Esercizi di Riepilogo

Informatica A AA 2023 / 2024

Giacomo Boracchi

Dicembre 2023

giacomo.boracchi@polimi.it

Si una funzione riordina che prende in ingresso due vettori **u** e **v** (più eventuali variabili necessarie) e che riordina gli elementi del primo vettore passato in input (**u** nel nostro caso) nel seguente modo:

- tutti gli elementi di **u** che non compaiono in **v** saranno nelle prime posizioni di **u** dopo l'invocazione
- tutti gli elementi di **u** che compaiono in **v** saranno nelle ultime posizioni di **u** dopo l'invocazione

Si faccia riferimento all'esempio sotto per eventuali dubbi su come modificare **u**.

Si invochi quindi la funzione relativamente al seguente codice e si stampi il contenuto del vettore **u** a schermo.

N.B non è possibile stampare all'interno della funzione riordina

```c
#include<stdio.h>
#define N 5

int main()
{
    int u[N] = {4,1,3,7,0}, v[N] = {1,6,8,4};

    // TODO stampa u e v

    // TODO invocazione riordina

    // TODO stampa u dopo invocazione

    return 0;
}
```

Risultato atteso

u = 4 1 3 7 0
v = 1 6 8 4
dopo invocazione u = 3 7 0 4 1

Esercizio Luglio 2023

Sia data la seguente definizione di un albero contenente un carattere in ogni nodo.

```
typedef struct El{
char c;
struct El *left, *right;
}Nodo;
typedef Nodo *Tree;
```

Un albero si dice ordinato se ogni nodo soddisfa le seguenti condizione:
- il figlio di sinistra (se esiste) contiene un carattere che precede il carattere nel nodo
- il figlio di destra (se esiste) contiene un carattere che segue il carattere nel nodo

si assuma che l'albero non abbia caratteri replicati.

Si scriva una funzione **verifica** che controlla se un albero è ordinato o meno.

```c
int main()
{
    Tree t = NULL;
    t=insert(t,'k');
    t=insert(t,'d');
    t=insert(t,'q');
    t=insert(t,'b');
    t=insert(t,'p');
    t=insert(t,'z');

    stampa(t);
    // invocazione alla funzione verifica

    return 0;

}
```

Ecco due esempi di invocazione della funzione

(((b) d) k ((p) q (z)))
albero ordinato

(((b) w) k ((p) q (z)))
albero NON ordinato

Il tipo Popolo definisce un popolo con nome e numero di abitanti.

```
typedef struct{
    char nome[N];
    int pop;
}Popolo;
```

Si scriva una funzione analizzaPopoli che prende in ingresso un array di elementi di tipo Popolo -- ed ulteriori variabili aggiuntive se necessarie-- e riporta al chiamante il nome del popolo con più abitanti tra quelli contenuti nell'array ed il nome del popolo con il numero di abitanti più vicino alla media degli abitanti dei vari popoli. Per identificare il popolo più vicino alla popolazione media, si consideri la differenza di popolazione in valore assoluto.

Si invochi la funzione analizzaPopoli nel main e si stampino i nomi dei due popoli selezionati.

In riferimento all'array popolato nel main nel seguente codice (da utilizzare per sviluppare la soluzione)
https://www.dropbox.com/s/c0ckq4moeggmwpr/indiani_TODO.c?dl=0

```
strcpy(indiani[0].nome, "Cayuga");
indiani[0].pop = 652;
strcpy(indiani[1].nome, "Oneida");
indiani[1].pop = 2123;
strcpy(indiani[2].nome, "Seneca");
indiani[2].pop = 3521;
strcpy(indiani[3].nome, "Onondaga");
indiani[3].pop = 1763;
strcpy(indiani[4].nome, "Mohawak");
indiani[4].pop = 4512;
```

ci si aspetta la seguente stampa dal main:

Più numerosi: Mohawak, più vicini alla media: Oneida

Esercizio Gennaio 2022

Si scriva una funzione rimuoviValori che prende in ingresso una matrice M di interi di dimensioni NxN (con #define N 4), un intero x e ulteriori variabili aggiuntive, se necessarie.

La funzione identifica i valori di M che sono a distanza minima da x e modifica M mettendo a zero tutti i valori che stanno sulla parallela alla diagonale principale passante per i valori sopra identificati. Si guardino gli esempi sotto per evitare fraintendimenti su quali elementi mettere a zero.

N.B. Qualora ci fossero più valori in M alla stessa distanza da x, la funzione rimuoviValori ripete l'operazione per tutti i valori a distanza minima.

N.B. Si consiglia fortemente l'uso di funzioni ausiliarie per produrre un codice semplice, leggibile e compatto

Si scriva nel main l'invocazione alla funzione rimuoviValori e si stampi la matrice nel main, non in rimuoviValori.

Esercizio Gennaio 2022

M =

1 3 5 3
5 6 7 5
5 6 2 1
3 6 9 4

rimuoviValori invocata su M con x = 7

1 0 5 3
5 6 0 5
5 6 2 0
3 6 9 4

M =

1 3 5 3
5 6 7 5
5 6 2 1
3 6 9 4

rimuoviValori invocata su M con x = 8

1 0 5 3
0 6 0 5
5 0 2 0
3 6 0 4

Si considerino due alberi TA e TB (alberi di esempio e chiamate di prova sono definiti nel codice C allegato). Si scriva una funzione f che prende in ingresso TA e TB e stampa a video il numero di volte che ogni valore di TA compare in TB.

Variante: restituire solo il valore di TA che compare più volte in TB

```c
#define N 100
typedef char Tipo[N];
typedef struct ndP{
    float costo;
    int tavolo;
    Tipo nome;
    struct ndP* next;
} Piatto;
typedef Piatto* ListaP;
```

Nel seguente main viene popolata la lista ordini contenente tutti i piatti che sono

stati ordinati al ristorante. Si definisca un nuovo tipo Tavolo per gestire una lista di tavoli, ciascuno contenente: *l'identificativo del tavolo ed una lista di piatti ordinati a quel tavolo*.

Si scriva quindi una funzione organizzaPerTavoli che prende in ingresso la lista ordini e popola una lista di tavoli dove ogni nodo contiene una sottolista di piatti ordinati per quel tavolo.
Si sviluppi e si invochi nel main una funzione VisualizzaTavoli che stampa il contenuto della lista tavoli come da esempio.

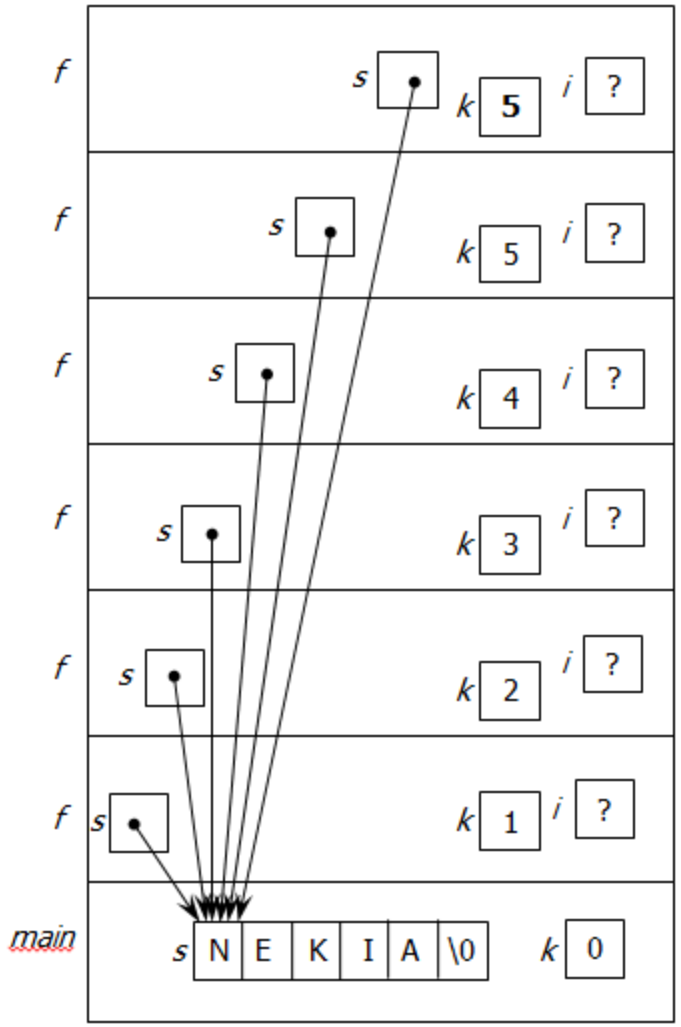
Esercizio (tdeB 9-9-2013)

