

Soluzioni (Esercizi 1.4, 1.5, 1.6, 2.2, 2.4)

Parte 1

Esercizio 1.4

Descrivere (anche graficamente) l'effetto del seguente frammento di programma.

```
char s[]="stringa";  
char *p=&s[2];  
char *q=p++;
```

Soluzione:

char s[]="stringa"; crea un array di caratteri di dimensione pari ad 8:

```
s=|s|t|r|i|n|g|a|\0|
```

char \*p=&s[2]; assegna al puntatore a carattere p l'indirizzo dell'elemento s[2] (il terzo dell'array s, in quanto occorre ricordare che la numerazione in un array comincia da 0).

```
s=|s|t|r|i|n|g|a|\0|  
  /|  
  |  
  |  
  p
```

Infine char \*q=p++; PRIMA assegna al puntatore a carattere q lo stesso indirizzo cui punta p, e POI sposta p di una posizione:

```
s=|s|t| r | i |n|g|a|\0|  
  /|  /|  
  |  |  
  |  |  
  q  p
```

Esercizio 1.5

Descrivere le funzionalità (cosa calcola) e discutere le assunzioni che devono essere verificate sui parametri delle seguenti funzioni.  
Esemplificare dettagliatamente il loro utilizzo.

A)

```
int Qui(char *a) {
```

```

int i, n=strlen(a);
for(i=0; i<n/2; ++i){
    if(a[i]!=a[n-1-i])
        return 0;
}
return 1;
}

```

B)

```

void Qua(int k){

    if(k==0) return;
    printf("hip ");
    Qua(k-1);
    printf("hurra ");
}

```

Soluzione:

Nota per chi legge: descrivere la funzionalità non significa descrivere passo per passo i diversi processi che compie una funzione, ma ciò che essa restituisce. Per assunzione su un parametro si intende una qualsiasi proprietà posseduta dai parametri non esplicitamente dichiarata, ma necessaria per il corretto funzionamento della funzione (ad esempio, nell'utilizzo della funzione Qui bisogna dire che viene passata una stringa...). Esempificare significa fornire un esempio. Quindi:

La funzione Qui verifica che la stringa passatagli come parametro sia palindroma, ossia possa essere ugualmente letta sia da destra verso sinistra che da sinistra verso destra. Essa restituisce 0 se la funzione NON è palindroma, 1 se lo è.

Esempio:

```

int i;
i=Qui("anna");

allora i vale 1.

int i;
i=Qui("ciao");

allora i vale 0.

```

La funzione ricorsiva Qua riceve innanzitutto un intero k e:

- se k=0 non stampa nulla.
- se k>0 stampa k "hip " seguiti da k "hurra ".

Esempio:

Sia  $k=2$ .

Ecco schematizzati i processi della funzione:

Qua(2) | 1)  $2=0$ ? No. Allora prosegui:

| 2) stampa a schermo hip

| 3)

| 9) stampa a schermo hurra

Qua(1) | 4)  $1=0$ ? No. Allora prosegui:

| 5) stampa a schermo hip

| 6) Qua(0) | 7)  $0=0$ ? Si. Esci da Qua(0),  
e torna a Qua(1).

| 8) stampa a schermo hurra

Seguendo la numerazione ci si accorge di come nell'ordine siano stampati a schermo due hip, seguiti da due hurra.

Quindi l'output della funzione, per  $k=2$ , è:

hip hip hurra hurra

## Esercizio 1.6

Realizzare un programma che richieda un intero ( $n$ ) e stampi la sua tabellina al contrario.

Esempio con  $n=3$ :

30 27 24 21 18 15 12 9 6 3

Soluzione:

Ecco il semplice codice sorgente:

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

```
int main(void){
```

```
int n, i;
```

```
printf("Inserisci un intero n: ");
```

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

```
/* Richiede l'intero n */
```

```
/* Legge l'intero n */
```

```
for(i=10; i>0; --i)
```

```
printf("%d ", n*i);
```

```
/*Stampa quanto richiesto*/
```

```
return 0;
```

```
}
```

## Parte 2

### Esercizio 2.2

Descrivere un algoritmo di ordinamento efficiente. Dire di quale algoritmo si tratta e dare un esempio di ordinamento.

Soluzione:

La mergesort è un algoritmo efficiente. Esso si serve di due funzioni:

```
void merge(int a[], int b[], int c[], int m, int n){
    int i=0; j=0; k=0;

    while(i<m && j<n)
        if(a[i]<b[j])
            c[k++]=a[i++];
        else
            c[k++]=b[j++];
    while(i<m)
        c[k++]=a[i++];
    while(j<n)
        c[k++]=b[j++];
}
```

La merge pone gli elementi di a e b in modo ordinato nell'array c (che all'entrata nella funzione deve già possedere la lunghezza necessaria per ospitare gli m+n elementi). Richiede altresì che gli array a e b siano ordinati quando vengono passati alla funzione.

L'altra funzione è la mergesort che analizziamo solo nel caso in cui si voglia ordinare un array di dimensione pari ad una potenza di due.

```
void mergesort(int key[], int n){

    int m, j, k, *w;

    for(m=1; m<n; m*=2)
        ;
    if(m<n){
        printf("%d non è potenza di due", n);
        exit(1);
    }

    w=calloc(n, sizeof(int));

    assert(w!=NULL);

    for(k=1; k<n; k*=2){
        for(j=0; j<n-k; j+=2*k)
            merge(key+j, key+j+k, w+j, k, k);
        for(j=0; j<n; ++j)
            key[j]=w[j];
    }
    free(w);
}
```

In pratica la mergesort fonde (ordinandoli) prima gli array di dimensione 1, due alla volta:

3 2 1 4      diventa      2 3 1 4,

poi quelli di dimensione 2, due alla volta:

2 3 1 4      diventa      1 2 3 4.

In seguito passa a quelli di dimensione 4 e così via (seguendo cioè le potenze di due).

#### Esercizio 2.4

Sia data la seguente struttura dati LISTA

```
struct ELEM{
int dato;
struct ELEM *prox;
};
```

```
typedef struct ELEM ElementoLista;
typedef ElementoLista *Lista;
```

Si implementino le seguenti funzioni:

```
int ContaElementi(Lista lista);
/*Conta gli elementi della lista*/

int SumLista(Lista lista);
/*Ritorna la somma degli elementi della lista*/

void StampaLista(Lista lista);
/*Stampa i dati contenuti nella lista*/
```

Soluzione:

```
int ContaElementi(Lista lista){

int cnt=0;

while((lista)!=NULL){
++cnt;
lista=lista->prox;
}
return cnt;
}

int SumLista(Lista lista){

int sum=0;
```

```
while (lista!=NULL) {
sum+=(lista->dato);
lista=lista->prox;
}
return sum;
}

void StampaLista(Lista lista) {

if (lista==NULL)
printf("NULL");
else{
printf("%d--> ", lista->dato);
StampaLista(lista->prox);
}
}
```