Esercizi

Credits Prof. Campi

Esercizio (tde 1-2-2008)

- Dato il tipo

  typedef struct { int V[20];

  int C; } STR;

- Scrivere una funzione con un parametro formale X di tipo STR e un parametro formale MATR di tipo matrice 20x20 interi. La funzione considera i primi C elementi contenuti nel campo V di X e restituisce la somma di quelli tra loro che sono multipli di almeno tre elementi della matrice MATR. Si supponga che il valore del campo C sia sempre compreso tra 0 e 19 (estremi inclusi). Si utilizzino opportune funzioni ausiliarie per dividere il problema in sottoproblemi più semplici.

```
int contaSMseMag3(int x, MATR m){
        int i,j,cont=0;
        for(i=0;i<20;i++)
                for(j=0;j<20;j++)
                        if(m(i)(j)!=0 && x%m(i)(j)==0)
                                cont++;
        if(cont>=3)
                return 1;
        else
                return 0;
}
int f(STR x, MATR m){
        int i,s=0;
        for(i=0;i<x.C;i++)
                if(contaSMseMag3(x.v(i), m)==1)
                        s+= x.v(i);
        return s;
}
```

```
int f(float m[][M]) {
  int i,j,k,t,cont,quanti=0;
  for(i=0; i < N;i++) {
            for(j=0; j < M;j++) {
                        cont=0;
                        for(k=i-1; k <= i+1;k++) {
                                    for(t=j-1; t <= j+1;t++) {
                                                if(k < 0 || k >= N || t < 0 || t >= M) {
                                                            cont++;
                                                }
                                                else if(!(k==i && t==j) && m[k][t]*2<m[i][j])
                                                            cont++;
                                    }
                        }
                        if(cont==8)
                                    quanti++;
            }
    }
    return quanti;
}
```

Esercizio (variante tde 25-2-2008)

- Si implementi una funzione che riceve in input una matrice NxM di float. Definito “picco” un numero circondato in tutte le posizioni intorno solo da numeri strettamente inferiori alla sua metà, la funzione conta il numero di “picchi” della matrice (attenzione a gestire correttamente gli elementi ai bordi della matrice).

Esercizio (tde 10-9-2008)

- Si implementi una funzione che riceve in input una matrice NxM di float. Definito “pozzo” un numero circondato in tutte le otto posizioni intorno solo da numeri strettamente superiori al suo doppio, la funzione conta il numero di “pozzi” della matrice (attenzione a gestire correttamente gli elementi ai bordi della matrice, anche per loro bisogna controllare se sono pozzi).

```
int f(float m[][M]) {
  int i,j,k,t,cont,quanti=0;
  for(i=0; i < N;i++) {
            for(j=0; j < M;j++) {
                        cont=0;
                        for(k=i-1; k <= i+1;k++) {
                                    for(t=j-1; t <= j+1;t++) {
                                                if(k < 0 || k >= N || t < 0 || t >= M) {
                                                            cont++;
                                                }
                                                else if(!(k==i && t==j) && m[k][t]>2*m[i][j])
                                                            cont++;
                                    }
                        }
                        if(cont==8)
                                    quanti++;
            }
    }
    return quanti;
}
```

Esercizio (tde 13-11-2009)

- Definito il tipo punto

  typedef struct {int x;

  int y;

  int NMonete;

  } punto;

  scrivere una funzione che riceve in input una matrice M di punti di dimensione N*N e due coordinate x, y.

- La funzione deve navigare la matrice, partendo dal punto indicato dalle coordinate x e y e spostarsi ogni volta nelle coordinate indicate dal punto raggiunto. Nello spostarsi la funzione deve sommare tutti i valori di NMonete incontrati. La funzione procede allo stesso modo finché o non torna su un punto già toccato o trova coordinate non valide. A quel punto restituisce la somma delle monete raccolte.

```
int caccia(punto M[][N], int x, int y) {
        int i,j,totale=0, M2[N][N], aux;

        for(i=0;i<N;i++)
                for(j=0;j<N;j++)
                        M2[i][j]=0;

        while(x>=0 && x<N && y>=0 && y<N && M2[x][y]==0){
                        M2[x][y]=1;
                        totale=totale+M[x][y].NMonete;
                        aux=M[x][y].x;
                        y=M[x][y].y;
                        x=aux;
        }

        return totale;
}
```

Esercizio (tde 14-11-2008)

- Si completi il programma sottostante, dove puzzle è una matrice di caratteri di R righe e C colonne che rappresenta un cruciverba. Le caselle nere sono rappresentate dal carattere '*'
- Si vogliono numerare le parole di almeno due lettere secondo la consueta numerazione dei cruciverba, partendo dalle celle in alto a sinistra e procedendo in ordine lessicografico
- Ad esempio, la matrice dichiarata nel programma corrisponde allo schema mostrato in figura (il numero di ciascuna parola è indicato nella cella contenente la prima lettera della parola)

[1]A	[2]B	I	■	[3]I
[4]T	O	■	[5]E	R
[6]O	B	A	M	A
S	■	■	[7]I	Q

Il programma deve determinare la numerazione delle parole e stampare, per ogni numero, le coordinate (riga e colonna) della casella a cui si riferisce e se si tratta di una parola orizzontale, verticale o sia orizzontale che verticale.
Con la matrice dichiarata nel programma, l'output è come segue.
Parola 1: riga 0, colonna 0 verticale orizzontale
Parola 2: riga 0, colonna 1 verticale
Parola 3: riga 0, colonna 4 verticale
Parola 4: riga 1, colonna 0 orizzontale
Parola 5: riga 1, colonna 3 verticale orizzontale
Parola 6: riga 2, colonna 0 orizzontale
Parola 7: riga 3, colonna 3 orizzontale
Per l'implementazione si realizzi e si faccia uso di una funzione con il seguente prototipo
int iniziaParola(int m[][C], int r, int c, int d);
che controlli se alla riga r e colonna c inizi una parola di almeno due lettere nella matrice m lungo la direzione d (orizzontale se d è 0, verticale altrimenti).
Si noti che una parola inizia in una certa casella se è preceduta, lungo la direzione considerata, o da una casella nera o dal bordo del puzzle.

```
#define R 4
#define C 5

#define Orizzontale 0
#define Verticale 1

#include <stdio.h>
int iniziaParola(char m[][C], int r, int c, int d);
int main() {
        char m[R][C] = {'A','B','I','*','I', 'T','O','*','E','R','O','B','A','M','A','S','*','*','I','Q'};
        int r, c, n=0,parolaOriz, parolaVert;
        for (r=0;r<R;r++) {
             for (c=0;c<C;c++) {
                        … …
                        if (parolaVert || parolaOriz) {
                                n++;
                                printf("Parola %d: riga %d, colonna %d", n, r, c);
                                if(parolaVert)
                                        printf(" verticale");
                                 if(parolaOriz)
                                        printf(" orizzontale");
                        }
             }
    }
}
```

```
int iniziaParola(char m[][C], int r, int c, int d) {
        … …
}
```

```
#define R 4
#define C 5

#define Orizzontale 0
#define Verticale 1

#include <stdio.h>
int iniziaParola(char m[][C], int r, int c, Direzione d);
int main() {
        char m[R][C] = {'A','B','I','*','I', 'T','O','*','E','R','O','B','A','M','A','S','*','*','I','Q'};
        int r, c, n=0,parolaOriz, parolaVert;
        for (r=0;r<R;r++) {
                for (c=0;c<C;c++) {
                        parolaOriz=iniziaParola(m, r, c, Orizzontale);
                        parolaVert=iniziaParola(m, r, c, Verticale);
                        if (parolaVert || parolaOriz) {
                                n++;
                                printf("Parola %d: riga %d, colonna %d", n, r, c);
                                if(parolaVert)
                                        printf(" verticale");
                                 if(parolaOriz)
                                        printf(" orizzontale");
                        }
                }
    }
}
```

```
int iniziaParola(char m[][C], int r, int c, int d) {
        if (m[r][c] == '*') return 0;
        switch(d) {
                case Orizzontale:
                        if (c-1<0 || m[r][c-1]=='*')
                                if(c+1<C && m[r][c+1]!='*')
                                        return 1;
                        return 0;
                default: // verticale
                        if (r-1<0 || m[r-1][c]=='*') {
                                if(r+1<R && m[r+1][c]!='*')
                                        return 1;
                        }
                        return 0;
        }
}
```

```
int iniziaParola(char m[][C], int r, int c, int d) {
        if(m[r][c]=='*')
                return 0;

        if(d==Verticale)
                if((r-1<0 || m[r-1][c]=='*') && (r+1<N &&
                  ( m[r+1][c]>='A' && m[r+1][c]<='Z' ||
                          m[r+1][c]>='a' && m[r+1][c]<='z') ) )
                  return 1;
                else
                  return 0;
   else
        if((c-1<0 || m[r][c-1]=='*') && (c+1<N &&
                  ( m[r][c+1]>='A' && m[r][c+1]<='Z' ||
                          m[r][c+1]>='a' && m[r][c+1]<='z') ) )
                  return 1;
                else
                  return 0;
}
```

