Università di Roma "La Sapienza"
Dispense del corso di
CALCOLATORI ELETTRONICI
Ing. Elettronica, Ing. delle Telecomunicazioni
Anno Accademico 1998-1999

Esercizi d'esame svolti di programmazione in C e selezione di domande d'esame

a cura di
Silvio Salza, Giuseppe Santucci, Andrea Schaerf

29 luglio 2002

Indice

Indice

Introduzione

Modalità d'esame

- L'esame di Calcolatori Elettronici è suddiviso in una prova scritta ed una orale da sostenere in un unico appello.
- La prova scritta è composta da due parti:
  - tre esercizi e tre domande di teoria a risposta multipla(1 ora e mezza di tempo);
  - prova di programmazione in C (1 ora e mezza di tempo).
- La prova orale consiste in una discussione della prova scritta.

Durante le prove scritte non è consentito portare in aula e consultare testi o appunti ed usare calcolatori e/o calcolatrici tascabili.

Inoltre, per quanto riguarda la possibilità di sostenere più volte l'esame nel corso di uno stesso anno, esistono le seguenti limitazioni:

- la prima volta che il candidato sostiene l'esame è considerata come colloquio, e quindi è concesso allo studente di ritirarsi senza che a ciò consegua una verbalizzazione;
- a parte il colloquio verrà applicata la normativa vigente, che consente di sostenere l'esame al più una volta per sessione e due per anno accademico;
- il fatto di rispondere all'appello e di ricevere il testo d'esame equivale a tutti gli effetti ad aver sostenuto l'esame.

Note sullo svolgimento della prova in C

Il compito scritto di C si compone di 2 esercizi e due domande a risposta multipla. Durante il compito non è consentito consultare libri, appunti, dispense, o qualsiasi altro materiale e non è consentito uscire dall'aula.

Nel tempo che intercorre tra il compito scritto e la prova orale (tipicamente una settimana) lo studente è tenuto a compilare ed eseguire la soluzione dei due esercizi del compito C.

La soluzione proposta a casa deve essere la stessa di quella svolta nel compito, e la sua correttezza deve essere verificata su dei dati di test scelti opportunamente. Nel caso la soluzione svolta in classe non sia corretta, questa deve essere modificata opportunamente fino ad ottenere il corretto funzionamente su tutti i dati di test.

Nel caso lo studente non abbia svolto un esercizio nel compito, la soluzione svolta a casa può, ovviamente, essere qualsiasi.

Lo studente deve presentarsi all'orale *obbligatoriamente* con un dischetto contenente i file che compongono la nuova soluzione e con la stampa dei file stessi e dei test effettuati.

La parte orale è costituita essenzialmente da una discussione del compito scritto e della soluzione svolta a casa dallo studente. Scopo principale della soluzione sviluppata a casa è quello di effettuare una autovalutazione precedente all'orale, che porta ad una discussione (e valutazione) dello scritto più proficua. In particolare, tale soluzione serve allo studente per mostrare che, a parte eventuali modifiche, la soluzione proposta nel compito è corretta.

Materiale didattico in formato elettronico

Il codice dei programmi contenuti in questa dispensa può essere reperito in formato elettronico in uno dei seguenti modi:

- Centro di calcolo della Facoltà, disco `i:`, nel direttorio `cal_elet\esami`
- Accesso alle pagine www:

  Santucci (elettronici)**:**
  `http://www.dis.uniroma1.it/~santucci/didattica.html`

  Schaerf (telecomunicazioni)**:**
  `http://www.dis.uniroma1.it/~aschaerf/didattica.html`

- `ftp` anonimo al sito `ftp.dis.uniroma1.it` direttorio `/pub/santucci/CE`

I direttori che contengono i compiti sono identificati della data dello scritto, e ciascun direttorio contiene 4 file con le seguenti modalità:

- `testo.txt`: testo del compito
- `compito1.c`: soluzione del primo esercizio
- `compito2.c`: soluzione del secondo esercizio
- `compito3.c`: soluzione del terzo esercizio

Negli stessi siti è anche reperibile altro materiale relativo al programma svolto in aula. È inoltre disponibile il compilatore C `DJGPP` della `GNU` di dominio pubblico (occorrono 3 dischetti per copiarlo).

Programma C 1998-1999

Espressioni, istruzioni, dichiarazioni. Strutture di ciclo. Input/Output da tastiera e video. Funzioni. Tipi primitivi. Vettori. Puntatori. Puntatori vs. vettori. Allocazione dinamica della memoria. Matrici. Stringhe. Record. Record e puntatori. File: apertura, chiusura, lettura e scrittura.

Utilizzo della libreria C standard

Nella soluzione degli esercizi abbiamo utilizzato soltanto le funzionalità di base del linguaggio C che sono state spiegate in classe. In particolare, le funzioni di libreria che vengono utilizzate appartengono tutte al seguente elenco di funzioni.

Allocazione dinamica:

- `malloc` (oppure `calloc`)
- `realloc`
- `free`

Gestione di stringhe:

- `strlen`
- `strcpy`
- `strcmp`

- `strcat`

Input/Output:

- `scanf` e `fscanf`
- `printf` e `fprintf`
- `fopen` (modalità `"r"`, `"w"` e `"a"`) e `fclose`
- `rewind`

Varie

- `exit`
- `sizeof`

Il presente elenco costituisce l'insieme di funzioni che è indispensabile conoscere e saper gestire correttamente per poter affrontare l'esame.

Alcuni esercizi possono essere risolti facendo ricorso anche ad altri costrutti del linguaggio C ed altre funzioni della libreria standard del C. Lo studente che abbia padronanza di tali strumenti è libero di farne uso per la soluzione del compito d'esame.

Note sulle soluzioni proposte

Ognuno degli esercizi presentati può essere risolto in molti modi diversi. La soluzione proposta in questa dispensa è solo una delle possibili scelte.

Nella soluzione abbiamo cercato di privilegiare la semplicità e la leggibilità dei programmi. Non abbiamo quindi posto enfasi sull'efficienza, evitando però che questa venisse eccessivamente sacrificata.

Nella soluzione vengono anche incluse alcune funzioni che non sono richieste per il compito scritto, ma che sono necessarie per poter compilare, eseguire e verificare il programma. Tali funzioni sono quindi di ausilio per lo svolgimento degli esercizi a casa per la preparazione al compito.

Per semplicità, le funzioni che risolvono l'esercizio, le funzioni ausiliarie e la funzione `main()` vengono poste in un unico file (con estensione `.c`). Una corretta progettazione del software richiederebbe invece che tali funzioni fossero poste in file separati e fossero corredate di un file *header* (estensione `.h`) che raccolga le dichiarazioni delle funzioni. In questo modo le funzioni sviluppate potrebbero essere compilate separatamente ed essere utilizzate da programmi diversi.

Inoltre, nel file contenente la soluzione viene incluso unicamente il file `stdio.h`. Per alcuni esercizi in alcuni compilatori può essere necessario includere anche i file `malloc.h` e `string.h`.

Capitolo 1

Esercizi su vettori, stringhe e liste

Esercizio 2 del 1-7-96

Una sequenza di 27 interi si dice *perfetta* se consiste di tre 1, tre 2, ..., tre 9, posizionati in modo che per ogni $i \in [1..9]$ ci sono esattamente i numeri tra le occorrenze successive di i. Ad esempio, la sequenza

`(1,9,1,2,1,8,2,4,6,2,7,9,4,5,8,6,3,4,7,5,3,9,6,8,3,5,7)`

è perfetta. Si scriva una funzione C che prende come parametro un vettore di 27 interi e restituisce 1 se il vettore contiene una sequenza perfetta e 0 altrimenti. Soluzione a pag. 27

Esercizio 2 del 15-7-96

Si assuma presente in memoria una lista composta di abbreviazioni (ad esempio TO, MI, RM) e dalle corrispondenti parole estese implementata tramite record e puntatori utilizzando la seguente struttura:

```
struct elem {

        char abbr[2];

        char *estesa;

        struct elem *next;

        }
```

Scrivere una funzione C che, ricevendo come parametri il puntatore all'inizio della lista, una abbreviazione e la corrispondente parola estesa cerchi l'abbreviazione nella lista e:

a) restituisca 0 se la coppia è presente nella lista

b) restituisca 1 se la coppia non è presente nella lista

c) restituisca 2 se l'abbreviazione è presente ma ad essa corrisponde una parola differente.

Inoltre, nel caso b) un nuovo record deve essere aggiunto in coda alla lista e nel caso c) la parola estesa deve essere sostituita da quella passata come parametro. Soluzione a pag. 28

Esercizio 1 del 17-1-97

Si assuma presente in memoria un vettore di puntatori a liste di valori interi positivi. Le liste sono realizzate tramite record del seguente tipo:

```
struct nodo
{
   int info;
   struct nodo *next;
};
```

Scrivere una funzione C che prende come parametri il vettore e la sua dimensione, e restituisce l'indice della locazione del vettore contenente il puntatore alla lista in cui compare l'elemento di valore massimo tra gli elementi di tutte le liste. Nel caso tutte le liste siano vuote oppure il vettore abbia dimensione 0, la funzione deve restituire 0. Soluzione a pag. 30

Esercizio 1 del 30-1-97

Si vuole gestire una classe di studenti tramite un vettore di dimensione variabile di record, dove i record hanno la seguente struttura:

```
struct elem
{
   char *nome;
   int eta;
};
```

Scrivere una funzione C che prende come parametro il numero di studenti da inserire e che restituisce l'indirizzo del vettore leggendo da tastiera i nomi (non più lunghi di 20 caratteri) e l'età degli studenti. Gestire eventuali errori nella fase di allocazione di memoria, restituendo `NULL` in caso di errori, l'indirizzo del vettore in caso di successo. Soluzione a pag. 32

Esercizio 2 del 20-2-97

Si considerino stringhe di lunghezza qualsiasi che contengono soltanto caratteri alfabetici minuscoli e spazi bianchi. Una stringa si dice *palindroma* se leggendola da destra verso sinistra ignorando gli spazi bianchi è uguale a se stessa. Ad esempio, le stringhe `"aerea"` e `"etna gigante"` sono palindrome.

Si consideri un vettore (di lunghezza ignota) di stringhe terminato dalla stringa vuota `""`.

Si scriva una funzione C che prende come parametro tale vettore e restituisce l'*ultima* stringa palindroma presente nel vettore. Nel caso il vettore non contenga stringhe palindrome, la funzione deve restituire il valore `NULL`.

È inoltre richiesto che la stringa restituita non condivida memoria con il vettore di stringhe. Soluzione a pag. 32

Esercizio 1 del 15-4-97

Si assuma presente in memoria una lista (di lunghezza ignota) di record. I record della sequenza hanno la seguente struttura:

```
struct elem
{
   char *nome;
   struct elem *next;
};
```

Scrivere una funzione C che prende come parametro il puntatore iniziale alla lista, e restituisce un vettore di puntatori, ciascuno dei quali punta ad un record della lista.
Soluzione a pag. 34

Esercizio 3 del 15-4-97

Si scriva una funzione C che prende come parametri una stringa ed un carattere e restituisce un'*altra* stringa, da cui sono state rimosse tutte le occorrenze del carattere. Ad esempio, se la funzione viene chiamata con parametri `"tutta statistica"` e `'t'` deve restituire la stringa `"ua saisica"`.

La lunghezza della stringa di ingresso non è nota. È richiesto di non sprecare memoria. In particolare, la memoria allocata per la stringa in uscita deve essere esattamente uguale a quella necessaria per la sua memorizzazione. Si richiede inoltre di gestire eventuali errori nella fase di allocazione di memoria, restituendo `NULL` in caso di insuccesso.
Soluzione a pag. 35

Esercizio 2 del 17-6-97

Sia dato un vettore di dimensione nota di stringhe contenenti tre o più caratteri (escluso il terminatore di stringa). Si vuole aggiungere ad ogni stringa s nel vettore la prima di quelle seguenti che abbia *esattamante* le prime due lettere iniziali in comune con le due finali di s, evitando di ripetere le due lettere in comune. Nel caso nessuna delle stringhe seguenti abbia due lettere in comune con s non bisogna modificare s.

Ad esempio, se il vettore contiene le seguenti stringhe:

```
casa
postino
sasso
osteria
salvia
notare
renna
```

dovrà essere trasformato nel seguente vettore:

```
casasso
postinotare
sasso
osteria
salvia
notarenna
renna
```

Si scriva una funzione C che ricevendo in ingresso l'indirizzo del vettore e la sua dimensione effettui la trasformazione di cui sopra.
Soluzione a pag. 36

Esercizio 2 del 1-7-97

Una stringa contiene nome e cognome di una persona separati tra loro da uno o più spazi. Una seconda stringa contiene il soprannome di una persona, e può eventualmente contenere degli spazi.

Si scriva una funzione C che prende in ingresso due stringhe del tipo suddetto e restituisce una nuova stringa in cui tra il nome ed il cognome è inserito il soprannome tra parentesi tonde. Nella nuova stringa, sia tra il nome e la parantesi aperta che tra la parentesi chiusa ed il cognome deve essere presente uno (ed un solo) spazio.

Ad esempio, se le due stringhe sono `"Bruce Springsteen"` e `"The Boss"`, la stringa restituita deve essere: `"Bruce (The Boss) Springsteen"`. Soluzione a pag. 36

Esercizio 2 del 14-7-97

Sia data una lista contenente almeno due elementi ed i cui record sono definiti tramite la seguente struttura C:

```
struct nodo{
       int valore;
       struct nodo* next;
       };
```

Si scriva una funzione C che ricevendo in ingresso un puntatore alla lista modifichi la stessa, memorizzando nell'ultimo nodo il prodotto del penultimo ed ultimo nodo, nel penultimo il prodotto del terzultimo e del penultimo e così via. Il primo record non deve essere modificato.

Ad esempio, una lista contenente la sequenza di interi `4 6 2 3 9` verrà modificata dalla funzione nella lista `4 24 12 6 27`. Soluzione a pag. 38

Esercizio 2 del 17-9-97

Un numero intero positivo *di lunghezza qualsiasi* viene rappresentato tramite una stringa di caratteri numerici (cioè caratteri compresi tra `'0'` e `'9'`), con la cifra più significativa nella posizione 0 della stringa.

Si scriva una funzione C che prende in ingresso due stringhe che rappresentato due interi posiviti nel modo suddetto e restituisce una nuova stringa che rappresenta (nello stesso modo) la somma dei due numeri dati.

Si supponga che le stringhe in ingresso siano di lunghezza uguale tra loro. Si assuma inoltre che la somma della cifra più significativa dei due numeri non generi riporto, e che, di conseguenza, la stringa risultato sia esattamente della stessa lunghezza delle stringhe date.

Ad esempio, se in ingresso vengono fornite le stringhe `"6795241135292314"` e `"2314332174634521"` il risultato deve essere la stringa `"9109573309926835"`.

Suggerimento: Si ricordi che i codici ASCII dei caratteri numerici sono consecutivi tra loro. Di conseguenza il numero corrispondente ad un carattere numerico `ch` viene ottenuto con `ch - '0'`. Soluzione a pag. 39

Esercizio 2 del 21-10-97

Sia dato un vettore di interi presente in memoria. Si scriva una funzione C che ricevendo il vettore e la sua dimensione restituisca la media degli interi presenti nel vettore non con-

siderando gli eventuali duplicati. Ad esempio, se il vettore contiene gli interi `7 6 4 6` la funzione deve restituire il valore 5.6, ovvero la media di `7`, `6` e `4`. Soluzione a pag. 40

Esercizio 2 del 19-1-98

Un numero in una base b, con $1 \leq b \leq 36$, può essere scritto utilizzando i primi b caratteri dell'insieme {`'0'`,..., `'9'`, `'A'`, ..., `'Z'`}. Ad esempio, un numero in base 24 può essere espresso utilizzando i caratteri {`'0'`,..., `'9'`, `'A'`,..., `'N'`}.

Si considerino stringhe composte da una sequenza di numeri in base b separati tra loro dal carattere `'-'`. Come esempio, si consideri la seguente stringa `"4FF-D4-F-345"`, contenente i numeri `4FF`, `D4`, `F` e `345`.

Si scriva una funzione C che prende in ingresso una stringa nel formato suddetto ed un intero rappresentate la base e restituisce il numero massimo presente nella stringa (come `int`). Nel caso un numero contenga un simbolo non ammesso dalla base, la funzione deve restituire `-1`.

Ad esempio, se la stringa in input è quella suddetta e la base è `16`, allora la funzione deve restituire `1279` (corrispondente al valore di `4FF` come numero esadecimale). Se invece la base è `20` la funzione deve restituire `1915` (corrispondente al valore di `4FF` come numero in base `20`). Infine se la base è `14`, la funzione deve restituire `-1` in quanto il carattere `'F'` presente nella stringa è fuori dalla base.

Sono disponibili le funzioni di libreria `isupper(char)`, che verifica se il carattere è una lettera maiuscola, e `isdigit(char)`, che verifica se un carattere è una cifra. Inoltre, si può utilizzare la seguente funzione.

```
int valore(char ch)  /* restituisce il valore corrispondente al
carattere ch */
{ if (isdigit(ch)) return ch - '0';
  else if (isupper(ch)) return ch - 'A' + 10;
  else return -1;
}
```

Soluzione a pag. 41

Esercizio 2 del 19-2-98

Sia dato un vettore di caratteri presente in memoria. Si scriva una funzione C che ricevendo in ingresso il vettore e la sua dimensione restituisca, tramite passaggio di parametri, il carattere che più frequentemente degli altri è seguito dal carattere successivo nell'ordine alfabetico. Ad esempio, se il vettore contiene i caratteri `A F L M P S T L M` la funzione dovrà restituire il carattere L, che per due volte è seguito dal carattere M. Soluzione a pag. 42

Esercizio 2 del 16-4-98

Sia dato un vettore contenente una sequenza ordinata di interi ed allocato dinamicamente. Si scriva una funzione C che ricevendo in ingresso il vettore, la sua dimensione ed un intero, modifichi il vettore in questione inserendo il nuovo intero nella posizione corretta. Ad esempio, se il vettore v contiene gli interi `4 32 57 98` la funzione chiamata con i parametri $v, 4, 50$ dovrà modificare v in un vettore di 5 elementi contenente `4 32 50 57 98`.
Soluzione a pag. 42

Capitolo 2

Esercizi sui file

Esercizio 1 del 3-6-96

Un file contiene una sequenza di stringhe formate da ripetizioni dell'unico carattere '`*`', separate tra loro da uno o più spazi e ritorni a capo. Le stringhe rappresentano una sequenza di interi positivi codificati in *codice unario*. Ad esempio, il file

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

corrisponde alla sequenza $\langle 7, 6, 2, 16\rangle$. Scrivere una funzione C che prende come parametro il nome del file e lo modifica appendendo un fondo al file la media (rappresentata in unario ed arrotondata per difetto) dei valori contenuti nel file. Nell'esempio precedente, all'uscita della funzione il file deve essere il seguente

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

in cui il valore 7 (cioè la media arrotondata per difetto dei valori 7, 6, 2 e 16) è stato appeso alla fine del file.

Si assuma che ogni riga contenga al più 80 caratteri e, conseguentemente, le stringhe siano composte di al massimo 80 caratteri '`*`'. Si assuma inoltre che i dati nel file siano corretti, ma che il file di ingresso possa non esistere. Soluzione a pag. 44

Esercizio 2 del 3-6-96

Il file "`matrice.dat`" contiene la matrice di interi quadrata A di dimensione $n \times n$. La matrice A è memorizzata nel file per righe ed ogni riga del file contiene una riga della matrice. I valori sono separati da spazi e le righe sono terminate dal carattere '`#`' preceduto da uno spazio bianco; inoltre il file non contiene alcun'altra informazione oltre ai valori degli elementi della matrice. Il seguente file, ad esempio, contiene una matrice 3×3 nel formato descritto.

```
 3  4  -11  #
-1  0    0  #
12  4    4  #
```

Si scriva una funzione C che prende come parametro la dimensione n della matrice e verifica se la matrice è *simmetrica*. Si ricordi che una matrice $A_{n\times n}$ è simmetrica se $a_{ij} = a_{ji}$ $(\forall i, j \in 1..n)$. Si assuma che i dati nel file siano corretti e che il file di ingresso esista sempre.

Nota: è richiesto che la funzione deallochi lo spazio di memoria eventualmente allocato al suo interno.

Soluzione a pag. 45

Esercizio 2 del 17-6-96

Un file (sicuramente presente sul disco) contiene una sequenza di parole. Le parole sono lunghe al massimo 80 caratteri e sono separate o da uno spazio o da un ritorno a capo. Scrivere una funzione C che ricevendo in ingresso il nome del file e l'indirizzo di un puntatore a carattere restituisca in tale puntatore una stringa contente la giustapposizione di tutte le parole presenti nel file.

Soluzione a pag. 46

Esercizio 1 del 1-7-96

Un file contiene una sequenza di stringhe alcune delle quali sono della forma `#`i, con i valore intero non negativo (si supponga che il carattere `#` non compaia nelle altre stringe del file). Come esempio si consideri il seguente file

```
Il sottoscritto #0, nato a #1 il #2, chiede di poter #3 #8

                              Firma #11
                              #0
Roma, li‘ #4
```

Si scriva una funzione C che prende come parametri: (1) il valore n, (2) il nome del file di input contenente il testo suddetto, (3) il nome di un file di output (4) l'indirizzo iniziale `punt_dati` di un vettore di stringhe di dimensione n già presente in memoria, e scriva nel file di output il testo ottenuto sostituendo ad ogni stringa della forma `#`i del file di input la stringa corrispondente alla locazione i del vettore.