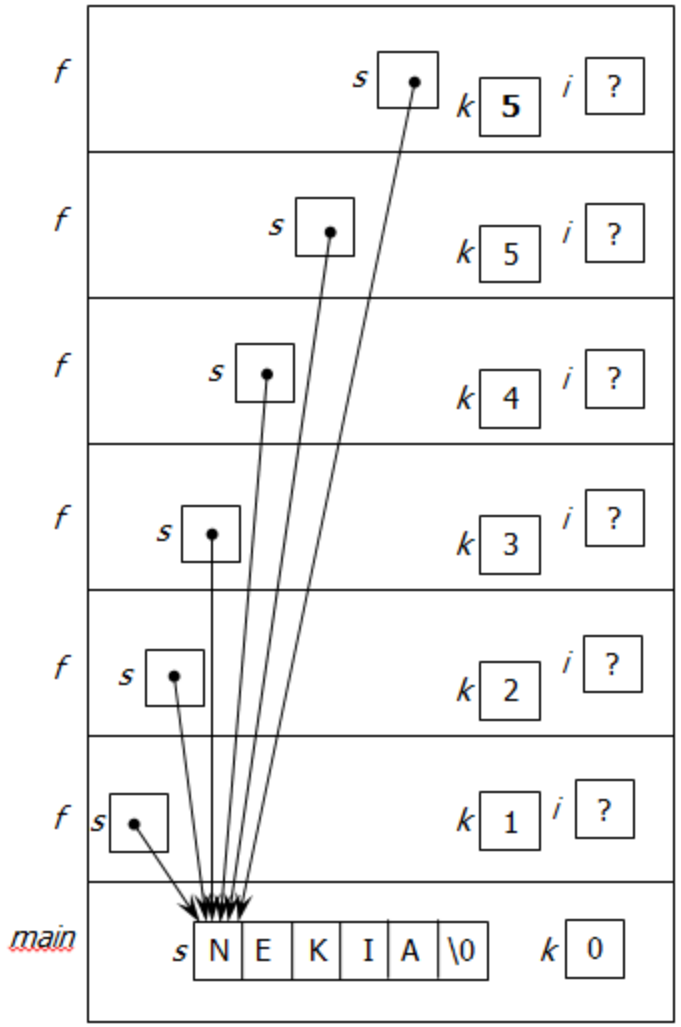
Si disegni lo **stack** dei record di attivazione nell'istante in cui la funzione f() inizia a eseguire per la prima volta l'istruzione indicata dalla freccia. Si rappresentino **tutte** le variabili (vettori: blocchi contigui; puntatori: frecce; valori indefiniti: punti interrogativi)

Dopo aver simulato l'esecuzione del programma, se ne indichi l'output

```c
int f(char * s, int k) {
  int i;
  if( strlen( s+k ) > 0 ) {
    i = f(s, ++k);
    printf("%c", *(s+k-1) + (k)%4 );
    return i;
  }
  return 42;
}
```

```c
int main() {
  char s[]="NEKIA";
  int k=0;
  k = f(s, k);
  printf(" !");
  return 0;
}
```


f
s
k 5
i ?
f
s
k 5
i ?
f
s
k 4
i ?
f
s
k 3
i ?
f
s
k 2
i ?
f
s
k 1
i ?
main
s N E K I A \0
k 0



Stampa: BINGO !

Esercizio (tde 24-2-2011)

Si dica che cosa stampa il seguente codice e si spieghi cosa calcola la funzione f

```c
#include<stdio.h>
 int f(int c, int d);
int main(){
    int c;
    c=f(4,2);
    printf ("c = %d\n", c);
    c=f(2,3);
    printf ("c = %d\n", c);
    c=f(2,4);
    printf ("c = %d\n", c);
    c=f(3,3);
    printf ("c = %d\n", c);
}
```

```c
int f(int d, int e){
    if( e > 0 )
        return f(d,e-1) + f(d,e-1);
    else
        return d;
}
```

```c
#include<stdio.h>
 int f(int c, int d);
int main(){
    int c;
    c=f(4,2);
    printf ("c = %d\n", c);
    c=f(2,3);
    printf ("c = %d\n", c);
    c=f(2,4);
    printf ("c = %d\n", c);
    c=f(3,3);
    printf ("c = %d\n", c);
}
```

```c
int f(int d, int e){
    if( e > 0 )
        return f(d,e-1) + f(d,e-1);
    else
        return d;
}
```

La funzione restituisce $d * 2^e$

TDE 26/1/2009

Si scriva una funzione che, dati in input due alberi binari con valori interi, restituisce 1 se i due alberi sono identici, o altrimenti.

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

typedef struct el{
    int v;
    struct el *left;
    struct el * right;
}Nodo;
typedef Nodo * Tree;

int somma(Tree);
int cercaMax(Tree);
Tree insert(Tree, int);
int uguali(Tree, Tree);
```

```c
int main(){
    Tree t1 = NULL, t2=NULL;

    t1=insert(t1,12); t1=insert(t1,21); t1=insert(t1,20);
    t1=insert(t1,1); t1=insert(t1,15);

    t2=insert(t2,12); t2=insert(t2,1); t2=insert(t2,20);
    t2=insert(t2,21); t2=insert(t2,15);

    printf("uguali: %d\n", uguali(t1, t2));


    return 0;
}
```

```c
int ugualiTree t, Tree s)
{
    if(t == NULL && s == NULL)
        return 1;

    if(t == NULL || s == NULL)
        return 0;

    if(t->v == s->v)
        return (ugualit->left, s->left) * ugualit->right, s->right));
    else
        return 0;
}
```

Esercizio 3 (8 punti) TDE 1 Marzo 2017

Scrivere una funzione che riceve due stringhe: *parola* e *elimina*. La funzione cerca in *parola* tutti i caratteri che compongono la stringa *elimina*, e li rimuove solamente se li trova tutti e nell'ordine in cui compaiono in *elimina*, anche se non consecutivi. L'operazione viene ripetuta finché in *parola* è contenuta un'intera istanza della stringa *elimina*. La funzione, oltre a modificare l'array *parola*, restituisce il numero di volte che ha eliminato i caratteri dell'intera stringa *elimina* da *parola*. Nell'eliminare caratteri non devono essere lasciati buchi, ma *parola* dev'essere ricompattata.

Esempio

parola: amaarrreeeemmmmarrreeaaaarrrmae

elimina: mare

La funzione trova tutte le lettere di "mare" in "aMAaRrrEeeemmmmarrreeaaaarrrmae" che diventa "aarreeemmmmarrreeaaaarrrmae"

La funzione trova tutte le lettere di "mare" in "aarreeeMmmmARrrEeaaaarrrmae" che diventa "aarreeemmmrreaaaarrrmae"

La funzione trova tutte le lettere di "mare" in "aarreeemmMrreAaaaRrrmaE" che diventa "aarreeemmrreaaarrma"

Poi non trova più tutte le lettere e restituisce 3

```c
#include<stdio.h>
#define N 40

int eliminaParola(char *, char *);
int trovaIndici(char *, char *, int *);
void rimuoviIndici(char *, int *, int);

int main()
{
    char parola[N] = "amaarrreeeemmmmarrreeaaaarrrmae";
    char elimina[N] = "mare";

    int n;

    printf("parola: %s,\nelimina: %s", parola, elimina);
    n = eliminaParola(parola, elimina);
    printf("\n\nparola: %s,\nelimina: %s\n n = %d", parola, elimina, n);

    return 1;
}
```

```c
int eliminaParola(char parola[], char elimina[])
{
    int indx[N], n = 0, i;
    while(trovaIndici(parola, elimina, indx))
    {
        /*for(i = 0; i< strlen(elimina); i++)
            printf("\n indx[%d] = %d", i, indx[i]); */

        rimuoviIndici(parola, indx, strlen(elimina));
        n++;
        printf("\nparola: %s,\nelimina: %s\n n = %d", parola, elimina, n);

    }
    return n;
}
```

```c
int trovaIndici(char parola[], char elimina[], int indx[])
{
    int i, j = 0;
    for(i = 0; i < strlen(parola); i++)
        if(parola[i] == elimina[j])
        {
            //printf("\n found %c at %d", elimina[j], i);
            indx[j] = i;
            j++;
        }
    // printf("%d, %d", j, strlen(elimina));
    if(j == strlen(elimina))
        return 1;
    else
        return 0;
}

void rimuoviIndici(char parola[], int indx[], int n)
{
    int i,j, len_p;
    for(i = n; i >= 0; i--) // parti dal fondo se no cambiano gli indici da rimuovere
    {   printf("\n rimuovo %d (%d)", indx[i], n);
        len_p = strlen(parola); // devo ricalcolare la lunghezza dopo che taglio
        // rimuovo l'i-simo elemento da parola
        for(j = indx[i]; j < len_p; j++)
            parola[j] = parola[j+1];
    }
}
```

Esercizio (tde 9-6-2009)

Si consideri la seguente definizione di un albero:

```
typedef struct EL { int dato;
                    struct EL * left, * right; } node;

typedef node * tree;
```

Si definisce livello di un nodo la sua distanza dalla radice. Si scriva una funzione che prende in ingresso un albero binario e restituisce 1 se tutti i nodi di livello pari contengono un numero pari e tutti i nodi di livello dispari contengono un numero dispari.

```
int f(tree t){
     if(t==NULL)
          return 1;
     return f2(t,0);// o 1
}