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 crescenti o decrescenti sono lunghe 3)

  0 1 0 -1 0 1 0 -1 non è *uniformemente oscillante* (la prima sequenza 0 1 è più corta delle 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;
}
```

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.
- Si supponga esista e dunque si faccia uso della funzione

  int rand(int inf, int sup);

  che restituisce un numero intero casuale compreso tra inf e sup.

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

void lancia(int lanci,int facce) {
        int i,j; faccia dado[N], temp;
        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);
        }
}
```

```
#define N 1000
void lanci(int f, int n) {
   int v[N]={0};
   for(i=0;i<n;i++)
      v[rand(1,f)]++;

               min=0;
   while(min<n+1)
      min=n+1;
      for(i=1;i<=f;i++)
         if(min>v[i] && v[i]!=0)
            min=v[i]; imin=i;

      if(min<n+1)
         printf("La faccia %d uscita %d volte",imin,min);

      v[imin]=0;
}
```

Esercizio (tdeB 19/11/2008)

- Definiamo *doppie* le parole in cui la prima metà è identica alla seconda (esempi: CECE, CANCAN, COUSCOUS) e *assonanti* le parole che sarebbero doppie se non avessero l'ultima lettera (esempi: AMAMI, MARMARA, PORPORA)
- Si definisca una funzione **… doppia(…)** che riceve come parametro una stringa (che supponiamo valida e ben formata) e restituisce **1** se la stringa rappresenta una parola doppia, **0** altrimenti
- Si definisca una funzione **… assonante(…)** che riceve una stringa e restituisce **1** se la stringa rappresenta una parola assonante, **0** altrimenti. **Si apprezzano, in particolare, soluzioni che riusano efficacemente la funzione definita al punto precedente**

```
int doppia(char str[]) {
        int lung,i;

        lung=strlen(str);

        if(lung%2==1)
            return 0;

        for(i=0;i<lung/2;i++)
            if(str[i]!=str[i+lung/2])
                return 0;

        return 1;
}

int assonante(char str[]) {
        int lung,i;

        lung=strlen(str);

        if(lung%2==0)
            return 0;

        for(i=0;i<lung/2;i++)
            if(str[i]!=str[i+lung/2])
```

Esercizio (tde 9-2-2010)

- Indicare cosa stampa a video il seguente programma. Si ricorda che il C utilizza l'alfabeto inglese in cui le lettere sono 26 (ci sono in più j, k, w, x, y).
- Si assuma che la dimensione di un int sia 4 byte e che la dimensione di un char sia 1 byte.

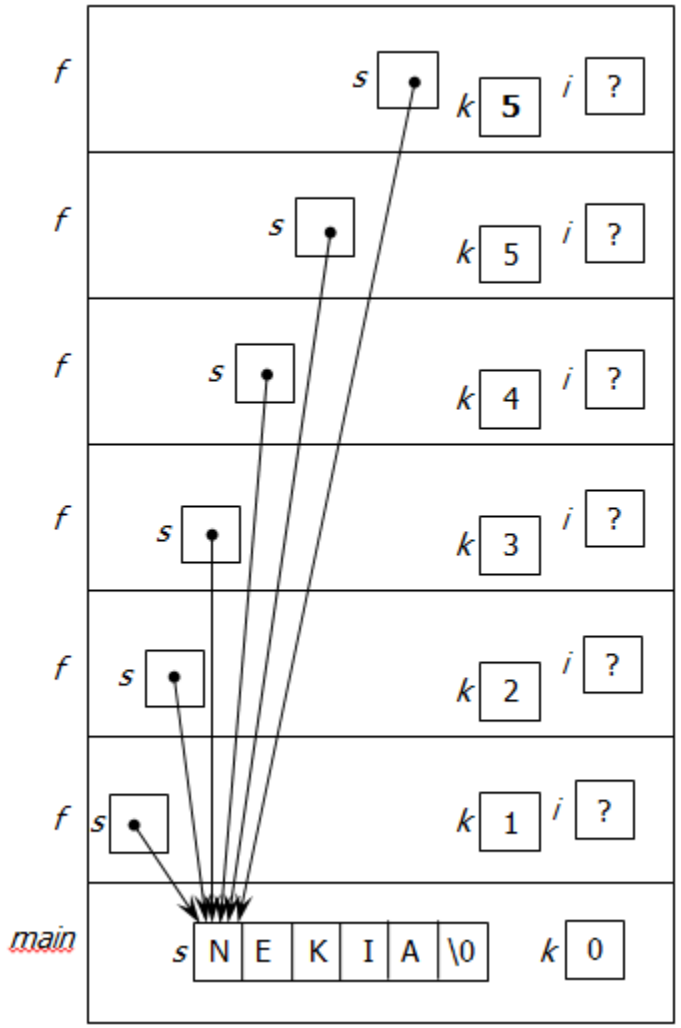
```c
int main () {
	char c, arr['z'-'a'], *pChar;
	int i, *pInt, int matr = __ ; // inserire qui le ultime due cifre della propria matricola
	//Inizio prima parte
	for ( c='a'; c<'z'; c++) {
		arr[c-'a'] = c;
		printf("%c ", arr[c-'a']);
	}
	printf("\n");
	//Inizio seconda parte
	pChar = arr + matr%15;
	for (i=0; i< 5; i++) {
		printf("%c ", *pChar);
		pChar++;
	}
	printf("\n");
	//Inizio terza parte
	pInt = (int *)arr + matr%2;
	for (i=0; i<5; i++) {
		pChar = (char *)pInt;
		printf("%c ", *pChar);
		pInt++;
	}
	return 0;
}
```