Le stringhe della forma `#`i con $i \geq n$ devono essere sostituite tutte con la stessa stringa `"..."`. Ad esempio, se $n = 5$ e `punt_dati = {"Paolo Rossi", "Torino", "13/7/71", "essere ammesso", "15/6/96"}` allora il corrispondente file di output deve contenere il seguente testo

```
Il sottoscritto Paolo Rossi, nato a Torino il 13/7/71, chiede di poter essere ammesso ...

                              Firma ...
                              Paolo Rossi
Roma, li‘ 15/6/96
```

Soluzione a pag. 47

Esercizio 1 del 15-7-96

Si assuma presente in memoria secondaria un file contenente le informazioni relative alle verbalizzazioni di un esame. Il file contiene le coppie nome-voto tra parentesi; il nome è separato dal voto da uno spazio, le coppie tra parentesi sono separate da spazi o ritorni a capo. Ad esempio, il contenuto del file potrebbe essere il seguente:

```
(gianni 27) (marco 28)
(luigi 20)
(giovanni 25) (sergio 24)
(luisa 29)
```

Scrivere una funzione C che ricevenco come parametro il nome del file stampi la media dei voti ottenuti dagli studenti ed il nome e il voto dello studente col voto piu' alto. Se più studenti condividono il voto più alto si deve stampare il nome del primo studente tra quelli che hanno il voto più alto. Soluzione a pag. 48

Esercizio 1 del 17-9-96

I tassi di cambio giornalieri di un insieme di valute straniere vengono memorizzati su dei file nel formato specificato tramite il seguente esempio.

```
Germania     Marco       1029.83
USA          Dollaro     1526.72
RegnoUnito   Sterlina    2378.94
Francia      Franco       303.61
```

Si supponga che in due file distinti siano memorizzati i tassi di cambio in due giorni successivi e che le valute nei due file siano le stesse e compaiano nello stesso ordine.

Si scriva una funzione C che prende come parametri i nomi dei due file ed il nome di una valuta (`Marco`, `Dollaro`, ...), e restituisce una stringa che descriva l'andamento della valuta. In particolare, la funzione deve restituire una delle seguenti tre stringhe:

`"In Rialzo"` se la valuta è salita di almeno 1%

`"In Ribasso"` se la valuta è scesa di almeno 1%

`"Stabile"` se il valore della valuta è variato di meno di 1%

Si supponga che i dati in ingresso siano corretti: i file siano presenti in memoria, il loro contenuto sia identico a meno dei valori dei tassi di cambio e la valuta sia presente nei file.
Soluzione a pag. 49

Esercizio 1 del 17-10-96

Si vuole scrivere una funzione per un correttore automatico, in grado di rimuovere gli errori di battitura che abbiano generato parole con tre o piu' *consonanti* uguali adiacenti eliminando tutte le occorrenze della consonante tranne 2 (ad esempio gatttto deve diventare gatto). Scrivere una funzione C che, ricevendo come parametri i nomi di due file, copi il primo nel secondo, correggendo automaticamente gli errori di cui sopra. La funzione non deve effettuare nessun'altra modifica.

`Esempio:` la chiamata `correggi("pippo","pluto")`, essendo il contenuto di `pippo` il seguente:

```
Nel mezzzo del cammin

di nostra     vita...
```

produce il file `pluto` contenente:

```
Nel mezzo del cammin

di nostra     vita...
```

Si assuma di avere a disposizione la funzione `int consonante(char)` che restituisce 0 se il parametro non é una consonante ed 1 in caso contrario. Soluzione a pag. 50

Esercizio 2 del 17-1-97

Un file contiene una matrice di reali di dimensione $n \times m$. La prima riga del file contiene le dimensioni n ed m della matrice. Le righe successive del file contengono la matrice memorizzata per righe, in modo tale che ogni riga del file contiene una riga della matrice. I valori sono separati da uno o più spazi. Il seguente file, ad esempio, contiene una matrice nel formato descritto.

```
4 5
1.22   2.31   1.00   0.23  1.21
1.12   1.14   1.10   1.01  1.11
1.22   3.34   2.00   0.40  0.01
1.32   3.02   1.30   0.01  1.22
```

Si scriva una funzione C che prende come parametro il nome del file (supposto correttamante presente sul disco) e restituisce il valore *minimo* tra gli elementi della colonna la cui somma dei valori degli elementi è *massima*.

Nell'esempio, la funzione deve restituire il valore `1.14`, essendo questo il minimo tra quelli della seconda colonna, la cui somma, `9.81` (ottenuta da `2.31 + 1.14 + 3.34 + 3.02`), è la massima tra quelle delle colonne della matrice.

Nota: è richiesto che la funzione deallochi lo spazio di memoria eventualmente allocato al suo interno. Soluzione a pag. 51

Esercizio 2 del 30-1-97

Un file contiene una lista di telefonate effettuate da un utente. La prima riga del file contiene il numero di telefonate effettuate. Ciascuna riga successiva contiene la città di destinazione della telefonata, la durata in secondi ed il numero di scatti. I dati sono separati da uno o più spazi bianchi. Un possibile esempio è il seguente:

```
3
ROMA   230   5
MILANO  30   3
ROMA    20   1
```

Si scriva una funzione C che prende come parametro il nome del file (supposto correttamante presente sul disco) ed il nome di una città e restituisce, relativamente alla città in questione, la durata della telefonata più lunga ed il numero *totale* di scatti. Soluzione a pag. 53

Esercizio 1 del 20-2-97

Un file contiene il numero di presenze degli studenti alle lezioni di un corso. In particolare, il file è composto da una sequenza di elementi separati da una virgola, e (dopo la virgola) da uno o più spazi e/o ritorni a capo). Come esempio si consideri il seguente file.

```
      139 (13/03),   123    (24/03), 94
  (15/03),    133  (21/02), 118    (22/04),
       116      (24/04)
```

Il valore intero rappresenta il numero di studenti presenti, e l'espressione tra parentesi tonde rappresenta giorno e mese della lezione (si suppone che le lezioni abbiano luogo tutte nello stesso anno solare). Si noti che il file non è ordinato.

Si scriva una funzione C che prende come parametro il nome del file (sicuramente presente su disco) e restituisce il numero di studenti della prima lezione del corso (cioè la lezione la cui data è minima tra quelle presenti nel file). Nell'esempio la funzione deve restituire `133`, cioè il numero di presenze del 21 febbraio.

La funzione deve restituire 0 nel caso che qualche data abbia contenuto non valido: valore del mese maggiore di 12, oppure valore del giorno maggiore del numero di giorni del mese (ad esempio `23/14`, `31/06`). A questo scopo si supponga disponibile già realizzata una funzione `int GiorniDelMese(int mese)` che dato il mese restituisce il numero di giorni di quel mese.

Soluzione a pag. 54

Esercizio 2 del 15-4-97

Un file contiene un insieme di parole di lunghezza minima pari a 2 e massima pari a 20, separate tra di loro da spazi e/o ritorni a capo. Un possibile esempio è il seguente file (dove il carattere ␣ indica lo spazio):

```
Questo␣e'␣un␣␣␣␣␣␣esempio

di␣file


compatibile␣con␣le␣␣␣␣␣specifiche
```

Si scriva una funzione C che prende come parametro il nome del file (supposto correttamante presente sul disco) e restituisce, con modalità scelte dallo studente:

- la parola di lunghezza massima (se più d'una, la prima);
- il numero totale di parole;
- la lunghezza della sequenza massima di spazi consecutivi.

Relativamente all'esempio, la funzione deve restituire la parola `compatibile`, 10 e 6 (numero di spazi tra `un` ed `esempio`).

Soluzione a pag. 55

Esercizio 1 del 3-6-97

Un file contiene una sequenza di parole composte di caratteri (al più 20) alfabetici minuscoli. Le parole sono separate tra loro da un *punto e virgola* e da uno o più spazi e/o ritorni a capo. La sequenza contiene almeno una parola ed è terminata da un punto. La sequenza appare all'inizio del file, il quale può contenere anche altri caratteri dopo il punto, che non sono significativi. Come esempio si consideri il seguente file.

```
letto;    comodino; lampada;
  armadio;
tappeto.    casa, 123! rrrr;
```

La sequenza corrispondente è composta dalle parole: `letto`, `comodino`, `lampada`, `armadio` e `tappeto`

Si scriva una funzione C che prende come parametri:

1. il nome del file
2. l'indirizzo iniziale di un vettore di stringhe contenente un insieme di parole
3. la lunghezza del vettore di stringhe,

e restituisce `1` se tutte le parole della sequenza nel file compaiono anche nel vettore, restituisce `0` se invece almeno una di esse non compare nel vettore.

Si assuma che il file sia presente su disco e che la sequenza sia corretta. Soluzione a pag. 56

Esercizio 2 del 3-6-97

Un file contiene una sequenza (di lunghezza ignota, possibilmente nulla) di valori reali separati da uno spazio. Come esempio si consideri il seguente file

```
4.522 5.32 4 5.001 16.2 34.2 45.6
```

Si scriva una funzione C che prende come parametro il nome del file e *sostituisce* completamente il contenuto del file stesso con i valori interi ottenuti per arrotondamento (all'intero più vicino) dei valori reali presenti nel file.

Ad esempio, dopo l'esecuzione della funzione, il contenuto del file dell'esempio deve essere

```
5 5 4 5 16 34 46
```

Se necessario, si utilizzi la funzione `int Arrotonda(float)` che restituisce l'intero più vicino al valore del parametro di tipo `float`. Si assuma che il file sia presente su disco e che la sequenza sia corretta. Soluzione a pag. 57

Esercizio 1 del 17-6-97

Un file, sicuramente presente su disco, contiene una sequenza di parole, ciascuna di lunghezza massima pari a 20. Le parole sono composte esclusivamente da caratteri alfabetici minuscoli ('a' ... 'z') e sono separate tra di loro da spazi e/o ritorni a capo.

Si scriva una funzione C che prende come parametro il nome del file e restituisce (tramite una struttura dati scelta dallo studente) il numero di occorrenze dei caratteri componenti solo le parole lunghe più di sei lettere.

Ad esempio, considerando il seguente file:

```
questo file

       contiene

due parole significative
```

la funzione dovrà restituire al programma chiamante e nelle modalità scelte dallo studente la seguente informazione: $\langle c : 2\rangle, \langle o : 1\rangle, \langle n : 3\rangle, \langle t : 2\rangle, \langle i : 5\rangle, \langle e : 3\rangle, \langle s : 1\rangle, \langle g : 1\rangle, \langle f : 1\rangle, \langle a : 1\rangle, \langle v : 1\rangle$. Soluzione a pag. 58

Esercizio 1 del 1-7-97

Un file contiene una sequenza di terne $\langle i, j, x \rangle$ con i e j interi positivi e x reale, separate tra loro da un punto e virgola e da uno o più spazi, come mostrato nel seguente esempio.

```
1 1 3.4;  3 4 4.56;   2 2 7.03;    3 5 7.455;
```

La sequenza rappresenta una matrice reale quadrata M con indici da 1 ad n, in modo tale che ogni terna $\langle i, j, x \rangle$ denota che la locazione M_{ij} ha valore x (si assuma $1 \leq i \leq n$ e $1 \leq j \leq n$). Le locazioni della matrice non contenute nella sequenza hanno tutte valore 0.

Si scriva una funzione C che prende come parametri i nomi di due file e un intero. Il primo file contiene la rappresentazione suddetta della matrice e l'intero denota la dimensione n della matrice. La funzione deve scrivere nel secondo file la rappresentazione estesa della matrice per righe (cioè una riga della matrice su ciascuna riga del file). Ad esempio, se $n = 5$ e il file di ingresso è quello mostrato precedentemente, il file di uscita dovrà essere

```
3.400 0.000 0.000 0.000 0.000
0.000 7.030 0.000 0.000 0.000
0.000 0.000 0.000 4.560 7.455
0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000
```

Soluzione a pag. 58

Esercizio 1 del 14-7-97

Esercizio 1 Un file contiene una sequenza di parole, ciascuna di lunghezza massima pari a 20. Le parole sono separate tra di loro da spazi e/o ritorni a capo.

Si scriva una funzione C che riceve come parametri il nome del file e due interi (i e j) e restituisce la giustapposizione della i-esima parola con la j-esima, separandole con uno spazio. Nel caso uno dei due interi sia maggiore del numero di parole presenti nel file, la stringa dovrà contenere la parola `MANCANTE` al posto della parola corrispondente. Ad esempio, considerando il seguente file:

```
    roma  milano

  torino
genova
```

la funzione, una volta chiamata con i valori 2 e 6, dovrà restituire la stringa `"milano MANCANTE"`.

Soluzione a pag. 59

Esercizio 1 del 17-9-97

Un file contiene una lista di alberghi con il loro prezzo e la lista di servizi che l'albergo mette a disposizione. In particolare, il file ha il seguente formato

```
albergo       prezzo    tv   bagno  condizionatore frigo_bar
miramonti     33000     SI     NO        SI           NO
olimpia       66000     NO     SI        SI           SI
rosa          54000     SI     NO        SI           SI
```

in cui la prima riga contiene le parole `albergo` e `prezzo` e la sequenza di servizi, e le righe successive contengono il nome di un albergo il suo prezzo e la disponibilità del servizio: `SI` denota la presenza del servizio e `NO` denota la sua assenza.

Si scriva una funzione C che prende in ingresso il nome del file e il nome di un servizio e restituisce il prezzo dell'albergo che ha quel servizio ed ha il prezzo minimo rispetto agli alberghi che hanno tale servizio.

Nell'esempio, se il servizio richiesto è `frigo_bar`, allora la funzione deve restituire il valore `54000`. Se invece il servizio richiesto è `condizionatore`, la funzione deve restituire il valore `33000`.

Si supponga il file sicuramente presente su disco. Si assuma inoltre che il servizio richiesto sia sicuramente elencato nel file e che esista almeno un albergo nel file che offre tale servizio. Si assuma, infine, che i nomi dei servizi e degli alberghi siano lunghi al più 20 caratteri.

Soluzione a pag. 60

Esercizio 1 del 21-10-97

Un file contiene una sequenza di parole, ciascuna di lunghezza massima pari a 20. Le parole sono separate tra di loro da spazi e/o ritorni a capo e rappresentano un messaggio "cifrato": ad ogni parola sono aggiunti all'inizio ed alla fine due caratteri fittizi che permettono di trovare la parola precedente e successiva del messaggio. Si assuma che:

- la prima parola nel file sia anche la prima del messaggio;
- non esistano due parole che cominciano con gli stessi due caratteri;
- il file possa contenere parole che non fanno parte del messaggio;
- l'ultima parola del messaggio termini con un punto.

Si scriva una funzione C che ricevendo come parametro il nome del file crei il file `frase.txt` contenente il messaggio corretto. Ad esempio, considerando il seguente file:

```
 Esempioab  bzcifrato.
ijmessaggiobz
paperino
 abdiij
llyuhghl
       minni
```

la funzione C dovrà creare il seguente file:

```
Esempio di messaggio cifrato.
```

Soluzione a pag. 62

Esercizio 1 del 19-1-98

Un file contiene il rapporto tra deficit e PIL (prodotto interno lordo) per una lista di nazioni europee. Ogni riga del file contiene il nome della nazione (massimo 20 caratteri) ed il rapporto suddetto. Per ogni nazione il rapporto può essere espresso tramite uno qualsiasi dei seguenti tre modi:

- un valore reale assoluto

- una frazione tra interi, cioè due interi separati dal carattere `/`
- un valore reale percentuale seguito dal carattere `%`

Come esempio si consideri in seguente file.

```
Germania    3.0%
Francia     54/2000
Irlanda     67/1500
Italia      0.027
Olanda      2.2%
```

Si scriva una funzione C che prende in ingresso il nome del file e restituisce il nome della nazione che ha il rapporto più alto. Nell'esempio, la funzione deve restituire `"Irlanda"`, in quanto questa ha un rapporto di 67/1500 (pari a 4.4%, ovvero pari a 0.044) che è il massimo tra quelli presenti nel file.

Si supponga che il file è sicuramente presente su disco e che contiene almeno una riga.

Soluzione a pag. 63

Esercizio 1 del 2-2-98

Un file contiene il cognome, l'età e la media dei voti (in questo ordine) di un insieme di studenti. Il formato del file è quello del seguente esempio

```
Rossi       24          24.56
Bianchi     23          18.2
Verdi       30          29.65
Blu         22          28.30
Rosa        19          27.3
Gialli      25          27.3
Neri        24          25.56
Viola       21          30.0
```

Si scriva una funzione C che prenda come parametri il nome di un file e due interi positivi `num_scelti` ed `eta_massima` e restituisca una struttura di dati (scelta opportunamente) che contenenga i nomi e la media dei `num_scelti` studenti del file con la media più alta e che abbiano un'età inferiore a `eta_massima`.

Nel file dell'esempio, con `num_scelti` uguale a `4` ed `eta_massima` uguale a `25` la struttura di dati deve contenere le coppie: `Viola 30`, `Blu 28.3`, `Rosa 27.3` e `Neri 25.56`

Soluzione a pag. 64

Esercizio 2 del 2-2-98

Un file contiene una sequenza ordinata di interi positivi (separati da uno spazio) con possibili ripetizioni. Come esempio si considerino i seguenti due file

```
1 3 4 5 5 6 7 7 7 7 8 8

3 3 3 3 3 3 3 4 5 9
```

Il numero di volte che un valore i compare in una sequenza è detto *molteplicità* di i. Diciamo *unione* di due sequenze s_1 ed s_2 la sequenza che contiene tutti i valori i di s_1 ed s_2 con molteplicità pari al massimo tra le molteplicità di i in s_1 ed s_2.

Si scriva una funzione C che prende come parametri i nomi di tre file e un intero `n` tali che i primi due file contengono due sequenze con valori tra 1 ed n. La funzione deve scrivere nel terzo file l'unione dei primi due. Ad esempio, l'unione delle sequenze dell'esempio è la seguente.

```
1 3 3 3 3 3 3 4 5 5 6 7 7 7 7 8 8 9
```

Soluzione a pag. 65

Esercizio 1 del 19-2-98

Il sistema che gestisce le prenotazioni degli esami invia automaticamente al docente di un corso un file contenete i dati degli studenti che si sono prenotati al suo esame. Il file contiene in ciascuna riga il numero di prenotazione, la matricola, il cognome ed il nome di uno studente. Prima e dopo *ciascun* dato nel file sono presenti uno o più spazi. Un esempio di file è il seguente:

```
39 09039515 BUCCI                VITTORIO
    84     09040050 GILIBERTI            MARIA_PIA
13  09040970  ASCATIGNO            ALESSANDRO
78 09046169 CARROCCIA             ROBERTO
     56 09046261   VITALE                ANGELO
41    09047080    IMPERIOLI             MARCO
17 09047660 BUONOCORE             MAURIZIO
```

Il docente del corso ha scritto un programma che tiene traccia degli esami effettuati dagli studenti e, a tal fine, ha bisogno di copiare il file di cui sopra in un nuovo file in cui per ogni riga, separati da virgole, appaiano la matricola, il cognome ed il nome di uno studente.

Scrivere una funzione C che ricevendo in input il nome di un file lo copi in un secondo file "lista.txt" rispettando *esattamente* le specifiche di cui sopra. La funzione in questione applicata al file di esempio dovrà creare il seguente file:

```
09039515,BUCCI,VITTORIO
09040050,GILIBERTI,MARIA_PIA
09040970,ASCATIGNO,ALESSANDRO
09046169,CARROCCIA,ROBERTO
09046261,VITALE,ANGELO
09047080,IMPERIOLI,MARCO
09047660,BUONOCORE,MAURIZIO
```

Nota: Si assuma che i nomi composti sia scritti utilizzando il carattere `'_'` al posto degli spazi bianchi.

Soluzione a pag. 66

Esercizio 1 del 16-4-98

Un file di nome `corrente.txt` contiene delle misure di corrente effettuate su n transistor, ciascuno identificato dalla sigla TRX ove X è un intero compreso tra 1 ed n. Per ogni transistor sono effettuate una o più misure, ciascuna delle quali contiene, nell'ordine, l'identificatore del transistor, il numero di secondi della misura t, la tensione collettore-emettitore V_{CE} e la corrente di emettitore I_E. Il numero complessivo n dei transistor sotto misura è indicato nella prima riga. Assumendo che la potenza dissipata per ciascuna misura sia data dalla formula $V_{CE} \cdot I_E \cdot t$, scrivere una funzione C che legga il file in questione producendo

un file `totale.txt` che contiene una linea per ogni transistor indicante la potenza totale dissipata, ottenuta come somma di tutte le misure relative al transistor in questione. Un esempio di file è il seguente:

```
4
TR3    0.5  3.2  2.1
TR1    1.0  2.0  3.5
TR2    2.5  3.2  4.8
TR3    0.6  4.2  1.5
TR4    1.1  2.5  1.0
```

La funzione in questione applicata al file di esempio dovrà creare il seguente file:

```
TR1  7.0
TR2 38.4
TR3  7.14
TR4  2.75
```

Soluzione a pag. 66

Capitolo 3

Esercizi sui bit

Nonostante nell'AA 1998-1999 le operazioni di manipolazione di byte e bit non siano state trattate a lezione si ritiene opportuno, ugualmente, presentare per completezza in questa dispensa gli esercizi di esame relativi a tali problematiche.

Esercizio 3 del 17-6-96

Assumendo che il compilatore usato assegni 2 byte ad un `unsigned int` si scriva una funzione C che, ricevendo come parametro un `unsigned int`, restituisca l'`unsigned int` ottenuto scambiando tra loro i due byte del parametro in ingresso. Soluzione a pag. 68

Esercizio 3 del 15-7-96

È noto che non è stato adottato uno standard per l'ordine in cui i byte di una parola vengono memorizzati in memoria (a partire dal byte più significativo o da quello meno significativo). In particolare, se le parole vengono memorizzate a partire dal byte più significativo il byte meno significativo è memorizzato nell'indirizzo di memoria più alto (finale grande); se le parole vengono memorizzate a partire dal byte meno significativo questo sarà memorizzato nell'indirizzo di memoria più piccolo (finale piccolo).

Sapendo che il compilatore C a vostra disposizione utilizza 2 byte per rappresentare il tipo unsigned int scrivere una funzione C che restituisca:

- 1 se il microprocessore su cui gira il vostro programma utilizza la convenzione a finale piccolo o
- 2 se il microprocessore utilizza la convenzione a finale grande.

Soluzione a pag. 69

Esercizio 2 del 17-9-96

Un file contiene una sequenza di interi positivi, rappresentati in binario con un massimo di 20 cifre, separati da uno o più spazi e ritorni a capo. Ad esempio, il file

```
  10011  110    111100
1  1010101
```

rappresenta la sequenza di numeri $\langle 19, 6, 60, 1, 85 \rangle$. Si scriva una funzione che prende come parametro il nome del file contenente la sequenza e restituisce in uscita la somma (in decimale) dei valori presenti nella sequenza. La funzione inoltre deve restituire `0` se il file è vuoto e `-1` se il file non esiste.

