;
    while(punt->p_next != p_head)
    {
        /* se strcmp restituisce 0, e' ok*/
        if ( !strcmp((*punt).info.nome, buffer) )
            return punt;
        punt = punt->p_next;
    }
    /* Il ciclo uscire ad un nodo prima, perA² punt sara' avanzato e controlliamo per l'ultima volta*/
    if ( !strcmp((*punt).info.nome, buffer) )
        return punt; //Eccolo: Era l'ultimo!
    else
        return NULL; //NON HO TROVATO NIENTE
}
```

**OUTPUT:**

```
*----- LISTA CIRCOLARE -----
*[1] - Inserisci un elemento in testa alla LISTA *
*[2] - Inserisci un elemento in mezzo alla LISTA *
*[3] - Elimina un elemento in testa alla LISTA *
*[4] - Elimina un elemento in mezzo alla LISTA *
*[5] - Visualizza la LISTA dalla testa *
*[6] - Uscire dal programma *
-----*
```

**inserisci il nome A**

**inserimento in testa effettuato**

**Scelta:**

**2**

**Inserire nome del nodo a cui aggiungere il successivo: A**  
**Inserire nome del nodo da agganciare: B**

**Scelta:**

**2**

**Inserire nome del nodo a cui aggiungere il successivo: B**  
**Inserire nome del nodo da agganciare: C**

**Scelta:**

**4**

**Inserire il nome della persona da eliminare dalla LISTA:B**  
**Nodo trovato ed eliminato**

**Scelta:**

**3**

**Eliminazione in testa...**  
**eliminazione dalla testa della lista effettuata !**

**Visita della lista**

**1 NOME = B**

## ESERCIZIO 47 [LIV.2]

```
/** [LIV2] Scrivere function C per la costruzione e visita per livelli di un albero
qualsiasi
rappresentato mediante array. [Suggerimento: la struct che definisce il generico nodo
dell'albero, deve contenere i seguenti campi: l'informazione, il grado ed un array di
puntatori (ai nodi figli) di dimensione pari al massimo grado dei nodi che si suppone
noto].**/
#include <stdlib.h>
#include <string.h>
#define MAX_GRADO 4
#define MAX_LEN 10 //Grandezza dei nodi

/* Struttura input */
typedef struct
{
    char nome[20];
    short padre; /* Indice che individua il padre*/
}nodo;
/* Struttura in output */
typedef struct
{
    char nome[20];
    short grado; /* Numero di figli per quel nodo */
    short figlio[MAX_GRADO]; /* Gli indici ai figli. Punta al iesimo figlio */
}nodo_link;

/* Prototipi */
void enqueue (nodo_link elem, int *fondo, nodo_link coda[]);
void dequeue (int *testa,nodo_link coda[],nodo_link *appoggio,int fondo);
void azzerar_grado (nodo_link link[]);
void costruisci_albero(nodo_link link[],nodo albero[],short *radice );
void visita_livelli (nodo_link link[], short radice);

int main()
{
    /* Definiamo la struttura iniziale dell'albero, dove i numeri indicano l'indice del
padre nel vettore */
    nodo
albero[MAX_LEN]={{ "Gabriele",6}, {"Giuseppe",4}, {"Pasquale",8}, {"Giulia",8}, {"Angela",8},
                 {"Maria",9}, {"Giovanni",-1}, {"Antonio",9}, {"Gennaro",6}, {"Anna",6}};

    /* Tabella inizialmente sporca; costruiremo l'albero, definendo i gradi(nuemro dei
figli) e gli indici in cui si trovano tali figli nel vettore. */
    nodo_link link[MAX_LEN];
    /* Indice che individua la radice; La ricaveremo dalla costruzione dell'albero */
    short radice;

    azzerar_grado(link); //Viene azzerato grado, inzialmente a 0
    /*Inizizlizzazione dell'albero*/

    costruisci_albero(link,albero,&radice); //Chiamata alla function per la
    //costruzione dell'albero
    printf("\t---- Costruzione e visita per livelli di un albero qualsiasi ---*\n\n");
    printf("\tRADICE\n");
    visita_livelli(link,radice); //Chiamata alla function per la visita dell'abero
    printf("\n");

    return 0;
}

/* Azzero il campo grado*/
void azzerar_grado (nodo_link link[])
{
    short i,j;
    for(i=0;i<MAX_LEN;i++)
        link[i].grado=0; //Mette a zero il campo grado
}
/*
Costruisci Albero: ha il compito di creare una tabella in cui, per ogni elemento della
```

```

tabella in cui sono presenti i nodi figli, ricaviamo l'indice del padre a cui
appartengono e con questo indice ci posizioniamo nella tabella in cui sono presenti i
nodi padri che stiamo creando, incrementando il medesimo grado e indicando chi siano i
figli(indici)
*/
void costruisci_albero(nodo_link link[], nodo_albero[], short *radice )
{
    short i, padre=0; /* In padre, va l'indice del padre, sulla quale costruiremo
                        la tabella 'padre' */
    for(i=0;i<MAX_LEN;i++)
    {
        /* Dimmi l'indice padre*/
        padre=albero[i].padre;
        /* Costruisci la tabella padre, inserendo il nome, link e aggiornando grado */
        strcpy(link[i].nome,albero[i].nome);

        /* Se il padre del nodo corrente Ã" diverso dalla radice */
        if(padre!=-1)
        {
            (link[padre].grado)++; //Aggiorna numero figli
            (link[padre].figlio[(link[padre].grado)-1])=i; /* Si salva il link tra padre e
figlio. */

        }
        else /* Se e' il nodo radice, salva l'indice della radice */
        *radice=i;
    }
}

/*
Visita Livelli: Conoscendo il numero dei figli (GRADO) e la loro posizione, posso
scorrere l'albero(Scorro per livelli). La radice inizialmente la metto in coda, la
estraggo e ricavo il grado.
Stampo i figli per grado_padre volte e li metto in coda.
*/
void visita_livelli (nodo_link link[], short radice)
{
    int testa=0,fondo=0,i,grado_padre=0;
    /* CODA in cui sono inseriti gli elementi per livelli*/
    nodo_link coda[MAX_LEN];
    nodo_link appoggio; //Nodo d'appoggio per cio' che estraggo

    /* Metti in coda l'indice della radice*/
    enqueue(link[radice],&fondo,coda);

    /* Cicla fin quando testa non raggiunge fondo (e quindi n nodi esauriti) */
    while(fondo!=testa)
    {
        /* Estrai dalla coda nodo inserito */
        dequeue(&testa,coda,&appoggio,fondo);

        printf("PADRE: %-15s\tgrado=%hd\n",appoggio.nome,appoggio.grado);
        grado_padre=appoggio.grado; //ricava il grado per quel nodo padre

        /* Stampa e inserisci in coda i figli di quel nodo, che sono lunghi grado_padre. */
        if(grado_padre>0) printf("FIGLI:\n");
        for(i=0;i<grado_padre;i++) //Si mettono in coda i figli del nodo corrente
        {
            printf(" %s\n", link[appoggio.figlio[i]].nome);
            enqueue(link[appoggio.figlio[i]],&fondo,coda);/* Inserisci in coda i figli */
        }
        printf("\n-----\n");
    }

    /* Inserisci in fondo alla coda*/
void enqueue (nodo_link elem, int *fondo, nodo_link coda[])
{

```

```

/* Vet_Coda[*Fondo]=Elemento -> (*Fondo)++;  

Inserisco in coda e aumento Fondo, che mi indica  

la il posto della fila successiva */  

if(*fondo<MAX_LEN)  

    coda[(*fondo)++]=elem;  

else printf("Non è possibile effettuare altri inserimenti!\n");  

}  

/* Estrai dalla testa della cod a*/  

void dequeue (int *testa,nodo_link coda[],nodo_link *appoggio,int fondo)  

{  

    /* Testa arrivata a LEN QUEUE, mi indichera' che non potro' ne inserire un elemento ne  

    eliminarlo.  

    Avanzando testa, elimino il primo elemento in testa. */  

    if(*testa < fondo)  

    {  

        *appoggio= coda[(*testa)];  

        (*testa)++;  

    }  

    else printf("\nNessun elemento presente nella pila!\n");  

}

```

**OUTPUT:**

**RADICE** Giovanni grado=3  
PADRE: Giovanni FIGLI: Gabriele Gennaro Anna

---

PADRE: Gabriele grado=0

---

PADRE: Gennaro grado=3 FIGLI: Pasquale Giulia Angela

---

PADRE: Anna grado=2 FIGLI: Maria Antonio

---

PADRE: Pasquale grado=0

---

PADRE: Giulia grado=0

---

PADRE: Angela grado=1 FIGLI: Giuseppe

---

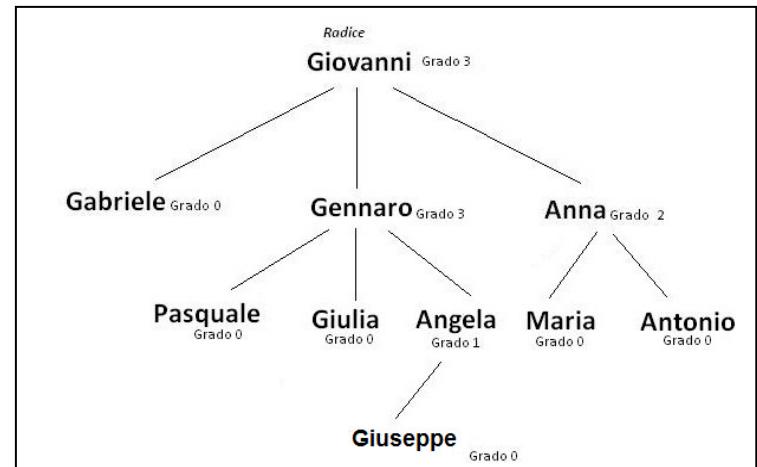
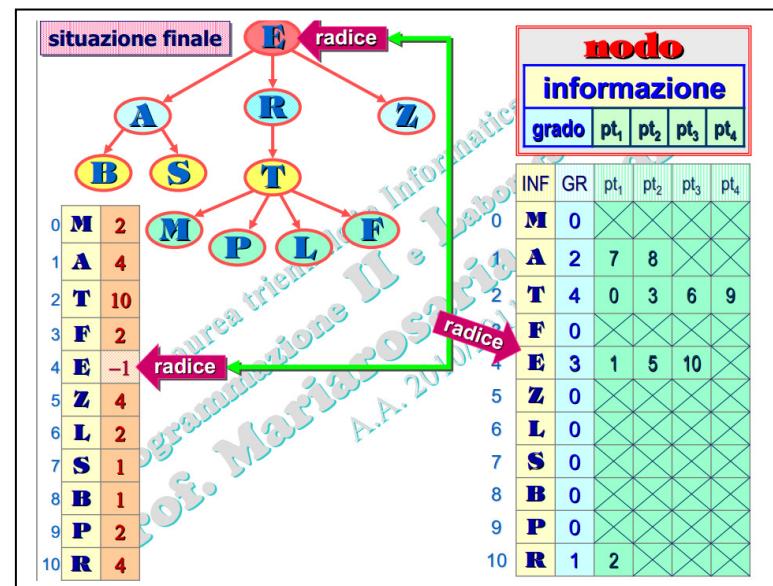
PADRE: Maria grado=0

---

PADRE: Antonio grado=0

---

PADRE: Giuseppe grado=0



## ESERCIZIO 49 [LIV.1]

```
/*
[liv1] Scrivere le function C per la visita (preorder, inorder e postorder) di un albero binario
rappresentato mediante array.
*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MAX 16

void visita_preorder(short albero[], short i);
void visita_inorder(short albero[], short i);
void visita_postorder(short albero[], short i);
short foglia(short [], short);

void main()
{
    /* Per rappresentare l'albero, al piu' necessito (2^l)-1 nodi.
       Io alloco staticamente 2^n elementi, perche la componente albero[0]
       non deve contenere il nodo dato che siccome in C il primo elemento
       parte da 0, se volessi trovare il figlio destro o sinistro 2*i risulterebbe
       essere sempre 0.
       A[2^n] dove A[1] contiene il primo nodo dell'albero */
    /* Albero completo albero[] = {-1, 0, 1, 2, 3, 4, 5, 6, 71, 8, 9, 10, 11, 12, 13, 14}; */
    /* Esempio su un albero non completo*/
    short albero[] = {-1, 0, 1, 2, 3, 4, 5, 6, -1, -1, 9, 10, 11, 12, 13, -1};
    short i=1; //Da quale nodo comincio

    printf("      %hd\n", albero[1]);
    puts("      / \\" );
    puts("      / \\" );
    puts("      / \\" );
    puts("      / \\" );
    printf("      %hd      %hd      \n", albero[2], albero[3]);
    puts("      / \\"      / \\" );
    puts("      / \\"      / \\" );
    printf("      %hd      %hd      %hd      %hd \n", albero[4], albero[5], albero[6], albero[7]);
    puts("      / \\"      / \\"      / \\" );
    printf("      %hd      %hd      %hd      %hd      %hd \n", albero[10], albero[11], albero[12], albero[13], albero[14]);

    puts("\nVisita PREORDER");
    visita_preorder(albero,i);
    printf("\n");

    puts("\nVisita INORDER");
    visita_inorder(albero,i);
    printf("\n");

    puts("\nVisita POSTORDER");
    visita_postorder(albero,i);
    printf("\n");
}

/*
DALLA TEORIA:
L'algoritmo di visita di un albero binario in generale deve prevedere una struttura dati di appoggio, uno "STACK", il quale conserva l'informazione sull'ordine dei nodi di cui non e' stata ancora completata la visita. Esso deve essere gestito direttamente dal programmatore nell'algoritmo iterativo, mentre viene gestito dal linguaggio di programmazione quando l'algoritmo e' descritto in maniera ricorsiva.

Nella visita PREORDER, lo stack conserva i nodi di cui non si e' ancora visitato il sottoalbero DX
Nella visita INORDER, lo stack conserva i nodi di cui non si e' ancora visitato il sottoalbero DX
Nella visita POSTORDER, lo stack conserva i nodi di cui non si e' ancora visitata la radice

QUANDO avviene il controllo sulla foglia, in realta' nell'istanza precedente abbiamo fatto un autoattivazione sulla foglia e SOLO nell'istanza successiva controlliamo che sia una foglia perche' non riuscirei a stampare il nodo.
*/
/*
PREORDER (ordine anticipato): - radice
                                - sottoalbero sinistro
                                - sottoalbero destro

```

```

*/
void visita_preorder(short albero[], short i)
{
    if (foglia(albero, i)) /*soluzione banale: indice>dimensione albero*/
        return; //indica il ritorno al precedente processo lasciato in sospeso

    printf("%hd ", albero[i]); /*stampa il nodo corrente dell'albero */

    /*ricorsione binaria, con l'istruzione 2*indice si passa alla function l'indice del figlio
    sinistro di ai in quanto esso ha indice 2*i; richiama la function sul sotto-albero di
    sinistra avente come radice il figlio sinistro */
    visita_preorder(albero, 2*i);

    /*con l'istruzione 2*indice+1 si passa alla function l'indice del figlio destro di ai in
    quanto esso ha indice 2*i+1; richiama la function sul sotto-albero di destra avente come
    radice il figlio destro*/
    visita_preorder(albero, 2*i+1);
}

/*
INORDER (ordine simmetrico): - sottoalbero sinistro
                            - radice
                            - sottoalbero destro
*/
void visita_inorder(short albero[], short i)
{
    if (foglia(albero, i)) /*soluzione banale: indice>dimensione albero*/
        return; //indica il ritorno al precedente processo lasciato in sospeso

    /*si passa alla function l'indice del figlio sinistro di ai in quanto esso ha indice 2*i;
    richiama la function sul sotto-albero di sinistra avente come radice il figlio sinistro */
    visita_inorder(albero, 2*i);

    printf("%hd ", albero[i]); /*stampa il nodo corrente dell'albero*/

    /*si passa alla function l'indice del figlio destro di ai in quanto esso ha indice 2*i+1;
    richiama la function sul sotto-albero di destra avente come radice il figlio destro */
    visita_inorder(albero, 2*i+1);
}

/*
POSTORDER (ordine posticipato : - sottoalbero sinistro
                            - sottoalbero destro
                            - radice
*/
void visita_postorder(short albero[], short i)
{
    if (foglia(albero, i)) /*soluzione banale= fermati se trovi una foglia */
        return; //indica il ritorno al precedente processo lasciato in sospeso

    /*si passa alla function l'indice del figlio sinistro di ai in quanto esso ha indice 2*i
    richiama la function sul sotto-albero di sinistra avente come radice il figlio sinistro */
    visita_postorder(albero, 2*i);

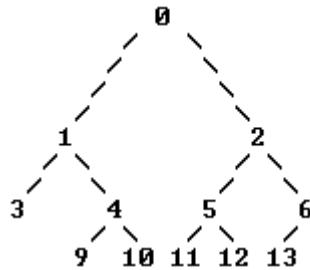
    /*si passa alla function l'indice del figlio destro di ai in quanto esso ha indice 2*i+1
    richiama la function sul sotto-albero di destra avente come radice il figlio destro */
    visita_postorder(albero, 2*i+1);

    /*stampa elemento corrente dell'albero */
    printf("%hd ", albero[i]);
}

/*
Funzione che restituisce se quell'elemento e' una foglia.
Il controllo non viene effettuato direttamente sulla foglia, ma sul
"Non esistente " figlio della foglia (arrivato alla foglia, tento di accedere
ai figli, ma la function mi dice ,APPUNTO , che non ci sono figli).
Non potevo gestire prima di richiamare foglia, perche' non mi avrebbe stampato
foglia.
*/
short foglia(short albero[], short i)
{
    /* Se i ha superato il massimo*/
    if(i>=MAX) return 1;
    /* Se il nodo puntato e' "vuoto" e quindi nella chiamata successiva avevamo una foglia*/
    if(albero[i]==-1) return 1;
    /* Non e' una foglia */
}

```

```
    return 0;
}
```

**OUTPUT:****Visita PREORDER**

```
0 1 3 4 9 10 2 5 11 12 6 13
```

**Visita INORDER**

```
3 1 9 4 10 0 11 5 12 2 13 6
```

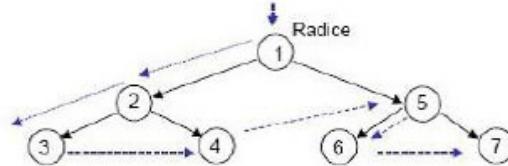
**Visita POSTORDER**

```
3 9 10 4 1 11 12 5 13 6 2 0
```

**VISITA PREORDER**

Nella visita in preordine, se l'albero non è vuoto:

- si analizza la radice dell'albero
- si visita in preordine il sottoalbero sinistro
- si visita in preordine il sottoalbero destro

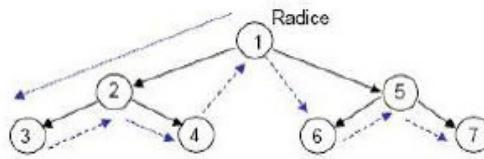


Nella visita in preordine del precedente albero i nodi verrebbero visitati nel seguente ordine:  
1 → 2 → 3 → 4 → 5 → 6 → 7

**VISITA INORDER**

Nella visita in ordine, se l'albero non è vuoto:

- si visita in ordine il sottoalbero sinistro
- si analizza la radice dell'albero
- si visita in ordine il sottoalbero destro

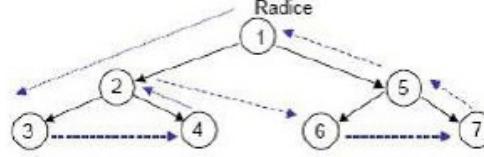


Nella visita in ordine del precedente albero i nodi verrebbero visitati nel seguente ordine:  
3 → 2 → 4 → 1 → 6 → 5 → 7

**VISITA POSTORDER**

Nella visita in postordine, se l'albero non è vuoto:

- si visita in postordine il sottoalbero sinistro
- si visita in postordine il sottoalbero destro
- si analizza la radice dell'albero

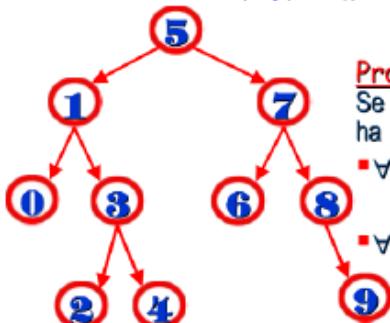


Nella visita in ordine del precedente albero i nodi verrebbero visitati nel seguente ordine:  
3 → 4 → 2 → 6 → 7 → 5 → 1

## ESERCIZIO 51 [LIV.2]

```
/*
[liv.2] Scrivere function C iterativa per la costruzione di un albero binario di ricerca rappresentato mediante array.
*/
```

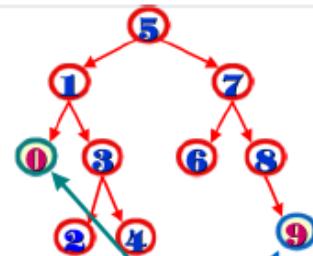
- È un particolare albero binario in cui la **visita inorder** (in **ordine simmetrico**) consente di accedere alle informazioni in ordine crescente di chiave (**key(nodo)**);



### Proprietà di un SBT:

Se  $y$  è un qualsiasi nodo dell'albero si ha

- ∀ nodo  $x$  del sottoalbero sinistro di  $y$   $\text{key}(x) \leq \text{key}(y)$
- ∀ nodo  $z$  del sottoalbero destro di  $y$   $\text{key}(y) \leq \text{key}(z)$



In un albero binario di ricerca con chiavi tutte diverse, si ha

- ✓ la chiave di **valore minimo** occupa il nodo foglia del sottoalbero più a sinistra (cioè la prima foglia che si incontra con la visita inorder);
- ✓ la chiave di **valore massimo** occupa il nodo foglia del sottoalbero più a destra (cioè l'ultima foglia che si incontra con la visita inorder);

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
typedef struct tree
{
    int valore;
    int esiste;
}ALBERO;

void costruisci_albero_binario(ALBERO albero[], int n, int valori_input[]);
void visita_inorder(ALBERO albero[], short i, short n);
short foglia(ALBERO albero[], short i, short n);

/* IDEA: l'albero binario come sappiamo e' un albero che possiede al piu' 2 vie. Costruire un albero binario di ricerca significa dare dei valori e su di esso costruire una struttura ordinata di quei valori, in cui il nodo "piu a sinistra" e' il minimo, mentre il nodo piu' a destra e' il massimo. Per stampare in ordine, utilizziamo un accesso INORDER.
*/
int main()
{
    /* in albero saranno disposti i nodi dando in input 'valori_input'*/
    ALBERO *albero;
    int i=0, *valori_input, n, nxn;

    printf("Inserisci il numero di nodi dell'albero: ");
    scanf("%d", &n);
    nxn=pow(2,n); //n^2 per i link perche' le moltiplicazioni per 2 possono far uscire fuori dal range
                    //ad esempio, con un albero non bilanciato

    /* Allociamo un albero vuoto e dei valori da inserire */
    albero = (int *)malloc( nxn*(sizeof(ALBERO)) );
    valori_input = (int *)malloc(n*(sizeof(int)));
    for(i=1;i<=nxn;i++) //puliamo l'albero
    {
        albero[i].valore=-1;
        albero[i].esiste=0;
    }
    /* input valori da inserire */
    for(i=0; i<n; i++)
    {
        printf("\n");
        printf("Inserisci un valore: ");
        scanf("%d", &valori_input[i]);
    }
    /* costruisci albero binario*/
    costruisci_albero_binario(albero, n, valori_input);
    /* Stampo l'albero sottoforma di array*/
    printf("\nVALORI DELL'ALBERO NELL'ARRAY\n");
    i=1;
```

```

while (i<=n)
{
    if(albero[i].esiste) printf("%d ", albero[i].valore);
    i++;
}
printf("\n");
/* visita in order. Passo n per il caso di un albero non bilanciato */
printf("\nVALORI DELL'ALBERO NELL'ARRAY CON VISITA IN ORDER (quindi ordinato)\n");
visita_inorder(albero, 1, n);
printf("\n");
return 0;
}
/*
costruisci albero binario: si e' eseguito l'algoritmo per la costruzione
dell'albero binario.
*/
void costruisci_albero_binario(ALBERO albero[], int n, int valori_input[])
{
    short i=0, i_radice; //negli alberi la primo componente non e' considerata

    /* Inseriamo la radice */
    albero[1].valore=valori_input[i++];
    albero[1].esiste=1;

    while(i<n) //Fin quando esistono nodi da inserire
    {
        i_radice = 1; //indice parte dalla radice

        /* Controllo se il valore val al sotto dell'albero di destra o di sinistra
         ricavo i_radice foglia, e in base ai casi, successivamente ai while,
         inserisco*/

        /* CONTROLLO SE VA SOTTO ALBERO SINISTRO*/
        while(valori_input[i]<=albero[i_radice].valore && albero[2*i_radice].esiste )
            i_radice=2*i_radice; //scendi al sottoalbero sinistro
        /* CONTROLLO SE VA SOTTO ALBERO DESTRO*/
        while(valori_input[i]>=albero[i_radice].valore && albero[(2*i_radice)+1].esiste )
            i_radice= (2*i_radice)+1; //scendi al sottoalbero destro

        /* TROVATO IL SOTTOALBERO, VEDO SE E' FIGLIO SINISTRO O DESTRO*/
        if(valori_input[i] < albero[i_radice].valore)
        {
            albero[2*i_radice].valore = valori_input[i];
            albero[2*i_radice].esiste = 1;
        }
        else
        {
            albero[(2*i_radice)+1].valore = valori_input[i];
            albero[(2*i_radice)+1].esiste = 1;
        }
        i++;
    }

}
/*
INORDER (ordine simmetrico): - sottoalbero sinistro
                            - radice
                            - sottoalbero destro
input: i = la radice parte da 1, dato da main
       albero = e' l'albero costruito
       n = numero dei nodi
*/
void visita_inorder(ALBERO albero[], short i, short n)
{
    if (foglia(albero, i, n)) /* controlla se e' foglia */
        return; //indica il ritorno al precedente processo lasciato in sospeso

    /*si passa alla function l'indice del figlio sinistro di ai in quanto esso ha indice 2*i;
     richiama la function sul sotto-albero di sinistra avente come radice il figlio sinistro */
    visita_inorder(albero, 2*i, n);

    printf("%hd ",albero[i].valore); /*stampa il nodo corrente dell'albero*/

    /*si passa alla function l'indice del figlio destro di ai in quanto esso ha indice 2*i+1;
     richiama la function sul sotto-albero di destra avente come radice il figlio destro */
    visita_inorder(albero, 2*i+1, n);
}

