POLITECNICO
DI TORINO
Dipartimento
di Automatica e Informatica

Capitolo 3: Problem-solving su dati scalari

DAL PROBLEMA AL PROGRAMMA: INTRODUZIONE AL PROBLEM-SOLVING IN LINGUAGGIO C

Problemi trattati

- numerici
- di codifica
- di text-processing
- di verifica, filtro e ordinamento

senza uso di vettori/matrici (solo dati scalari)

con costrutti condizionali (problemi più semplici) e/o iterativi.

I dati scalari

- I dati scalari includono:
  - numeri
  - caratteri/stringhe
  - aggregati eterogenei (`struct`)
- Sono esclusi vettori e matrici, come collezioni di dati numerabili (e individuati da indici)
- Sono incluse:
  - le stringhe, viste come dati unitari (parole/frasi) e non come vettori di caratteri
  - le `struct`, in quanto aggregati non numerabili

- Qualora la soluzione sia iterativa, è possibile elaborare l’i-esimo dato senza ricordare tutti i precedenti.
- Esempio: possono essere necessari ultimo e penultimo dato, ma non serve un vettore contenente tutti i dati.

Problemi numerici

- Problemi (in generale di ricerca) di algebra, geometria, statistica, ecc., caratterizzati, in genere da:
  - dati numerici (reali o interi)
  - valutazione di espressioni (formule) matematiche o sequenze di calcoli iterati
- Vantaggio: facilità di codifica
  - i problemi matematici sono spesso già espressi in formati rigorosi e non ambigui

- Attenzione: rispetto alla formulazione matematica, occorre tener conto di:
  - rappresentazione finita dei numeri (overflow/underflow)
  - problemi di arrotondamento/troncamento e conversioni intero-reale.

Problemi numerici non iterativi

- Problemi caratterizzati da selezione in base a sottoproblemi diversi, nei quali si applicano formule diverse
- Di solito i singoli casi vengono selezionati mediante costrutti if (eventualmente annidati). E' raro l'utilizzo del costrutto switch
- Pur se sono possibili diversi schemi di organizzazione dei costrutti condizionali, la soluzione (struttura dati e algoritmo) è semplice
- È talvolta possibile che si debbano affrontare conversioni di tipo o arrotondamenti/troncamenti.

Equazione di II grado

- Formulazione:
  - dati i tre coefficienti (a, b, c) di un'equazione
  - $ax^2 + bx + c = 0$
  - determinare le soluzioni dell'equazione, distinguendo i casi di equazione impossibile, indeterminata, di primo grado, di secondo grado con soluzioni reali distinte, reali coincidenti o complesse coniugate
- Soluzione: si tratta di un tipico esempio di selezione (mediante costrutti condizionali) tra più casi, ad ognuno dei quali corrispondono formule e messaggi in output diversi

- Struttura dati: variabili scalari, di tipo float (in alternativa double), per:
  - coefficienti: a, b, c
  - determinante: delta
  - soluzioni: x0, x1 per soluzioni reali, re, im per complesse
- Algoritmo: selezione tra 5 casi mediante schemi di if … else non annidati o annidati per distinguere impossibile, indeterminata, I grado, II grado.
- NB: non è consigliabile il costrutto switch in quanto la selezione andrebbe fatta su valori interi.

Codice versione 1: `if` non annidati

```c
#include <math.h>
#include <stdio.h>
int main(void) {
  float a,b,c,delta,x0,x1,re,im;
  printf("Coefficienti (a b c): ");
  scanf("%f%f%f",&a,&b,&c);
  if (a==0 && b==0 && c==0)
    printf("Equazione indeterminata\n");
  if (a==0 && b==0 && c!=0)
    printf("Equazione impossibile\n");
  if (a==0 && b!= 0) {
    printf("Equazione di I grado\n »);
    printf("Soluzione: %f\n", -c/b);
  }
  if (a!=0) {
    delta = b*b-4*a*c;
    if (delta==0) {
      x0 = (-b)/(2*a);
      x1 = (-b)/(2*a);
      printf("2 sol. reali coincidenti: ");
      printf("%f %f\n",x0,x1);
    }
    if (delta > 0) {
      x0 = (-b-sqrt(delta))/(2*a);
      x1 = (-b+sqrt(delta))/(2*a);
      printf("2 sol. reali distinte: ");
      printf("%f %f\n",x0,x1);
    }
    if (delta < 0){
      re = -b/(2*a);
      im = sqrt(-delta)/(2*a);
      printf("2 sol. compl. coniug.: ");
      printf("x0=%f-i*%f ",re, im);
      printf("x1=%f+i*%f\n", re, im);
    }
  }
}
```

Codice versione 2: `if` annidati

```c
#include <math.h>
#include <stdio.h>
int main(void) {
  float a,b,c,delta,x0,x1,re,im;
  printf("Coefficienti (a b c): ");
  scanf("%f%f%f",&a,&b,&c);
  if (a==0) {
    if (b==0) {
      if (c==0)
        printf("Equazione indeterminata\n");
      else
        printf("Equazione impossibile\n");
    }
    else {
      printf("Equazione di I grado\n");
      printf("Soluzione: %f\n", -c/b);
    }
  }
  else {
    delta = b*b-4*a*c;
    if (delta == 0) {
      x0 = (-b)/(2*a);
      x1 = (-b)/(2*a);
      printf("2 sol. reali coinc.: ");
      printf("%f %f\n",x0,x1);
    }
    ...
  }
```