Soluzione a pag. 69

Esercizio 3 del 17-9-96

I numeri interi da 0 a 9999 possono essere rappresentati in 2 byte utilizzando 4 bit per ogni cifra del numero. I 4 bit più significativi contengono la cifra più significativa e le altre cifre seguono in ordine. Ad esempio, i 2 byte `10010011 00010110` rappresentano il numero decimale 9316.

Le configurazioni di 4 bit che contengono numeri maggiori di 9 non sono ammissibili. Ad esempio, i 2 byte `10010011 00011101` contengono una configurazione non ammissibile: i 4 bit meno significativi contengono il valore non ammissibile `1101` (13).

Supponendo che il tipo `unsigned int` occupi 2 byte, si scriva una funzione C che prende come paramentro un `unsigned int` che contiene un numero rappresentato nel modo suddetto e restituisce un `unsigned int` che contiene il numero stesso rappresentato in modo usuale. Nel caso il parametro di ingresso contenga una configurazione non ammissibile, la funzione deve restituire il valore costante `10000`.

Soluzione a pag. 70

Esercizio 3 del 17-1-97

Una zona contigua di memoria (di dimensione ignota) contiene una sequenza di byte, ed è terminata da un byte contenente otto zeri. Diremo che un byte è *integro* se il numero di `1` presenti nel byte è pari, è *corrotto* altrimenti. Scrivere una funzione C che prende come parametro l'indirizzo iniziale dell'area di memoria e restituisce `1` se tutti i byte sono integri, `0` se almeno un byte è corrotto.

Soluzione a pag. 71

Esercizio 3 del 30-1-97

Una zona contigua di memoria (di dimensione ignota) contiene una sequenza di byte, tutti diversi tra loro, ciascuno dei quali contiene un intero tra -127 e +127 rappresentato in notazione modulo e segno (bit più significativo = 0: numero positivo; bit più significativo = 1: numero negativo). La sequenza è terminata da due byte identici. Scrivere una funzione C che riceve come parametro l'indirizzo iniziale della zona di memoria e restituisce la somma di tutti gli interi memorizzati nella stessa. **N.B.**: la seconda copia dell'ultimo byte serve solo ad indicare la fine della zona di memoria, e non va considerata nella somma.

Soluzione a pag. 72

Esercizio 3 del 3-6-97

Si scriva una funzione C che prende come parametro l'indirizzo di una locazione intera, e modifica il suo contenuto *invertendo* ciascun byte che compone la sua rappresentazione. Un byte viene invertito rovesciando l'ordine dei bit che lo compongono (cioè il byte $b_7b_6b_5b_4b_3b_2b_1b_0$ diventa il byte $b_0b_1b_2b_3b_4b_5b_6b_7$).

Ad esempio, supponendo che un intero occupi 2 byte e che il suo valore in bit sia:

```
11000000  00100001
```

il suo valore deve essere modificato in:

```
00000011  10000100
```

La funzione deve lavorare correttamente indipendentemente dal numero di byte occupato da in intero. Soluzione a pag. 73

Esercizio 3 del 17-6-97

Una zona di memoria di lunghezza ignota contiene un intero codificato BCD ed è terminata da un byte con tutti i bit ad 1. Si scriva una funzione C che prende come parametro l'indirizzo iniziale della zona di memoria e restituisce l'intero ivi contenuto.

Si ricorda che la codifica BCD consiste nel memorizzare in ogni byte due cifre decimali, ciascuna rappresentata tramite quattro bit. Ad esempio, la zona di memoria `00100001 00110100 1111111` rappresenta l'intero 2134. Soluzione a pag. 74

Esercizio 3 del 14-7-97

Si scriva una funzione C che riceve in ingresso un `unsigned int` e restituisce la stringa ottale corrispondente al **numerale** associato all'intero. Si assuma che un `unsigned int` occupi due byte e che la rappresentazione sia a finale piccolo, ovvero che il byte meno significativo abbia l'indirizzo più piccolo. Ad esempio, se il numerale corrispondente all'intero è `01001111 11000011` la funzione deve restituire la stringa `"047703"`. Soluzione a pag. 75

Esercizio 3 del 17-9-97

Si supponga che il tipo `unsigned int` occupi 4 byte di memoria. Una variabile di tipo `unsigned int` può codificare una colonna del totocalcio (cioè una sequenza di 13 simboli appartenenti all'insieme {1,2,X}) nel seguente modo: i due bit più significativi della variabile contengono il primo risultato, il terzo ed il quarto contengono il secondo risultato e così via. Ogni risultato è codificato nel seguente modo: la coppia di bit `01` corrisponde al simbolo `1`, la coppia `10` corrisponde al simbolo `2`, la coppia `11` corrisponde al simbolo `X` e la coppia `00` non codifica alcun risultato. I sei bit meno significativi della variabile non vengono utilizzati.

Si scriva una funzione C che prende in ingresso un `unsigned int` e restituisce una stringa contenente la colonna corrisponende. Nel caso che il parametro di ingresso non sia corretto, cioè contenga almeno una coppia di bit pari a `00`, la funzione deve restituire la stringa vuota.

Ad esempio, se il valore in ingresso è rappresentato dalla sequenza di bit

```
11 11 11 10 01 01 01 10 10 11 10 11 01 001111
```

allora la stringa in uscita è `"XXX211122X2X1"`. Se invece il valore di ingresso è rappresentato dalla sequenza

```
11 00 11 10 01 01 01 10 10 00 10 11 01 101011
```

allora la stringa in uscita è la stringa vuota, in quanto il secondo ed il decimo simbolo non sono codificati correttamente. Soluzione a pag. 75

Esercizio 3 del 21-10-97

Si scriva una funzione C che riceve in ingresso una stringa di otto caratteri e restituisce il carattere il cui codice ASCII corrisponde al byte formato giustapponendo gli otto bit meno significativi dei caratteri appartenenti alla stringa. L'ordine dei bit nel byte è tale che il bit più significativo è estratto dal primo carattere della stringa. Soluzione a pag. 76

Esercizio 3 del 19-1-98

Si assuma che il tipo `unsigned short int` occupi 2 byte di memoria. Una parola di tre lettere alfabetiche maiuscole è rappresentata in un `unsigned short int` memorizzando l'ultima lettere della parola nei 5 bit meno significativi, la lettera di mezzo nei successivi 5 bit e la prima lettera nei bit ancora successivi. Il bit più significativo è posto sempre a 0.

Per ogni singola lettera, la rappresentazione è tale che la lettera `A` è rappresentata da `00000`, `B` da `00001` e così via fino a `Z` che è rappresentata da `11001`.

Si scriva una funzione C che dati in ingresso una stringa `s` contenente una parola di tre lettere alfabetiche maiuscole ed un `unsigned short int num` restituisca `1` se `num` è uguale alla rappresentazione della parola nel modo suddetto, `0` se non lo è. Inoltre, la funzione deve restituire `-1` nel caso in cui `num` non sia una codifica corretta. Un numero non è una codifica corretta quando il primo bit è posto ad `1`, oppure uno dei tre quintetti non codifica una lettera (cioè abbia un valore maggiore o uguale a `26`).

Soluzione a pag. 77

Esercizio 3 del 19-2-98

Si scriva una funzione C che riceve in ingresso un intero (senza segno) e restituisce il numero massimo di bit posti a 1 adiacenti (si supponga che un intero occupi 2 byte). Ad esempio, se il parametro di ingresso è un intero il cui corrispondente numerale binario è 0001110001111100 la funzione dovrà restituire 5.

Soluzione a pag. 78

Esercizio 3 del 16-4-98

Si scriva una funzione C che riceve in ingresso un carattere senza segno ed un intero n con $0 \leq n \leq 7$ e restituisce un carattere uguale al carattere in ingresso ed in cui il bit in posizione n-esima è complementato. Ad esempio, se il numerale binario corrispondente al carattere in ingresso è `00011100` ed $n = 6$ la funzione dovrà restituire un carattere il cui numerale binario è `01011100`.

Soluzione a pag. 79

Capitolo 4

Soluzioni degli esercizi su vettori, stringhe e liste

Soluzione esercizio 2 del 1-7-96

Esistono moltissimi algoritmi per verificare la proprietà richiesta. La nostra soluzione si basa sull'idea di verificare per ciascun numero `i` tra `1` e `9` se, trovata la locazione delle prima occorrenza di `i`, le due occorrenze successive esistono e sono nella posizione desiderata.

In dettaglio, il ciclo `for` esterno è governato dalla variabile `i` che varia da `1` a `9`. Il ciclo più interno cerca una locazione `j` tale che esistano le tre occorrenze di `i` nel vettore nelle posizioni `j`, `j+i+1` e `j+2*i+2`. Se una tale locazione `j` non esiste, oppure è oltre la posizione `24-2*i` (che porterebbe il valore `j+2*i+2` oltre la locazione `26`), allora la funzione restituisce `0`. Se invece tale `j` esiste, la funzione passa ad analizzare il successivo valore di `i` (istruzione `break;`).

Si noti che utilizzare `24-2*i` come estremo per `j` invece che `26` è indispensabile per evitare che la funzione acceda a locazioni inesistenti di `v` (con indice maggiore di `26`).

```
#include <stdio.h>

int perfetta(int* v)
{
  int i,j;
  for (i = 1; i <= 9; i++)
    { /* 24-2*i e' l'ultima locazione per la prima occorrenza di i che lasci
         spazio sufficiente per le altre 2 occorrenze di i */
      for (j = 0; j <= 24-2*i; j++)
        if (v[j] == i && v[j+i+1] == i && v[j+2*i+2] == i)
          break;
      if (j > 24-2*i) /* non ho trovato un j che soddisfi la condizione per i */
        return 0;
    }
  return 1;
}

main()
{ /* programma principale di prova (non richiesto) */
  int VS[4][27] = {
      /* VS[0]: sequenza perfetta */
```

```
    {1,9,1,2,1,8,2,4,6,2,7,9,4,5,8,6,3,4,7,5,3,9,6,8,3,5,7},
     /* VS[1], VS[2], VS[3]: sequenze non perfette */
    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
    {1,9,1,1,2,8,2,4,6,2,7,9,4,5,8,6,3,4,7,5,3,9,6,8,3,5,7},
    {1,9,1,2,1,8,2,4,6,2,0,9,4,5,8,6,3,4,0,5,3,9,6,8,3,5,0}};

  int i;
  for (i = 0; i < 4; i++)
    {
      printf("La sequenza %d e' perfetta?  ",i);
      if (perfetta(VS[i]))
        printf("Si\n");
      else
        printf("No\n");
    }
}
```

Soluzione esercizio 2 del 15-7-96

La funzione scandisce la lista sino a trovare l'abbreviazione cercata o l'ultimo record. Si noti che il test del ciclo `l->next`, equivalente a `l->next != NULL`, è inconsistente nel caso di liste vuote. Inoltre, avendo utilizzato un array di due elementi per memorizzare le abbreviazioni *non* è possibile utilizzare le funzioni di libreria `strcmp()` e `strcpy()` che si basano sulla presenza del terminatore di stringa.

```
#include <stdio.h>

struct elem
{
  char abbr[2];
  char *estesa;
  struct elem *next;
};

int controlla(struct elem *l, char a[2], char *p)
{
  int app;
  while ( l->next && (l->abbr[0] != a[0] || l->abbr[1] != a[1]) )
    l = l->next;
  /* si esce dal ciclo perche' siamo sull' ultimo record (CASO A)
     o sul record che contiene l'abbreviazione          (CASO B)
     o entrambe le cose contemporaneamente              (CASO C) */

  if ( (l->abbr[0] == a[0]) && (l->abbr[1] == a[1]) )
    if ( strcmp(l->estesa, p) !=0 )
      {/* Caso B o C. La parola estesa e' differente: la sostituisco */
       l->estesa = (struct elem *) realloc(l->estesa, strlen(p)+1);
       strcpy(l->estesa, p);
       app = 2;   /* abbreviazione esistente parola estesa differente */
      }
    else app = 0; /* Caso B o C. La parola estesa e' OK */
  else
    {   /* Caso A: siamo sull'ultimo record che NON contiene l'abbreviazione
           aggiungo un record in coda alla lista */
     l->next = (struct elem *)malloc(sizeof(struct elem));
     l = l->next;
```

```
     l->estesa = (char *) malloc(strlen(p)+1);
     strcpy(l->estesa, p);
     l->abbr[0] = a[0];
     l->abbr[1] = a[1];
     l->next = 0,
     app = 1; /* abbreviazione non esistente */
    }
  return app;
}

void stampa(struct elem * l)
{  /* stampa la lista, non richiesta */
  while (l != NULL){
    printf("%c%c %s \n",l->abbr[0],l->abbr[1],l->estesa);
    l = l->next;
  }
}

struct elem *crealista(int k)    /* funzione di appoggio, non richiesta */
{                                /* crea una lista di k record          */
  int i;
  struct elem *l, *app;
  char c[21];
  if(k == 0)
    return NULL;
  else
    {
     app = l = (struct elem *) malloc(sizeof(struct elem)); /* record generatore */
     for(i = 0;i < k;++i)
       {
        l->next = (struct elem *)malloc(sizeof(struct elem));
        l = l->next;
        printf("Abbreviazione : ");
        scanf("%c%c",&(l->abbr[0]), &(l->abbr[1]));
        printf("Estesa        : ");
        scanf("%s%*c",c);        /* %*c consuma il \n */
        l->estesa = (char *) malloc(strlen(c)+1);
        strcpy(l->estesa,c);
       }
     l->next = NULL;
     l = app->next;
     free (app);
     return l;
    }
}

main () /* programma principale di test, non richiesto dal compito */
{
  struct elem *p1, **v;
  int i;
  char app[2] = {'z','z'},str[21];
  p1 = crealista(3);  /*crea una lista di 3record */
  stampa(p1);

  printf("inserire xx per uscire dal ciclo di controllo...\n");

  while ((app[0] != 'x') || (app[1] != 'x') )
```

```
   {
    printf ("Abbreviazione : ");
    scanf("%c%c",&app[0],&app[1]);
     printf("Estesa        : ");
    scanf("%s%*c",str);
   printf("%c%c %s : %d \n",app[0],app[1],str,controlla(p1,app,str));
   }
   stampa(p1);
}
```

Soluzione esercizio 1 del 17-1-97

La funzione richiesta si compone di un doppio ciclo di scansione. Il ciclo esterno scandisce il vettore di liste, e quello interno scandisce ogni singola lista.

Quando viene incontrato un elemento maggiore del massimo corrente viene memorizzato il suo valore e l'indice della lista in cui compare. Si noti che non è necessario memorizzare il massimo di ogni singola lista, ma è sufficiente cercare il massimo globalmente su tutto il vettore.

Sfruttando il fatto che i valori sono tutti interi positivi è possibile inizializzare l'ottimo corrente al valore 0. Questa inizializzazione permette anche di trattare implicitamente i casi che i vettore abbia lunghezza 0 e le liste siano tutti vuoti. In questi due casi infatti, il massimo corrente non viene mai aggiornato, e viene quindi restituito correttamente il valore 0.

```
#include <stdio.h>

struct nodo
{
  int info;
  struct nodo *next;
};

int ElementoMassimo(struct nodo **v, int n)
{
  int i, indice_max = 0, val_max = 0;
  struct nodo *p;

  for (i = 0; i < n; i++)
    {
      p = v[i];
      while (p != NULL)
        {
          if (p->info > val_max)
            {
              val_max = p->info;
              indice_max = i;
            }
          p = p->next;
        }
    }
  return indice_max;
}

/* funzioni ausiliarie di prova (non richeste dal testo) */
```

```
struct nodo **LeggiVettore(int n)
{
  struct nodo **v;
  struct nodo *p;
  int i,j;

  v = (struct nodo **) malloc(n * sizeof(struct nodo*));
  for (i = 0; i < n; i++)
    {
      printf("Elementi (rovesciati) della riga %d [0 per finire]: ",i);
      v[i] = NULL;
      scanf("%d",&j);
      while (j > 0)
        {
          p = (struct nodo *) malloc(sizeof(struct nodo));
          p->info = j;
          p->next = v[i];
          v[i] = p;
          scanf("%d",&j);
        }
    }
  return v;
}

void StampaVettore(struct nodo **v, int n)
{
  int i;
  struct nodo *p;

  for (i = 0; i < n; i++)
    {
      p = v[i];
      while (p != NULL)
        {
          printf("%d ", p->info);
          p = p->next;
        }
      printf("\n");
    }
}

main()
{ /* programma principale di prova (non richiesto) */
  int n;
  struct nodo **vett;

  printf("Quante liste? ");
  scanf("%d",&n);

  vett = LeggiVettore(n);
  StampaVettore(vett,n);

  printf("Il valore massimo e' nella lista %d\n", ElementoMassimo(vett,n));
}
```

Soluzione esercizio 1 del 30-1-97

La funzione alloca dinamicamente un vettore di n record e gestisce in un ciclo `for` l'input dei dati relativi agli n studenti. Si noti che in ogni record del vettore c'è lo spazio per un puntatore a carattere e non per una stringa: è quindi necessario allocare dinamicamente, per ogni record, la memoria per il nome dello studente.

```
#include <stdio.h>

struct elem
{
   char *nome;
   int eta;
};

struct elem * leggi(int n)
{
  struct elem *p;
  char app[21];
  int i;
  p = (struct elem *) malloc(n*sizeof(struct elem));
  if (p == NULL) return NULL;
  for (i = 0; i < n; ++i)
    {
     printf("Nome: ");
     scanf("%s", app);
     printf("Eta': ");
     scanf("%d", &(p[i].eta));
     p[i].nome = (char *)malloc(strlen(app)+1);
     if (p[i].nome == NULL) return NULL;
     strcpy(p[i].nome, app);
    }
  return p;
}

main () /* programma principale di test, non richiesto dal compito */
{
  struct elem *p;
  int i;
  p = leggi(3);
  printf("*************\n");
  for (i = 0;i < K;++i)
   printf("Nome: %-20s eta': %3d\n", p[i].nome, p[i].eta);
}
```

Soluzione esercizio 2 del 20-2-97

La funzione `UltimaPalindroma()` che risolve l'esercizio si avvale di una funzione ausiliaria `Palindroma()` che verifica se una stringa è palindroma o meno.

La funzione principale si avvale di una scansione in avanti del vettore, durante la quale ogni volta che viene incontrata una stringa palindroma, viene registrato nella variabile `indice` l'indice della locazione in cui compare. In questo modo ci si assicura che alla fine della scansione l'indice memorizzato sia quello dell'ultima palindroma.

Il test che governa il ciclo `while` si basa sulla presenza del terminatore di stringa nella prima posizione della stringa.

Analogamente avremmo potuto usare dei test del tipo

`strlen(vettore_stringhe[i]) == 0`

oppure

`!strcmp(vettore_stringhe[i],"")`

Sbagliato sarebbe stato invece il test

`vettore_stringhe[i] == NULL`

in quanto in questo caso si sarebbe verificato il fatto che il puntatore ha valore nullo e non che questo punta ad una locazione contenente la stringa nulla.

Riguardo alla funzione `Palindroma()`, questa deve gestire l'eventuale presenza di spazi bianchi nella stringa. Si noti che i due cicli `while` iniziali sono necessari perché si assume che gli spazi bianchi possano trovarsi anche ad inizio e fine di stringa. Il secondo ciclo ha la condizione aggiuntiva `j > 0` per il caso che la stringa contenga solo spazi bianchi (si considera tale stringa come palindroma).

```
#include <stdio.h>

int Palindroma(char* s)
{/* funzione ausiliaria: restituisce 1 se la stringa s
    e' palindroma, restituisce 0 altrimenti*/
  int i = 0, j = strlen(s) - 1;

  while (s[i] == ' ')
    i++;
  while (s[j] == ' ' && j > 0)
    j--;
  while (i < j)
    {
      if (s[i] != s[j]) return 0;
      while (s[++i] == ' ')
        ; /* istruzione vuota */
      while (s[--j] == ' ')
        ; /* istruzione vuota */
    }
  return 1;
}

char* UltimaPalindroma(char* vettore_stringhe[])
{/* funzione principale dell'esercizo */
  int i = 0;
  char* ultima_palindroma;
  int indice = -1;

  while (vettore_stringhe[i][0] != '\0')
    {
      if (Palindroma(vettore_stringhe[i]))
        indice = i;
      i++;
    }

  if (indice == -1)
    return NULL;
  else
    {
      ultima_palindroma = (char*) malloc(strlen(vettore_stringhe[indice]) + 1);
```

```
      strcpy(ultima_palindroma,vettore_stringhe[indice]);
      return ultima_palindroma;
    }
}

main()
{ /* programma principale di prova (non richiesto) */
  char* v[] = {"   a  d da", "etna gigate", "paolo", "marco", ""};
  char* palindroma = UltimaPalindroma(v);
  if (palindroma == NULL)
    printf("Il vettore non contiene parole palindrome\n");
  else
    printf("L'ultima parola palindroma e': ``%s''\n", palindroma);
}
```

Soluzione esercizio 1 del 15-4-97

Dovendo la funzione restituire un array di puntatori a record decidiamo che il tipo della funzione stessa è di puntatore a puntatore di record (`struct elem **`). La funzione scandisce due volte la lista: la prima per contare i record e, dopo aver allocato memoria, la seconda per assegnare agli elementi del vettore appena creato gli indirizzi dei record della lista.

```
#include <stdio.h>

struct elem
{
  char *nome;
  struct elem *next;
};

struct elem **list2vet(struct elem *p)
{
  int i, n = 0;
  struct elem *app, **v;
  app = p;
  while (app != NULL)
    {
     ++n;        /* conta i record della lista */
     app = app->next;
    }
  if (n != 0)
    {
     v=(struct elem **) malloc(n*sizeof(struct elem*));
     for (i = 0; i < n; ++i)
       {
        v[i] = p;
        p = p->next;
       }
     return v;
    }
  else return NULL;   /* lista vuota */
}

struct elem *crealista(int k)    /* funzione di appoggio, non richiesta */
{                                /* crea una lista di k record          */
  int i;
```

```
  struct elem *l, *app;
  char c[21];
  if (k == 0)
    return NULL;
  else
    {
     app=l=(struct elem *) malloc(sizeof(struct elem)); /* record generatore */
     for(i = 0;i < k;++i)
       {
        l->next = (struct elem *)malloc(sizeof(struct elem));
        l = l->next;
        printf("Inserire un elemento : ");
        scanf("%s",c);
        l->nome = (char *) malloc(strlen(c)+1);
        strcpy(l->nome,c);
       }
     l->next = NULL;
     l = app->next;
     free(app);
     return l;
    }
}

main () /* programma principale di test, non richiesto dal compito */
{
  struct elem *p1, **v;
  int i;
  p1 = crealista(3);  /*crea una lista di 3 stringhe */
  v = list2vet(p1);
  for(i = 0; i < 3; ++i)
    printf("Nell' elemento %d di v c'e': %s\n",i,v[i]->nome);
}
```

Soluzione esercizio 3 del 15-4-97

La funzione alloca, in una variabile di appoggio, memoria sufficiente a contenere la stessa stringa passata per parametro. Un ciclo di `strlen(str) + 1` iterazioni scandisce la stringa di ingresso copiandola, carattere per carattere su quella di appoggio, saltando i caratteri uguali al parametro di ingresso `c`. In particolare la variabile `i` è utilizzata per scandire la stringa sorgente mentre la variabile `j` é utilizzata per scandire la stringa destinazione. L'ultimo ciclo del for copierà il terminatore di stringa.