```

```

/*
Funzione che restituisce se quell'elemento e' una foglia.
Il controllo non viene effettuato direttamente sulla foglia, ma sul
"Non esistente " figlio della foglia (arrivato alla foglia, tento di accedere
ai figli, ma la function mi dice, APPUNTO, che non ci sono figli).
Non potevo gestire prima di richiamare foglia, perche' non mi avrebbe stampato
foglia.
*/
short foglia(ALBERO albero[], short i, short n)
{
    /* Se i ha superato il massimo*/
    if(i>n) return 1;
    /* Se il nodo puntato e' "vuoto", nella chiamata precedente avevamo una foglia*/
    if(!albero[i].esiste) return 1;
    /* Non e' una foglia */
    return 0;
}

```

### OUTPUT:

Inserisci il numero di nodi dell'albero: 10

Inserisci un valore: 5

Inserisci un valore: 7

Inserisci un valore: 8

Inserisci un valore: 1

Inserisci un valore: 6

Inserisci un valore: 0

Inserisci un valore: 3

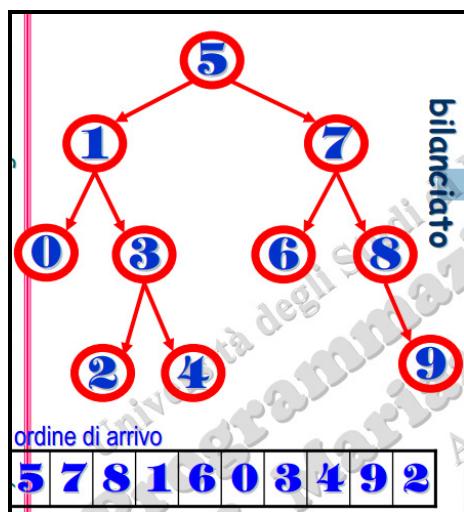
Inserisci un valore: 4

Inserisci un valore: 9

Inserisci un valore: 2

VALORI DELL'ALBERO NELL'ARRAY  
5 1 7 0 3 6 8 2 4 9

VALORI DELL'ALBERO NELL'ARRAY CON VISITA IN ORDER <quindi ordinato>  
0 1 2 3 4 5 6 7 8 9



## ESERCIZIO 52 [LIV.3]

```

/*
[liv.3] Scrivere function C iterativa per la costruzione di un albero binario di ricerca
rappresentato mediante liste multiple.
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
struct ALBERO
{
    int valore;
    //int esiste;
    struct ALBERO *figlio_dx;
    struct ALBERO *figlio_sx;
}ALBERO;

void costruisci_albero_binario(struct ALBERO **p_head, int n, int valori_input[]);
void visita_inorder(struct ALBERO *p_punt);
short foglia(struct ALBERO *p_punt);
void inizializza_albero(struct ALBERO **p_head);
/*
IDEA: l'albero binario come sappiamo e' un albero che possiede al piu' 2 vie.
Costruire un albero binario di ricerca significa dare dei valori e su di esso
costruire una struttura ordinata di quei valori, in cui il nodo "piu a sinistra"
e' il minimo, mentre il nodo piu' a destra e' il massimo. Per stampare in ordine,
utilizziamo un accesso INORDER.
*/
int main()
{
    /* in albero saranno disposti i nodi dando in input 'valori_input'*/
    struct ALBERO *p_head; //Puntatore alla testa dell'albero, ossia la radice
    int i=0, *valori_input, n;

    printf("Inserisci il numero di nodi dell'albero: ");
    scanf("%d", &n);
    valori_input = (int *)malloc(n*sizeof(int));
    if(!valori_input) exit("ERRORE DI ALLOCAZIONE");

    /* input valori da inserire */
    for(i=0; i<n; i++)
    {
        printf("\n");
        printf("Inserisci un valore: ");
        scanf("%d", &valori_input[i]);
    }

    /* STAMPA ARRAY DI INPUT */
    printf("\nVALORI NELL'ARRAY IN INPUT \n");
    for(i=0; i<n; i++)
        printf("%d ", valori_input[i]);
    printf("\n");

    /* Crea lista */
    inizializza_albero(&p_head);
    /* Costruisci albero binario */
    costruisci_albero_binario(&p_head, n, valori_input);
    /* VISITA INORDER PER STAMPARE IN ORDINE */
    printf("\nVISITA IN ORDER DELL'ALBERO BINARIO COSTRUITO (valori in ordine) \n");
    visita_inorder(p_head);

    printf("\n");
    return 0;
}
/*
inizializza_albero: Assegno la testa dell'albero un indirizzo vuoto, dato che
dobbiamo ancora comporlo.
*/
void inizializza_albero(struct ALBERO **p_head)
{
    *p_head=NULL;
    return;
}
/*
costruisci_albero_binario: costruisce un albero binario mediante delle liste

```

```

    multiple.

*/
void costruisci_albero_binario(struct ALBERO **p_head, int n, int valori_input[])
{
    struct ALBERO *ptr, *punt; /* puntatore a nodo nuovo e punt=puntatore che visita l'albero */
    int i=0;

    /* Crea radice */
    ptr=calloc(1, sizeof(struct ALBERO)); //Facendo calloc, i puntatori dx e sx punteranno a NULL
    if(!ptr) exit("ERRORE DI ALLOCAZIONE");
    ptr->valore = valori_input[i++]; //Inizializzo radice
    *p_head=ptr; //la nuova testa sara' la radice

    /* fin quando ho degli elementi da inserire */
    while(i<n)
    {
        punt = *p_head; // Parti dalla radice

        /* Crea nodo */
        ptr=calloc(1, sizeof(struct ALBERO)); //Facendo calloc, i puntatori dx e sx punteranno a NULL
        if(!ptr) exit("ERRORE DI ALLOCAZIONE");

        /* Controllo dove devo inserire tale nodo */
        /* CONTROLLO SE VA SOTTO ALBERO SINISTRO*/
        while(valori_input[i] <= punt->valore && punt->figlio_sx!=NULL )
            punt=punt->figlio_sx; //scendi al sottoalbero sinistro
        /* CONTROLLO SE VA SOTTO ALBERO DESTRO*/
        while(valori_input[i]>= punt->valore && punt->figlio_dx!=NULL )
            punt=punt->figlio_dx; //scendi al sottoalbero destro
        /* TROVATO IL SOTTOALBERO, VEDO SE E' FIGLIO SINISTRO O DESTRO*/
        if(valori_input[i] < punt->valore)
        {
            punt->figlio_sx=ptr; //Aggancia come figlio sinistro
            ptr->valore = valori_input[i]; //inserisci valore nel nodo
        }
        else
        {
            punt->figlio_dx=ptr; //Aggancia come figlio destro
            ptr->valore = valori_input[i]; //inserisci valore nel nodo
        }
        i++; //vai ai prossimi valori input
    }

}

/*
INORDER (ordine simmetrico): - sottoalbero sinistro
                            - radice
                            - sottoalbero destro
input: *p_punt= Puntatore all'albero. Richiama sempre sotto albero sinistro,
       fin quando non trova uno che non ha figli sinistri. In tal caso prendera'
       il nodo radice e si sposta al sottoalbero destro.
*/
void visita_inorder(struct ALBERO *p_punt)
{
    if (foglia(p_punt)) /* controlla se e' foglia, se ci troviamo in un nodo nullo */
        return; //indica il ritorno al precedente processo lasciato in sospeso

    /* passiamo al padre del sottoalbero sinistro */
    visita_inorder(p_punt->figlio_sx);

    /*stampa il nodo corrente dell'albero*/
    //dopo che il sottoalbero sinistro e' finito, mi stampa la radice del sotto albero per
    //poi passare al sotto albero destro
    printf("%hd ", p_punt->valore);

    /* passiamo al padre del sottoalbero destro */
    visita_inorder(p_punt->figlio_dx);
}

/*
Funzione che restituisce se quell'elemento e' una foglia.
Effettuando le chiamate ricorsive per il sotto albero a sinistra o destro, trovera
un nodo che punta a null e quindi significhera' che e' una foglia. Se lo e' torna
all'autoattivazione
*/
short foglia(struct ALBERO *p_punt)
{

```

```

/* Se il nodo a cui ha acceduto punta a null, l'autoattivazione ritorna al programma
   chiamante per segnalare la foglia riguardante la precedente chiamate */
//Se non e' una foglia
if(p_punt!=NULL)
    return 0;
/* se punta a null significa che il nodo che l'ha autoattivata era una foglia */
return 1;
}

```

**OUTPUT:**

Inserisci il numero di nodi dell'albero: 10

Inserisci un valore: 5

Inserisci un valore: 7

Inserisci un valore: 8

Inserisci un valore: 1

Inserisci un valore: 6

Inserisci un valore: 0

Inserisci un valore: 3

Inserisci un valore: 4

Inserisci un valore: 9

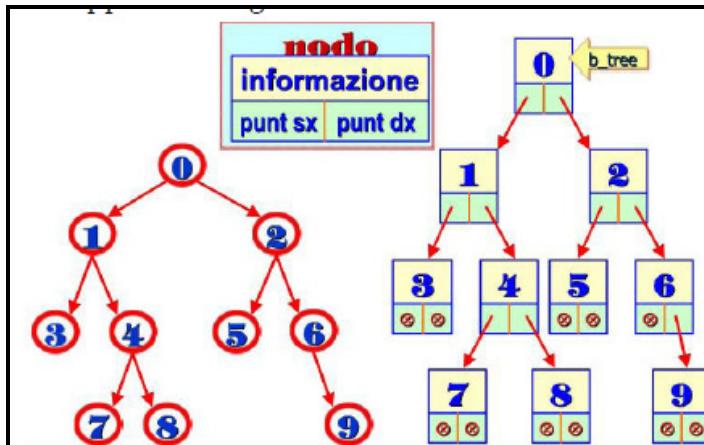
Inserisci un valore: 2

VALORI NELL'ARRAY IN INPUT

5 7 8 1 6 0 3 4 9 2

VISITA IN ORDER DELL'ALBERO BINARIO COSTRUITO (valori in ordine)

0 1 2 3 4 5 6 7 8 9



### ESERCIZIO 53 [LIV.2]

```
/*[liv.2] Scrivere function C iterativa per la costruzione di un heap rappresentato mediante array.*/
#include <stdio.h>
#include <stdlib.h>
void costruisci_heap(short heap[], short n);
void heapify(short heap[], short i);
void visualizza_heap(short heap[], short n);
/*CENNI TEORICI
Un heap è una struttura dati gerarchica. In particolare è un albero binario completo o quasi completo in cui ogni nodo verifica la proprietà dell'heap:
key(X) <= key(padre(X))
Cioè, dato un nodo, il suo valore è sicuramente <= rispetto al valore del padre.
Da ciò ne consegue che l'elemento massimo è memorizzato nella radice dell'heap.*/
```

## Struttura dati HEAP

Un heap è un albero binario quasi completo i cui nodi sono etichettati tramite chiavi (da un insieme ordinato).

### Proprietà heap:

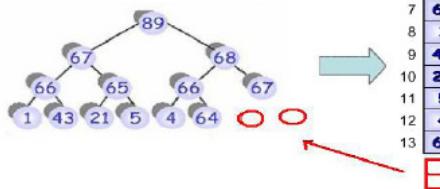
Se  $x$  è un qualsiasi nodo dell'heap (ad esclusione della radice) si ha

$$\text{key}(x) \leq \text{key}(\text{padre}(x))$$

Essendo un albero binario, un heap può essere memorizzato in un array con solite le relazioni:

$$\begin{aligned} \forall i \neq 1 \\ \forall i \leq \lfloor n/2 \rfloor \\ \forall i \leq \lfloor (n-1)/2 \rfloor \end{aligned}$$

$$\begin{aligned} \text{padre}(a_i) = a_{\lfloor i/2 \rfloor} \\ \text{figlio\_sx}(a_i) = a_{2i} \\ \text{figlio\_dx}(a_i) = a_{2i+1} \end{aligned}$$



```
int main()
{
    short n, i;
    short *heap; // dichiaro un puntatore ad un albero heap

    printf("Inserisci il numero di elementi da inserire nella heap: ");
    scanf("%hd", &n);
    /* Alloco un albero heap. N+1 perche' il primo elemento sara' vuoto*/
    heap = (short *)malloc(sizeof(short)*(n+1)); /* n+1, perche' ricavo figli partendo da v[1]*/
    if(!heap) exit("Errore di allocazione");
    heap[0]=-1; //primo valore array vuoto
    /* Inserisci elementi da cui costruire la heap*/
    for(i=1; i<=n; i++)
    {
        printf("Inserisci nodo: ");scanf("%hd", &heap[i]);
    }
    /* Costruisci heap */
    costruisci_heap(heap, n);
    printf("\nSINGOLI NODI DELLA HEAP.\n");
    printf("OGNI NODO HA UN VALORE MAGGIOR DEI FIGLI E LA RADICE E' IL VALORE MAX.\n\n");
    visualizza_heap(heap, n);
    return 0;
}
/*
costruisci heap: semplicemente scorre 'i' che punta al nodo corrente e mediante
heapify posizioniamo i figli nella giusta posizione tale che
padre(x)>x
*/
void costruisci_heap(short heap[], short n)
{
    short i=1;

    for(i=1; i<=n; i++)
        heapify(heap, i);
}
/*
heapify: Ha il compito di controllare se quel nodo e' piu' grande del padre.
```

```

    se lo e' scambiamo e passiamo all'eventuale padre successore dividendo i/2.
    Se il nodo e' piu' piccolo del padre, ci fermiamo perche' vale la proprieta'.
*/
void heapify(short heap[], short i)
{
    short tmp;

    /*
        Ho fatto direttamente heap[i]>heap[i/2] nel while, perche'
        appena la condizione e' falsa, esce dal ciclo perche' non devo continuare
        a confrontare gli altri padri superiori
        Arrivato alla radice, finisco
    */
    while(i!=1 && heap[i]>heap[i/2])
    {
        //scambia
        tmp = heap[i];
        heap[i] = heap[i/2];
        heap[i/2] = tmp;

        i=i/2; /* Passa al padre superiore */
    }
}
/*
visualizza_heap: stampiamo per ogni singolo il relativo padre e figli.
                    Il primo nodo che stampera' sara' la radice eventualmente
                    trovata.
*/
void visualizza_heap(short heap[], short n)
{
    short i;

    for(i=1; i<=n ; i++)
    {
        printf("Nodo: %3hd Padre: %3hd ", heap[i], heap[i/2]);
        /* Controllo se sono presenti dei figli */
        if(i*2<=n) printf("Figlio sinistro: %3hd ", heap[i*2]);
        if((i*2)+1<=n) printf("Figlio destro: %3hd", heap[(i*2)+1]);
        printf("\n\n");
    }
}

```

**OUTPUT:**

```

Inserisci il numero di elementi da inserire nella heap: 10
Inserisci nodo: 2
Inserisci nodo: 6
Inserisci nodo: 18
Inserisci nodo: 3
Inserisci nodo: 42
Inserisci nodo: 12
Inserisci nodo: 55
Inserisci nodo: 44
Inserisci nodo: 94
Inserisci nodo: 79

```

**SINGOLI NODI DELLA HEAP.**

**OGNI NODO HA UN VALORE MAGGIORE DEI FIGLI E LA RADICE E' IL VALORE MAX**

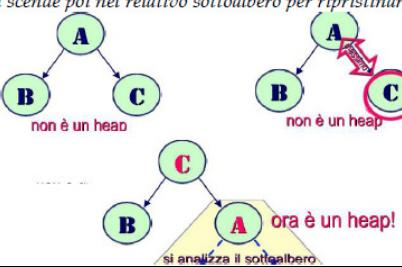
```

Nodo: 94 Padre: -1 Figlio sinistro: 79 Figlio destro: 42
Nodo: 79 Padre: 94 Figlio sinistro: 44 Figlio destro: 55
Nodo: 42 Padre: 94 Figlio sinistro: 6 Figlio destro: 12
Nodo: 44 Padre: 79 Figlio sinistro: 2 Figlio destro: 18
Nodo: 55 Padre: 79 Figlio sinistro: 3
Nodo: 6 Padre: 42
Nodo: 12 Padre: 42
Nodo: 2 Padre: 44
Nodo: 18 Padre: 44
Nodo: 3 Padre: 55

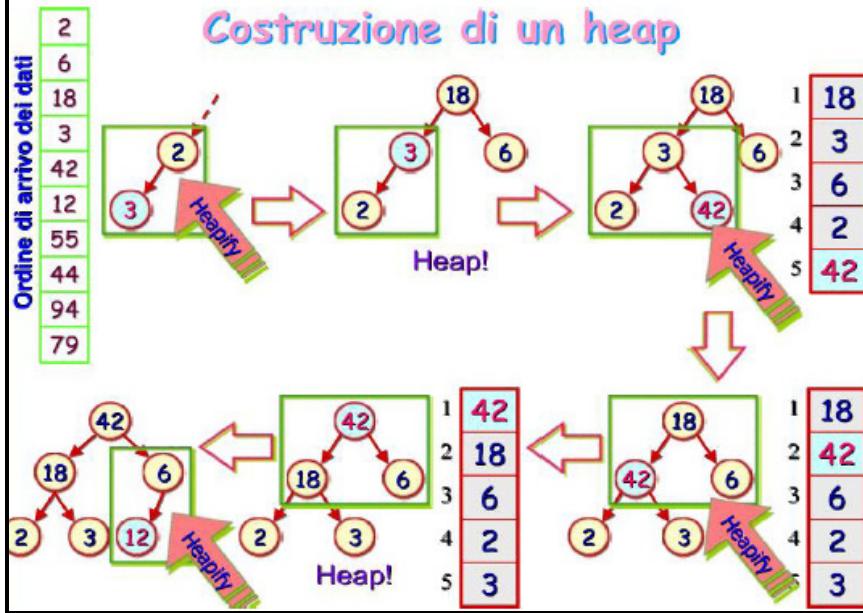
```

## Operazione base: procedura Heapify

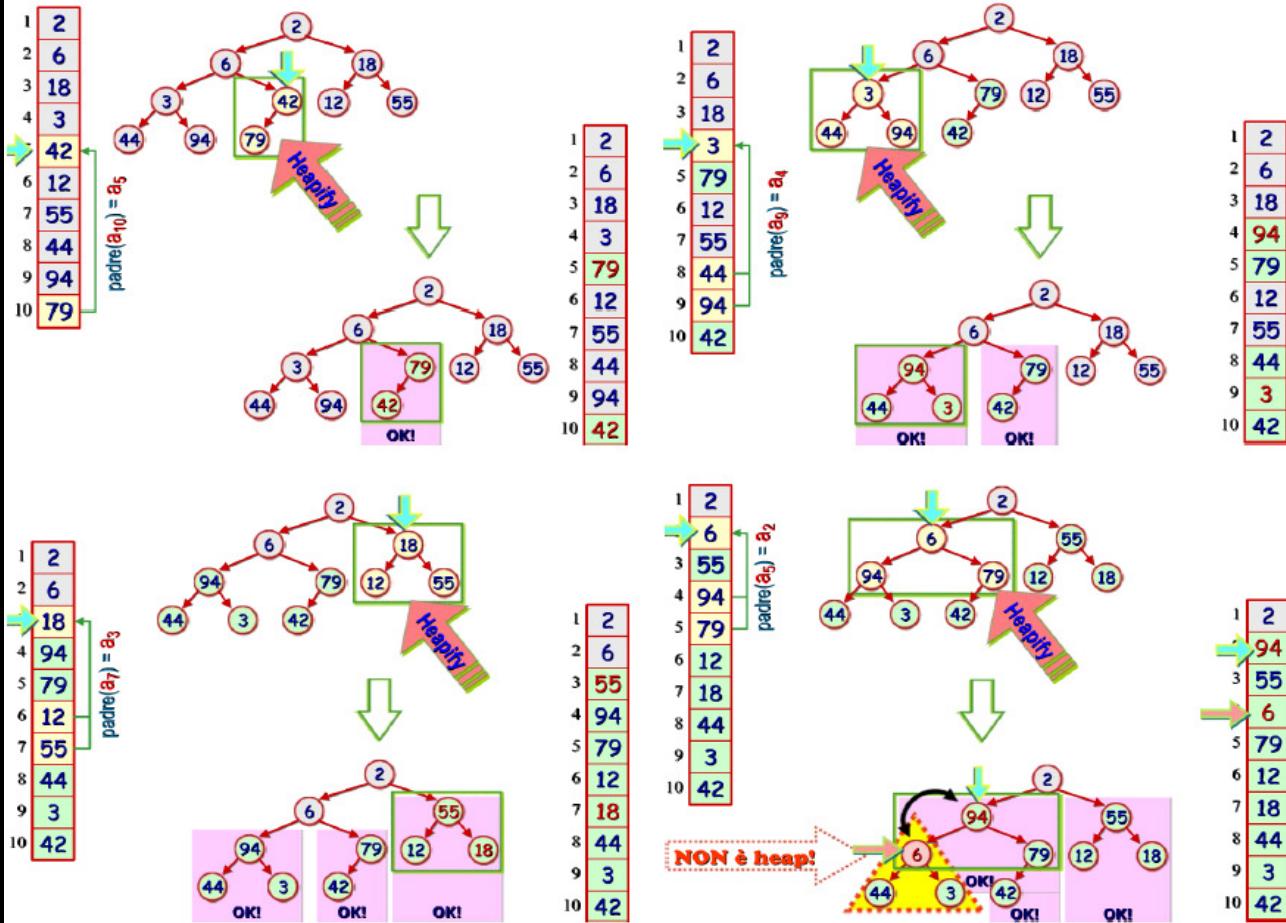
Tra i figli del nodo, si determina il figlio con chiave massima, successivamente si confronta il figlio di chiave massima col padre, ed eventualmente si scambiano i due nodi se non verificano la proprietà heap. Se è avvenuto lo scambio, si scende poi nel relativo sottoalbero per ripristinare la proprietà heap.



## Costruzione di un heap



## Come trasformare un albero binario in un heap?



## ESERCIZIO 54 [LIV.3]

```
/*
[liv.3] Scrivere function C iterativa per la trasformazione in un heap di un albero binario
rappresentato
mediante array.
*/

#include <stdio.h>
#include <stdlib.h>
void costruisci_heap(short heap[], short n);
short heapify(short heap[], short i, short n);
void visualizza_heap(short heap[], short n);
short is_foglia(short heap[], short i, short n);
void scambia(short *p, short *q);

/*
CENNI TEORICI
Un heap è una struttura dati gerarchica. In particolare è un albero binario completo o quasi
completo in cui ogni nodo verifica la proprietà dell'heap:
key(X) <= key(padre(X))
HEAPIFY
Tra i figli del nodo, si determina il figlio con chiave massima, successivamente si confronta il
figlio di chiave massima col padre, ed eventualmente si scambiano i due nodi se non verificano
la proprietà heap.
Se è avvenuto lo scambio, si scende poi nel relativo sottoalbero per ripristinare la proprietà
heap.
*/
int main()
{
    short n, i;
    short *heap; //dichiaro un puntatore ad un albero heap

    printf("Inserisci il numero di elementi da inserire nella heap: ");
    scanf("%hd", &n);
    /* Alloco un albero heap. N+1 perche' il primo elemento sara' vuoto*/
    heap = (short *)malloc(sizeof(short)*(n+1)); /* n+1, perche' ricavo figli partendo da v[1]*/
    if (!heap) exit("Errore di allocazione");
    heap[0]=-1; //primo valore array vuoto

    /* Inserisci elementi da cui costruire la heap*/
    for(i=1; i<=n; i++)
    {
        printf("Inserisci nodo: ");scanf("%hd", &heap[i]);
    }

    /* Costruisci heap */
    costruisci_heap(heap, n);
    /* visualizza heap */
    printf("\nSINGOLI NODI DELLA HEAP.\n");
    printf("OGNI NODO HA UN VALORE MAGGIORE DEI FIGLI E LA RADICE E' IL VALORE MAX.\n\n");
    visualizza_heap(heap, n);

    return 0;
}

/*
costruisci heap: In questo caso, 'i' punta ai padri.
Heapify opera mediante i figli del padre[i].
Il ciclo principale termina quando i=0 perche' la radice
deve essere considerata per l'heapify dato che avra' dei figli
NB 'i' SCORRE I PADRI.
*/
void costruisci_heap(short heap[], short n)
{
    /* Partiamo dal padre dell'ultima foglia perche' le foglie per definizione gia' sono heap
       i scorrera' i vari padri di cui controlleremo i relativi figli */
    short i=n/2;

    short esito, esito_foglia, j=0;
```

```

/* Fin quando non si arriva oltre la radice */
while(i>=1)
{
    /* effettua un heapify tra heap[i] (il padre) e i figli ( heap[sx] e heap[dx] ) */
    esito = heapify(heap, i, n);
    j=esito; /* contengono l'indice di dove avviene lo scambio. Tale indice ci serve per
               scendere nel relativo sottoalbero perche' avendo effettuato lo scambio,
               dobbiamo verificare se abbiamo violato nel sotto albero le proprieta'
               dell'heap.
               SE NON E' AVVENUTO LO SCAMBIO, VIENE SEGNALATO DA ESITO CON -1 */

    /* Quindi se avviene uno scambio, effettuiamo le stesse operazione nel sottoalbero
       in cui era avvenuto lo scambio.
       Cicliamo fin quando non abbiamo una foglia o non avviene uno scambio */
    while(esito>=0 && is_foglia(heap, j, n))
    {
        esito = heapify(heap, j, n);
        j=esito;
    }

    i--; //passa ai padri successivi
}
}

/*
   is_foglia: controlla semplicemente se quel nodo e' una foglia
*/
short is_foglia(short heap[], short i, short n)
{
    if(2*i>n && (2*i)+1>n )
        return 0; //non e' foglia
    /* Altrimenti e' una foglia se */
    return 1;
}

/*
   heapify: rispetto all'esercizio vecchio, abbiamo una variazione di heapify.
   controllo se ci sono figli per padre (heap[i]) e inoltre mi trovo il piu' grande tra il
padre
   e i figli.
*/
short heapify(short heap[], short i, short n)
{
    short tmp, scambio=0, figlio_sx=2*i, figlio_dx=(2*i)+1, maggiore;

    /* Controlla se sx non superi n e se il figlio sinistro sia maggiore del padre.
       Se si, conservati l'indice maggiore, ossia il figlio, altrimenti se il padre
       e' piu' grande, conserveremo l'indice del padre in maggiore*/
    if((figlio_sx<=n) && (heap[figlio_sx]>heap[i]))
        maggiore=figlio_sx;
    else
        maggiore=i;

    /* Ora confrontiamo il maggiore trovato con il figlio destro. Se la condizione e' valida
       avremo un nuovo maggiore, altrimenti rimane incavariato*/
    if((figlio_dx<=n) && (heap[figlio_dx]>heap[maggiore]))
        maggiore=figlio_dx;

    /* Se il padre e' piu' grande di uno dei 2 figli (quindi maggiore non e' l'indice padre)*/
    if(maggior != i)
    {
        /* Scambia i 2 nodi e ritorna l'indice di maggiore, ossia quello che ora
           contiene un valore minore di maggiore. tale ci serve per scendere nel relativo
           sotto albero */
        scambia(&heap[maggiore], &heap[i]);
        return maggiore;
    }

    /* Il padre e' un heap effettuando nessuno scambio */
    return -1;
}