Codice versione 2: `if` annidati

```
#include <math.h>
#include <stdio.h>
int main(void) {
  float a,b,c,delta,x0,x1,re,im;
  printf("Coefficienti (a b c): ");
  scanf("%f%f%f",&a,&b,&c);
  if (a==0) {
    ...
  }
  else {
    delta = b*b-4*a*c;
    if (delta == 0) {
    }
    else /* if delta != 0 */
      if (delta > 0) {
        x0 = (-b-sqrt(delta))/(2*a);
        x1 = (-b+sqrt(delta))/(2*a);
        printf("reali dist.: %f %f\n",x0,x1);
      }
      else { /* delta < 0 */
        re = -b/(2*a);
        im = sqrt(-delta)/(2*a);
        printf("comp.con.: \n
                x0=%f-i*%f x1=%f+i*%f\n",
                re, im, re, im);
      }
  }
}
```

Area di triangolo rettangolo

- Formulazione:
  - date le lunghezze dei tre lati (a, b, c) di un triangolo rettangolo (la lunghezze sono una terna pitagorica di numeri interi)
  - determinare quale dei tre lati è l'ipotenusa
  - calcolare l'area del triangolo
- Soluzione:
  - determinare l'ipotenusa trovando il lato più lungo
  - calcolare l'area come numero reale

- Struttura dati: variabili scalari per rappresentare:
  - i tre lati: a, b, c (tipo int)
  - l'area: area (tipo float)
- Algoritmo: occorre selezionare tra i 3 casi possibili per l'ipotenusa (a, b oppure c). Alternative:
  - permutare i lati, in modo da raggiungere sempre la stessa configurazione
  - predisporre 3 calcoli distinti di area (scelta proposta).
  - Nel calcolo dell'area è necessaria la divisione tra reali (e non fra interi).

Codice versione 1: 3 `if`, 6 confronti

```c
#include <math.h>
#include <stdio.h>
int main(void) {
  int a,b,c;
  float area;

  printf("Lati del triangolo (a b c): ");
  scanf("%d%d%d",&a,&b,&c);

  if (a>b && a>c) {
    printf("L'ipotenusa e' a\n");
    area = b*c/2.0;
  }
 else if (b>a && b>c) {
    printf("L'ipotenusa e' b\n");
    area = a*c/2.0;
 }
 else if (c>a && c>b) {
    printf("L'ipotenusa e' c\n");
    area = a*b/2.0;
 }
 printf("L'area e': %f\n", area);
}
```

Codice versione 1: 3 `if`, 6 confronti

```
#include <math.h>
#include <stdio.h>
int main(void) {
  int a,b,c;
  float area;

  printf("Lati del triangolo
  scanf("%d%d%d",&a,&b,&c);

  if (a>b && a>c) {
    printf("L'ipotenusa e  a\n");
    area = b*c/2.0;
  }
```

```
  else if (b>a && b>c) {
    area = a*c/2.0;
  }
  else {
    area = a*b/2.0;
  }
  printf("L'area e': %f\n", area);
}
```

La costante float (2.0) garantisce divisione tra float e risultato float.
In alternativa si potrebbe scrivere:

area = (float)(b*c)/2.0;

oppure

area = (float)b*(float)c/2.0;

Codice versione 2: 3 `if`, 3 confronti

```
...

if (a>b) /* L'ipotenusa non e' b */
  if (a>c) {
    printf("L'ipotenusa e' a \n"); area = ((float) (b * c)) / 2.0;
  }
  else {
    printf("L'ipotenusa e’ c \n"); area = ((float) (a * b)) / 2.0;
  }

} else /* L'ipotenusa non e' a */
  if (b>c) {
    printf("L'ipotenusa e' b \n"); area = ((float) (a * c)) / 2.0;
  }
  else {
    printf("L'ipotenusa e' c \n"); area = ((float) (a * b)) / 2.0;
  }

}

...
```

Problemi numerici iterativi

- Problemi di natura simile ai precedenti, con l'aggiunta di calcoli iterativi o applicazione ripetuta di formule
- Esempi:
  - successioni o serie numeriche, ad esempio i numeri di Fibonacci
  - calcoli geometrici con poligoni, sequenze di punti e/o segmenti
  - formulazione iterativa di problemi matematici, ad esempio il fattoriale
  - calcolo di massimi/minimi, sommatorie, medie o statistiche su sequenze di dati.

Ridotta n-esima di serie armonica

- Formulazione:
  - serie armonica: successione dei reciproci dei numeri positivi (1, 1/2, 1/3, …)
  - ridotta n-esima della serie è definita come:

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} = \sum_{i=1}^{n} \frac{1}{i}$$

  - la serie armonica è data da H_n con $n \to \infty$
  - Si scriva un programma che ripeta i passi seguenti:
    - legga da tastiera un numero intero n
    - se $n \leq 0$ termini l'esecuzione, in caso contrario determini e stampi la ridotta H_n
- Soluzione:
  - generare iterativamente i termini della successione 1, 1/2, 1/3, ...
  - calcolare in modo incrementale la sommatoria dei termini generati.

- Struttura dati: variabili scalari per rappresentare:
  - il parametro n e un indice i, contatore di iterazioni (variabili int)
  - la sommatoria H, calcolata iterativamente
- Algoritmo:
  - iterazione esterna per acquisizione (ripetuta) di valori interi n (termine se n<=0)
  - iterazione interna per generazione della successione 1..n, e sommatoria H.

Codice

```c
#include <math.h>
#include <stdio.h>
int main(void) {
  int n, i;
  float HN;
  printf("Num. di termini (<=0=FINE): ");
  scanf("%d",&n);
  while ( n>0 ) {
    /* calcola e stampa H */
    HN = 0.0;
    for (i=1; i<=n ; i++)
      HN = HN + 1.0/((float)i);
    printf("Risultato: %f\n", H);

    printf("Num. di termini (<=0=FINE): ");
    scanf("%d",&n);
  }
}
```

Problemi di codifica/decodifica

- Problemi di ricerca nei quali occorre riconoscere o generare la codifica di informazioni di carattere numerico o non numerico
- Codici numerici: si possono gestire numeri interi o reali, in base 2 o altre basi. I problemi possono essere:
  - conversione tra basi (incluso lettura/scrittura di un numero in una certa base)
  - operazioni in una data base
- Codici non numerici (ad esempio caratteri): si possono gestire codifiche binarie di caratteri:
  - decodifica/riconoscimento di codici interni
  - cambio di codifica (ri-codifica/crittografia): occorre conoscere le regole e/o le tabelle di codifica.