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

```
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;
}


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



Stampa: BINGO !

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 26-1-2009)

- Si definisca una funzione che riceve in input un array a di 10 interi e una matrice m di 32x32 interi. L'array contiene una serie di interi tutti a 1 o a 0.
- La funzione calcola il numero decimale corrispondente alla codifica binaria rappresentata dalla sequenza di 1 e 0 dei primi 5 elementi dell'array e pone questo numero in una variabile intera x. Quindi calcola il numero decimale corrispondente alla codifica binaria rappresentata dalla sequenza di 1 e 0 dei rimanenti 5 elementi dell'array e pone questo numero in una variabile y.
- Infine la funzione considera gli elementi della riga con indice x in m e controlla se tra questi quelli maggiori di m[x][y] siano di più di quelli minori di m[x][y]. Se questo è il caso, il programma restituisce 1, altrimenti 0.

Esercizio (tde 8-7-2010)

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

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

#define L 5

void f(char val[],char comp[]);

int main(){
   char val[L+1], comp[L+1];
   strcpy(val,"11111");
   f(val,comp);
   printf("\n%s > %s\n", val, comp);
   strcpy(val,"00000");
   f(val,comp);
   printf("\n%s > %s\n", val, comp);
   strcpy(val,"10101");
   f(val,comp);
   printf("\n%s > %s\n", val, comp);
   strcpy(val,"01010");
   f(val,comp);
   printf("\n%s > %s\n", val, comp);
   system("PAUSE");
}
```

```c
void f(char val[],char comp[]){
   int i=0;
   int notOK=0, first1=0;
   do {
      for(i = 0; val[i] != '\0' && !notOK; i++)
         if(val[i] != '0' && val[i] != '1')
            notOK = 1;
   } while(notOK);
   comp[i] = val[i];
   for(i--; i >= 0; i--)
      if(first1)
         if(val[i] == '0')
            comp[i] = '1';
         else
            comp[i] = '0';
      else {
         comp[i] = val[i];
         if(val[i] == '1')
            first1 = 1;
      }
}
```

Esercizio (tde 26-1-2009)

- La matrice M[6][156] contiene le 156 estrazioni del 2010 dei 6 numeri del SuperEnalotto.
- Scrivere una funzione

  int f(int M[][156])

  che riceve la matrice M e restituisce quanti numeri tra 1 e 90 non sono mai usciti nel 2010.

```
int f(int M[][156]) {
        int lung,i,cont,k,t,nonTrovati=0;

        for(i=1;i<=90;i++){
            cont=0;
            for(k=0;k<6 && cont==0;k++)
                for(t=0;t<156 && cont==0;t++)
                    if(M[k][t]==i)
                        cont++;
            if(cont==0)
                nonTrovati++;
    }

        return nonTrovati;
}
```

Esercizio (tde 26-1-2009)

- Si progetti e codifichi una funzione C che riceve in ingresso un array a di interi non negativi.
- La funzione calcola il numero degli elementi dell'array che godono della seguente proprietà: il valore dell'elemento è pari al numero di elementi con valore inferiore considerando solo quelli in posizione precedente.
- Ad esempio, si consideri il seguente vettore:
- 1 2 1 **3** 6 **4** 14
- In questo caso solo i due numeri in grassetto (il 3 e il 4) soddisfano la proprietà descritta. Quindi la funzione restituisce 2.

```
int contaPrec(int a[],int p) {
        int i=0,cont=0;
        for(i=0;i<p;i++) {
                if(a[i]<a[p])
                        cont++;
        }
        return cont;
}

int f(int a[]) {
        int i,cont=0;
        for(i=0;i<N;i++)
                if(contaPrec(a,i)==a[i])
                        cont++;
        return cont;
}
```

Esercizio (tde 7-9-2009)

- Si consideri il seguente programma C, completando la definizione di **MATR** con la propria matricola.

  #define MATR "....."

- Si spieghi brevemente il comportamento del programma; per comprenderlo, si suggerisce di simularne l'esecuzione (prestando attenzione a "impilare" ed eventualmente "disimpilare" bene le chiamate ricorsive e a prestare attenzione a quali istruzioni sono eseguite e quali no).
- Indicare anche l'output stampato a video dal programma (attenzione alla posizione delle chiamate a printf).

```
void foo(char * parola) {   // 19
  if (parola == NULL)
    return;
  else {
    printf("%c", *parola);
    parola=parola+1;
    if(strlen(parola) == 0)
      return;
    else {
      foo(parola-1);
      printf("%c", *parola);
      return;
    }
  }
}

int main() {
  char m[7] = “623619”;
  foo(m+4);
  printf("SE ARRIVO QUI HO FINITO\n");
  return 0;
}
```

Esercizio (tde 9-2-2010)

- Si definisca una funzione che riceve in input due matrici **m1** e **m2** di NxN interi. La funzione calcola e restituisce il numero di elementi di **m2** che sono pari alla somma di due qualsiasi elementi di **m1** (se un dato elemento di **m2** è somma di più coppie di elementi di **m1**, viene contato comunque una volta sola)
- La funzione abbia la seguente intestazione: int numSomme (int m1[][N], int m2[][N])

Esercizio (tde 9-2-2010)

- Si progetti e codifichi una funzione C

  int numSommaDiff (int a[])

  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.

```
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;
}
```

Esercizio (tde 10-9-2010)

- Si consideri il seguente programma C, completando la definizione di **MATR** con la propria matricola.

  #define MATR "....."

  .....

- Dire cosa stampa il programma.

```
int f( char * pcA, char * pcB, int * num ) {
  int valore=0;
  *num = *num + 1;
  if ( pcA >= pcB )
     return *num;
  else
     valore=f( pcA+1, pcB-1, num );
  if ( *pcA >= *pcB ) {
     pcA = pcB;
     *num = *num + 10;
  }
  printf("%c", *pcA);
  return *num;
}
int main() {
  char matricola[7] = MATR;
  int cont = 0, val=0;
  printf("%s - ", matricola);
  val=f( matricola, matricola+5, &cont );
  printf(" - %d - %d", cont, val);
  return 0;
}
```

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.

```
void f(int a[], int b[]){
        int i,j=0;
        for(i=0;i<N;i++)
                if(f1(a,i)==f2(a,1)){
                        b[j]=a[i];
                        j++;
                }
        for(;j<N;j++)
                b[j]=0;
}
int f1(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 f2(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;
}
```

Esercizio (tde 12-11-2010)

- Si considerino le seguenti definizioni di tipo:

  ```
  #define N 20
  typedef int matrice[N][N];
  typedef matrice matmat[N][N];
  ```

- Implementare una funzione C che riceve in input una matrice di matrici (una matmat) e un intero k e restituisce 1 se tra le matrici contenute nella matmat ricevuta ce ne sono almeno k che contengono il numero k almeno k volte. Si consiglia di spezzare la soluzione del problema in più funzioni.

```
int f(matmat m,int k){
        int i,j,cont=0;
        for(i=0;i<N;i++)
                for(j=0;j<N;j++)
                        if(f1(m[i][j],k))
                                cont++;
        if(cont>=k)
                return 1;
        else
                return 0;
}
int f1(matrice m;int k){
        int i,j,cont=0;
        for(i=0;i<N;i++)
                for(j=0;j<N;j++)
                        if(m[i][j]==k)
                                cont++;
        if(cont>=k)
                return 1;
        else
                return 0;
}
```

Esercizio (continua precedente)

- Implementare una funzione C che riceve in input una matrice di matrici (una matmat) e restituisce il massimo numero k per cui è vero che tra le matrici contenute nella matmat ricevuta ce ne sono almeno k che contengono il numero k almeno k volte. Si consiglia di spezzare la soluzione del problema in più funzioni

```
int f2(matmat m) {
        int i;
        for(i=N*N;i>0;i--)
                if(f(m,i))
                        return i;
        return 0;
}
```