```

```

visualizza_heap: stampiamo per ogni singolo il relativo padre e figli.
Il primo nodo che stampera' sara' la radice eventualmente
trovata.

*/
void visualizza_heap(short heap[], short n)
{
    short i;

    for(i=1; i<=n ; i++)
    {
        printf("Nodo: %3hd Padre: %3hd ", heap[i], heap[i/2]);
        /* Controllo se sono presenti dei figli */
        if(i*2<=n) printf("Figlio sinistro: %3hd ", heap[i*2]);
        if((i*2)+1<=n) printf("Figlio destro: %3hd", heap[(i*2)+1]);
        printf("\n\n");
    }
}

/*
scambia 2 elementi per indirizzo.
*/
void scambia(short *a,short *b)
{
    short tmp;
    tmp=*a;
    *a=*b;
    *b=tmp;
}

```

**OUTPUT:**

```

Inserisci il numero di elementi da inserire nella heap: 10
Inserisci nodo: 2
Inserisci nodo: 6
Inserisci nodo: 18
Inserisci nodo: 3
Inserisci nodo: 42
Inserisci nodo: 12
Inserisci nodo: 55
Inserisci nodo: 44
Inserisci nodo: 94
Inserisci nodo: 79

```

**SINGOLI NODI DELLA HEAP.**

**OGNI NODO HA UN VALORE MAGGIORE DEI FIGLI E LA RADICE E' IL VALORE MAX**

```

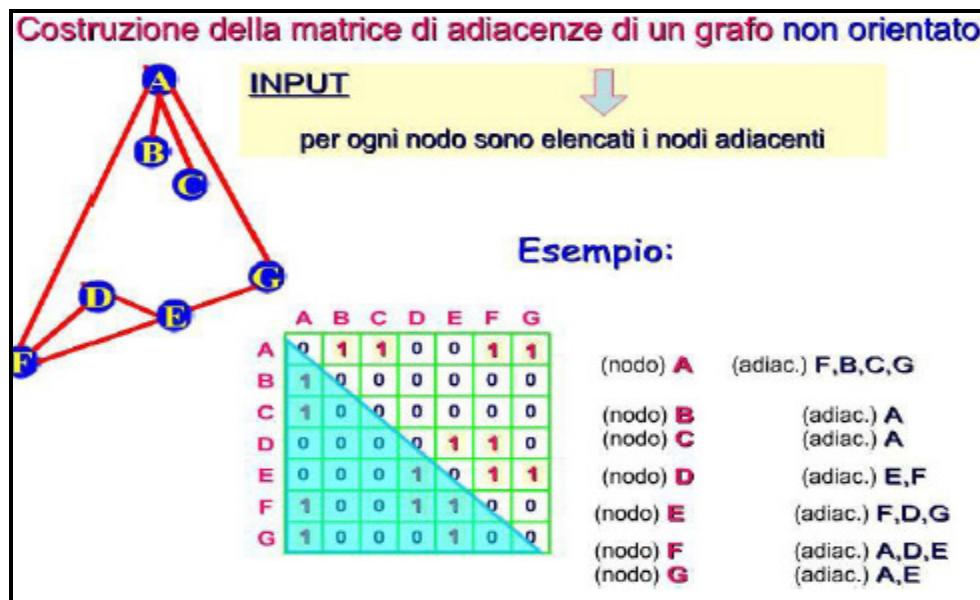
Nodo: 94 Padre: -1 Figlio sinistro: 79 Figlio destro: 55
Nodo: 79 Padre: 94 Figlio sinistro: 44 Figlio destro: 42
Nodo: 55 Padre: 94 Figlio sinistro: 12 Figlio destro: 18
Nodo: 44 Padre: 79 Figlio sinistro: 6 Figlio destro: 3
Nodo: 42 Padre: 79 Figlio sinistro: 2
Nodo: 12 Padre: 55
Nodo: 18 Padre: 55
Nodo: 6 Padre: 44
Nodo: 3 Padre: 44
Nodo: 2 Padre: 42

```

PS: Controlla gli esempi illustrativi mostrate nelle pagini precedenti

## ESERCIZIO 55 [LIV.1]

/\*\* [liv.1] Scrivere function C per la costruzione di un grafo non orientato mediante matrice di adiacenze: in input per ogni nodo sono specificati quelli adiacenti. Scegliendo in input un nodo, scrivere una function C che restituisca il suo grado.\*\*/



```
#include <stdio.h>
#include <stdlib.h>
#define MAX_NODI 20
void costruisci_grafo_non_orientato(short *Mat, short n_nodi);
void stampa_adiacenze(short *Mat, short n_nodi);
short inserito(short *Mat, short i, short j, short n);
void stampa_matrice(short *Mat, short n_nodi);
short calcola_grado_nodo(short *Mat, short n_nodi, short nodo_da_ric);
/* COSTRUZIONE GRAFO NON ORIENTATO MEDIANTE MATRICE DI ADIACENZE: un grafo non orientato e' rappresentato mediante una matrice di adiacenza. Essa e' una matrice che contiene le adiacenze tra il nodo RIGA e COLONNA(immagine sopra). Esso ha la particolarita' di essere una matrice simmetrica.
```

**ESEMPIO**

```

A B C
A 0 0 1
B 0 0 1
C 1 1 0
Tale matrice descrive un grafo non orientato      (A)-(C)-(B), dove c e' adiacente a A e B (ecco perche' gli 1).
Il grado di un nodo, indica in numero di adiacenze di quel nodo.*/

```

```

int main()
{
    short *Mat, n_nodi;
    char nodo_grado=0;

    puts ("*--- Costruzione di un GRAFO NON ORIENTATO con matrice di adiacenza ---* ");
    /* _____ inserisci numero nodi _____*/
    do{
        printf("Inserire il NUMERO di nodi del grafo: ");
        scanf("%hd",&n_nodi);fflush(stdin);
    }while(n_nodi<2 || n_nodi>MAX_NODI);
    //Alloca matrice vuota
    Mat = (short*)calloc(n_nodi*n_nodi,sizeof(short));

    costruisci_grafo_non_orientato(Mat, n_nodi);

    /* VISUALIZZIAMO LE ADIACENZE DI OGNI NODO */
    printf("\nVisualizza le varie ADIACENZE del GRAFO NON ORIENTATO \n");
    stampa_adiacenze(Mat, n_nodi);

    /* STAMPIAMO LA MATRICE, CHE SARA' SIMMETRICA */
    printf("\nVisualizza la MATRICE DI ADIACENZA(non orientata) SIMMETRICA \n");
    stampa_matrice(Mat, n_nodi);

    /* Calcola grado di un nodo */
    puts("\nInserisci il nodo[A...Z] di cui si vuole conosce il grado: ");
    scanf("%c", &nodo_grado);
    printf("Il grado del nodo [%c] e' %hd \n\n", nodo_grado, calcola_grado_nodo(Mat, n_nodi,

```

```

nodo_grado=65 );
    return 0;
}
/*
COSTRUISCI GRAFO: ha il compito di scorrere tutti i nodi e per i singoli assegnare le adiacenze
Dei nodi a cui sono collegato.
L'algoritmo puo' essere riassunto in:
    - Scorri nodi del grafo;
    - Per ogni nodo inseriamo il nodo adiacente.
    - l'inserimento viene effettuato anche al
        simmetrico
        poiche' e' un grafo non orientato */
void costruisci_grafo_non_orientato(short *Mat, short n_nodi)
{
    short i=0, j=0, n_adiacenze=0, flag=1, nodo_j;
    char nome_nodo;

    /* SCORRI I NODI DEL GRAFO (quelli in riga nella matrice di adiacenza) */
    for(i=0; i<n_nodi; i++)
    {
        /* Aggiungi il numero di adiacenze per quel nodo i-esimo */
        do{
            printf("\nInserire il NUMERO di adiacenze del nodo [%c] : ", 65+i);
            scanf("%hd",&n_adiacenze);fflush(stdin);
        }while(n_adiacenze<0 || n_adiacenze>n_nodi-1); /* posso avere massimo n_nodi-1 adiacenze,
                                                        perche' non devo considerare l'iesimo nodo
                                                        */

        /* Inseriamo il NODO ADIACENTE al NODO I-ESIMO */
        for(j=0; j<n_adiacenze; j++)
        {
            /* Controllo se e' un cappio (ripeti ciclo) e dopo aver inserito il nodo adiacente,
               andiamo a inserire '1' dove il nodo i-esimo e' adiacente rispetto al nodo aggiunto e
               al suo simmetrico (proprietà dei grafi non orientati) */
            flag=1; //flag per semplificare ciclo
            do
            {
                /* Iserisci un nodo adiacente */
                do{
                    printf("-Inserire nome[A...Z] del nodo %d adiacente ad [%c] : ", j, 65+i);
                    scanf("%c",&nome_nodo);fflush(stdin);
                }while(nome_nodo<'A' || nome_nodo>(65+n_nodi)); /* posso inserire nodi che vanno da A
                                                        ad A+n_nodi(guarda immagine) */

                nodo_j = nome_nodo-65;//equivalente del carattere sulle colonne
                /* Controlla se il nodo j coincide con il nodo i stesso (il cappio)*/
                if(i == nodo_j) printf("Tentativo di inserire un cappio; Riprova!\n");
                /* Se gia' e' inserito (ovviamente anche le simmetrie vale), sovrascrive */
                else if(inserito(Mat, i, nodo_j, n_nodi))
                {
                    puts("ATTENZIONE: adiacenza gia' inserita o presente per la simmetria\n");
                    flag=0;
                }
                // else ok, esco dal ciclo e posso inserirlo
                else flag=0;
            }while(flag==1);

            /* INSERISCI ADIACENZE PER QUEL NODO E PER IL SUO SIMMETRICO */
            *(Mat+n_nodi*i+nodo_j)=1; //Mat[i][j], inseriamo adiacenza
            *(Mat+n_nodi*nodo_j+i)=1; //Mat[j][i], inseriamo adiacenza al suo corrispondente
                                         simmetrico
            /*-----*/
        }
    }
}

/* grado nodo: calcola il grado di adiacenza di un nodo dato.
   nodo da ricercare e' l'indice di riga dove devo scorrere le colonne e contare le
   adiacenze per quel nodo.*/
short calcola_grado_nodo(short *Mat, short n_nodi, short nodo_da_ric)
{
    int j, cnt=0; //scorro le colonne, le righe le tengo bloccate da nodo da ric
    for(j=0; j<n_nodi; j++)
        if(*(Mat+n_nodi*nodo_da_ric+j)==1) cnt++;
    return cnt;
}

/* stampa adiacenze */

```

```

void stampa_adiacenze(short *Mat, short n_nodi)
{
    short i, j;

    for(i=0; i<n_nodi; i++)
    {
        printf("NODI ADIACENTI A [%c]: ", 65+i);
        for(j=0; j<n_nodi; j++)
        {
            if(inserito(Mat, i, j, n_nodi)) printf("[%c] ", j+65);
        }
        printf("\n");
    }
}
/* inserito: controllo se e' gia' stata effettuata un'adiacenza */
short inserito(short *Mat, short i, short j, short n)
{
    /* Se quel nodo e' gia' adiacente, restituisci un 1 ad indicare che gia' e' inserito */
    if(*Mat+i*n+j)==1) return 1;
    /* Nel caso non ci sia adiacenza */
    else return 0;
}
void stampa_matrice(short *Mat, short n_nodi)
{
    short i, j;
    for(i=0; i<n_nodi; i++)
    {
        for(j=0; j<n_nodi; j++)
            printf("%hd", *(Mat+n_nodi*i+j));
        printf("\n");
    }
}

```

**OUTPUT:**

----- Costruzione di un GRAFO NON ORIENTATO con matrice di adiacenza -----  
Inserire il NUMERO di nodi del grafo: 7

Inserire il NUMERO di adiacenze del nodo [A] : 4  
-Inserire nome[A...Z] del nodo 0 adiacente ad [A] : B  
-Inserire nome[A...Z] del nodo 1 adiacente ad [A] : C  
-Inserire nome[A...Z] del nodo 2 adiacente ad [A] : G  
-Inserire nome[A...Z] del nodo 3 adiacente ad [A] : F

Inserire il NUMERO di adiacenze del nodo [B] : 1  
-Inserire nome[A...Z] del nodo 0 adiacente ad [B] : A  
ATTENZIONE: adiacenza già inserita o presente per la simmetria.

Inserire il NUMERO di adiacenze del nodo [C] : 1  
-Inserire nome[A...Z] del nodo 0 adiacente ad [C] : A  
ATTENZIONE: adiacenza già inserita o presente per la simmetria.

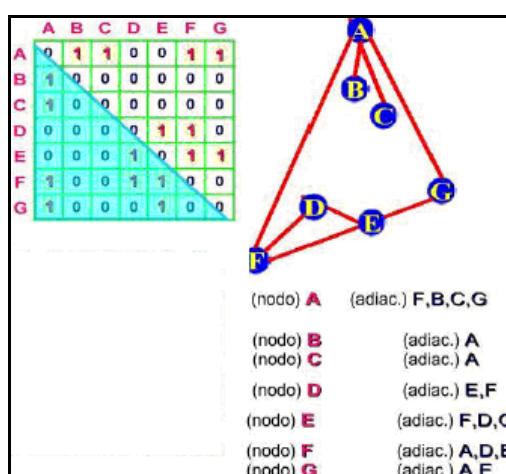
Inserire il NUMERO di adiacenze del nodo [D] : 2  
-Inserire nome[A...Z] del nodo 0 adiacente ad [D] : E  
-Inserire nome[A...Z] del nodo 1 adiacente ad [D] : F

Inserire il NUMERO di adiacenze del nodo [E] : 3  
-Inserire nome[A...Z] del nodo 0 adiacente ad [E] : D  
ATTENZIONE: adiacenza già inserita o presente per la simmetria.

...  
Visualizza le varie ADIACENZE del GRAFO NON ORIENTATO      Visualizza la MATRICE DI ADIACENZA<non orientata> SIMMETRICA  
NODI ADIACENTI A [A]: [B] [C] [F] [G]      0110011  
NODI ADIACENTI A [B]: [A]      1000000  
NODI ADIACENTI A [C]: [A]      1000000  
NODI ADIACENTI A [D]: [E] [F]      0000110  
NODI ADIACENTI A [E]: [D] [F] [G]      0001011  
NODI ADIACENTI A [F]: [A] [D] [E]      1001100  
NODI ADIACENTI A [G]: [A] [E]      1000100

Inserisci il nodo[A...Z] di cui si vuole conosce il grado:

F  
Il grado del nodo [F] e' 3



## ESERCIZIO 56 [LIV.1]

/\*\*  
 [l1v.1] Scrivere function C per la costruzione di un grafo orientato mediante matrice di adiacenze:  
 in input per ogni nodo sono specificati quelli raggiungibili. Scegliendo in input un nodo, scrivere  
 una function C che restituisca il numero degli archi uscenti e quello degli archi entranti. \*\*/



```
#include <stdio.h>
#include <stdlib.h>
#define MAX_NODI 20

void costruisci_grafo_orientato(short *Mat, short n_nodi);
void stampa_adiacenze(short *Mat, short n_nodi);
short inserito(short *Mat, short i, short j, short n);
void stampa_matrice(short *Mat, short n_nodi);
void calcola_archi_nodo(short *Mat, short n_nodi, short nodo_da_ric, short *, short *);

/*
COSTRUZIONE GRAFO ORIENTATO MEDIANTE MATRICE DI ADIACENZE: un grafo orientato e' rappresentato
mediante una matrice di adiacenza. Essa e' una matrice che contiene le adiacenze tra il nodo RIGA e
COLONNA(immagine sopra).
Esso ha la particolarita' di non essere una matrice simmetrica.
ESEMPIO
  A B C
A 0 0 0
B 0 0 0
C 1 1 0
Tale matrice descrive un grafo ORIENTATO      (A)<->(C)->(B), dove da C posso andare solo da A e B, ma
non viceversa.
*/
int main()
{
    short *Mat, n_nodi, nodo_intero, n_arco_entranti=0, n_arco_uscenti=0;
    char nodo=0;

    puts ("---- Costruzione di un GRAFO ORIENTATO con matrice di adiacenza ----");
    /* inserisci numero nodi ____*/
    do{
        printf("Inserire il NUMERO di nodi del grafo: ");
        scanf("%hd",&n_nodi);fflush(stdin);
    }while(n_nodi<2 || n_nodi>MAX_NODI);
    //Alloca matrice vuota
    Mat = (short*)calloc(n_nodi*n_nodi,sizeof(short));

    costruisci_grafo_orientato(Mat, n_nodi);

    /* VISUALIZZIAMO LE ADIACENZE DI OGNI NODO */
    printf("\nVisualizza le varie ADIACENZE del GRAFO ORIENTATO \n");
    stampa_adiacenze(Mat, n_nodi);

    /* STAMPIAMO LA MATRICE */
    printf("\nVisualizza la MATRICE DI ADIACENZA(orientata) SIMMETRICA \n");
    stampa_matrice(Mat, n_nodi);
}
```

```

/* Calcola grado di un nodo */
puts("\nInserisci il nodo[A...Z] di cui si vuole conosce gli archi entranti e uscenti: ");
scanf("%c", &nodo);
nodo_intero = nodo-65;
calcola_archi_nodo(Mat, n_nodi, nodo_intero, &n_arco_entranti, &n_arco_uscenti);
printf("Il numero di archi uscenti dal nodo [%c] e' %hd \n", nodo, n_arco_uscenti);
printf("Il numero di archi entranti nel nodo [%c] e' %hd \n\n", nodo, n_arco_entranti);
return 0;
}
/*
 * COSTRUISCI GRAFO: ha il compito di scorrere tutti i nodi e per i singoli assegnare le adiacenze
 * Dei nodi a cui sono collegato.
 * L'algoritmo puo' essere riassunto in: - Scorri nodi del grafo;
 *                                         - Per ogni nodo inseriamo il nodo raggiungibile.*/
void costruisce_grafo_orientato(short *Mat, short n_nodi)
{
    short i=0, j=0, n_adiacenze=0, flag=1, nodo_j;
    char nome_nodo;

    /* SCORRI I NODI DEL GRAFO (quelli in riga nella matrice di adiacenza)*/
    for(i=0; i<n_nodi; i++)
    {
        /* Aggiungi il numero di adiacenze per quel nodo i-esimo */
        do{
            printf("\nInserire il NUMERO di nodi raggiungibili da [%c] : ", 65+i);
            scanf("%hd", &n_adiacenze);fflush(stdin);
        }while(n_adiacenze<0 || n_adiacenze>n_nodi-1); /* posso avere massimo n_nodi-1 adiacenze,
                                                       perche' non devo considerare l'iesimo nodo
                                                       */
        /* Inseriamo il NODO RAGGIUNGIBILE DA NODO I-ESIMO */
        for(j=0; j<n_adiacenze; j++)
        {
            /* Controllo se e' un cappio (ripeti ciclo) e dopo aver inserito il nodo adiacente,
               andiamo a inserire '1' dove il nodo i-esimo e' adiacente rispetto al nodo aggiunto e
               al suo simmetrico (proprietà dei grafi non orientati) */
            flag=1; //flag per semplificare ciclo
            do
            {
                /* Inserisci un nodo raggiungibile */
                do{
                    printf("-Inserire nome[A...Z] del nodo %hd raggiungibile da [%c] : ", j, 65+i);
                    scanf("%c", &nome_nodo);fflush(stdin);
                }while(nome_nodo<'A' || nome_nodo>(65+n_nodi)); /* posso inserire nodi che vanno da A
                                                               ad A+n_nodi(guarda immagine) */

                nodo_j = nome_nodo-65;//equivalente del carattere sulle colonne
                /* Controlla se il nodo_j coincide con il nodo i stesso (il cappio)*/
                if(i == nodo_j) printf("Tentativo di inserire un cappio. Riprova!\n");
                /* Se gia' e' inserito */
                else if(inserito(Mat, i, nodo_j, n_nodi))
                {
                    puts("Adiacenza gia' presente. Sovrascrivo!\n");
                    flag=0;
                }
                // else ok, esco dal ciclo e posso inserirlo
                else flag=0;
            }while(flag==1);

            /* INSERISCI NODO RAGGIUNGIBILE PER QUEL NODO iesimo */
            *(Mat+n_nodi*i+nodo_j)=1; //Mat[i][j], inseriamo nodo per quel verso
            /*-----*/
        }
    }
}
/* calcola_archi_nodo: calcola il numero di archi entranti e uscenti per quel nodo */
void calcola_archi_nodo
(short *Mat, short n_nodi, short nodo_da_ric, short *n_arco_entranti, short *n_arco_uscenti)
{
    int j; //scorro le colonne, le righe le tengo bloccate da nodo da ric
    /* Calcola il numero di archi uscenti dal nodo da ric */
    for(j=0; j<n_nodi; j++)
    {
        if(*(Mat+n_nodi*nodo_da_ric+j)==1) (*n_arco_uscenti)+=1;
        if(*(Mat+n_nodi*j+nodo_da_ric)==1) (*n_arco_entranti)+=1;
    }
}

```

```

/* stampa adiacenze */
void stampa_adiacenze(short *Mat, short n_nodi)
{
    short i, j;

    for(i=0; i<n_nodi; i++)
    {   printf("NODI RAGGIUNGIBILI DA [%c]: ", 65+i);
        for(j=0; j<n_nodi; j++)
        {
            if(inserito(Mat, i, j, n_nodi)) printf("[%c] ", j+65);
        }
        printf("\n");
    }
}

/* inserito: controllo se e' gia' stata effettuata un'adiacenza */
short inserito(short *Mat, short i, short j, short n)
{
    /* Se quel nodo e' gia' adiacente, restituisci un 1 ad indicare che gia' e' inserito */
    if(*(Mat+i*n+j)==1) return 1;
    /* Nel caso non ci sia adiacenza */
    else return 0;
}

void stampa_matrice(short *Mat, short n_nodi)
{
    short i, j;
    for(i=0; i<n_nodi; i++)
    {
        for(j=0; j<n_nodi; j++)
            printf("%hd", *(Mat+n_nodi*i+j));
        printf("\n");
    }
}

```

**OUTPUT:**

---- Costruzione di un GRAFO ORIENTATO con matrice di adiacenza ----\*

Inserire il NUMERO di nodi del grafo: 7

Inserire il NUMERO di nodi raggiungibili da [A] : 7

-Inserire nome[A...Z] del nodo 0 raggiungibile da [A] : F  
 -Inserire nome[A...Z] del nodo 1 raggiungibile da [A] : C

Inserire il NUMERO di nodi raggiungibili da [B] : 1

-Inserire nome[A...Z] del nodo 0 raggiungibile da [B] : A

Inserire il NUMERO di nodi raggiungibili da [C] : 0

Inserire il NUMERO di nodi raggiungibili da [D] : 0

Inserire il NUMERO di nodi raggiungibili da [E] : 3  
 -Inserire nome[A...Z] del nodo 0 raggiungibile da [E] : D  
 -Inserire nome[A...Z] del nodo 1 raggiungibile da [E] : F  
 -Inserire nome[A...Z] del nodo 2 raggiungibile da [E] : G

Inserire il NUMERO di nodi raggiungibili da [F] : 2  
 -Inserire nome[A...Z] del nodo 0 raggiungibile da [F] : A  
 -Inserire nome[A...Z] del nodo 1 raggiungibile da [F] : D

Inserire il NUMERO di nodi raggiungibili da [G] : 1  
 -Inserire nome[A...Z] del nodo 0 raggiungibile da [G] : A

Visualizza le varie ADIACENZE del GRAFO ORIENTATO

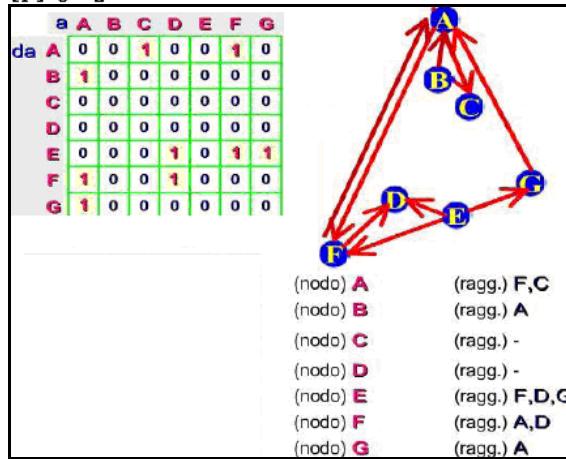
NODI RAGGIUNGIBILI DA [A]: [C] [F]  
 NODI RAGGIUNGIBILI DA [B]: [A]  
 NODI RAGGIUNGIBILI DA [C]:  
 NODI RAGGIUNGIBILI DA [D]:  
 NODI RAGGIUNGIBILI DA [E]: [D] [F] [G]  
 NODI RAGGIUNGIBILI DA [F]: [A] [D]  
 NODI RAGGIUNGIBILI DA [G]: [A]

Visualizza la MATRICE DI ADIACENZA(orientata)