Essendo `str` una copia del parametro di ingresso è possibile utilizzarlo per copiarci sopra la nuova stringa. Guadagnando in leggibilità ed utilizzando una variabile in più la stessa cosa poteva essere fatta utilizzando una ulteriore variabile di appoggio.

```
#include <stdio.h>

char * cancella(char *str, char c)
{
  int i, j = 0;
  char *app;
  app = (char *) malloc(strlen(str)+1);
  for (i = 0; i <= strlen(str); ++i) /* copia anche il terminatore di stringa */
    if (str[i] != c)
      app[j++] = str[i];
  str = (char *) malloc(strlen(app)+1);  /* str non serve piu' */
```

```
  strcpy(str,app);
  free(app);
  return str;
}

main () /* programma principale di test, non richiesto dal compito */
{
   char *s = "sschissa se ss s sfunzionasssssssssss";
   printf("%s\n",s);
   printf("%s\n",cancella(s,'s'));
}
```

Soluzione esercizio 2 del 17-6-97

```
#include <stdio.h>

void modifica(char **v, int n)
{
  int i,j;
  char *app;

  for (i = 0; i < n-1; ++i)
    for(j = i+1; j < n; ++j)
      if ( (v[i][ strlen(v[i])-2 ] == v[j][0]) &&
           (v[i][ strlen(v[i])-1 ] == v[j][1]) )
        {
          v[i] = (char *) realloc(v[i],strlen(v[i])+strlen(v[j])-2+1);
          strcat(v[i],v[j]+2);
          break;
        }
}

main()
{/* programma principale di prova (non richiesto) */
  char *vett[7], buffer[21];
  int i;
  FILE *p;
  p=fopen("dati2.txt","r");
  for (i=0;i<7;++i) {
    fscanf(p,"%s",buffer);
    vett[i]=(char *)malloc(strlen(buffer)+1);
    strcpy(vett[i],buffer);
  }
  fclose(p);
  for (i=0;i<7;++i) printf("vett[%d]=%s\n",i,vett[i]);
  modifica(vett,7);
  printf("\n");
  for (i=0;i<7;++i) printf("vett[%d]=%s\n",i,vett[i]);
}
```

Soluzione esercizio 2 del 1-7-97

La funzione inizialmente conta il numero di spazi presenti tra il nome e il cognome, in modo da poter allocare correttamente la stringa risultante (`ris`). In particolare, la dimensione della stringa risultante è pari alla dimensione della stringa `nome_cognome` privata delgi spazi,

più la dimensione del soprannome più cinque (2 per le parentesi, 2 per gli spazi e uno per il terminatore di stringa).

Nei tre cicli successivi vengono scritti nella stringa il nome, il soprannome e il cognome rispettivamente. Le parentesi, gli spazi e il terminatore di stringa vengono inseriti nella stringa da apposite istruzioni di assegnazioni.

```
#include <stdio.h>

char* AggiungiSoprannome(char* nome_cognome, char* soprannome)
{
  char *ris; /* stringa risultato */
  int i,  /* indice per scorrere il soprannome */
      j,  /* indice per scorrere il risultato */
      k;  /* indice per scorrere nome e cognome */
  int num_spazi = 0; /* numero di spazi tra nome e cognome */

  for (k = 0; nome_cognome[k] != '\0'; k++)
    if (nome_cognome[k] == ' ')
        num_spazi++;

  ris = (char*) malloc(strlen(nome_cognome) - num_spazi +
                       strlen(soprannome) + 5);
     /* 5 caratteri extra: 2 parentesi, 2 spazi, 1 terminatore */

  for (k = 0; nome_cognome[k] != ' '; k++)
    ris[k] = nome_cognome[k];
  j = k;
  ris[j++] = ' ';
  ris[j++] = '(';

  for (i = 0; soprannome[i] != '\0'; i++, j++)
    ris[j] = soprannome[i];
  ris[j++] = ')';
  ris[j++] = ' ';

  for ( ; nome_cognome[k] == ' '; k++)
    ;
  for ( ; nome_cognome[k] != '\0'; k++, j++)
    ris[j] = nome_cognome[k];
  ris[j] = '\0';
  return ris;
}

main()
{ /* programma principale (non richesto dal testo) */
  char stringa_nome[31], stringa_soprannome[21], ch;
  int i = 0;
  printf("Inserisci la stringa con nome e cognome\n");
  while ( (ch = getchar()) != '\n')
    stringa_nome[i++] = ch;
  stringa_nome[i] = '\0';

  i = 0;
  printf("Inserisci la stringa con il soprannome\n");
  while ( (ch = getchar()) != '\n')
    stringa_soprannome[i++] = ch;
  stringa_soprannome[i] = '\0';
```

```
  printf("%s\n",AggiungiSoprannome(stringa_nome, stringa_soprannome));
}
```

Soluzione esercizio 2 del 14-7-97

Proponiamo per questo esercizio due soluzioni diverse una iterativa (funzione `modifica()`) ed una ricorsiva (funzione `modifica2()`).

```
#include <stdio.h>

struct elem
{ int valore;
  struct elem * next;
};


void modifica(struct elem *l)
{
  int prec, copia;
  prec = l->valore;
  do
    {
      l = l->next;
      copia = l->valore;
      l->valore = l->valore + prec;
      prec = copia;
    }
  while (l->next);
}

int modifica2(struct elem *inizio)
{
  if (inizio->next->next == NULL)
    inizio->next->valore += inizio->valore;
  else
    {
      modifica2(inizio->next);
      inizio->next->valore += inizio->valore;
    }
}

struct elem* crea(int n)
{ /* funzione ausiliaria (non richiesta): crea una lista
     con i valori n, n-1, ..., 2, 1 */
  int i;
  struct elem *l, *app;
  app = (struct elem *) malloc(sizeof(struct elem));
  app->next = NULL;
  for (i = 1; i <= n; ++i)
    {
      app->valore = i;
      l = app;
      app = (struct elem *) malloc(sizeof(struct elem));
      app->next = l;
  }
```

```
  free(app);
  return l;
}

void stampa(struct elem *l)
{/* funzione ausiliaria (non richiesta): stampa una lista */
  while (l != NULL)
    {
      printf("%d ",l->valore);
      l=l->next;
    }
  printf("\n");
}

main()
{/* programma principale di prova (non richiesto) */
  int i;
  struct elem *p;
  p = crea(10);
  stampa(p);
  modifica(p);
  stampa(p);
  modifica2(p);
  stampa(p);
}
```

Soluzione esercizio 2 del 17-9-97

Le assunzioni fatte nel testo ci garantiscono con certezza che la stringa risultante ha la stessa lunghezza delle due stringhe in ingresso. È quindi possibile allocare subito tale stringa della sua lughezza definitiva.

Successivamente, si esegue un ciclo in cui viene calcolata ogni cifra del risultato. Tale ciclo parte dall'ultima locazione delle stringhe (cioè le cifre meno significative dei numeri) per poter propagare correttamente il riporto verso le cifre più significative.

La conversione tra i caratteri `'0'`, `'1'`, ..., `'9'` e i valori interi 0, 1, ..., 9, viene gestito tramite sottrazione e somma del valore del carattere `'0'`.

```
#include <stdio.h>

char* Somma(char* s1, char* s2)
{
  char* ris;
  int n = strlen(s1);
  int i, carry = 0, somma;

  ris = (char*) malloc(n + 1);
  ris[n] = '\0';
  for (i = n-1; i >= 0; i--)
    {
      somma = (s1[i] - '0') + (s2[i] - '0') + carry;
      carry = somma / 10;
      ris[i] = somma % 10 + '0';
    }
  return ris;
}
```

```
main()
{ /* programma principale di prova (non richiesto) */
  char a[101], b[101];
  printf("Inserisci il primo numero : ");
  scanf("%s",a);
  printf("Inserisci il secondo numero (%d cifre): ", strlen(a));
  scanf("%s",b);
  printf("La somma di %s e %s e' : %s\n",a,b,Somma(a,b));
}
```

Soluzione esercizio 2 del 21-10-97

La funzione `Media()` somma (e conta) i valori presenti nel vettore che non siano già presenti nella parte *precedentemente* analizzata del vettore stesso. Quest'ultimo controllo è affidato alla funzione `gia_presente()`.

```
#include <stdio.h>

int gia_presente(int v[], int e)
{/* funzione ausiliaria: controlla se l'elemento di indice e
    e' presente nella parte del vettore v tra 0 ed e-1 */
  int j;
  for (j = 0; j < e; j++)
    if (v[j] == v[e])
      return 1;
  return 0;
}

float Media(int vet[], int n)
{
  int somma = 0, conta = 0, i;
  for (i = 0; i < n; i++)
    if (!gia_presente(vet,i))
      {
        somma += vet[i];
        conta++;
      }
  return ((float) somma) / conta;
}

main()
{ /* programma principale (non richiesto) */
  int i,n;
  int *p;
  printf("Numero elementi : ");
  scanf("%d%",&n);
  p = (int*) malloc(n * sizeof(int));
  printf("Inserisci gli %d elementi\n",n);
  for (i = 0; i < n; i ++)
    scanf("%d",&p[i]);
  printf("La media senza duplicati e' %f\n",Media(p,n));
}
```

Soluzione esercizio 2 del 19-1-98

La funzione si compone di due cicli annidati. Il ciclo più esterno scandisce tutta la stringa, ed è infatti controllato dalla condizione `s[i] != '\0'`. Il ciclo più interno calcola il valore di un singolo numero presente nella sequenza, ed è controllato dalla condizione che il carattere incontrato sia non alfabetico e non numerico.

Il calcolo del valore del numero viene fatto a partire dalla cifra più significativa (cioè quella di indice più basso). Per ogni cifra che si incontra, se ne verifica l'appartenenza alla base b e si aggiunge alla variabile `n` che contiene il numero corrente. Per ogni nuova cifra aggiunta, il valore di `n` viene moltiplicato per la base, in quanto la presenza della nuova cifra fa aumentare di uno la significatività delle cifre precedenti.

```
#include <stdio.h>

/* funzione ausiliaria fornita nel testo */
int valore(char ch)  /* restituisce il valore corrispondente al carattere ch */
{ if (isdigit(ch)) return ch - '0';
  else if (isupper(ch)) return ch - 'A' + 10;
  else return -1;
}

long Massimo(char* s, int base)
{ /* funzione richiesta */
  int i = 0;
  long n, max = 0;
  while (s[i] != '\0')
    {
      long n = 0;
      while (isupper(s[i]) || isdigit(s[i]))
        {
          int val = valore(s[i]);
          if (val >= base)
            return -1;
          else
            n = n * base + val;
          i++;
        }
      if (n > max)
        max = n;
      if (s[i] != '\0')
        i++; /* avanza oltre il carattere '-' */
    }
  return max;
}

main()
{ /* programma principale di test (non richiesto dal compito) */
  char buffer[81];
  int base;
  printf("Inserisci la stringa\n");
  scanf("%s",buffer);
  printf("Inserisci la base\n");
  scanf("%d",&base);
  printf("Il massimo e' %d\n",Massimo(buffer,base));
}
```

Soluzione esercizio 2 del 19-2-98

La funzione `conta()` utilizza un vettore `app` in cui memorizza per ogni lettera il numero di volta che è seguita della sua successiva. Successivamente, la funzione cerca il massimo del vettore `app`.

Il vettore è allocato di dimensione pare al numero di lettere dell'alfabeto (cioè 26), indicato tramite l'espressione `'Z' -'A' + 1`.

```
#include <stdio.h>

char conta(char *v, int n ){
   int i, app['Z'-'A'+1]; /* assumo i caratteri tutti maiuscoli */
   int max = 0;

   for (i = 0; i < ('Z'-'A'+1); ++i)
      app[i] = 0;

   for (i = 0 ; i < n-1; ++i)
      if (v[i+1] == v[i] + 1)
          ++app[v[i]-'A'];
   for (i = 1; i < ('Z'-'A'+1); ++i)
       if (app[i] > app[max])
         max = i;
   return max+'A';
}

int main()
{/* programma principale di prova (non richiesto) */
  printf("%c\n",conta("ABABC",5));
  printf("%c\n",conta("BCFLZ",5));
  printf("%c\n",conta("ABCDE",5));
  printf("%c\n",conta("FFGFF",5));
}
```

Soluzione esercizio 2 del 16-4-98

Per poter modificare la dimensione del vettore è necessario che il puntatore iniziale del vettore sia modificato, e quindi questo deve essere passato per indirizzo. Infatti, la funzione `realloc()`, che utilizzeremo, in generale restituisce un valore diverso da quello passato come primo parametro.

Il primo parametro `punt_vettore` della funzione `ModificaVettore()` che risolve l'esercizio sarà quindi il puntatore al puntatore iniziale del vettore, e quindi sarà di tipo `int**`. Per indirizzare l'elemento `i` del vettore utilizzeremo quindi l'espressione `(*punt_vettore)[i]`.

La funzione inizialmente rialloca il vettore, poi sposta i suoi elementi in avanti di una locazione a partire dall'ultima locazione fino a giungere alla posizione dove inserire l'elemento passato come terzo parametro.

Come soluzione alternativa, si sarebbe potuto passare il puntatore iniziale del vettore per valore (parametro di tipo `int*`) restituendo il nuovo puntatore iniziale come valore restituito della funzione.

```
#include <stdio.h>

void ModificaVettore(int **punt_vettore, int n, int elem)
{
```

```
  int i;
  *punt_vettore = (int *) realloc (*punt_vettore, (n+1) * sizeof(int));
  for (i = n-1; (*punt_vettore)[i] > elem; i--)
    (*punt_vettore)[i+1] = (*punt_vettore)[i];
  (*punt_vettore)[i+1] = elem;
}

main()
{ /* programma principale di prova (non richiesto) */
  int n, i, e, *v;

  printf("Dimensione del vettore : ");
  scanf("%d",&n);

  v = (int*) malloc(n * sizeof(int));
  printf("Inserisci gli %d elementi : ", n);
  for (i = 0; i < n; i++)
    scanf("%d",&v[i]);

  printf("Elemento da inserire nel vettore : ");
  scanf("%d",&e);

  ModificaVettore(&v,n,e);

  printf("Vettore modificato ");
  for (i = 0; i < n+1; i++)
    printf("%d ",v[i]);
  printf("\n");
}
```

Capitolo 5

Soluzioni degli esercizi sui file

Soluzione esercizio 1 del 3-6-96

La funzione che risolve l'esercizio si compone di due fase distinte di lettura e scrittura nel file.

Nella prima fase, viene letto il file una stringa alla volta. Sfruttando il fatto che gli spazi ed i ritorni a capo vengono ignorati dal formato `%s` della funzione `fscanf()`, leggendo il file con tale formato, non è necessario ispezionare il contenuto delle stringhe lette, ma soltanto la loro lunghezza.

Per poter utilizzare unicamente le modalità semplici di apertura dei file (`"r"`, `"w"` e `"a"`), dopo la lettura il file viene chiuso, per essere successivamente riaperto in modalità di inserimento.

```
#include <stdio.h>

void AggiungiMedia(char *nomefile)
{
  FILE *fp;
  int somma = 0, conta = 0, media, i;
  char a[81];

  if ((fp = fopen(nomefile,"r")) == NULL)
    {
      printf("Il file %s non esiste\n",nomefile);
      exit(1);
    }
  while (fscanf(fp,"%s",a) == 1)
    {
      somma += strlen(a);
      conta++;
    }
  media = somma/conta;   /* la divisione tra interi avviene
                            con troncamento, restituendo la media
                            arrotondata per difetto come richiesto */
  fclose(fp);

  fp = fopen(nomefile,"a");
  putc('\n',fp); /* inserisce la media a riga nuova per essere
                    sicuro che entri nelle 80 colonne */
```

```
  for (i = 0; i < media; i++)
    putc('*',fp);
  fclose(fp);
}

main()
{ /* programma principale di prova (non richiesto) */
  char file[20];
  printf("Nome file: ");
  scanf("%s",file);
  AggiungiMedia(file);
}
```

Soluzione esercizio 2 del 3-6-96

La soluzione dell'esercizio viene scomposta in 4 funzioni.

- `Simmetrica()` verifica se la matrice passata per parametro è simmetrica.
- `LeggiMatrice()` alloca in memoria lo spazio necessario per la matrice, e legge quest'ultima dal file.
- `DeallocaMatrice()` dealloca la memoria dinamica utilizzata dalla matrice.
- `MatriceSimmetrica()` è la funzione principale che risolve l'esercizio invocando le altre funzioni.

L'allocazione della matrice è completamente dinamica e utilizza un vettore di puntatori ad interi.

Si noti che la lettura del carattere `'#'` avviene utilizzando il formato `%*s`. Il formato `%*c` non sarebbe stato adeguato, in quanto questo formato non ignora gli spazi, e quindi avrebbe letto semplicemente il primo spazio dopo l'ultimo valore della riga. Non avrebbe quindi portato il cursore al di là del carattere `'#'`, facendo fallire la successiva lettura di un valore intero con il formato `%d`.

```
#include <stdio.h>

typedef int** matrice;

int Simmetrica(matrice m, int n)
{
  int i,j;
  for (i = 0; i < n; i++)
    for (j = i+1; j < n; j++)
        if (m[j][i] != m[i][j])
          return 0;
  return 1;
}

matrice LeggiMatrice(char* nomefile, int n)
{
  int i,j;
  FILE *fp;
  matrice m = (matrice) malloc(n*sizeof(int*));
  for (i = 0; i < n; i++)
    m[i] = (int*) malloc(n*sizeof(int));
```

```
  fp = fopen(nomefile,"r");
  for (i = 0; i < n; i++)
    {
      for (j = 0; j < n; j++)
        fscanf(fp,"%d",&(m[i][j]));
      fscanf(fp,"%*s"); /* legge il simbolo # nel file */
    }
  fclose(fp);
  return m;
}

void DeallocaMatrice(matrice m, int n)
{
  int i;
  for (i = 0; i < n; i++)
    free(m[i]);
  free(m);
}

int MatriceSimmetrica(int n)
{
  int s;
  matrice m = LeggiMatrice("matrice.dat",n);
  s = Simmetrica(m,n);
  DeallocaMatrice(m,n);
  return s;
}

main()
{ /* programma principale di prova (non richiesto) */
  int n;
  printf("Dimensione della matrice: ");
  scanf("%d",&n);
  if (MatriceSimmetrica(n))
    printf("La matrice e' simmetrica\n");
  else
    printf("La matrice non e' simmetrica\n");
}
```

Soluzione esercizio 2 del 17-6-96

La soluzione proposta si basa sull'idea di leggere il file due volte, la prima per calcolare lo spazio occupato da tutte le parole e la seconda per concatenarle in un'area di memoria appositamente allocata.

Per motivi didattici, i cicli di lettura del file sono stati implementati in due modi differenti. Il primo utilizza la funzione `feof()` che, per assumere valore vero, necessita che una lettura dal file fallisca, incontrando la fine del file. Il secondo ciclo si basa sul valore restituito dalla `fscanf()` che assume il valore costante `EOF` qualora la lettura fallisca incontrando la fine del file. Il comportamento della funzione `feof()` è legato al meccanismo con cui il sistema operativo gestisce le fine del file e varia da compilatore a compilatore. Nel seguito, quindi, le soluzioni proposte si baseranno esclusivamente sul meccanismo mostrato nel secondo ciclo, inerentemente più elegante. Tra le due scansioni del file è stata utilizzata la funzione `rewind()`, che riposiziona il puntatore in all'inizio del file.

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

```
void stringona(char *f, char **s)
{
  /* restituisce un puntatore alla stringa o NULL in caso di errore */
  FILE *in;
  char a[81];
  unsigned long int tot = 0;
  int i;

  in = fopen(f,"r");     /* apro il file in lettura e memorizzo in tot
                            lo spazio occupato da tutte le parole    */
  fscanf(in,"%s",a);
  while (!feof(in))
    {
     tot += strlen(a);
     fscanf(in, "%s", a);
    }
  rewind(in);
  *s = (char *)malloc(tot+1);          /* alloco memoria per tot+1 caratteri */
  if (*s != NULL)                      /* se l'allocazione ha avuto successo */
     {
      *s[0] = '\0';                     /* *s punta ad una stringa vuota,
                                           altrimenti strcat() non funziona  */
       while (fscanf(in,"%s",a) != EOF) /* leggo le stringhe e le concateno   */
         strcat(*s, a);                 /* con la funzione di libreria strcat */
      }
  fclose(in);
}

main ()
{ /* programma principale di prova (non richiesto) */
  char filedos[13];
  char *stringa;
  printf("Inserire il nome di un file (provate con compito2.c ...)");
  scanf("%s",filedos);
  stringona(filedos,&stringa);
  printf("\n\n %s",stringa);

}
```

Soluzione esercizio 1 del 1-7-96

La soluzione dell'esercizio si basa sull'idea di copiare il file di ingresso nel file di uscita carattere per carattere fino al punto in cui si incontra il carattere speciale `'#'`. Incontrato questo carattere è necessario effettuare una lettura formattata, tramite la funzione `fscanf()`, per ottenere il numero intero che segue il carattere `'#'`. Sulla base di questo numero avviene la scrittura sul file di uscita.

