# Programmazione dei Calcolatori con Laboratorio

Esame del 17 giugno 2013 per informatici

Una sequenza di interi `a` si dice *ruotata* se questa può essere divisa in due sotto-sequenze, eventualmente vuote, $\mathtt{a}'$ e $\mathtt{a}''$ tali che $\mathtt{a} = \mathtt{a}'\mathtt{a}''$ e la sequenza ottenuta invertendo le posizioni di $\mathtt{a}'$ e $\mathtt{a}''$, ovvero $\mathtt{a}''\mathtt{a}'$, è ordinata in senso non decrescente.

Ad esempio $\mathtt{a} = [12, 14, 14, 15, 3, 3, 6, 8]$ è ruotata: basta prendere $\mathtt{a}' = [12, 14, 14, 15]$ e $\mathtt{a}'' = [3, 3, 6, 8]$.

Si implementi una funzione in C, denominata `Ruota()`, che prende in input una lista `a` che contiene una sequenza di interi e, nel caso in cui la sequenza sia ruotata, la riordina restituendo la lista modificata, altrimenti restituisce la lista originale.

Si calcoli il costo della funzione sia in termini di tempo che di memoria supplementare utilizzata.

Nel risolvere il problema si tenga conto che una lista è un puntatore a `nodo` (il primo nodo della lista) e che il tipo `nodo` è definito come segue.

```
struct nodo {
    int inf;
    struct nodo *succ;
    struct nodo *prec;
};
typedef struct nodo nodo;
```

Si ricorda che il campo `inf` contiene il dato del nodo e i campi `succ` e `prec` fanno riferimento rispettivamente al nodo successivo e a quello precedente.

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`Ruota()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle strutture dati e tipi eventualmente utilizzati (compreso il `nodo` ) e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

# Programmazione dei Calcolatori con Laboratorio

Esame del 9 luglio 2013 per informatici

La *porzione* di una lista `a` è un segmento della lista composta da elementi contigui di `a`. Si vuole implementare una funzione denominata `Porzione()` che prende in input la lista `a` e due interi `i` e `j`. Questa deve restituire una copia della porzione di `a` racchiusa tra l'elemento in posizione `i` e quello in posizione `j` escluso (si adotta la convenzione che il primo elemento della lista è in posizione `0`). Inoltre, se `j` = `0` o se `j` supera la lunghezza della lista si deve restituire la porzione di lista che va da `i` in poi. Infine se `i` supera la lunghezza della lista oppure se `i` $\geq$ `j` si deve restituire la lista vuota.

Ad esempio se `a` = [12, 14, 14, 15, 3, 3, 6, 8], `Porzione(a, 1, 4)` deve restituire la nuova lista [14, 14, 15] mentre `Porzione(a, 0, 1)` deve restituire la lista [12]. Infine `Porzione(a, 5, 0)` deve restituire [3, 6, 8].

Si implementi in C la funzione `Porzione()`, che prende in input la lista `a` e i due interi `i` e `j` e restituisce la copia della porzione di `a` individuata dai due interi.

Si calcoli il costo della funzione sia in termini di tempo che di memoria supplementare utilizzata.

Nel risolvere il problema si tenga conto che una lista è un puntatore a `nodo` (il primo nodo della lista) e che il tipo `nodo` è definito come segue.

```
struct nodo {
    int inf;
    struct nodo *succ;
    struct nodo *prec;
};
typedef struct nodo nodo;
```

Si ricorda che il campo `inf` contiene il dato del nodo e i campi `succ` e `prec` fanno riferimento rispettivamente al nodo successivo e a quello precedente.

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`Porzione()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle strutture dati e tipi eventualmente utilizzati (compreso il `nodo` ) e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

# Programmazione dei Calcolatori con Laboratorio

Esame del 23 settembre 2013 per informatici

Si vuole implementare una funzione che crei una lista contenente gli interi di una progressione aritmetica generata da tre interi (`i`, `j`, `k`): l'intero `i` identifica il primo elemento della progressione; l'intero `j` identifica il termine della progressione e l'intero `k` specifica la differenza tra due elementi consecutivi della progressione.

Se $k > 0$, la progressione aritmetica generata da (`i`, `j`, `k`) è $i, i + k, i + 2k, ..., i + tk$ dove `t` è il massimo intero per cui $i + tk < j$. Altrimenti, se $k < 0$ la progressione aritmetica generata da (`i`, `j`, `k`) è $i, i + k, i + 2k, ..., i + tk$ dove `t` è il massimo intero per cui $i + tk > j$. Se $k = 0$ la progressione aritmetica è vuota.

Ad esempio se $i = 1, j = 10, k = 2$ la progressione aritmetica è $[1, 3, 5, 7, 9]$; se $i = 1, j = 1, k = 1$ la progressione aritmetica è vuota; se $i = 2, j = -3, k = -1$ la progressione aritmetica è $[2, 1, 0, -1, -2]$.

Si implementi in C una funzione denominata `Range()`, che prende in input una lista `a` e tre interi `i`, `j` e `k` e restituisce la lista `a` contenente la progressione aritmetica generata dalla tripla (`i`, `j`, `k`).

Si calcoli il costo della funzione sia in termini di tempo che di memoria supplementare utilizzata.

Nel risolvere il problema si tenga conto che una lista è un puntatore a `nodo` (il primo nodo della lista) e che il tipo `nodo` è definito come segue.

```
struct nodo {
    int inf;
    struct nodo *succ;
    struct nodo *prec;
};
typedef struct nodo nodo;
```

Si ricorda che il campo `inf` contiene il dato del nodo e i campi `succ` e `prec` fanno riferimento rispettivamente al nodo successivo e a quello precedente.

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`Range()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle strutture dati e tipi eventualmente utilizzati (compreso il `nodo` ) e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

# Programmazione dei Calcolatori con Laboratorio

Esame del 3 febbraio 2014

Sia $\mathtt{a} = \mathtt{a}_0\mathtt{a}_1 \ldots \mathtt{a}_{n-1}$ una lista e $\mathtt{i}, \mathtt{j}, \mathtt{k}$ tre interi: si vuole progettare una funzione che, nel caso $0 \leq \mathtt{i} < \mathtt{j} < \mathtt{k} \leq \mathtt{n}$, modifica la lista $\mathtt{a}$ trasformandola scambiando le posizioni delle sotto-liste $\mathtt{a}_\mathtt{i}\mathtt{a}_{\mathtt{i}+1} \ldots \mathtt{a}_{\mathtt{j}-1}$ e $\mathtt{a}_\mathtt{j}\mathtt{a}_{\mathtt{j}+1} \ldots \mathtt{a}_{\mathtt{k}-1}$, ovvero deve restituire

$$\begin{cases} \mathtt{a}_0 \ldots \mathtt{a}_{\mathtt{i}-1}\mathtt{a}_\mathtt{j}\mathtt{a}_{\mathtt{j}+1} \ldots \mathtt{a}_{\mathtt{k}-1}\mathtt{a}_\mathtt{i}\mathtt{a}_{\mathtt{i}+1} \ldots \mathtt{a}_{\mathtt{j}-1}\mathtt{a}_\mathtt{k} \ldots \mathtt{a}_{\mathtt{n}-1} & \text{se } 0 < \mathtt{i} < \mathtt{j} < \mathtt{k} < \mathtt{n} \\ \mathtt{a}_\mathtt{j}\mathtt{a}_{\mathtt{j}+1} \ldots \mathtt{a}_{\mathtt{k}-1}\mathtt{a}_\mathtt{i}\mathtt{a}_{\mathtt{i}+1} \ldots \mathtt{a}_{\mathtt{j}-1}\mathtt{a}_\mathtt{k} \ldots \mathtt{a}_{\mathtt{n}-1} & \text{se } 0 = \mathtt{i} < \mathtt{j} < \mathtt{k} < \mathtt{n} \\ \mathtt{a}_0 \ldots \mathtt{a}_{\mathtt{i}-1}\mathtt{a}_\mathtt{j}\mathtt{a}_{\mathtt{j}+1} \ldots \mathtt{a}_{\mathtt{k}-1}\mathtt{a}_\mathtt{i}\mathtt{a}_{\mathtt{i}+1} \ldots \mathtt{a}_{\mathtt{j}-1} & \text{se } 0 < \mathtt{i} < \mathtt{j} < \mathtt{k} = \mathtt{n} \\ \mathtt{a}_\mathtt{j}\mathtt{a}_{\mathtt{j}+1} \ldots \mathtt{a}_{\mathtt{k}-1}\mathtt{a}_\mathtt{i}\mathtt{a}_{\mathtt{i}+1} \ldots \mathtt{a}_{\mathtt{j}-1} & \text{se } 0 = \mathtt{i} < \mathtt{j} < \mathtt{k} = \mathtt{n} \\ \mathtt{a} & \text{altrimenti.} \end{cases}$$

Si implementi in C una funzione denominata `ScambiaSottoliste()`, che prende in input una lista `a` e tre interi `i`, `j` e `k` e restituisce la lista `a` modificata come spiegato in precedenza. La funzione non deve creare nuovi nodi ma deve utilizzare quelli della lista di input.

Si calcoli il costo della funzione sia in termini di tempo che di memoria supplementare utilizzata.

Nel risolvere il problema si tenga conto che una lista è un puntatore a `nodo` (il primo nodo della lista) e che il tipo `nodo` è definito come segue.

```
struct nodo {
    char info;
    struct nodo *succ;
    struct nodo *prec;
};
typedef struct nodo nodo;
```

Si ricorda che il campo `info` contiene il dato del nodo e i campi `succ` e `prec` fanno riferimento rispettivamente al nodo successivo e a quello precedente.

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`ScambiaSottoliste()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle strutture dati e tipi eventualmente utilizzati (compreso il tipo `nodo` ) e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

# Programmazione dei Calcolatori con Laboratorio

Esame del 10 giugno 2014

Si progetti una funzione in C che, data una lista `a` di punti del piano cartesiano, restituisce la sottolista di `a` contente solo i punti contenuti all'interno del quadrato più piccolo con centro in $(0, 0)$ contenente il primo punto della lista `a`.

Si implementi in C una funzione denominata `InQuadrato()`, che prende in input la lista di punti `a` e restituisce la lista `a` modificata come spiegato in precedenza. La funzione non deve creare nuovi nodi ma deve utilizzare quelli della lista di input ed inoltre deve essere preservato l'ordinamento dei nodi rispetto alla lista originale.

Si calcoli il costo della funzione sia in termini di tempo che di memoria supplementare utilizzata.

Nel risolvere il problema si tenga conto che un punto è descritto nel seguente modo

```
struct punto {
 float x, y;
};
typedef struct punto punto;
```

Inoltre una lista è un puntatore a `nodo` (il primo nodo della lista) e che il tipo `nodo` è definito come segue.

```
struct nodo {
    punto info;
    struct nodo *succ;
    struct nodo *prec;
};
typedef struct nodo nodo;
```

Si ricorda che il campo `info` contiene il dato del nodo e i campi `succ` e `prec` fanno riferimento rispettivamente al nodo successivo e a quello precedente.

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`InQuadrato()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle strutture dati e tipi eventualmente utilizzati (compresi i tipi `punto` e `nodo` ) e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

# Programmazione dei Calcolatori con Laboratorio

Esame del 2 luglio 2014

Si progetti una funzione in C che dati tre interi `x`, `y` e `r` restituisce la lista di punti del piano cartesiano a coordinate intere contenuti nel cerchio di raggio `r` e centro $(\texttt{x}, \texttt{y})$. I punti nella lista devono essere ordinati per ordinate crescenti e poi per ascisse crescenti.

Nel risolvere il problema si tenga conto che un punto è descritto nel seguente modo

```
struct punto {
 float x, y;
};
typedef struct punto punto;
```

Inoltre una lista è un puntatore a `nodo` (il primo nodo della lista) e che il tipo `nodo` è definito come segue.

```
struct nodo {
    punto info;
    struct nodo *succ;
    struct nodo *prec;
};
typedef struct nodo nodo;
```

Si ricorda che il campo `info` contiene il dato del nodo e i campi `succ` e `prec` fanno riferimento rispettivamente al nodo successivo e a quello precedente.