0010010  
 1000000  
 0000000  
 0000000  
 0001011  
 1001000  
 1000000

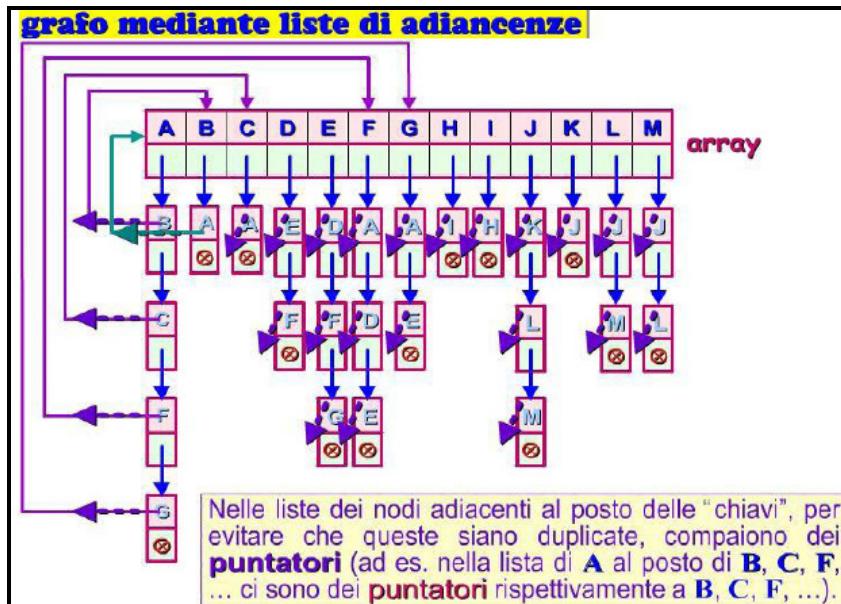
Inserisci il nodo[A...Z] di cui si vuole conosce gli archi entranti e uscenti:

F  
 Il numero di archi uscenti dal nodo [F] e' 2  
 Il numero di archi entranti nel nodo [F] e' 2



## ESERCIZIO 57 [LIV.2]

```
/** [liv.2] Scrivere function C per la costruzione di un grafo non orientato mediante liste di adiacenze:  
in input per ogni nodo sono specificati quelli adiacenti. **/
```



```
#include <stdio.h>
#include <stdlib.h>
#define MAX_NODI 20

/* NODI ADIACENTI PER QUEL VERTICE (Individuato dal vettore) */
struct NODO
{
    struct VERTICE *info; //punta ad un vertice (per non ridondare le info)
    struct NODO *next; //nodo agganciato
};

/* STRUCT DEL VETTORE CHE CONTERRA' I VERTICI (un campo info e un puntatore ai suoi adiacenti)*/
struct VERTICE
{
    char nome_vertice; //nome di quel vertice
    struct NODO *p_lista; //insieme di nodi adiacenti a quel vertice
};

/* Prototipi */
void crea_vertici(struct VERTICE *, short);
void costruisci_liste_adiacenza(struct VERTICE *, short);
void stampa_adiacenze(struct VERTICE *, short);
void inserisci_nodo_adiacente(struct VERTICE *a, short nodo_j, struct NODO **punt);
void inserisci_primo_nodo_adiacente(struct VERTICE *a, short i, short nodo_j, struct NODO
                                         **punt);
short cerca_arco(struct VERTICE *, short, short);

int main()
{
    /* Avere un vettore di vertici che puntano ai loro nodi rispettivi nodi adiacenti, ossia  
n liste */

    short n_nodi;
    struct VERTICE *p_array; //p_array e' un vettore di vertici che puntano ai loro adiacenti

    /* INSERISCI I VERTICI */
    do {
        printf("COSTRUZIONE LISTA DI ADIACENZE DI GRAFO NON ORIENTATO\n\n");
        printf("Introdurre il numeri di nodi 2<=n<=%d del grafo non orientato : ", MAX_NODI);
        scanf("%hd", &n_nodi);
    } while (n_nodi<1 || n_nodi>MAX_NODI);
    /* COSTRUISCI UN ARRAY DI VERTICI, CHE PUNTANO AI LORO NODI ADIACENTI */
    p_array=(struct VERTICE*) malloc(n_nodi*sizeof(struct VERTICE));
    /* Crea dei vertici [A...Z] */
    crea_vertici(p_array,n_nodi);
    /* COSTRUISCI LE LISTE DI ADIACENZE */
```

```

costruisci_lista_adiacenza(p_array,n_nodi);
stampa_adiacenze(p_array,n_nodi);
puts("\n");
return 0;
}

/*
CREA VERTICI: Assegna al vettore i vertici dove ognuno punta ai suoi nodi adiacenti
(inizialmente non puntano a nulla)
*/
void crea_vertici(struct VERTICE a[],short n_nodi)
{
    short i;
    for(i=0;i<n_nodi;i++)
    {
        a[i].nome_vertice=i+65; /* assegna vertice */
        a[i].p_lista=NULL; /* inizialmente puntano a nulla*/
    }
}
/* costruisci_lista_adiacenza: e' il cuore del programma poiche' organizza i nodi adiacenti.
COME FUNZIONA: - SCORRI I VERTICI
                  - PER OGUNO DI ESSO, INSERISCI I NODI ADIACENTI
Rispetto alla matrice di adiacenza, qui gestiamo tutto con delle liste: per ogni vertice
(elementi nel vettore) un puntatore ad una lista. Questa lista conterrà i nodi adiacenti
per quel vertice considerato.*/
void costruisci_lista_adiacenza(struct VERTICE a[],short n_nodi)
{
    short flag,i,j,n_adiacenze,nodo_j; //nodo_j ci serve per indirizzare le info ad un vertice
    char nome_nodo;
    struct NODO *punt; /* Se per quel vertice la lista è vuota, viene creata e egli
                          punterà al nodo inserito inizialmente.
                          Dopo il primo nodo, esso punterà all'ultimo NODO inserito,
                          in tal modo da inserire in coda.
                          Quando passeremo al prossimo vertice, ad egli verrà assegnata
                          il nuovo nodo adiacente a quel vertice */

    /* SCORRI I NODI DEL GRAFO (vertici) */
    for(i=0;i<n_nodi;i++)
    {
        /* Aggiungi il numero di adiacenze per quel nodo i-esimo */
        do{
            printf("\nQuanti nodi sono adiacenti a %c (0<=n<=%hd) ?: ",i+65,n_nodi-1);
            scanf("%hd",&n_adiacenze);fflush(stdin);
        }while(n_adiacenze<0 || n_adiacenze>(n_nodi-1));
        /* SCORRI I NODI ADIACENTI PER QUEL NODO */
        for(j=0;j<n_adiacenze;j++)
        {
            flag=1;
            do
            {
                /* Inserisci un nodo raggiungibile */
                do{
                    printf("-Inserire nome[A...%c] del nodo %hd ADIACENTE a [%c] : ",n_nodi+65-
                           1,j,65+i);
                    scanf("%c",&nome_nodo);fflush(stdin);
                }while(nome_nodo<'A' || nome_nodo>(65+n_nodi)); /* posso inserire nodi che vanno
                                                               da A ad A+n_nodi(guarda
                                                               immagine) */
                nodo_j = nome_nodo-65;//ricavo indice del nodo j adiacente a i
                /* Controlla se il nodo_j coincide con il nodo i stesso (il cappio)*/
                if(i == nodo_j)printf("Non sono previsti cappi!!!\n");
                /* Controlla se quel nodo è già adiacente al vertice */
                else if(cerca_arco(a,i,nodo_j))printf("Arco già esistente... riprovare!\n");
                // Tutto ok, continuo uscendo dal ciclo
                else flag=0;
            }while(flag);

            /* TROVATO IL GIUSTO NODO ADIACENTE, AGGANCIALO ALLA LISTA DI QUEL VERTICE I-ESIMO*/
            // Se è il primo nodo adiacente da inserire per quella lista
            if(j==0) inserisci_primo_nodo_adiacente(a, i, nodo_j, &punt);
            // Se non è il primo e punt punta all'ultimo inserito
            else inserisci_nodo_adiacente(a, nodo_j, &punt);
        }
    }
}

```

```

}

/* Inserisci Nodo Adiacente: Viene creato un nodo contenente le info puntate a nodo_j.
Effettuiamo i vari agganci utilizzando punt, che punta all'ultimo nodo inserito.
Il nuovo nodo, automaticamente, punta a NULL, dato che facciamo un inserimento in coda. */
void inserisci_nodo_adiacente(struct VERTICE *a, short nodo_j, struct NODO **punt)
{
    struct NODO *ptr;
    /* CREA NODO E INSERISCI INFO (l'indirizzo corrisponde alle informazione di un vertice) */
    ptr=(struct NODO*)malloc(sizeof(struct NODO));
    ptr->info = &a[nodo_j];
    /* AGGANCIA IL NODO PUNTATO DA PUNT A PTR*/
    (*punt)->next=ptr;
    ptr->next = NULL; //dato che sara' inserito in coda, puntera' sicuramente a NULL
    /* Dopo aver aggiunto al nodo, punt puntera' all'ultimo nodo */
    *punt=ptr;
}

/*
Inserisci Primo Nodo Adiacente: Viene creato un nodo contenente le info puntate a nodo_j.
L'aggancio e' tra il puntatatore del vertice al nuovo nodo e il nuovo nodo agganciato a
NULL.
punt da qui comincera' il suo lavoro, dato che successivamente inserira' da questo nodo in
poi.
*/
void inserisci_primo_nodo_adiacente(struct VERTICE *a, short i, short nodo_j, struct NODO
**punt)
{
    struct NODO *ptr;
    /* CREA NODO E INSERISCI INFO (l'indirizzo corrisponde alle informazione di un vertice) */
    ptr=(struct NODO*)malloc(sizeof(struct NODO));
    ptr->info = &a[nodo_j];
    /* AGGANCIA IL PRIMO ELEMENTO DEL VETTORE NEXT AL PRIMO NODO AD ESSO ADIACENTE*/
    a[i].p_lista=ptr;
    ptr->next=NULL; //Essendo il primo, questo primo nodo avra' come next NULL
    /* Dopo aver aggiunto al nodo, punt puntera' al primo nodo */
    *punt=ptr;
}

/*
Stampa adiacenze: Per ogni vertice a[i].p_lista, stampa le relative liste, ossia
i nodi adiacenti a tale vertice.
*/
void stampa_adiacenze(struct VERTICE a[], short n_nodi)
{
    short i;
    struct NODO* p;

    printf("\nMatrice delle adiacenze\n");
    for(i=0;i<n_nodi;i++)
    {
        p=a[i].p_lista;//considera il vertice i

        printf("\nNodo %c : ",i+65);
        /* Fin quando ci sono nodi adiacenti per quel vertice*/
        while(p!=NULL)
        {
            printf("[%c] ", p->info->nome_vertice); //stampa il carattere
            p=p->next; //avanza nella lista
        }
    }
}

/*
Cerca se quell'adiacenza e' stata gia' inserita per quella lista:
- vertice = indice vertice
- nuovo = indice del nodo da cercare
In base al vertice che passo, scorro la lista di quel vertice e confronto gli elementi
della lista con il carattere "nuovo" (passato come indice)
*/
short cerca_arco(struct VERTICE a[], short vertice, short nuovo)
{
    struct NODO *ptr;
    char car_nodo=new+65;

```

```

ptr=a[vertice].p_lista;
while(ptr!=NULL)
{
    /*Se l'informazione puntata da ptr e' uguale a car_nodo, significa che gia' e' presente */
    if(ptr->info->nome_vertice==car_nodo) return 1;
    ptr=ptr->next;
}
return 0; /*l'arco non orientato non Ã" gia' presente*/
}

```

**OUTPUT:****COSTRUZIONE LISTA DI ADIACENZE DI GRAFO NON ORIENTATO**

Introdurre il numeri di nodi  $2 \leq n \leq 20$  del grafo non orientato : 7

Quanti nodi sono adiacenti a A  $0 \leq n \leq 6$ ??: 4  
 -Inserire nome[A...G] del nodo 0 ADIACENTE a [A] : B  
 -Inserire nome[A...G] del nodo 1 ADIACENTE a [A] : C  
 -Inserire nome[A...G] del nodo 2 ADIACENTE a [A] : F  
 -Inserire nome[A...G] del nodo 3 ADIACENTE a [A] : G

Quanti nodi sono adiacenti a B  $0 \leq n \leq 6$ ??: 1  
 -Inserire nome[A...G] del nodo 0 ADIACENTE a [B] : A

Quanti nodi sono adiacenti a C  $0 \leq n \leq 6$ ??: 1  
 -Inserire nome[A...G] del nodo 0 ADIACENTE a [C] : A

Quanti nodi sono adiacenti a D  $0 \leq n \leq 6$ ??: 2  
 -Inserire nome[A...G] del nodo 0 ADIACENTE a [D] : F  
 -Inserire nome[A...G] del nodo 1 ADIACENTE a [D] : E

Quanti nodi sono adiacenti a E  $0 \leq n \leq 6$ ??: 3  
 -Inserire nome[A...G] del nodo 0 ADIACENTE a [E] : D  
 -Inserire nome[A...G] del nodo 1 ADIACENTE a [E] : F  
 -Inserire nome[A...G] del nodo 2 ADIACENTE a [E] : G

Quanti nodi sono adiacenti a F  $0 \leq n \leq 6$ ??: 3  
 -Inserire nome[A...G] del nodo 0 ADIACENTE a [F] : A  
 -Inserire nome[A...G] del nodo 1 ADIACENTE a [F] : D  
 -Inserire nome[A...G] del nodo 2 ADIACENTE a [F] : E

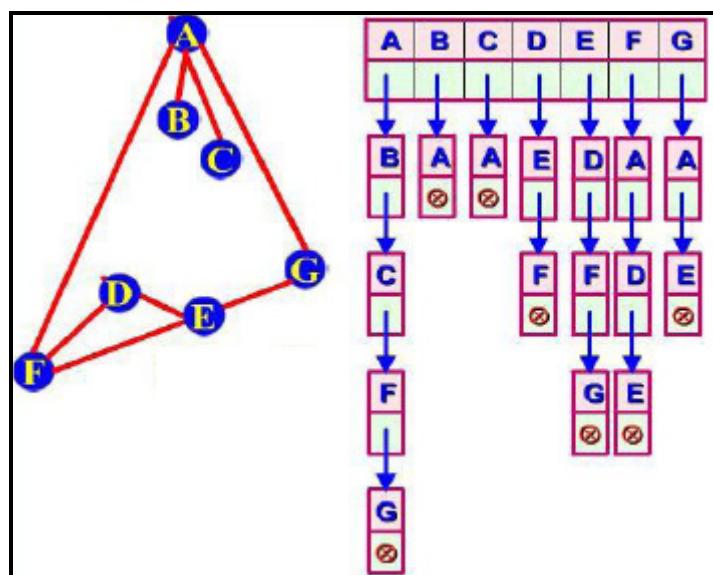
Quanti nodi sono adiacenti a G  $0 \leq n \leq 6$ ??: 2  
 -Inserire nome[A...G] del nodo 0 ADIACENTE a [G] : A  
 -Inserire nome[A...G] del nodo 1 ADIACENTE a [G] : E

**Matrice delle adiacenze**

```

Nodo A : [B] [C] [F] [G]
Nodo B : [A]
Nodo C : [A]
Nodo D : [F] [E]
Nodo E : [D] [F] [G]
Nodo F : [A] [D] [E]
Nodo G : [A] [E]

```



## ESERCIZIO 58 [LIV.2]

```
/**[liv.2] Scrivere function C per la visita in ordine anticipato (pre-order) di un albero
qualsiasi.**/
#include <stdio.h>
#include <stdlib.h>
#define MAX_GRADO 4
#define MAX_LEN 8 //Grandezza dei nodi

/* STRUTTURA DOPO LA COSTRUZIONE */
typedef struct nodo_link
{
    char dato;
    short grado; /* Numero di figli per quel nodo */
    short figlio[MAX_GRADO]; /* Gli indici ai figli. Punta al iesimo figlio */
    short id; /* Contiene indice del nodo */
}NODO_LINK;

/* Prototipi */
void pop(NODO_LINK *, NODO_LINK [], int *);
void push(NODO_LINK , NODO_LINK [], int *);
void visita_albero_generico_ordine_anticipato(NODO_LINK albero[], short radice);

/** NB: E' stato ripreso l'esercizio 47 per la costruzione di un albero generico. **/
int main()
{
    /* Definiamo la struttura iniziale dell'albero, dove i numeri indicano l'indice del padre nel
vettore */
    NODO_LINK albero[MAX_LEN];

    /* COSTRUIAMO MANUALMENTE L'ALBERO GENERICO
    INSERIAMO LE INFO E I FIGLI DEI NODI*/
    albero[0].dato='C';albero[0].grado=3;albero[0].id=0;
    albero[0].figlio[0] = 1;albero[0].figlio[1] = 2;albero[0].figlio[2] = 3;
    albero[1].dato='D'; albero[1].grado=1; albero[1].id=1;
    albero[1].figlio[0] = 4;
    albero[2].dato='F';albero[2].grado=1;albero[2].id=2;
    albero[2].figlio[0] = 5;
    albero[3].dato='G';albero[3].grado=0;albero[3].id=3;
    albero[4].dato='H';albero[4].grado=0;albero[4].id=4;
    albero[5].dato='B';albero[5].grado=1;albero[5].id=5;
    albero[5].figlio[0] = 6;
    albero[6].dato='A';albero[6].grado=1;albero[6].id=6;
    albero[6].figlio[0] = 7;
    albero[7].dato='E';albero[7].grado=0;albero[7].id=7;
    /*-----*/

    /* Indice che individua la radice; La ricaveremo dalla costruzione dell'albero */
    short radice=0;
    /* STAMPA ALBERO GENERICO */
    printf("\nALBERO QUALSIASI INIZIALE\n");
    printf("      %c\n",albero[0].dato);
    puts("      / | \\" );
    puts("      / | \\" );
    printf("      %c %c %c\n",albero[1].dato,albero[2].dato,albero[3].dato);
    puts("      |   | ");
    puts("      |   | ");
    printf("      \t %c %c \n",albero[4].dato,albero[5].dato);
    puts("      \t   | ");
    printf("      %c \n",albero[6].dato);
    puts("      \t   | ");
    printf("      %c \n",albero[7].dato);
    /*-----*/

    printf("\nVISITA ORDINE ANTICIPATO PER ALBERO QUALSIASI o PREORDER \n");
    visita_albero_generico_ordine_anticipato(albero, radice);

    printf("\n");
    return 0;
}
/*
visita_albero_generico_ordine_anticipato: tale ha il compito di visitare un albero
generico per ordine anticipato (PREORDER):
    -Inserisco radice nello stack
    -estraggo nodo e visualo
    -inserisci nello stack gli indici dei
        suoi figli (ciclo for interno)
    -ripeti fin quando lo stack non diventa vuoto

```

```

*/
void visita_albero_generico_ordine_anticipato(NODO_LINK albero[], short radice)
{

    int head=-1; /* indice che punta allo stack*/
    int id, grado_nodo, k; /* i campi che estrapoliamo dalla pila*/
    char dato;
    NODO_LINK estratto; //struct figlio estratta
    NODO_LINK stack[MAX_LEN]; //Stack usato per inserire i figli

    /* Inseerisci radice nello stack */
    push(albero[radice], stack, &head);

    while (head!=-1) //fin quando lo stack non e' vuoto
    {
        /* ESTRAI NODO dalla testa dello STACK e togilo dallo stack (head--)*/
        /*ps non metto tutto in una struct perche' diventerebbe meno leggibile
        pop(&estratto, stack, &head);
        //Salva i campi della struct estratta, per migliorare la lettura del codice
        id=estratto.id;
        dato=estratto.dato;
        grado_nodo=estratto.grado;

        /* Visita nodo estratto */
        printf("[%c] ", dato);

        /*Inserisce i figli del nodo estratto nello stack.
        Per avere successivamente una visita in ordine anticipato, rimpiamo lo stack dal figlio
        piu' a destra a quello piu' a sinistra, cosA- verra' estratto per prima quello LIFO
        a sinistra (ecco perche' k parte da grado-1)*/
        for (k=grado_nodo-1; k>=0; k--)
        {
            /*Per ogni figlio, a partire da quello a destra, pusha il figlio k.
            albero[id].figlio[k] individua l'indice del figlio considerato, quindi
            albero[albero[id].figlio[k]] ricava la struttura figlio*/
            push(albero[albero[id].figlio[k]], stack, &head);
        }
    }

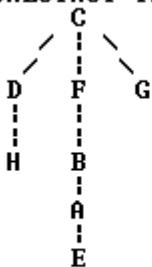
    /* push: inserisci i figli da estrarre successivamente */
    void push(NODO_LINK albero, NODO_LINK stack[], int *head)
    {
        (*head)++; //avanza indice per pushare
        /* Inserisci informazioni del nodo da pushare*/
        stack[*head] = albero;
    }

    /* pop: estrai un figlio precedentemente pushato */
    void pop(NODO_LINK *estratto, NODO_LINK stack[], int *head)
    {
        /*estrai informazioni*/
        *estratto = stack[*head];
        /* dopo l'estrazione, head si "svuota" */
        (*head)--;
    }
}

```

### OUTPUT:

#### ALBERO QUALESIASI INIZIALE



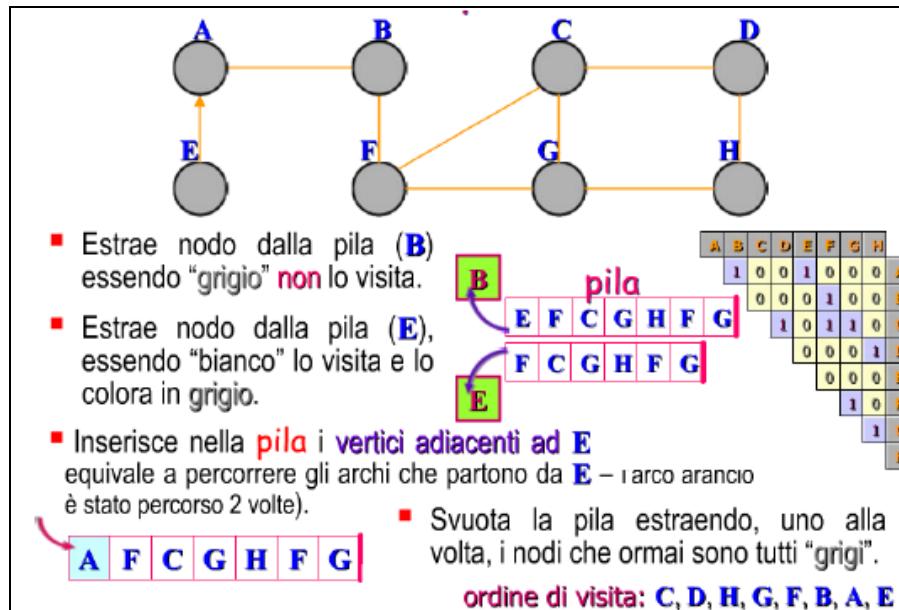
VISITA ORDINE ANTICIPATO PER ALBERO QUALESIASI o PREORDER  
 [C] [D] [H] [F] [B] [A] [E] [G]

### ESERCIZIO 59 [LIV.3]

/\*\* [liv.3] Scrivere function C per la visita di un grafo mediante l'algoritmo Depth First Search iterativo.

Applicare l'algoritmo che descrive il seguente Labirinto per determinare un cammino che parte dall'ingresso A e termina nell'uscita Q.