Problemi su valori numerici

- I numeri in C sono codificati:
  - internamente in base 2 (complemento a 2, FP IEEE-754, ...)
  - esternamente (input/output) in decimale, ottale, esadecimale
- Le operazioni aritmetiche sono gestite automaticamente in base 2. Gli unici problemi da tener presenti sono quelli di overflow e/o underflow.

Problemi su valori numerici

- I numeri in C sono codificati:
  - internamente in base 2 (complemento a 2, FP IEEE-754, ...)
  - esternamente (input/output) in decimale, ottale, esadecimale
- Le operazioni aritmetiche sono gestite automaticamente in base 2. Gli unici problemi da tener presenti sono quelli di overflow e/o underflow.

La gestione dei codici interni/esterni è automatica.
Perché affrontare problemi di codifica?

- Per gestire esplicitamente la codifica di numeri può essere necessario se si vogliono:
  - utilizzare codifiche non standard (ad esempio in base 4, 5 o altre)
  - modificare i limiti di rappresentazione
  - “decodificare” esplicitamente i codici numerici
- Per affrontare problemi di:
  - conversione tra basi: una delle codifiche può essere quella interna, utilizzabile anche come passaggio intermedio tra altre codifiche
  - calcolo esplicito di operazioni aritmetiche, lavorando sulle singole cifre.

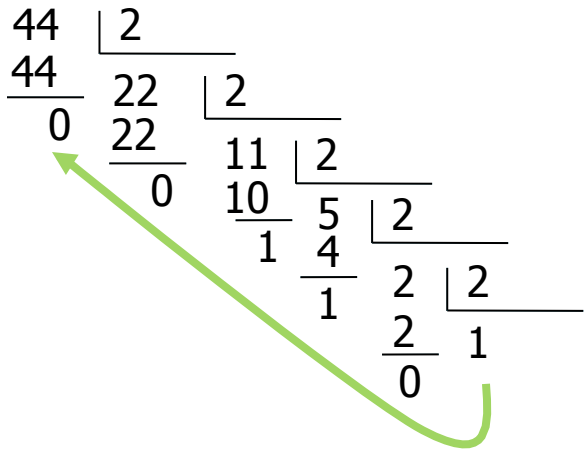
- I problemi di codifica si risolvono spesso mediante algoritmi iterativi. Ora si trattano solo algoritmi che non richiedono vettori (per memorizzare cifre)
- A seconda del formato, può essere necessario gestire cifre numeriche, segno e/o esponente (potenza della base)
- Si possono eventualmente effettuare operazioni aritmetiche utilizzando rappresentazioni delle singole cifre.

Codifica binaria di un intero

- Formulazione:
  - realizzare una funzione C che, ricevuto come parametro un intero (>=0)
  - ne determini la codifica binaria e visualizzi i bit
- Soluzione:
  - l'algoritmo classico di generazione di una codifica binaria procede per divisioni successive per 2. Purtroppo genera i bit a partire dal meno significativo (da destra a sinistra)
  - la generazione dei bit a partire dal più significativo può essere fatta trovando iterativamente la potenza di 2 più grande minore o uguale al numero.

Esempio: convertire 44_{10} in base 2

Divisioni successive:
serve un vettore



$44_{10} = 101100_2$

Algoritmo da SX a DX:
non serve un vettore

32: max potenza di 2 <= 44

32 <= 44	44 – 32 = 12; 32/2 = 16	1
16 > 12	16/2 = 8	0
8 <= 12	12 – 8 = 4; 8/2 = 4	1
4 <= 4	4 – 4 = 0; 4/2 = 2	1
2 > 0	2/2 = 1	0
1 > 0	1/2 = 0	0

- Struttura dati: 2 variabili intere:
  - un parametro formale (intero): n
  - una variabile da utilizzare per generare potenze decrescenti di 2: p
- Algoritmo:
  - prima iterazione per generare in p la più grande potenza di 2 minore o uguale a n
  - seconda iterazione per generare i bit. Ad ogni iterazione
    - se n >= p bit = 1, n = n-p
    - altrimenti bit = 0
    - dimezza p (passa alla potenza di 2 inferiore).

Codice

```
void binario (int n) {
  int p;

  for (p=1; 2*p<=n; p=p*2);
  while (p>0) {
    if (p<=n) {
      printf("1"); n=n-p;
    }
    else printf("0");
    p = p/2;
  }
  printf("\n");
}
```

Codice

```
void binario (int n) {
  int p;

  for (p=1; 2*p<=n; p=p*2);
  while (p>0) {
    if (p<=n) {
      printf("1"); n=n-p;
    }
    else printf("0");
    p = p/2;
  }
  printf("\n");
}
```

Trova max potenza di 2 <= n

Codice

```
void binario (int n) {
  int p;

  for (p=1; 2*p<=n; p=p*2);
  while (p>0) {
    if (p<=n) {
      printf("1"); n=n-p;
    }
    else printf("0");
    p = p/2;
  }
  printf("\n");
}
```

Per p = potenze di 2 decrescenti
- se contenuto in n stampa 1 e sottrai da n

Codice

```
void binario (int n) {
  int p;

  for (p=1; 2*p<=n; p=p*2);
  while (p>0) {
    if (p<=n) {
      printf("1"); n=n-p;
    }
    else printf("0");
    p = p/2;
  }
  printf("\n");
}
```

Per p = potenze di 2 decrescenti
- se non contenuto in n stampa 0

Conversione tra basi

- Formulazione:
  - acquisire da tastiera due numeri b0 e b1 (compresi tra 2 e 9) da considerare quali base iniziale e finale
  - acquisire iterativamente numeri interi (senza segno) nella base b0 (massima cifra ammessa b0-1, gli interi sono separati da spazi o a-capo)
  - ogni intero acquisito va convertito dalla base b0 alla base b1 e stampato a video
  - l'acquisizione/conversione termina inserendo una cifra non valida.
- Soluzione:
  - acquisire da tastiera b0 e b1
  - iterare per acquisire e controllare le cifre
  - convertire passando attraverso la rappresentazione interna standard (binaria).