La funzione da implementare deve essere denominata `InCerchio()`; prende come parametri di input, nell'ordine, `x`, `y` e `r` (ascissa e ordinata del centro del cerchio ed il suo raggio); e restituisce il puntatore al primo nodo della lista creata.

Si calcoli il costo della funzione sia in termini di tempo che di memoria supplementare utilizzata.

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`InCerchio()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle strutture dati e tipi eventualmente utilizzati (compresi i tipi `punto` e `nodo` ) e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione; non è possibile utilizzare funzioni non definite all'interno del file consegnato.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

# Programmazione dei Calcolatori con Laboratorio

Esame del 16 settembre 2014

Due stringhe sono *quasi uguali* se queste differiscono soltanto perché a qualche carattere alfabetico che in una stringa è minuscolo corrisponde, nella stessa posizione dell'altra stringa, lo stesso carattere alfabetico maiuscolo. Ad esempio le stringhe `pRograMma` e `progrAmmA` e le stringhe `Apollo-11` e `apollo-11` sono quasi uguali.

Si progetti una funzione in C che data una lista di stringhe `a` ed una stringa `s` restituisce il numero di stringhe in `a` che sono quasi uguali ad `s`.

Nel risolvere il problema si tenga conto che una lista è un puntatore a `nodo` (il primo nodo della lista) e che il tipo `nodo` è definito come segue.

```
struct nodo {
    char *info;
    struct nodo *succ;
    struct nodo *prec;
};
typedef struct nodo nodo;
```

Si ricorda che il campo `info` contiene il dato del nodo e i campi `succ` e `prec` fanno riferimento rispettivamente al nodo successivo e a quello precedente nella lista.

La funzione da implementare deve essere denominata `ContaRipetizioniIgnoraMaiuscolo()`; deve prendere come parametri di input, nell'ordine, `a`, e `s` (la lista e la stringa); deve restituire l'intero che contiene il valore cercato.

Si calcoli il costo della funzione sia in termini di tempo che di memoria supplementare utilizzata.

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`ContaRipetizioniIgnoraMaiuscolo()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle strutture dati e tipi eventualmente utilizzati (compreso il tipo `nodo` ) e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione; non è possibile utilizzare funzioni non definite all'interno del file consegnato.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

# Programmazione dei Calcolatori con Laboratorio

Esame del 25 febbraio 2015

Si progetti una funzione in C che restituisce la lista di parole che compongono una stringa data in input. L'ordine con cui le parole compaiono nella lista deve ricalcare quello sulla stringa.

La funzione, denominata `StringSplit()`, deve avere il seguente prototipo:

```
nodo *StringSplit(char a[]);
```

dove:

- `a[]` rappresenta la stringa in input;
- il puntatore restituito è l'indirizzo al primo elemento della lista creata. Ogni nodo della lista di output è di tipo `nodo` che è definito come segue

```
struct nodo {
    char *info;
    struct nodo *succ;
    struct nodo *prec;
};
typedef struct nodo nodo;
```

La funzione deve utilizzare soltanto i dati appena descritti pertanto non è possibile fare uso di variabili globali.

Dopo aver progettato ed implementato la funzione, se ne calcoli il costo sia in termini di tempo che di memoria supplementare utilizzata (al netto di quella utilizzata per memorizzare l'input e l'output).

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`StringSplit()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle strutture dati e dei tipi eventualmente utilizzati (compresi `struct nodo` e `nodo`) e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

# Programmazione dei Calcolatori con Laboratorio

Esame del 10 giugno 2015

Si vuole una funzione che elimina tutti i nodi di una lista contenenti una data chiave. Quella di input è una lista concatenata composta da nodi così descritti

```
struct nodo {
    char *chiave;
    struct nodo *prec;
    struct nodo *succ;
};
typedef struct nodo nodo;
```

quindi ogni nodo contiene un riferimento a quello che lo segue ed a quello che lo precede. Il primo nodo ha il campo `prec` uguale a `NULL` mentre l'ultimo ha il campo `succ` uguale a `NULL`. La chiave contenuta in ogni nodo è una stringa. Una lista è identificata con il puntatore al suo primo nodo.

La funzione, denominata `CancellaTutto()`, deve avere il seguente prototipo:

```
nodo *CancellaTutto(nodo *a, char *k);
```

dove `a` è la lista da cui bisogna eliminare tutti i nodi che contengono, nel campo `chiave`, una stringa uguale a `k`. La funzione restituisce la lista modificata.

La funzione deve utilizzare soltanto i dati appena descritti pertanto non è possibile fare uso di variabili globali. Non si possono alterare le posizioni dei nodi residui e non possono essere creati nuovi nodi.

Dopo aver progettato ed implementato la funzione, se ne calcoli il costo sia in termini di tempo che di memoria supplementare utilizzata (al netto di quella utilizzata per memorizzare l'input e l'output).

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`CancellaTutto()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle strutture dati e dei tipi eventualmente utilizzati (compresi `struct nodo` e `nodo`) e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

# Programmazione dei Calcolatori con Laboratorio

Esame del 25 giugno 2015

Si vuole una funzione che da una lista estrapoli la sotto-lista delimitata da due nodi individuati attraverso la loro chiave. Quella di input è una lista concatenata composta da nodi così descritti

```
struct nodo {
    char *chiave;
    struct nodo *prec;
    struct nodo *succ;
};
typedef struct nodo nodo;
```

quindi ogni nodo contiene un riferimento a quello che lo segue ed a quello che lo precede. Il primo nodo ha il campo `prec` uguale a `NULL` mentre l'ultimo ha il campo `succ` uguale a `NULL`. La chiave contenuta in ogni nodo è una stringa. Una lista è identificata con il puntatore al suo primo nodo.

La funzione, denominata `EstraiSegmento()`, deve avere il seguente prototipo:

```
nodo *EstraiSegmento(nodo *a, char *k1, char *k2);
```

dove `a` è una lista e `k1` e `k2` sono le due chiavi utilizzate per descrivere la sottolista di `a` da estrarre. La funzione deve restituire la sottosequenza di `a` composta dai nodi compresi tra il primo nodo, `n1`, contenente la chiave `k1` ed il primo nodo, `n2`, contenente la chiave `k2`. Il nodo `n2` non deve precedere `n1`. I nodi estremi sono compresi, ovvero `n1` ed `n2` devo far parte della sotto-sequenza di output. Tutti gli altri nodi di `a` non compresi nella nuova lista devono essere eliminati. Se uno dei due nodi (ovvero `n1` o `n2`) non esiste deve essere restituita la lista iniziale non modificata.

La funzione deve utilizzare soltanto i dati appena descritti pertanto non è possibile fare uso di variabili globali e deve essere utilizzata una quantità costante di memoria supplementare.

Dopo aver progettato ed implementato la funzione calcolarne il costo in termini di tempo e dire perché la funzione utilizza una quantità di memoria costante (al netto di input ed output). Inserire questo considerazioni come commento nel file sorgente che verrà consegnato.

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`EstraiSegmento()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle strutture dati e dei tipi eventualmente utilizzati (compresi `struct nodo` e `nodo`);
- ogni altra funzione utilizzata dalla soluzione;
- gli *header* delle librerie utilizzate.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

# Programmazione dei Calcolatori con Laboratorio

Esame del 23 settembre 2015

Si vuole una funzione che divide la lista di input in due parti della stessa lunghezza. In particolare se la lista di input `a` è composta dagli `n` elementi $\texttt{x}_0, \texttt{x}_1, \ldots, \texttt{x}_{n-1}$, la funzione deve restituire la sottolista composta dei primi $\lceil \texttt{n}/2 \rceil$ elementi di `a` ovvero $\texttt{x}_0, \texttt{x}_1, \ldots, \texttt{x}_{\lceil n/2 \rceil - 1}$ e la sottolista $\texttt{x}_{\lceil n/2 \rceil}, \texttt{x}_{\lceil n/2 \rceil + 1}, \ldots, \texttt{x}_{n-1}$ composta dai restanti elementi di `a`. Quella di input è una lista concatenata composta da nodi così descritti

```
struct nodo {
    char *chiave;
    struct nodo *prec;
    struct nodo *succ;
};
typedef struct nodo nodo;
```

quindi ogni nodo contiene un riferimento a quello che lo segue ed a quello che lo precede. Il primo nodo ha il campo `prec` uguale a `NULL` mentre l'ultimo ha il campo `succ` uguale a `NULL`. La chiave contenuta in ogni nodo è una stringa. Una lista è identificata con il puntatore al suo primo nodo.

La coppia di sottoliste di output deve essere memorizzata nella seguente struttura dati:

```
struct coppialiste {
    nodo *testa;
    nodo *coda;
};
typedef struct coppialiste coppialiste;
```

ovvero il campo `testa` dovrà contenere il riferimento alla prima sotto-lista ed il campo `coda` alla seconda sotto-lista. Quindi la funzione, denominata `SpezzaLista()`, deve avere il seguente prototipo:

```
coppialiste SpezzaLista(nodo *a);
```

dove `a` è la lista di input.

La funzione deve utilizzare soltanto i dati appena descritti pertanto non è possibile fare uso di variabili globali. Dopo aver progettato ed implementato la funzione, se ne calcoli il costo sia in termini di tempo che di memoria supplementare utilizzata (al netto di quella utilizzata per memorizzare l'input e l'output).

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`SpezzaLista()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle strutture dati e dei tipi eventualmente utilizzati (compresi `struct nodo`, `nodo`, `struct coppialiste` e `coppialiste`);
- ogni altra funzione utilizzata dalla soluzione;
- gli *header* delle librerie utilizzate.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

# Programmazione dei Calcolatori con Laboratorio

Esame del 18 febbraio 2016

Sia `a` una sequenza ordinata, si vuole progettare una funzione che localizzi il punto di inserimento di un nuovo elemento `x` nella sequenza mantenendola ordinata. Ovvero, qualora sia possibile aggiungere un nuovo elemento `x` in `a`, qual è la posizione in cui dovrebbe essere aggiunto in modo tale che la nuova eventuale sequenza risulti ancora ordinata? Se un elemento uguale a `x` è già presente nella sequenza il nuovo punto di inserimento deve essere quello più a sinistra, ovvero quello precedente a tutti gli elementi uguali a `x`.

La sequenza in questione contiene date ordinate in modo non decrescente, ovvero stringhe della forma `GGmese`, dove `GG` indica il giorno con due caratteri *numerici* e `mese` è una stringa composta da uno, due, tre o quattro caratteri ed indica il mese nella numerazione romana (`I`, `II`,..., `VIII`, ..., `XI`, `XII`).

Esempi: se `a` = (`02II`, `18II`, `18II`, `25XII`) e `x` = `01I`, la funzione deve restituire `0`; se `x` = `18II` la funzione deve restituire `1`; se `x` = `31XII` la funzione deve restituire `4`.

La funzione deve avere il seguente prototipo:

```
int Bisect( char *a[], int n,  char *x )
```

Ovvero la sequenza di date è memorizzata all'interno di un array di stringhe; `n` è la dimensione dell'array e `x` è la data di cui si vuole conoscere la posizione.

Come pre-condizione si assuma che le date `x` e quelle in `a` siano corrette (ovvero tali stringhe codificano date corrette rispettando il formato descritto) e che in `a` compaiano effettivamente in ordine non decrescente.

La funzione deve utilizzare soltanto i dati appena descritti pertanto non è possibile fare uso di variabili globali.

Dopo aver progettato ed implementato la funzione, se ne calcoli il costo sia in termini di tempo che di memoria supplementare utilizzata (al netto di quella utilizzata per memorizzare l'input e l'output).

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`Bisect()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle strutture dati e dei tipi eventualmente utilizzati;
- ogni altra funzione utilizzata dalla soluzione;
- gli *header* delle librerie utilizzate.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

# Programmazione dei Calcolatori con Laboratorio

Esame del 18 febbraio 2016

`L` è una lista di interi, progettare ed implementare una funzione nel linguaggio C che sostituisce il contenuto informativo di ogni nodo di `L` (dal terzo in poi) con la somma del contenuto dei due nodi che lo precedono. Il contenuto del primo e del secondo nodo resta invariato.