int f2(tree t,int livello){
     if(t==NULL)
          return 1;
     if(livello%2!=t->dato%2)
          return 0;
     return f2(t->left,livello+1) && f2(t->right,livello+1);
}
```

Liste Equipotenti

Due liste di interi si dicono *equipotenti* se sono di uguale lunghezza e, confrontando i valori in posizioni corrispondenti, risulta che i valori della prima lista maggiori dei corrispondenti valori nella seconda sono esattamente in numero uguale ai valori della seconda lista maggiori dei corrispondenti valori nella prima.

Si scriva una funzione che verifichi se due liste sono equipotenti.

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

typedef struct nodo{
    int v;
    struct nodo *next;
}Nodo;
typedef Nodo * Lista;

Lista ins_testa(Lista, int);
void print_lista(Lista);
int equipotenti(Lista, Lista);
int equipotenti_ric(Lista, Lista);
int check(Lista, Lista, int);

int main(){
    Lista l1=NULL, l2=NULL, l3=NULL, l4=NULL;

    l1=ins_testa(l1,2);
    l1=ins_testa(l1,3);
    l1=ins_testa(l1,5);
    l1=ins_testa(l1,1);
    print_lista(l1);

    l2=ins_testa(l2,3);
    l2=ins_testa(l2,5);
    l2=ins_testa(l2,1);
    l2=ins_testa(l2,0);
    printf("Primo check, l2:\n");
    print_lista(l2);
    printf("Risultato: %d\n",equipotenti_ric(l1,l2));

    l3=ins_testa(l3,2);
    l3=ins_testa(l3,3);
    l3=ins_testa(l3,5);
    printf("Secondo check, l3:\n");
    print_lista(l2);
    printf("Risultato: %d\n",equipotenti_ric(l1,l3));

    l4=ins_testa(l4,2);
    l4=ins_testa(l4,10);
    l4=ins_testa(l4,10);
    l4=ins_testa(l4,10);
    printf("Terzo check, l4:\n");
    print_lista(l2);
    printf("Risultato: %d\n",equipotenti_ric(l1,l4));

    return 0;
}
```

```
int equipotenti(Lista l1, Lista l2){
    int maggiori = 0, minori = 0;
    while(l1 != NULL && l2 != NULL){
        if(l1->v>l2->v)
            maggiori++;
        else if(l1->v<l2->v)
            minori++;
        l1=l1->next;
        l2=l2->next;
    }
    return (l1==l2) && (maggiori==minori);
}
```

```
int equipotenti_ric(Lista l1, Lista l2){
    return check(l1,l2,0);
}


int check(Lista l1, Lista l2, int c){
    if(l1==NULL || l2==NULL)
        return (l1==l2 && c==0);

    if(l1->v > l2->v)
        c++;
    else if (l1->v < l2->v)
        c--;
    return check(l1->next,l2->next,c);
}
```

```
Lista ins_testa (Lista l, int val) {
    Lista newPtr;
    newPtr = (Lista) malloc(sizeof(Nodo));
    if(newPtr!=NULL){
        newPtr->v=val;
        newPtr->next=l;
    }else{
        printf("No memory available\n");
    }
    return newPtr;
}

void print_lista(Lista l) {
    if (l == NULL)
        printf(" ---| \n");
    else {
        printf(" %d ---> ", l->v);
        print_lista(l->next);
    }
}
```

Max Ricorsivo

Scrivere un programma che calcola il massimo di un vettore di interi di lunghezza `N` (costante predenita) in modo ricorsivo.

```c
#include <stdio.h>
define N 10

int maxArray(int *array, int n);

int main() {
    int test[N] = {2, 3, 9, 2, 13, 4, 34, 2, 9, 5};

    printf("Max = %d \n", maxArray(test,N));

    return 0;
}

int maxArray(int *array, int n) {
    int maxsub;
    if (n == 1)
        return array[0];
    if (n >= 2) {
        maxsub = maxArray(&array[1],n-1);
        if (array[0] > maxsub)
            return array[0];
        else
            return maxsub;
    }
    return -1;  /* non raggiungibile */
}
```

Cosa fa?

```c
#define MATR "123456"
void f(char * str);

int main() {
  char V[7] = MATR;
  f(V+3);
  return 0;
}
```

```c
void f(char * str) {
    printf("%c", *str);
    str=str+1;
    if(strlen(str) == 0)
      return;
    else {
      f(str);
      printf("%c", *str);
      return;
    }
}
```

Cosa fa?

```c
#include<stdio.h>
#include<stdlib.h>
#define M 6

void m(int k);
void f(int * a, int b, int c);
void g(int* d, int e);

int main(){
    m(1);
    m(2);
    m(4);
    m(12);
    m(24);
}
```

Cosa fa?

```
void m(int k) {
    int* h = (int*) malloc(sizeof(int) * M);

    f(h + M - 1, k, 0);

    g(h, M);

    if(h)
      free(h);
}
```

```
void f(int * a, int b, int c) {
    if(c == M)
        return;
    c++;
    *(a) = b % 2;
    a--;

    f(a, b/2, c);
}

void g(int* d, int e) {
    if (e) {
        printf("%d ",*d);
        d++;
        g(d, e - 1);
    } else printf("\n");
}
```

Esercizio

Si progetti una **funzione ricorsiva** che svolge il compito seguente. Siano dati due vettori V1 e V2, di dimensione N1 e N2, rispettivamente (con 1< N2 < N1).

La funzione restituisce **1 se tutti gli elementi del vettore V2 si trovano consecutivi e all'inizio del vettore V1, ma nell'ordine inverso** rispetto a quello in cui essi figurano in V2. Altrimenti (ovvero se questo non si verifica) la funzione restituisce valore 0.

Esempio: $V1 = [2; 3; 9; 2; 13; 4; 34; 2; 9; 5]$e $V2 = [13; 2; 9; 3; 2]$ soddisfano la proprietà desiderata, mentre se $V2 = [5; 2; 5; 2; 2]$ la proprietà non e soddisfatta.

```c
#include <stdio.h>
#define N1 10
#define N2 5

int contiene(int v1[], int v2[], int n);

int main() {
  int v1[N1] = {123, 2, 3, 9, 2, 13, 4, 34, 2, 9};
  int v2[N2] = {13, 9, 3, 2};

 printf("Result2 = %d \n", contiene2(v1, v2, 0));

  return 0;
 }

int contiene(int v1[], int v2[], int n)
{
    if(n == N2)
        return 1;
    if(v1[n] == v2[N2 - 1 - n])
        return contiene(v1, v2, ++n);
    return 0;
}
```

```c
#include <stdio.h>
#define N1 10
#define N2 5

int contiene(int v1[], int v2[], int iniz, int fine);

int main() {
    int v1[N1] = {2, 3, 9, 2, 13, 4, 34, 2, 9, 5};
    int v2[N2] = {4, 2, 9, 3, 2};

    printf("Result = %d \n", contiene(v1, v2, 0, N2 - 1));

    return 0;
 }
int contiene(int v1[], int v2[], int iniz, int fine) {
    if (fine < 0)
        return 1; /* v2 è vuoto, quindi termino con esito positivo. */
    if (iniz > N1 - 1)
        return 0; /* v1 è vuoto, quindi termino con esito negativo. */
    // passo induttivo e chiamata ricorsiva
    if (v1[iniz] == v2[fine])
        return contiene(v1, v2, iniz+1, fine-1);

    return 0;
}
```

Si progetti una funzione ricorsiva che svolge il compito seguente. Siano dati due vettori V1 e V2, di dimensione N1 e N2, rispettivamente (con 1< N2 < N1).

La funzione restituisce **1 se tutti gli elementi del vettore V2 si trovano all'inizio del vettore V1, ma nell'ordine inverso** rispetto a quello in cui essi figurano in V2, **ma non necessariamente in posizioni immediatamente consecutive;** Altrimenti (ovvero se questo non si verifica) la funzione restituisce valore 0.

Esempio: $V1 = [2; 3; 9; 2; 13; 4; 34; 2; 9; 5]$ e $V2 = [5; 2; 34; 2; 2]$ soddisfano la proprietà

desiderata, mentre se $V2 = [5; 2; 5; 2; 2]$ la proprietà non e soddisfatta.

```c
#include <stdio.h>
#define N1 10
#define N2 5

int contiene(int v1[], int v2[], int iniz, int fine);

int main() {
    int v1[N1] = {2, 3, 9, 2, 13, 4, 34, 2, 9, 5};
    int v2[N2] = {4, 2, 9, 3, 2};

    printf("Result = %d \n", contiene(v1, v2, 0, N2 - 1));

    return 0;
 }
int contiene(int v1[], int v2[], int iniz, int fine) {
    if (fine < 0)
        return 1; /* v2 è vuoto, quindi termino con esito positivo. */
    if (iniz > N1 - 1)
        return 0; /* v1 è vuoto, quindi termino con esito negativo. */
    // passo induttivo e chiamata ricorsiva
    if (v1[iniz] == v2[fine])
        return contiene(v1, v2, iniz+1, fine-1);

    return contiene(v1, v2, iniz+1, fine);
}
```

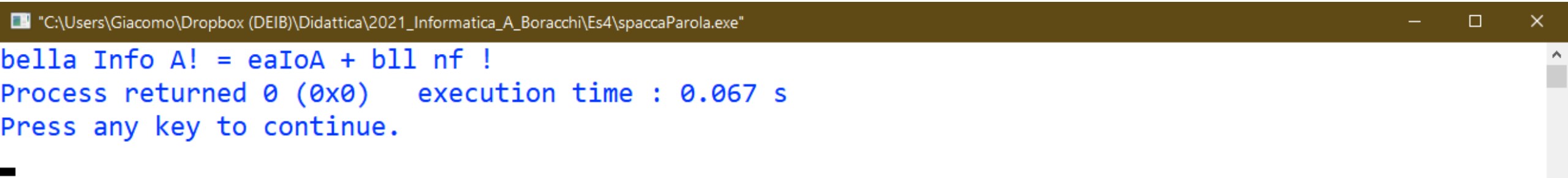
Funzione con Array

Si scriva una funzione che prende in ingresso una stringa – e altre variabili che si ritiene necessario – e riporta al chiamante una stringa di vocali ed una stringa di altre lettere

Si invochi la funzione nel main

```c
#include<stdio.h>
#include<string.h>
#define N 20

void spaccaParola(char*, char*, char*);
int vocale(char);

int main()
{
char parola[N] = "bella Info A!";
char vocali[N], altro[N];

spaccaParola(parola, vocali, altro);
printf("%s = %s + %s", parola, vocali, altro);

return 0;
}

int vocale(char c)
{
    if(c=='a' || c == 'A')
        return 1;
    if(c=='e' || c == 'E')
        return 1;
    if(c=='i' || c == 'I')
        return 1;
    if(c=='o' || c == 'O')
        return 1;
    if(c=='u' || c == 'U')
        return 1;
    return 0;
}
```

```c
void spaccaParola(char in[N], char voc[N], char other[N])
{
    int i= 0, k =0, j=0, len =0;
    len =strlen(in);
    while(i<len)
    {
        if(vocale(in[i]))
        {
            voc[j]=in[i];
            j++;
        }
        else
        {
            other[k]=in[i];
            k++;
        }
        i++;
    }
    voc[j] ='\0';
    other[k]='\0';
}
```


"C:\Users\Giacomo\Dropbox (DEIB)\Didattica\2021_Informatica_A_Boracchi\Es4\spaccaParola.exe"
bella Info A! = eaIoA + bll nf !
Process returned 0 (0x0)   execution time : 0.067 s
Press any key to continue.