- Struttura dati: variabili intere:
  - due variabili per le basi b0 e b1
  - una variabile per il numero (n), convertito direttamente nella base b0, a partire dalle cifre
  - una variabile p per le potenze decrescenti della base b1 (come nella conversione a binario)
  - una variabile fine usata come flag del ciclo di acquisizione delle sequenze.

- Algoritmo:
  - acquisizione da tastiera delle basi b0 e b1 e inizializzazione n=0
  - iterazione di acquisizione di caratteri (uno alla volta). Per ogni carattere:
    - se è spazio o a-capo, convertire numero appena acquisito, nella base b1, mediante funzione converti
    - se è una cifra nella base b0, conversione in numero e aggiornamento di n (moltiplica per b0 e somma nuova cifra)
    - se non è una cifra corretta, termina
  - la funzione converti è simile alla precedente (binario).

Codice

```c
#include <stdio.h>
void converti(int n, int b);

int main(void) {
  int b0, b1, n, p, cifra, fine=0;
  char c;

  printf("b0 (2..9): "); scanf("%d",&b0);
  printf("b1 (2..9): "); scanf("%d\n",&b1);

  n = 0;
```

```c
  while (!fine) {
    scanf("%c",&c);
    if (c== ' ' || c== '\n') {
      converti(n,b1); n=0;
    }
    else {
      cifra = c- '0';
      if (cifra>=0 && cifra<b0)
        n = b0*n + cifra;
      else fine=1;
    }
  }
}
```

```
void converti(int n, int b) {
  int p;

  for (p=1; b*p<=n; p=p*b);
  while (p>0) {
    if (p<=n) {
      printf("%d",n/p); n=n%p;
    }
    else printf("0");
    p = p/b;
  }
  printf("\n");
}
```

Problemi su valori non numerici

- I caratteri in C sono in genere codificati secondo lo standard ASCII:
  - internamente il codice ASCII utilizza 8 bit
  - NON è necessario conoscere le codifiche. E' sufficiente ricordare che i sottoinsiemi di caratteri alfabetici (maiuscoli e minuscoli) e numerici sono contigui
  - esternamente (input/output) i caratteri sono visualizzati in forma testuale
- Le operazioni aritmetiche, di confronto tra caratteri, e il loro ordine, rispettano la codifica ASCII. Per gli operatori aritmetici i codici sono assimilati a binari interi su 8 bit
  - Esempio: 'C' - 'A' vale 2, 'Z' - 'A' vale 25, ma 'Z' e 'a' non sono contigui.

Problemi di codifica di caratteri

- Perchè?
  - Può essere necessario determinare il codice ASCII di un carattere:
    - basta una funzione di conversione di un numero a binario
  - Si può effettuare una ricodifica per:
    - crittografare/decrittografare o compattare/scompattare un testo
- Che Algoritmi?
  - I problemi di codifica si risolvono spesso mediante algoritmi con:
    - iterazioni per trattare sequenze di caratteri
    - gestione del singolo carattere mediante manipolazioni (eventualmente numeriche) sul codice del carattere (codifica/decodifica).

Crittografia semplice

- Formulazione:
  - crittografare il contenuto di un file testo, immagazzinando il risultato in un secondo file
  - la crittografia consiste nel modificare i codici dei caratteri alfabetici e numerici, secondo le regole seguenti:
    - ogni codice numerico n (0..9) viene trasformato nel codice complemento-a-9 (9-n):
    - ogni codice alfabetico ch viene trasformato scambiando maiuscole e minuscole, e facendo il complemento-a- 'z' ('a' + 'z' -ch)
- Soluzione:
  - leggere iterativamente i caratteri dal primo file
  - a seconda del tipo di carattere applicare la codifica opportuna
  - scrivere il carattere via via ottenuto sul secondo file

- Struttura dati:
  - due variabili di tipo puntatore a FILE fpin, fpout per gestire i due file in lettura e scrittura
  - una stringa nomefile per i nomi dei file
  - una variabile char per lettura e ri-codifica dei caratteri.
- Algoritmo:
  - acquisizione dei nomi di file e loro apertura
  - iterazione di lettura di un carattere, ri-codifica, scrittura nel secondo file
    - la ri-codifica viene fatta mediante selezione del sottoinsieme di caratteri, e relativa operazione (sfruttando l'aritmetica dei codici). Ad es., il nuovo codice della 'c' si ottiene sommando ad 'A' la differenza tra 'z' e 'c'

Codice

```
#define MAXRIGA 30
int main(void) {
  char ch, nomefile[MAXRIGA+1];
  FILE *fpin, *fpout;

  printf("nome file in ingresso: ");
  scanf(“%s”, nomefile);
  fpin = fopen(nomefile, "r");
  printf("nome file in uscita: ");
  scanf("%s", nomefile);
  fpout = fopen(nomefile, "w");
```

```
  while (fscanf(fpin, "%c", &ch) == 1) {
    if (ch>= '0' && ch<= '9')
      ch = '0' +('9' -ch);
    else if (ch>= 'a' && ch<= 'z')
      ch = 'A' +('z' -ch);
    else if (ch>= 'A' && ch<= 'Z')
      ch = 'a' +('Z' -ch);
    fprintf(fpout, "%c",ch);
  }
  fclose(fpin); fclose(fpout);
}
```

Problemi di text-processing

- Problemi nei quali occorre manipolare sequenze di caratteri e/o stringhe.
  - Esempio: costruzione o modifica di testo, creazione di messaggio in un dato formato
- Scopi possibili:
  - riconoscimento di un testo (input: menu per input di comandi)
  - visualizzazione di un testo (output: grafica «elementare»)
  - elaborazione di un testo (modifica: formattazione)
- Le operazioni possono essere effettuate a livello di:
  - singoli caratteri (livello più basso)
  - stringhe (livello più alto: una stringa corrisponde a una parola o frase).