La soluzione alternativa di leggere il numero che segue il carattere `'#'` ancora tramite la funzione `getc()` ha due inconvenienti:

- Il numero potrebbe essere composto da più di una cifra, quindi la lettura di un singolo carattere non è sufficiente;
- Il numero corrispondente ad un carattere letto è il suo codice ASCII e non il numero stesso. Ad esempio, il carattere `'1'` corrisponde all'intero `49` e non a `1`.

Si noti inoltre che una lettura di tutto il file stringa per stringa utilizzando il formato `%s` non avrebbe copiato nel file di output i caratteri di spaziatura (gli spazi bianchi e i ritorni a capo).

```
#include <stdio.h>

void sostituisci(int n, char* inputfile, char* outputfile, char** punt_dati)
{
  FILE *ifp,*ofp;
  char ch;
  int i;

  ifp = fopen(inputfile,"r");
  ofp = fopen(outputfile,"w");

  while ((ch = getc(ifp)) != EOF)
    if (ch == '#')
      {
        fscanf(ifp,"%d",&i);
        if (i < n)
          fprintf(ofp,"%s",punt_dati[i]);
        else
          fprintf(ofp,"...");
      }
    else
      putc(ch,ofp); /* i caratteri diversi da # vengono
                       copiati normalmente */

  fclose(ifp);
  fclose(ofp);
}

main()
{ /* programma principale (non richiesto) */
  char *inputfile = "modello.txt";
  char *outputfile = "domanda.txt";
  char *punt_dati[5] = {"Paolo Rossi","Torino", "13/7/71",
                        "essere ammesso","15/6/96"};

  sostituisci(5,inputfile,outputfile,punt_dati);
}
```

Soluzione esercizio 1 del 15-7-96

La scansione del file avviene in un ciclo che legge dal file un carattere alla volta, fino a quando il file non finisce. Se il carattere letto è una parentesi aperta una ulteriore lettura preleva il nome dello studente, il voto corrispondente e consuma la parentesi chiusa senza assegnarla ad una variabile usando il formato di conversione (`%*c`). Se necessario, vengono aggiornati i valori relativi allo studente più bravo.

```
#include <stdio.h>

void media(char *nomefile)
{
  float media;
  int quanti, voto, votomax;
```

```
  char bravo[21], studente[21], app;
  FILE *p;

  quanti = votomax = 0;
  p = fopen(nomefile, "r");
  while ( fscanf(p, "%c", &app) != EOF )
    {
     if (app == '(')
       {
        fscanf(p,"%s%d%*c", studente, &voto);
        media += voto;
        ++quanti;
        if (voto > votomax)
          {
           strcpy(bravo, studente);
           votomax = voto;
          }
       }
  }
  fclose(p);
  printf("Media         : %f\n",media/quanti);
  printf("Studente bravo: %s voto: %d \n",bravo, votomax);
}

main () /* programma principale di test, non richiesto dal compito */
{
   media("voti.txt");
}
```

Soluzione esercizio 1 del 17-9-96

La soluzione dell'esercizio si basa sulla scansione parallela dei due file di ingresso. Confidando sulla precondizione che i file siano identici a meno dei valori dei tassi, utilizziamo un ciclo di scansione `while` condizionato dalla presenza dei dati solo sul primo file. Anche il nome della valuta viene memorizzato solo per il primo file, mentre per il secondo viene ignorato (tramite il formato `%*s`). Il tasso di cambio viene invece estratto da entrambi i file, e viene memorizzato nelle due variabili `tasso1` e `tasso2`, mentre il nome della nazione viene ignorato in entrambi i file.

La chiamata alla funzione `strcmp()` verifica se la valuta letta sia uguale quella cercata. In tal caso, l'incremento viene calcolato e confrontato con i valori 0.01 (cioè 1%) e -0.01 (cioè -1%). Il risultato da restituire viene assegnato alla variabile `risultato` utilizzando delle stringhe costanti, e infine l'istruzione `break;` interrompe il ciclo.

Si ricordi che le stringhe costanti vengono allocate (staticamente) in un area dati esterna al record di attivazione della funzione in cui compaiono. Quindi lo spazio di memoria che le contiene non viene deallocato alla fine dell'esecuzione della funzione `Variazione()`.

```
#include <stdio.h>

char* Variazione(char* nomefile1, char* nomefile2, char* valuta_cercata)
{
  char valuta[21], *risultato = NULL;
        /* si assume che i nome delle valute sia di
           al massimo 20 caratteri */
  float tasso1, tasso2, incremento;
  FILE *fp1, *fp2;
```

```
  fp1 = fopen(nomefile1,"r");
  fp2 = fopen(nomefile2,"r");
  while(fscanf(fp1,"%*s %s %f",valuta,&tasso1) != EOF)
    {
      fscanf(fp2,"%*s %*s %f",&tasso2);
      if (strcmp(valuta,valuta_cercata) == 0)
        {
          incremento = (tasso2 - tasso1)/tasso1;
          if (incremento >= 0.01)
            risultato = "In Rialzo";
          else if (incremento <= -0.01)
            risultato = "In Ribasso";
          else
            risultato = "Stabile";
          break;
        }
    }
  fclose(fp1);
  fclose(fp2);
  return risultato; /* se la valuta non esiste restituisce NULL */
}


main()
{ /* programma principale di prova (non richiesto dal testo) */
  char valuta[21];
  char* variazione;
  printf("Valuta : ");
  scanf("%s",valuta);
  variazione = Variazione("cambi1.txt","cambi2.txt",valuta);
  if (variazione != NULL)
    printf("La valuta %s e' %s\n",valuta,variazione);
  else
    printf("Valuta inesistente\n");
}
```

Soluzione esercizio 1 del 17-10-96

Il programma legge un carattere alla volta dal primo file copiandolo nel secondo file fino a che il primo non finisce. Durante la copia si tiene traccia degli ultimi due caratteri copiati (`a` ultimo, `b` penultimo) e, nel caso siano uguali e consonanti, si attiva un ulteriore ciclo in cui si leggono ma non si copiano tutte le eventuali consonanti uguali alle ultime due copiate. Il ciclo in questione termina quando incontra un carattere differente o la fine del file. Si noti l'equivalenza tra la `fscanf()` con stringa di conversione `%c` e la `fgetc()`. Analoga considerazione vale per la `fprinf ()` e la `fputc()`. A scopo didattico sono state utilizzate entrambe le alternative, mettendo in evidenza il test di fine file. A tale proposito si noti che le parentesi nell' `if ( (b = fgetc(f1)) != EOF )` sono necessarie, in quanto l'operatore `!=` ha la precedenza sull'operatore `=`. Inoltre, essendo la costante `EOF` un intero negativo è necessario dichiarare `b` come `signed char`.

```
#include <stdio.h>

int consonante(char c)     /* facoltativa */
{
  c = tolower(c);   /* converte il carattere in minuscolo */
```

```
  if ( (c > 'a') && (c <= 'z') && (c != 'e') &&
       (c != 'i') && (c != 'o') && (c != 'u') )
     return 1;
  else return 0;
}

void correggi(char *file1, char *file2)
{
  FILE *f1, *f2;
  char a;
  signed char b;          /* serve per il confronto con EOF */
  f1 = fopen(file1, "r");
  f2 = fopen(file2, "w");
  if ( (b = fgetc(f1)) != EOF )       /*legge primo carattere */
    fputc(b, f2);                   /*se non EOF lo copia nel file2  */
  while ( fscanf(f1, "%c", &a) != EOF )    /* a: ultimo carattere  copiato */
    {
     fprintf(f2, "%c", a);                 /* b: penultimo caratt. copiato */
     while ( a == b && consonante(a) )     /* due consonanti uguali copiate */
       {
        if ( fscanf(f1,"%c",&a) == EOF )   /* si legge senza copiare */
           break;
        if (a != b)
           fprintf(f2, "%c", a);
       }
     b = a;                                /* si aggiorna b */
    }
  fclose(f1);
  fclose(f2);
}

main () /* programma principale di test, non richiesto dal compito */
{
  correggi("errato.txt", "ok.txt");
}
```

Soluzione esercizio 2 del 17-1-97

La funzione `TrovaMinimo()` che risolve l'esercizio si compone sostanzialmente di tre passi, che vengono eseguiti da tre funzioni ausiliarie.

- La funzione `LeggiMatrice()` alloca lo spazio necessario per la matrice e legge la matrice dal file di ingresso.
- La funzione `MinimoColonnaMassima()` calcola l'elemento richiesto dall'esercizio.
- La funzione `DeallocaMatrice()` dealloca lo spazio utilizzato per la matrice.

La funzione `MinimoColonnaMassima()` si compone a sua volta di due passi. Nel primo passo viene individuata la colonna di somma massima, e nel secondo passo se ne calcola l'elemento minimo. La soluzione in un unico passo, tramite l'identificazione del minimo all'interno del calcolo della somma, sarebbe stata comunque possibile.

Si noti inoltre che la soluzione proposta, sebbene concettualmente semplice, non è molto efficiente dal punto di vista dello spazio di memoria occupanto. Si può infatti facilmente

rendersi conto che non è strettamente necessario memorizzare interamente la matrice per risolvere l'esercizio. Sarebbe stato anche possibile ottenere il risultato memorizzando soltanto due vettori contenenti la somma ed il minimo (correnti) di ogni colonna.

```
#include <stdio.h>

typedef double** matrice;

matrice LeggiMatrice(FILE* fp, int n, int m)
{
  int i,j;

  matrice mat = (matrice) malloc(n * sizeof(double*));
  for (i = 0; i < n; i++)
    mat[i] = (double*) malloc(m * sizeof(double));

  for (i = 0; i < n; i++)
      for (j = 0; j < m; j++)
        fscanf(fp,"%lf",&mat[i][j]);

  return mat;
}

void DeallocaMatrice(matrice m, int n)
{
  int i;
  for (i = 0; i < n; i++)
    free(m[i]);
  free(m);
}

double MinimoColonnaMassima(matrice mat, int n, int m)
{
  int i, j;

  /* variabili per la ricerca della colonna di somma massima */
  double somma_max, somma_colonna;
  int colonna_max;

  /* variabili per la ricerca dell'elemento minimo */
  int min;

  /* ricerca della colonna di somma massima */
  for (j = 0; j < m; j++)
    {
      somma_colonna = 0;
      for (i = 0; i < n; i++)
        somma_colonna += mat[i][j];
      if (j == 0 || somma_colonna > somma_max)
        {
          somma_max = somma_colonna;
          colonna_max = j;
        }
    }
  /* ricerca dell'elemento minimo */
  min = 0;
  for (i = 1; i < n; i++)
```

```
    if (mat[i][colonna_max] < mat[min][colonna_max])
      min = i;

  return mat[min][colonna_max];
}

double TrovaMinimo(char* nomefile)
{
  int n, m;
  FILE *fp;
  double v;
  matrice mat;

  fp = fopen(nomefile,"r");
  fscanf(fp,"%d%d",&n,&m);
  mat = LeggiMatrice(fp,n,m);
  fclose(fp);
  v = MinimoColonnaMassima(mat,n,m);
  DeallocaMatrice(mat,n);
  return v;
}

main()
{ /* programma principale (non richiesto per il compito) */
  printf("Il minimo della colonna di somma massima e' %f\n",
         TrovaMinimo("matrice.dat"
```

Soluzione esercizio 2 del 30-1-97

Si è scelto di restituire i valori richiesti tramite un vettore di due interi (di cui la funzione restituisce l'indirizzo). Si noti che la memoria per tale vettore deve essere allocata dinamicamente: la memoria utilizzata da una eventuale variabile locale, del tipo `int ris[2]`, non sarebbe più disponibile alla fine della funzione. Alternativamente, è possibile restituire i valori richiesti utilizzando una `struct` invece del vettore oppure utilizzando un passaggio di parametri per variabile, ovvero passando alla funzione dei puntatori ai dati da modificare (e.g., soluzione dell'esercizio 2 del 15 aprile 97).

```
#include <stdio.h>

int * leggi(char *f, char *c)
{
  FILE *p;
  int i,n, t, s, *ris;
  char citta[21];
  p = fopen(f, "r");
  fscanf(p, "%d",&n);
  ris = (int *)malloc(2*sizeof(int));
  ris[0] = 0;
  ris[1] = 0;
  for ( i = 0;i < n;++i)
    {
     fscanf(p,"%s %d %d", citta, &t, &s);
     if ( strcmp(citta,c) == 0 )
       {
        if (t > ris[0]) ris[0]=t;
        ris[1] += s;
```

```
      }
    }
  fclose(p);
  return ris;
}

main () /* programma principale di test, non richiesto dal compito */
{
   int *totale;
   totale=leggi("tele.txt","ROMA");
   printf("Piu' lunga: %d totale scatti: %d\n", totale[0], totale[1]);
}
```

Soluzione esercizio 1 del 20-2-97

La funzione si compone essenzialmente di un ciclo di scansione del file di ingresso. Ad ogni iterazione viene innanzitutto letto il numero di studenti, e questa lettura governa anche la condizione di permanenza nel ciclo. Sucessivamente si ha una lettura carattere per carattere fino ad incontrare il carattere `'('`. In base al formato del file, dopo il carattere `'('` viene letta la data, che è memorizzata nelle variabili `mese` e `giorno`.

Successivamente avvengono i due controlli che la data sia legale e che sia minore della data minima corrente. Infine viene letto un ulteriore carattere per saltare il carattere `','` eventualmente presente.

```
#include <stdio.h>

int GiorniDelMese(int m)
{ /* funzione non richiesta per l'esercizio */
  switch (m)
    {
    case 4: case 6: case 9: case 11:
      return 30;
    case 2:
      return 28;
    default:
      return 31;
    }
}

int StudentiPrimaLezione(char* nome_file_studenti)
{
  int studenti, studenti_prima_lezione;
  int giorno, mese;
  int giorno_prima_lezione,
      mese_prima_lezione = 13;
  char ch;
  FILE* fp;

  fp = fopen(nome_file_studenti,"r");
  while (fscanf(fp,"%d",&studenti) == 1)
    {
      do
        ch = getc(fp);
      while (ch != '(');
      fscanf(fp,"%d/%d)",&giorno,&mese);
      if (mese > 12 || mese < 1 || giorno > GiorniDelMese(mese))
```

```
        return 0;
      if (mese < mese_prima_lezione
          || (mese == mese_prima_lezione && giorno < giorno_prima_lezione))
        {
          mese_prima_lezione = mese;
          giorno_prima_lezione = giorno;
          studenti_prima_lezione = studenti;
        }
      getc(fp); /* legge l'eventuale virgola */
    }
  fclose(fp);
  return studenti_prima_lezione;
}

main()
{ /* programma principale di prova (non richiesto) */
  printf("Studenti prima lezione: %d\n",StudentiPrimaLezione("presenze.dat"));
}
```

Soluzione esercizio 2 del 15-4-97

Si decide di restituire i parametri per variabile, ovvero passando alla funzione il loro indirizzo. Per quanto riguarda la scansione del file si leggono i caratteri nel primo elemento di un array di caratteri di appoggio (`char parola[21]`) verificando che siano spazi ed incrementando un contatore. Quando si incontra un carattere diverso da spazio si aggiorna, se necessario, il numero massimo di spazi consecutivi trovati nel file, e si effettua un test per vedere se il carattere incontrato sia un `'\n'` od il primo carattere di una stringa. In quest'ultimo caso, essendo il primo carattere della stringa già memorizzato in `parola[0]` ed essendo le stringhe lunghe almeno 2 caratteri è possibile effettuare una lettura del resto della stringa memorizzandola a partire da `parola[1]`.

```
#include<stdio.h>

void conta(char *nfile, char **lunga, int *nparole, int *nspazi)
{
  FILE *f;
  char parola[21], maxparola[21];

  int spazi;
  spazi = *nparole = *nspazi = 0;

  f = fopen(nfile, "r");
  maxparola[0] = '\0';
  while ( fscanf(f, "%c", &parola[0]) != EOF )
    {
     if (parola[0] == ' ') ++spazi;
     else
       { /* la sequenza di spazi e' terminata */
        if (spazi > *nspazi)
          *nspazi = spazi;
        spazi = 0;
        if (parola[0] != '\n')
          { /* stiamo leggendo una stringa il cui primo carattere e' gia' in
               parola[0] */
           fscanf(f, "%s", &parola[1] ); /* legge il resto della stringa */
           if (strlen(parola) > strlen(maxparola) )
```

```
            strcpy(maxparola, parola);
           *nparole = *nparole+1;
          }
       }
    }
  *lunga = (char *) malloc (strlen(maxparola)+1);
  strcpy(*lunga, maxparola);
  fclose(f);
}

main () /* programma principale di test, non richiesto dal compito */
{       /* si assume un file di nome pippo.txt presente sul disco */
  int parole, spazi;
  char *lunga;
  conta("pippo.txt",&lunga,&parole,&spazi);
  printf("parola piu' lunga : %s\n",lunga);
  printf("numero parole: %d numero max spazi %d",parole,spazi);
}
```

Soluzione esercizio 1 del 3-6-97

La funzione `VerificaParole()` legge una parola alla volta dal file utilizzando il formato `%s` della funzione `fscanf()`). Ad ogni lettura, l'ultimo carattere della stringa (che è `';'` oppure `'.'`) viene eliminata dalla stringa e memorizzato nella variabile `ch`. La variabile `ch` viene utilizzata per la condizione di permanenza nel ciclo di lettura. Ovviamente, come condizione non può essere usato il test sul valore restituito dalla funzione `fscanf()` in quanto il file può contenere altri caratteri dopo il punto.

Il controllo di appartenenza al vettore viene eseguito dala funzione esterna `appartiene()`.

```
#include <stdio.h>

int appartiene(char* buffer, char *vet[], int n)
{ /* funzione ausiliaria: verifica l'appartenenza di una
     parola al vettore di parole */
  int i;
  for (i = 0; i < n; i++)
    if (strcmp(buffer,vet[i]) == 0)
      return 1;
  return 0;
}

int VerificaParole(char* nomefile, char *vet[], int n)
{ /* funzione principale */
  int lunghezza;
  char ch;
  FILE* fp;
  char buffer[22]; /*   20 caratteri della parola
                     + 1 per il punto e virgola
                     + 1 per il terminatore di stringa */
  fp = fopen(nomefile,"r");
  do
    { fscanf(fp,"%s",buffer);
      lunghezza = strlen(buffer);
      ch = buffer[lunghezza-1];   /* ch == ';' oppure ch == '.' */
      buffer[lunghezza-1] = '\0'; /* elimina ch dalla parola */
      if (!appartiene(buffer,vet,n))
```

```
        { fclose(fp);
          return 0;
        }
    }
  while (ch == ';'); /* se ch == '.' esce dal ciclo */
  fclose(fp);
  return 1;
}

main()
{ /* programma principale di prova (non richiesto dal testo) */
  char *v[] = {"comodino", "armadio", "letto", "tappeto", "lampada"};
  char *file = "sequenza.txt";
  if (VerificaParole(file,v,5))
    printf("Sequenza verificata\n");
  else
    printf("Sequenza non verificata\n");
}
```

Soluzione esercizio 2 del 3-6-97

La soluzione proposta si basa sull'idea di scorrere il file 3 volte. La prima volta viene calcolato il numero n dei reali presenti nel file e viene allocato un vettore di interi della dimensione n. Nella seconda scansione i valori letti vengono arrotondati e memorizzati nel vettore. Nella terza scansione i valori nel vettore vengono riscritti nel file (aperto in scrittura).

Possibili soluzioni alternative sono:

- utilizzare un file di appoggio dove copiare i valori letti
- fare una sola scanzione di lettura utilizzando la funzione `realloc()` per il vettore, oppure memorizzando i dati in una lista collegata

```
#include <stdio.h>

int Arrotonda(float x)
{ return x + 0.5; } /* conversione automatica da float a int */

void ArrotondaFile(char* nomefile)
{
  float x;
  FILE* fp;
  int* v;     /* vettore di interi per memorizzare
                 la sequenza gia' arrotondata */
  int i = 0,  /* indice del vettore */
      n = 0;  /* numero di elementi nel file */

  fp = fopen(nomefile,"r");
  while (fscanf(fp,"%*f") != EOF)
    n++;
  v = (int*) malloc(n* sizeof(int));
  rewind(fp);
  for (i = 0; i < n; i++)
     {
       fscanf(fp,"%f",&x);
       v[i] = Arrotonda(x);
     }
```

```
  fclose(fp);
  fp = fopen(nomefile,"w");
  for (i = 0; i < n; i++)
    fprintf(fp,"%d ",v[i]);
  fclose(fp);
}

main()
{ /* programma principale di prova (non richiesto) */
  ArrotondaFile("reali.dat");
}
```

Soluzione esercizio 1 del 17-6-97

Per rappresentare i dati di ingresso utilizziamo un vettore dinamico di dimensione pari al numero di lettere dell'alfabeto (cioè 26), che identifichiamo tramite l'espressione `'z' - 'a' + 1`.

Per ogni parola lunga più di sei lettere, viene aggiornato il vettore per ogni lettera della parola.

```
#include <stdio.h>

int* conta(char *nome)
{
  FILE *p;
  char a[21];
  int *vett,i;
  p = fopen(nome,"r");
  vett = (int *) calloc(('z'-'a'+1),sizeof(int)); /*allocazione dinamica !!*/
  while (fscanf(p,"%s",a)!=EOF)
    if (strlen(a) > 6)
      for (i = 0; i < strlen(a); ++i)
        ++vett[a[i]-'a'];
  return vett;
}

main()
{ /* programma principale di prova (non richiesto) */
  int *v,i;
  v = conta("dati1.txt");
  for (i = 'a'; i <= 'z'; ++i)
    if (v[i-'a'])
      printf("Ci sono %d occorrenze del carattere %c \n",v[i-'a'],i);
}
```

Soluzione esercizio 1 del 1-7-97

La soluzione si compone di tre passi:

1. Si alloca dinamicamente una matrice di dimensione $n \times n$ (quindi con indici da 0 a $n-1$) e si pongono a zero tutti i suoi elementi.

2. Gli elementi significativi della matrice vengono letti dal file di input e memorizzati nella matrice.

In particolare, l'elemento di posizione (i, j) viene scritto nella locazione $(i-1, j-1)$ della matrice. In questo modo si gestisce il fatto che la matrice originaria ha indici che variano da 1 ad n.