Esercizio (tde 9-2-2010)

Si progetti e codifichi una funzione C

```
int numSommaDiff (int a[], int len)
```

che riceve in ingresso un array **a** di interi. La funzione restituisce 1 se esiste almeno un elemento in **a** pari alla somma degli elementi che lo seguono diminuita della somma degli elementi che lo precedono. Altrimenti restituisce 0.

Ad esempio, nel vettore 1 2 1 **20** -6 16 14 il 20 soddisfa la proprietà descritta. Quindi la funzione restituisce 1.

Se invece si considera il vettore 1 2 1 20 -14 20 14 nessun numero soddisfa la proprietà descritta. Quindi la funzione restituisce 0.

```c
int somma(int v[],int da, int a) {
  int i, somma=0;
  for(i = da; i <= a ; i++)
    somma=v[i];

  return somma;
}



int numSommaDiff(int a[]) {
  int i;
  for(i = 0; i < N; i++)
        if(a[i] == somma(a, i+1, N-1)-somma(a,0,i-1))
          return 1;

  return 0;
}
```

Si considerino due alberi T1 e T2 definiti nel codice C seguente. Si scriva una funzione *alberiOccorrenze* che prende in ingresso T1 e T2 -- più eventuali variabili aggiuntive se necessarie -- e riporta al chiamante il valore di T1 che compare più volte in T2.

Dopo l'invocazione della funzione si stampi nel main un messaggio come:

T1: (((9) 3 (10)) 7 ((5) 8 ((11) 12 (9))))
T2: ((((6) 12 (11)) 7 (9)) 7 ((10) 7 (9)))
T3: ((((6) 12 (2)) 4 (3)) 3 ((10) 3 (9)))

max occorrenze di T1 in T2: 7 appare 3 volte

max occorrenze di T2 in T1: 9 appare 2 volte

max occorrenze di T1 in T3: 3 appare 3 volte

max occorrenze di T2 in T3: 12 appare 1 volte

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

typedef struct n {
        int val;
        struct n * left;
        struct n * right;
} nodo;
typedef nodo * Albero;

Albero createVal(int val);
Albero creaAlbero1();Albero creaAlbero2();Albero creaAlbero3();
void print(Albero t);
void stampa(Albero T);
int maxOccorrenze(Albero TA, Albero TB, int *valore);
int contaOccorrenze(Albero T, Albero nod);
```

```c
int main(){
    int ris=0, valore = -1;
    Albero T1,T2,T3;
    T1 = creaAlbero1(); T2 = creaAlbero2(); T3 = creaAlbero3();
    printf("\nT1: "); stampa(T1);
    printf("\nT2: "); stampa(T2);
    printf("\nT3: "); stampa(T3);

    //
    // TODO: SVILUPPARE QUI DENTRO QUANTO RICHIESTO
    //

    //FUNZIONI AUSILIARIE CON PARAMETRI DIVERSI SONO MOLTO CONSIGLIATE

    printf("\nmax occorrenze di T1 in T2: %d appare %d volte", valore, maxOccorrenze(T1, T2, &valore));
    printf("\nmax occorrenze di T2 in T1: %d appare %d volte", valore, maxOccorrenze(T2, T1, &valore));
    printf("\nmax occorrenze di T1 in T3: %d appare %d volte", valore, maxOccorrenze(T1, T3, &valore));
    printf("\nmax occorrenze di T2 in T3: %d appare %d volte", valore, maxOccorrenze(T2, T3, &valore));

   //printf("\n occorrenze di %d in T3: %d", T1->left->val, contaOccorrenze(T3, T1->left));

    return 0;
}
```

```c
int maxOccorrenze(Albero TA, Albero TB, int *valore)
{
    int occ, occL, occR, massimo, valL, valR;

    if(TA==NULL)
        return 0;

    *valore = TA->val;
    occ = contaOccorrenze(TB, TA);
    occL = maxOccorrenze(TA->left, TB, &valL);
    occR = maxOccorrenze(TA->right, TB, &valR);

    massimo = occ;

   if(massimo < occL && TA->left != NULL)
   {
         *valore = valL;
         massimo = occL;
   }

   if(massimo < occR && TA->right != NULL)
   {
         *valore = valR;
         massimo = occR;
   }

   return massimo;
}
```

```c
int max3(int a, int b, int c)
{
    return max(max(a, b), c);
}

int max(int a, int b)
{
    if(a > b)
        return a;
    return b;
}

int contaOccorrenze(Albero T, Albero nod)
{
    if(T==NULL)
        return 0;

    if (nod == NULL)
        return 0;

    if(T->val == nod->val)
        return 1 + contaOccorrenze(T->left, nod) + contaOccorrenze(T->right, nod);
    else
        return contaOccorrenze(T->left, nod) + contaOccorrenze(T->right, nod);

}
```

TDE Gennaio 2020

Si scriva una funzione **comprimiStringa** che prende in ingresso una sequenza di caratteri composta solo da lettere dell'alfabeto e la comprime nel modo seguente: ogni sequenza di due o più caratteri identici consecutivi viene sostituita da due caratteri

[NUMERO] [CARATTERE]

dove NUMERO è il numero di occorrenze e CARATTERE è il carattere nella stringa.

Ad esempio la stringa **ciiiaaaaooooo** viene compressa in **c3i4a5o**

Si scriva poi una funzione **decomprimiStringa** che prende in ingresso una sequenza di caratteri compressa da **comprimiStringa** e restituisce la sequenza originale.

Ad esempio la stringa **c3i4a5o** viene decompressa in **ciiiaaaaooooo**

Per semplificare lo sviluppo, si assuma che **non possano capitare che vi siano più di 9 occorrenze consecutive** dello stesso carattere e che nessuna stringa sia più lunga di 1000 caratteri.

Si ricordi che per passare da un valore char all'intero che esso rappresenta basta sottrarre il carattere **'0'**.

'1' → 1

'1' - '0'

```c
#include<stdio.h>
#define N 100
void comprimiStringa(char *, char *);
void deComprimiStringa(char *, char *);

int main()
{
    char in[N], out[N], decomp[N];

    printf("inserire stringa: ");
    scanf("%s", in);
    comprimiStringa(in, out); // la funzione non può restituire un array (dichiarato nel R.A. della funzione
    printf("comprimi: \n %s -> %s\n", in, out);
    deComprimiStringa(out, decomp);
    printf("decomprimi: \n %s -> %s\n", out, decomp);
    printf("\nsanity Check: \n %s\n %s", in, decomp);

    return 0;
}
```

```c
void comprimiStringa(char * in, char *out)
{
    int len, indxCons, indxIn = 0, indxOut = 0;
    len = strlen(in);
    while(indxIn < len) // preferisco il while perchè l'incremento di indxIn è regolato anche da
 quanti caratteri identici consecutivi incontra
    {
        indxCons = 1;
        while(indxIn + indxCons < len && in[indxIn + indxCons] == in[indxIn])
            indxCons++;

        if (indxCons > 1)
        {
            out[indxOut] = (char) indxCons + '0';
            indxOut++;
        }

        out[indxOut] = in[indxIn];
        indxOut++;
        indxIn += indxCons;
    }
    out[indxOut] = '\0';
   }
```

```
void decomprimiStringa(char * in, char * out)
{
    int indxIn, indxOut = 0, count =0;
    char c;
    indxIn = 0;

