

Docente: Monica Nesi

Nota Nelle soluzioni di alcuni degli esercizi riportati si suppone di avere le seguenti direttive e definizioni globali:

```
#include <stdio.h>
#define lung 10
```

```
typedef enum {false,true} boolean;
```

Esercizi

1. Scrivere una funzione che, dati un intero n ed un array di interi a , restituisce *true* se n occorre in a , *false* altrimenti.

```
boolean occorre (int n, int a[], int l)
{
    int i;
    for (i=0; i<l; i++)
        if (n == a[i]) return true;
    return false;
}
```

Scrivere una funzione che, dati un intero n ed un array di interi a , restituisce il numero delle occorrenze di n in a .

```
int occorrenze (int n, int a[], int l)
{
    int i, conta;
    conta = 0;
    for (i=0; i<l; i++)
        if (n == a[i])
            conta = conta+1;
    return conta;
}
```

Scrivere una funzione che, dati un intero n ed un array di interi a , restituisce la posizione della prima occorrenza di n in a (se n occorre in a), altrimenti restituisce -1 .

```
int posizione (int n, int a[], int l)
{
    int i, pos;
    pos = 1;
    for (i=0; i<l; i++)
    {
```

¹Si prega di segnalare ogni eventuale errore all'indirizzo di posta elettronica monica@di.univaq.it.

```

        if (n == a[i]) return pos;
        pos = pos+1;
    }
    return -1;
}

```

Utilizzando tutte le funzioni finora definite, si può avere il seguente programma in C:

```

main()
{
    int a[lung];
    int i, num, c, p;

    boolean occorre (int n, int a[], int l);    /* prototipo funzioni */
    int occorrenze (int n, int a[], int l);
    int posizione (int n, int a[], int l);

    i = 0;
    scanf("%d%d", &num, &a[i]);
    while (a[i] != 0 && i < lung)
    {
        i = i+1;
        scanf("%d", &a[i]);
    }

                                                                    /* i = lunghezza sequenza */
    if (occorre(num,a,i))
    {
        c = occorrenze(num,a,i);
        p = posizione(num,a,i);
        printf("Il numero %d occorre nella sequenza %d volte e la sua prima
    occorrenza si trova in posizione %d della sequenza.\n", num, c, p);
    }
    else printf("Il numero %d non occorre nella sequenza.\n", num);
}

boolean occorre (int n, int a[], int l)
{
    int i;
    for (i=0; i<l; i++)
        if (n == a[i]) return true;
    return false;
}

int occorrenze (int n, int a[], int l)
{
    int i, conta;
    conta = 0;
    for (i=0; i<l; i++)
        if (n == a[i])
            conta = conta+1;
    return conta;
}

int posizione (int n, int a[], int l)
{

```

```

    int i, pos;
    pos = 1;
    for (i=0; i<l; i++)
    {
        if (n == a[i]) return pos;
        pos = pos+1;
    }
    return -1;
}

```

2. Scrivere una funzione che, dato un numero intero positivo n , restituisce *true* se n è un numero primo, altrimenti restituisce *false*.

```

boolean primo (int n)
{
    int cont;
    boolean div;
    if (n == 1) return false;
    else
    { cont = 2;
      div = false;
      while (cont <= n/2 && !div)
      {
          if (n%cont == 0 ) div = true;
          else cont = cont+1;
      }
      if (div) return false;
      else return true;
    }
}

```

3. Scrivere due funzioni che, dati due numeri interi positivi, restituiscono, rispettivamente, il loro massimo comun divisore ed il loro minimo comune multiplo.

```

int MCD (int n, int m)
{
    int x, y;
    x = n;
    y = m;
    while (y != x)
    {
        if (y > x) y = y-x;
        else x = x-y;
    }
    return x;
}

```

```

int mcm (int n, int m)
{
    int cont, prodotto;
    boolean trovato;
    prodotto = n*m;
    trovato = false;
    if (n > m) cont = n;

```

```

    else cont = m;
    while (cont < prodotto && !trovato)
    {
        if (cont%n == 0 && cont%m == 0) trovato = true;
        else cont = cont+1;
    }
    return cont;
}

```

4. Scrivere una funzione che calcola il fattoriale di un numero naturale n .

```

int fattoriale (int n)          /* Soluzione iterativa */
{
    int f, i;
    if (n < 0) return -1;
    f = 1;
    for (i=1; i<=n; i++)
        f = f*i;
    return f;
}

```

```

int fattRic (int n)           /* Soluzione ricorsiva */
{
    if (n < 0) return -1;
    if (n == 0) return 1;
    return n*fattRic(n-1);
}