- Il C consente input/output sia a livello di carattere che di stringa
- La manipolazione di un testo a livello di stringa dipende dalla disponibilità di funzioni di libreria (o altre scritte dall'utente).
- Il confronto tra testi si effettua:
  - a livello di caratteri, mediante gli operatori relazionali (==, !=, >, <, >=, <=)
  - a livello di stringhe si richiede la funzione strcmp (o strncmp)
- Il costrutto di selezione if:
  - è il più generale (switch può essere espresso in termini di if e non viceversa)
  - utilizzabile sempre (pur di formulare correttamente l'espressione di controllo)

- Il costrutto switch richiede selezione in base a valori costanti. Non può quindi essere applicato:
  - a costanti stringa (perchè sarebbero confrontati i puntatori e non i contenuti delle stringhe)

```
switch (stato) {
  case "Italia": ...
  ...
}
```

  - a insiemi o intervalli di caratteri (se non mediante enumerazione di tutti i caratteri possibili)

```
switch (car) {
  case 'A': case 'B': case 'C': ...
  ...
}
```

- Il costrutto switch richiede selezione in base a valori costanti. Non può quindi essere applicato:
  - a costanti stringa (perchè sarebbero confrontati i puntatori e non i contenuti delle stringhe)

```
switch (stato) {
  case "Italia": ...
  ...
}
```

  - a insiemi di valori (se non mediante enumerazione di tutti i

```
if (strcmp (stato, "Italia") == 0) {
   ...
}
else if (...)
...
```

- Il costrutto switch richiede selezione in base a valori costanti. Non può quindi essere applicato:
  - a costanti stringa (perchè sarebbero confrontati i puntatori e non i
  - tutti i caratteri possibili

```
if (car>='A' && car<='Z') {
   ...
}
else if (...)
...
```

```
switch (car) {
  case 'A': case 'B': case 'C': ...
  ...
}
```

- Il costrutto switch richiede selezione in base a valori costanti. Non pu
  - a
  - a insiemi o int caratteri (se non mediante enumerazione di tutti i caratteri possibili)

i e non i

Si possono usare le funzioni di libreria (ctype.h): **isalpha, isupper, islower, isdigit, isalnum, isxdigit, ispunct, isgraph, isprint, isspace, iscntrl**

```
switch (car) {
  case 'A': case 'B': case 'C': ...
  ...
}
```

- Il costrutto switch richiede selezione in base a valori costanti. Non può quindi essere applicato:
  - a costanti stringa (perchè sarebbero confrontati i puntatori e non i
  - a insiemi o i... ...tteri (se non mediante enumerazione di tutti i caratteri possibili)

```
if (isalpha(car)) {
   ...
}
else if (...)
...
```

```
switch (car) {
  case 'A': case 'B': case 'C': ...
  ...
}
```

Selezione a menu

- La selezione a menu consiste nell'effettuare una scelta, tra le varie disponibili, per eseguire un'azione (un calcolo, una chiamata di funzione o altro):
  - spesso la scelta si basa su un'informazione testuale (un comando o opzione)
  - l'elenco delle possibili opzioni è visualizzata a video (se si tratta di I/O tastiera/video)
  - occorre prevedere un caso di errore (o scelta non valida)
  - il menu viene di solito iterato (una delle opzioni indica fine/uscita).

Menu con scelta su un carattere

- È il caso più semplice. Si prevedono un certo numero di casi, ognuno selezionato da una costante carattere distinta: l'iniziale di un comando oppure il comando stesso
- Eventuali complicazioni:
  - può esser necessario saltare un certo numero di spazi (prima di individuare il carattere di selezione
  - se il carattere è alfabetico, è possibile che occorra ignorare la differenza maiuscolo/minuscolo
- Soluzione: i costrutti if e switch sono entrambi adatti (di solito si utilizza switch)

- Formulazione:
  - scrivere una funzione che, iterativamente, acquisisca da tastiera una stringa (al massimo 50 caratteri, contenente eventuali spazi)
    - il primo carattere diverso da spazio costituisce il selettore
    - se il carattere è 'u' (uscita), occorre terminare l'iterazione
    - se il carattere è uno tra 'A', 'L', 'T' (eventualmente minuscoli), occorre attivare, rispettivamente, le funzioni fA, fL, fT, passando loro come parametro il resto della stringa (oltre il carattere selettore)
    - ogni altro carattere va segnalato come errato.

Codice

```
void menuCarattere (void) {
  const int MAXL=51;
  char riga[MAXL], sel;
  int i, continua=1;

  while (continua) {
    printf("comando (A/L/T, U=uscita): ");
    scanf(" %c", &sel);
    gets(riga);
    ...
```

```
    ...
    switch (toupper(sel)) {
      case 'A' : fA(riga); break;
      case 'L' : fL(riga); break;
      case 'T' : fT(riga); break;
      case 'U' : continua=0; break;
      default: printf("comando errato\n");
    }
  }
}
```

Menu con scelta su una parola

- È il caso meno semplice. Si prevedono un certo numero di casi, ognuno selezionato da una costante stringa distinta: la prima parola di un comando oppure il comando stesso
- Eventuali complicazioni:
  - può esser necessario saltare un certo numero di spazi (prima di individuare la parola di selezione
  - se i caratteri sono alfabetici, è possibile che occorra ignorare la differenza maiuscolo/minuscolo
- Soluzione: è necessario il costrutto if.