    while(indxIn <strlen(in)) // non usare il for se no ti trovi a dover modificare la variabile de
l loop nel corpo, sconsigliato
    {
        if(in[indxIn] > '1' && in[indxIn] <= '9')
        {   count = in[indxIn] - '0';
            c = in[indxIn + 1]; // carattere successivo
            while(count > 0)
            {
                out[indxOut] = c;
                indxOut++;
                count--;
            }
            indxIn+=2; // devi saltare oltre il carattere
        }
        else
        {
            out[indxOut] = in[indxIn];
            indxOut++;
            indxIn++;
        }
    }
    out[indxOut] = '\0';
    }
```

Un esercizio su un toro (senza corna)

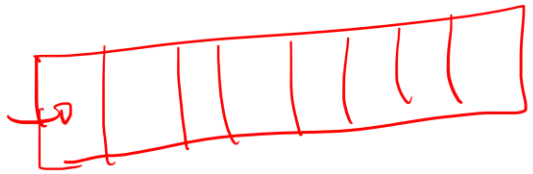
da 40_1_tdeB

La funzione **… toro(…)** riceve come parametro un vettore di interi e la specifica della sua dimensione, e alloca e restituisce una lista **dinamica** "circolare" di interi che contiene solo i valori del vettore positivi e divisibili per 11. Ovviamente la lista può essere circolare solo se non è vuota, quindi si suggerisce di renderla circolare solo alla fine dell'analisi del vettore

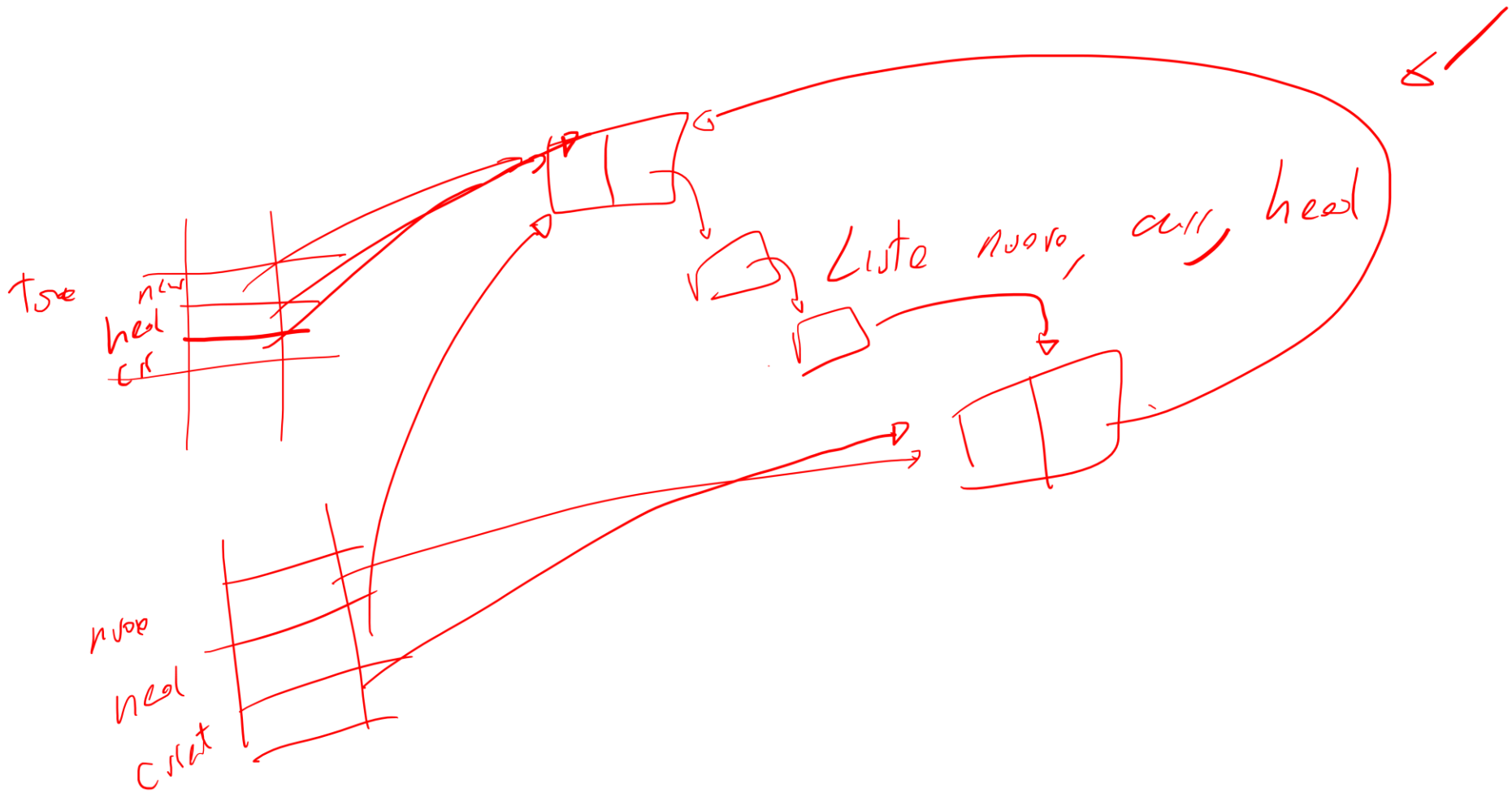
[N.B. una lista è circolare se l'ultimo elemento invece di avere un puntatore a NULL ha un puntatore alla testa].

Si definiscano in C la struttura della lista e il prototipo della funzione **toro**. Si codifichi in C la funzione (unitamente alle eventuali funzioni di supporto).

vettore



if (vett [i] %. 11 == 0)



```c
#include<stdio.h>
#define N 5

typedef struct nodo{
    int v;
    struct nodo *next;
}Nodo;
typedef Nodo * Lista;

Lista toro(int vett[],int dim);

int main()
{
    int vettore[N] = {11, 12, 13, 121, 22};
    Lista l;
    int i;

    for(i = 0; i<N; i++)
        printf("%d ", vettore[i]);

    l = toro(vettore, N);

    stampaLista(l, 100);
    return 0;
}
```

```c
Lista toro(int vettore[], int len)
{
    Lista head, curr, toAdd;
    int i;

    head = NULL;

    for(i = 0; i < len; i++)
        if(vettore[i] % 11 == 0)
        {
            // alloco il nuovo nodo da aggiungere
            toAdd = (Nodo *) malloc(sizeof(Nodo));
            toAdd->v = vettore[i];
            toAdd->next = NULL;

            if(head == NULL)
                {
                    head = toAdd;
                    curr = head;
                }
            else
            {
                curr->next = toAdd;
                curr = curr->next;
            }
         }
    if(curr != 0) // rendo la lista ciclica (solo alla fine e se non vuota)
        curr->next = head;

    return head;
}
```

```c
void stampaLista(Lista l, int n)
{
    int i = 0;
    printf("\n[ ");
    while(l != NULL && i < n)
    {
        printf("%d --> ", l->v);
        l = l->next;
        i++;
    }
    printf("]\n");

}
```

Si scriva una funzione che calcola la differenza simmetrica degli elementi di due liste

- ordinate in senso crescente
- prive di duplicati,

restituendola come una nuova lista (allocata allo scopo), anch'essa ordinata.

La differenza simmetrica è costituita dagli elementi che appartengono a una delle due liste ma non all'altra lista (contiene cioè tutti gli elementi che non sono in comune alle due liste).

```
lista1: 1 --->  2 --->  10 --->  20 --->  ---|
lista2: 2 --->  3 --->  4 --->  10 --->  ---|
diff simmetrica: 1 --->  3 --->  4 --->  20 --->  ---|
```

Si consideri la seguente definizione di lista:

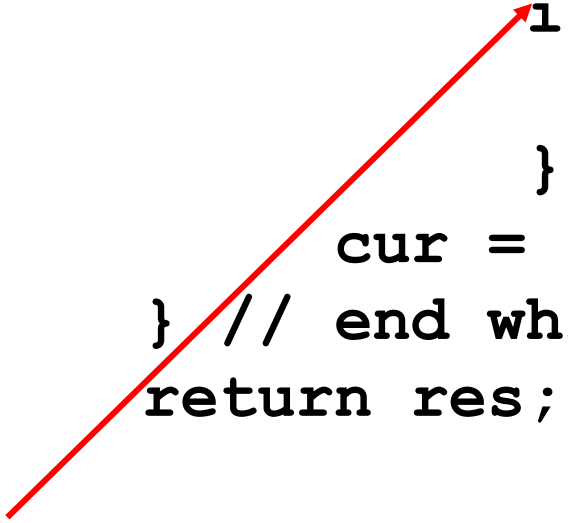
```
typedef struct EL {
int dato;
struct EL * next;
} nodo;
typedef nodo * lista;
```

Scrivere una funzione -- e tutte le funzioni ausiliarie per facilitare la risoluzione -- che prende in ingresso due liste e restituisce una nuova lista contenente i valori che compaiono in entrambe le liste lo stesso numero di volte. La nuova lista non deve contenere valori duplicati (anche se questi compaiono più volte nelle due liste di partenza).

```
typedef struct EL {
      int dato;
      struct EL * next;
} Nodo;

Nodo * Lista;

Lista InsInTesta (Lista lista, int elem) {
      Lista punt;
      punt = (Lista) malloc(sizeof(Nodo));
      punt->dato = elem;
      punt->next = lista;
      return punt;
}
```

```
int ContaInLista (Lista lista, int elem) {
      if (lista == NULL)
            return 0;
      else {
            if (lista->dato == elem)
                  return 1 + ContaInLista(lista->next, elem);
            else
                  return ContaInLista(lista->next, elem);
      }
}
```

```
Lista Intersezione (Lista lista_1, Lista lista_2) {
      Lista res = NULL;
      Lista cur = lista_1;
      int count;
      while (cur != NULL) {
            count = ContaInLista(lista_1, cur->dato);
            if (count == ContaInLista(lista_2, cur->dato)) {
                  count = ContaInLista(res, cur->dato);
                  if (count == 0)
                        res = InsInTesta(res, cur->dato);
                  }
            cur = cur->next;
      } // end while
      return res;
}
```



Occorre controllare che l'elemento non sia già stato inserito nella lista intersezione

Biliardo (variante da Febbraio 2019)

Si consideri un tavolo da biliardo rappresentato da una matrice

```
int M[R][C];
```

in cui ogni elemento rappresenta una possibile coordinata all'interno del tavolo. Si supponga che inizialmente tutte le celle valgano 0.

Si assuma ora che una palla all'interno del tavolo possa muoversi solo lungo 4 direzioni 45°, 135°, 225°, 315°, che quando colpisce una sponda rimbalzi seguendo le leggi della fisica (e.g. se colpisce una sponda verticale in direzione 45° poi prosegue in direzione 135°). Si ignori il caso in cui la palla colpisce uno degli angoli del biliardo e che dopo aver toccato la quarta sponda si arresti.

Si scriva una funzione tiroMigliore che prende in ingresso la matrice M e due coordinate intere (che supponiamo essere ammissibili) che indicano la posizione della palla. La funzione deve:

1) calcolare qual è la "direzione migliore", cioè quella che corrisponde al tiro più lungo
2) modificare la matrice M in modo da sommare +1 in ogni cella attraversata dalla palla lanciata lungo la direzione migliore.

Si supponga di avere a disposizione la funzione