\*\*/



```
#include <stdio.h>
#include <stdlib.h>
#define MAX_INCROCI 27
#define MAXADIACENZE 4
#define LEN_STACK 30

/* dichiarazione del tipo INFO_FIELD, serve per le informazioni del nodo */
typedef struct
{
    char nome;
}INFO_FIELD;
/* dichiarazione della LISTA */
typedef struct grafo_labirinto
{
    INFO_FIELD info;
    short Adiacenze[MAXADIACENZE]; /* Indici che puntano ai n_grado adiacenti */
    short n_nodi_adiac;
    short visitato; /* bool */
}VERTICE_LABIRINTO;

/* PROTOTIPI */
void assegna_adiacenze(VERTICE_LABIRINTO Labirinto[], short *inizio_lab);
void DFS_visita_labirinto(VERTICE_LABIRINTO Labirinto[], short inizio_lab);
void Disegna_labirinto();
void Push(short Vet_Stack[], short Elemento, short *Testa);
void Pop(short Vet_Stack[], short *i_estratto, short *Testa);
/*
Viene creato un percorso seguendo una struttura reticolare semplice.
L'algoritmo principale applicato e' il DFS e in base a come carico nello stack
i figlio o come sono disposti in ordine i figli, avro' percorsi diversi.
*/
int main()
{
    /* Dichiariamo un array di VERTICI*/
    VERTICE_LABIRINTO Labirinto[MAX_INCROCI];
    short inizio_lab; /* Radice del grafo */

    puts("----- LABIRINTO PARTE DA 'A' PER USCIRE A 'Q' -----*\n");
    Disegna_labirinto();

    /* Assegna le adiacenze manualmente, dato che abbiamo uno specifico problema */
    assegna_adiacenze(Labirinto, &inizio_lab);

    /* Visualizza il CAMMINO per uscire dal labirinto, seguendo il DFS */
    puts("\nCAMMINO DA 'A' VERSO 'Q'\n");
    DFS_visita_labirinto(Labirinto, inizio_lab);
    puts("\n");
}
```

```

    return 0;
}

/*
DFS_visita_labirinto: Esso e' semplicemente l'algoritmo di DFS. Per poterlo utilizzare,
necessito di uno stack oppure potevo ricorrere alla ricorsione.
Si compone dei seguenti passaggi: - Visito radice e pusho i figli
                                    - Estraggo dallo stack e visito
                                    - inserisco nello stack i n nodi adiacenti
                                    - L'algoritmo si conclude o appena mi trovo
                                      nella soluzione, ossia si e' raggiunti Q,
                                      oppure ho visitato tutti e 26 nodi. Utilizzando
                                      un contatore, evitiamo inutili passaggi, dovuti a valore
                                      precedentemente pushati, ma gia' che sono
                                      stati visitati.

*/
void DFS_visita_labirinto(VERTICE_LABIRINTO Labirinto[], short inizio_lab)
{
    /* n_cnt conta i nodi visitati */
    short i_testa=-1, Stack[LEN_STACK], n_cnt=1, i, n_adiacenze=0;
    short i_estratto; /* Quando estraggo l'indice dalla pila, lo metto qui */

    /* VISITA LA RADICE DA CUI COMINCIA(inizio_labirinto)e PUSHO le adiacenze */
    printf("[%c] ", Labirinto[inizio_lab].info.nome); //Visita radice
    Labirinto[inizio_lab].visitato=1;
    //Inserisci le adiacenze nello stack per n nodi adiacenti ad esso
    for(i=0; i<Labirinto[inizio_lab].n_nodi_adiac; i++)
        Push(Stack, Labirinto[inizio_lab].Adiacenze[i], &i_testa);

    while(i_testa!=-1 && n_cnt<MAX_INCROCI)
    {
        /* ESTRAI NODO ADIACENTE */
        Pop(Stack, &i_estratto, &i_testa);

        /* SE IL NODO NON E' GIA' STATO VISITATO, VISITALO */
        if( Labirinto[i_estratto].visitato==0 )
        {
            //Visita nodo
            printf("[%c] ", Labirinto[i_estratto].info.nome);
            Labirinto[i_estratto].visitato=1;
            n_cnt++; /* Ad ogni estrazione, conto */

            /* SE SI E' ARRIVATI A VISITARE L'USCITA Q, Esci dalla VISITA! */
            if(Labirinto[i_estratto].info.nome=='Q')
                return; /* SIAMO ARRIVATI A Q, sarebbe inutile continuare perche' era
                           lo SCOPO RICHIESTO */

            /* INSERISCI ADIACENZE NELLO STACK per n nodi adiacenti al nodo estratto */
            n_adiacenze = Labirinto[i_estratto].n_nodi_adiac; //n adiacenze per quel nodo
            for(i=0; i<n_adiacenze; i++)
                Push(Stack, Labirinto[i_estratto].Adiacenze[i], &i_testa);
        }
    }

    /* Push: Inserire in testa dell'array, appunto, come lo stack. */
    void Push(short Vet_Stack[], short Elemento, short *Testa)
    {
        if(*Testa<LEN_STACK) //Se non ho superato il massimo dello stack
        {
            /* *Testa=*Testa+1 -> Vet_Stack=Elemento
               La mia testa punterà SEMPRE all'ULTIMO ELEMENTO inserito, quindi incrementando
               metterò l'elemento in testa o PUSHO */
            Vet_Stack[*(Testa)]=Elemento;
        }
        else {puts("\n----- NON INSERISCO NULLA, STACK PIENO! -----*\n");exit(1);}
    }

    /* Pop: Eliminare dalla testa dell'array. */
    void Pop(short Vet_Stack[], short *i_estratto, short *Testa)
    {
        if(*Testa>-1) //Se c'e' qualcosa
        {
            *i_estratto = Vet_Stack[*Testa];
            /* Decrementando l'indice dell'ultimo elemento inserito, appunto, non verrà
               piu' "visto" l'ultimo elemento inserito, nascondendolo*/

            (*Testa)--;
            //altrimenti "Stack vuoto" viene gestito nel main
        }
    }
}

```

```

        }
    else {puts("---- ERRORE Stack vuoto!! ----");exit(1);}
}

void Disegna_labirinto()
{
    puts("\tK---&---A---B---C ");
    puts("\t|     \\'     /     | ");
    puts("\t|     \\'     /     | ");
    puts("\tJ      D      E ");
    puts("\t|     / \\'     | ");
    puts("\t|     / \\'     | ");
    puts("\tY--- X     G ---F");
    puts("\t|     |     | ");
    puts("\t|     |     | ");
    puts("\tW     Z     H     I");
    puts("\t|     |     | ");
    puts("\t|     |     | ");
    puts("\tV--- U     L ---M");
    puts("\t|     \\'     /     | ");
    puts("\t|     \\'     /     | ");
    puts("\t|     \\'     /     | ");
    puts("\t|     Q     " );
}

/*
Assegna Adiacenze: Potremo usare sia una matrice di adiacenze che una lista di adiacenze,
ma per il problema interessato e per rapidita', sceglieremo di implementare una struttura
reticolare molto semplice (e' come se avessimo un albero generico)
*/
void assegna_adiacenze(VERTICE_LABIRINTO Labirinto[], short *inizio_lab)
{
    /* Da dove comincio a visitare? Da A individuato dall'indice 0 */
    *inizio_lab = 0;
    Labirinto[0].info.nome='A';
    Labirinto[0].n_nodi_adiac=2;
    Labirinto[0].Adiacenze[0] = 26; //Il vertice A Ã¨ collegato al vertice #
    Labirinto[0].Adiacenze[1] = 1; //Il vertice A Ã¨ collegato al vertice B
    Labirinto[0].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
    Labirinto[1].info.nome='B';
    Labirinto[1].n_nodi_adiac=3;
    Labirinto[1].Adiacenze[0] = 0; //Il vertice B Ã¨ collegato al vertice A
    Labirinto[1].Adiacenze[1] = 2; //Il vertice B Ã¨ collegato al vertice C
    Labirinto[1].Adiacenze[2] = 3; //Il vertice B Ã¨ collegato al vertice D
    Labirinto[1].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
    Labirinto[2].info.nome='C';
    Labirinto[2].n_nodi_adiac=2;
    Labirinto[2].Adiacenze[0] = 1; //Il vertice C Ã¨ collegato al vertice B
    Labirinto[2].Adiacenze[1] = 4; //Il vertice C Ã¨ collegato al vertice E
    Labirinto[2].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
    Labirinto[3].info.nome='D';
    Labirinto[3].n_nodi_adiac=4;
    Labirinto[3].Adiacenze[0] = 26; //Il vertice D Ã¨ collegato al vertice #
    Labirinto[3].Adiacenze[1] = 1; //Il vertice D Ã¨ collegato al vertice B
    Labirinto[3].Adiacenze[2] = 23; //Il vertice D Ã¨ collegato al vertice X
    Labirinto[3].Adiacenze[3] = 6; //Il vertice D Ã¨ collegato al vertice G
    Labirinto[3].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
    Labirinto[4].info.nome='E';
    Labirinto[4].n_nodi_adiac=2;
    Labirinto[4].Adiacenze[0] = 2; //Il vertice E Ã¨ collegato al vertice C
    Labirinto[4].Adiacenze[1] = 5; //Il vertice E Ã¨ collegato al vertice F
    Labirinto[4].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
    Labirinto[5].info.nome='F';
    Labirinto[5].n_nodi_adiac=3;
    Labirinto[5].Adiacenze[0] = 4; //Il vertice F Ã¨ collegato al vertice E
    Labirinto[5].Adiacenze[1] = 6; //Il vertice F Ã¨ collegato al vertice G
    Labirinto[5].Adiacenze[2] = 8; //Il vertice F Ã¨ collegato al vertice I
    Labirinto[5].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
    Labirinto[6].info.nome='G';
    Labirinto[6].n_nodi_adiac=3;
    Labirinto[6].Adiacenze[0] = 3; //Il vertice G Ã¨ collegato al vertice D
    Labirinto[6].Adiacenze[1] = 5; //Il vertice G Ã¨ collegato al vertice F
    Labirinto[6].Adiacenze[2] = 7; //Il vertice G Ã¨ collegato al vertice H
    Labirinto[6].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
    Labirinto[7].info.nome='H';
    Labirinto[7].n_nodi_adiac=3;
}

```

```

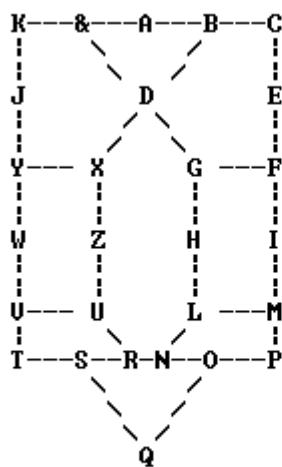
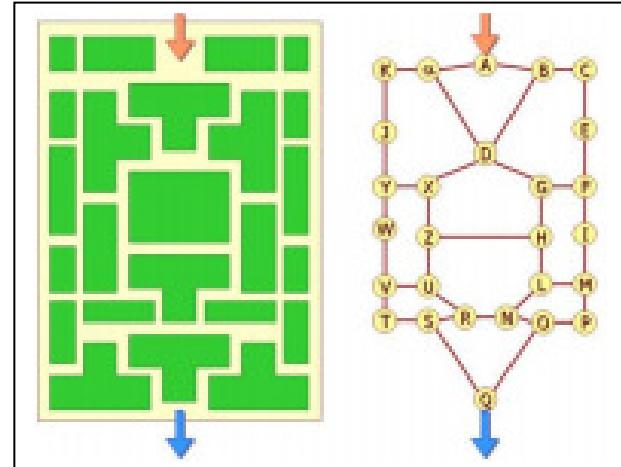
Labirinto[7].Adiacenze[0] = 6; //Il vertice H Ã¨ collegato al vertice G
Labirinto[7].Adiacenze[1] = 25; //Il vertice H Ã¨ collegato al vertice Z
Labirinto[7].Adiacenze[2] = 11; //Il vertice H Ã¨ collegato al vertice L
Labirinto[7].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
Labirinto[8].info.nome='I';
Labirinto[8].n_nodi_adiac=2;
Labirinto[8].Adiacenze[0] = 5; //Il vertice I Ã¨ collegato al vertice F
Labirinto[8].Adiacenze[1] = 12; //Il vertice I Ã¨ collegato al vertice M
Labirinto[8].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
Labirinto[9].info.nome='J';
Labirinto[9].n_nodi_adiac=2;
Labirinto[9].Adiacenze[0] = 10; //Il vertice J Ã¨ collegato al vertice K
Labirinto[9].Adiacenze[1] = 24; //Il vertice J Ã¨ collegato al vertice Y
Labirinto[9].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
Labirinto[10].info.nome='K';
Labirinto[10].n_nodi_adiac=2;
Labirinto[10].Adiacenze[0] = 26; //Il vertice K Ã¨ collegato al vertice #
Labirinto[10].Adiacenze[1] = 9; //Il vertice K Ã¨ collegato al vertice J
Labirinto[10].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
Labirinto[11].info.nome='L';
Labirinto[11].n_nodi_adiac=3;
Labirinto[11].Adiacenze[0] = 7; //Il vertice L Ã¨ collegato al vertice H
Labirinto[11].Adiacenze[1] = 12; //Il vertice L Ã¨ collegato al vertice M
Labirinto[11].Adiacenze[2] = 13; //Il vertice L Ã¨ collegato al vertice N
Labirinto[11].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
Labirinto[12].info.nome='M';
Labirinto[12].n_nodi_adiac=3;
Labirinto[12].Adiacenze[0] = 8; //Il vertice M Ã¨ collegato al vertice I
Labirinto[12].Adiacenze[1] = 11; //Il vertice M Ã¨ collegato al vertice L
Labirinto[12].Adiacenze[2] = 15; //Il vertice M Ã¨ collegato al vertice P
Labirinto[12].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
Labirinto[13].info.nome='N';
Labirinto[13].n_nodi_adiac=3;
Labirinto[13].Adiacenze[0] = 11; //Il vertice N Ã¨ collegato al vertice L
Labirinto[13].Adiacenze[1] = 14; //Il vertice N Ã¨ collegato al vertice O
Labirinto[13].Adiacenze[2] = 17; //Il vertice N Ã¨ collegato al vertice R
Labirinto[13].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
Labirinto[14].info.nome='O';
Labirinto[14].n_nodi_adiac=3;
Labirinto[14].Adiacenze[0] = 13; //Il vertice O Ã¨ collegato al vertice N
Labirinto[14].Adiacenze[1] = 15; //Il vertice O Ã¨ collegato al vertice P
Labirinto[14].Adiacenze[2] = 16; //Il vertice O Ã¨ collegato al vertice Q (uscita)
Labirinto[14].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
Labirinto[15].info.nome='P';
Labirinto[15].n_nodi_adiac=2;
Labirinto[15].Adiacenze[0] = 12; //Il vertice P Ã¨ collegato al vertice M
Labirinto[15].Adiacenze[1] = 14; //Il vertice P Ã¨ collegato al vertice O
Labirinto[15].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
Labirinto[16].info.nome='Q';
Labirinto[16].n_nodi_adiac=2;
Labirinto[16].Adiacenze[0] = 14; //Il vertice Q Ã¨ collegato al vertice O
Labirinto[16].Adiacenze[1] = 18; //Il vertice Q Ã¨ collegato al vertice S
Labirinto[16].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
Labirinto[17].info.nome='R';
Labirinto[17].n_nodi_adiac=3;
Labirinto[17].Adiacenze[0] = 13; //Il vertice R Ã¨ collegato al vertice N
Labirinto[17].Adiacenze[1] = 16; //Il vertice R Ã¨ collegato al vertice U
Labirinto[17].Adiacenze[2] = 18; //Il vertice R Ã¨ collegato al vertice S
Labirinto[17].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
Labirinto[18].info.nome='S';
Labirinto[18].n_nodi_adiac=3;
Labirinto[18].Adiacenze[0] = 17; //Il vertice S Ã¨ collegato al vertice N
Labirinto[18].Adiacenze[1] = 19; //Il vertice S Ã¨ collegato al vertice T
Labirinto[18].Adiacenze[2] = 16; //Il vertice S Ã¨ collegato al vertice Q (uscita)
Labirinto[18].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
Labirinto[19].info.nome='T';
Labirinto[19].n_nodi_adiac=2;
Labirinto[19].Adiacenze[0] = 21; //Il vertice T Ã¨ collegato al vertice V
Labirinto[19].Adiacenze[1] = 18; //Il vertice T Ã¨ collegato al vertice S
Labirinto[19].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
Labirinto[20].info.nome='U';
Labirinto[20].n_nodi_adiac=3;
Labirinto[20].Adiacenze[0] = 25; //Il vertice U Ã¨ collegato al vertice Z
Labirinto[20].Adiacenze[1] = 21; //Il vertice U Ã¨ collegato al vertice V
Labirinto[20].Adiacenze[2] = 17; //Il vertice U Ã¨ collegato al vertice R
Labirinto[20].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
Labirinto[21].info.nome='V';

```

```

Labirinto[21].n_nodi_adiac=3;
Labirinto[21].Adiacenze[0] = 22; //Il vertice V Ã¨ collegato al vertice W
Labirinto[21].Adiacenze[1] = 20; //Il vertice V Ã¨ collegato al vertice U
Labirinto[21].Adiacenze[2] = 19; //Il vertice V Ã¨ collegato al vertice T
Labirinto[21].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
Labirinto[22].info.nome='W';
Labirinto[22].n_nodi_adiac=2;
Labirinto[22].Adiacenze[0] = 24; //Il vertice W Ã¨ collegato al vertice Y
Labirinto[22].Adiacenze[1] = 21; //Il vertice W Ã¨ collegato al vertice V
Labirinto[22].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
Labirinto[23].info.nome='X';
Labirinto[23].n_nodi_adiac=3;
Labirinto[23].Adiacenze[0] = 3; //Il vertice X Ã¨ collegato al vertice D
Labirinto[23].Adiacenze[1] = 24; //Il vertice X Ã¨ collegato al vertice Z
Labirinto[23].Adiacenze[2] = 25; //Il vertice X Ã¨ collegato al vertice Y
Labirinto[23].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
Labirinto[24].info.nome='Y';
Labirinto[24].n_nodi_adiac=3;
Labirinto[24].Adiacenze[0] = 9; //Il vertice Y Ã¨ collegato al vertice J
Labirinto[24].Adiacenze[1] = 23; //Il vertice Y Ã¨ collegato al vertice X
Labirinto[24].Adiacenze[2] = 22; //Il vertice Y Ã¨ collegato al vertice W
Labirinto[24].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
Labirinto[25].info.nome='Z';
Labirinto[25].n_nodi_adiac=3;
Labirinto[25].Adiacenze[0] = 23; //Il vertice Z Ã¨ collegato al vertice X
Labirinto[25].Adiacenze[1] = 7; //Il vertice Z Ã¨ collegato al vertice H
Labirinto[25].Adiacenze[2] = 20; //Il vertice Z Ã¨ collegato al vertice U
Labirinto[25].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
Labirinto[26].info.nome='&';
Labirinto[26].n_nodi_adiac=3;
Labirinto[26].Adiacenze[0] = 0; //Il vertice # Ã¨ collegato al vertice A
Labirinto[26].Adiacenze[1] = 10; //Il vertice # Ã¨ collegato al vertice K
Labirinto[26].Adiacenze[2] = 3; //Il vertice # Ã¨ collegato al vertice D
Labirinto[26].visitato = 0; //1 - nodo visitato; 0 - nodo non visitato
}

```

**OUTPUT:****----- LABIRINTO PARTE DA 'A' PER USCIRE A 'Q' -----\*****CAMMINO DA 'A' VERSO 'Q'****[A] [B] [D] [G] [H] [L] [N] [R] [S] [Q]**

## ESERCIZIO 61 [LIV.1]

/\*\* [liv.1] Scrivere delle function C (rispettivamente iterativa e ricorsiva) per calcolare (con ricorsione sia lineare sia binaria) la somma delle componenti di un array. \*\*/

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

int somma_iterativa(int a[], int n);
int somma_ricorsiva_lin(int a[], int n);
int somma_ricorsiva_bin(int a[], int n);

int main()
{
    int n=5, a[]={1,2,3,4,5}, i=0;
    printf("***** ESEMPI DI SOMME *****\n");

    printf("ARRAY INIZIALE\n");
    for(i=0; i<n; i++)
        printf("[%d]", a[i]);

    printf("\n\n");
    printf("Somma iterativa : %d\n", somma_iterativa(a, n));
    printf("Somma ricorsiva lineare: %d\n", somma_ricorsiva_lin(a, n));
    printf("Somma iterativa binaria: %d\n", somma_ricorsiva_bin(a, n));

    return 0;
}
/* somma iterativa */
int somma_iterativa(int a[], int n)
{
    int i=0, acc=0;

    for(i=0; i<n; i++)
        acc+=a[i];
    return acc;
}
/* somma ricorsiva lineare: logica delle formule ricorrenti, ossia  $S(n)=a[n]+S(n-1)$ 
   Effettuo tanto autoattivazioni fin quando non arrivo ad un solo elemento (caso base)
   da lì' iniziero' a restituire il primo valore e pian piano si formerà la soluzione. */
int somma_ricorsiva_lin(int a[], int n)
{
    if(n==1) return a[0];
    return a[n-1]+somma_ricorsiva_lin(a, n-1);
}
/* somma ricorsiva binaria: qui si è utilizzata la logica del divide et impera, ossia si
   è diviso il problema a 2 a 2.
   Quando abbiamo la soluzione più semplice, ossia un solo elemento, lo restituiamo.
   ESEMPIO a={1,2,3}. Divido l'array in 2 parti: la prima parte sarà {1,2} mentre l'altra {3}.
   {1,2} non è ancora una soluzione banale, quindi divido quest'altra parte.
   {1,2} saranno soluzioni semplici, quindi ritorneranno nell'autoattivazione in cui avevamo
   composto la "formula" in cui ritorneranno i valori come 1+2. Ora il risultato di tale operazione
   ritornerà alla chiamata precedente come 3 + 3(quello inizialmente che aveva soluzione banale) e
   otterremo il risultato finale che tornerà al programma chiamante. */
int somma_ricorsiva_bin(int a[], int n)
{
    int mediano;
    mediano = (n-1)/2;

    if(n==1) return a[0];
    return somma_ricorsiva_bin(a, mediano+1)+somma_ricorsiva_bin( &(a[mediano+1]), n-mediano-1);
}
```

### OUTPUT:

\*\*\*\*\* ESEMPI DI SOMME \*\*\*\*\*  
ARRAY INIZIALE  
[1][2][3][4][5]

Somma iterativa : 15  
Somma ricorsiva lineare: 15  
Somma iterativa binaria: 15

## ESERCIZIO 62 [LIV.1]

```
/**[liv.1]Scrivere delle function C (rispettivamente iterativa e ricorsiva) per calcolare (con
ricorsione sia lineare sia binaria) la potenza intera  $x^n$  di un numero reale.**/
#include <stdio.h>
#include <stdlib.h>
int potenza_iterativa(int x, int n);
int potenza_ricorsiva_lin(int x, int n);
int potenza_ricorsiva_bin(int x, int inizio, int fine);

int main()
{
    int x=2, n=4; //2^4=16
    printf("----- ESEMPI DI POTENZE -----*\n");
    printf("Algoritmo che effettua l'operazione di: %d^%d\n\n", x,n);

    printf("Potenza iterativa : %d\n", potenza_iterativa(x, n));
    printf("Potenza ricorsiva lineare : %d\n", potenza_ricorsiva_lin(x, n));
    printf("Potenza ricorsiva binaria : %d\n", potenza_ricorsiva_bin(x, 0, n-1));

    return 0;
}
/* potenza iterativa */
int potenza_iterativa(int x, int n)
{
    int acc=1, i;

    for(i=1; i<=n; i++)
        acc=acc*x;
    return acc;
}
/* potenza ricorsiva lineare: utilizzo la formula ricorrente Pow(n)= x*Pow(n-1).
Il caso base rappresenta un numero elevato a 0, che viene sempre 1 e da qui' si vengono
a comporre tutte le soluzioni. */
int potenza_ricorsiva_lin(int x, int n)
{
    /* Caso base: n=0, ossia un numero elevato a 0 fa 1*/
    if(n==0) return 1;
    /* autoattivazione e composizione della formula ricorrente */
    return x*potenza_ricorsiva_lin(x, n-1);
}
/* potenza ricorsiva binaria: applico il divide et impera spezzettando le varie moltiplicazioni
che mi daranno la potenza finale.
Il caso base e' quando si ha la singola base.
PS fine===-1 sta ad indicare che sto tentando di effettuare una potenza di 0 (nel main faccio n-1)
e quindi restituisco 1 al main.
ESEMPIO 2^3 rappresenta (2*2*2). Con il divide et impera faccio (2*2)*(2), ma il primo termine
lo scompongo ancora diventando (2)*(2). Verra' restituito quindi 4 che si combinera' per 2 e
avremo 8. */
int potenza_ricorsiva_bin(int x, int inizio, int fine)
{
    /* Se nel main passo n-1 e n vale 0, faccio un controllo iniziale */
    if(fine!= -1)
    {
        int mediano;
        /* Caso base (inizio=fine, cioe' la soluzione elementare e' solo una base) */
        if(inizio>=fine) return x;

        mediano = (inizio+fine)/2;
        /* Auto attivazione */
        return potenza_ricorsiva_bin(x, inizio, mediano)*potenza_ricorsiva_bin(x, mediano+1, fine);
    }
    else return 1;
}
```

### OUTPUT:

----- ESEMPI DI POTENZE -----\*
Algoritmo che effettua l'operazione di: 2^4

Potenza iterativa : 16  
 Potenza ricorsiva lineare : 16  
 Potenza ricorsiva binaria : 16

### ESERCIZIO 63 [LIV.3]

```
/*
[liv.3] Scrivere due function C (rispettivamente iterativa e ricorsiva) per valutare un polinomio
mediante algoritmo di Horner.
**/

#include <stdio.h>
#include <stdlib.h>
double Horner_iterativo(double c[], double x, short grado);
double Horner_ricorsivo(double c[], double x, short grado);

int main()
{
    short grado, i;
    double *coef, x, horner_ite, horner_ric;

    puts("----- ALGORITMO DI HORNER ITERATIVO E RICORSIVO -----");
    printf("Inserire il grado del polinomio: ");
    scanf("%hd", &grado);
    fflush(stdin);
    /* ALLOCÒ DINAMICAMENTE ARRAY COEFFICIENTI */
    coef = (double *)malloc(sizeof(double)*grado);
    if(coef==NULL) { puts("ERRORE DI ALLOCAZIONE"); exit(1); }

    /* Inserimento dei coefficienti(Dal piu' alto al piu' piccolo) _____ */
    for(i=grado;i>=0;i--)
    {
        printf("\nInserisci il coefficiente di x^%hd: ",i);
        scanf("%lf", &coef[i]);
        fflush(stdin);
    }
    /* _____ INSERIRE IL PUNTO DI DOVE SI VUOLE CALCOLARE IL POLINOMIO _____ */
    printf("\nAdesso inserisci la x di P(x) per cui vuoi calcolare il polinomio: ");
    scanf("%lf", &x);

    /* CHIAMATA FUNCTION */
    horner_ite = Horner_iterativo(coef, x, grado);
    horner_ric = Horner_ricorsivo(coef, x, grado);

    /* _____ OUTPUT _____ */
    printf("\nHORNER ITERATIVO = %lf\n", horner_ite);
    printf("\nHORNER RICORSIVO = %lf\n", horner_ric);

    return 0;
}

/*
HORNER ITERATIVO: : effettuando un ciclo for per 'grado' volte e partendo dal
coefficiente di grado piÃ¹ basso, la funzione calcola il valore
del polinomio in un punto prefissato.

Se ho x^3, avro' ad esempio 4 coefficienti (il termine noto)

P(x) = c[N]+x(c[N-1]+x((( ... c[3]+x( c[2]+x (c[1]+c[0]*x) )))...))

*/
double Horner_iterativo(double c[], double x, short grado)
{
    short i=grado;
    long double ris=c[0]; //Soluzione base, primo coefficiente

    for(i=1;i<=grado;i++)
        ris = c[i]+x*ris;

    return ris;
}

/*
HORNER RICORSIVO: Seguendo la formula ricorrente, ricaviamo che effettuando
le autochiamate con coef[grado]+x(horner del grado precedente),
iniziera' a restituire x[0] che si combinerà il risultato e cosÃ¬
via, rispecchiando la forma di sotto: e' bottom up.