- Formulazione:
  - scrivere una funzione che, iterativamente, acquisisca da tastiera una stringa (al massimo 50 caratteri, contenente eventuali spazi)
    - la prima parola diversa da spazio costituisce il selettore
    - se la parola è "fine", occorre terminare l'iterazione
    - se la parola è uno tra "cerca", "mod", "sta" (ignorare differenza maiuscole/minuscole), occorre attivare, rispettivamente, le funzioni cerca, sostituisci, stampa, passando loro come parametro il resto della stringa (oltre la parola di selezione)
    - ogni altra parola va segnalata come errata.

Codice

```
void menuParola (void){
  const int MAXL=51;
  char comando[MAXL], riga[MAXL];
  int i, continua=1;
  while (continua) {
    printf("comando (cerca/modifica");
    printf("stampa/uscita): ");
    scanf("%s", comando); /* comando */
    for (i=0; i<strlen(comando); i++)
      comando[i] = toupper(comando[i]);
    gets(riga); /* resto della riga */
    ...
    ...
    if (strcmp(comando, "CERCA")==0) {
      cerca(riga);
    } else if (strcmp(comando, "MOD")==0) {
      sostituisci(riga);
    } else if (strcmp(comando, "STA")==0) {
      stampa(riga);
    } else if (strcmp(comando, "FINE")==0) {
      continua=0;
    } else {
      printf("comando errato\n");
    }
  }
}
```

Elaborazione testi - livello carattere

- Un testo può essere costruito o modificato a livello di caratteri perchè:
  - il problema viene posto a livello di singoli caratteri (e non parole/stringhe). Non ci sono quindi alternative
  - il problema potrebbe essere risolto (anche parzialmente) a livello di stringhe, ma si sceglie di lavorare a livello di caratteri. Motivi: varianti di funzioni di libreria o miglior gestione di casi particolari
- Attenzione! Nei casi di manipolazione mista (caratteri e stringhe) occorre gestire il terminatore di stringa ('\0')

Costruzione di figure/grafici

- Si tratta di figure formate da caratteri testuali (es. video visto come matrice di caratteri di 25 righe e 80 colonne)
- NON si tratta di grafica basata su punti (pixel), per cui occorrono apposite librerie o linguaggi
- La visualizzazione di caratteri su video (o su file testo) viene effettuata in modo sequenziale (successione di caratteri e righe)
- Un disegno o grafico può essere costruito (e visualizzato):
  - riga per riga (esempi in questa unità)
  - su una matrice di caratteri, da cui si passa successivamente a visualizzare le righe (unità successiva).

Visualizzazione di un rettangolo

- Formulazione:
  - scrivere una funzione che, ricevuti come parametri due numeri interi (identificati con b, h) tracci su video un rettangolo di caratteri '*', avente base e altezza, rispettivamente, di b e h asterischi
- Esempio: per b=5 e h=4 occorre visualizzare

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

- Algoritmo: E' sufficiente una coppia di costrutti iterativi annidati per gestire la stampa dei caratteri organizzandoli per righe e colonne

Codice

```
void rettangolo (int b, int h) {
  int i, j;

  for (i=0; i<h; i++) {
    for (j=0; j<b; j++)
      if (i!=0 && i!=h-1 && j!=0 && j!=b-1)
        printf(" ");
      else
        printf("*");
    printf("\n");
  }
}
```

Visualizzazione di una parabola

- Formulazione:
  - data la parabola di equazione
  - y = ax2 + bx + c = 0
  - si scriva un programma che:
    - acquisisca da tastiera i coefficienti a, b, c
    - acquisisca un intero n (n>0), e i valori degli estremi (x0, xn) di un intervallo per le ascisse
    - acquisisca da tastiera i valori estremi (ymin, ymax) di un intervallo per le ordinate
    - suddivida l'intervallo [x0,xn] in n sotto-intervalli della stessa lunghezza, delimitati dalle ascisse x0,x1,x2,…,xn
  - calcoli i valori di y(xi) per ognuna delle ascisse xi = x0..xn
  - stampi su file un grafico (con asse delle ascisse verticale) che rappresenti la funzione nel rettangolo del piano cartesiano compreso negli intervalli [x0,xn], [ymin,ymax] una riga per ogni valore di ascissa, una colonna per ogni unità sulle ordinate

- Esempio: se si acquisissero da tastiera i valori:
  - a=1.0, b=2.0, c=1.0, n=5, x0=0.0, xn=5.0, ymin=0.0, ymax=50.0
- verrebbero calcolati
  - y(0.0)=1.0, y(1.0)=4.0, y(2.0)=9.0, y(3.0)=16.0, x(4.0)=25.0, x(5.0)=36.0
  - contenuto del file

- Struttura dati: sono sufficienti variabili scalari, per rappresentare:
  - variabile di tipo puntatore a FILE (fpout)
  - coefficienti: a, b, c (float)
  - numero di intervalli: n (int)
  - intervalli: x0, xn, ymin, ymax (float)
  - dati intermedi: passo (lunghezza degli n intervalli) x, y (float)
  - contatori per grafico: i, j (int)
- Algoritmo:
  - input dati e calcolo passo (= lunghezza intervalli)
  - iterazione su x=x0..xn
    - calcolo y(x)
    - se è nell'intervallo [ymin,ymax] converti in intero (j) e visualizza un asterisco dopo j spazi