Esercizio (tde 8-2-2011)

- Un archivio musicale è organizzato nel seguente modo: gli artisti sono disposti in un array e sono ordinati per nome; ogni artista ha associata un array di album organizzati per anno e a parità di anno per ordine alfabetico; ogni album contiene un array di canzoni ordinate per posizione all'interno dell'album

Le strutture dati utilizzate sono le seguenti:

```
typedef struct Song { char * titolo;
                      int durata;
                      int pos;
                      int valido; } Canzone;
typedef Canzone SequenzaCanzoni[1000];

typedef struct Album { char * titolo;
                       int n_canzoni;
                       int anno;
                       SequenzaCanzoni canzoni;
                       int valido; } Disco;
typedef Disco SequenzaDischi[1000];

typedef struct Singer { char * nome;
                        int n_dischi;
                        SequenzaDischi dischi;
                        int valido; } Artista;
typedef Artista SequenzaArtisti[1000];
```

Il campo “valido” presente in tutte le struct serve a dire se la casella dell’array contiene contenuto valido (nel caso l’attributo ha valore 1) o è da considerarsi vuota (nel caso l’attributo ha valore 0).

Si codifichi in C la seguente funzione:

```
int invalidaCanzone (SequenzaArtisti artisti,
                     char * artista, char * disco, char * canzone)
```

che invalida dal catalogo la canzone qualora questa esista e restituisce 0 in caso di esito positivo, -1 in caso di canzone non esistente. Qualora la canzone sia l'unica del disco, occorre inoltre invalidare il disco. Infine, se il disco invalidato dovesse essere l'unico esistente dell'artista, è necessario invalidare l'artista stesso. Conseguentemente, vanno opportunamente aggiornati i contatori del numero di canzoni e di dischi.

```
int invalidaCanzone (SequenzaArtisti a,char* artista,char* disco,char* canzone){
    int i,j,k;
    for(i=0; strcmp(a[i].nome,artista)<=0 && i<1000; i++){
        if(strcmp(a[i].nome,artista)==0){
            for(j=0; strcmp(a[i].dischi[j].titolo,disco)<=0 && j<a[i].n_dischi; j++){
                if(strcmp(a[i].dischi[j].titolo,disco)==0){
                    k=0;
                    for(k=0; k<a[i].dischi[j].n_canzoni; k++){
                        if(strcmp(a[i].dischi[j].canzoni[k].titolo,canzone)==0){
                            a[i].dischi[j].canzoni[k].valido = 0;
                            a[i].dischi[j].n_canzoni--;
                            if(a[i].dischi[j].n_canzoni==0){
                                a[i].dischi[j].valido = 0;
                                a[i].n_dischi--;
                                if(a[i].n_dischi==0){  a[i].valido = 0;  }
                            }
                            return 0;
                        }
                    }
                }
            }
        }
    }
    return -1;
}
```

Esercizio (tde 8-2-2011)

- Si scriva una funzione che riceve in ingresso un vettore di N numeri interi (N è una costante prefissata). Esso contiene dei valori sempre positivi o nulli (mai negativi). Una posizione contenente il numero 0 è da considerare vuota, cioè non contenente alcun valore.
- La funzione deve calcolare e stampare a video la media dei valori e poi deve modificare i valori che distano più di del 10% del valore della media dalla media stessa mettendoli a 0. La procedura deve essere ripetuta fino a che non ci sono più valori che vengono modificati.

Esercizio (tde 8-2-2011)

- Descrivere il comportamento della funzione m.
- Determinare cosa verrebbe stampato invocando m(12) e m(14560)

```
#define M 16
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 (tde 24-2-2011)

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

```
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 8-7-2011)

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

```c
#define L 5
int f(char val[],char comp[]);
int main(){
    char val[L+1], comp[L+1];
    strcpy(val,"11111");
    f(val,comp);
    printf("\n%s > %s\n", val, comp);
    strcpy(val,"00000");
    f(val,comp);
    printf("\n%s > %s\n", val, comp);
    strcpy(val,"10101");
    f(val,comp);
    printf("\n%s > %s\n", val, comp);
    strcpy(val,"01010");
    f(val,comp);
    printf("\n%s > %s\n", val, comp);
    system("PAUSE");
}
```

```c
int f(char val[],char comp[]){
    int i=0;
    int notOK=0, first1=0;
    do {
        for(i = 0; val[i] != '\0' && !notOK; i++)
            if(val[i] != '0' && val[i] != '1')
                notOK = 1;
    } while(notOK);
    comp[i] = val[i];
    for(i--; i >= 0; i--)
        if(first1)
            if(val[i] == '0')
                comp[i] = '1';
            else
                comp[i] = '0';
        else {
            comp[i] = val[i];
            if(val[i] == '1')
                first1 = 1;
        }
}
```

Esercizio (tde 8-7-2011)

- La matrice M[6][156] contiene le 156 estrazioni del 2010 dei 6 numeri del SuperEnalotto.
- Scrivere una funzione

  int f(int M[][156])

  che riceve la matrice M e restituisce quanti numeri tra 1 e 90 non sono mai usciti nel 2010.

Esercizio (tde 12-9-2011)

- Si dica cosa stampa il seguente codice

```
int f( char *a, char *b, int x ) {
  if( !x )
    printf(" & ");
  else {
    printf("%c", b[x%2]);
    x = f( a, b, x-1 );
    printf("%c", b[x%2]);
  }
  return x+1;
}
```

```
int main() {
    int i = 5 ;

    char bs[] = "BS", na[]  = "NA";
    ++i;
    printf("%c", bs[i%2]);
    i = f( bs, na, i-1 );
    ++i;
    printf("%c", bs[i%2]);

    system("PAUSE");

    return 0;
}
```

Esercizio (tde 12-9-2011)

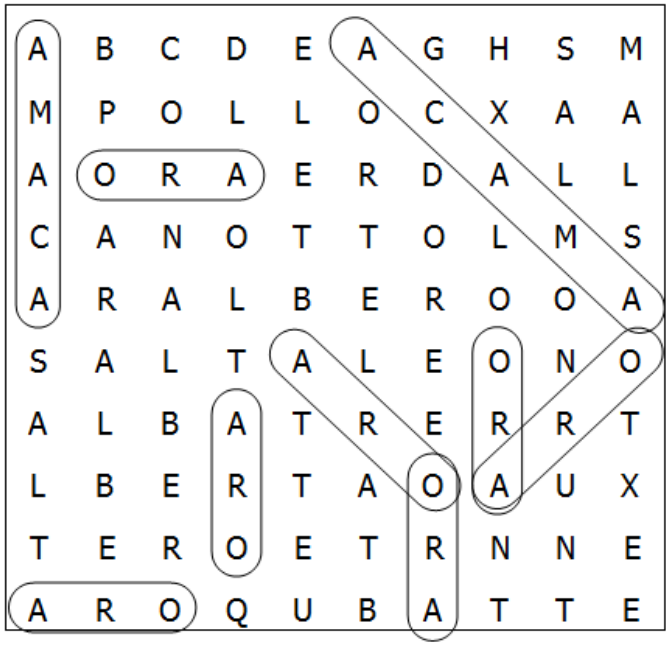
- La matrice di interi M[N][N] (con N costante predefinita) contiene degli interi generati a caso.
- Scrivere una funzione

  int f(int M[][N])

  che riceve la matrice M e restituisce quanti valori distinti contiene la matrice M