Se ho x^3, avro' ad esempio 4 coefficienti (il termine noto)

P(x) = c[N]+x(c[N-1]+x((( ... c[3]+x( c[2]+x (c[1]+c[0]*x) )))...))

```

```
/*
double Horner_ricorsivo(double c[],double x, short grado)
{
    if(grado==0) return c[0];
    return c[grado]+x*Horner_ricorsivo(c,x, grado-1);
}

OUTPUT:
----- ALGORITMO DI HORNER ITERATIVO E RICORSIVO -----
Inserire il grado del polinomio: 3
Inserisci il coefficiente di x^3: 2
Inserisci il coefficiente di x^2: 0
Inserisci il coefficiente di x^1: 5
Inserisci il coefficiente di x^0: 2
Adesso inserisci la x di P(x) per cui vuoi calcolare il polinomio: 6
HORNER ITERATIVO = 614.000000
HORNER RICORSIVO = 614.000000
```

## ESERCIZIO 64 [LIV.3]

```
/**[liv.3]Scrivere due function C (rispettivamente iterativa e ricorsiva) per visitare una lista
lineare, stampando le informazioni.**/

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

/* Semplice struct di dati */
typedef struct
{
    char nome[20];
    short eta;
}INFO_FIELD;
/* Dichiarazione della struttura del singolo nodo "struct NODO" che contiene i due campi:
 - informazione che si chiama 'info' ed e' di tipo INFO_FIELD dichiarato tramite una typedef
 - p_next ed e' un puntatore autoriferente
*/
struct NODO
{
    INFO_FIELD info;
    struct NODO *p_next;
};

//Prototipi delle function
struct NODO *crea_lista();
void insl_testa (INFO_FIELD Dati, struct NODO **head);
void insl_nodo (INFO_FIELD Dati, struct NODO **punt);
void visita_iterativa (struct NODO *head);
void visita_ricorsiva (struct NODO *head);
int cerca_nodo (struct NODO **punt, char key[]); //ricerca per nome

int main()
{
    struct NODO *head, *punt; /* Head=Punta solo la testa, Punt=un nodo qualunque */
    INFO_FIELD New_Dato;

    char Scelta, testo_key[20];
    int Esito;
    /* Crea lista. Head inizialmente punta a Head */
    head = crea_lista();
    do
    {
        printf("----- VISITA ITERATIVA E RICORSIVA DI UNA LISTA LINEARE -----*\n");
        printf("*[1] - Inserisci in testa alla lista *\n");
        printf("*[2] - Inserisci un elemento nel nodo successivo *\n");
        printf("*[3] - VISITA ITERATIVA *\n");
        printf("*[4] - VISITA RICORSIVA *\n");
        printf("*[5] - Esci dal programma *\n");
        printf("-----*\n");

        printf("\nScelta: ");
        scanf("%c", &Scelta); fflush(stdin); printf("\n");
        fflush(stdin);
        switch(Scelta)
        {
            case '1': printf("---- INSERIMENTO IN TESTA ---*\n");
                        printf("Inserisci nome: "); gets(New_Dato.nome);
                        printf("Inserisci eta': "); scanf("%hd", &New_Dato.eta);
                        insl_testa (New_Dato, &head); //Inserisci in testa
                        break;

            case '2': if(head!=NULL) //Ho almeno un elemento nella lista
            {
                printf("---- INSERIMENTO NEL NODO SUCCESSIVO ---*\n");
                printf("Inserisci nome del nodo a cui agganciare successivamente il nuovo
                      nodo: \n");
                gets(testo_key);
                punt=head; /* Punt punterà all'indirizzo di base inizialmente
                           per non perdere head*/
                Esito = cerca_nodo (&punt, testo_key); //cerca nodo

                if(Esito)//se nodo e' stato trovato, aggiungi
                {
                    printf("Inserire nome: "); gets(New_Dato.nome);
                    printf("Inserire eta': "); scanf("%hd", &New_Dato.eta);
                    insl_nodo (New_Dato, &punt); /* Inserisci dopo nodo trovato.
                                         Passiamo per indirizzo perche' punt
                }
            }
        }
    } while(Scelta != '5');
}
```

```

dovra' puntere al nuovo nodo */
    }
    else printf("Il nodo non e' stato trovato! \n");
}
else printf("Lista Vuota!!\n");
break;

case '3': if(head!=NULL) //Ho almeno un elemento nella lista
{
    printf("---- VISITA ITERATIVA ---*\n\n");
    visita_iterativa(head);
}
else printf("Lista vuota!!\n");
break;
case '4': if(head!=NULL) //Ho almeno un elemento nella lista
{
    printf("---- VISITA RICORSIVA ---*\n\n");
    visita_ricorsiva(head);
}
else printf("Lista vuota!!\n");
break;
}
printf("\n");
fflush(stdin);
}while(Scelta!='5');
return 0;
}

/*
visita_iterativa: semplicemente effettua la visita. Head e' un puntatore passato per valore
perche' la modifica del valore a cui punta, non mi interessa nel main e inoltre,
passando da head a head->next, nel main non perdero' head iniziale.
*/
void visita_iterativa(struct NODO *head)
{
/* Visita della lista */
/* Facciamo un do while perche' head==NULL la prima volta, gia' e' stato controllato */
do
{
    printf(" Nome: %s % \n Eta : %hd\n\n", head->info.nome, head->info.eta);
    head=head->p_next;
}while(head!=NULL);
}

/*
visita_ricorsiva: semplicemente effettua la visita. Ad ogni autoattivazione passiamo il
nodo successivo, fin tanto che non passiamo NULL, ossia la fine, dove la ricorsione
si interrompera'.
*/
void visita_ricorsiva(struct NODO *head)
{
/* soluzione banale */
if(head==NULL) return;
/* ALTRIMENTI VISITA ED AUTOATTIVATI*/
printf(" Nome: %s % \n Eta : %hd\n\n", head->info.nome, head->info.eta);
visita_ricorsiva(head->p_next);
}

/*
Cerca Nodo: In base ad un nome dato, tale funzione ricerca la lista contenente il medesimo
nome. La ricerca avviene tramite una visita e la restituzione dei parametri e' per indirizzo.
*/
int cerca_nodo (struct NODO **punt, char key[])
{
/* visita della lista*/
while(*punt!=NULL)
{ /* Se il nome del nodo e' uguale alla stringa cercata*/
    if(strcmp( (*punt)->info.nome, key)==0) // (**punt).info.testa.
        return 1; //Ok, nodo trovato; *punt punterà al nodo trovato
    *punt=(*punt)->p_next;
}
return 0; //se non trovi niente, segnala errore
}

/*
Inserisci in testa: Date le informazioni nuove, viene allocato un nuovo nodo e dopo
aver inserito le info, viene agganciato al primo nodo (indicato da Head) e per poi far puntare

```

```

    head a tale nodo nuovo .
    HEAD e' puntatore doppio, perche' dovremo modificare l'indirizzo a cui punta.
*/
void insl_testa (INFO_FIELD Dati, struct NODO **head)
{
    struct NODO *ptr; //Puntatore al NUOVO NODO
    /* Crea nodo e inserisci dati */
    ptr=(struct NODO *)calloc(1,sizeof (struct NODO)); //Richiediamo di allocare un nodo di grandezza
                                                       NODO
    ptr->info=Dati; //Inserisci dati; (*ptr).info
    /* Aggancia il new nodo al nodo in testa (me lo dice head dove sta)*/
    ptr->p_next = *head; //Head contiene l'indirizzo del nodo in testa
    /* Aggancia testa al new nodo */
    *head=ptr;
}
/* Inserisci in mezzo: Date le informazioni nuove, viene allocato un nuovo nodo e verra' aggiunto
dopo il nodo puntato da *punt. Quindi il nuovo nodo punterà al prossimo di punt e punt punterà
proprio al nuovo nodo.
Dopo aver aggiunto, PUNT punterà al nuovo nodo.*/
void insl_nodo(INFO_FIELD Dati, struct NODO **punt)
{
    struct NODO *ptr; //Puntatore al NUOVO NODO
    /* Crea nodo e inserisci dati */
    ptr=(struct NODO *)calloc(1,sizeof (struct NODO)); //Richiediamo di allocare un nodo di grandezza
                                                       NODO
    ptr->info=Dati; //Inserisci dati; (*ptr).info
    /* Aggancia il new nodo al successivo di punt */
    ptr->p_next= (*punt)->p_next;
    /* Aggancia il nodo considerato a punt al nuovo nodo */
    (*punt)->p_next = ptr;
    /* Punt ora lo faremo puntare al nuovo nodo, perche' sara' il corrente (facoltativo) */
    *punt=ptr;
}
/* Crea lista: Restituisce un punto iniziale della testa. */
struct NODO *crea_lista()
{
    struct NODO *head;
    head =NULL;
    return head;
}

```

**OUTPUT:**

```
***** VISITA ITERATIVA E RICORSIVA DI UNA LISTA LINEARE ****
*[1] - Inserisci in testa alla lista *
*[2] - Inserisci un elemento nel nodo successivo *
*[3] - VISITA ITERATIVA *
*[4] - VISITA RICORSIVA *
*[5] - Esci dal programma *
*****
```

**Scelta: 1****--- INSERIMENTO IN TESTA ---\***

Inserisci nome: A  
Inserisci eta': 9

**--- INSERIMENTO NEL NODO SUCCESSIVO ---\***

Inserisci nome del nodo a cui agganciare successivamente il nuovo nodo:  
A

Inserire nome: B  
Inserire eta': 2

**--- INSERIMENTO NEL NODO SUCCESSIVO ---\***

Inserisci nome del nodo a cui agganciare successivamente il nuovo nodo:  
B  
Inserire nome: C  
Inserire eta': 8

**--- VISITA ITERATIVA ---\***

Nome: A  
Eta : 9

Nome: B  
Eta : 2

Nome: C  
Eta : 8

**--- VISITA RICORSIVA ---\***

Nome: A  
Eta : 9

Nome: B  
Eta : 2

Nome: C  
Eta : 8

## ESERCIZIO 65(Ricorsivo) [LIV.2]

```
/** [liv.2] Scrivere una function C RICORSIVA per visitare un albero binario di ricerca (risp. In ordine anticipato, simmetrico e differito) stampando le informazioni.**/
/* 90% fa parte della seguente traccia
[liv.3] Scrivere function C iterativa per la costruzione di un albero binario di ricerca rappresentato mediante liste multiple.
*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
struct ALBERO
{
    int valore;
    struct ALBERO *figlio_dx;
    struct ALBERO *figlio_sx;
}ALBERO;

void costruisci_albero_binario(struct ALBERO **p_head, int n, int valori_input[]);
void visita_preorder(struct ALBERO *p_punt);
void visita_inorder(struct ALBERO *p_punt);
void visita_postorder(struct ALBERO *p_punt);
short foglia(struct ALBERO *p_punt);
void inizializza_albero(struct ALBERO **p_head);
/* IDEA: l'albero binario come sappiamo e' un albero che possiede al piu' 2 vie. Costruire un albero binario di ricerca significa dare dei valori e su di esso costruire una struttura ordinata di quei valori, in cui il nodo "piu a sinistra" e' il minimo, mentre il nodo piu' a destra e' il massimo. Per stampare in ordine, utilizziamo un accesso INORDER.*/
int main()
{
    /* in albero saranno disposti i nodi dando in input 'valori_input'*/
    struct ALBERO *p_head; //Puntatore alla testa dell'albero, ossia la radice
    int i=0, *valori_input, n;

    printf("Inserisci il numero di nodi dell'albero: ");
    scanf("%d", &n);
    valori_input = (int *)malloc(n*sizeof(int));
    if(!valori_input) exit("ERRORE DI ALLOCAZIONE");

    /* input valori da inserire */
    for(i=0; i<n; i++)
    {
        printf("\n");
        printf("Inserisci un valore: ");
        scanf("%d", &valori_input[i]);
    }

    /* STAMPA ARRAY DI INPUT */
    printf("\nVALORI NELL'ARRAY IN INPUT \n");
    for(i=0; i<n; i++)
        printf("%d ", valori_input[i]);
    printf("\n");

    /* Crea lista */
    inizializza_albero(&p_head);
    /* Costruisci albero binario */
    costruisci_albero_binario(&p_head, n, valori_input);

    /** _____ VISITE _____ ***/
    /* VISITA PREORDER PER STAMPARE IN ORDINE */
    printf("\nVISITA PREORDER DELL'ALBERO BINARIO COSTRUITO \n");
    visita_preorder(p_head);
    /* VISITA INORDER PER STAMPARE IN ORDINE */
    printf("\nVISITA INORDER DELL'ALBERO BINARIO COSTRUITO (valori in ordine) \n");
    visita_inorder(p_head);
    /* VISITA POST-ORDER PER STAMPARE IN ORDINE */
    printf("\nVISITA POSTORDER DELL'ALBERO BINARIO COSTRUITO \n");
    visita_postorder(p_head);

    printf("\n");
    return 0;
}

/*
    inizializza_albero: Assegno la testa dell'albero un indirizzo vuoto, dato che

```

```

    dobbiamo ancora comporlo.

*/
void inizializza_albero(struct ALBERO **p_head)
{
    *p_head=NULL;
    return;
}
/* costruisce_albero_binario: costruisce un albero binario mediante delle liste
multiple. */
void costruisce_albero_binario(struct ALBERO **p_head, int n, int valori_input[])
{
    struct ALBERO *ptr, *punt; /* puntatore a nodo nuovo e punt=puntatore che visita l'albero */
    int i=0;

    /* Crea radice */
    ptr=calloc(1, sizeof(struct ALBERO)); //Facendo calloc, i puntatori dx e sx punteranno a NULL
    if(!ptr) exit("ERRORE DI ALLOCAZIONE");
    ptr->valore = valori_input[i++]; //Inizializzo radice
    *p_head=ptr; //la nuova testa sara' la radice

    /* fin quando ho degli elementi da inserire */
    while(i<n)
    {
        punt = *p_head; // Parti dalla radice

        /* Crea nodo */
        ptr=calloc(1, sizeof(struct ALBERO)); //Facendo calloc, i puntatori dx e sx punteranno a NULL
        if(!ptr) exit("ERRORE DI ALLOCAZIONE");

        /* Controllo dove devo inserire tale nodo */
        /* CONTROLLO SE VA SOTTO ALBERO SINISTRO*/
        while(valori_input[i] <= punt->valore && punt->figlio_sx!=NULL )
            punt=punt->figlio_sx; //scendi al sottoalbero sinistro
        /* CONTROLLO SE VA SOTTO ALBERO DESTRO*/
        while(valori_input[i]>= punt->valore && punt->figlio_dx!=NULL )
            punt=punt->figlio_dx; //scendi al sottoalbero destro
        /* TROVATO IL SOTTOALBERO, VEDO SE E' FIGLIO SINISTRO O DESTRO*/
        if(valori_input[i] < punt->valore)
        {
            punt->figlio_sx=ptr; //Aggancia come figlio sinistro
            ptr->valore = valori_input[i]; //inserisci valore nel nodo
        }
        else
        {
            punt->figlio_dx=ptr; //Aggancia come figlio destro
            ptr->valore = valori_input[i]; //inserisci valore nel nodo
        }
        i++; //vai ai prossimi valori input
    }

}
/*
INORDER (ordine simmetrico): - sottoalbero sinistro
                            - radice
                            - sottoalbero destro
input: *p_punt= Puntatore all'albero. Richiama sempre sotto albero sinistro,
       fin quando non trova uno che non ha figli sinistri. In tal caso prendera'
       il nodo radice e si sposta al sottoalbero destro.*/
void visita_inorder(struct ALBERO *p_punt)
{
    if (foglia(p_punt)) /* controlla se e' foglia, se ci troviamo in un nodo nullo */
        return; //indica il ritorno al precedente processo lasciato in sospeso

    /* passiamo al padre del sottoalbero sinistro */
    visita_inorder(p_punt->figlio_sx);

    /*stampa il nodo corrente dell'albero*/
    //dopo che il sottoalbero sinistro e' finito, mi stampa la radice del sotto albero per
    //poi passare al sotto albero destro
    printf("%hd ", p_punt->valore);

    /* passiamo al padre del sottoalbero destro */
    visita_inorder(p_punt->figlio_dx);
}

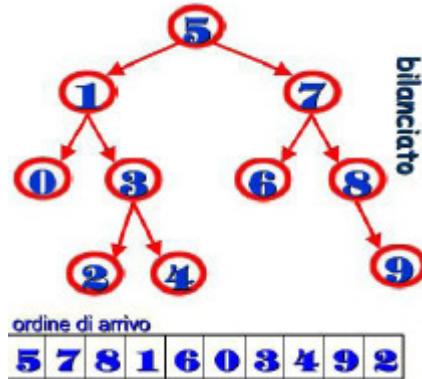
/*
PREORDER (ordine simmetrico): - sottoalbero sinistro

```

```

        - radice
        - sottoalbero destro
input: *p_punt= Visita il puntatore e chiama sottoalbero sinistro. Appena non ci sono piu' sotto
       alberi sinistri, visita quelli a destra*/
void visita_preorder(struct ALBERO *p_punt)
{
    if (foglia(p_punt)) /* controlla se e' foglia, se ci troviamo in un nodo nullo */
        return; //indica il ritorno al precedente processo lasciato in sospeso
    /*stampa il nodo corrente dell'albero*/
    //dopo che il sottoalbero sinistro e' finito, mi stampa la radice del sotto albero per
    //poi passare al sotto albero destro
    printf("%hd ", p_punt->valore);
    /* passiamo al padre del sottoalbero sinistro */
    visita_preorder(p_punt->figlio_sx);
    /* passiamo al padre del sottoalbero destro */
    visita_preorder(p_punt->figlio_dx);
}
/*POSTORDER (ordine simmetrico): - sottoalbero sinistro
   - radice
   - sottoalbero destro
input: *p_punt= Scendi al nodo piu' a sinistra e destro. Se non ha altri figli sx o dx visita
       il nodo*/
void visita_postorder(struct ALBERO *p_punt)
{
    if (foglia(p_punt)) /* controlla se e' foglia, se ci troviamo in un nodo nullo */
        return; //indica il ritorno al precedente processo lasciato in sospeso
    /* passiamo al padre del sottoalbero sinistro */
    visita_postorder(p_punt->figlio_sx);
    /* passiamo al padre del sottoalbero destro */
    visita_postorder(p_punt->figlio_dx);
    /*stampa il nodo corrente dell'albero*/
    //dopo che il sottoalbero sinistro e' finito, mi stampa la radice del sotto albero per
    //poi passare al sotto albero destro
    printf("%hd ", p_punt->valore);
}
/* Funzione che restituisce se quell'elemento e' una foglia.
Effettuando le chiamate ricorsive per il sotto albero a sinistra o destro, trovera
un nodo che punta a null e quindi significhera' che e' una foglia. Se lo e' torna
all'autoattivazione */
short foglia(struct ALBERO *p_punt)
{
    /* Se il nodo a cui ha acceduto punta a null, l'autoattivazione ritorna al programma
       chiamante per segnalare la foglia riguardante la precedente chiamate */
    //Se non e' una foglia
    if (p_punt!=NULL)
        return 0;
    /* se punta a null significa che il nodo che l'ha autoattivata era una foglia */
    return 1;
}
OUTPUT:
Inserisci il numero di nodi dell'albero: 10
Inserisci un valore: 5
Inserisci un valore: 7
Inserisci un valore: 8
Inserisci un valore: 1
Inserisci un valore: 6
Inserisci un valore: 0
Inserisci un valore: 3
Inserisci un valore: 4
Inserisci un valore: 9
Inserisci un valore: 2
VALORI NELL'ARRAY IN INPUT
5 7 8 1 6 0 3 4 9 2
VISITA PREORDER DELL'ALBERO BINARIO COSTRUITO
5 1 0 3 2 4 7 6 8 9
VISITA INORDER DELL'ALBERO BINARIO COSTRUITO <valori in ordine>
0 1 2 3 4 5 6 7 8 9
VISITA POSTORDER DELL'ALBERO BINARIO COSTRUITO
0 2 4 3 1 6 9 8 7 5

```



## ESERCIZIO 65(Iterativo) [LIV.2]

```
/** [liv.2] Scrivere una function C ITERATIVA per visitare un albero binario di ricerca (risp. In ordine anticipato, simmetrico e differito) stampando le informazioni. **/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define MAX_SIZE 30
struct ALBERO
{
    int valore;
    //int esiste;
    struct ALBERO *figlio_dx;
    struct ALBERO *figlio_sx;
};

void costruisci_albero_binario(struct ALBERO **p_head, int n, int valori_input[]);
void visita_preorder_Iterativa(struct ALBERO *p_punt);
void visita_inorder_Iterativa(struct ALBERO *p_punt);
void visita_postorder_Iterativa(struct ALBERO *p_punt);
void Push(struct ALBERO *Vet_Stack[], struct ALBERO *Indirizzo_nodo, short *Testa);
void Pop(short *Testa);

short foglia(struct ALBERO *p_punt);
void inizializza_albero(struct ALBERO **p_head);
/* IDEA: l'albero binario come sappiamo e' un albero che possiede al piu' 2 vie. Costruire un albero binario di ricerca significa dare dei valori e su di esso costruire una struttura ordinata di quei valori, in cui il nodo "piu a sinistra" e' il minimo, mentre il nodo piu' a destra e' il massimo. Per stampare in ordine, utilizziamo un accesso INORDER.*/
/** DALLA FORMA RICORSIVA POSSIAMO RICAVARE QUELLA ITERATIVA, USANDO UNO STACK ESPLICITO. */
int main()
{
    /* in albero saranno disposti i nodi dando in input 'valori_input'*/
    struct ALBERO *p_head; //Puntatore alla testa dell'albero, ossia la radice
    int i=0, *valori_input, n;

    printf("Inserisci il numero di nodi dell'albero: ");
    scanf("%d", &n);
    valori_input = (int *)malloc(n*sizeof(int));
    if(!valori_input) exit("ERRORE DI ALLOCAZIONE");

    /* input valori da inserire */
    for(i=0; i<n; i++)
    {
        printf("\n");
        printf("Inserisci un valore: ");
        scanf("%d", &valori_input[i]);
    }

    /* STAMPA ARRAY DI INPUT */
    printf("\nVALORI NELL'ARRAY IN INPUT \n");
    for(i=0; i<n; i++)
        printf("%d ", valori_input[i]);
    printf("\n");

    /* Crea lista */
    inizializza_albero(&p_head);
    /* Costruisci albero binario */
    costruisci_albero_binario(&p_head, n, valori_input);

    /** _____ VISITE _____ */
    /* VISITA PREORDER PER STAMPARE IN ORDINE */
    printf("\nVISITA PREORDER DELL'ALBERO BINARIO COSTRUITO \n");
    visita_preorder_Iterativa(p_head);
    /* VISITA INORDER PER STAMPARE IN ORDINE */
    printf("\nVISITA INORDER DELL'ALBERO BINARIO COSTRUITO (valori in ordine) \n");
    visita_inorder_Iterativa(p_head);
    /* VISITA POST-ORDER PER STAMPARE IN ORDINE */
    printf("\nVISITA POSTORDER DELL'ALBERO BINARIO COSTRUITO \n");
    visita_postorder_Iterativa(p_head);

    printf("\n");
    return 0;
}
/*
```

```

inizializza_albero: Assegno la testa dell'albero un indirizzo vuoto, dato che
dobbiamo ancora comporlo.
*/
void inizializza_albero(struct ALBERO **p_head)
{
    *p_head=NULL;
    return;
}
/*
costruisci_albero_binario: costruisce un albero binario mediante delle liste
multiple.
*/
void costruisci_albero_binario(struct ALBERO **p_head, int n, int valori_input[])
{
    struct ALBERO *ptr, *punt; /* ptr=puntatore a nodo nuovo e punt=puntatore che visita l'albero */
    int i=0;

    /* Crea radice */
    ptr=malloc(1, sizeof(struct ALBERO)); //Facendo malloc, i puntatori dx e sx punteranno a NULL
    if(!ptr) exit("ERRORE DI ALLOCAZIONE");
    ptr->valore = valori_input[i++]; //Inizializzo radice
    *p_head=ptr; //la nuova testa sara' la radice

    /* fin quando ho degli elementi da inserire */
    while(i<n)
    {
        punt = *p_head; // Parti dalla radice

        /* Crea nodo */
        ptr=malloc(1, sizeof(struct ALBERO)); //Facendo malloc, i puntatori dx e sx punteranno a NULL
        if(!ptr) exit("ERRORE DI ALLOCAZIONE");

        /* Controllo dove devo inserire tale nodo */
        /* CONTROLLO SE VA SOTTO ALBERO SINISTRO*/
        while(valori_input[i] <= punt->valore && punt->figlio_sx!=NULL )
            punt=punt->figlio_sx; //scendi al sottoalbero sinistro
        /* CONTROLLO SE VA SOTTO ALBERO DESTRO*/
        while(valori_input[i]>= punt->valore && punt->figlio_dx!=NULL )
            punt=punt->figlio_dx; //scendi al sottoalbero destro
        /* TROVATO IL SOTTOALBERO, VEDO SE E' FIGLIO SINISTRO O DESTRO*/
        if(valori_input[i] < punt->valore)
        {
            punt->figlio_sx=ptr; //Aggancia come figlio sinistro
            ptr->valore = valori_input[i]; //inserisci valore nel nodo
        }
        else
        {
            punt->figlio_dx=ptr; //Aggancia come figlio destro
            ptr->valore = valori_input[i]; //inserisci valore nel nodo
        }
        i++; //vai ai prossimi valori input
    }

}
/*
INORDER (ordine simmetrico): - sottoalbero sinistro
                            - radice
                            - sottoalbero destro*/
void visita_inorder_Iterativa(struct ALBERO *p_punt)
{
    struct ALBERO *Stack[MAX_SIZE], *top;
    short i_testa=-1;

    while( !(i_testa===-1) || p_punt)
    {
        if(p_punt)
        {
            Push(Stack, p_punt, &i_testa);
            p_punt=p_punt->figlio_sx;
        }
        else
        {
            p_punt = Stack[i_testa];
            Pop(&i_testa);
            printf("%d ", p_punt->valore);
            p_punt=p_punt->figlio_dx;
        }
    }
}

```

```

    }
}

/*
PREORDER (ordine simmetrico): - radice
                                - sottoalbero sinistro
                                - sottoalbero destro
*/
void visita_preorder_Iterativa(struct ALBERO *p_punt)
{
    struct ALBERO *Stack[MAX_SIZE], *top;
    short i_testa=-1;

    Push(Stack, NULL, &i_testa);
    top=p_punt;

    while (i_testa!=-1)
    {
        if (top!=NULL) printf("%d ", top->valore);

        if (top->figlio_dx) Push(Stack, top->figlio_dx, &i_testa);
        if (top->figlio_sx) Push(Stack, top->figlio_sx, &i_testa);
        top=Stack[i_testa];
        Pop(&i_testa);
    }
}

/*
POSTORDER (ordine simmetrico): - sottoalbero sinistro
                                - sottoalbero destro
                                - radice
*/
void visita_postorder_Iterativa(struct ALBERO *p_punt)
{
    struct ALBERO *Stack[MAX_SIZE], *ultimo_visitato=NULL, *peek_nodo;
    short i_testa=-1;

    while ( !(i_testa===-1) || p_punt)
    {
        /* Vai a quello piu' a sinistra, se punta a qualcosa*/
        if (p_punt)
        {
            Push(Stack, p_punt, &i_testa);
            p_punt=p_punt->figlio_sx;
        }
        else
        {

            peek_nodo=Stack[i_testa]; //Copia la testa dello stack (e' un indirizzo)
            if (peek_nodo->figlio_dx && ultimo_visitato!=peek_nodo->figlio_dx)
                p_punt=peek_nodo->figlio_dx;
            else
            {
                Pop(&i_testa);
                printf("%d ", peek_nodo->valore);
                ultimo_visitato=peek_nodo;
            }
        }
    }