```

5. Scrivere una funzione che calcola l' n -esimo numero di Fibonacci:

$$\begin{aligned}
 fib(0) &= 1 \\
 fib(1) &= 1 \\
 fib(n) &= fib(n-2) + fib(n-1) \quad n > 1
 \end{aligned}$$

```

int fibonacci (int n)        /* Soluzione iterativa */
{
    int fib, x, i, temp;
    if (n < 0) return -1;
    fib = 1;
    x = 0;
    for (i=1; i<=n; i++)
    {
        temp = x+fib;
        x = fib;
        fib = temp;
    }
    return fib;
}

```

```

int fibRic (int n)          /* Soluzione ricorsiva */
{
    if (n < 0) return -1;
    if (n < 2) return 1;
    return fibRic(n-2) + fibRic(n-1);
}

```

6. Scrivere due funzioni che, dato un array di elementi a , restituiscono, rispettivamente, il valore minimo ed il valore massimo di a . *Nota:* le versioni ricorsive scorrono l'array dall'ultimo elemento fino al primo. Il parametro l indica la prima posizione libera nell'array e coincide con la lunghezza dell'array se l'array è stato tutto riempito.

```
int minimo (int a[], int l)                /* Soluzione iterativa */
{
    int i, min;
    i = 0;
    min = a[0];
    while (i < l)
    {
        if (a[i] < min) min = a[i];
        i = i+1;
    }
    return min;
}

int minimoRic (int a[], int l)            /* Soluzione ricorsiva */
{
    int minimoRicAux (int a[], int i, int min);

    return minimoRicAux(a,l-1,a[l-1]);
}

int minimoRicAux (int a[], int i, int min)
{
    if (i < 0) return min;
    if (a[i] < min) min = a[i];
    return minimoRicAux(a,i-1,min);
}

int massimo (int a[], int l)             /* Soluzione iterativa */
{
    int i, max;
    i = 0;
    max = a[0];
    while (i < l)
    {
        if (a[i] > max) max = a[i];
        i = i+1;
    }
    return max;
}

int massimoRic (int a[], int l)          /* Soluzione ricorsiva */
{
    int massimoRicAux (int a[], int i, int max);

    return massimoRicAux(a,l-1,a[l-1]);
}

int massimoRicAux (int a[], int i, int max)
{
    if (i < 0) return max;
```

```

    if (a[i] > max) max = a[i];
    return massimoRicAux(a,i-1,max);
}

```

7. Scrivere una funzione che, dato un array di interi (non vuoto), restituisce *true* se l'array è ordinato in modo non decrescente, altrimenti restituisce *false*.

```

boolean nondecr (int a[], int l)
{
    /* versione iterativa */
    int i;
    i = 0;
    while (i < l-1)
    {
        if (a[i] > a[i+1]) return false;
        i = i+1;
    }
    return true;
}

```

```

boolean nondecrRic (int a[], int l)
{
    /* versione ricorsiva */
    boolean nondecrRicAux (int a[], int l, int i);

    return nondecrRicAux (a,l,0);
}

```

```

boolean nondecrRicAux (int a[], int l, int i)
{
    if (i == l-1) return true;
    if (a[i] > a[i+1]) return false;
    return nondecrRicAux(a,l,i+1);
}

```

8. Scrivere una procedura che, dato un array di interi *a* ed un intero *n*, inserisce *n* nella prima posizione libera di *a*.

Soluzione con variabili globali

```

typedef int MioArray[lung];
MioArray a;
int cont;

main()
{
    int n;

    void inserisciInt (int n);      /* prototipo procedure */
    void stampaMioArray();

    cont = 0;
    scanf("%d", &n);
    while (n != 0 && cont < lung)
    {
        inserisciInt(n);
        scanf("%d", &n);
    }
    /* cont = lunghezza sequenza */
}

```

```

    printf("L'array e' il seguente:\n");
    stampaMioArray();
}

void inserisciInt (int n)
{
    if (cont < lung)          /* condizione sempre verificata dal main */
    {
        a[cont] = n;
        cont = cont+1;
    }
    else printf("array pieno");
}

void stampaMioArray()
{
    int i;
    for (i=0; i<cont; i++)
        printf("%d ",a[i]);
    printf("\n");
}

```

Soluzione con parametri di tipo puntatore

```

main()
{
    int n, cont;
    int a[lung];

    void inserisciInt (int x, int a[], int l, int *pcont);
    void stampaArray (int a[], int l);

    cont = 0;
    scanf("%d", &n);
    while (n != 0 && cont < lung)
    {
        inserisciInt(n,a,lung,&cont);
        scanf("%d", &n);
    }
    /* cont = lunghezza sequenza */

    printf("L'array e' il seguente:\n");
    stampaArray(a,cont);
}

void inserisciInt (int x, int a[], int l, int *pcont)
{
    if (*pcont < l)
    {
        a[*pcont] = x;
        *pcont = *pcont + 1;
    }
    else printf("array pieno");
}