Esercizio (tdeB 24-11-2005)

Si completi opportunamente il programma sottostante, dove `puzzle` è una matrice di caratteri dichiarata come variabile globale, e quindi visibile a tutte le funzioni. Il programma visualizza un messaggio con tre numeri, che indicano quante volte la parola inserita si legge nel puzzle in verticale (↑ o ↓), in orizzontale (← o →) e in diagonale (↗ o ↘ o ↙ o ↖). Esempi e avvertimenti:




```
> DICIOTTO
> La parola non si trova

> AMACA
> H: 0   V: 1   D: 1

> ORA
> H: 2   V: 3   D: 2
```

Si presti attenzione
- ai bordi del puzzle (le parole non continuano ciclicamente)
- all'assenza del terminatore di stringa nel puzzle

Per scrivere una soluzione sintetica e coerente con i principi della buona programmazione, si **suggerisce** (e si apprezza) l'uso della funzione `controlla`, che verifica la presenza di una parola a partire da una casella del puzzle, in modo parametrico rispetto alla direzione in cui cercarla (tra le 8 direzioni possibili) – i parametri cioè indicano, rispetto alle direzioni verticale e orizzontale, se lo spostamento non avviene, avviene in senso positivo, o avviene in senso negativo. **Ad esempio**, con i valori **0** e **1** si può indicare la direzione →, con **-1** e **0** la direzione ↑, con **1** e **-1** la direzione ↙.

```
#define N 10
char puzzle[N][N] = { 'A', 'B', 'C',... 'T', 'E' };
/* Controlla se da puzzle[i][j] inizia la parola p, muovendosi in orizzontale,
    verticale o diagonale in base ai valori di sv e so (spost. vert. e orizz. */
int controlla (char p[], int i, int j, int sv, int so);
int main() {
 char parola[N+1];  // +1 per il terminatore
 int i, j;
 int hor = 0, ver = 0, dia = 0;  // numero di occorrenze orizzontali, verticali e diagonali di “parola” in “puzzle”
 printf("\nInserisci la parola da cercare (max %d caratteri) : ", N);
 scanf("%s", parola);
 /* Per ogni carattere di puzzle controllo le 8 direzioni, sommando l’esito
   (1, 0) al contatore opportuno (2 controlli per hor, 2 per ver, 4 per dia) */
 for ( i=0 ; i<N ; i++ ) {
  for ( j=0 ; j<N ; j++ ) {

    hor += controlla(parola, i, j, 0, 1) + controlla(parola, i, j, 0,-1);
    ver += controlla(parola, i, j, 1, 0) + controlla(parola, i, j,-1, 0);
    dia += controlla(parola, i, j, 1, 1) + controlla(parola, i, j,-1,-1) +
           controlla(parola, i, j, 1,-1) + controlla(parola, i, j,-1, 1);
  }
 }
 if ( hor + ver + dia > 0 )
  printf("H: %d   V: %d   D: %d\n", hor, ver, dia);
 else
  printf("La parola non si trova\n");

 return 0;
}
```

```
int controlla (char p[], int i, int j, int sv, int so) {
  int len = strlen(p), k = 0;
  while ( i>=0 && j>=0 && i<N && j<N && k<len ) { // Esce se usciamo dai bordi o
                                                  // se abbiamo conrollato tutta p
    if ( p[k++] != puzzle[i][j] )
      return 0;                    // se il k-esimo è diverso, return 0 !
    i += sv;
    j += so;                     // modifichiamo pure i e j – cono copie!
  }
  return k == len;             // Se arriva qui, i primi k caratteri di p
}                             // sono stati trovati in sequenza nella martice,
                             // a partire dalla cella [i,j] e nella direzione
                            // indicata da sv e so, quindi la parola c'è tutta
                           // se e solo se k è pari alla lunghezza di p
```

Esercizio (tdeB 24-11-2006)

Si definisce *terna pitagorica intera (TP)* una terna ⟨a,b,c⟩ di numeri interi positivi che soddisfano la relazione di Pitagora (cioè sono le lunghezze dei cateti e dell'ipotenusa di un triangolo rettangolo). Esempio: ⟨10,6,8⟩ (infatti 100=36+64). Una TP si dice *irriducibile (**TPI**)* se non esiste una TP da cui la si deriva moltiplicando per un fattore costante (Es: ⟨3,4,5⟩ è irriducibile, ⟨8,6,10⟩ e ⟨15,9,12⟩ non lo sono).

Si codifichi un programma C che legge da standard input una sequenza (di lunghezza a priori illimitata) di interi terminata dal valore 0 e, al termine della sequenza, visualizza su standard output un messaggio che indica ***quante TPI*** di numeri adiacenti sono contenute nella sequenza. Esempio (si noti che ogni numero può appartenere a 0, 1, 2 o 3 TPI):

```
> -4 11  4  5  3  4  5 12 13 -6 -8 26 10 24 70 74 -4 -5 -3 1 35 12 37 6 0
> Ci sono 5 terne irriducibili
```

riducibile riducibile

N.B.: Nella soluzione **si definisca e si utilizzi** la funzione:

```
int tpi(int,int,int); // Restituisce 1 se i parametri sono una TPI, 0 altrimenti
```

Suggerimenti e aiuti:

- È **inutile** (**e** fortemente **sconsigliato**) memorizzare la sequenza
- La funzione **`mcd(int,int)`**, che calcola e restituisce il M.C.D. dei parametri, può essere usata senza ridefinirla (è definita qui a lato)
- Attenzione: ⟨4,5,3⟩ ⟨5,3,4⟩ ⟨3,4,5⟩ ⟨4,3,5⟩ ⟨5,4,3⟩ ⟨3,5,4⟩ sono tutte TPI
- I tre numeri che formano una TPI sono a due a due primi fra loro

```
int mcd(int m, int n)
{
  while ( m != n )
    if ( n > m ) n -= m;
    else         m -= n;
  return n;
}
```

```
int tpi (int a, int b, int c) {
 if ( a<=0 || b<=0 || c<=0 )     // Controlla che sia un triangolo non degenere
   return 0;
 if ( ( a*a == b*b + c*c ||  b*b == a*a + c*c || c*c == a*a + b*b ) && mcd(a,b) == 1 ) //se è una TP controlla
   return 1;                                                                          //anche che sia una TPI
 else return 0;
}

int main() {
 int terneirriducibili = 0, a, b = 0, c = 0;

 do {
   a = b;                        // Slittamento in avanti della "memoria"
   b = c;
   scanf("%d", &c);              // Lettura del nuovo valore

   terneirriducibili += tpi(a,b,c);     // Controlla l'irriducibile pitagoricità

 } while ( c != 0 );             // Termina digitando lo 0 ("sentinella")

 printf("Ci sono %d terne irriducibili\n", terneirriducibili);

 return 0;
}
```

Esercizio (tdeB 24-11-2006)