3. La matrice viene trascritta integralmente sul file di output. Contemporaneamente, la matrice in memoria centrale viene deallocata.

Si noti che dal testo si evince che il carattere `';'` è sempre "attaccato" al numero che lo precede. Sfruttando questo fatto è possibile leggere questo carattere insieme ai tre numeri che lo precedono con un'unica istruzione `fscanf()`.

```
#include <stdio.h>

void ConvertiMatrice(char *nomefileinput, char* nomefileoutput, int n)
{
  float **mat;
  float val;
  int i, j;
  FILE *ifp, *ofp;

  mat = (float**) calloc(n, sizeof(float*));
  for (i = 0; i < n; i++)
    mat[i] = (float*) calloc(n, sizeof(float));
  /* calloc azzera automaticamente tutte le locazioni */

  ifp = fopen(nomefileinput,"r");
  while (fscanf(ifp,"%d%d%f;",&i,&j,&val) != EOF)
    mat[i-1][j-1] = val;
  fclose(ifp);

  ofp = fopen(nomefileoutput,"w");
  for (i = 0; i < n; i++)
    {
      for (j = 0; j < n; j++)
        fprintf(ofp,"%f ",mat[i][j]);
      fprintf(ofp,"\n");
      free(mat[i]);
    }
  fclose(ofp);
  free(mat);
}

main()
{/* programma principale di prova (non richiesto) */
  ConvertiMatrice("Compatta.dat","Estesa.dat",5);
}
```

Soluzione esercizio 1 del 14-7-97

Utilizziamo due stringhe `pi` e `pj` che vengono inizializzate con la stringa `MANCANTE`. Successivamente, vengono cercate le stringhe corrispondenti agli indici `i` e `j`, e, se queste esistono, vanno a sostituire la stringa iniziale.

Infine, si alloca dinamicamente e si compone (utilizzando `strcat()` la stringa risultato.

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

```
char * giusta(char* nomefile, int i, int j)
{
  FILE *p;
  char buffer[21];
  char pi[21] = "MANCANTE";
  char pj[21] = "MANCANTE";
  char *ris;
  int tot=1,lette=0;
  p=fopen(nomefile,"r");
  while ( (fscanf(p,"%s",buffer)!=EOF) && (lette<2) )
    {
      if (tot==i) {
        strcpy(pi,buffer);
        ++lette;
      }
      if (tot==j) {
        strcpy(pj,buffer);
        ++lette;
      }
      ++tot;
    }
  ris = (char *) malloc(strlen(pi)+strlen(pj)+2);
  strcpy(ris,pi);
  strcat(ris," ");
  strcat(ris,pj);
  return ris;
}

main()
{/* programma principale di prova (non richiesto) */
  printf("1 e 5 : %s\n",giusta("pippo",1,5));
  printf("5 e 1 : %s\n",giusta("pippo",5,1));
  printf("5 e 5 : %s\n",giusta("pippo",5,5));
  printf("500 e 5 : %s\n",giusta("pippo",500,5));
  printf("5 e 500 : %s\n",giusta("pippo",5,500));
  printf("500 e 500 : %s\n",giusta("pippo",500,500));
}
```

Soluzione esercizio 1 del 17-9-97

La soluzione che proponiamo non si basa sull'assunzione che il numero di servizi sia necessariamente pari a quattro e che questi siano posti in un particolare ordine.

La prima fase della soluzione, realizzata dalla funzione `PosizioneServizio()` identifica la posizione del servizio richiesto all'interno della prima riga del file. Tale posizione viene memorizzata nella variabile `pos`.

Nel ciclo principale del programma per ogni albergo si effettuano le seguenti operazioni sul file

- Si ignora il nome (che non è richiesto) e si legge il suo prezzo
- Si ignorano le prime `pos-1` stringhe (che riguardono i servizi che precedono quello richiesto)
- Si legge il valore corrispondente al servizio richiesto. Se il valore è `"SI"` allor si cofronta il prezzo dell'albergo con quello minimo memorizzato fino a quel punto.

- Si ignorano i servizi successivi portandosi avanti nel file fino ad incontrare il carattere `'\0'`. Questo viene fatto dalla funzione `RigaNuova()`.

```
#include <stdio.h>

void RigaNuova(FILE *fp)
{ /* funzione ausiliaria: muove il cursore fino al primo
     carattere della riga successiva */
  int ch;
  while ((ch = fgetc(fp)) != '\n' && ch != EOF)
    ;
}

int PosizioneServizio(FILE *fp, char* servizio)
{ /* funzione ausiliaria: cerca sulla prima riga del file la posizione del
   servizio richiesto */
  char buffer[21];
  int i = 0;
  fscanf(fp,"%*s%*s");
  do
    { i++;
      fscanf(fp,"%s",buffer);
    }
  while (strcmp(buffer,servizio) != 0);
  RigaNuova(fp);
  return i;
}

int CercaHotel(char* nomefile, char* servizio)
{/* funzione principale dell'esercizio */
  int pos, i;
  FILE *fp;
  char presenza_opzione[3];
  int prezzo, prezzo_min = -1;

  fp = fopen(nomefile,"r");
  pos = PosizioneServizio(fp,servizio);
  while (fscanf(fp,"%*s%d", &prezzo) != EOF)
    {
      for (i = 1; i <= pos; i++)
        fscanf(fp,"%s",presenza_opzione);
      if (strcmp(presenza_opzione,"SI") == 0)
        if (prezzo_min == -1 || prezzo < prezzo_min)
          prezzo_min = prezzo;
      RigaNuova(fp);
    }
  return prezzo_min;
}

main()
{ /* programma principale (non richiesto dal testo) */
  char servizio[21];
  printf("Servizio richiesto : ");
  scanf("%s",servizio);
  printf("Prezzo minimo : %d\n", CercaHotel("alberghi.txt",servizio));
}
```

Soluzione esercizio 1 del 21-10-97

La funzione `decifra()` utilizza due funzioni ausiliarie: `stampa_parola()` per stampare una parola e `next_parola()` per cercare nel file la parola successiva.

La funzione `decifra()` si compone di un ciclo che chiama le due funzioni ausiliarie fino alla fine del messaggio, che viene riconosciuta dalla presenza del carattere `'.'` alla fine della parola.

Per eliminare i caratteri in più all'inizio di ogni parola, si chiama la funzione `stampa_parola()` con parametro `buffer+2` e non `buffer` (tranne che per la prima parola).

Per eliminare i caratteri in più alla fine della parola si inserisce il terminatore di stringa nella parola da stampare in posizione `l-2`, dove `l` è la lunghezza della stringa.

```
#include <stdio.h>

void  stampa_parola(FILE *p, char* next, char b[2])
{ /* funzione ausiliaria: stampa sul file una parola del messaggio
     e contemporaneamente scrive in b le due lettere per trovare
     la parola successiva */
  int l;
  l = strlen(next);
  b[0] = next[l-2];
  b[1] = next[l-1];
  if (b[1] != '.')      /* Se non e' l'ultima parola */
    next[l-2]= '\0';    /* allora elimina gli ultimi due caratteri*/
  fprintf(p,"%s ",next);
}

void next_parola(FILE *p, char *s, char b[2])
{ /* funzione ausiliaria: cerca la parola successiva nel file */
  rewind (p);
  while (fscanf(p,"%s",s)!=EOF)
    if (s[0] == b[0] && s[1] == b[1] ) break;
}

void decifra(char * nomefile){
   FILE *in, *out;
   char buffer[21], fine[2];

   in=fopen(nomefile,"r");
   out=fopen("frase.txt","w");

   if (fscanf(in,"%s",buffer)!=EOF)
     { /* C'e' almeno una parola */
       fine[1]=' ';
       stampa_parola(out,buffer,fine); /* stampa la prima parola */

       while (fine[1]!='.'){
         next_parola(in,buffer,fine);
         stampa_parola(out,buffer+2,fine);
       }
       fclose(in);
       fclose(out);
     }
}

main()
{/* programma principale di prova (non richiesto) */
```

```
  decifra("mess.txt");
}
```

Soluzione esercizio 1 del 19-1-98

L'esercizio viene risolto dalla funzione `MassimoRopporto()` che utilizza la funzione `LeggiRapporto()` per la lettura del rapporto.

La funzione ausiliaria `LeggiRapporto()` legge dal file identificato da `fp` il rapporto (in qualsiasi formato sia) e lo restituisce come `float`; il cursore di `fp` viene lasciato alla fine della riga oppure all'inizio della riga successiva.

Nella funzione `LeggiRapporto()`, per gestire correttamente tutti i casi, il primo valore viene letto comunque come un `float`. Nel caso questo sia un intero, la funzione `scanf()` è tale che esso viene automaticamente convertito a `float` (e il cursore si ferma alla fine del numero, cioè prima del carattere `'/'`).

Nella funzione `MassimoRopporto()` il nome della nazione con il rapporto più alto viene temporaneamente memorizzato in un vettore statico, alla fine del ciclo il nome è copiato in un vettore dinamico della dimensione esatta necessaria.

```
#include <stdio.h>

float LeggiRapporto(FILE* fp)
{
  float val;
  int denominatore;
  char ch;

  fscanf(fp,"%f",&val);
  ch = fgetc(fp);
  if (ch == '%')
    return val/100;
  else if (ch == '/')
    {
      fscanf(fp,"%d",&denominatore);
      return val/denominatore;
    }
  else /* ch e' \n (oppure un blank spurio alla fine della riga) */
    return val;
}

char* MassimoRapporto(char* nome_file)
{
  char nome_nazione[21], nome_nazione_massimo[21];
  char* nome_restituito;
  float rapporto, rapporto_massimo = 0;
  FILE* fp;

  fp = fopen(nome_file,"r");
  while(fscanf(fp,"%s",nome_nazione) != EOF)
    {
      rapporto = LeggiRapporto(fp);
      if (rapporto > rapporto_massimo)
        {
          strcpy(nome_nazione_massimo,nome_nazione);
          rapporto_massimo = rapporto;
        }
```

```
    }
  fclose(fp);
  nome_restituito = (char*) malloc(strlen(nome_nazione_massimo) + 1);
  strcpy(nome_restituito,nome_nazione_massimo);
  return nome_restituito;
}

main()
{ /* programma principale: non richiesto per il compito */
  printf("Il massimo rapporto e' quello di %s\n",
         MassimoRapporto("rapporti.txt"));
}
```

Soluzione esercizio 1 del 2-2-98

La funzione alloca un vettore di n studenti che viene mantenuto ordinato con i migliori n tra quelli letti fino a quel momento. Ogni nuovo studente letto viene confrontato inizialmente con l'ultimo (posizione $n-1$) del vettore e successivamente fatto "salire" (tipo *bubblesort*) fino alla sua posizione finale. La funzione utilizza la funzione `calloc()` e sfrutta il fatto che la `calloc()` azzera tutte le locazioni, quindi inizialmente il vettore è vuoto con tutte le medie poste pari a 0.

```
#include <stdio.h>

struct studente
{
  float media;
  char nome[21];
};

struct studente* MiglioriEnneStudenti(char* nome_file, int n, int a)
{
  FILE *fp;
  int eta, i;
  struct studente* vet;
  struct studente stud;

  fp = fopen(nome_file,"r");
  vet = (struct studente*) calloc(n,sizeof(struct studente));

  while(fscanf(fp,"%s%d%f",stud.nome,&eta,&(stud.media)) != EOF)
    {
      if (eta < a && stud.media > vet[n-1].media)
        {
          vet[n-1] = stud;
          for (i = n-2; i >= 0; i--)
            if (vet[i+1].media > vet[i].media)
              { struct studente temp = vet[i+1];
                vet[i+1] = vet[i]; vet[i] = temp; }
            else
              break;
        }
    }
  return vet;
}
```

```
main()
{ /* programma principale di prova (non richiesto) */
  int i, n, e;
  struct studente* s;
  printf("Eta' massima : ");
  scanf("%d",&e);
  printf("Numero studenti : ");
  scanf("%d",&n);
  s = MiglioriEnneStudenti("studenti.txt",n,e);
  for (i = 0; i < n; i++)
    printf("%s\t%.2f\n",s[i].nome,s[i].media);
}
```

Soluzione esercizio 2 del 2-2-98

Sfruttando il fatto che è nota il valore massimo presente nei file (parametro `n`), basiamo la soluzione sull'utilizzo di due vettori di interi (allocati dinamicamente) che contano le occorrenze dei valori tra `1` ed `n`. Per comodità i vettori vengono allocati di dimensione `n+1` e la locazione `0` non viene utilizzata.

Dopo aver scritto i due vettori con le informazioni estratte dai file, si scrivono tutti gli elementi diversi da zero per un numero di volta pari alla loro molteplicità; quest'ultima è ottenuta semplicemente tramite la funzione `Max()`.

```
#include <stdio.h>

int Max(int i, int j)
{
  if (i > j) return i;
  else return j;
}

void UnioneMultiinsiemi(char* file_m1, char* file_m2, char* file_ris, int n)
{
  FILE *fp1, *fp2, *fpr;
  int *v1, *v2;
  int i,j;

  fp1 = fopen(file_m1,"r");
  fp2 = fopen(file_m2,"r");
  fpr = fopen(file_ris,"w");

  v1 = (int *) calloc(n+1,sizeof(int));
  v2 = (int *) calloc(n+1,sizeof(int));

  while(fscanf(fp1,"%d",&i) != EOF)
    v1[i]++;
  while(fscanf(fp2,"%d",&i) != EOF)
    v2[i]++;

  for (i = 1; i <= n; i++)
    {
      for (j = 0; j < Max(v1[i],v2[i]); j++)
        fprintf(fpr,"%d ",i);
    }

  free(v1);
```

```
  free(v2);

  fclose(fp1);
  fclose(fp2);
  fclose(fpr);
}

main()
{/* programma principale di prova (non richiesto) */
  UnioneMultiinsiemi("m1.dat","m2.dat","m3.dat",100);
}
```

Soluzione esercizio 1 del 19-2-98

Con l'assunzione che i nomo composti siano scritti utilizzando il carattere `'_'`, è possibile risolvere il problema utilizzando per la lettura unicamente un ciclo governato dalla funzione `fscanf()`.

Per leggere e trascrivere correttamente la matricola è necessario utilizzare una stringa e non un numero intero. Infatti utilizzando un intero non sarebbe possibile gestire correttamente il caso che la matricola contenga uno o più zero al principio.

Si noti che il nome del file di output è dato come costante e quindi non viene passato come parametro alla funzione.

```
#include <stdio.h>

void TrascriviEsame(char* nome_file_input)
{ FILE* ifp, *ofp;
  char nome[21], cognome[21], matricola[9];

  ifp = fopen(nome_file_input, "r");
  ofp = fopen("lista.txt", "w");

  while (fscanf(ifp,"%*d%s%s%s",matricola,cognome,nome) != EOF)
    fprintf(ofp,"%s,%s,%s\n",matricola,cognome,nome);

  fclose(ifp);
  fclose(ofp);
}

main()
{ /* programma principale di prova (non richiesto) */
  TrascriviEsame("esame.txt");
}
```

Soluzione esercizio 1 del 16-4-98

Il programma utilizza un vettore dimanico di dimensione pari al numero di transistor per nemorizzare la potenza totale di ciascun transistor (che si ottiene sommando tutte le misure corrispondenti).

Per la scansione del file di ingresso è necessario che dopo la lettura di ciascuna misura il cursore sia riportato all'inizio della riga successiva in modo da separare correttamente il numero del transistor dai caratteri `'T'` ed `'R'`.

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

```
void potenza()
{
  FILE *p;
  float *v, t, vce, ie;
  int i, n;
  char app = 'k';

  p = fopen("corrente.txt","r");
  fscanf(p,"%d",&n);
  v = (float *) calloc(n,sizeof(float));
  while(app != '\n')
    fscanf(p,"%c",&app); /* porta il cursore all'inizio della seconda riga */

  while (fscanf(p,"%*c%*c%d%f%f%f",&i,&t,&vce,&ie) != EOF)
    {
      v[i-1] += vce * ie * t;
      app = 'k';
      while( (app!='\n') && (fscanf(p,"%c",&app) != EOF) );
    }
  fclose(p);

  p = fopen("totale.txt","w");
  for (i = 0; i < n; ++i)
    fprintf(p,"TR%d %f\n",i+1,v[i]);
  fclose(p);
  free(p);
}

main()
{ /* programma principale di prova (non richiesto) */
  potenza();
}
```

Capitolo 6

Soluzioni degli esercizi sui bit

Soluzione esercizio 3 del 17-6-96

La soluzione proposta si basa sull'idea utilizzare un puntatore a carattere per accedere ad uno specifico byte dell'intero. Si noti che la conversione di tipo effettuata nell'assegnare ad `in` ed `out` gli indirizzi degli interi fa sì che gli incrementi di indirizzo (ad esempio `in[0]`, `in[1]`) siano esattemente di un byte.

```
#include <stdio.h>

unsigned int scambia (unsigned int i)
{
  char *in, *out;
  unsigned int ris;

  in = (char *)&i;  /* assegno a in l'indirizzo di i e quindi ho che in[0]
                       e' il primo byte di i e in[1] e' il secondo byte */

  out=(char *)&ris; /* assegno ad out l'indirizzo di ris e quindi ho che out[0]
                       e' il primo byte di ris e out[1] e' il secondo byte */

  out[0] = in[1];   /* modifico ris tramite out */
  out[1] = in[0];
  return ris;
}

main () /* programma principale di test, non richiesto dal compito */
{
  unsigned int i,j;
  i = 1;           /* i= 00000000 00000001 */
  j = scambia(i);  /* j= 00000001 00000000  ovvero 256 */
  printf("i vale: %d ,  j vale: %d\n",i, j);

  i = 129;         /* i= 0000000 10000001 */
  j = scambia(i);  /* j= 10000001 00000000  ovvero -32512 */
  printf("i vale: %d ,  j vale: %d\n",i, j);
}
```

Soluzione esercizio 3 del 15-7-96

Il modo piú semplice per risolvere l'esercizio è quello di mettere due valori noti e distinti nei due byte assegnati all'intero per poi controllare, tramite un puntatore a carattere, quale valore viene memorizzato nel byte di indirizzo minore.

In particolare, decidiamo di memorizzare `2 (in binario 00000010)` nel byte più significativo ed `1 (in binario 00000001)` in quello meno significativo. In questo modo il byte di indirizzo minore conterrà direttamente il valore che la funzione deve restituire.

Per poter memorizzare tali valori nell'intero è sufficiente assegnare allo stesso il valore decimale 513 che in complemento a 2 vale `0000001000000001`. Si noti che non è necessario passare alcun parametro alla funzione, essendo 513 una costante decisa a priori.

```
int final()
{
  char *a;
  int i = 513;           /* 00000010 00000001 */
                         /* il byte + significativo contiene 2
                            il byte - significativo contiene 1 */

  a = (char *)&i;       /* a punta al byte con l'indirizzo + basso di i
                           che conterra' 1 nel caso di finale piccolo
                           o 2 nel caso di finale grande. Quindi :     */

   return(*a);
}

main () /* programma principale di test, non richiesto dal compito */
{
  if (final() == 1)
    printf("finale piccolo");
  else
    printf("finale grande");
}
```

Soluzione esercizio 2 del 17-9-96

La soluzione di questo esercizio si basa sulla memorizzazione in una stringa (di 20 caratteri) delle stringhe via via lette dal file d'ingresso. A tale scopo utilizziamo il formato `%s` della funzione `fscanf()`, che memorizza i caratteri significativi (`'0'` e `'1'` in questo caso) e ignora gli spazi bianchi e i ritorni a capo.

Si noti che la cifra più significativa del numero binario viene a trovarsi nella posizione 0 del vettore. Per scandire la stringa e ricostruire il numero binario in essa contenuto, ispezioniamo le sue cifre a partire dalla più significativa (quindi dalla locazione 0 del vettore).

Per ogni cifra letta, il risultato accumulato viene moltiplicato per 2 perché la presenza della nuova cifra fa aumentare di un posto la significatività delle cifre precedentemente lette. La cifra letta viene quindi sommata al risultato precedentemente accumulato moltiplicato per 2.

```
#include <stdio.h>

int Somma(char *nomefile)
{
  FILE *fp;
  int i, valore, somma = 0;
  char a[21];
```

```
  if ((fp = fopen(nomefile,"r")) == NULL)
    return -1;

  while (fscanf(fp,"%s",a) == 1)
    {
      valore = 0;
      for (i = 0; i < strlen(a); i++)
        if (a[i] == '1')
          valore = valore*2 + 1;
        else       /* a[i] vale '0' */
          valore = valore*2;
      somma += valore;
    }
  close(fp);
  return somma;
}

main()
{ /* programma principale di prova (non richiesto) */
  char file[20];
  printf("Nome file: ");
  scanf("%s",file);
  printf("Somma = %d\n",Somma(file));
}
```

Soluzione esercizio 3 del 17-9-96

Utilizziamo un ciclo composto da 4 iterazioni per estrarre le 4 cifre memorizzate nel parametro di ingresso, chiamato `bcd`, di tipo `unsigned int`.

Per poter isolare 4 bit alla volta all'interno di una variabile che occupa 2 byte di memoria facciamo uso di una maschera. La maschera inizialmente ha il valore binario `0000 0000 0000 1111` (corrispondente a `000F` esadecimale), e viene traslata a sinistra di 4 posizioni ad ogni iterazione del ciclo.

All'interno del ciclo, la variabile `bcd` viene posta in "AND" con la maschera e traslata a destra di un numero di posizioni necessarie affinché il risultato occupi i 4 bit meno significativi della variabile `cifra` a cui è assegnata.