Ad esempio, se $\texttt{L} = (4, 1, 6, 9, 2, 8, 3)$ allora, dopo l'esecuzione della funzione, `L` diventa $(4, 1, 5, 7, 15, 11, 10)$.

Una lista è definita come una successione di nodi che, a loro volta, sono definiti come segue

```
struct nodo {
    char *chiave;
    struct nodo *succ, *prec;
};
typedef struct nodo nodo;
```

dove ogni nodo è collegato al nodo che lo segue (attraverso il puntatore `succ`) ed a quello che lo precede (con il puntatore `prec`). Il primo (rispettivamente, l'ultimo) nodo della lista hanno i campi `prec` (rispettivamente, `succ`) uguali a `NULL`. La lista è identificata attraverso il puntatore al primo nodo della stessa.

La funzione deve avere il seguente prototipo:

```
int PrecSum( nodo *L )
```

La funzione deve utilizzare soltanto i dati appena descritti pertanto non è possibile fare uso di variabili globali.

Dopo aver progettato ed implementato la funzione, se ne calcoli il costo sia in termini di tempo che di memoria supplementare utilizzata (al netto di quella utilizzata per memorizzare l'input e l'output).

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`PrecSum()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle strutture dati e dei tipi eventualmente utilizzati (compresi `struct nodo` e `nodo`);
- ogni altra funzione utilizzata dalla soluzione;
- gli *header* delle librerie utilizzate.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

# Programmazione dei Calcolatori con Laboratorio

Esame del 18 febbraio 2016

`L` è una lista di interi, progettare ed implementare una funzione nel linguaggio C che sostituisce il contenuto informativo di ogni nodo di `L` (fino al terzultimo) con la somma del contenuto dei due nodi che lo seguono. Il contenuto degli ultimi due nodi resta invariato.

Ad esempio, se $\texttt{L} = (4, 1, 6, 9, 2, 8, 3)$ allora, dopo l'esecuzione della funzione, `L` diventa $(7, 15, 11, 10, 11, 8, 3)$.

Una lista è definita come una successione di nodi che, a loro volta, sono definiti come segue

```
struct nodo {
    int chiave;
    struct nodo *succ, *prec;
};
typedef struct nodo nodo;
```

dove ogni nodo è collegato al nodo che lo segue (attraverso il puntatore `succ`) ed a quello che lo precede (con il puntatore `prec`). Il primo (rispettivamente, l'ultimo) nodo della lista hanno i campi `prec` (rispettivamente, `succ`) uguali a `NULL`. La lista è identificata attraverso il puntatore al primo nodo della stessa.

La funzione deve avere il seguente prototipo:

```
nodo *SuccSum( nodo *L )
```

La funzione deve utilizzare soltanto i dati appena descritti pertanto non è possibile fare uso di variabili globali.

Dopo aver progettato ed implementato la funzione, se ne calcoli il costo sia in termini di tempo che di memoria supplementare utilizzata (al netto di quella utilizzata per memorizzare l'input e l'output).

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`SuccSum()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle strutture dati e dei tipi eventualmente utilizzati (compresi `struct nodo` e `nodo`);
- ogni altra funzione utilizzata dalla soluzione;
- gli *header* delle librerie utilizzate.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

# Programmazione dei Calcolatori con Laboratorio

Esame del 20 luglio 2016

La stringa *più popolare* tra quelle in una lista `L` di stringhe ordinate lessicograficamente è quella che compare il maggior numero di volte. In caso di parità quella più popolare è quella più in fondo nella lista.

Progettare ed implementare una funzione nel linguaggio C che restituisce il puntatore al nodo contenente la prima occorrenza della stringa più popolare. Se la lista è vuota la funzione deve restituire `NULL`.

Ad esempio, se `L` = (`aaa`, `aaa`, `aaa`, `bb`, `bb`, `ccc`, `ccc`, `ccc`, `ddd`, `ddd`) la stringa più popolare è `"ccc"` quindi la funzione deve restituire il puntatore al sesto nodo di `L`.

Una lista è definita come una successione di nodi che, a loro volta, sono definiti come segue

```
struct nodo {
    char *chiave;
    struct nodo *succ, *prec;
};
typedef struct nodo nodo;
```

dove ogni nodo è collegato al nodo che lo segue (attraverso il puntatore `succ`) ed a quello che lo precede (con il puntatore `prec`). Il primo (rispettivamente, l'ultimo) nodo della lista hanno i campi `prec` (rispettivamente, `succ`) uguali a `NULL`. La lista è identificata attraverso il puntatore al primo nodo della stessa.

La funzione deve avere il seguente prototipo:

```
nodo *PiuPopolare( nodo* L );
```

Come pre-condizione si assuma che la lista `L` sia ordinata lessicograficamente in modo non decrescente.

La funzione deve utilizzare soltanto i dati appena descritti pertanto non è possibile fare uso di variabili globali.

Dopo aver progettato ed implementato la funzione, se ne calcoli il costo sia in termini di tempo che di memoria supplementare utilizzata (al netto di quella utilizzata per memorizzare l'input e l'output).

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`PiuPopolare()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle strutture dati e dei tipi eventualmente utilizzati (compresi `struct nodo` e `nodo`);
- ogni altra funzione utilizzata dalla soluzione;
- gli *header* delle librerie utilizzate.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

# Programmazione dei Calcolatori con Laboratorio

20 luglio 2016

La stringa *più popolare* tra quelle in una lista `L` di stringhe ordinate lessicograficamente è quella che compare il maggior numero di volte. In caso di parità quella più popolare è quella che precede le altre nella lista.

Progettare ed implementare una funzione nel linguaggio C che restituisce il puntatore al nodo contenente l'ultima occorrenza della stringa più popolare. Se la lista è vuota la funzione deve restituire `NULL`.

Ad esempio, se `L` = (`aaa`, `aaa`, `bb`, `bb`, `bb`, `ccc`, `ccc`, `ccc`, `ddd`, `ddd`) la stringa più popolare è `"bb"` quindi la funzione deve restituire il puntatore al quinto nodo di `L`.

Una lista è definita come una successione di nodi che, a loro volta, sono definiti come segue

```
struct nodo {
char *chiave;
struct nodo *succ, *prec;
};
typedef struct nodo nodo;
```

dove ogni nodo è collegato al nodo che lo segue (attraverso il puntatore `succ`) ed a quello che lo precede (con il puntatore `prec`). Il primo (rispettivamente, l'ultimo) nodo della lista hanno i campi `prec` (rispettivamente, `succ`) uguali a `NULL`. La lista è identificata attraverso il puntatore al primo nodo della stessa.

La funzione deve avere il seguente prototipo:

```
nodo *PiuPopolare( nodo* L );
```

Come pre-condizione si assuma che la lista `L` sia ordinata lessicograficamente in modo non decrescente.

La funzione deve utilizzare soltanto i dati appena descritti pertanto non è possibile fare uso di variabili globali.

Dopo aver progettato ed implementato la funzione, se ne calcoli il costo sia in termini di tempo che di memoria supplementare utilizzata (al netto di quella utilizzata per memorizzare l'input e l'output).

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`PiuPopolare()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle strutture dati e dei tipi eventualmente utilizzati (compresi `struct nodo` e `nodo`);
- ogni altra funzione utilizzata dalla soluzione;
- gli *header* delle librerie utilizzate.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

# Programmazione dei Calcolatori con Laboratorio

Esame del 22 settembre 2016 (A)

L'array `a` è composto da `n` stringhe che indicano date nel formato `dd-mm-yyyy` dove `d`, `m` e `y` sono `char` da `'0'` a `'9'`. Progettare ed implementare nel linguaggio C una funzione che, dato `a`, restituisce un nuovo array composto dalle stringhe di `a` ordinate per mese in modo non decrescente. Ad esempio se `a` contiene, nell'ordine, le stringhe

`27-09-2016`, `11-08-1972`, `31-07-1955`, `22-09-2016` e `01-01-2017`,

la funzione deve restituire un nuovo array contenente, nell'ordine, le stringhe

`01-01-2017`, `31-07-1955`, `11-08-1972`, `22-09-2016` e `27-09-2016`.

*L'ordine in cui compaiono le ultime due stringhe è indifferente.*

La funzione deve utilizzare l'algoritmo di ordinamento Bubble-sort e non deve modificare l'array `a` in input. Il suo prototipo deve essere il seguente:

```
char **OrdinaPerMeseBS(char *a[], int n);
```

Dove `a` rappresenta l'array di input ed `n` la sua dimensione. La funzione deve restituire un array di stringhe ovvero un puntatore a `*char`. Come pre-condizione si assuma che le `n` stringhe in `a` siano 'ben formate' ovvero rispettino il formato `dd-mm-yyyy`.

La funzione deve utilizzare soltanto i dati appena descritti pertanto non è possibile fare uso di variabili globali.

Dopo aver progettato ed implementato la funzione, se ne calcoli il costo sia in termini di tempo che di memoria supplementare utilizzata (al netto di quella utilizzata per memorizzare l'input e l'output).

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`OrdinaPerMeseBS()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle strutture dati e dei tipi eventualmente utilizzati (compresi `struct nodo` e `nodo`);
- ogni altra funzione utilizzata dalla soluzione;
- gli *header* delle librerie utilizzate.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

# Programmazione dei Calcolatori con Laboratorio

Esame del 22 settembre 2016 (B)

L'array `a` è composto da `n` stringhe che indicano date nel formato `dd-mm-yyyy` dove `d`, `m` e `y` sono `char` da `'0'` a `'9'`. Progettare ed implementare nel linguaggio C una funzione che, dato `a`, restituisce un nuovo array composto dalle stringhe di `a` ordinate per mese in modo non decrescente. Ad esempio se `a` contiene, nell'ordine, le stringhe

`27-09-2016`, `11-08-1972`, `31-07-1955`, `22-09-2016` e `01-01-2017`,

la funzione deve restituire un nuovo array contenente, nell'ordine, le stringhe

`01-01-2017`, `31-07-1955`, `11-08-1972`, `22-09-2016` e `27-09-2016`.

*L'ordine in cui compaiono le ultime due stringhe è indifferente.*

La funzione deve utilizzare l'algoritmo di ordinamento Insertion-sort e non deve modificare l'array `a` in input. Il suo prototipo deve essere il seguente:

```
char **OrdinaPerMeseIS(char *a[], int n);
```

Dove `a` rappresenta l'array di input ed `n` la sua dimensione. La funzione deve restituire un array di stringhe ovvero un puntatore a `*char`. Come pre-condizione si assuma che le `n` stringhe in `a` siano 'ben formate' ovvero rispettino il formato `dd-mm-yyyy`.

La funzione deve utilizzare soltanto i dati appena descritti pertanto non è possibile fare uso di variabili globali.

Dopo aver progettato ed implementato la funzione, se ne calcoli il costo sia in termini di tempo che di memoria supplementare utilizzata (al netto di quella utilizzata per memorizzare l'input e l'output).

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`OrdinaPerMeseIS()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle strutture dati e dei tipi eventualmente utilizzati (compresi `struct nodo` e `nodo`);
- ogni altra funzione utilizzata dalla soluzione;
- gli *header* delle librerie utilizzate.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

# Programmazione dei Calcolatori con Laboratorio

Esame del 2 febbraio 2017 (A)

Progettare ed implementare nel linguaggio C una funzione che, dato un array `a` di `n` stringhe e la sua dimensione, restituisce un nuovo array di `n` elementi corrispondenti alle stringhe di `a`. Ognuno di questi elementi contiene una copia della stringa di `a` a cui fa riferimento ed il numero di caratteri non alfabetici di questa. La sequenza degli elementi nel nuovo array deve rispettare quella dell'array di input.

Gli elementi dell'array di output devono essere **struct** con due campi, uno per la stringa ed uno per l'intero. Ecco la definizione in C:

```
struct xstringa {
    char *s;
    int nonalpha;
};
typedef struct xstringa xstringa;
```

La funzione dovrà restituire un array di `xstringa`, una per ogni stringa in `a`. Il campo `s` delle `xstringa` dovrà contenere una copia della stringa di `a` ed il campo `nonalpha` il numero dei suoi caratteri non alfabetici.

La funzione deve avere il seguente prototipo:

```
xstringa *XstringaArray(char*[], int);
```

Dove il primo parametro è l'array di stringhe ed il secondo la sua dimensione.

