```
/*Esercizio 1.
Si scriva una funzione C ricorsiva che, preso in input un numero intero,
ne stampa le cifre decimali in ordine inverso,
ovvero dalla meno significativa alla più significativa.
La vostra  soluzione è tail recursive? Si motivi la riposta.
Esempio. Se il numero intero ricevuto come parametro di input è 1234,
 la funzione deve stampare 4321. */

/* Soluzione.*/
void invertiCifre (long n)
  /*postc: stampa a video le cifre di n  in ordine inverso. */
{
  printf("%d", n % 10);
  if (n / 10 != 0)
    invertiCifre(n / 10);
}


Esercizio  2 (Tail Recursion)
Si scriva una funzione ricorsiva in coda che riceve in input una lista
concatenata  di stringhe e restituisce
una lista di stringhe di circa metà elementi in cui l'i-simo (i≤1)
elemento contiene la concatenazione
degli elementi in posizione 2i-1 e 2i (se esiste)  della lista originaria,
che deve rimanere invariata.

Esempio:

input : ab -> cd -> ef
Output : abcd -> ef

Soluzione.


char* conc(char * s, char * t);
/*prec: s != NULL && t != NULL
* postc: restituisce, allocando la memoria, la concatenazione di s e t*/
char* conc(char * s, char * t)
{char* x;
x = (char*) malloc((strlen(s) + strlen(t)+1)*sizeof(char));
x = strcpy(x,s);
x = strcat(x,t);
return x;}

 ListeStrP divMetaConc(ListeStrP lista)
/* post: restituisce una lista contenente nell'elemento i-simo la stringa ottenuta
concatenando le stringhe
*contenute negli elementi di posto 2i-1 e 2i della lista data, lasciandola
inalterata*/
{ListeStrP app;
char* temp;
if (!lista || !(lista->nextPtr)) return lista;
app = (ListeStrP)malloc(sizeof(ListeStr));
```

```
assert(app);
app -> elem = conc(lista -> elem,lista->nextPtr -> elem);
app->nextPtr = divMetaConc(lista ->nextPtr ->nextPtr);
return app;}
```

Esercizio 3.

Si scriva una funzione che, mediante scambi della posizione dei figli
dei nodi di un albero binario
(sottoalbero destro che diventa sottoalbero sinistro e viceversa),
riorganizza un albero in modo tale che, per ogni nodo,
il sottoalbero sinistro contiene un numero di nodi maggiore o
uguale di quello dei nodi del sottoalbero destro.
Analizzare la complessità computazionale della soluzione proposta.
Nota. E' possibile fornire una soluzione che risolve l'esercizio in tempo lineare.

```
/*Soluzione*/

int riordina(TreePtr t)
/*postc: restituisce il numero dei nodi di t e lo riorganizza in modo tale che, per ogni nodo,
*il sottoalbero sinistro contiene un numero di nodi maggiore o uguale di quello dei nodi del sottoalbero destro.*/
{TreePtr temp;
int s,d;
if (!t) return 0;
s = riordina(t->left);
d = riordina (t->right);
if (s < d) /*scambio dei sottoalberi*/
{temp = t ->left;
t -> left  = t -> right;
t -> right = temp;}
return s+d+1;}
```

Esercizio 4

Si individuino opportuni dati di test per la seguente funzione,
evidenziando per quali dati si e' seguito l'approccio a scatola nera
e per quali quello a scatola trasparente.
Si dia una breve motivazione della scelta.

```
char * Rstringchr (char *s, int c)
/*individua la prima occorrenza del carattere c nella stringa s*/
/*Pre: s!= NULL*/
/* Post: restituisce il puntatore alla prima occorrenza di c nella stringa.
Se tale occorrenza non esiste, restituisce NULL*/

{if (*s =='\0')
        return (NULL);
    else if (*s == c)
        return s;
    else
        return (Rstringchr(s+1,c));
}
```