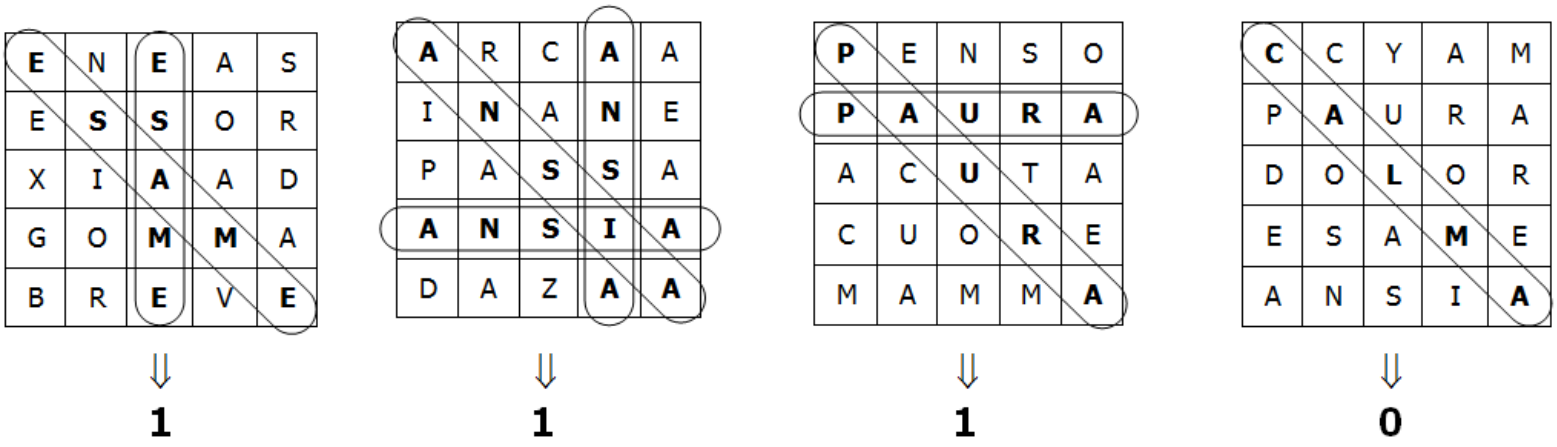
Si definisca una funzione C di prototipo

```
int controlla(char m[][N])
```

che (1) riceve come unico parametro una matrice quadrata NxN (dove N è una costante positiva già definita), (2) controlla se la parola leggibile sulla diagonale principale è leggibile anche in una delle righe o colonne della matrice stessa, e (3) restituisce **1** in caso affermativo, **0** altrimenti.

Prima del codice [***2,5** punti*] si spieghi (in modo sintetico ma preciso) l'algoritmo scelto [***1,5** punti*]

Esempi (per N = 5):



Per i che va da 0 a N-1 scandisco dapprima la i-esima riga e poi la i-esima colonna, controllando (un carattere alla volta) l'identità tra la parola in diagonale e la parola via via scandita.
Quando trovo due caratteri diversi in posizioni corrispondenti posso interrompere subito la scansione corrente e passare alla riga o colonna "successiva".
La prima volta che trovo interamente la parola cercata posso terminare la funzione restituendo 1, mentre solo dopo aver controllato anche l'ultima colonna posso restituire 0.

```
int controlla(char m[][N]) {
  int i, j, trovato;
  for( i=0; i<N; i++ ) {          /* Ogni iterazione controlla una riga e una colonna          */
    j = 0;
    trovato = 1;
    while ( trovato && j<N ) {    /* Scansione della i-esima riga          */
      if ( m[j][j] != m[i][j] )
        trovato = 0;              /* carattere diverso -> parola non trovata */
      j++;
    }
    if ( trovato )                /* Può restituire 1 appena trova la parola */
      return 1;

    j = 0;                        /* reset per controllare la colonna      */
    trovato = 1;
    while ( trovato && j<N ) {    /* Scansione della i-esima colonna       */
      if ( m[j][j] != m[j][i] )
        trovato = 0;              // idem...
      j++;
    }
    if ( trovato )
      return 1;
  }
  return 0;
}
```

Un'altra possibile soluzione, più sintetica ma forse meno intuitiva:

```
int controlla(char m[][N]) {
  int i, j, contOriz, contVert;
  for( i=0; i<N; i++ ) {          /* Ogni iterazione controlla una riga e una colonna          */
    contOriz = contVert = 0;      /* Riazzera i contatori                                      */
    for( j=0; j<N; j++ ) {        /* Scansione simultanea                                      */
      if ( m[j][j] == m[i][j] )
        contOriz++;               /* conta caratteri uguali sulla riga i    */
      if ( m[j][j] == m[j][i] )
        contVert++;               /* conta caratteri uguali sulla colonna i */
    }
    if ( contVert==N || contOriz==N ) /* restituisce 1 appena trova la parola */
      return 1;                       /* cioè se ci sono N caratteri uguali   */
  }                                   /* su una stessa riga o colonna         */
  return 0;
}
```

Esercizio (tdeB 29-1-2008)

- Si consideri il seguente programma:
- Si indichi sinteticamente qual è l'effetto della procedura sfp(n)
- Riportare qui di seguito l'output del programma su stdout

```
int div( int *n, int dc ) {
  if( *n % dc != 0 )
    return div( n, dc+1 );
  *n = *n / dc;
  return dc;
}

void sfp( int n ) {
  int md = min_div(&n);
  if( md > 1 )
    printf("%d ", md);
  if( n > 1 )
    sfp(n);
}

int min_div( int *n ) {
  if( *n <= 0 )
    printf("non positivo! ");
  if( *n == 1 )
    return 1;
  return div(n, 2);
}
```

```
int main() {
  int i, v[6] = { 15, 14, 121, 64, 1, 0 };
  for( i=0; i<6; i++ ) {
    printf("\n%d: ", v[i]); sfp(v[i]);
  }
  return 0;
}
```

15: 3 5
14: 2 7
121: 11 11
64: 2 2 2 2 2 2
1:
0: non positivo! 2

Esercizio (tdeB 18-2-2008)

- Si indichi quale funzione è calcolata da ep() e quale effetto ha myf() quando è eseguita, come nell'esempio, con la variabile e posta a 0
- Si disegni lo stack dei record di attivazione nel momento in cui per la prima volta la funzione ep() esegue l'istruzione **return** 1 indicata dalla freccia. Si rappresentino tutti i record (incluso quello del main), mostrando **tutte** le variabili e adottando le solite convenzioni (vettori: blocchi contigui; puntatori: frecce; valori indefiniti: punti interrogativi)
- Si indichi la linea stampata dal programma su stdout