La funzione deve utilizzare soltanto i dati appena descritti pertanto non è possibile fare uso di variabili globali.

Dopo aver progettato ed implementato la funzione, se ne calcoli il costo sia in termini di tempo che di memoria supplementare utilizzata (al netto di quella utilizzata per memorizzare l'input e l'output).

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`XstringaArray()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle strutture dati e dei tipi eventualmente utilizzati (compresi `struct xstringa` e `xstringa`);
- ogni altra funzione utilizzata dalla soluzione;
- gli *header* delle librerie utilizzate.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

# Programmazione dei Calcolatori con Laboratorio

Esame del 2 febbraio 2017 (B)

Progettare ed implementare nel linguaggio C una funzione che, dato un array `a` di `n` stringhe e la sua dimensione, restituisce una lista concatenata di `n` elementi corrispondenti alle stringhe di `a`. Ognuno di questi elementi dovrà contenere una copia della stringa di `a` a cui fa riferimento e la sua lunghezza. La sequenza degli elementi nella lista deve rispettare quella dell'array di input.

La lista è definita come una sequenza di nodi (uno per ogni stringa di `a`) che, a loro volta, sono definiti come segue

```
struct nodo {
    char *s;
    int len;
    struct nodo *succ, *prec;
};
typedef struct nodo nodo;
```

ogni nodo è collegato al nodo che lo segue (per mezzo del puntatore `succ`) e quello che lo precede (attraverso il puntatore `prec`). Il primo (rispettivamente, l'ultimo) nodo della lista ha il campo `prec` (rispettivamente, `succ`) uguale a `NULL`. Infine il campo `s` contiene una copia della stringa corrispondente nell'array di input e `len` la sua lunghezza. La lista è definita attraverso il puntatore al suo primo nodo.

La funzione da progettare dovrà avere il seguente prototipo:

```
nodo *XstringaLista(char*[], int);
```

Dove il primo parametro è l'array di stringhe ed il secondo la sua dimensione.

La funzione deve utilizzare soltanto i dati appena descritti pertanto non è possibile fare uso di variabili globali.

Dopo aver progettato ed implementato la funzione, se ne calcoli il costo sia in termini di tempo che di memoria supplementare utilizzata (al netto di quella utilizzata per memorizzare l'input e l'output).

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`XstringaLista()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle strutture dati e dei tipi eventualmente utilizzati (compresi `struct nodo` e `nodo`);
- ogni altra funzione utilizzata dalla soluzione;
- gli *header* delle librerie utilizzate.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

# Programmazione dei Calcolatori con Laboratorio

Simulazione esame 1 (marzo 2015)

Due stringhe sono *quasi uguali* se queste differiscono soltanto perché a qualche carattere alfabetico che in una stringa è minuscolo corrisponde, nella stessa posizione dell'altra stringa, lo stesso carattere alfabetico maiuscolo. Ad esempio le stringhe `pRograMma` e `progrAmmA` e le stringhe `Apollo-11` e `apollo-11` sono quasi uguali. Data una sequenza di stringhe ed una ulteriore stringa `s` si deve progettare e implementare in C un algoritmo che identifica le stringhe nella sequenza che risultano essere quasi uguali ad `s`.

In particolare si deve scrivere una funzione denominata `QuasiUguali()` avente il seguente prototipo:

```
int *QuasiUguali(char *a[], char *s, int n);
```

dove `a[]` è l'array contente le `n` stringhe e `s` è la stringa chiave. La funzione deve restituire un array `b` di `n` elementi in $\{0, 1\}$ dove `b[i]` vale `1` se e solo se `a[i]` è quasi uguale a `s`.

Si calcoli il costo della funzione sia in termini di tempo che di memoria supplementare utilizzata.

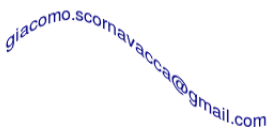
**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`QuasiUguali()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle strutture dati e tipi eventualmente utilizzati e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione in quanto non permesso l'utilizzo di funzioni non definite all'interno del file consegnato.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

**Inviare le soluzioni entro il 29/03 al seguente indirizzo**



# Programmazione dei Calcolatori con Laboratorio

Simulazione esame 2 (Pasqua 2015)

Si vuole creare una funzione che restituisce una copia di una porzione di una sequenza data in input. La sequenza è memorizzata in un array `a`, la funzione, oltre ad `a`, prende in input due interi `i` e `j`; questa deve restituire un nuovo array contenente una copia del sotto-array di `a` formato da tutti gli elementi compresi tra quello in posizione `i` e quello in posizione `j` escluso. Se `j` = `0` o se `j` supera la lunghezza della sequenza si deve restituire la porzione di sequenza che va da `i` in poi. Infine se `i` supera la lunghezza della sequenza oppure se `i` $\geq$ `j` si deve restituire la sequenza vuota.

Ad esempio con `a` = $[10, 2, 5, 9, 4, 7, 6]$

- se `i` = `1`, `j` = `4` la funzione deve restituire la sequenza $[2, 5, 9]$;
- se `i` = `3`, `j` = `0` la funzione deve restituire la sequenza $[9, 4, 5, 6]$
- se `i` = `0`, `j` = `21` la funzione deve restituire la sequenza $[10, 2, 5, 9, 4, 7, 6]$;
- se `i` = `1`, `j` = `1` la funzione deve restituire la sequenza vuota.

La funzione da implementare deve avere il seguente prototipo:

```
void **Porzione(void **a, int i, int j, int n);
```

dove `a`, `i` e `j` sono rispettivamente l'array e i due indici che individuano la porzione mentre `n` è la dimensione dell'array. Si noti che l'array in input ed output contiene puntatori a `void` per fare in modo che `a` possa contenere riferimenti ad elementi di tipo qualsiasi.

Si calcoli il costo della funzione sia in termini di tempo che di memoria supplementare utilizzata (al netto di quella necessaria per memorizzare input ed output).

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`Porzione()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle strutture dati e tipi eventualmente utilizzati e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione in quanto non permesso l'utilizzo di funzioni non definite all'interno del file consegnato.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

**Inviare le soluzioni entro il 12/04 al seguente indirizzo**

# Programmazione dei Calcolatori con Laboratorio

Simulazione esame 3 (Aprile)

Si vuole creare una funzione che aggiunge in fondo ad una sequenza data una copia della sequenza originale. Ad esempio se la sequenza è `a` = $[0, 1, 2, 3, 4]$ la funzione deve restituire la sequenza `a` = $[0, 1, 2, 3, 4, 0, 1, 2, 3, 4]$. La sequenza è memorizzata in una lista concatenata.

La funzione da implementare deve avere il seguente prototipo:

```
nodo *Raddoppia( nodo *a );
```

dove `a` è il puntatore al primo nodo della lista di input. La funzione deve restituire la lista (puntatore a nodo) modificata. Le chiavi nella lista sono di tipo `char` quindi `nodo` è così definito:

```
struct nodo {
    char chiave;
    struct nodo *prec;
    struct nodo *succ;
};
typedef struct nodo nodo;
```

Si calcoli il costo della funzione sia in termini di tempo che di memoria supplementare utilizzata (al netto di quella necessaria per memorizzare input ed output).

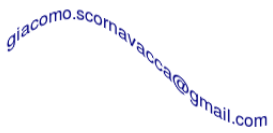
**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`Raddoppia()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione della `struct nodo` e del tipo `nodo`, delle altre strutture dati e tipi eventualmente utilizzati e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione in quanto non permesso l'utilizzo di funzioni non definite all'interno del file consegnato.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

**Inviare le soluzioni entro il 19/04 al seguente indirizzo**



# Programmazione dei Calcolatori con Laboratorio

Simulazione esame 4 (Liberazione)

Data una sequenza $\mathtt{a} = [\mathtt{a}_0, \mathtt{a}_1, \ldots, \mathtt{a}_{n-1}]$ di `n` elementi si vuole progettare una funzione che aggiunge un nuovo elemento `k` in posizione `i` con $-\mathtt{n} \leq \mathtt{i} \leq \mathtt{n}$. Per valori di `i` al di fuori di questo intervallo la sequenza non deve subire modifiche: se $\mathtt{i} \geq 0$ la sequenza risultante dovrà essere $[\mathtt{a}_0, \mathtt{a}_1, \ldots, \mathtt{a}_{i-1}, \mathtt{k}, \mathtt{a}_i, \ldots, \mathtt{a}_{n-1}]$; se $\mathtt{i} < 0$ la sequenza risultante dovrà essere $[\mathtt{a}_0, \mathtt{a}_1, \ldots, \mathtt{a}_{n-i}, \mathtt{k}, \mathtt{a}_{n-i+1}, \ldots, \mathtt{a}_{n-1}]$.

Di seguito sono riportati alcuni esempi per $\mathtt{a} = [0, 1, 2]$, e diversi valori di `i`.

per $\mathtt{i} = 0$, $\mathtt{a} = [\mathtt{k}, 0, 1, 2]$; per $\mathtt{i} = -1$, $\mathtt{a} = [0, 1, 2, \mathtt{k}]$; per $\mathtt{i} = 3$, $\mathtt{a} = [0, 1, 2, \mathtt{k}]$
per $\mathtt{i} = -3$, $\mathtt{a} = [0, \mathtt{k}, 1, 2]$; per $\mathtt{i} = 2$, $\mathtt{a} = [0, 1, \mathtt{k}, 2]$; per $\mathtt{i} = -4$, $\mathtt{a} = [0, 1, 2]$;

Nel progettare la funzione si assuma che la sequenza è memorizzata in una lista concatenata *circolare*: in questo particolare tipo di lista il successore dell'ultimo elemento è il primo elemento ed il predecessore del primo elemento è l'ultimo elemento. Inoltre le chiavi contenute nei nodi della lista sono di tipo `char*`, ovvero stringhe.

La funzione da implementare deve avere il seguente prototipo:

```
nodo *Inserisci( nodo *a, char *k, int i );
```

dove `a` è il puntatore al primo nodo della lista di input, `k` rappresenta la stringa da inserire nel nuovo nodo ed `i` è la posizione. La funzione deve restituire la lista (puntatore a nodo) modificata. Le chiavi nella lista sono di tipo `char*` quindi `nodo` è così definito:

```
struct nodo {
    char *chiave;
    struct nodo *prec;
    struct nodo *succ;
};
typedef struct nodo nodo;
```

Si calcoli il costo della funzione sia in termini di tempo che di memoria supplementare utilizzata (al netto di quella necessaria per memorizzare input ed output).

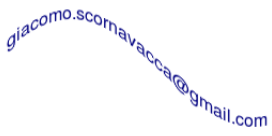
**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`Inserisci()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione della `struct nodo` e del tipo `nodo`, delle altre strutture dati e tipi eventualmente utilizzati e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione in quanto non permesso l'utilizzo di funzioni non definite all'interno del file consegnato.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

**Inviare le soluzioni entro il 26/04 al seguente indirizzo**



# Programmazione dei Calcolatori con Laboratorio

Simulazione esame 5 (Lavoro)

Si implementi una funzione che genera tutte le sequenze binarie di lunghezza `n` dove `n` è un parametro di input. La funzione **non** deve essere ricorsiva o invocare funzioni ricorsive. Le sequenze generate devono essere mostrate a video dalla stessa funzione.

La funzione da implementare deve avere il seguente prototipo:

```
void GeneraSeqBin( int n );
```

dove `n` identifica la lunghezza delle sequenze da generare.

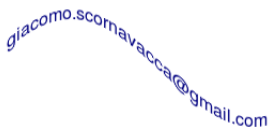
**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`GeneraSeqBin()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- le definizioni delle strutture dati e tipi eventualmente utilizzati e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione in quanto non permesso l'utilizzo di funzioni non definite all'interno del file consegnato.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

**Inviare le soluzioni entro il 3/5 al seguente indirizzo**



# Programmazione dei Calcolatori con Laboratorio

Simulazione esame 6 (Ei fu...)

Affinché su un array associativo `d` implementato con liste di trabocco si possa mantenere costante il rapporto tra il numero `n` di elementi (`k`, `v`) in esso presenti ed il numero `m` di liste di trabocco, è necessario disporre di una funzione che ridimensiona il numero di queste ultime.