```

```

void stampaArray (int a[], int l)
{
    int i;
    for (i=0; i<l; i++)
        printf("%d ",a[i]);
    printf("\n");
}

```

9. Scrivere una procedura che, dato un array di interi a ordinato in modo non decrescente ed un intero n , inserisce n in a in modo tale che l'array risultante sia ancora ordinato in modo non decrescente (*ordinamento di un array per inserimento*).

```

void inserisciOrd (int n)                /* Soluzione con variabili globali */
{                                        /* MioArray a; int cont; */
    int i,j;
    boolean trovato;
    if (cont < lung)
    {
        i = 0;
        trovato = false;
        while (i < cont && !trovato)    /* si ricerca la posizione in */
        { if (a[i] <= n)                /* cui inserire l'elemento */
            i++;
            else trovato = true;
        }
        for (j = cont; j > i; j--)      /* si spostano gli elementi in */
            a[j] = a[j-1];             /* posiz. > i a partire dall'ultimo */
        a[i] = n;                       /* si inserisce l'elemento */
        cont++;
    }
    else printf("array pieno");
}

```

10. Scrivere una procedura che, dato un array di interi, lo ordina in modo non decrescente (*ordinamento di un array per selezione*).

```

void sort (int a[], int l)
{
    int k, minpos;

    int minPos (int a[], int l, int from);
    void swap (int a[], int i, int j);

    for (k=0; k< l; k++)
    { minpos = minPos(a,l,k);
      if (minpos != k)
          swap(a,minpos,k);
    }
}

int minPos (int a[], int l, int from)
{
    int minpos, i;
    minpos = from;
    for (i=from+1; i<l; i++)

```



```

        if (a[i] < a[minpos])
            minpos = i;
    return minpos;
}

void swap (int a[], int i, int j)
{
    int temp;
    temp = a[i];
    a[i] = a[j];
    a[j] = temp;
}

```

11. Scrivere una funzione che, dato un array di interi a ed un intero n , restituisce la posizione di n in a se n occorre in a , altrimenti restituisce -1. (Simile alla funzione `posizione` che però conta la posizione a partire da 1.)

```

int ricercaSeq (int n, int a[], int l)
{
    /* ricerca sequenziale o lineare */
    /* versione iterativa */
    int i, pos;
    pos = 0;
    for (i=0; i<l; i++)
    {
        if (n == a[i]) return pos;
        pos = pos+1;
    }
    return -1;
}

```

```

int ricercaSeqRic (int n, int a[], int l)
{
    /* versione ricorsiva */
    int ricercaSeqRicAux (int n, int a[], int l, int i);

    return ricercaSeqRicAux(n,a,l,0);
}

```

```

int ricercaSeqRicAux (int n, int a[], int l, int i)
{
    if (i == l) return -1;
    if (a[i] == n) return i;
    return ricercaSeqRicAux(n,a,l,i+1);
}

```

12. Scrivere una funzione che, dato un array di interi a ordinato in modo non decrescente ed un intero n , restituisce la posizione di n in a se n occorre in a , altrimenti restituisce -1.

```

int ricercaSeqOrd (int n, int a[], int l)
{
    /* ricerca sequenziale o lineare */
    int i;
    i = 0;
    while (i < l)
    { if (a[i] < n) i = i+1;
      else if (a[i] == n) return i;
      else return -1;
    }
}

```

```

    return -1;
}

int ricercaBin (int v, int a[], int l)
{
    int from, to, mid;
    from = 0;
    to = l-1;
    while (from <= to)
    {
        mid = (from+to)/2;
        if (a[mid] == v)
            return mid;
        else
            if (a[mid] < v) from = mid+1;
            else to = mid-1;
    }
    return -1;
}

int ricercaBinRic (int v, int a[], int l)
{
    int ricercaBinRicAux (int v, int a[], int from, int to);

    return ricercaBinRicAux(v,a,0,l-1);
}

int ricercaBinRicAux (int v, int a[], int from, int to)
{
    int mid;
    if (from > to) return -1;
    mid = (from+to)/2;
    if (a[mid] == v)
        return mid;
    else
        if (a[mid] < v)
            return ricercaBinRicAux(v,a,mid+1,to);
        else
            return ricercaBinRicAux(v,a,from,mid-1);
}