    /* Push: Inserire in testa dell'array, appunto, come lo stack. */
void Push(struct ALBERO *Vet_Stack[], struct ALBERO *Indirizzo_nodo, short *Testa)
{
    if (*Testa<MAX_SIZE) //Se non ho superato il massimo dello stack
    {
        /* *Testa=*Testa+1 -> Vet_Stack=Elemento
           La mia testa punterà sempre all'ultimo elemento inserito, quindi incrementando
           metterò l'elemento in testa o PUSHO */
        Vet_Stack[(*Testa)++] = Indirizzo_nodo;
    }
    else printf("\n*----- NON INSERISCO NULLA, STACK PIENO! -----*\n");
}
/* Pop: Eliminare dalla testa dell'array. */
void Pop(short *Testa)

```

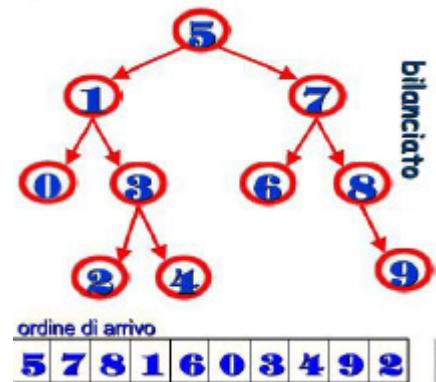
```

{
    if (*Testa > -1) //Se c'e' qualcosa
    {
        /* Decrementando l'indice dell'ultimo elemento inserito, appunto, non verra'
           piu' "visto" l'ultimo elemento inserito, nascondendolo*/
        (*Testa)--;
    }
    //altrimenti "Stack vuoto" viene gestito nel main
}

```

### OUTPUT:

Inserisci il numero di nodi dell'albero: 10  
Inserisci un valore: 5  
Inserisci un valore: 7  
Inserisci un valore: 8  
Inserisci un valore: 1  
Inserisci un valore: 6  
Inserisci un valore: 0  
Inserisci un valore: 3  
Inserisci un valore: 4  
Inserisci un valore: 9  
Inserisci un valore: 2  
VALORI NELL'ARRAY IN INPUT  
5 7 8 1 6 0 3 4 9 2  
VISITA PREORDER DELL'ALBERO BINARIO COSTRUITO  
5 1 0 3 2 4 7 6 8 9  
VISITA INORDER DELL'ALBERO BINARIO COSTRUITO <valori in ordine>  
0 1 2 3 4 5 6 7 8 9  
VISITA POSTORDER DELL'ALBERO BINARIO COSTRUITO  
0 2 4 3 1 6 9 8 7 5



## ESERCIZIO 67 (Scambi reali) [LIV.1]

```
/**[liv1] Scrivere due function C (rispettivamente iterativa e ricorsiva) per implementare
L'algoritmo Selection Sort su un array di struttura, sia mediante scambi REALI.
**/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_NAME 20
/* Definiamo la struttura */
typedef struct Persona
{
    char Nome[MAX_NAME];
    short Eta;
} PERSONA;
/* Prototipi */
void Inserisci_Struct(PERSONA Vet_ite[], PERSONA Vet_ric[], short N);
void Stampa_Struct(PERSONA Vet[], short N);
void selection_sort_Ite(PERSONA Vet[], short N);
void selection_sort_Ric(PERSONA Vet[], short N, short inizio);
short Min(PERSONA Vet[], short N);
void Scambia(PERSONA *a, PERSONA *b);
/***
VERSIONE CON SCAMBI REALI: Tale significa che andiamo a manipola le informazioni
all'interno della memoria, cambiando proprio il contenuto delle locazioni di memoria
in cui sono contenute
**/
```

```
int main()
{
    PERSONA *Vet_ite, *Vet_ric;
    short N;

    puts ("----- SELECTION SORT ITERATIVO E RICORSIVO CON SCAMBI REALI -----");
    /* ____ INPUT ____ */
    puts("Inserire la lunghezza N della struttura:\n");
    scanf("%hd", &N);

    Vet_ite = (PERSONA *) malloc(sizeof(PERSONA)*N);
    Vet_ric = (PERSONA *) malloc(sizeof(PERSONA)*N);
    /* Per comodita', inserisco i valori in ambo i vettori */
    Inserisci_Struct(Vet_ite, Vet_ric, N);

    /* ORDINA */
    selection_sort_Ite(Vet_ite, N);
    selection_sort_Ric(Vet_ric, N, N-1);
    printf("\nORDINAMENTO PER NOME...\n\n");
    printf("ORDINAMENTO REALE CON SELECTION SORT ITERATIVO\n");
    Stampa_Struct(Vet_ite, N);
    printf("ORDINAMENTO REALE CON SELECTION SORT RICORSIVO\n");
    Stampa_Struct(Vet_ric, N);

    printf("\n");

    return 0;
}
/*
    Inserisce la struct sia per il vettore iterativo che ricorsivo
*/
void Inserisci_Struct(PERSONA Vet_ite[], PERSONA Vet_ric[], short N)
{
    short i, eta;
    char nome[MAX_NAME];

    for(i=0;i<N;i++)
    {
        printf("STRUCT N°%hd \n", i+1);
        printf("Inserisci nome :"); fflush(stdin);
        gets(nome);
```

```

printf("Inserisci eta :");fflush(stdin);
scanf("%hd", &eta);

/* Copia le info nei 2 vettori struct */
strcpy(Vet_ite[i].Nome, nome);strcpy(Vet_ric[i].Nome, nome);
Vet_ite[i].Eta=eta;Vet_ric[i].Eta=eta;
}

}

void Stampa_Struct (PERSONA Vet[], short N)
{
    short i;
    for(i=0;i<N; i++)
        printf("Nome: %s \n Eta: %hd\n\n", Vet[i].Nome, Vet[i].Eta);
}
/*
O(N^2)
Passaggi effettuati: - Cicla da 0 a N-1
                      - Trova il minimo
                      - Inserisci minimo e considera le porzioni successive(i++)
*/
void selection_sort_Ite(PERSONA Vet[], short N)
{
    short i, i_Min;

    for(i=0; i<N-1; i++)
    {
        i_Min = Min(&Vet[i], N-i);
        Scambia(&Vet[i], &Vet[i_Min+i]);
    }
}
/*
    Trova il minimo nel vettore
*/
short Min(PERSONA Vet[], short N)
{
    short i=0, i_min;
    i_min = i;
    for(i=1; i<N; i++)
        if(strcmp(Vet[i].Nome, Vet[i_min].Nome)<0 ) i_min=i;

    return i_min;
}
/*
    Scambia 2 valori passati per indirizzo
*/
void Scambia(PERSONA *a, PERSONA *b)
{
    PERSONA tmp;
    tmp=*a;
    *a=*b;
    *b=tmp;
}
/*
    L'idea e' identica all'iterativo, solo che utilizziamo una variabile 'inizio' che
    parte da N e con le autoattivazioni la facciamo diventare 0.
    Ora a ritroso, quando devo no ritornare le chiamate,
    simula un ciclo for da inizio=0 fino ad N-1 (controlla la chiamata) */
void selection_sort_Ric(PERSONA Vet[], short N, short inizio)
{
    short i_Min;
    /* CASOBASE */
    if(inizio<0) return;

    /* AUTOATTIVAZIONE */
    selection_sort_Ric(Vet, N, inizio-1);
    i_Min = Min(&Vet[inizio], N-inizio);
    Scambia(&Vet[inizio], &Vet[i_Min+inizio]);
}

```

}

**OUTPUT:**

----- SELECTION SORT ITERATIVO E RICORSIVO CON SCAMBI REALI -----\*

Inserire la lunghezza N della struttura:

```
4
STRUCT N1
Inserisci nome :Vittorio
Inserisci eta :22
STRUCT N2
Inserisci nome :Giancarlo
Inserisci eta :19
STRUCT N3
Inserisci nome :Giuseppe
Inserisci eta :29
STRUCT N4
Inserisci nome :Antonio
Inserisci eta :17
Nome: Vittorio
Eta: 22

Nome: Giancarlo
Eta: 19

Nome: Giuseppe
Eta: 29

Nome: Antonio
Eta: 17
```

**ORDINAMENTO PER NOME...**

ORDINAMENTO CON SELECTION SORT ITERATIVO  
 Nome: Antonio  
 Eta: 17

Nome: Giancarlo  
 Eta: 19

Nome: Giuseppe  
 Eta: 29

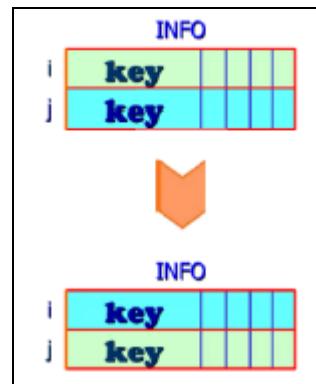
Nome: Vittorio  
 Eta: 22

ORDINAMENTO REALE CON SELECTION SORT RICORSIVO  
 Nome: Antonio  
 Eta: 17

Nome: Giancarlo  
 Eta: 19

Nome: Giuseppe  
 Eta: 29

Nome: Vittorio  
 Eta: 22



## ESERCIZIO 67 (Scambi virtuali) [LIV.1]

```
/*
[liv1] Scrivere due function C (rispettivamente iterativa e ricorsiva) per implementare
L'algoritmo Selection Sort su un array di struttura, sia mediante scambi VIRTUALI.
**/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_NAME 20
/* Definiamo la struttura */
typedef struct Persona
{
    char Nome[MAX_NAME];
    short Eta;
} PERSONA;
/* Prototipi */
void Inserisci_Struct(PERSONA Vet[], short N);
void Stampa_Struct(PERSONA Vet[], short N);
void Stampa_Struct_punt(PERSONA *Vet[], short N);
void selection_sort_Ite(PERSONA *Vet[], short N);
void selection_sort_Ric(PERSONA *Vet[], short N, short inizio);
short Min(PERSONA *Vet[], short N);
void Scambia_punt(PERSONA **a, PERSONA **b);
/**/
VERSIONE CON SCAMBI VIRTUALI: Tale significa che andiamo a manipolare un array di puntatori
che puntano a delle locazioni di memoria le cui informazioni non sono toccate.
Per trovare il minimo, scambio o considerare una certa parte da ordinare, andiamo a lavorare
con un puntatore di array, le cui celle puntano alla cella di memoria interessata.
Quindi, ordineremo e scambieremo gli indirizzi in base ai loro "puntati"
**/

int main()
{
    PERSONA *Vet, **a_punt_ite, **a_punt_ric;
    short N, i;

    puts ("----- SELECTION SORT ITERATIVO E RICORSIVO CON SCAMBI VIRTUALI -----");
    /* ____ INPUT ____ */
    puts("Inserire la lunghezza N della struttura:\n");
    scanf ("%hd", &N);
    Vet = (PERSONA *) malloc(sizeof(PERSONA)*N);
    /* Per comodita', inserisco i valori in ambo i vettori */
    Inserisci_Struct(Vet, N);

    /* ____ ALLOCA E COSTRUISCI ARRAY DI PUNTATORI ____ */
    a_punt_ite = (PERSONA **) malloc(sizeof(PERSONA *)*N);
    a_punt_ric = (PERSONA **) malloc(sizeof(PERSONA *)*N);
    for(i=0; i<N; i++)
    {
        //Assegno gli indirizzi
        a_punt_ite[i]=&Vet[i];
        a_punt_ric[i]=&Vet[i];
    }
    /* Stampa vet iniziale*/
    Stampa_Struct(Vet, N);

    /* ORDINA */
    selection_sort_Ite(a_punt_ite, N);
    selection_sort_Ric(a_punt_ric, N, N-1);
    printf ("\nORDINAMENTO PER NOME...\n\n");
    printf ("ORDINAMENTO REALE CON SELECTION SORT ITERATIVO\n");
    Stampa_Struct_punt(a_punt_ite, N);
    printf ("ORDINAMENTO REALE CON SELECTION SORT RICORSIVO\n");
    Stampa_Struct_punt(a_punt_ric, N);

    printf ("\n");

    return 0;
}
/*
    Inserisce la struct sia per il vettore iterativo che ricorsivo
*/
void Inserisci_Struct(PERSONA Vet[], short N)
{
    short i, eta;
    char nome[MAX_NAME];
    for(i=0; i<N; i++)
    {
        printf("Inserisci nome: ");
        gets(nome);
        printf("Inserisci età: ");
        scanf ("%hd", &eta);
        Vet[i].Nome = nome;
        Vet[i].Eta = eta;
    }
}
```

```

for(i=0; i<N; i++)
{
    printf("STRUCT N°%hd \n", i+1);
    printf("Inserisci nome :"); fflush(stdin);
    gets(nome);

    printf("Inserisci eta :"); fflush(stdin);
    scanf("%hd", &eta);

    /* Copia le info nei 2 vettori struct */
    strcpy(Vet[i].Nome, nome);
    Vet[i].Eta=eta;
}
}

void Stampa_Struct_punt(PERSONA *Vet[], short N)
{
    short i;
    for(i=0; i<N; i++)
        printf("Nome: %s \n Eta: %hd\n\n", (*Vet[i]).Nome, (*Vet[i]).Eta);
}
void Stampa_Struct(PERSONA Vet[], short N)
{
    short i;
    for(i=0; i<N; i++)
        printf("Nome: %s \n Eta: %hd\n\n", Vet[i].Nome, Vet[i].Eta);
}
/*
O(N^2)
Passaggi effettuati: - Cicla da 0 a N-1
                     - Trova il minimo, passando l'indirizzo dell'array puntatore
                     - Inserisci minimo e considera le porzioni successive(i++)
Virtuale: Lavoriamo con l'array a puntatore, passando l'indirizzo base di tale array punt.

In Conclusione, STAMPERA' IN ORDINE, QUANDO GLI INDIRIZZI DENTRO A QUESTO ARRAY SONO DISPOSTI
IN TAL MODO CHE STAMPANDO I LORO PUNTATI, VENGA VISUALIZZATO IN ORDINE.
*/
void selection_sort_Ite(PERSONA *Vet[], short N)
{
    short i, i_Min;

    for(i=0; i<N-1; i++)
    {
        i_Min = Min(&Vet[i], N-i);
        Scambia_punt(&Vet[i], &Vet[i_Min+i]);
    }
}
/*
    Trova il minimo nel vettore puntato da *Vet[i]
*/
short Min(PERSONA *Vet[], short N)
{
    short i=0, i_min;
    i_min = i;
    for(i=1; i<N; i++)
        if(strcmp(Vet[i]->Nome, Vet[i_min]->Nome)<0 ) i_min=i;

    return i_min;
}
/*
    Scambia 2 valori passati per indirizzo.
    Scambiamo gli indirizzi dentro l'array puntatore
*/
void Scambia_punt(PERSONA **a, PERSONA **b)
{
    PERSONA *tmp;
    tmp=*a;
    *a=*b;
    *b=tmp;
}

/*
L'idea e' identica all'iterativo, solo che utilizziamo una variabile 'inizio' che
parte da N e con le autoattivazioni la facciamo diventare 0.
Ora a ritroso, quando devo no ritornare le chiamate,
simula un ciclo for da inizio=0 fino ad N-1 (controlla la chiamata)

```

```
/*
void selection_sort_Ric(PERSONA *Vet[], short N, short inizio)
{
    short i_Min;
    /* CASOBASE */
    if(inizio<0) return;

    /* AUTOATTIVAZIONE */
    selection_sort_Ric(Vet, N, inizio-1);
    i_Min = Min(&Vet[inizio], N-inizio);
    Scambia_punt(&Vet[inizio], &Vet[i_Min+inizio]);
}
```

**OUTPUT:**

----- SELECTION SORT ITERATIVO E RICORSIVO CON SCAMBI VIRTUALI -----  
Inserire la lunghezza N della struttura:

```
4
STRUCT M#1
Inserisci nome :Vittorio
Inserisci eta :22
STRUCT M#2
Inserisci nome :Giancarlo
Inserisci eta :19
STRUCT M#3
Inserisci nome :Giuseppe
Inserisci eta :29
STRUCT M#4
Inserisci nome :Antonio
Inserisci eta :17
Nome: Vittorio
Eta: 22

Nome: Giancarlo
Eta: 19

Nome: Giuseppe
Eta: 29

Nome: Antonio
Eta: 17
```

**ORDINAMENTO PER NOME...**

ORDINAMENTO VIRTUALE CON SELECTION SORT ITERATIVO  
Nome: Antonio  
Eta: 17

Nome: Giancarlo  
Eta: 19

Nome: Giuseppe  
Eta: 29

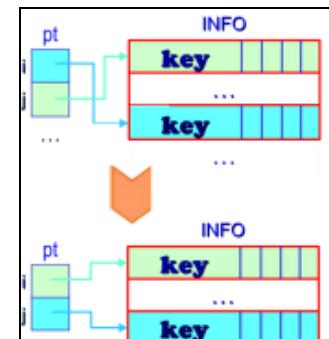
Nome: Vittorio  
Eta: 22

ORDINAMENTO VIRTUALE CON SELECTION SORT RICORSIVO  
Nome: Antonio  
Eta: 17

Nome: Giancarlo  
Eta: 19

Nome: Giuseppe  
Eta: 29

Nome: Vittorio  
Eta: 22



## ESERCIZIO 69 (Scambi reali) [LIV.1]

```
/** [liv1] Scrivere una function C per implementare l'algoritmo Bubble Sort su un array di struttura,
sia mediante scambi REALI. **/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_NAME 20
/* Definiamo la struttura */
typedef struct Persona
{
    char Nome[MAX_NAME];
    short Eta;
} PERSONA;
/* Prototipi */
void bubble_sort_Ite(PERSONA V[], short N);
void bubble_sort_Ric(PERSONA V[], short N);
void Inserisci_Struct(PERSONA Vet_ite[], PERSONA Vet_ric[], short N);
void Stampa_Struct(PERSONA Vet[], short N);
void Scambia(PERSONA *a, PERSONA *b);
/** VERSIONE CON SCAMBI REALI: Tale significa che andiamo a manipola le informazioni
all'interno della memoria, cambiando proprio il contenuto delle locazioni di memoria
in cui sono contenute*/
int main()
{
    PERSONA *Vet_ite, *Vet_ric;
    short N;

    puts("----- BUBBLE SORT ITERATIVO E RICORSIVO CON SCAMBI REALI -----");
    /* _____ INPUT _____ */
    puts("Inserire la lunghezza N della struttura:\n");
    scanf("%hd", &N);

    Vet_ite = (PERSONA *) malloc(sizeof(PERSONA)*N);
    Vet_ric = (PERSONA *) malloc(sizeof(PERSONA)*N);
    /* Per comodita', inserisco i valori in ambo i vettori */
    Inserisci_Struct(Vet_ite, Vet_ric, N);
    /* ORDINA */
    bubble_sort_Ite(Vet_ite, N);
    bubble_sort_Ric(Vet_ric, N);
    printf("\nORDINAMENTO PER NOME...\n\n");
    printf("ORDINAMENTO REALE CON BUBBLE SORT ITERATIVO\n");
    Stampa_Struct(Vet_ite, N);
    printf("ORDINAMENTO REALE CON BUBBLE SORT RICORSIVO\n");
    Stampa_Struct(Vet_ric, N);

    printf("\n");
    return 0;
}
/* bubble_sort_Ite: Confronta a coppia, facendo "risalire" i valori piccoli.
Se c'e' uno scambio, si salva l'eventuale ultimo indice in cui e' avvenuto lo
scambio.
Se c'e' stato lo scambio, ricicla di nuovo fermandosì però all'ultimo
indice salvato, per risparmiare molto tempo*/
void bubble_sort_Ite(PERSONA V[], short N)
{
    short Scambiato, i, ultimo_scambio;

    do
    {
        Scambiato=0;
        for(i=0; i<N-1; i++)
        {
            //Confronto a 2 a 2
            if(strcmp(V[i].Nome, V[i+1].Nome)>0)
            {
                Scambia(&V[i], &V[i+1]);
                Scambiato = 1;
                ultimo_scambio = i+1; /* Salva eventuale ultimo indice*/
            }
        }
        N = ultimo_scambio;
    }while(Scambiato); //Se non avviene scambio, e' ordinato
}
/* bubble_sort_Ric: e' meno efficiente, non salva l'indice dell'ultimo scambio.
Se avviene uno scambio, richiama se stesso passando N-1, perche' l'ultimo
elemento sara' il piu' grande.*/
void bubble_sort_Ric(PERSONA V[], short N)
```

```

{
    short Scambiato=0, i;

    for(i=0; i<N-1; i++)
    {
        if(strcmp(V[i].Nome, V[i+1].Nome)>0)
        {
            Scambia(&V[i], &V[i+1]);
            Scambiato = 1;
        }
    }

    if(Scambiato==0) return; //CASO BASE
    bubble_sort_Ric(V, N--); //AUTOATTIVAZIONE

    return;
}
/* Scambia 2 valori passati per indirizzo */
void Scambia(PERSONA *a, PERSONA *b)
{
    PERSONA tmp;
    tmp=*a;
    *a=*b;
    *b=tmp;
}
/* Inserisce la struct sia per il vettore iterativo che ricorsivo */
void Inserisci_Struct(PERSONA Vet_ite[], PERSONA Vet_ric[], short N)
{
    short i, eta;
    char nome[MAX_NAME];

    for(i=0;i<N;i++)
    {
        printf("STRUCT N°%hd \n", i+1);
        printf("Inserisci nome :");fflush(stdin);
        gets(nome);

        printf("Inserisci eta :");fflush(stdin);
        scanf("%hd", &eta);

        /* Copia le info nei 2 vettori struct */
        strcpy(Vet_ite[i].Nome, nome);strcpy(Vet_ric[i].Nome, nome);
        Vet_ite[i].Eta=eta;Vet_ric[i].Eta=eta;
    }
}

void Stampa_Struct(PERSONA Vet[], short N)
{
    short i;
    for(i=0;i<N; i++)
        printf("Nome: %s \n Eta: %hd\n\n", Vet[i].Nome, Vet[i].Eta);
}

```

**OUTPUT:**

\*\*\*\*\* BUBBLE SORT ITERATIVO E RICORSIVO CON SCAMBI REALI \*\*\*\*\*  
Inserire la lunghezza N della struttura:

```

4
STRUCT N°1
Inserisci nome :Giovanni
Inserisci eta :20
STRUCT N°2
Inserisci nome :Andrea
Inserisci eta :21
STRUCT N°3
Inserisci nome :Mario
Inserisci eta :22
STRUCT N°4
Inserisci nome :Francesco
Inserisci eta :19
Nome: Giovanni
Eta: 20
Nome: Andrea
Eta: 21
Nome: Mario
Eta: 22
Nome: Francesco
Eta: 19

```

**ORDINAMENTO PER NOME...**

ORDINAMENTO REALE CON BUBBLE SORT ITERATIVO  
Nome: Andrea  
Eta: 21

Nome: Francesco  
Eta: 19

Nome: Giovanni  
Eta: 20

Nome: Mario  
Eta: 22

**ORDINAMENTO REALE CON BUBBLE SORT RICORSIVO**

Nome: Andrea  
Eta: 21

Nome: Francesco  
Eta: 19

Nome: Giovanni  
Eta: 20

Nome: Mario  
Eta: 22

## ESERCIZIO 69 (Scambi virtuali) [LIV.1]

```
/*
[liv1] Scrivere una function C per implementare l'algoritmo Bubble Sort su un array di struttura,
sia mediante scambi VIRTUALI.