Si scriva una funzione in C, che dato un array associativo `d` ed un intero `m`, rimpiazza `d` con un array associativo con `m` liste di trabocco equivalente a `d`.

La funzione da implementare deve avere il seguente prototipo:

```
mappa RidimensionaMappa( mappa d, int m );
```

dove `d` rappresenta l'array associativo di input ed `m` la nuova dimensione. La funzione deve restituire il nuovo array associativo. Si ricorda che `mappa` è così definito

```
struct mappa {
    int dim;
    nodo **tabella;
};
typedef struct mappa mappa;
```

mentre `nodo` è

```
struct nodo {
    chiaveMappa chiave;
    struct nodo *prec, *succ;
};
typedef struct nodo nodo;
```

dove

```
struct chiaveMappa{
    char *k, *v;
};
typedef struct chiaveMappa chiaveMappa;
```

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`RidimensionaMappa()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle struct `mappa`, `nodo` e `chiaveMappa` e dei tipi ad esse associati, delle altre strutture dati e tipi eventualmente utilizzati e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione in quanto non permesso l'utilizzo di funzioni non definite all'interno del file consegnato.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

**Inviare le soluzioni entro il 10/05 al seguente indirizzo**

giacomo.scornavacca@gmail.com

# Programmazione dei Calcolatori con Laboratorio

Simulazione esame 7 (Telecomunicazioni ed Informazione)

Si scriva una funzione in C che inverte l'ordine dei nodi di una lista circolare. La funzione da implementare deve avere il seguente prototipo:

```
nodo *InvertiLista( nodo *a );
```

dove `a` è il puntatore al primo nodo della lista circolare in input. La funzione restituisce la lista modificata, ovvero il puntatore al primo nodo di questa. Si assuma che un nodo è definito come

```
struct nodo {
    int chiave;
    struct nodo *prec, *succ;
};
typedef struct nodo nodo;
```

La funzione *non può creare nuovi nodi* ma utilizzare quelli originali, deve usare una *quantità costante di memoria supplementare* e *non deve sovrascrivere il campo* `chiave` dei nodi.

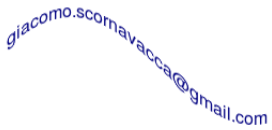
**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`InvertiLista()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione della struct e del tipo `nodo` e delle altre strutture dati e tipi eventualmente utilizzati e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione in quanto non permesso l'utilizzo di funzioni non definite all'interno del file consegnato.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

**Inviare le soluzioni entro il 17/05 al seguente indirizzo**



# Programmazione dei Calcolatori con Laboratorio

Simulazione esame 8 (Ultima)

Da una lista di interi se ne vogliono creare due: quella contenente i numeri dispari e quella contenete i pari. L'ordine degli elementi nelle liste di output deverispettare quello nella lista di partenza. Le liste di input e di output sono *liste concatenate circolari* composte da nodi definiti come

```
struct nodo {
    int chiave;
    struct nodo *prec, *succ;
};
typedef struct nodo nodo;
```

Una lista è identificata con il puntatore al primo nodo della essa. Poiché le liste sono circolari il campo `succ` dell'ultimo nodo della lista punta al primo nodo della lista ed il campo `prec` del primo nodo della lista punta all'ultimo nodo.

La funzione deve essere scritta nel linguaggio C e deve avere il seguente prototipo:

```
paridispari PariDispati( nodo *a );
```

dove `paridispari` è una **struct** che contiene le due liste di output. Più precisamente

```
struct paridispari {
    nodo *pari;
    nodo *dispari;
};
typedef struct paridispari paridispari;
```

Nel primo campo va memorizzato l'indirizzo del primo nodo della lista dei numeri pari mentre nel secondo quello del primo nodo della lista dei numeri dispari

La funzione *non può creare nuovi nodi* ma utilizzare quelli originali, deve usare una *quantità costante di memoria supplementare* e *non deve sovrascrivere il campo* `chiave` dei nodi.

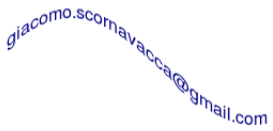
**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`PariDispari()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle **struct** e dei tip `nodo` e `paridispari` e delle altre strutture dati e tipi eventualmente utilizzati e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione in quanto non permesso l'utilizzo di funzioni non definite all'interno del file consegnato.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

**Inviare le soluzioni entro il 24/05 al seguente indirizzo**



# Programmazione dei Calcolatori con Laboratorio

Simulazione esame 1 (Dicembre)

Si vuole progettare una funzione C che calcoli l'area di un triangolo isoscele. La funzione prende in input un triangolo e se questo è isoscele ne restituisce l'area, altrimenti restituisce $-1$.

Un triangolo è descritto da una **struct** così definita:

```
struct triangolo {
    punto p0, p1, p2;
};
typedef struct triangolo triangolo;
```

e dove `punto` è definito come segue:

```
struct punto {
    float x,y;
};
typedef struct punto punto;
```

Ovvero un triangolo è descritto dai suoi tre vertici dove ogni vertice è dato dalle sue coordinate euclidee.

La funzione da progettare deve avere il seguente prototipo:

```
float AreaTriangoloIsoscele( triangolo );
```

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`AreaTriangoloIsoscele()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle **struct** e dei tipi `punto` e `triangolo` e delle altre strutture dati e tipi eventualmente utilizzati e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione in quanto non è permesso l'utilizzo di funzioni non definite all'interno del file consegnato.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

**Inviare le soluzioni entro il 18/12 al seguente indirizzo**

# Programmazione dei Calcolatori con Laboratorio

Simulazione esame 2 (Gennaio)

Il cifrario di Vigenère è un semplice metodo che permette di crittografare un testo mediante una parola chiave segreta. Con la stessa chiave è possibile decodificare il testo crittografato. Per tutti i dettagli su come funziona la cifratura e la decifratura si rimanda alla pagina di Wikipedia

https://it.wikipedia.org/wiki/Cifrario_di_Vigenère

Si devono progettare le due funzioni di cifratura e decifratura: entrambe prendono in input due stringhe delle quali una rappresenta il messaggio da codificare o decodificare e l'altra rappresenta la chiave. In particolare la funzione di cifratura deve avere il seguente prototipo:

```
void Cifra( char[], char[] );
```

dove il primo parametro contiene il messaggio da cifrare ed il secondo contiene la chiave. Il risultato dell'operazione, vale a dire il messaggio cifrato, deve essere sovrascritto sull'array contente il messaggio originale. La funzione di decifratura deve avere il seguente prototipo:

```
void DeCifra( char[], char[] );
```

anche in questo caso il primo parametro contiene il messaggio da decifrare ed il secondo contiene la chiave. Il messaggio decifrato va scritto nel primo array, ovvero quello contente il messaggio cifrato.

Nota bene: tutte le stringhe devono essere composte soltanto da caratteri alfabetici maiuscoli.

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- le funzioni richieste (`Cifra()` e `DeCifra()`) che a loro volta devono rispettare le specifiche imposte dal problema;
- la definizione di tutte le **struct** e dei tipi eventualmente utilizzati e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione in quanto non è permesso l'utilizzo di funzioni non definite all'interno del file consegnato.

La soluzione non può far uso di variabili globali. Infine la funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

**Inviare le soluzioni entro il 7 gennaio al seguente indirizzo**

# Programmazione dei Calcolatori con Laboratorio

Esame del 18 febbraio 2016

Sia `a` una sequenza ordinata, si vuole progettare una funzione che localizzi il punto di inserimento di un nuovo elemento `x` nella sequenza mantenendola ordinata. Ovvero, qualora sia possibile aggiungere un nuovo elemento `x` in `a`, qual è la posizione in cui dovrebbe essere aggiunto in modo tale che la nuova eventuale sequenza risulti ancora ordinata? Se un elemento uguale a `x` è già presente nella sequenza il nuovo punto di inserimento deve essere quello più a sinistra, ovvero quello precedente a tutti gli elementi uguali a `x`.

La sequenza in questione contiene date ordinate in modo non decrescente, ovvero stringhe della forma `GGmese`, dove `GG` indica il giorno con due caratteri *numerici* e `mese` è una stringa composta da uno, due, tre o quattro caratteri ed indica il mese nella numerazione romana (`I`, `II`,..., `VIII`, ..., `XI`, `XII`).

Esempi: se `a` = (`02II`, `18II`, `18II`, `25XII`) e `x` = `01I`, la funzione deve restituire `0`; se `x` = `18II` la funzione deve restituire `1`; se `x` = `31XII` la funzione deve restituire `4`.

La funzione deve avere il seguente prototipo:

```
int Bisect( char *a[], int n,  char *x )
```

Ovvero la sequenza di date è memorizzata all'interno di un array di stringhe; `n` è la dimensione dell'array e `x` è la data di cui si vuole conoscere la posizione.

Come pre-condizione si assuma che le date `x` e quelle in `a` siano corrette (ovvero tali stringhe codificano date corrette rispettando il formato descritto) e che in `a` compaiano effettivamente in ordine non decrescente.

La funzione deve utilizzare soltanto i dati appena descritti pertanto non è possibile fare uso di variabili globali.

Dopo aver progettato ed implementato la funzione, se ne calcoli il costo sia in termini di tempo che di memoria supplementare utilizzata (al netto di quella utilizzata per memorizzare l'input e l'output).

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funzione richiesta (`Bisect()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle strutture dati e dei tipi eventualmente utilizzati;
- ogni altra funzione utilizzata dalla soluzione;
- gli *header* delle librerie utilizzate.

La funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

# Programmazione dei Calcolatori con Laboratorio

Simulazione esame 4 (Aprile)

Si vuole progettare ed implementare nel linguaggio C una funzione che rimuova tutti gli elementi ripetuti da una lista concatenata data in input.

La lista è una successione di nodi definiti come segue

```
struct nodo {
    char *chiave;
    struct nodo *succ, *prec;
};
typedef struct nodo nodo;
```

dove ogni nodo è collegato al nodo che lo segue (attraverso il puntatore `succ`) ed a quello che lo precede (con il puntatore `prec`). Il primo (rispettivamente, l'ultimo) nodo della lista hanno i campi `prec` (rispettivamente, `succ`) uguali a `NULL`. La lista è identificata attraverso il puntatore al primo nodo della stessa.

La funzione da progettare deve avere il seguente prototipo:

```
nodo *EliminaRipetizioni( nodo *L );
```

La funzione deve eliminare tutti i nodi con contenuti ripetuti e restituire la lista modificata preservando l'ordinamento originale dei nodi residui.

Ad esempio se la lista originale contiene le stringhe `aaa, sss, sss, ddd, aaa, sss, ccc, ccc`, la funzione deve ritornare la lista contenente le stringhe `aaa, sss, ddd, ccc`.

**Suggerimento**: Utilizzando una opportuna struttura dati di appoggio si può fare in modo che il numero di operazioni eseguito sia, in media, lineare nella lunghezza della lista di input.

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funziona richiesta (`EliminaRipetizioni()` che a loro volta devono rispettare le specifiche imposte dal problema;
- la definizione delle **struct**, del tipo `nodo` e delle altre strutture dati e tipi eventualmente utilizzati e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione in quanto non è permesso l'utilizzo di funzioni non definite all'interno del file consegnato.

La soluzione non può far uso di variabili globali. Infine la funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

**Inviare le soluzioni entro il 6 maggio al seguente indirizzo**

# Programmazione dei Calcolatori con Laboratorio

Simulazione esame 5 (Maggio)

Scrivere una funzione in C che, date due liste `L1` e `L2`, restituisce una lista ottenuta alternando nodi di `L1` a nodi di `L2`. In particolare se $\texttt{L1} = \texttt{a}_0, \texttt{a}_1, \ldots, \texttt{a}_{n-1}$ e $\texttt{L2} = \texttt{b}_0, \texttt{b}_1, \ldots, \texttt{b}_{m-1}$, la lista risultante deve essere

$$\begin{cases} \texttt{a}_0, \texttt{b}_0, \texttt{a}_1, \texttt{b}_1, \ldots, \texttt{a}_{m-1}, \texttt{b}_{m-1}, \texttt{a}_m, \texttt{a}_{m+1}, \ldots, \texttt{a}_{n-1} & \text{se } \texttt{n} > \texttt{m} \\ \texttt{a}_0, \texttt{b}_0, \texttt{a}_1, \texttt{b}_1, \ldots, \texttt{a}_{n-1}, \texttt{b}_{n-1}, \texttt{b}_n, \texttt{b}_{n+1}, \ldots, \texttt{b}_{m-1} & \text{altrimenti} \end{cases}$$

La lista risultante deve contenere i nodi delle due liste in input, pertanto queste, dopo l'esecuzione della funzione, cesseranno di esistere.