Codice

```c
#include <stdio.h>
#include <math.h>
int main(void) {
  float a,b,c,x,passo,x0,xn,y,ymin,ymax;
  int i, j, n;
  FILE *fpout = fopen("out.txt", "w");
  printf("Coefficienti (a b c): ");
  scanf("%f%f%f",&a,&b,&c);
  printf("Numero di intervalli: ");
  scanf("%d",&n);
  printf("Intervallo per ascisse: ");
  scanf("%f%f",&x0,&xn);
  printf("Intervallo per ordinate: ");
  scanf("%f%f",&ymin,&ymax);
  ...
```

```c
  ...
  passo = (xn-x0)/n;
  for (i=0; i<=n; i++) {
    x = x0 + i*passo;
    y = a*x*x + b*x + c;
    if (y>=ymin && y<=ymax) {
      for (j=round(y-ymin); j>0; j--)
        fprintf(fpout," ");
      fprintf(fpout,"*");
    }
    fprintf(fpout,"\n");
  }
  fclose(fpout);
}
```

Elaborazione testi - livello stringa

- Un testo può essere costruito o modificato a livello di stringhe se:
  - è possibile identificare sottostringhe (sequenze di caratteri) sulle quali applicare operazioni di tipo unitario
  - le operazioni su stringhe debbono trovarsi in libreria, oppure essere chiamate funzioni realizzate dal programmatore
- Spesso le sottostringhe sono sequenza di caratteri separate da spazi (facile l'input)
- Talvolta le sottostringhe includono spazi e/o sono delimitate da altri caratteri.

Formattazione di testo

- Formulazione:
  - è dato un file testo, visto come un insieme di righe, scomponibili in sottostringhe (di non più di 20 caratteri) separate da spazi (oppure '\t' o '\n').
  - si realizzi una funzione C che, letto il file, ne copi il contenuto in un altro file (i nomi dei file sono ricevuti come parametri), dopo aver:
    - ridotto le sequenze di più spazi ad un solo spazio
    - inserito (in sostituzione di spazi) o eliminato caratteri a-capo ('\n') in modo tale che ogni riga abbia la massima lunghezza possibile, minore o uguale a lmax (terzo parametro della funzione)

- Soluzione:
  - è possibile operare sia a livello di caratteri che di stringhe
  - una possibile soluzione per la gestione di stringhe parte dal fatto che l'input mediante scanf("%s") permette di isolare in modo automatico stringhe separate da spazi (oppure '\t' o '\n')
    - iterazione di input di stringhe, con contestuale output e conteggio della lunghezza di una riga
    - prima di fare l'output di una stringa, si decide (in base al conteggio di lunghezza riga) se occorre stampare un carattere a-capo.

Codice

```
void formatta (char nin[], char nout[],
               int lmax) {
  const int STRLEN=21;
  FILE *fin=fopen(nin, "r");
  FILE *fout=fopen(nout, "w");
  char parola[STRLEN];
  int l;

  l=0;
  ...
```

```
  while (fscanf(fin, "%s",parola)==1) {
    if (l+1+strlen(parola) > lmax) {
      fprintf(fout, "\n%s",parola);
      l=strlen(parola);
    }
    else {
      fprintf(fout,"%s%s",
                    l==0? "":" ",parola);
      l+=l==0?0:1+strlen(parola);
    }
  }
  fclose(fin); fclose(fout);
}
```

Codice

```
void formatta (char nin[], char nout[],
               int lmax) {
  const int STRLEN=21;
  FILE *fin=fopen(nin, "r");
  FILE *fout=fopen(nout, "w");
  char parola[STRLEN];
```

```
  while (fscanf(fin, "%s",parola)==1) {
    if (l+1+strlen(parola) > lmax) {
      fprintf(fout, "\n%s",parola);
      l=strlen(parola);
    }
    else {
      fprintf(fout,"%s%s",
                    l==0? "":" ",parola);
      l+=l==0?0:1+strlen(parola);
    }
  }
  fclose(fin); fclose(fout);
}
```

Attenzione: test per non stampare uno spazio prima della prima parola
Espressione condizionale C:
`cond ? exprT : exprF`

Problemi di verifica e selezione

- Verifica: decidere se un insieme di informazioni o dati rispettano un determinato criterio di accettazione:
  - la risposta ad un problema di verifica è Booleana (SÌ/NO)
  - si possono verificare dati singoli oppure sequenze (insiemi) di dati
  - un problema può consistere in una sola verifica o in più verifiche (su dati diversi)
- Selezione: separare i dati che rispettano un criterio di accettazione/verifica (rispetto a quelli che non lo rispettano).

Criteri di accettazione

∀, ∃ Gottlob Frege, 1879

- I criteri possono essere:
- espressioni logiche (logica proposizionale)
- condizione (proprietà) p su di un insieme di dati S espressa mediante:
  - quantificatore universale: ∀:

    ∀ x ∈S | p è vera

    per tutti gli x appartenenti ad S la proprietà p è vera
  - quantificatore esistenziale ∃:

    ∃ x ∈S | p è vera

    esiste almeno un x appartenente ad S per cui la proprietà p è vera

I quantificatori NON appartengono alla logica proposizionale ma alle logiche di ordine superiore (es. primo/secondo ordine)

Simboli logici (logica proposizionale)

- Logica proposizionale e algebra di Boole sono ***in pratica*** equivalenti, pur denotando due contesti distinti
- Gli operatori logici di base (NOT, AND, OR) sono rappresentati in modi diversi a seconda del contesto (matematico, applicativo, linguaggio di programmazione)

Operazione	Simboli			Linguaggio C	Risultato
Congiunzione logica (AND)	&	∧	•	&&	vero se operandi veri
Disgiunzione logica (OR)	|	∨	+	||	vero se almeno un operando vero
Complementazione logica o negazione (NOT)	!	¬	~	!	vero se operando falso (e viceversa)