```

13. Scrivere una funzione ricorsiva che, dato un array di interi (non vuoto), restituisce *true* se ogni elemento dell'array (tranne il primo) è il triplo dell'elemento che lo precede, altrimenti restituisce *false* (*versione con funzione ricorsiva di un esercizio della prova intermedia del 13/2/2003*).

```

boolean triplo (int a[], int l)
{
    boolean triploRic (int a[], int l, int i);

    return triploRic (a,l,1);
}

boolean triploRic (int a[], int l, int i)
{

```

```

    if (i == 1) return true;
    if (a[i] != 3*a[i-1]) return false;
    return triploRic(a,l,i+1);
}

```

oppure scorrendo l'array dall'ultimo al primo elemento ed utilizzando solo 1, parametro che indica la prima posizione "libera" dell'array:

```

boolean triploA (int a[], int l)
{
    if (l == 1) return true;
    if (a[l-1] != 3*a[l-2]) return false;
    return triploA(a,l-1);
}

```

14. Scrivere una procedura ricorsiva che, dati tre interi i , j ed n ($n \geq 2$), stampa una sequenza di n numeri, i cui primi due elementi sono i e j , ed ogni elemento successivo è uguale al prodotto dei due elementi che lo precedono nella sequenza (*versione con procedura ricorsiva di un esercizio della prova intermedia del 13/2/2003*).

```

void prodottoElem (int i, int j, int n)
{
    void prodottoElemRic (int i, int j, int n, int k);

    printf("%d %d", i, j);
    prodottoElemRic(i,j,n,2);
}

```

```

void prodottoElemRic (int i, int j, int n, int k)
{
    if (k == n) printf("\n");
    else
    { printf(" %d", i*j);
      prodottoElemRic(j,i*j,n,k+1);
    }
}

```

15. Scrivere una procedura ricorsiva che, dato un array di interi a , stampa la sequenza di quegli elementi che compaiono esattamente una volta in a nello stesso ordine in cui compaiono nell'array (*versione con procedura ricorsiva di un esercizio della prova intermedia del 13/2/2003*).

```

void occorr1 (int a[], int l)
{
    void occorr1Ric (int a[], int l, int i);

    occorr1Ric(a,l,0);
}

```

```

void occorr1Ric (int a[], int l, int i)
{
    int conta (int n, int a[], int l);

    if (i == l) printf("\n");
    else

```

```

    {
        if (conta(a[i],a,l) == 1)
            printf("%d ", a[i]);
        occorr1Ric(a,l,i+1);
    }
}

int conta (int n, int a[], int l)
{
    int contaRic (int n, int a[], int i, int k);

    return contaRic(n,a,l,0);
}

int contaRic (int n, int a[], int i, int k)
{
    if (i == 0) return k;
    if (a[i-1] == n) k++;
    return contaRic(n,a,i-1,k);
}

```

16. Scrivere una procedura ricorsiva che stampa le mosse che risolvono il problema delle torri di Hanoi (vedere Esercizio 8.3 del testo di riferimento per una descrizione di tale problema). Si supponga di indicare con i numeri 1, 2 e 3 i pioli di sorgente, tramite e destinazione. Sia n il numero dei dischi da spostare.

```

main()
{
    int n;

    void hanoi (int n, int s, int d, int t);

    scanf("%d", &n);
    printf("Le mosse per %d dischi sono le seguenti:\n", n);
    hanoi(n,1,3,2);
}

void hanoi (int n, int s, int d, int t)
{
    if (n == 1)
        printf("Sposta disco da piolo %d a piolo %d\n", s, d);
    else
    {
        hanoi(n-1,s,t,d);
        hanoi(1,s,d,t);
        hanoi(n-1,t,d,s);
    }
}

```

17. Scrivere una procedura ricorsiva che, data una stringa s , stampa tutti gli anagrammi di s .

```

#include <stdio.h>
#include <string.h>
#define lung 10

```

```

main()
{
    int contatore;
    char s[lung];

    void anagrammi (char prefix[], char s[]);

    scanf("%s", &s);
    printf("Anagrammi:\n");
    anagrammi("",s);
}

void anagrammi (char prefix[], char s[])
{
    if (strlen(s) == 1)
        printf("%s%s\n",prefix,s);
    else
    {
        int i;
        for (i=0; i<strlen(s); i++)
        {
            char sub1[lung];
            char sub2[lung];
            /* Primo argomento */
            strcpy(sub1,prefix);
            strncat(sub1,s+i,1);
            /* Secondo argomento */
            strncpy(sub2,s,i);
            sub2[i]='\0';
            strcat(sub2,s+i+1);
            /* Chiamata ricorsiva */
            anagrammi(sub1,sub2);
        }
    }
}

```