Una lista è definita come una successione di nodi che, a loro volta, sono definiti come segue

```
struct nodo {
    char *chiave;
    struct nodo *succ, *prec;
};
typedef struct nodo nodo;
```

dove ogni nodo è collegato al nodo che lo segue (attraverso il puntatore `succ`) ed a quello che lo precede (con il puntatore `prec`). Il primo (rispettivamente, l'ultimo) nodo della lista hanno i campi `prec` (rispettivamente, `succ`) uguali a `NULL`. La lista è identificata attraverso il puntatore al primo nodo della stessa.

La funzione da progettare deve avere il seguente prototipo:

```
nodo *IntrecciaListe( nodo *L1, nodo *L2 );
```

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funziona richiesta (`IntrecciaListe()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle **struct**, del tipo `nodo` e delle altre strutture dati e tipi eventualmente utilizzati e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione in quanto non è permesso l'utilizzo di funzioni non definite all'interno del file consegnato.

La soluzione non può far uso di variabili globali. Infine la funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

**Inviare le soluzioni entro la mezzanotte di oggi, 25 maggio, al seguente indirizzo**

# Programmazione dei Calcolatori con Laboratorio

Simulazione esame 6 (Giugno)

Scrivere una funzione **ricorsiva** in C che, date due liste `L1` e `L2` contenenti stringhe, restituisce una nuova lista (con nuovi nodi) in cui il nodo in posizione `i` contiene la stringa ottenuta concatenando le due stringhe contenute nei nodi in posizione `i` di `L1` e `L2` separate da uno spazio. La nuova lista avrà la lunghezza della lista più corta tra quelle in input. Le liste `L1` e `L2` non devono essere alterate.

Ad esempio se `L1` contiene le stringhe "`prima`", "`seconda`" e `L2` le stringhe "`terza`", "`quarta`" e "`quinta`", la lista risultante dovrà contenere le stringhe "`prima terza`", "`seconda quarta`".

Una lista è definita come una successione di nodi che, a loro volta, sono definiti come segue

```
struct nodo {
    char *chiave;
    struct nodo *succ, *prec;
};
typedef struct nodo nodo;
```

dove ogni nodo è collegato al nodo che lo segue (attraverso il puntatore `succ`) ed a quello che lo precede (con il puntatore `prec`). Il primo (rispettivamente, l'ultimo) nodo della lista hanno i campi `prec` (rispettivamente, `succ`) uguali a `NULL`. La lista è identificata attraverso il puntatore al primo nodo della stessa.

La funzione da progettare deve avere il seguente prototipo:

```
nodo *Zip( nodo *L1, nodo *L2 );
```

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funziona richiesta (`Zip()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle **struct**, del tipo `nodo` e delle altre strutture dati e tipi eventualmente utilizzati e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione in quanto non è permesso l'utilizzo di funzioni non definite all'interno del file consegnato.

La soluzione non può far uso di variabili globali. Infine la funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

**Inviare le soluzioni alla email del docente**.

# Programmazione dei Calcolatori con Laboratorio

Simulazione esame 7 (Giugno)

Scrivere una funzione **ricorsiva** in C che inverte la lista in input. La lista in output, inversione di quella in input, deve contenere gli stessi nodi di quella in input in ordine invertito, ovvero non devono essere creati nuovi nodi.

Una lista è definita come una successione di nodi che, a loro volta, sono definiti come segue

```
struct nodo {
    char *chiave;
    struct nodo *succ, *prec;
};
typedef struct nodo nodo;
```

dove ogni nodo è collegato al nodo che lo segue (attraverso il puntatore `succ`) ed a quello che lo precede (con il puntatore `prec`). Il primo (rispettivamente, l'ultimo) nodo della lista hanno i campi `prec` (rispettivamente, `succ`) uguali a `NULL`. La lista è identificata attraverso il puntatore al primo nodo della stessa.

La funzione da progettare deve avere il seguente prototipo:

```
nodo *Reverse( nodo *L );
```

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funziona richiesta (`Reverse()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle **struct**, del tipo `nodo` e delle altre strutture dati e tipi eventualmente utilizzati e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione in quanto non è permesso l'utilizzo di funzioni non definite all'interno del file consegnato.

La soluzione non può far uso di variabili globali. Infine la funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

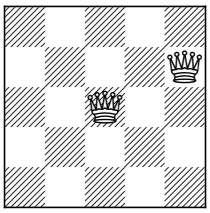
**Inviare le soluzioni alla email del docente**.

# Programmazione dei Calcolatori con Laboratorio

Simulazione esame 1 (Gennaio)

Il problema delle `n` regine consiste nel posizionare `n` regine in una scacchiera di dimensioni `n` per `n` in modo tale che queste non si minaccino reciprocamente. Nella variante *con vincoli* sono date le posizioni di $\texttt{k} \leq \texttt{n}$ regine e si vogliono posizionare le restanti.

Scrivere una funzione in C che elenca tutte le soluzioni del problema delle `n` regine con vincoli. La funzione, oltre al parametro `n`, prende in input un array `a` che codifica la disposizione delle regine già posizionate. Questo è un array di `n` interi (uno per ogni colonna della scacchiera) tale che se `a[c]` è un intero compreso tra `0` e $\texttt{n} - \texttt{1}$ allora la colonna `c` è vincolata ad avere una regina nella riga `a[c]`. Per tutti gli altri valori di `a[c]` la colonna `c` non risulta vincolata. Ad esempio se $\texttt{n} = \texttt{5}$ e $\texttt{a}[] = \{-\texttt{1}, -\texttt{1}, \texttt{2}, -\texttt{1}, \texttt{1}\}$ allora la funzione deve elencare tutte le soluzioni in cui le colonne `2` e `4` sono definite nel seguente modo:



La funzione da progettare deve avere il seguente prototipo:

```
void RegineVincolate( int a[], int n );
```

dove `a` è l'array che rappresenta i vincoli iniziali e `n` la sua dimensione. La soluzione consisterà nella stampa a video di tutte le configurazioni trovate attraverso tabelle binarie di dimensione `n` per `n` dove `1` sta a significare la presenza di una regina, `0` altrimenti.

**ATTENZIONE:** la funzione deve utilizzare al massimo una quantità di memoria lineare in `n`.

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funziona richiesta (`RegineVincolate()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle altre strutture dati e tipi eventualmente utilizzati e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione in quanto non è permesso l'utilizzo di funzioni non definite all'interno del file consegnato.

La soluzione non può far uso di variabili globali. Infine la funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

**Inviare le soluzioni al seguente indirizzo email entro il 24 gennaio 2017**.

tutor.programmazionec@gmail.com

# Programmazione dei Calcolatori con Laboratorio

Simulazione esame 2 (Marzo, 2017)

Si supponga di avere un file video dal quale si vogliono estrarre delle sequenze e concatenarle tra di loro in modo da ottenere un secondo file video. Ogni sequenza è individuata da una coppia di interi che indicano il primo e l'ultimo frame della sequenza (il primo frame precede l'ultimo). Se ci limitassimo semplicemente a concatenare gli spezzoni il video risultante potrebbe contenere delle scene ripetute in quanto le sequenze potrebbero sovrapporsi. Ad esempio se le sequenze fossero

$$(1,4),(10,15),(12,18),(16,20),$$

poiché le sequenze $(10,15)$ e $(12,18)$ e le sequenze $(12,18)$ e $(16,20)$ si sovrappongono, sul video risultante comparirebbero due volte gli spezzoni composti dai frame

$$(12,13,14,15) \text{ e } (16,17,18).$$

Si vuole progettare una funzione che elimina queste ripetizioni fondendo intervalli di frame che si sovrappongono.

Le sequenze le identificheremo con un *intervallo* che è una coppia di interi con la proprietà che il primo elemento della coppia precede il secondo.

```
struct intervallo{
    int pr, ul;
};
typedef struct intervallo intervallo;
```

quindi se `x` è di tipo `intervallo`,

$$\texttt{x.pr} \leq \texttt{x.ul}. \tag{1}$$

Le sequenze sono date in un array `s` di `n struct intervallo` con la seguente proprietà: per ogni $\texttt{i} = 0,\ldots,\texttt{n}-2$,

$$\texttt{s[i].pr} \leq \texttt{s[i+1].pr} \quad \text{e} \quad \texttt{s[i].ul} \leq \texttt{s[i+1].ul}. \tag{2}$$

Quindi se due sequenze successive di `s` si sovrappongono allora l'ultimo estremo della prima sequenza si trova all'interno della seconda (o, equivalentemente, il primo estremo della seconda si trova all'interno della prima). L'esempio mostrato in precedenza verifica questa proprietà.

La funzione che si deve progettare deve prendere in input un array di sequenze con le proprietà di cui sopra e sostituire le sequenze che si sovrappongono con un'unica sequenza data dall'unione di queste. Ad esempio se

$$\texttt{s} = [(1,4),(10,15),(12,18),(16,20),(30,40),(35,40),(40,50)],$$

la funzione deve restituire

$$[(1,4),(10,20),(30,50)].$$

L'output della funzione deve essere memorizzato in un `vector` (ovvero un array dinamico) di `struct intervallo`.

La funzione da progettare deve avere il seguente prototipo:

```
vector UnisciIntervalliSovraposti( intervallo s[], int n );
```

dove `s` è l'array contenente la descrizione delle sequenze per mezzo di `struct intervallo`: ogni `intervallo` soddisfa la proprietà (1) mentre `s` soddisfa la proprietà (2).

Il `vector` restituito dalla funzione deve rispettare la seguente definizione:

```
struct vector{
    intervallo *a;
    int c;
    int n;
};
typedef struct vector vector;
```

dove `c` indica la capacità di `a` ed `n` il numero di `struct intervallo` che effettivamente contiene. Le funzioni di gestione del `vector` dovranno essere modificate in modo opportuno.

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funziona richiesta (`UnisciIntervalliSovraposti()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle **struct**, dei tipi `intervallo` e `vector`, delle altre strutture dati e tipi eventualmente utilizzati e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione in quanto non è permesso l'utilizzo di funzioni non definite all'interno del file consegnato.

La soluzione non può far uso di variabili globali. Infine la funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

**Inviare le soluzioni al seguente indirizzo email entro il 28 marzo 2017**.

# Programmazione dei Calcolatori con Laboratorio

Simulazione esame 3 (Aprile, 2017)

*Stesso problema della precedente esercitazione ma con output su lista concatenata.*

Si supponga di avere un file video dal quale si vogliono estrarre delle sequenze e concatenarle tra di loro in modo da ottenere un secondo file video. Ogni sequenza è individuata da una coppia di interi che indicano il primo e l'ultimo frame della sequenza (il primo frame precede l'ultimo). Se ci limitassimo semplicemente a concatenare gli spezzoni il video risultante potrebbe contenere delle scene ripetute in quanto le sequenze potrebbero sovrapporsi. Ad esempio se le sequenze fossero

$$(1,4),(10,15),(12,18),(16,20),$$

poiché le sequenze $(10,15)$ e $(12,18)$ e le sequenze $(12,18)$ e $(16,20)$ si sovrappongono, sul video risultante comparirebbero due volte gli spezzoni composti dai frame

$$(12,13,14,15) \text{ e } (16,17,18).$$

Si vuole progettare una funzione che elimina queste ripetizioni fondendo intervalli di frame che si sovrappongono.