La ricostruzione del risultato decimale corrispondente avviene utilizzando la variabile `coefficiente` che assume i valori 1, 10, 100 e 100 nelle quattro iterazioni del ciclo.

```
#include <stdio.h>

unsigned int ConvertiBCD(unsigned int bcd)
{
  unsigned int cifra, risultato = 0,
               maschera = 0x000F, coefficiente = 1, i;

  for (i = 0; i < 4; i++)
    {
      cifra = (bcd & maschera) >> 4*i;
      if (cifra > 9)
        return 10000;
      else
        risultato += coefficiente * cifra;
      maschera = maschera << 4;
```

```
      coefficiente *= 10;
    }
  return  risultato;
}

main()
{ /* programma principale di prova (non richiesto) */
  unsigned int n;
  printf("Inserisci numero: ");
  scanf("%x",&n); /* legge il numero in esadecimale */
  printf("%d\n",ConvertiBCD(n));
}
```

Soluzione esercizio 3 del 17-1-97

La zona di memoria viene scandita facendo uso di un puntatore a caratteri `v` ed un indice intero `i`. La condizione di uscita è che venga incontrato un carattere di valore 0, cioè che sia verificata la condizione `v[i] == 0`.

Ad ogni iterazione, il byte contenuto in `v[i]` viene copiato nella variabile `a`. I bit della variabile `a` vengono sommati partendo dal meno significativo. Il bit meno significativo viene ottenuto tramite l'operazione "modulo 2", e ad ogni iterazione la variabile `a` viene traslata a destra tramite la divisione per due (equivalentemente avremmo potuto usare l'operatore "`>>`").

```
#include <stdio.h>

int MemoriaIntegra(char* v)
{
  int somma_bit, i = 0, j;
  char a;

  while (v[i] != 0)
    {
      somma_bit = 0;
      a = v[i];
      for (j = 0; j < 8; j++)
        {
          somma_bit += a % 2;
          a /= 2;
        }
      if (somma_bit % 2 == 1)
        return 0;
      i++;
    }
  return 1;
}

void StampaMemoria(char* v)
{ /* funzione ausiliaria (non richiesta) */
  char a;
  int i = 0, j;
  char stringa_byte[9];

  printf("Stampa della memoria in binario\n");
  while (v[i] != 0)
    {
```

```
      a = v[i];
      for (j = 7; j >= 0; j--)
        {
          if (a % 2 == 1)
            stringa_byte[j] = '1';
          else
            stringa_byte[j] = '0';
          a /= 2;
        }
      stringa_byte[8] = '\0';
      printf("%s\n",stringa_byte);
      i++;
    }
}

main()
{ /* programma principale di prova (non richiesto) */
  char vett[20];
  printf("Scrivi 20 byte di memoria (come stringa di caratteri) ");
  scanf("%s",vett);
  if (MemoriaIntegra(vett))
    printf("La memoria e' integra\n");
  else
    printf("La memoria e' corrotta\n");
  StampaMemoria(vett);
}
```

Soluzione esercizio 3 del 30-1-97

Le specifiche dell'esercizio rendono impossibile sommare direttamente i byte visti come caratteri. Il problema nasce dal fatto che il numerale corrispondente ad un intero (di uno o più byte) viene valutato in complemento a 2 che, per i numeri negativi, fornisce valori differenti dalla rappresentazione modulo e segno. Ad esempio, assumiamo che due byte contengano i numerali `00000001` e `10000001`. Il primo numerale corrisponde al valore decimale 1 (sia in complemento a 2 che nella notazione modulo e segno), mentre il secondo numerale vale -127 in complemento a 2 e -1 in modulo e segno.

Occorre, quindi, nel ciclo che scandisce la memoria estrarre il modulo con la maschera `01111111` e valutarne il segno con la maschera `10000000`.

```
#include <stdio.h>

long int leggi(char *p)      /* oppure int leggi(unsigned char *p) */
{
  int  val;
  long int somma=0;

  do
    {
     val = *p & 0x7F;  /* Estrae il modulo con la maschera 01111111 (0x7F) */
     if ( (*p & 0x80) == 0x80) /* Verifica il bit di segno con 10000000 (0x80)*/
       val = -val;             /* se 1 cambia il segno al modulo */
     somma = somma + val;
     ++p;
    }
  while ( *p != *(p-1) );
  return somma;
```

```
}

main () /* programma principale di test, non richiesto dal compito */
{
   int i;
   char *a = "23dd";                    /* 2 codice ASCII 50  00110010 */
                                        /* 3 codice ASCII 51  00110011 */
                                        /* d codice ASCII 100 01100100 */
   printf("%s vale %d\n",a,leggi(a));   /* 50+51+100 */
   a[0] = 129;
   printf("%s vale %d\n",a,leggi(a));   /* -1+51+100 */
}
```

Soluzione esercizio 3 del 3-6-97

La funzione `InvertiIntero()` invoca la funzione ausiliaria `InvertiByte()` per invertire ogni singolo byte che compone l'intero. La funzione `InvertiByte()` scrive il byte invertito nella variabile `dest`. In particolare, la funzione isola ogni singolo bit del byte di ingresso a partire dal meno significativo ($i = 0$) e lo scrive a partire dal piu' significativo in un byte in uscita in posizione $7 - i$. A questo scopo viene utilizzata una maschera che ha valore iniziale `0000 0001` e viene traslata di un bit ad ogni iterazione.

```
#include <stdio.h>

void InvertiByte(unsigned char* p)
{/* funzione ausiliaria: inverte un singolo byte */
  unsigned char dest = 0;
  int i, bit, maschera = 1;
  for (i = 0; i < 8; i++)
    {
      bit = (*p & maschera) >> i;  /* isola il bit i-esimo di *p */
      dest = dest | (bit << 7 - i);
      maschera = maschera << 1;
    }
  *p = dest;
}

void InvertiIntero(int* p)
{/* funzione principale: inverte un intero */
  int i;
  unsigned char* q = (unsigned char *) p;
  for (i = 0; i < sizeof(int); i++)
    InvertiByte(&q[i]);
}

void StampaMemoria(unsigned char* v, int n)
{ /* funzione ausiliaria (non richiesta) */
  unsigned char a;
  int i = 0, j;
  char stringa_byte[9];

  printf("Stampa della memoria in binario\n");
  while (i < n)
    { a = v[i];
      for (j = 7; j >= 0; j--)
        { if (a % 2 == 1)
```

```
            stringa_byte[j] = '1';
          else
            stringa_byte[j] = '0';
          a /= 2;
        }
      stringa_byte[8] = '\0';
      printf("%s\n",stringa_byte);
      i++;
    }
}

main()
{ /* programma principale di prova (non richiesto) */
  int i;
  printf("Valore da invertire in esadecimale : ");
  scanf("%x",&i);
  StampaMemoria((unsigned char*)&i,sizeof(int));
  InvertiIntero(&i);
  StampaMemoria((unsigned char*)&i,sizeof(int));
  printf("%x\n",i);
}
```

Soluzione esercizio 3 del 17-6-97

La funzione scorre la zona di memoria dalla prima locazione fino a quando non incontra il byte composto di tutti 1 (valore 0xFF).

Ad ogni nuovo byte incontrato, il valore corrente accumulato aumento la sia significatività di 100 (due cifre decimali) e gli viene sommato il numero contenuto nel byte. Tale numero viene ottenuto estraendo le due cifre in esso contenute e sommarle moltiplicando la prima per 10.

```
#include <stdio.h>

long int somma(unsigned char *m)
{
  long int ris=0;
  while (*m != 0xFF)
    {
      ris = 100*ris + (*m >> 4) * 10 + (*m & 0x0F);
      ++m;
    }
  return ris;
}

main()
{/* programma principale di prova (non richiesto) */
  unsigned char mem[3];
  mem[0]=0x56;
  mem[1]=0x32;
  mem[2]=0xFF;
  printf("%ld \n",somma(mem));
}
```

Soluzione esercizio 3 del 14-7-97

La funzione si avvale di una maschera i cui valore è posto inizialmente a 7 (ultimi tre bit a `1` e il resto a `0`).

Successivamente la maschera viene traslata a destra e la stringa contenente al risultato viene riempita corrispondentemente. Tale stringa ha lunghezza pari a 7 (una locazione è per il terminatore), dato che 6 cifre vengono codificate in 16 bit. Si noti che la prima cifra può essere solo `0` oppure `1`.

```
#include <stdio.h>

char* converti(unsigned int n)
{
  char* out;
  unsigned int maschera = 7;         /* 7 = 00000000 00000111  */
  int i;

  out = (char *) malloc(7);
  out[6] = '\0';

  for (i = 0; i < 6; i++)
    {
      out[5-i] = '0' + ( (n&maschera)>>(i*3) );
      maschera = maschera * 8;  /* oppure maschera=maschera<<3 */
    }
  return out;
}

main()
{/* programma principale di prova (non richiesto) */
  printf("       3 = %s \n",converti(3) );
  printf("       8 = %s \n",converti(8) );
  printf("       9 = %s \n",converti(9) );
  printf("    128  = %s \n",converti(128) );
  printf("    257  = %s \n",converti(257) );
  printf("    8k   = %s \n",converti(1024*8) );
  printf("    64k-1= %s \n",converti(1024*64-1) );
}
```

Soluzione esercizio 3 del 17-9-97

La soluzione si avvale di una maschera che isola due bit alla volta. La maschera viene inizialmente posta al valore `0xc0000000` in modo da catturare i due bit più significativi e viene traslata a destra ad ogni iterazione di due bit.

Ad ogni iterazione la maschera viene posto in "AND" con il dato in ingresso il il risultato viene traslato di $32-2\cdot(i+1)$ in modo da portarlo nella parte meno significativa della variabile `pronostico`.

Ad ogni iterazione viene scritta la locazione i della stringa `colonna`, tranne nel caso che si incontri la coppia di bit `00`; in tal caso si esce dalla funzione restituendo la stringa vuota.

```
#include <stdio.h>
#define N 13

char* Schedina(unsigned int num)
{ unsigned int i, pronostico, maschera;
```

```
  char* colonna;

  colonna = (char*) malloc(N+1);
  maschera = 0xc0000000;  /* 1100 0000 0000 0000 ... */
  for (i = 0; i < N; i++)
    {
      pronostico = (num & maschera) >> (32 - 2*(i+1));
      switch (pronostico)
        {
        case 1: colonna[i] = '1'; break;
        case 2: colonna[i] = '2'; break;
        case 3: colonna[i] = 'X'; break;
        default:
          free(colonna);
          return "";   /* "" e' la stringa vuota */
        }
      maschera = maschera >> 2;
    }
  colonna[N] = '\0';
  return colonna;
}

main()
{ /* programma principale di prova (non richiesto) */
  unsigned int n;
  printf("Inserisci il numero (in esadecimale) : ");
  scanf("%x",&n);
  printf("La colonna e' : %s\n", Schedina(n));
}
```

Soluzione esercizio 3 del 21-10-97

Risolviamo l'esercizio in due modi. La funzione `CreaByte()` utilizza una maschera con il solo bit meno significativo posto a `1`. Ciascun byte viene posto in `AND` con la maschera per estrarne il bit meno significativo. Tale bit viene traslato (per posizionarlo correttamente nel byte di uscita) e viene posto in `OR` con il byte di uscita. Si utilizza l'`OR` in modo da non modificare i bit fino a quel momento calcolati.

La soluzione alternativa (funzione `CreaByte2()`) realizza la stessa soluzione utilizzando invece gli operatori di somma e modulo.

```
#include <stdio.h>

unsigned char CreaByte(char v[])
{
  int mask = 1, i;
  unsigned char byte = 0;
  for (i = 0; i < 8; i++)
    byte = byte | ((v[i] & mask) << 7-i);
  return byte;
}

unsigned char CreaByte2(char v[])
{ /* soluzione alternativa che utilizza le operazioni aritmetiche
     invece che le operazioni sui bit */
  int i, coeff = 1;
  unsigned char byte = 0;
```

```
  for (i = 7; i >= 0; i--)
    {
      byte += (v[i] % 2) * coeff;
      coeff *= 2;
    }
  return byte;
}

main()
{ /* programma principale (non richiesto) */
  char buffer[9];
  unsigned char ch;
  printf("Inserisci 8 caratteri : ");
  scanf("%s", buffer);
  ch = CreaByte(buffer);
  printf("Il byte cercato e' %c (codice ASCII %d)\n",ch,ch);
}
```

Soluzione esercizio 3 del 19-1-98

La stringa codificata nel parametro `num` viene decodificata e memorizzata nella stringa `tripletta` di quattro caratteri complessivi (tre significativi più il teminatore).

La decodifica utilizza una maschera con i 5 bit meno significativi ad `1`. Il numero viene traslato ad ogni iterazione per portare di volta in volta la parte in esame a conbacire con i cinque bit meno significativi.

Successivamente, la stringa `tripletta` viene confrontata con la stringa di ingresso `s` tramite la funzione di libreria `strcmp()`.

```
#include <stdio.h>

int Codifica(char* s, unsigned short int num)
{
  int j, i = 0;
  unsigned short int maschera = 0x001F; /* corrisponde a 0000 0000 0001 1111 */
  char tripletta[4];

  tripletta[3] = '\0';
  if (num >= 0x7000) /* corrisponde a 1000 0000 0000 0000 */
    return -1; /* primo bit ad 1 ---> restituisci -1*/
  for (j = 0; j < 3; j++)
    {
      if ((num & maschera) < 26)
        /* codifica correttamente una lettera dell'alfabeto */
        tripletta[2-j] = (num & maschera) + 'A';
      else /* codifica non corretta */
        return -1;
      num = num >> 5;
    }
  if (strcmp(tripletta,s) == 0)
    return 1;
  else
    return 0;
}

main()
{ /* programma principale di test (non richiesto dal compito) */
```

```
  int i;
  char parola[4];
  printf("Inserisci la parola di 3 lettere: ");
  scanf("%s",parola);
  printf("Inserisci un numero: ");
  scanf("%u",&i);
  switch(Codifica(parola,i))
      {
      case -1:
        printf("Codifica impossibile\n"); break;
      case 0:
        printf("Codifica Sbagliata\n"); break;
      case 1:
        printf("Codifica Corretta\n");
      }
}
```

Soluzione esercizio 3 del 19-2-98

Per questo esercizio è necessario esaminare tutti i bit uno ad uno, ma non è importante in quale ordine vengono esaminati. Decidiamo quindi di partire dal meno significativo che può facilmente essere estratto calcolando in numero di ingresso "modulo" 2.

La funzione si compone di un ciclo in cui ad ogni iterazione si estrae il bit meno significativo (nel modo descritto) e si trasla il valore di ingresso verso destra.

Quando il bit estratto è uguale a `0` vuol dire che si è interrotta una sequenza (eventualmente nulla) di `1`, e quindi è necessario verificare se la sequenza sotto esame è la più lunga incontrata fino a quel momento.

Quando invece il bit estratto è uguale a `1` is incrementa la lunghezza della sequenza corrente.

```
#include <stdio.h>

char conta(unsigned int n)
{
  int bit, i, max, corr;
  max = corr = 0;
  for (i = 0; i < 16; ++i)
    {
      bit = n % 2;
      n = n >> 1;
      if (bit == 1)
        ++corr;
      else
        {
          if (corr > max)
            max = corr;
          corr = 0;
        }
    }
  if(corr > max)
    max = corr;
  return max;
}

main()
{/* programma principale di prova (non richiesto) */
```

```
  printf("%d\n",conta(0xFFFF));
  printf("%d\n",conta(0xF000));
  printf("%d\n",conta(0x0000));
  printf("%d\n",conta(0x0001));
  printf("%d\n",conta(0x8000));
  printf("%d\n",conta(0xF1F0));
}
```

Soluzione esercizio 3 del 16-4-98

Sfruttando l'operatore '^' di XOR (OR esclusivo), questo esercizio viene risolto da una funzione di una sola riga. Infatti è sufficiente porre in XOR il carattere in ingresso con un "1" traslato di n posizioni a sinistra.

```
#include <stdio.h>

unsigned char ComplementaBit(unsigned char ch, int n)
{
  return ch ^ (1 << n);
}

/* funzione ausiliaria (non richiesta) */
void StampaBit(unsigned char ch)
{ int i;
  unsigned int maschera = 1 << 7;
  for (i = 0; i < 8; i++)
    {
      if ((ch & maschera) == 0)
        putchar('0');
      else
        putchar('1');
      ch = ch << 1;
    }
}
main()
{ /* programma principale (non richiesto) */
 unsigned char ch;
 int n;
 printf("Inserisci un carattere senza segno : ");
 scanf("%c", &ch);
 printf("Quale bit vuoi complementare : ");
 scanf("%d", &n);
 printf("Carattere in ingresso :\t");
 StampaBit(ch);
 printf("\n");
 printf("Carattere in uscita :\t");
 StampaBit(ComplementaBit(ch,n));
 printf("\n");
}
```

Capitolo 7

Esercizi di teoria

Si riportano, di seguito, alcuni tra i più significativi esercizi di teoria dati nelle ultime prove di esame e di esonero. Si noti che negli esercizi X rappresenta la cifra meno significativa non nulla del proprio numero di matricola.

7.1 Sistemi di numerazione binari

1) Si consideri una rappresentazione binaria in virgola mobile a 24 bit, di cui (*nell'ordine da sinistra a destra*) 1 per il segno (0=positivo), 7 per l'esponente, che è rappresentato in eccesso 64, e 16 per la parte frazionaria della mantissa. In corrispondenza a tutti valori dell'esponente diversi da 0000000 la mantissa è normalizzata tra 1 e 2 ($1 \leq m < 2$). Con l'esponente 0000000 si rappresentano invece numeri denormalizzati, con esponente uguale a -63 e mantissa normalizzata tra 0 e 1 ($0 \leq m < 1$).

- **a)** calcolare il massimo e il minimo numero positivo rappresentabile (normalizzato e denormalizzato), specificando anche i rispettivi numerali nella notazione suddetta;
- **b)** calcolare l'ordine di grandezza in termini di potenze di 10 dei numeri calcolati al punto a);
- **c)** calcolare il valore arrotondato alla più vicina potenza di 2 del numero rappresentato nella notazione suddetta dai 24 bit espressi in esadecimale da 650XF3.

2) Si consideri una rappresentazione binaria in virgola mobile a 16 bit, di cui (*nell'ordine da sinistra a destra*) 1 bit per il segno (0=positivo), e per l'esponente, che è rappresentato in complemento a 2, ed i rimanenti per la parte frazionaria della mantissa che è normalizzata tra 1 e 2 ($1 \leq m < 2$):

- **a)** determinare il valore minimo di e che consenta di rappresentare nella notazione data il numero n rappresentato in notazione eccesso 2^{31} dalla stringa esadecimale 8XFAX34B;
- **b)** determinare l'errore di troncamento assoluto e relativo che si commette rappresentando n nella notazione in virgola mobile suddetta, esprimendolo in termini della più vicina potenza di 2;
- **c)** determinare il numerale corrispondente, nella notazione data, al più piccolo numero positivo rappresentabile.

3) Si consideri una rappresentazione binaria in virgola mobile a 24 bit, di cui (*nell'ordine da sinistra a destra*) 1 bit per il segno (0=positivo), 7 per l'esponente, che è rappresentato in eccesso 64, ed i rimanenti per la parte frazionaria della mantissa che è normalizzata tra 1 e 2 ($1 \leq m < 2$):

- **a)** rappresentare nella notazione suddetta la somma dei numeri corrispondenti in tale notazione alle stringhe esadecimali 3X8X4B e 44X545;
- **b)** determinare gli errori di troncamento assoluto e relativo che si commettono rappresentando il risultato della somma in questione, esprimendoli in termini della più vicina potenza di 2;
- **c)** determinare il numerale corrispondente, nella notazione data, al più piccolo numero positivo rappresentabile, indicandone l'ordine di grandezza come potenza di 10.

4) Si considerino due notazione binarie in virgola mobile a 16 bit, entrambe con (*nell'ordine da sinistra a destra*) 1 bit per il segno (0=positivo), e bit per l'esponente, rappresentato in complemento a 2, ed i rimanenti m bit per la parte frazionaria della mantissa che è normalizzata tra 1 e 2. Nella prima notazione $e = 4$ ed $m = 11$, nella seconda $e = 11$ ed $m = 4$.

- **a)** dati i numeri r ed s rappresentati in complemento a 2 rispettivamente dalle stringhe esadecimali F0X e 6X3, scegliere per ciascuno la notazione più adatta a rappresentarlo fra le due proposte e calcolare i rispettivi numerali;
- **b)** specificare se si commette o meno un errore nell'utilizzare per r ed s le rappresentazioni scelte al punto **a)**, e, in caso positivo, valutare l'errore relativo ed assoluto.

5) Si considerino due notazione binarie in virgola mobile a 16 bit, entrambe con (*nell'ordine da sinistra a destra*) 1 bit per il segno (0=positivo), e bit per l'esponente, rappresentato in eccesso 2^{e-1} , ed i rimanenti m bit per la parte frazionaria della mantissa che è normalizzata tra 1 e 2. Nella prima notazione $e = 4$ ed $m = 11$, nella seconda $e = 8$ ed $m = 7$.

- **a)** dato il numero n rappresentato nella prima notazione dalla stringha 3XD7, rappresentarlo nella seconda notazione;
- **b)** calcolare l'errore relativo ed assoluto che si commette nel passaggio di notazione;
- **c)** dato il numero n rappresentato in complemento a 2 dalla stringa BXF742, definire una terza notazione, analoga alle precedenti ma con valore di e tale da rappresentare n col minimo errore relativo;
- **d)** calcolare l'ordine di grandezza decimale dell'errore relativo di cui al punto c).

6) Si considerino il numero intero n rappresentato in notazione binaria naturale dalla stringa esadecimale AX, ed il numero frazionario $f = n + n * 2^{-8}$. Data una rappresentazione binaria in virgola mobile a 16 bit di cui (da sinistra a destra) 1 per il segno (0=positivo), m dedicati alla parte frazionaria della mantissa normalizzata tra 1 e 2, e i rimanenti e all'esponente rappresentato in eccesso 2^{e-1}:

- **a)** determinare il valore e_{min} di e che permette di rappresentare il numero f con la massima precisione possibile nella notazione suddetta;
- **b)** determinare il numerale che rappresenta f nella notazione suddetta (con e_{min} bit per l'esponente);
- **c)** calcolare l'errore relativo ed assoluto commessi rappresentando f nella notazione suddetta.