*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_NAME 20

/* Definiamo la struttura */
typedef struct Persona
{
    char Nome[MAX_NAME];
    short Eta;
} PERSONA;

/* Prototipi */
void bubble_sort_Ite(PERSONA *V[], short N);
void bubble_sort_Ric(PERSONA *V[], short N);
void Inserisci_Struct(PERSONA Vet[], short N);
void Stampa_Struct(PERSONA Vet[], short N);
void Stampa_Struct_punt(PERSONA *Vet[], short N);
void Scambia(PERSONA **a, PERSONA **b);

/**VERSIONE CON SCAMBI VIRTUALI: Tale significa che andiamo a manipolare un array di puntatori
che puntano a delle locazioni di memoria le cui informazioni non sono toccate.
Per trovare il minimo, scambio o considerare una certa parte da ordinare, andiamo a lavorare
con un puntatore di array, le cui celle puntano alla cella di memoria interessata.
Quindi, ordineremo e scambieremo gli indirizzi in base ai loro "puntati"**/

int main()
{
    PERSONA *Vet, **a_punt_ite, **a_punt_ric;
    short N, i;

    puts ("----- BUBBLE SORT ITERATIVO E RICORSIVO CON SCAMBI REALI -----");

    /* ____ INPUT ____ */
    puts("Inserire la lunghezza N della struttura:\n");
    scanf ("%hd", &N);

    Vet = (PERSONA *) malloc(sizeof(PERSONA)*N);
    /* Per comodita', inserisco i valori in ambo i vettori */
    Inserisci_Struct(Vet, N);

    /* ____ ALLOCA E COSTRUISCI ARRAY DI PUNTATORI ____ */
    a_punt_ite = (PERSONA **) malloc(sizeof(PERSONA *)*N);
    a_punt_ric = (PERSONA **) malloc(sizeof(PERSONA *)*N);
    for(i=0; i<N; i++)
    {
        //Assegno gli indirizzi
        a_punt_ite[i]=&Vet[i];
        a_punt_ric[i]=&Vet[i];
    }

    /* Stampa vet iniziale*/
    Stampa_Struct(Vet, N);

    /* ORDINA */
    bubble_sort_Ite(a_punt_ite, N);
    bubble_sort_Ric(a_punt_ric, N);
    printf ("\nORDINAMENTO PER NOME...\n\n");
    printf ("ORDINAMENTO REALE CON BUBBLE SORT ITERATIVO\n");
    Stampa_Struct_punt(a_punt_ite, N);
    printf ("ORDINAMENTO REALE CON BUBBLE SORT RICORSIVO\n");
    Stampa_Struct_punt(a_punt_ric, N);

    printf ("\n");

    return 0;
}

/*
bubble_sort_Ite: Confronta a coppia, facendo "risalire" i valori piccoli.
Se c'è uno scambio, si salva l'eventuale ultimo indice in cui e' avvenuto lo

```

```

scambio.
Se c'è stato lo scambio, ricicla di nuovo fermandosi però all'ultimo
indice salvato, per risparmiare molto tempo
*/
void bubble_sort_Ite(PERSONA *V[], short N)
{
    short Scambiato, i, ultimo_scambio;
    do
    {
        Scambiato=0;
        for(i=0; i<N-1; i++)
        {
            //Confronto a 2 a 2
            if(strcmp(V[i]->Nome, V[i+1]->Nome)>0) // (*V[i]).Nome, (*V[i+1]).Nome
            {
                Scambia(&V[i], &V[i+1]);
                Scambiato = 1;
                ultimo_scambio = i+1; /* Salva eventuale ultimo indice*/
            }
        }
        N = ultimo_scambio;
    }while(Scambiato); //Se non avviene scambio, e' ordinato
}
/* bubble_sort_Ric: e' meno efficiente, non salva l'indice dell'ultimo scambio.
Se avviene uno scambio, richiama se stesso passando N-1, perche' l'ultimo
elemento sara' il piu' grande.*/
void bubble_sort_Ric(PERSONA *V[], short N)
{
    short Scambiato=0, i;

    for(i=0; i<N-1; i++)
    {
        if(strcmp((*V[i]).Nome, (*V[i+1]).Nome)>0) //V[i]->Nome, V[i+1]->Nome
        {
            Scambia(&V[i], &V[i+1]);
            Scambiato = 1;
        }
    }

    if(Scambiato==0) return; //CASO BASE
bubble_sort_Ric(V, N--); //AUTOATTIVAZIONE

    return; //per ricordare che ritornerà all'istanza precedente
}
/*
    Scambia 2 valori passati per indirizzo
*/
void Scambia(PERSONA **a, PERSONA **b)
{
    PERSONA *tmp;
    tmp=*a;
    *a=*b;
    *b=tmp;
}
/*
    Inserisce la struct sia per il vettore iterativo che ricorsivo
*/
void Inserisci_Struct(PERSONA Vet[], short N)
{
    short i, eta;
    char nome[MAX_NAME];

    for(i=0;i<N;i++)
    {
        printf("STRUCT N°%hd \n", i+1);
        printf("Inserisci nome :");fflush(stdin);
        gets(nome);

        printf("Inserisci eta :");fflush(stdin);
        scanf("%hd", &eta);

        /* Copia le info nei 2 vettori struct */
        strcpy(Vet[i].Nome, nome);
        Vet[i].Eta=eta;
    }
}

void Stampa_Struct(PERSONA Vet[], short N)

```

```

{
    short i;
    for(i=0;i<N; i++)
        printf("Nome: %s \n Eta: %hd\n\n", Vet[i].Nome, Vet[i].Eta);
}
void Stampa_Struct_punt(PERSONA *Vet[], short N)
{
    short i;
    for(i=0;i<N; i++)
        printf("Nome: %s \n Eta: %hd\n\n", (*Vet[i]).Nome, (*Vet[i]).Eta);
}

```

**OUTPUT:**

\*\*\*\*\* BUBBLE SORT ITERATIVO E RICORSIVO CON SCAMBI VIRTUALI \*\*\*\*\*  
Inserire la lunghezza N della struttura:

```

4
STRUCT N1
Inserisci nome :Giovanni
Inserisci eta :20
STRUCT N2
Inserisci nome :Andrea
Inserisci eta :19
STRUCT N3
Inserisci nome :Mario
Inserisci eta :21
STRUCT N4
Inserisci nome :Francesco
Inserisci eta :22
Nome: Giovanni
Eta: 20

Nome: Andrea
Eta: 19

Nome: Mario
Eta: 21

Nome: Francesco
Eta: 22

```

**ORDINAMENTO PER NOME...**

**ORDINAMENTO VIRTUALE CON BUBBLE SORT ITERATIVO**

```

Nome: Andrea
Eta: 19

Nome: Francesco
Eta: 22

Nome: Giovanni
Eta: 20

Nome: Mario
Eta: 21

```

**ORDINAMENTO VIRTUALE CON BUBBLE SORT RICORSIVO**

```

Nome: Andrea
Eta: 19

Nome: Francesco
Eta: 22

Nome: Giovanni
Eta: 20

Nome: Mario
Eta: 21

```

## ESERCIZIO 70 [LIV.1]

```
/*
*[liv1]Scrivere una function C per implementare l'algoritmo Insertion Sort su un array di struttura.
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_NAME 20

/* Definiamo la struttura */
typedef struct Persona
{
    char Nome[MAX_NAME];
    short Eta;
} PERSONA;

/* Prototipi */
void insertion_sort(PERSONA V[], short N);
void Inserisci_Struct(PERSONA Vet[], short N);
void Stampa_Struct(PERSONA Vet[], short N);

int main()
{
    PERSONA *Vet;
    short N;

    puts ("***** BUBBLE SORT ITERATIVO E RICORSIVO CON SCAMBI REALI *****");

    /* ____ INPUT ____ */
    puts("Inserire la lunghezza N della struttura:\n");
    scanf("%hd", &N);
    Vet = (PERSONA *) malloc(sizeof(PERSONA)*N);
    /* Per comodita', inserisco i valori in ambo i vettori */
    Inserisci_Struct(Vet, N);

    /* ORDINA */
    insertion_sort(Vet, N);
    printf ("\nORDINAMENTO PER NOME...\n\n");
    printf ("ORDINAMENTO INSERTION SORT\n");
    Stampa_Struct(Vet, N);

    printf ("\n");

    return 0;
}

/*
L'algoritmo utilizza due indici: il primo punta inizialmente al secondo elemento dell'array,
il secondo inizia dal primo. Se il primo elemento > maggiore del secondo, i due valori vengono
scambiati. Poi il primo indice avanza di una posizione e il secondo indice riparte dall'elemento
precedente quello puntato dal primo. Se l'elemento puntato dal secondo indice non > maggiore di
quello a cui punta il primo indice, il secondo indice indietreggia; e cosà fa, finchà si trova
nel
    punto in cui il valore del primo indice deve essere inserito (da qui insertion). L'algoritmo
cosà
    tende a spostare man mano gli elementi maggiori verso destra.
*/
void insertion_sort(PERSONA V[], short N)
{
    PERSONA el_da_ins;
    short i, j;

    for(i=1; i<N; i++)
    {
        el_da_ins = V[i];
        j=i-1;
        while(j>=0 && (strcmp(V[j].Nome, el_da_ins.Nome)>0) )
        {
            V[j+1] = V[j];
            j--;
        }
        V[++j]=el_da_ins;
    }
}

/*
    Inserisce la struct sia per il vettore iterativo che ricorsivo

```

```

*/
void Inserisci_Struct (PERSONA Vet[], short N)
{
    short i, eta;
    char nome[MAX_NAME];

    for (i=0; i<N; i++)
    {
        printf("STRUCT N°%d \n", i+1);
        printf("Inserisci nome :"); fflush(stdin);
        gets(nome);

        printf("Inserisci eta :"); fflush(stdin);
        scanf("%hd", &eta);

        /* Copia le info nei 2 vettori struct */
        strcpy(Vet[i].Nome, nome);
        Vet[i].Eta=eta;
    }
}

void Stampa_Struct (PERSONA Vet[], short N)
{
    short i;
    for (i=0; i<N; i++)
        printf("Nome: %s \n Eta: %hd\n\n", Vet[i].Nome, Vet[i].Eta);
}

```

**OUTPUT:**

Inserire la lunghezza N della struttura:

```

4
STRUCT N°1
Inserisci nome :Mario
Inserisci eta :19
STRUCT N°2
Inserisci nome :Andrea
Inserisci eta :20
STRUCT N°3
Inserisci nome :Francesco
Inserisci eta :22
STRUCT N°4
Inserisci nome :Antonio
Inserisci eta :21

```

ORDINAMENTO PER NOME...

ORDINAMENTO INSERTION SORT  
 Nome: Andrea  
 Eta: 20

Nome: Antonio  
 Eta: 21

Nome: Francesco  
 Eta: 22

Nome: Mario  
 Eta: 19

## ESERCIZIO 71 [LIV.2]

```
/** [liv2] Scrivere function C per implementare l'algoritmo Merge Sort ***/
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 5
void stampa_vet(short V[], short N);
void merge_sort(short V[], short Inizio, short Fine);
void merge(short V[], short Inizio, short Mez, short Fine);

int main()
{
    short V[]={5,4,3,2,1,6,0}, N=7;
    puts("----- MERGE SORT -----*\n");
    /* STAMPA */
    puts("ARRAY INIZIALE:");
    stampa_vet(V, N);
    /* ORDINA */
    merge_sort(V, 0, N-1);
    /* STAMPA DOPO ORDINAMENTO */
    puts("ARRAY ORDINATO MEDIANTE MERGE-SORT:");
    stampa_vet(V, N);
    printf("\n");
    return 0;
}
/*
merge_sort: Questa versione ricorsiva, divide il nostro vettore in tanti sottovettori.
Se Inizio<Fine significa che l'algoritmo di merge comincia quando considera minimo
un sottovettore di 2 elementi.
*/
void merge_sort(short V[], short Inizio, short Fine)
{
    short Mez;
    /* Se ho un vettore con N > 1 */
    if(Inizio<Fine)
    {
        Mez=(Inizio+Fine)/2;
        merge_sort(V, Inizio, Mez); /* Considera la metà' di sinistra */
        merge_sort(V, Mez+1, Fine); /* Considera la metà' di destra */
        merge(V, Inizio, Mez, Fine); /* Di quel sottovettore, fai un merge considerando
                                       meta' sotto vettore */
    }
    return;
}
/* merge: Dato il sotto vettore, faccia la fusione tra il sottovettore compreso
   tra [inizio, Mez] e [Mez+1, Fine] e il risultato va in un array in piu' Aux.
   Finita la fusione, copia Aux nell'array, a partire da inizio fino a fine */
void merge(short V[], short Inizio, short Mez, short Fine)
{
    /* i=Inizio sottovettore di sinistra Mez=Fine sottovettore di sinistra
       j=Inizio sottovettore di destra Fine=Fine sottovettore di Destra
       k=Indice del vettore ausiliare y=Lo sfrutteremo per scorrere aux e copiare
                                         in V[]
    */
    short i=Inizio, j=Mez+1, k=0, *Aux, y;

    //Alloco giusto i valori che andranno in aux (Fine-Inizio+1)
    Aux=malloc(sizeof(short)*(Fine-Inizio+1));

    /* _____ MERGE _____ */
    // <= e non < Perche' devo considerare anche Mez - Fine, che sarebbero N-1
    while(i<=Mez && j<=Fine)
    {
        if(V[i]<V[j])
            Aux[k++]=V[i++];
        else
            Aux[k++]=V[j++];
    }
    while(i<=Mez)
        Aux[k++]=V[i++];
    while(j<=Fine)
        Aux[k++]=V[j++];

    //Copia la porzione di array costruito A partire da inizio
    for(y=0; y<k; y++)
        V[Inizio+y]=Aux[y];
}
```

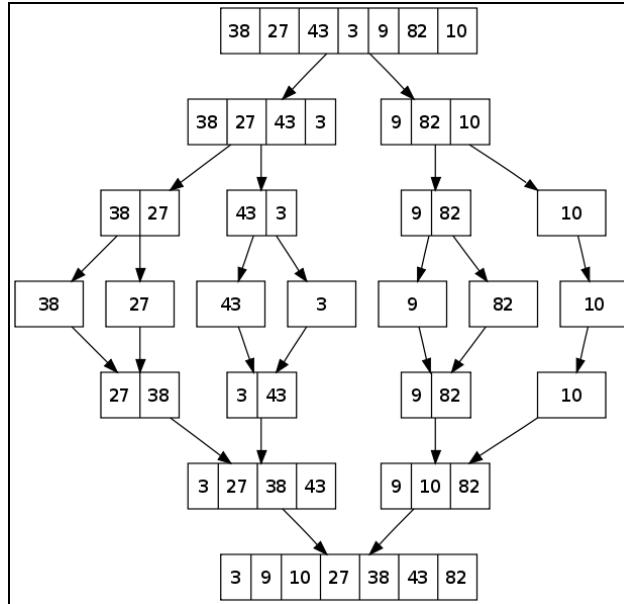
```
void stampa_vet(short v[], short N)
{
    int i;
    for(i=0; i<N; i++)
        printf("%hd ", v[i]);
    puts("\n");
}
```

**OUTPUT:****MERGE SORT****ARRAY INIZIALE:**

5 4 3 2 1 6 0

**ARRAY ORDINATO MEDIANTE MERGE-SORT:**

0 1 2 3 4 5 6



**Complessità di spazio =  $O(n) + O(n)$**   
algoritmo base: non è in place

$$\downarrow \quad T_{\text{fusion}}(n) \times \text{Numero passi} \\ O(n) \times O(\log_2 n)$$

**Complessità di tempo**

$$\left. \begin{array}{l} \text{Numero confronti} \\ \text{ed assegnazioni} \end{array} \right\} = O(n \log_2 n)$$

### ESERCIZIO 73 [LIV.3]

```
/** [liv2] Scrivere function C per implementare l'algoritmo Quick Sort ***/
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#define MAX_SIZE 5
void stampa_vet(short V[], short N);
void quick_sort(short V[], short Inizio, short Fine);
short Partiziona(short V[], short Inizio, short Fine);
void Scambia(short *a, short *b);

int main()
{
    short V[]={6,2,2,10,10,0,0,-23,99,1}, N=10;
    srand(time(NULL));
    puts("----- QUICK SORT -----*\n");
    /* STAMPA */
    puts("ARRAY INIZIALE:");
    stampa_vet(V, N);
    /* ORDINA */
    quick_sort(V, 0, N-1);
    /* STAMPA DOPO ORDINAMENTO */
    puts("ARRAY ORDINATO MEDIANTE QUICK-SORT:");
    stampa_vet(V, N);
    printf("\n");
    return 0;
}
/* quick_sort: Se ho almeno 2 elementi (Inizio<Fine) Calcolo il perno con partiziona.
   Il perno o pivotNuovo, sara' il valore per cui i suoi valori alla destra sono maggiori
   e viceversa, quindi andiamo ad ordinare da INIZIO a PERTNO-1 (PORZIONE DI SINISTRA)
   e da PERTNO+1 a FINE (PORZIONE DI DESTRA) */
void quick_sort(short V[], short Inizio, short Fine)
{
    int Perno;

    if(Inizio<Fine)
    {
        Perno = Partiziona(V, Inizio, Fine); //Trova il partizionatore
        quick_sort(V, Inizio, Perno-1); //Ordina la porzione a sinistra di partizionatore
        quick_sort(V, Perno+1, Fine); //Ordina la porzione a destra di partizionatore
    }
}
/*
PARTIZIONA: IL CUORE DEL PROGRAMMA.
CASO MIGLIORE O(nlogn)
CASO PEGGIORE O(n^2)
Funzionamento: Ha il compito di ordinare gli elementi in base al x = V[PIVOT] e disporre
l'elemento PIVOT nella sua reale posizione ordinata.
Il fine e' quello di ottenere la porzione alla sua destra con valori maggiori o uguali di x e
alla sua porzione di sinistra valori minori o uguali a x.
*/
short Partiziona(short V[], short Inizio, short Fine)
{
    short Pivot=Fine; //Pivot sistematico
    short i=Inizio, j=Fine, x=V[Pivot]; /* j parte da fine e non fine-1!
                                              verifica con {2,6} e si capirà che subito
                                              uscirebbe dal ciclo e scambierebbe l pivot,
                                              quando poi stanno bene così! */

    /* Fin quando i e j non si incontrano */
    while(i<j)
    {
        /* Mentre V[i] <= al valore Pivot, da sinistra a destra, va bene e avanza i
           Mentre V[j] >= al valore Pivot, da destra a sinistra, va bene e arretra j*/
        while(V[i]<=x && i<j)
            i++;
        while(V[j]>=x && i<j)
            j--;
        /* se vengono trovati sia a[i] che a[j] allora si scambiano */
        if(i<j) Scambia(&V[i], &V[j]);
    }
    /* Scambia il pivot nella sua posizione reale */
    Scambia(&V[Pivot], &V[j]);
    /* Ritorna il PARTIZIONATORE*/
    return j;
}
```

```

void Scambia (short *a, short *b)
{
    short tmp;
    tmp=*a;
    *a=*b;
    *b=tmp;
}
void stampa_vet (short V[], short N)
{
    int i;
    for(i=0; i<N; i++)
        printf ("%hd ", V[i]);
    puts ("\n");
}

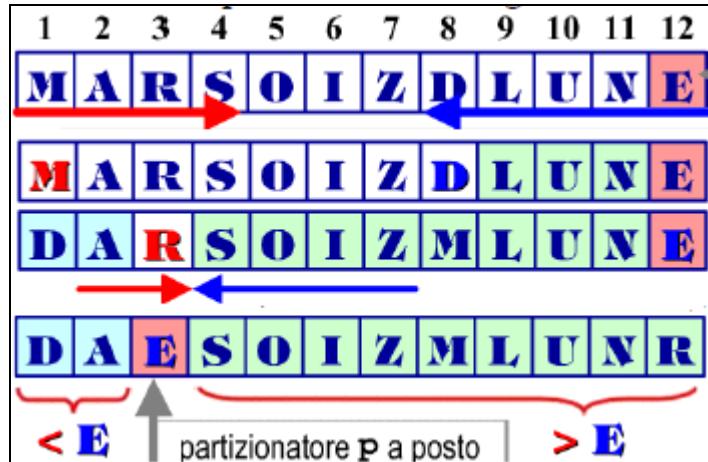
```

**OUTPUT:****\*----- QUICK SORT -----\*****ARRAY INIZIALE:**

6 2 2 10 10 0 0 -23 99 1

**ARRAY ORDINATO MEDIANTE QUICK-SORT:**

-23 0 0 1 2 2 6 10 10 99

**Complessità di spazio =  $O(n)$** **Complessità di tempo**

(caso migliore) (caso peggiore)

**Numero confronti =  $O(n \log_2 n)$        $O(n^2)$** 

Nel caso di dati “quasi” già ordinati il **Quicksort** fornisce la peggiore prestazione e non va usato!

## ESERCIZIO 74 [LIV.3]

```
/**[liv.3] Scrivere function C iterativa per la trasformazione in un heap di un albero binario
rappresentato mediante array.
PS ESSO E' UNA VARIAZIONE DELL'ESERCIZIO 54*/
#include <stdio.h>
#include <stdlib.h>
void costruisci_heap(short heap[], short n);
short heapify(short heap[], short i, short n);
short is_foglia(short heap[], short i, short n);
void scambia(short *p, short *q);
void heap_sort(short V[], short n);
void stampa_vet(short V[], short N);
/*
CENNI TEORICI
Un heap è una struttura dati gerarchica. In particolare è un albero binario completo o quasi
completo in cui ogni nodo verifica la proprietà dell'heap:
key(X) ≤ key(padre(X))
HEAPIFY
Tra i figli del nodo, si determina il figlio con chiave massima, successivamente si confronta il
figlio di
chiave massima col padre, ed eventualmente si scambiano i due nodi se non verificano la proprietà
heap.
Se è avvenuto lo scambio, si scende poi nel relativo sottoalbero per ripristinare la proprietà heap.
*/
int main()
{
    short n, i;
    short *heap; //dichiaro un puntatore ad un albero heap

    puts("----- HEAP SORT -----*\n");
    printf("Inserisci il numero di elementi da inserire nella heap: ");
    scanf("%hd", &n);
    /* Alloco un albero heap. N+1 perche' il primo elemento sara' vuoto*/
    heap = (short *)malloc(sizeof(short)*(n+1)); /* n+1, perche' ricavo figli partendo da v[1]*/
    if(!heap) exit("Errore di allocazione");

    heap[0]=-1; //primo valore array vuoto

    /* INSERISCI DATI */
    for(i=1; i<=n; i++)
    {   printf("Inserisci nodo: ");scanf("%hd", &heap[i]); }

    /* STAMPA PRIMA DI ORDINARE*/
    puts("\nARRAY INIZIALE:");
    stampa_vet(&heap[1], n);
    /* ORDINA */
    heap_sort(heap, n);
    /* STAMPA DOPO ORDINAMENTO */
    puts("ARRAY ORDINATO MEDIANTE HEAP-SORT:");
    stampa_vet(&heap[1], n); //Heap e' di n+1 elementi: io devo stampare da v[1] per le proprietà
                           //degli alberi binari con (n+1)-1, quindi n
    return 0;
}
/* heap-sort: Tenendo in considerazione "costruisci heap", dopo che ho ricavato l'heap,
prendo il primo elemento, ossia la RADICE che per definizione e' il valore piu'
grande di tutti e lo metto all'ultimo posto, decrementando n.
Quando rieffettuo' la costruzione dell'heap, la precedente radice messa
all'ultima posizione dell'array non sara' piu' considerata (per il n--) e
si trovera' gia' nella sua posizione finale ordinata.*/
void heap_sort(short V[], short n)
{
    short tmp;

    /* Finche' non ho un solo elemento (n>1 e non 0 perche' l'heap o l'albero binario parte da 1)*/
    while(n>1)
    {
        costruisci_heap(V, n);
        /* Scambia la radice(valore massimo) e mettila nella posizione piu' bassa */
        tmp = V[n];
        V[n]=V[1];
        V[1]=tmp;
        /* Ricostruisci l'heap, non considerando l'ultimo elemento messo in ordine */
        n--;
    }
}
```

```

/* costruisci heap: In questo caso, 'i' punta ai padri.
   Heapify opera mediante i figli del padre[i].
   Il ciclo principale termina quando i=0 perche' la radice
   deve essere considerata per l'heapify dato che avra' dei figli
NB 'i' SCORRE I PADRI. */
void costruisci_heap(short heap[], short n)
{
    /* Partiamo dal padre dell'ultima foglia perche' le foglie per definizione gia' sono heap
       i scorrera' i vari padri di cui controlleremo i relativi figli */
    short i=n/2;

    short esito, esito_foglia, j=0;

    /* Fin quando non si arriva oltre la radice */
    while(i>=1)
    {
        /* effettua un heapify tra heap[i] (il padre) e i figli ( heap[sx] e heap[dx] ) */
        esito = heapify(heap, i, n);
        j=esito; /* contengono l'indice di dove avviene lo scambio. Tale indice ci serve per scendere
                   nel relativo sottoalbero perche' avendo effettuato lo scambio, dobbiamo
                   verificare se abbiamo violato nel sotto albero le proprietà dell'heap
                   SE NON E' AVVENUTO LO SCAMBIO, VIENE SEGNALATO DA ESITO CON -1 */

        /* Quindi se avviene uno scambio, effettuiamo le stesse operazione nel sottoalbero
           in cui era avvenuto lo scambio.
           Cicliamo fin quando non abbiamo una foglia o non avviene uno scambio */
        while(esito>=0 && is_foglia(heap, j, n))
        {
            esito = heapify(heap, j, n);
            j=esito;
        }

        i--; //passa ai padri successivi
    }
}

/* is_foglia: controlla semplicemente se quel nodo e' una foglia */
short is_foglia(short heap[], short i, short n)
{
    if(2*i>n && (2*i)+1>n)
        return 0; //non e' foglia
    /* Altrimenti e' una foglia se */
    return 1;
}

/* heapify: rispetto all'esercizio vecchio, abbiamo una variazione di heapify.
   controllo se ci sono figli per padre (heap[i]) e inoltre mi trovo il piu' grande tra il padre
   e i figli. */
short heapify(short heap[], short i, short n)
{
    short tmp, scambio=0, figlio_sx=2*i, figlio_dx=(2*i)+1, maggiore;

    /* Controlla se sx non superi n e se il figlio sinistro sia maggiore del padre.
       Se si, conservati l'indice maggiore, ossia il figlio, altrimenti se il padre
       e' piu' grande, conserveremo l'indice del padre in maggiore*/
    if((figlio_sx<=n) && (heap[figlio_sx]>heap[i]))
        maggiore=figlio_sx;
    else
        maggiore=i;

    /* Ora confrontiamo il maggiore trovato con il figlio destro. Se la condizione e' valida
       avremo un nuovo maggiore, altrimenti rimane incavariato*/
    if((figlio_dx<=n) && (heap[figlio_dx]>heap[maggiore]))
        maggiore=figlio_dx;

    /* Se il padre e' piu' grande di uno dei 2 figli (quindi maggiore non e' l'indice padre)*/
    if(maggiore != i)
    {
        /* Scambia i 2 nodi e ritorna l'indice di maggiore, ossia quello che ora
           contiene un valore minore di maggiore. tale ci serve per scendere nel relativo
           sotto albero */
        scambia(&heap[maggiore], &heap[i]);
        return maggiore;
    }
    /* Il padre e' un heap effettuando nessuno scambio */
    return -1;
}

```

```

/* visualizza_heap: stampiamo per ogni singolo il relativo padre e figli.
   Il primo nodo che stamperà sarà la radice eventualmente trovata.*/
void visualizza_heap(short heap[], short n)
{
    short i;
    for(i=1; i<=n; i++)
    {
        printf("Nodo: %3hd Padre: %3hd ", heap[i], heap[i/2]);
        /* Controllo se sono presenti dei figli */
        if(i*2<=n) printf("Figlio sinistro: %3hd ", heap[i*2]);
        if((i*2)+1<=n) printf("Figlio destro: %3hd ", heap[(i*2)+1]);
        printf("\n\n");
    }
}
/* scambia 2 elementi per indirizzo.*/
void scambia(short *a, short *b)
{
    short tmp;
    tmp=*a;
    *a=*b;
    *b=tmp;
}
void stampa_vet(short V[], short N)
{
    int i;
    for(i=0; i<N; i++)
        printf("%hd ", V[i]);
    puts("\n");
}

```

### OUTPUT:

\*----- HEAP SORT -----\*

Inserisci il numero di elementi da inserire nella heap: 10

Inserisci nodo: 2  
 Inserisci nodo: 6  
 Inserisci nodo: 18  
 Inserisci nodo: 3  
 Inserisci nodo: 42  
 Inserisci nodo: 12  
 Inserisci nodo: 55  
 Inserisci nodo: 44  
 Inserisci nodo: 94  
 Inserisci nodo: 79

ARRAY INIZIALE:

2 6 18 3 42 12 55 44 94 79

ARRAY ORDINATO MEDIANTE HEAP-SORT:  
 2 3 6 12 18 42 44 55 79 94

### Heapsort (algoritmo base)

### 2. ordinamento heap

Per la proprietà heap il valore massimo si trova nella radice dell'albero (prima componente dell'array). Poiché nell'array ordinato il massimo deve occupare l'ultima componente, si scambiano prima ed ultima componente dell'heap.