Le sequenze le identificheremo con un *intervallo* che è una coppia di interi con la proprietà che il primo elemento della coppia precede il secondo.

```
struct intervallo{
    int pr, ul;
};
typedef struct intervallo intervallo;
```

quindi se `x` è di tipo `intervallo`,

$$\texttt{x.pr} \leq \texttt{x.ul}. \tag{1}$$

Le sequenze sono date in un array `s` di `n struct intervallo` con la seguente proprietà: per ogni $\texttt{i} = 0,\ldots,\texttt{n}-2$,

$$\texttt{s[i].pr} \leq \texttt{s[i+1].pr} \quad \text{e} \quad \texttt{s[i].ul} \leq \texttt{s[i+1].ul}. \tag{2}$$

Quindi se due sequenze successive di `s` si sovrappongono allora l'ultimo estremo della prima sequenza si trova all'interno della seconda (o, equivalentemente, il primo estremo della seconda si trova all'interno della prima). L'esempio mostrato in precedenza verifica questa proprietà.

La funzione che si deve progettare deve prendere in input un array di sequenze con le proprietà di cui sopra e sostituire le sequenze che si sovrappongono con un'unica sequenza data dall'unione di queste. Ad esempio se

$$\texttt{s} = [(1,4),(10,15),(12,18),(16,20),(30,40),(35,40),(40,50)],$$

la funzione deve restituire

$$[(1,4),(10,20),(30,50)].$$

L'output della funzione deve essere memorizzato in una lista concatenata di `struct intervallo`. Questa lista è definita da una sequenza di nodi, uno per ogni intervallo in output, che a loro volta sono definiti nel seguente modo:

```
struct nodo {
    intervallo chiave;
    struct nodo *succ, precc;
};
typedef struct nodo nodo;
```

ogni nodo è collegato a quello che lo segue attraverso il puntatore `succ` e a quello che lo precede attraverso il puntatore `prec`. Il primo (rispettivamente, l'ultimo) nodo della lista ha il campo `succ` (rispettivamente, `prec`) uguale a `NULL`. La lista è definita attraverso il puntatore al suo primo nodo.

La funzione da progettare deve avere il seguente prototipo:

```
nodo *UnisciIntervalliSovraposti( intervallo s[], int n );
```

dove `s` è l'array contenente la descrizione delle sequenze per mezzo di `struct intervallo`: ogni `intervallo` soddisfa la proprietà (1) mentre `s` soddisfa la proprietà (2). La funzione deve restituire la lista di **intervallo**, ovvero il puntatore al primo nodo della lista concatenata generata dalla funzione.

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funziona richiesta (`UnisciIntervalliSovraposti()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle **struct**, dei tipi `intervallo` e `nodo`, delle altre strutture dati e tipi eventualmente utilizzati e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione in quanto non è permesso l'utilizzo di funzioni non definite all'interno del file consegnato.

La soluzione non può far uso di variabili globali. Infine la funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

**Inviare le soluzioni al seguente indirizzo email entro il 19 aprile 2017**.

# Programmazione dei Calcolatori con Laboratorio

Simulazione esame 4 (Aprile, 2017)

Dato un dizionario `D` coppie *chiave-valore* con valore di tipo intero, creare una lista contenente tutti gli elementi in `D` ordinati per valore decrescente.

Il dizionario è rappresentato per mezzo di liste di trabocco. Gli elementi del dizionario, le coppie chiave-valore, sono memorizzati nella seguente **struct**:

```
struct coppia{
    char *k;
    int v;
};
typedef struct coppia coppia;
```

Dove la stringa `k` è la chiave e `v` l'intero associato a `k`. Le liste di trabocco sono composte da nodi contenenti `struct coppia`.

```
struct nodo {
    coppia chiave;
    struct nodo *succ;
    struct nodo *prec;
};
typedef struct nodo nodo;
```

Ogni nodo delle liste è collegato a quello che lo precede attraverso il puntatore `prec` ed a quello che lo segue per mezzo del puntatore `succ`. Il campo `prec` del primo nodo della lista ed il campo `succ` dell'ultimo valgono `NULL`. Il dizionario è composto da un insieme di liste di trabocco delle quali si conoscono i puntatori ai primi nodi di queste memorizzati un un array di puntatori a `nodo`. La struttura dati `mappa` che memorizza il dizionario è composta da questo array ed un intero che ne indica la dimensione.

```
struct mappa{
    int dim;
    nodo **tabella;
};
typedef struct mappa mappa;
```

Il campo `tabella` è l'array di `dim` puntatori a `nodo` ognuno dei quali identifica una lista di trabocco.

La funzione da progettare deve avere il seguente prototipo:

```
nodo *OrdinaChiavi( mappa D );
```

dove `D` è il dizionario di input. La lista restituita dalla funzione deve contenere una copia dei nodi delle liste di trabocco di `D` ordinati in modo non crescente rispetto al campo `v` della `coppia` nel campo `chiave` dei nodi.

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome). Tale file deve contenere soltanto:

- la funziona richiesta (`OrdinaChiavi()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- la definizione delle **struct**, dei tipi `coppia`, `nodo` e `mappa`, delle altre strutture dati e tipi eventualmente utilizzati e le funzioni che le utilizzano nonché gli *header* delle librerie utilizzate.
- ogni altra funzione utilizzata dalla soluzione in quanto non è permesso l'utilizzo di funzioni non definite all'interno del file consegnato.

La soluzione non può far uso di variabili globali. Infine la funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

**Inviare le soluzioni al seguente indirizzo email entro il 26 aprile 2017**.

# Programmazione dei Calcolatori con Laboratorio

Simulazione esame 5 (Maggio, 2017)

`a` e `b` sono due liste di elementi di tipo `double` tali che `a` è ordinata in modo *crescente* e `b` in modo *decrescente*. Si chiede di progettare una funzione che ordina in modo *crescente* gli elementi di `a` e `b` in un'unica lista.

I nodi che compongono le liste sono definiti come segue:

```
struct nodo {
    double chiave;
    struct nodo *succ;
    struct nodo *prec;
};
typedef struct nodo nodo;
```

Ogni nodo delle liste è collegato a quello che lo precede attraverso il puntatore `prec` ed a quello che lo segue per mezzo del puntatore `succ`. Il campo `prec` del primo nodo della lista ed il campo `succ` dell'ultimo valgono `NULL`.

La funzione da progettare deve avere il seguente prototipo:

```
nodo *MergeInv( nodo *a, nodo *b );
```

dove `a` e `b` sono le due liste ordinate in modo crescente (`a`) e decrescente (`b`). La lista restituita dalla funzione deve essere composta dai nodi delle liste `a` e `b`, ovvero non è permesso creare nuovi nodi.

La soluzione non può far uso di variabili globali.

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome).

Tale file, con l'aggiunta di una opportuna funzione `main` che non deve essere contenuta nel file consegnato, deve poter essere compilato senza errori. Quindi deve contenere:

- la funziona richiesta (`MergeInv()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- tutte le definizioni di tipi e **struct**;
- ogni altra funzione utilizzata dalla soluzione in quanto non è permesso l'utilizzo di funzioni non definite all'interno del file consegnato;
- tutti gli *header* delle librerie utilizzate.

Infine la funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

**Inviare le soluzioni al seguente indirizzo email entro il 9 maggio 2017**.

# Programmazione dei Calcolatori con Laboratorio

Simulazione esame 6 (Maggio, 2017)

Si progetti una funzione che data una lista concatenata `l` ed il puntatore ad uno dei suoi nodi `x`, scambia la posizione di `x` con il nodo successivo (se questo è presente).

I nodi che compongono la lista sono definiti come segue:

```
struct nodo {
    double chiave;
    struct nodo *succ;
    struct nodo *prec;
};
typedef struct nodo nodo;
```

Ogni nodo è collegato a quello che lo precede attraverso il puntatore `prec` ed a quello che lo segue per mezzo del puntatore `succ`. Il campo `prec` del primo nodo della lista ed il campo `succ` dell'ultimo valgono `NULL`.

La funzione da progettare deve avere il seguente prototipo:

```
nodo *ListaScambiaNodiCons( nodo *l, nodo *x );
```

dove `l` è il puntatore al primo nodo della lista e `x` è i primo dei due nodi consecutivi che devono essere scambiati. Lu funzione restituisce la lista modificata.

Si risolva il problema tenendo conto che la funzione verrà utilizzata all'interno di un grande progetto di cui non conosciamo tutte le parti. Sappiamo però che esiste una struttura dati con un riferimento al nodo contenente il minimo della lista.

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome).

Tale file, con l'aggiunta di una opportuna funzione `main` che non deve essere contenuta nel file consegnato, deve poter essere compilato senza errori. Quindi deve contenere:

- la funziona richiesta (`ListaScambiaNodiCons()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- tutte le definizioni di tipi e **struct**;
- ogni altra funzione utilizzata dalla soluzione in quanto non è permesso l'utilizzo di funzioni non definite all'interno del file consegnato;
- tutti gli *header* delle librerie utilizzate.

Infine la funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

**Inviare le soluzioni al seguente indirizzo email entro il 23 maggio 2017**.

# Programmazione dei Calcolatori con Laboratorio

Test

---

1. Cosa viene restituito dalla seguente funzione C?

```
f1(){
    int s1[1000] = {0};
    int *s2 = malloc(sizeof(s1));

    if( sizeof(s1) == sizeof(s2))
        return 0;
    if( sizeof(s1) > sizeof(s2))
        return 1;
    return -1;
}
```

**Risposte**

- ◯ -1
- ◯ 0
- ◯ 1

---

2. La lista `a` viene generata utilizzando il seguente codice Python dove `n` è un intero positivo.

```
a = []
for i in range(n):
    a.append(i*n)
```

Quale tra i seguenti algoritmi risulta più efficiente per ordinare `a`?

**Risposte**

- ◯ BubbleSort
- ◯ CountingSort
- ◯ QuickSort

---

3. Cosa viene stampato dal seguente frammento di codice?

```
char s[] = "7+2+1=10";
int i;

for(i =0; i < strlen(s); i++)
    if(s[i] <= '9' && s[i] >= '0')
        if( (s[i]-'0')%2 == 0 )
            s[i]++;
        else
            s[i]--;

printf("%s\n", s);
```

**Risposte**

- ◯ 6+3+0=11
- ◯ 6+3+0=01
- ◯ 6+3+0=9

---

4. Cosa viene stampato dal seguente frammento di codice?

```
void f3(char *a){
    if(*a != '\0'){
        *a = 'B' + *a - 'a';
        f3(a+1);
    }
}

char s[] = "abcdef";
f3(s);
printf("%s\n", s);
```

**Risposte**

- ◯ La stringa `abcdef`
- ◯ Nulla, c'è un errore
- ◯ La stringa `BCDEFG`

---

5. Qual è il valore di `b[0]`?

```
char a[] = {10,20,30,40};
char *b = a;
b = b+2;
```

**Risposte**

- ◯ 12
- ◯ 30
- ◯ Nessuno, c'è un errore

6. Sia `D` un dizionario implementato con liste di trabocco. Quale delle seguenti affermazioni è vera?

   (a) Le coppie contenute nella stessa lista di trabocco hanno tutte la stessa chiave;

   (b) Le coppie contenute nella stessa lista di trabocco collidono;

**Risposte**

- ◯ (a)
- ◯ (b)
- ◯ Nessuna delle due

---

7. Qual è il tipo di ritorno della seguente funzione Python?

```
def f(a):
    b = a[0]
    for x in a[1:]:
        try:
            b += x
        except TypeError:
            ''
    return b
```

**Risposte**

- ◯ Intero
- ◯ Float
- ◯ Il tipo di `a[0]`

---

8. Si consideri il seguente frammento di codice Python, cosa è `b`?

```
a = (1,2,3)
b = a*2
```

**Risposte**

- ◯ Una tripla
- ◯ Una tupla con 6 elementi
- ◯ Un intero

---

9. Si consideri il codice Python riportato di seguito, si dica quale delle affermazioni riportate a destra non è vera.

```
a = [0,1,2]
a.append([30,40,50,60])
```

**Risposte**

- ◯ `len(a[3]) > len(a)`
- ◯ `id(a[3]) != id(a)`
- ◯ `type(a[3]) == type(a)`

---

10. Quanto vale `strlen(s)` al termine del seguente frammento di codice C?

```
char s[100];
int i;
for(i = 9; i >= 0; i--){
    s[i] = 'a'; s[i+1] = '\0';
}
```

**Risposte**

- ◯ 1
- ◯ 10
- ◯ 100

---

Nome e Cognome (in stampatello): _____

Data: _____ Firma: _____

# Programmazione dei Calcolatori con Laboratorio

Esame del 15 giugno 2017

La data del 15 giungo 2017 può essere indicata anche nel seguente modo: giovedì 24, 2017 indicando il giorno della settimana ed il numero di settimane trascorso dall'inizio dell'anno. Si vuole scrivere una funzione che confronta due date espresse utilizzando questa notazione alternativa.