```
void direzioni(int angolo, int * sp_r, int * sp_c);
```

Parte da posizione 6,1 in direzione 315

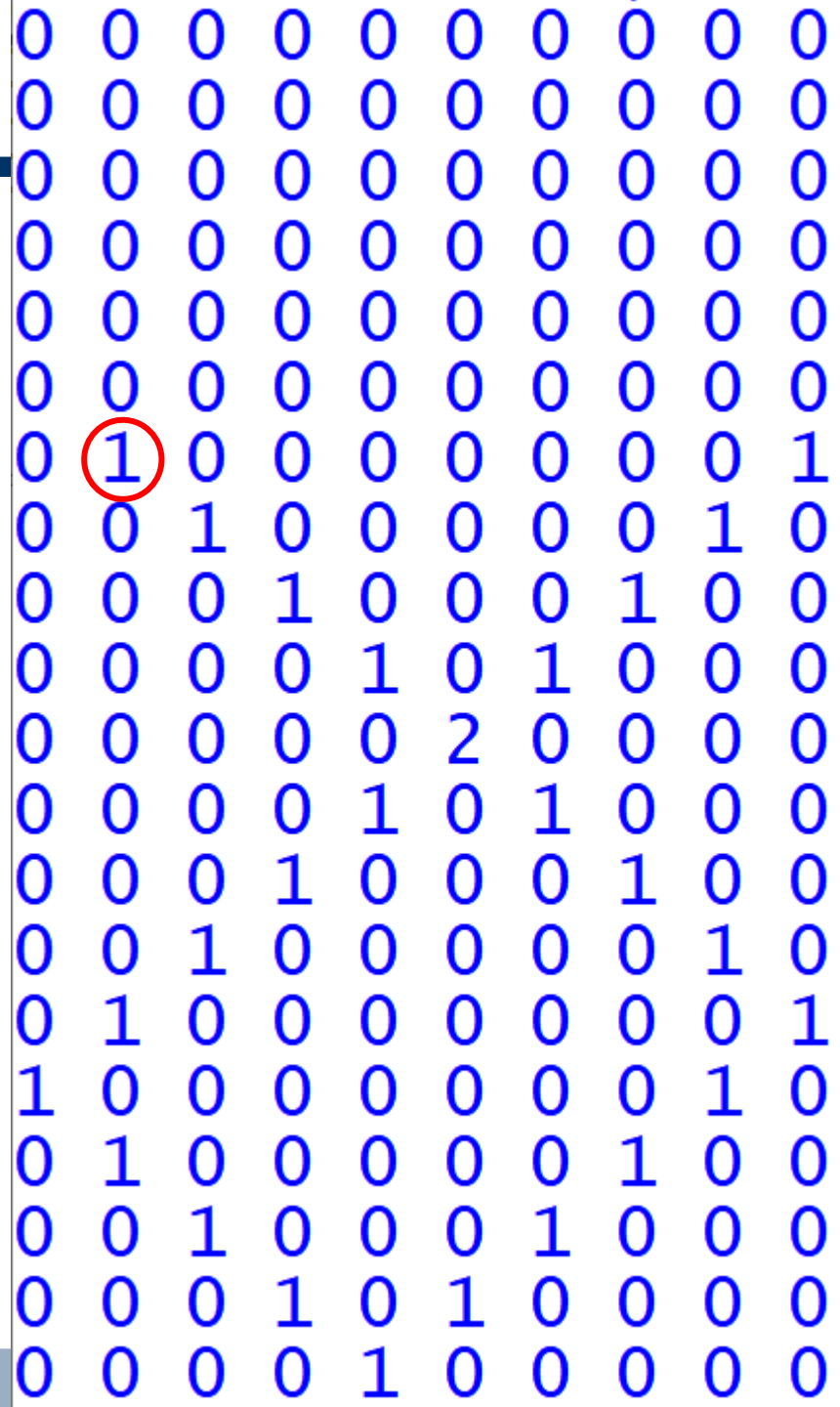
0 dove la pallina non passa

1 dove passa 1 volta

2 dove passa 2 volte

…

E si ferma alla 4ta sponda



```
#include<stdio.h>
   #define R 20
   #define C 10
   void direzioni(int, int*, int*);
   int traiettoria(int M[][C], int, int, int);
   void stampa(int M[][C]);
```

```
int main()
    {
        int M[R][C] = {0};
        int i, j, k, max, i_max, tot, pr = 6, pc = 1;
        int direzioni[4] = {45, 135, 225, 315};

        max = 0;
        i_max = 0;
        for(i = 0; i < 4; i++)
        {
            tot = traiettoria(M, pr, pc, direzioni[i]);
            printf("\ndirezione: %d, %d", direzioni[i], tot);
            if(max < tot)
            {
                tot = max;
                i_max = i;
            }
            for(j = 0; j < R; j ++)
                for(k = 0; k < C; k++)
                    M[j][k] = 0;
        }
```

```
    tot = traiettoria(M, pr, pc, direzioni[i_max]);
    printf("\ndirezione: %d, %d", direzioni[i_max], tot);
    stampa(M);

    return 0;
}
```

```
int traiettoria(int M[][C], int pr, int pc, int angle)
    {
        int dr, dc, tot, i, j, bumps = 0;

        M[pr][pc] = 1;
        tot = 0;

        direzione(angle, &dr, &dc);

        while(bumps < 4)
                {
                    pc += dc;
                    pr += dr;
                    M[pr][pc]++;
                    tot++;

                    if(pr == 0 || pr == R - 1)
                    {
                       dr = -dr;
                       bumps++;
                    }

                    if(pc == 0 || pc == C -1)
                    {
                        dc = -dc;
                        bumps++;
                    }
                }
        return tot;
    }
```

```
void direzione(int angle, int* dr, int* dc)
   {
       if(angle == 45)
           {
               *dr = -1;
               *dc = 1;
           }

       if(angle == 135)
           {
               *dr = -1;
               *dc = -1;
           }

       if(angle == 225)
           {
               *dr = 1;
               *dc = -1;
           }

       if(angle == 315)
           {
               *dr = 1;
               *dc = 1;
           }

   }
```

```
void stampa(int M[][C])
    {
     int i,j;
     printf("\n");
     for(i = 0; i < R; i++)
             {
                 for(j = 0; j < C; j++)
                     printf("%d ", M[i][j]);
                 printf("\n");
             }
    }
```

Mistero (Gen 2019)

```c
#include <stdio.h>

void mistero1(int x, int y);
int mistero2(int *x, int y);
int mistero3(int x, int * y, int z);

int main() {
  char c;int i, v[4] = { 15, 4, 12, 625 };
  for( i=0; i<4; i++ ) {
    printf("\n%d: ", v[i]); mistero1(v[i],2);
  }
  return 0;
}

void mistero1(int x, int y) {
  int z = mistero2(&x,y);
  if( z > 1 )
    printf("%d ", z);
  if( x > 1 )
    mistero1(x,y);
}

int mistero2(int *x, int y) {
  if( *x <= 0 )
    return 0;
  if( *x == 1 )
    return 1;
  return mistero3(y,x,1);
}

int mistero3(int x, int * y, int z) {
  if( *y % x != 0 )
    return mistero3( x+z, y, z );
  *y = *y / x;
  return x;
}
```

Esercizio 5 (12 punti)

Si definisca un tipo di dato atto a contenere una lista di studenti con i campi numero di matricola, nome, cognome e voto all'esame di "Informatica A". Si implementi quindi la funzione eliminaStudente che rimuove da una lista definita come sopra tutti gli studenti che hanno conseguito un punteggio all'esame minore di 18.

```
typedef struct studente_s {
        int matr, voto;
        struct studente_s * next;
} studente;
```

```
studente* elimina(studente* l){
 studente *tmp,*e;
 if(l==NULL)
        return l;
 tmp=l;
while(tmp->next != NULL)
        if((tmp->next->voto)<18){
                e=tmp->next;
                tmp->next=tmp->next->next;
                free(e);
        }
        else{
                tmp=tmp->next;     }
if((l->voto)<18){
        e=l;
        l=l->next;
        free(e);
        }
return l;
}
```

Esercizio 5 (12 punti)

Si implementi quindi la funzione listaDiStudendiPerVoto che riceve una lista definita come sopra e restituisce una lista di liste di studenti composta da sottoliste costruite dividendo gli studenti per voto ottenuto. Avremo quindi la sottolista degli studenti che hanno preso 18, quella degli studenti che hanno preso 19, e così via fino alla lista degli studenti che hanno preso 30 (per semplicità non si consideri il “30 e Lode” e si immagini non esistano voti mai assegnati).

Soluzione vista nella lezione sui Files

```
int verifica(char s[], char M[][N], int i, int j){
int flag=0, h,k;
if(s[0]==M[i][j]) {
if(strlen(s)==1)
        return 1;
for (h = i - 1; h <= i + 1 && !flag; h++)
        for (k = j - 1; k <= j + 1 && !flag; k++)
                if (h < N && h >= 0 && k < N && k >= 0 && (h!=i || k!=j))
                        flag = verifica(&s[1], M, h, k);
return flag; }
else
        return 0; }
```

Esercizio 6 (4 punti)

Si consideri la seguente definizione di un albero binario

```
typedef struct nodeS { int val;
                       struct nodeS * left, * right; } node;
typedef node * tree;
```

Scrivere una funzione che riceve un albero T e restituisce 1 se esiste almeno un percorso dalla radice ad una foglia in cui si alternano elementi pari e dispari, o altrimenti.

Esercizio (tde 27-2-2013)

Si progetti e codifichi una funzione che riceve in ingresso un intero k e una lista di matrici così definita

```
#define N 100
typedef struct NodeM { int numeri[N][N];
                       struct NodeM * next; } NodoM;
typedef NodoM * ListaM;
```

e restituisce una lista di interi del tipo

```
typedef struct NodeI { int numero;
                       struct NodeI * next; } NodoI;
typedef NodoI * ListaI;
```

contenente il più piccolo intero maggiore di k contenuto in ognuna della matrici (se esiste).