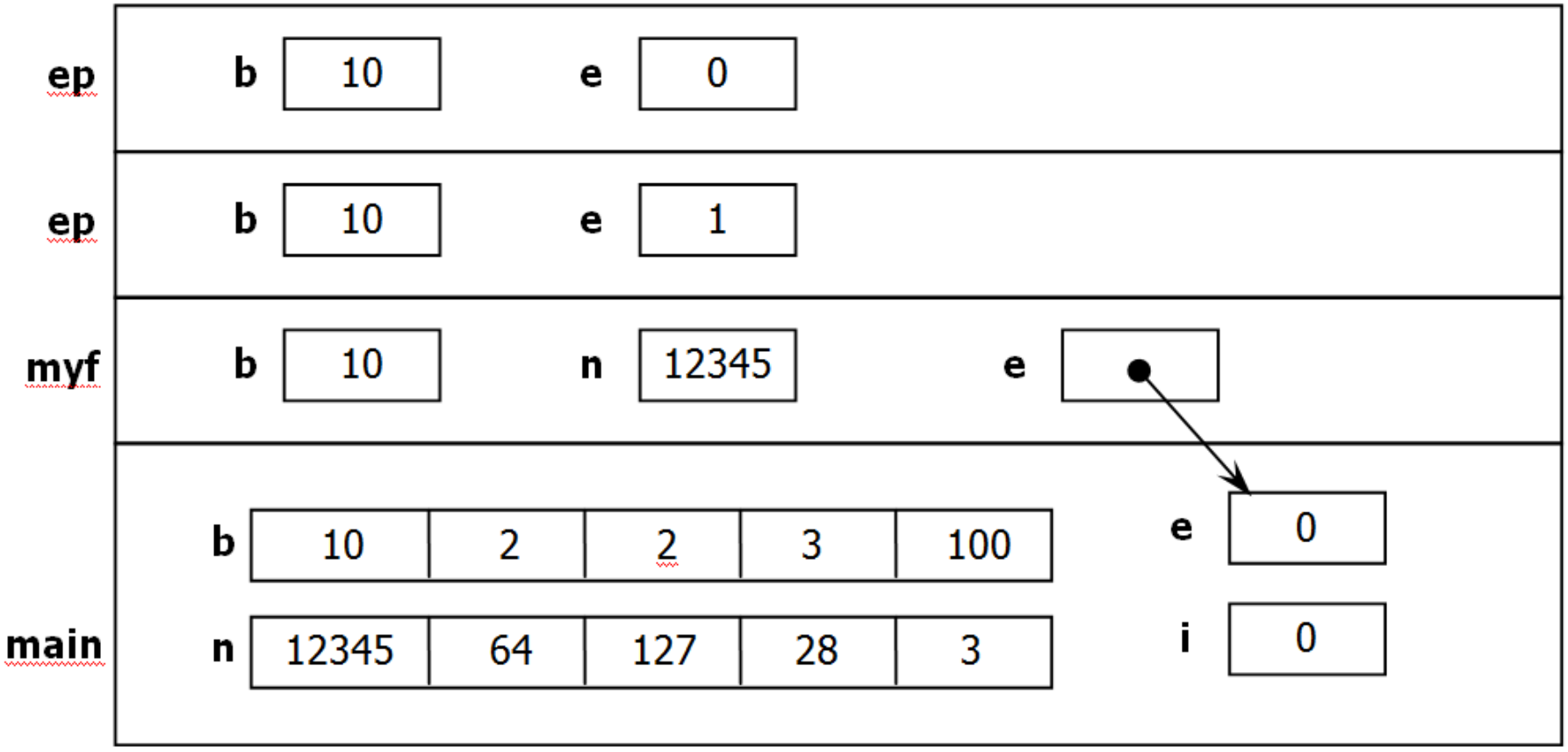
```
int ep( int b, int e ) {
  if( e > 0 )
    return b * ep( b, e-1 );
  if( e < 0 )
    printf("Negativo!! -> NO");
  return 1;
}

void myf( int b, int n, int *e ) {
  if( ep( b, (*e)+1 ) > n )
    return;
  (*e)++;
  myf( b, n, e );
}

int main() {  int b[5] = {   10,  2,   2,  3, 100};
          int n[5] = {12345, 64, 127, 28,   3}, i, e;
          for( i=0; i<5; i++ ) {
               e = 0;
               myf( b[i], n[i], &e );
               printf("%d ", e);
           }
           return 0;
}
```

ep() effettua l'elevamento di b all'esponente e, a patto che e sia non negativo.

myf() verifica se b^{e+1} eccede n, e se non è così incrementa e (nell'ambiente del chiamante) fino a quando tale condizione non sia verificata.
Se e parte da 0, la procedura termina quando e rappresenta *la parte intera del logaritmo* in base b di n



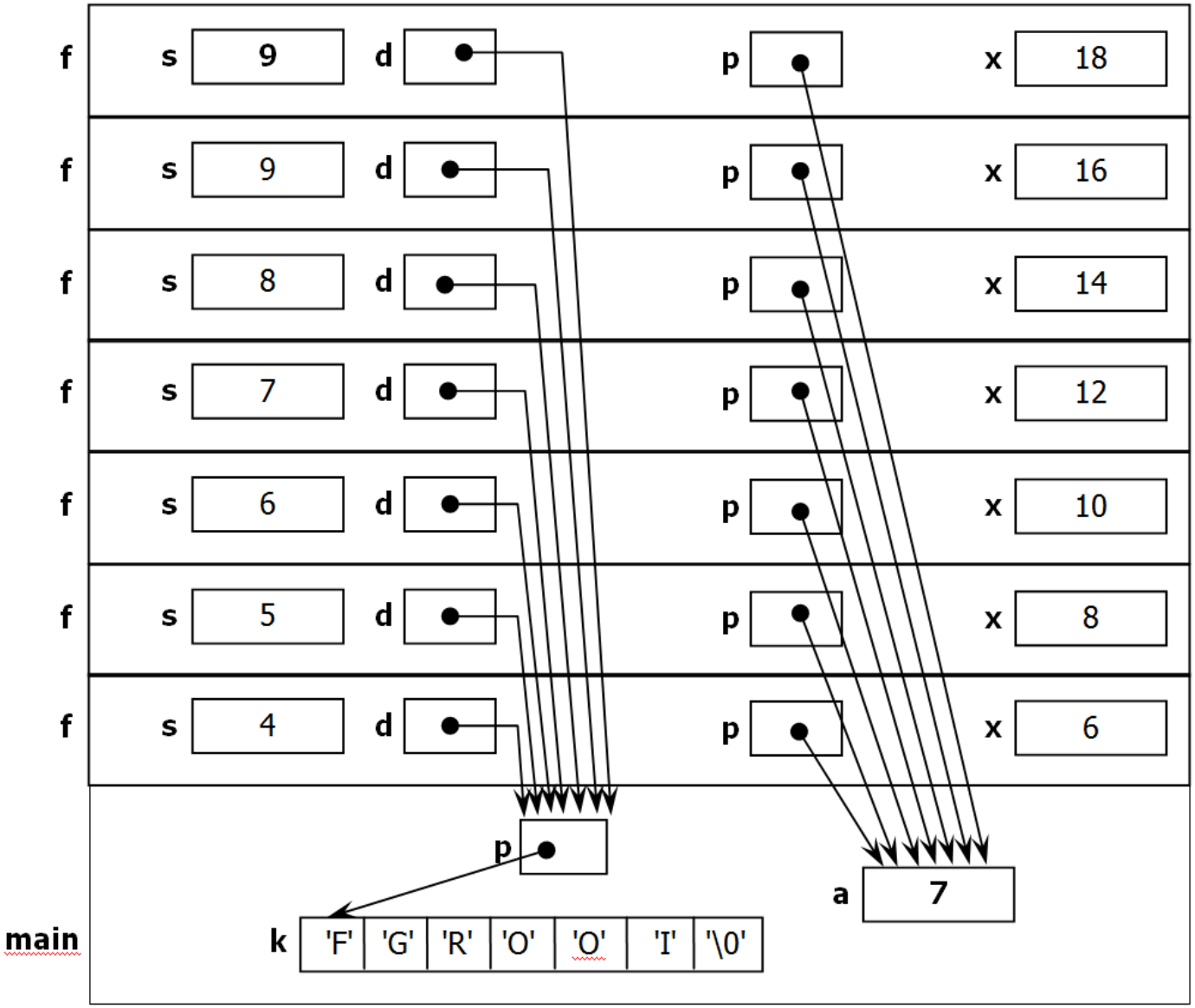
La linea stampata dal programma su stdout.
4 6 6 3 0

Esercizio (tdeB 21/7/2008)