`d1` e `d2` sono due date nel formato `gggssaaaa` dove

- `ggg` sta per il giorno della settimana (`lun`, `mar`, `mer`, `gio`, `ven`, `sab`, `dom`);
- `ss` indica il numero della settimana espresso con due cifre da `00` a `52`;
- `aaaa` rappresenta l'anno espresso con quattro cifre decimali.

Tenendo conto che la settimana comincia dal lunedì, progettare ed implementare nel linguaggio C una funzione che, date `d1` e `d2`, restituisce

$$\begin{cases} -1 & \text{se } \texttt{d1} \text{ precede } \texttt{d2}; \\ 0 & \text{se } \texttt{d1} \text{ coincide con } \texttt{d2}; \\ +1 & \text{altrimenti.} \end{cases}$$

La funzione deve avere il seguente prototipo:

```
int dwacomp(char *d1, char *d2);
```

dove `d1` e `d2` sono le due stringhe che codificano le due date secondo quanto detto in precedenza. Si può assumere come pre-condizione che `d1` e `d2` rispettino il formato imposto.

Dopo aver progettato ed implementato la funzione, se ne calcoli il costo sia in termini di tempo che di memoria supplementare utilizzata (al netto di quella utilizzata per memorizzare l'input e l'output).

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome).

Tale file, con l'aggiunta di una opportuna funzione `main` che non deve essere contenuta nel file consegnato, deve poter essere compilato senza errori. Quindi deve contenere:

- la funziona richiesta (`dwacomp()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- ogni altra funzione utilizzata dalla soluzione in quanto non è permesso l'utilizzo di funzioni non definite all'interno del file consegnato;
- tutti gli *header* delle librerie utilizzate.

Infine la funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

# Programmazione dei Calcolatori con Laboratorio

Esame del 28 giugno 2017

Siano date due sequenze di interi ordinati in modo crescente, una contenente soltanto numeri pari e l'altra soltanto numeri dispari. Si progetti una funzione C che crei una terza sequenza ordinata in modo crescente contenente elementi delle due sequenze.

Le sequenze in questione sono memorizzate in liste concatenate i cui nodi sono definiti come segue:

```
struct nodo {
    int chiave;
    struct nodo *succ;
    struct nodo *prec;
};
typedef struct nodo nodo;
```

Ogni nodo è collegato a quello che lo precede attraverso il puntatore `prec` ed a quello che lo segue per mezzo del puntatore `succ`. Il campo `prec` del primo nodo della lista ed il campo `succ` dell'ultimo valgono `NULL`.

La funzione da progettare deve avere il seguente prototipo:

```
nodo *FondiPariDispari( nodo *pari, nodo *dispari );
```

dove `pari` è il puntatore al primo nodo della lista contenente numeri pari e `dispari` è il puntatore al primo nodo della lista contenente numeri dispari. La funzione deve restituire una nuova lista ordinata contenente gli interi delle due liste.

Nel risolvere il problema si tenga conto che la funzione `FondiPariDispari()` così come le liste `pari` e `dispari` fanno parte di un progetto su cui stanno lavorando altri sviluppatori.

Dopo aver progettato ed implementato la funzione, se ne calcoli il costo sia in termini di tempo che di memoria supplementare utilizzata (al netto di quella utilizzata per memorizzare l'input e l'output).

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome).

Tale file, con l'aggiunta di una opportuna funzione `main` che non deve essere contenuta nel file consegnato, deve poter essere compilato senza errori. Quindi deve contenere:

- la funziona richiesta (`FondiPariDispari()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- tutte le definizioni di tipi e **struct**;
- ogni altra funzione utilizzata dalla soluzione in quanto non è permesso l'utilizzo di funzioni non definite all'interno del file consegnato;
- tutti gli *header* delle librerie utilizzate.

Infine la funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

# Programmazione dei Calcolatori con Laboratorio

Esame del 26 settembre 2017

Data una sequenza `a` contenente `n` interi si progetti una funzione C che, per ogni $0 \leq i < n$, aggiunge ad `a` un nuovo elemento in posizione `2i` contenente la somma degli elementi nelle posizioni $i, i+1, \ldots, n-1$ della sequenza originale. Ad esempio se $a = (2, 3, 5, 1, 7)$ la funzione deve restituire $a = (18, 2, 16, 3, 13, 5, 8, 1, 7, 7)$.

La sequenza è memorizzata in una lista concatenata i cui nodi sono definiti come segue:

```
struct nodo {
    int chiave;
    struct nodo *succ;
    struct nodo *prec;
};
typedef struct nodo nodo;
```

Ogni nodo è collegato a quello che lo precede attraverso il puntatore `prec` ed a quello che lo segue per mezzo del puntatore `succ`. Il campo `prec` del primo nodo della lista ed il campo `succ` dell'ultimo valgono `NULL`.

La funzione da progettare deve avere il seguente prototipo:

```
nodo *AggiungiSomme( nodo *a );
```

dove `a` è il puntatore al primo nodo della lista. La funzione deve restituire la lista `a` modificata come discusso in precedenza.

Dopo aver progettato ed implementato la funzione, se ne calcoli il costo sia in termini di tempo che di memoria supplementare utilizzata (al netto di quella utilizzata per memorizzare l'input e l'output).

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome).

Tale file, con l'aggiunta di una opportuna funzione `main` che non deve essere contenuta nel file consegnato, deve poter essere compilato senza errori. Quindi deve contenere:

- la funziona richiesta (`AggiungiSomme()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- tutte le definizioni di tipi e **struct**;
- ogni altra funzione utilizzata dalla soluzione in quanto non è permesso l'utilizzo di funzioni non definite all'interno del file consegnato;
- tutti gli *header* delle librerie utilizzate.

Infine la funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

# Programmazione dei Calcolatori con Laboratorio

Simulazione esame 1
Dicembre, 2017

Si progetti una funzione che dato un array di `n` interi `H` disegna su terminale il contenuto di `H` utilizzando barre verticali. Ovvero, per ogni indice `i` di `H` la barra relativa a `H[i]` deve essere composta da `H[i]` caratteri '*'. Ad esempio se $\texttt{H} = \{2, 4, 0, 3\}$ la funzione deve mostrare

```
 *
 * *
** *
** *
```

La funzione da progettare deve avere il seguente prototipo:

```
void MostraIstogramma( int H[], int n );
```

dove `H` è l'array e `n` la sua dimensione. Si assuma che gli elementi di `H` siano tutti non negativi.

La soluzione deve utilizzare una quantità costante di memoria supplementare (a netto di quella utilizzata per l'input e l'output).

**Facolatativo** Risolvere il problema anche quando `H` contiene valori negativi. In tal caso mostrare anche l'asse delle ascisse. Ad esempio se $\texttt{H} = \{2, -4, 0, 3\}$ la funzione deve stampare su terminale

```
   *
*  *
*  *
----
 *
 *
 *
 *
```

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome).

Tale file, con l'aggiunta di una opportuna funzione `main` che non deve essere contenuta nel file consegnato, deve poter essere compilato senza errori. Quindi deve contenere:

- la funziona richiesta (`MostraIstogramma()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- ogni altra funzione utilizzata dalla soluzione in quanto non è permesso l'utilizzo di funzioni non definite all'interno del file consegnato;
- tutti gli *header* delle librerie utilizzate.

Infine la funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

**Inviare le soluzioni al seguente indirizzo email entro il 18 dicembre 2017**.

# Programmazione dei Calcolatori con Laboratorio

Simulazione esame 2
Gennaio, 2018

Il LOGO è un linguaggio di programmazione orientato alla grafica con finalità didattiche ideato negli anni '60. Le istruzioni servono a guidare la matita su un foglio che in questo modo produrrà un disegno. Per maggiori informazioni sul linguaggio LOGO si consulti l'apposita pagina di Wikipedia.

Lo scopo di questa esercitazione è quello di progettare un interprete di un linguaggio simil-LOGO anche se molto semplificato. Le istruzioni che si possono inviare alla matita sono:

- `Gx,y` ovvero salta alla posizione `x`, `y` del foglio sollevando la matita;
- `Ny` ovvero vai a nord di `y` punti sul foglio tenendo la matita poggiata sul foglio tracciando una linea;
- `Ex` come sopra ma a est di `x` punti;
- `Sy` come sopra ma a sud di `y` punti;
- `Wx` come sopra ma a ovest di `x` punti.

Il foglio è rappresentato da una matrice di dimensione $20 \times 20$ di interi inizializzati a zero (foglio bianco). Le caselle del foglio sulle quali passa la matita assumeranno il valore 1. La matita parte dalla posizione `0,0`.

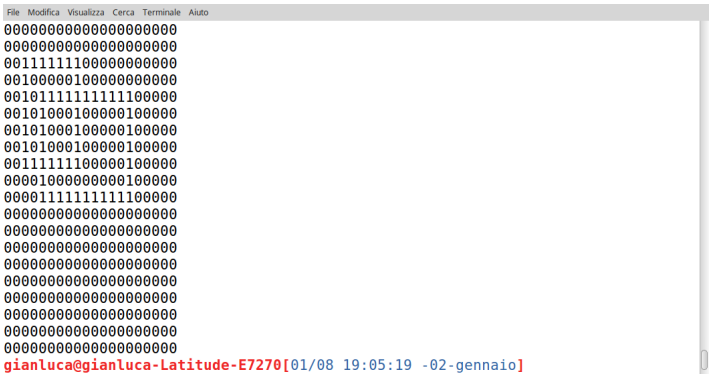
La funzione da progettare deve prendere in input la matrice che rappresenta il foglio, un array di stringhe ognuna delle quali contiene un comando e la dimensione dell'array dei comandi. In particolare la funzione deve avere il seguente prototipo:

```
void Logo( int canvas[20][20], char *commands[], int n );
```

dove `canvas` è la matrice che rappresenta il foglio, `commands` è l'array di stringhe ognuna delle quali è un comando come descritto precedentemente ed `n` è la sua dimensione. La funzione interpreta i comandi in `commands` e modifica `canvas` di conseguenza. Nell'interpretare i comandi la funzione deve ignorare quelli sintatticamente errati (ad esempio `e11` invece di `E11`) e quelli che portano la matita fuori dal foglio. La matrice `canvas` modificata sarà il risultato della funzione. Ad esempio, se `commands` contiene le stringhe

```
G2,2 S6 E6 N6 W6 r1 G4,10 W12 N6 E10 S6 W10
```

il contenuto di `canvas` deve essere




la stringa `r1` non è un comando quindi è ignorata ed il comando `W12` non viene eseguito perché porta la penna oltre il foglio.

**Modalità di consegna:** Lo studente deve consegnare **un unico** file denominato `CognomeNome.c` (dove `Cognome` e `Nome` stanno rispettivamente per il proprio cognome ed il proprio nome).

Tale file, con l'aggiunta di una opportuna funzione `main` che non deve essere contenuta nel file consegnato, deve poter essere compilato senza errori. Quindi deve contenere:

- la funziona richiesta (`Logo()`) che a sua volta deve rispettare le specifiche imposte dal problema;
- ogni altra funzione utilizzata dalla soluzione in quanto non è permesso l'utilizzo di funzioni non definite all'interno del file consegnato;
- tutti gli *header* delle librerie utilizzate.

Infine la funzione `main()` **non** deve essere inclusa nel file `CognomeNome.c` pertanto si consiglia di definirla in un secondo file denominato `main.c`. I due file possono essere compilati insieme utilizzando il comando

```
gcc main.c CognomeNome.c
```

**Inviare le soluzioni al seguente indirizzo email entro il 15 gennaio 2018**.