```
int cerca(int **numeri, int k) {
        int cur = k, i, j;
        for (i = 0; i < N; i++) {
                for (j = 0; j < N; j++) {
                        if (numeri[i][j] > k) {
                                if (cur == k) {
                                        cur = numeri[i][j];
                                }
                                else {
                                        if (numeri[i][j] < cur) {
                                                cur = numeri[i][j];
                                        }
                                }
                        }
                }
        }
        return cur; }
```

```
ListaI f(ListaM lista, int k) {
        ListaI head = NULL, tmp;
        ListaM iteratore = lista;
        int i;
        while (iteratore != NULL) {
                i = cerca(iteratore->numeri, k);
                if (i > k) {
                        tmp = malloc(sizeof(NodoI));
                        if (tmp == NULL) {
                                // gestione dell'errore
                                return NULL;
                        }
                        tmp->numero = i;
                        tmp->next = head;
                        head = tmp;
                }
                iteratore = iteratore->next;
        }
        return head; }
```

Max Ricorsivo

Scrivere un programma che calcola il massimo di un vettore di interi di lunghezza `N` (costante predenita) in modo ricorsivo.

Esercizio (tde 9-6-2009)

Si consideri la seguente definizione di un albero:

```
typedef struct EL { int dato;
                    struct EL * left, * right; } node;

typedef node * tree;
```

Si definisce livello di un nodo la sua distanza dalla radice. Si scriva una funzione che prende in ingresso un albero binario e restituisce 1 se tutti i nodi di livello pari contengono un numero pari e tutti i nodi di livello dispari contengono un numero dispari.

```
int f(tree t){
     if(t==NULL)
          return 1;
     return f2(t,0);// o 1
}

int f2(tree t,int livello){
     if(t==NULL)
          return 1;
     if(livello%2!=t->dato%2)
          return 0;
     return f2(t->left,livello+1) && f2(t->right,livello+1);
}
```

Sia dato un albero binario definito come segue:

```
typedef struct nodeS { int v;
                       struct nodeS * left, right; }
               node;

typedef node * tree;
```

Si scriva una funzione:

```
int f(tree t)
```

che, dato in ingresso l'albero, restituisce 1 se esiste un percorso a profondità massima che alterna valori pari e dispari nei nodi.

```
int f(Tree t)
{

        if (t == NULL)
                return 1;

        if (t->v % 2 == 0 ) // devo
percorrere solo un ramo
                return f_aux(t, 0);

        if (t->v % 2 == 1)
                return f_aux(t, 1);
}
```

```
int f_aux(Tree t, int lev)
{
if (t == NULL)
                return 0;

if(lev %2 != t->v %2)
        return 0;

if (depth(t->right) == depth(t->left))
        return f_aux(t->right, lev + 1) ||
                f_aux(t->left, lev + 1) ;

if (depth(t->right) > depth(t->left))
        return f_aux(t->right, lev + 1);

return f_aux(t->left, lev + 1);
}
```

```
int depth(Tree t)
{
        int d, s;

        if (t ==  NULL)
                return 0;

        d = depth(t->right);
        s = depth(t->left);

        if(d > s)
                return d + 1;

        return s + 1;
}
```

Esercizio (tde 9-6-2009)

Si consideri la seguente definizione di un albero:

```
typedef struct EL { int dato;
                    struct EL * left, * right; } node;

typedef node * tree;
```

Si definisce livello di un nodo la sua distanza dalla radice. Si scriva una funzione che prende in ingresso un albero binario e restituisce 1 se tutti i nodi di livello pari contengono un numero pari e tutti i nodi di livello dispari contengono un numero dispari.

```
int f(tree t){
     if(t==NULL)
          return 1;
     return f2(t,0);// o 1
}

int f2(tree t,int livello){
     if(t==NULL)
          return 1;
     if(livello%2!=t->dato%2)
          return 0;
     return f2(t->left,livello+1) && f2(t->right,livello+1);
}
```

Esercizio (tde 13-11-2009)

Implementare una funzione C per il lancio di dadi.

La funzione prenda in ingresso il numero di facce del dado, e il numero di lanci che si vuole effettuare. La funzione simula i lanci del dado e stampa a video quante volte è uscita ciascuna faccia del dado. Si faccia inoltre in modo che la stampa risulti ordinata per numero di volte che è uscita una faccia in senso crescente.

Ad esempio, dopo 5 lanci con un dado a 6 facce in cui sono usciti i numeri (1, 1, 5, 4, 6), si stampi a video:

- La faccia 4 è uscita 1 volta
- La faccia 5 è uscita 1 volta
- La faccia 6 è uscita 1 volte
- La faccia 1 è uscita 2 volte

Attenzione, devono essere riportati sia il numero della faccia che il numero di volte che è uscita.

In C esiste una funzione

```
int rand();
```

che restituisce un numero intero casuale compreso tra 1 e RAND_MAX

```
#define N 1000
void lanci(int, int);
int randomRange(int, int);

int main()
{
int f = 6;

lanci(f, 1000);
return 1;
}

int randomRange(int lower, int upper)
{
    int n;
    n = rand(); // numero tra 1 e RAND_MAX
    n = (n % (upper - lower + 1)); // numero tra 0 e upper-lower
    n = n + lower; // numero tra lower e upper (estremi inclusi)
    return n;
}
```

```
void lanci(int f, int n) {
    int v[N]={0}, imin, i, min = 0;

    // genero il vettore di n lanci
    for(i=0;i<n;i++)
        v[randomRange(1,f)]++;

    // stampo il dato con meno occorrenze
    min=0;
    while(min<n+1)
    {
    min=n+1; // lo impongo oltre il max, in modo da non dover
    for(i=1;i<=f;i++) // passo le posizioni da 1 a f (quelle che ho riempito)
        if(min>v[i] && v[i]!=-1)
            {
                min=v[i];
                imin=i;
            }

    if(min<n+1)
       printf("La faccia %d uscita %d volte\n",imin,min);
    v[imin]=-1; // questi sono i valori che ho già stampato
    }
```

```
void lanci(int f, int n) {
    int v[N]={0}, imin, i, min = 0;

    // genero il vettore di n lanci
    for(i=0;i<n;i++)
        v[randomRange(1,f)]++;

    // stampo il dato con meno occorrenze
    min=0;
    while(min<n+1)
    {
    min=n+1; // lo impongo oltre il max, in modo da non dover
    for(i=1;i<=f;i++) // passo le posizioni da 1 a f (quelle che ho riempito)
        if(min>v[i] && v[i]!=-1)
            {
                min=v[i];
                imin=i;
            }
    if(min<n+1)
       printf("La faccia %d uscita %d volte\n",imin,min);
    v[imin]=-1; // questi sono i valori che ho già stampato
    }
}
```

bublesort

Soluzione altenativa

```
#define N 1000
typedef struct { int valore;  int quanti; } faccia;

void lancia(int lanci,int facce) {
        int i,j; faccia dado[N], temp; // dado è di dimensione N ma utilizzo solo [0 – faccia -1]
        for(i=0;i<facce;i++) {
                dado[i].faccia=i+1; dado[i].quanti=0;
        }
        for(i=0;i<lanci;i++) {
                val=rand(1,facce);
                dado[val-1].quanti++;
        }
        for(i=0;i<facce;i++) {
                for(j=0;j<facce;j++) {
                        if(dado[i].quanti>dado[i+1].quanti) {
                                temp=dado[i];
                                dado[i]=dado[i+1];
                                dado[i+1]=temp;
                        }
                }
        }
        for(i=0;i<facce;i++) {
                if(dado[i].quanti==1)
                        printf("La faccia %d è uscita 1 volta", dado[i].valore);
                if(dado[i].quanti>1)
                        printf("La faccia %d è uscita %d volte" , dado[i].valore, dado[i].quanti);
        }
}
```

Esercizio (tde 24-2-2011)

Si dice cosa stampa il seguente codice e si spieghi cosa calcola la funzione f

```
#include<stdio.h>

int f(int c,int d);

int main(){
    int c;
    c=f(2,3);
    printf ("c = %d\n", c);
    c=f(4,2);
    printf ("c = %d\n", c);
    c=f(2,4);
    printf ("c = %d\n", c);
    c=f(3,3);
    printf ("c = %d\n", c);
}
int f(int d,int e){
    if( e > 0 )
        return f(d,e-1) + f(d,e-1);
    else
        return d;
}
```

Esercizio (tde 12-11-2010)

Si scriva una funzione che riceve in input due array di dimensione N (con N costante predefinita).

```
void f(int a[], int b[])
```

Si definisce *equibilatero* un elemento di un vettore preceduto da tanti numeri pari più grandi quanti sono gli elementi dispari più piccoli che lo seguono. La funzione f deve copiare tutti gli elementi *equibilateri* di a in b in posizioni contigue partendo dalla prima posizione di b senza lasciare buchi. Le posizioni finali di b che restano libere devono essere riempite di zeri.