7) Si consideri una rappresentazione binaria in virgola mobile a 24 bit di cui (da sinistra a destra) 1 per il segno (1=positivo), n dedicati alla mantissa e i rimanenti e all'esponente, con esponente in eccesso 2^{e-1}, e mantissa $1 \leq m < 2$ normalizzata di cui viene rappresentata solo la parte frazionaria. Tramite tale notazione si vogliono rappresentare numeri reali compresi tra 10^{-99} e 10^{+99} con la maggiore precisione possibile:

- **a)** calcolare il numero di bit e_{min} da dedicare all'esponente, e l'effettivo intervallo di numeri rappresentati (in termini di potenze di 2);
- **b)** rappresentare nella notazione sopra definita il numero k rappresentato in complemento a 2 dalla stringa esadecimale $9XF4AX$;
- **c)** indicare l'errore relativo ed assoluto commessi rappresentando k nella notazione suddetta.

8) Si consideri una rappresentazione binaria in virgola mobile a 16 bit, di cui (*nell'ordine da sinistra a destra*) 1 per il segno (1=negativo), 5 per l'esponente, che è rappresentato in eccesso 16, e 10 per la parte frazionaria della mantissa. In corrispondenza a tutti valori dell'esponente diversi da 00000 la mantissa è normalizzata tra 1 e 2 ($1 \leq m < 2$). Con l'esponente 00000 si rappresentano invece numeri denormalizzati, con esponente convenzionalmente uguale a -15 e mantissa compresa tra 0 e 1 ($0 < m < 1$):

- **a)** calcolare il massimo e il minimo numero positivo rappresentabili, sia normalizzati che denormalizzati, specificando anche i rispettivi numerali nella notazione suddetta;
- **b)** calcolare l'ordine di grandezza in termini di potenze di 10 della diffrenza fra il mimimo positivo normalizzato e il massimo positivo denormalizzato;
- **c)** calcolare la potenza di 2 che approssima per eccesso il numero n rappresentato nella notazione suddetta dai 16 bit espressi in esadecimale da 80XF;
- **d)** rappresentare in complemento a due col numero minimo di bit il numero $n \cdot 2^{32}$ dove n è il numero di cui al punto precedente.

7.2 Memorie Cache

9) Una cache a mappa diretta con 16k slot e blocchi di 64 byte, è istallata in un sistema con indirizzi a 32 bit:

- **a)** specificare la struttura di ciascuna slot, indicando esplicitamente la dimensione complessiva della slot e quella di ciascun campo;
- **b)** calcolare il numero di slot e la posizione nella slot corrispondenti al byte di indirizzo esadecimale 7BA3FF7D;
- **c)** supponendo che ogni blocco una volta entrato in cache viene in media acceduto 8.6 volte dare una stima della *cache hit ratio*.

10) Si consideri una cache a mappa diretta con 32k slot, con blocchi di 64 byte. Assumendo di lavorare con indirizzi a 32 bit:

- **a)** descrivere la struttura della slot, e la dimensione in bit delle sue componenti;
- **b)** specificare gli indirizzi di due qualsiasi blocchi che condividono l'ultima slot della cache;
- **c)** verificare se i due byte di indirizzo esadecimale X2X5379X e 3F5537BC collidono sulla stessa slot o no;

11) Si consideri una memoria cache associativa ad insiemi composta da 4k slot di 4 elementi ciascuno in un sistema con indirizzi a 24 bit e blocchi di memoria da 16 byte. Indicare:

a) la struttura dell'indirizzo, specificando la dimensione dei vari campi in bit;

b) la struttura di uno slot, specificando la dimensione dei vari campi in bit;

c) i passi necessari alla ricerca nella cache del byte di indirizzo BXAXF2.

12) Si consideri un'architettura con indirizzi a 20 bit e blocchi di memoria di 4 byte, in cui è presente una cache associativa con 1K slot. Il contenuto (escluso il bit valid) della slot di indirizzo decimale 2X3 è dato dalla stringa esadecimale AX5FXB1XA1:

a) specificare la struttura degli indirizzi nel sistema sopra descritto;

b) determinare l'indirizzo di memoria del byte di indirizzo più alto del blocco contenuto nella slot in questione, fornendo la risposta sotto forma di stringa esadecimale;

c) specificare l'indirizzo esadecimale di un qualsiasi byte appartenente ad un blocco cui corrisponde l'ultima slot della cache.

13) Si consideri una cache a mappa diretta con 256k slot, con blocchi di 32 byte. Assumendo di lavorare con indirizzi a 32 bit, specificare:

a) la struttura della slot, e la dimensione delle sue varie componenti;

b) la struttura dell'indirizzo ed il meccanismo che associa ad un byte una slot e una posizione nella slot;

c) supponendo, a parità degli altri parametri, di voler limitare l'indicatore (*tag*) a soli 7 bit, come varia la dimensione della parte di spazio indirizzabile coperta dalla cache.

14) Si consideri un'architettura con indirizzi a 20 bit e blocchi di memoria di 4 byte, in cui è presente una cache associativa con 1K slot. Il contenuto (escluso il bit valid) della slot di indirizzo decimale 2X3 è dato dalla stringa esadecimale AX5FXB1XA1:

a) specificare la struttura degli indirizzi nel sistema sopra descritto;

b) determinare l'indirizzo di memoria del byte di indirizzo più alto del blocco contenuto nella slot in questione, fornendo la risposta sotto forma di stringa esadecimale;

c) specificare l'indirizzo esadecimale di un qualsiasi byte appartenente ad un blocco cui corrisponde l'ultima slot della cache.

7.3 Formato delle istruzioni e degli indirizzi

15) Supponendo che in un architettura con istruzioni a 16 bit siano necessari 4 bit per indirizzare ciascun operando, e che si voglia utilizzare la tecnica dell'espansione dei codici operativi per avere 6 istruzioni a tre indirizzi e 48 istruzioni a due indirizzi:

a) quante istruzioni ad un indirizzo è possibile avere al massimo?

b) mostrare la corrispondente organizzazione dei codici operativi.

16) In un architettura con istruzioni a 16 bit sono destinati 4 bit per indirizzare ciascun operando. Si supponga di utilizzare la tecnica dell'espansione dei codici operativi e di voler avere 9 istruzioni a tre indirizzi, 18 istruzioni ad un indirizzo e 7 istruzioni a zero indirizzi:

a) quante istruzioni a due indirizzi è possibile avere al massimo?

b) mostrare la corrispondente organizzazione dei codici operativi.

17) Si supponga di progettare architettura con istruzioni a 16 bit, in cui occorrono i 4 bit per indirizzare ciascun operando, e in cui si vuole utilizzare la tecnica dell'espansione dei codici operativi:

- **a)** volendo avere 18 istruzioni a due indirizzi, 8 ad un indirizzo e nessuna a zero indirizzi, quante istruzioni a tre indirizzi è possibile avere al massimo?
- **b)** mostrare la corrispondente organizzazione dei codici operativi;
- **c)** cosa cambia se si vogliono avere 80 istruzioni a un indirizzo invece di 8?

18) In un sistema a memoria virtuale con tavola delle pagine a più livelli, 8 dei 32 bit dell'indirizzo non sono utilizzati e si usa una tavola delle pagine a tre livelli, utilizzando gruppi di 4,3,5 bit. Specificare:

- **a)** la dimensione delle pagine, e il numero complessivo di pagine nello spazio di indirizzamento virtuale;
- **b)** quante tavole e di quanti elementi ci sono a ciascun livello;
- **c)** dato l'indirizzo esadecimele 90FX07AX, specificare il numero della tavola e dell'elemento nella tavola acceduti a ciascun livello durante il processo di traduzione e l'offset nella pagina.

7.4 File Systems

19) Illustrare *concisamente* la modalità di accesso a file tramite indice hash. Si consideri un file di 120.000 record di 163 byte ciascuno, con un campo chiave di 23 byte, sul quale è stato costruito un indice, usando una funzione hash con un codominio di cardinalità 50. Supponendo di disporre di blocchi di 4K byte, specificare:

- **a)** La dimensione (in blocchi) del file dati.
- **b)** Il numero medio di accessi a disco necessari per accedere ad un record del file.

20) Illustrare *concisamente* la modalità di accesso a file tramite indici ISAM. In particolare dato un file di 100.000 record di 240 byte ciascuno, con chiavi di 20 byte, un sistema con blocchi su disco di 2k byte e indirizzi su disco di 4 byte?:

- **a)** quanti blocchi occupa il file dati;
- **b)** quanti blocchi occupa il file indice;
- **c)** il numero massimo e minimo di accessi a disco necessari per accedere ad un record, effettuando una scansione sequenziale dell'indice;
- **d)** quanti accessi si risparmierebbero effettuando una ricerca binaria sull'indice?

21) Si consideri un file di 12.000 record di 350 byte ciascuno, con un campo chiave di 78 byte, sul quale è stato costruito un indice ISAM. Supponendo di *non frazionare mai un record su due blocchi*, e che i blocchi su disco siano di 1Kbyte e gli indirizzi di 4 byte, calcolare:

- **a)** la dimensione (in blocchi) del file dati e del file indice;
- **b)** il numero medio minimo e massimo di accessi a disco necessario per accedere ad un record di chiave data;
- **c)** Come dovrebbe essere costruito un file hash in modo da avere lo stesso numero medio di accessi.

22) Illustrare *concisamente* la modilatà di accesso ai file tramite indice ISAM. Dato un file costituito da 9500 record di 157 byte di cui di 16 byte di chiave, e un sistema con blocchi su disco di 512 byte ed indirizzi su disco di 4 byte, calcolare:

a) il numero di accessi a disco nel caso peggiore *senza* utilizzare alcun indice, necessari per accedere ad un record di chiave data;

b) la dimensione in record e blocchi del file indice;

c) se l'accesso al file venisse fosse gestito utilizzando una funzione hash con codominio di cardinalità 30, il costo di accesso migliorerebbe o peggiorerebbe rispetto al caso dell'indice ISAM, e perchè?

23) Si consideri in un sitema Unix un disco con blocchi di 2K byte, e indirizzi su disco di 4 byte, e un file di 270.000 record di 550 byte:

a) quanti blocchi sono necessari per rappresentare il file senza mai frazionare un record su due blocchi diversi;

b) quanti liveli di indirezione nell'i-node sono necessari per rappresentare il file in questione;

c) calcolare l'ordine di grandezza approssimativo del piu' grande file rappresentabile nel sitema dato;

24) Illustrare *concisamente* la struttura di un i-node. In particolare dato un file di 120.000 record di 220 byte ciascuno mai frammentati nei blocchi, con chiavi di 10 byte, un sistema con blocchi su disco di 1k byte e indirizzi su disco di 4 byte?:

a) quanti blocchi occupa il file dati;

b) il livello di indirezione utilizzato nell' i-node;

c) il numero massimo di record che un file può contenere?

7.5 Architetture parallele

25) Si consideri un'architettura ad ipercubo a 4 dimensioni. Specificare:

a) il numero di nodi, il numero totale di collegamenti, la distanza massima fra due nodi;

b) quanti e quali percorsi esistono tra i nodi 1111 e 0100

26) Considerare un'architettura ad ipercubo di dimensione 6. Specificare:

a) quanti cammini di lunghezza minima ci sono tra i due nodi 100110 e 011101;

b) confrontare il numero di connessioni (bidirezionali) dell'ipercubo dato e di una griglia con lo stesso numero di processori.

c) quanti nodi distinti sono raggiungibili da un nodo dato con cammini di lunghezza non superiore a 4?

27) Si consideri un'architettura ad ipercubo con 256 processori:

a) specificare quanti e quali cammini di lunghezza minima ci sono tra i due nodi 01100110 e 01010010;

b) calcolare il numero dei nodi che *non* sono raggiungibili dal nodo 01101101 con cammini di lunghezza 6;

c) confrontare il numero totale di connessioni (bidirezionali) dell'ipercubo dato e di una griglia con lo stesso numero di processori;

d) calcolare la massima distanza tra due nodi nella griglia di cui al punto precedente.

28) Si consideri un ipercubo con $n = 64$ nodi:

a) specificare *quanti* e *quali* nodi si trovano a distanza 5 dal nodo 011010;

b) calcolare quanti cammini distinti di lunghezza minima esistono tra due qualsiasi dei nodi di cui al punto **a)**;

c) calcolare quanti collegamenti (bidirezionali) si risparmiano *complessivamente* interconnettendo gli stessi n nodi con una struttura a griglia;

d) quali vantaggi presenta la struttura ad ipercubo rispetto alla griglia?

Capitolo 8

Domande a risposta multipla

Si riportano, nel seguito, alcune tra le più significative tra le domande a risposta multipla dati nelle ultime prove di esame e di esonero. Si noti che il doppio asterisco indica le risposte esatte.

RM1) Si consideri un ipercubo di dimensione n. Indicare le affermazioni esatte tra le seguenti:

A) Se due nodi differiscono tra loro per k bit la loro distanza minima è $k+1$;

**[B)] Se le etichette di due nodi differiscono tra loro per k bit la loro distanza è k;

C) Tra ogni coppia di nodi esiste sempre uno ed un solo cammino;

D) Se le etichette di due nodi differiscono tra loro per k bit la loro distanza minima è $k!$;

E) La distanza minima tra due nodi è $\lceil n/2 \rceil$;

**[F)] Da ogni nodo partono n collegamenti;

**[G)] La distanza massima tra due nodi è n;

H) Ogni nodo vede $d!$ nodi a distanza d;

RM2) Considerando la microarchitettura vista a lezione, indicare le affermazioni esatte tra le seguenti:

A) Il programmatore assembler può utilizzare direttamente le microistruzioni;

B) Ad ogni istruzione della macchina standard corrisponde una ed una sola microistruzione;

**[C)] Utilizzando la microprogrammazione verticale lo spazio necessario in ROM per memorizzare il microcodice diminuisce;

**[D)] L'insieme delle microistruzioni è cablato in una ROM interna al microprocessore;

**[E)] Una microistruzione viene eseguita in un ciclo di clock;

F) Ad ogni microistruzione corrisponde una ed una sola istruzione della macchina standard;

G) Durante l'esecuzione di una istruzione della macchina standard il contenuto del registro program counter viene copiato nel registro microprogram counter;

RM3) Facendo riferimento alle architetture RISC viste a lezione, indicare le affermazioni esatte tra le seguenti:

**[A)] A volte i compilatori RISC devono inserire nel codice delle istruzioni nulle (NOP)

B) L'uso dei registri nelle macchine RISC, a parte la maggiore velocità di esecuzione delle istruzioni è analogo a quello delle macchine CISC;

C) Nelle architetture RISC l'utilizzo dello stack è più intenso;

D) Ogni istruzione RISC è eseguita in un ciclo di clock;

**[E)] I compilatori per microprocessori RISC sono più complessi;

**[F)] L'occupazione di memoria (disco e RAM) dei programmi RISC è maggiore di quella richiesta da analoghi programmi CISC;

**[G)] In un processore RISC il microcodice è assente;

RM4) Facendo riferimento ai bus sincroni ed asincroni ed ai relativi problemi di arbitraggio, indicare le affermazioni corrette tra le seguenti:

**[A)] In un bus sincrono la temporizzazione prevede dei tempi minimi tra due segnali per dar tempo ai dati di stabilizzarsi sul bus.

**[B)] In un bus asincrono è necessario utilizzare un protocollo di comunicazione più complesso(full handshake).

C) Nel progetto del protocollo di un bus asincrono i problemi relativi alla propagazione dei segnali (bus skew) sono trascurabili.

**[D)] In un bus sincrono una operazione di accesso a memoria occupa un periodo temporale che è un multiplo intero del ciclo di clock.

E) In un arbitraggio centralizzato a due livelli è garantito un tempo massimo di attesa per l'accesso al bus da parte di qualunque dispositivo.

**[F)] In un arbitraggio centralizzato ad un livello la disposizione fisica dei componenti ha un effetto determinante sulla priorità degli stessi.

G) L'inizio di un ciclo in un bus asincrono può avenire solo in istanti predeterminati.

**[H)] A parità di altre condizioni un bus asincrono è più veloce.

RM5) Facendo riferimento alla struttura delle memorie cache (mappa diretta, associativa ed associativa ad insiemi), indicare le affermazioni corrette tra le seguenti:

**[A)] Nelle cache associative pure il problema della collisione sistematica di due blocchi sullo stesso slot è assente.

B) Una cache a mappa diretta è più veloce di una associativa.

C) In una cache a mappa diretta il numero di blocchi che condivide lo stsso slot dipende solo dal numero di slot e dalla dimensione dello spazio di indirizzamento.

D) In una cache a mappa diretta a k slot le collisioni si possono verificare solo se $(k/2) + 1$ slot sono occupati.

**[E)] A parità di slot le cache associative hanno un costo più alto di quelle a mappa diretta.

F) A parità di numero i blocchi memorizzabili una cache associativa ad insiemi in cui la cardinalità dell'insieme è 2 non riduce il numero di collisioni rispetto ad una cache a mappa diretta.

**[G)] In una cache associativa ad insiemi la ricerca nei blocchi relativi ad un insieme avviene in parallelo.

**[H)] Se in una cache a mappa diretta il campo indicatore (tag) ha 11 bit il numero di blocchi che condivide uno slot è 2^{11}.

RM6) Facendo riferimento alla gestione della memoria virtuale del 386, indicare le affermazioni corrette tra le seguenti:

**[A)] Lo spazio di indirizzamento virtuale è più grande di quello di indirizzamento fisico massimo (2^{32}).

B) I segmenti sono sempre tutti presenti in memoria centrale.

C) La dimensione dello spazio di indirizzamento lineare può superare la dimensione massima della memoria fisica.

D) I rgistri segmento contengono i descrittori dei segmenti.

E) I segmenti hanno dimensione fissa.

**[F)] È possibile avere contemporaneamente sia segmentazione che paginazione.

G) Un segmento può essere più grande del massimo spazio di indirizzamento fisico (2^{32}).

**[H)] Si è mantenuta una totale compatibilità con il 286.

RM7) Facendo riferimento ai meccanismi per garantire la coerenza delle memorie cache in un ambiente multiprocessore, indicare le affermazioni corrette tra le seguenti:

A) Usando la politica write once e le snooping cache ogni volta che un micropocessore modifica un blocco in cache lo aggiorna anche in memoria.

**[B)] La politica write once prevede quattro stati distinti per ogni blocco in cache.

C) L'unico modo per evitare inconsistenze è quello di utilizzare delle snooping cache.

D) Usando la politica write once e le snooping cache ad ogni istante i blocchi validi nelle memorie cache sono sempre allineati con quelli in memoria centrale.

**[E)] Usando la politica write once e le snooping cache statisticamente il traffico sul bus diminuisce.

F) La politica write once prevede quattro stati distinti per ogni blocco in memoria centrale.

**[G)] Usando la politica write through e le snooping cache ad ogni istante i blocchi validi nelle memorie cache sono sempre allineati con quelli in memoria centrale.

H) Usando la politica write through e le snooping cache, una cache che vede scrivere in memoria un blocco lo annulla solo se la sua copia è differente da qulla appena scritta.

RM8) Facendo riferimento all'utilizzo dello stack per la chiamata di procedure (A chiama B) indicare le affermazioni corrette tra le seguenti:

A) il puntatore allo stack viene incrementato esclusivamente dalla procedura chiamata (B);

B) senza il puntatore LB è impossibile accedere ai parametri della procedura chiamata (B);

**[C)] se lo stack cresce verso l'alto (per indirizzi decrescenti) all'interno dello stack frame di B lo SP ha un valore più piccolo del LB;

**[D)] il salvataggio dell'indirizzo di ritorno di A avviene contemporaneamente alla chiamata della procedura B;

E) la procedura A salva sullo stack il vecchio valore di LB;

F) la procedura B spinge sullo stack i parametri necessari al suo funzionamento;

G) nelle operazioni di push e pop occorre indicare in modo esplicito il valore di SP;

H) se lo stack cresce verso l'alto (per indirizzi decrescenti) all'interno dello uno stack frame di B gli indirizzi in cui sono memorizzati i parametri sono più piccoli di quello relativo all'indirizzo di ritorno;

**[I)] la procedura B salva sullo stack il contenuto dei registri che intende utilizzare per il suo funzionamento.

RM9) Si consideri la seguente sequenza di caratteri digitati da tastiera:`casa23 55 \n32\npippo\n` per il seguente programma c:

```
#include<stdio.h> int main(){
   int i;
   char a, b,s1[20], s2[20];
   scanf("%c%s%d%c%*c%s",&a,s1,&i,&b,s2);
   printf("%s%c%d%",s1,b,i);
   }
```

Indicare le affermazioni corrette tra le seguenti:

A) la stringa di ingresso viene completamente consumata;

B) il programma non termina;

C) il programma termina in errore;

D) i vale 23;

**[E)] i vale 55;

F) i vale 32;

G) s1 vale "casa";

**[H)] s1 vale "asa23";

I) il programma stampa sulla prima riga `casa` e sulla seconda `23`;

L) il programma stampa sulla prima riga `asa23` e sulla seconda `55`.

RM10) Facendo riferimento al seguente programma C (in cui sono state omesse per brevità le #include...):

```
int main(){
(1) int v[2], *p, **k;
(2) v[0]=55;
(3) v[1]=56;
(4)p=v;
(5) k=&p;
(6) printf("%d%d",k[0][0],k[0][1]);
}
```

Indicare le affermazioni corrette tra le seguenti:

A) il programma contiene errori sintattici;

B) prima di usare p e k occorre allocare memoria;

**[C)] il programma stampa 5556;

**[D)] k[0] è un puntatore ad interi;

E) l'istruzione (5) fa perdere della memoria;

F) v[1] è un puntatore ad interi;

G) k è un vettore di due interi;

H) l'istruzione (5) può essere sostituita da $k = p[0]$;

**[I)] l'istruzione (4) può essere sostituita da $p = \&v[0]$.

RM11) Assumendo che il contenuto del file *dati.txt* sia:

```
prova di esame del    15/2/99.
```

e considerando il seguente programma C indicare le affermazioni esatte tra le seguenti:

```
#include<stdio.h> int main(){ FILE *p; int i; char s[15];
p=fopen("dati.txt","r");
i=fscanf(p,"%c%s%c%s%s%s",s,s,s,s,s,s);
fclose(p);}
```

**[A)] La fscanf, in corrispondenza delle conversioni %s, aggiunge il terminatore di stringa.

B) s[14] contiene il terminatore di stringa.

**[C)] i vale 6.

D) s contiene *esame*.

E) L'uso di s nella scanf è errato.

F) Il file viene totalmente letto.

**[G)] s contiene *del*.

H) Il programma non termina.

I) Nella fscanf occorre aggiungere & alla prima e terza occorrenza di s.