Scelta dei dati a scatola nera
Dalle postcondizioni si hanno due casi:
1. la funzione restituisce il puntatore NULL quando il carattere non occorre.
Per esempio con
s = “stampa” e c = ‘r’, come caso limite prendiamo una stringa di lunghezza 1
diverso dal carattere cercato e la stringa vuota: s = 'b' e s=”\0” e c = 'a'

2. la funzione restituisce un puntatore non NULL
quando il carattere occorre nella stringa:
s = “stampa” e c = ‘m’, come caso limite prendiamo una stringa di lunghezza 1
che contiene il carattere cercato. Per esempio s= "a"e c='a'

A scatola trasparente:
La condizione di uscita dal primo if è
già soddisfatta dalla stringa vuota scelta precedentemente,
così come anche l’uscita dal primo if senza chiamate
(una stringa di lunghezza 1 che contiene il carattere cercato).
Verifichiamo se con una stringa di lunghezza
almeno due troviamo c nelle due posizioni possibili,
per esempio con s= “ai” e c =’a’ e poi c=’i’.
Nel primo caso non ci sono altre chiamate e nel secondo una sola .
Passiamo a stringhe di lunghezza 3 e prevediamo
una scelta di c per ogni posizione, per esempio:
s= “ape” e c=’a’, c=’p’ e c = ‘e’.
Qui nel secondo e terzo caso abbiamo una rispettivamente due chiamate.
Infine testiamo su una o più stringhe “lunghe”, sempre tutti i possibili casi di c,
Per esempio
s = “12%&()?mn<-]”

Esercizio 6
Si individuino opportuni dati di test per la seguente funzione,
evidenziando per quali dati si e’ seguito l’approccio a scatola
nera e per quali quello a scatola trasparente.
Si dia una breve motivazione della scelta.

```
MinMass* MinMax (int* vett, int i, int j)
/* prec 0<=i<=j<=num. el. vettore input && vett!=NULL;
postc: restituisce un puntatore a una struttura i cui campi
interi contengono il minimo e il massimo
* sugli elementi di vett tra vett[i] e vett[j], compresi*/
{MinMass *temp,*Mml,*Mmr;
int m;
temp =(MinMass*)malloc(sizeof(MinMass));
assert(temp);
assert(vett)
if(i == j) {temp -> min = vett[i];temp -> max = vett[i];return temp;}
m = (j+i)/2;
Mml  = MinMax(vett,i,m);
Mmr  = MinMax(vett,m+1,j);
temp -> max = max( Mml -> max, Mmr -> max);
temp -> min = min( Mml -> min, Mmr -> min);
return temp;}



A scatola nera
vettore con un solo elemento,
con due elementi uguali,
e due elementi diversi in ordine crescente e in quello ordine inverso.
vettore di tre elementi tutti uguali, diversi in ordine crescente,
in ordine inverso, con il minimo  e il massimo tute le posizioni.


a scatola trasparente
verifica precondizioni trattate nel codice: vett = NULL.
nessuna chiamata, gia considerata co il vettore di un elemento
per avere una sola chiamata sulle due meta vettore dobbiamo considerare
vettori con 2 elementi, gia considerato
per avere una  chiamata con uscita immediata e due chiamate sulla seconda meta
vettore dobbiamo considerare 3 elementi,
per avere due chiamate su ciascuna meta vettore dobbiamo considerare almeno 4
elementi.
Con vettori di lunghezza 8  si verifica il caso di 3 chiamate per ogni meta vettore,
considerando che dalla terza si esce immediatamente.
In tutti i casi con il mimimo e il massimo in tutte le posizioni possibili.
Facciamo notare che questa scelta di dati e minimale e puo essere utile per un
controllo manuale.
Il testing organizzato al calcolatore puo prevedere altri casi di dimensioni maggiori
e soprattuto
con scelte di dati memorizzati in modo da consentire la ripetibilita del test a codic
e modificato.
```