```c
#include<stdio.h>
#define N 5

void equibilatero(int*, int*);
int pariBefore(int*, int);
int dispariAfter(int*, int);

int main()
{
    int v[N] = {12, 14, 7, 5, 3}, u[N], i;
    equibilatero(v, u);
    printf("\n");
    for(i = 0; i <N; i++)
        printf("%d, ", v[i]);

    printf("\n");
    for(i = 0; i <N; i++)
        printf("%d, ", u[i]);

    return 1;
}
```

```
void equibilatero(int a[], int b[]){
    int i,j=0;
    for(i = 0; i < N ; i++)
        if(pariBefore(a,i) == dispariAfter(a,i)){
            b[j]=a[i];
            j++;
        }
    for(; j < N; j++)// non devo inizializzare, j è già il primo libero
        b[j]=0;
}
int pariBefore(int a[],int pos){
    int i,cont=0;
    for(i=0;i<pos;i++)
        if(a[i]%2==0 && a[i]> a[pos])
            cont++;
    return cont;
}
int dispariAfter(int a[],int pos){
    int i,cont=0;
    for(i=N;i>pos;i--)
        if(a[i]%2==1 && a[i]<a[pos])
            cont++;
    return cont;
}
```

TDE 1/3/2017

Si costruisca la tabella di verità della seguente espressione booleana, badando alla precedenza tra gli operatori logici. (1 punto).

not ((not (A and not B)) or B or (C and A))

TDE 1/3/2017

Si stabilisca il minimo numero di bit sufficiente a rappresentare in complemento a due i numeri $A = 111_{dec}$ e $B = -81_{dec}$, li si converta, se ne calcolino la somma (A+B) e la differenza (A–B) in complemento a due e si indichi se si genera riporto sulla colonna dei bit più significativi e se si verifica overflow (1 punto).

Esercizio

Si consideri la seguente definizione di un albero

```
typedef struct EL { int dato;
                    struct EL * left, *right; } node;
typedef node * tree;
```

Scrivere una funzione che riceve il puntatore alla radice di un albero e lo scandisce interamente costruendo una lista tale che abbia tanti nodi quanti sono i nodi dell'albero e che ogni nodo della lista punti ad una diverso nodo dell'albero. La funzione deve restituire al chiamante il puntatore all'inizio della lista creata.

```
typedef struct ELLista { tree foglia;
                         struct ELLista * next; } nodeLista;
typedef nodeLista * Lista;
```

Si noti che esistono diversi modi di visitare l'albero che originano diverse liste come risultato.

```
Lista demiurgo ( tree t ) {
   return demiurgino(t,NULL)
}

Lista demiurghino ( tree t, Lista L ) {
        if(t==NULL)
            return NULL;

        L=insInCoda(L,t);

        if(t->left==NULL && t->right==NULL)
            return L;

    if(t->left==NULL)
            return demiurghino(t->right,L);

    if(t->right==NULL)
            return demiurghino(t->left,L);

        L=demiurghino(t->left,L);
        L=demiurghino(t->right,L);

        return L;
}
```

Esercizio

Si consideri la seguente definizione di un albero binario (binario=con due rami in ogni nodo):

```
typedef struct EL { int dato;
                    struct EL * left, right; } node;
typedef node * tree;
```

Scrivere una funzione che riceve il puntatore alla radice di un albero e lo scandisce interamente costruendo una lista tale che abbia tanti nodi quante sono le foglie dell'albero e che ogni nodo della lista punti ad una diversa foglia dell'albero. La funzione deve restituire al chiamante il puntatore all'inizio della lista creata.

```
typedef struct ELLista { tree foglia;
                         struct ELLista * next; } nodeLista;
typedef nodeLista * Lista;
```

Si noti che esistono diversi modi di visitare l'albero che originano diverse liste

```
Lista creaLista(tree t, Lista lis) {
        if (t==NULL)
                return NULL;
   if(t->left==NULL && t->right==NULL)
                return inserisciInCoda(t,lis);
   if(t->left==NULL)
                return creaLista(t->right,lis);
   if(t->right==NULL)
                return creaLista(t->left,lis);
   return creaLista(t->left, creaLista(t->right,lis));
```

Esercizio

Si consideri la seguente definizione di un albero binario

```
typedef struct EL { int dato;
                    struct EL * left, right; } node;
typedef node * tree;
```

in cui dato assume sempre valori positivi.

Supponiamo che percorrendo un cammino dalla radice alle foglie si totalizzi un punteggio pari alla somma dei valori contenuti nei nodi percorsi.

Scrivere una funzione maxPunti che calcola il punteggio massimo che possiamo totalizzare percorrendo un cammino dalla radice alle foglie.

Vogliamo percorrere l'albero dalla radice ad una foglia totalizzando esattamente un certo punteggio: né far meno, né sforare. Scrivere una funzione esisteCammino che dati un albero ed un intero k, dice se esiste un cammino dalla radice ad una foglia che permette di totalizzare esattamente k punti.

Si consideri la seguente definizione di un albero binario

```
typedef struct EL { int dato;
                    struct EL * left, right; } node;
    typedef node * tree;
```

in cui dato assume sempre valori positivi.

Supponiamo che percorrendo un cammino dalla radice alle foglie si totalizzi un punteggio pari alla somma dei valori contenuti nei nodi percorsi.

Scrivere una funzione maxPunti che calcola il punteggio massimo che possiamo totalizzare percorrendo un cammino dalla radice alle foglie.

Vogliamo percorrere l'albero dalla radice ad una foglia totalizzando esattamente un certo punteggio: né far meno, né sforare. Scrivere una funzione esisteCammino che dati un albero ed un intero k, dice se esiste un cammino dalla radice ad una foglia che permette di totalizzare esattamente k punti.

```
int maxPunti ( tree t ) {
  int D, S;
  if (t == NULL)
    return 0;
  S = maxPunti( t->left );
  D = maxPunti( t->right );
  if ( S > D )
      return S + t->dato;
  else
      return D + t->dato;
}
```

```
int esisteCammino ( tree t, int k ) {
if (t == NULL && k==0)
      return 1;
  if (t == NULL)
      return 0;
   if (k – t->dato <0)
      return 0;
 if( t->left==NULL)
                return esisteCammino(t->right, k - t->dato);
if( t->right==NULL)
                return esisteCammino(t->left, k - t->dato);
return (esisteCammino(t->left, k - t->dato)  ||
              esisteCammino(t->right, k - t->dato));
}
```

Si consideri la seguente definizione di un albero binario:

```
typedef struct Elemento { char parola[30];
                          int occorrenze;
                          struct Elemento * left, * right;  } Nodo;

typedef Nodo * Tree;
```

La seguente funzione inserisce nell'albero t tutte le parole contenute nella lista l. L'indice deve contenere una sola volta le parole del testo, ordinate alfabeticamente secondo il criterio per cui in ogni nodo n dell'albero tutte le parole del sottoalbero destro precedono la parola di n, mentre quelle del sottoalbero sinistro la seguono.

```
Tree creaIndice( ListaParole l ) {
  Tree t = NULL;
  while( l != NULL ) {
    t = inserisciOrd( t, l->word );
    list = list->next;
   }
   return t;
}
```

Si codifichi in C la funzione inserisciOrd, badando ad allocare i nodi per le parole non presenti nell'indice e aumentare il contatore delle occorrenze per le parole già presenti.

```
Tree inserisciOrd( Tree t, char * p ) {
  if( t == NULL ) {
    t = (Tree) malloc(sizeof(Nodo));
    t->left = NULL; t->right = NULL;
    t->occorrenze = 1;
    strcpy(t->parola, p);
  }
  else if ( strcmp(p, t->parola)==0 )
    t->occorrenze += 1;
  else if ( strcmp(p, t->parola) < 0 )
    t->left = inserisciOrd( t->left, p );
  else
    t->right = inserisciOrd( t->right, p );
  return t;
}
```

Si consideri la seguente definizione di albero binario:

```
typedef struct t { char parola[1000];
                   struct t * left, * right; } Nodo;
typedef Nodo * Tree;
```

Due parole si dicono simili se hanno al più due caratteri diversi (cioè se hanno gli stessi caratteri nello stesso ordine, eccetto due di essi che possono essere diversi o mancare nella seconda o essere in più nella seconda parola).

Una catena di parole si dice compatibile col telefono senza fili (cctsf) se ogni parola è simile alle adiacenti.

La funzione int simili (char *s1, char *s2); restituisce 1 se s1 e s2 sono simili, o altrimenti.

Usando la funzione simili(…) (senza codificarla), si codifichi in C una funzione f che riceve come parametro un albero di parole (secondo la definizione soprastante) e restituisce 1 se tutti i cammini dalla radice alle foglie rappresentano catene cctsf, o altrimenti.

Esercizio (tde 14-11-2008)

Definiamo sequenza monotona crescente in un vettore una sequenza di elementi contigui in cui ogni elemento in posizione i+1 è più grande di quello in posizione i.

4 5 7 è una sequenza monotona crescente di lunghezza 3

6 8 è una sequenza monotona crescente di lunghezza 2

Definiamo sequenza monotona decrescente in un vettore una sequenza di elementi contigui in cui ogni elemento in posizione i+1 è più piccolo di quello in posizione i.

7 3 1 è una sequenza monotona decrescente di lunghezza 3

4 2 è una sequenza monotona decrescente di lunghezza 2

Un array di interi si dice uniformemente oscillante se tutte le sequenze monotone (crescenti o decrescenti) massime (cioè non contenute in altre sequenze monotone) che contiene hanno la stessa lunghezza. Codificare una funzione che riceve in ingresso un vettore V e la sua dimensione N e restituisce 1 se il vettore è uniformemente oscillante, 0 altrimenti.

Esempi: 4 5 7 3 1 5 9 4 3 è uniformemente oscillante (tutte le sequenze cres. o decr. sono lunghe 3)

0 1 0 -1 0 1 0 -1 non è uniformemente oscillante (la prima sequenza 0 1 è più corta di altre)

```
int uniformementeOscillante(int v[], int N){
    int i,asc,cont=1,oldCont,prima=1;
    if(v[0]<v[1])
        asc=1;
    else
        asc=0;
    for(i=1;i<N-1;i++) {
        if( (asc==1 && v[i]<v[i+1]) || (asc==0 && v[i]>v[i+1])){
            cont++;
        } else if( (asc==1 && v[i]>v[i+1]) || (asc==0 && v[i]<v[i+1])) {
            if (prima==1) {
                prima=0;
            } else {
                 if(oldCont!=cont)
                     return 0;
            }
            asc=!asc; oldCont=cont;  cont=1;
        }
    }
    if(oldCont!=cont)
          return 0;
    return 1;
}
```