- Si consideri il seguente programma:
- Si disegni lo stack dei record di attivazione nel momento in cui la funzione f() esegue per la prima volta l'istruzione **return** indicata dalla freccia. Si rappresentino tutti i record (incluso quello del main), mostrando **tutte** le variabili e adottando le solite convenzioni (vettori: blocchi contigui; puntatori: frecce; valori indefiniti: punti interrogativi). Si tenga presente che le lettere dell'alfabeto sono codificate in ASCII nel seguente ordine: … E F G H I J K L M N O P Q R S T …
- Si indichi la linea stampata dal programma su stdout

```
void f( char **d, int *p, int s ) {
  int x = 2*s;
  if ( strlen(*d) > 0 ) {
    printf("%c", (**d)+((*p)++) );
    ++(*d);
    f( d, p, ++s );
  }
  else
    printf("  ->  %d!", x-13 );
  return;
}

int main() {
  int a = 1;
  char k[7] = "FGROOI";
  char *p = k;
  f( &p, &a, 3 );
  return 0;
}
```



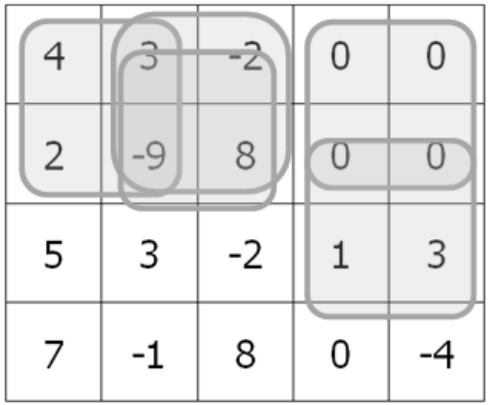
Stampa

GIUSTO -> 5!

Esercizio (tdeB 16/9/2009)

Si consideri una matrice A di interi, di dimensione N x M. Definiamo *clique* una sottomatrice di dimensione 2 x 2 i cui valori abbiano somma algebrica pari a zero. Nella matrice in figura sono evidenziate tutte le clique.

Le funzioni `...contaclique( ... )` e `...senzaclique(...)` ricevono in ingresso una matrice di interi di dimensione N x M e restituiscono, rispettivamente, il numero di clique contenute nella matrice e 1 o 0 a seconda che la matrice sia o meno priva di clique. Si implementino le due funzioni in C



Definiamo una funzione int quiclique(int m[][M], int i, int j) che restituisce 1 se la sottomatrice con elemento superiore sinistro in posizione i,j è una clique.
Per la funzione contaclique possiamo scandire tutti gli elementi della matrice da (0,0) fino a (N-2,M-2) e contare per quanti per quanti elementi (i,j) quiclique restituisce 1.
Per la funzione senzaclique possiamo semplicemente invocare contaclique e verificare che il numero sia zero. Questo però comporta sempre la scansione dell'intera matrice, quando si potrebbe invece arrestare l'analisi già dopo il ritrovamento della prime clique. Nel caso d'esempio, infatti, si potrebbe arrestare l'analisi già dopo aver verificato che qiuclique(0,0) == 1.

```
int quiclique( int m[][M], int i, int j ) {
  return m[i][j] + m[i+1][j] + m[i][j+1] + m[i+1][j+1] == 0;
}

int contaclique( int m[][M] ) {
  int i, j, cont = 0;
  for ( i = 0; i < N-1; i++ )
    for ( j = 0; j < M-1; j++ )
      cont += quiclique( m, i, j );
  return cont;
}

int senzaclique1( int m[][M] ) {
  return contaclique(m) == 0;
}

int senzaclique2( int m[][M] ) {
  int i, j;
  for ( i = 0; i < N-1; i++ )
    for ( j = 0; j < M-1; j++ )
      if ( quiclique(m, i, j) )
        return 0;
  return 1;
}
```

Esercizio (tdeB 25-2-2011)

- Un semplicissimo (e fragile) sistema di cifratura dei messaggi consiste nel trasformare ogni lettera in una diversa particolare lettera. Il sistema che vi si propone agisce sulle sole lettere alfabetiche e dispone di un vettore chiave di interi, che definisce lo spiazzamento specifico da applicare alle 26 diverse lettere dell'alfabeto inglese. Per semplicità consideriamo solo le lettere maiuscole.
- **typedef int** chiave[26]**;**
- **typedef char *** messaggio;
- Il primo elemento del vettore chiave indicherà lo spiazzamento da applicare alle *a* (se è 3, significa che tutte le *a* si tradurranno in *d*), il secondo lo spiazzamento per le *b*, ..., l'ultimo quello per le *z* (se è -13, si in *m*). Tutti gli altri eventuali caratteri (non alfabetici) del messaggio restano inalterati.

Si codifichi in C la funzione ...cipher(...) che riceve come parametri una chiave e un messaggio e alloca e restituisce il messaggio cifrato (che è una stringa dinamica).

Si dica qual è la minima modifica della funzione precedente con cui è possibile ottenere una funzione ...decipher(...)che effettui la trasformazione inversa.

Non tutte le chiavi sono altrettanto efficaci. Alcune possono essere ambigue e rendere più o meno indecifrabile il messaggio. Ad esempio una chiave k = {0, -1, -2, -3, ..., -25} è estremamente ambigua perché trasforma ogni lettera in una *a*. Si codifichi in C una funzione ...nonlossy(...) che controlla se una chiave permette sempre di ricostruire i messaggi non causando perdita d'informazione.

Esercizio (tdeB 14-9-2011)

- Un programma legge dallo standard input una sequenza di caratteri teoricamente illimitata, costituita solo da cifre decimali [0-9] e dai caratteri '+', '–' e '='. Il segno + indica che le cifre successive sono da considerarsi positive, il segno – che le cifre successive (tutte, fino al ricevimento del prossimo segno) sono da considerarsi negative. Un segno può anche “confermare” il segno “corrente”. Se il primo carattere non è un segno, inizialmente le cifre si considerano positive.
- Il programma calcola la somma algebrica delle cifre lette, e quando riceve un '=' stampa la somma accumulata fino a quel momento e la reinizializza a 0 (senza modificare il “segno corrente”); se, nell’accumulare la somma, rileva che si è verificato un overflow (dovuto a un’eccessiva accumulazione di valori tutti positivi o tutti negativi), stampa “OVERFFFFF...!” e **termina**.
- Esempio:

  stdin: 1311=23=4-45-11=121+15+7--4=1+9999999999 ... 999999 ... 99999999

  output: 6 5 -7 5 OVERFFFFF...!
- Si codifichi il programma in C.

Esercizio (tdeB 14-11-2011)

- Si consideri il seguente programma. Si disegni lo stack dei record di attivazione nel momento in cui la sua dimensione è massima
- Si rappresentino tutti i record (incluso quello del main), mostrando tutte le variabili e adottando le solite convenzioni (vettori: blocchi contigui; puntatori: frecce; valori indefiniti: punti interrogativi)
- Si indichi la linea stampata dal programma su stdout
- Attenzione! Il parametro c è un puntatore passato per indirizzo.

```
void g(char **d, int n){
 int i = strlen(*d);
 if( n < 0 )
  printf("%c",(*d)[i+n]);
 else {
  f( n-1, *d );
  printf("%c",*((*d)+n));
 }
 return;
}

void f(int n, char * p){
  char a = p[n];
  if( strlen(p) == 0 )
    return;
  g( &p, n-1 );
  printf("%c", *(&a) );
  return;
}
```

```
int main() {
 char cpu[]="RISC" ;
 char *s = cpu;
 int k = 3;
 f( --k, cpu );
 printf("%c :(",*(++s));
 return 0;
}
```