Dualità ∀, ∃

- $\neg(\forall\, x \in S \mid p \text{ è vera}) \Leftrightarrow \exists\, x \in S \mid p \text{ è falsa}$

  non è vero che per tutti gli x appartenenti ad S la proprietà p è vera

  EQUIVALE A

  esiste almeno un x appartenente ad S per cui la proprietà p è falsa

- $\neg(\exists\, x \in S \mid p \text{ è vera}) \Leftrightarrow \forall\, x \in S \mid p \text{ è falsa}$

  non è vero che esiste almeno un x appartenente ad S per cui la proprietà p è vera

  EQUIVALE A

  per tutti gli x appartenenti ad S la proprietà p è falsa

Esempio: monotonicità di una sequenza

Data una sequenza di N interi $S = (x_0, x_1, \dots, x_{N-1})$:

- proprietà p: è una sequenza monotona crescente

$$\forall\, x_i, x_{i+1} \in S \mid (x_i \leq x_{i+1}) \quad 0 \leq i < N-1$$

per tutte le coppie di elementi adiacenti vale la relazione d'ordine ≤

- proprietà ¬p: NON è una sequenza monotona crescente

$$\exists\, x_i, x_{i+1} \in S \mid (x_i > x_{i+1}) \quad 0 \leq i < N-1$$

esiste almeno una coppia di elementi adiacenti per cui non vale la relazione d'ordine ≤, quindi vale >

Realizzazione in C

- variabile intera utilizzata come logica e inizializzata al verdetto a 1 del quantificatore universale
- costrutto iterativo per enumerare tutti gli elementi dell'insieme S
- per ogni iterazione: controllo se il verdetto iniziale è confermato o contraddetto
  - la variabile logica viene modificata solo in caso di verdetto contraddetto

Codice

```
...
int Monotona = 1; // verdetto inizializzato a 1
int i;
int xi, xj;
printf("x0= "); scanf("%d", &xi);

for (i=1; i<N; i++) {
  printf("x%d= ", i); scanf("%d", &xj);
  if(xi > xj)  // test sul verdetto tra due dati adiacenti
    Monotona = 0;  // eventuale modifica del verdetto
  xi = xj;
}
```

Errore comune

```
int Monotona = 1;
int i;
int xi, xj;
printf("x0= "); scanf("%d", &xi);
for (i=1; i<N; i++) {
  printf("x%d= ", i); scanf("%d", &xj);
  if(xi > xj)
    Monotona = 0;
  else
    Monotona = 1;
  xi = xj;
}
```

il verdetto dipende dall'esito dell'ultimo test

Variante: uscita anticipata strutturata

```
...
int Monotona = 1;
int i;
int xi, xj;
printf("x0= "); scanf("%d", &xi);

for (i=1; i<N && Monotona; i++) {
  printf("x%d= ", i); scanf("%d", &xj);
  if(xi > xj)
    Monotona = 0;
  xi = xj;
}
```

uscita strutturata con flag

Variante: uscita anticipata non strutturata (1)

```
int Monotona = 1;
int i;
int xi, xj;
printf("x0= "); scanf("%d", &xi);
for (i=1; i<N; i++) {
  printf("x%d= ", i); scanf("%d", &xj);
  if(xi > xj) {
    Monotona = 0;
    break;
  }
  xi = xj;
}
```

uscita non strutturata dal ciclo for

Variante: uscita anticipata non strutturata (2)

```
int Monotona = 1;
int i;
int xi, xj;
printf("x0= "); scanf("%d", &xi);
for (i=1; i<N; i++) {
  printf("x%d= ", i); scanf("%d", &xj);
  if(xi > xj) {
    Monotona = 0;
    return 0;
  }
  xi = xj;
}
return 1;
```

uscita non strutturata dalla funzione

```
int Monotona = 1;
int i;
int xi, xj;
printf("x0= "); scanf("%d", &xi);
for (i=1; i<N; i++) {
  printf("x%d= ", i); scanf("%d"
  if(xi > xj) {
    Monotona = 0;
    return 0;
  }
  xi = xj;
}
return 1;
```

verdetto ridondante sostituito dal valore di ritorno

Verifiche su sequenze

- Verificare una sequenza di dati significa decidere se la sequenza rispetta un criterio di accettazione.
- Esempi di criteri:
  - sommatoria/media dei dati: superiore o inferiore a limite
  - confronto tra dati adiacenti: sequenza di dati crescenti, differenza inferiore a massimo
  - regole su gruppi di dati adiacenti:
    - sono vietate più di 2 consonanti consecutive, i segni di punteggiatura sono seguiti da spazio, 'p' e 'b' non possono essere precedute da 'n' …
    - un pacchetto di dati trasmessi deve rispettare regole di formato.

Verifica di ordine alfabetico

- Formulazione:
  - un file testo contiene un elenco di persone, ognuna delle quali è rappresentata, su una riga del file (al max. 50 caratteri), da cognome e nome (eventualmente contenenti spazi)
  - si scriva una funzione C, ricevuto come parametro il puntatore al file (già aperto), verifichi che i dati siano in ordine alfabetico, ritornando 1 in caso affermativo (0 in caso negativo)
- Soluzione:
  - si tratta di analizzare iterativamente i dati, confrontando progressivamente i due ultimi acquisiti
  - criterio di accettazione: quantificazione
  - verdetto: valore di ritorno della funzione di verifica
  - uscita anticipata non strutturata.

- Struttura dati:
  - due stringhe, rispettivamente per l'ultima (riga1) e la penultima riga (riga0) acquisite dal file
- Algoritmo:
  - si leggono progressivamente le righe del file, confrontando le ultime due:
    - non appena viene violato il criterio di ordinamento si ritorna 0
    - se viene rispettato il criterio di ordinamento si procede
    - al termine di ogni iterazione l'ultima riga diviene la penultima
    - se non è mai stato violato il criterio si ritorna 1
  - occorre gestire a parte la lettura della prima riga.

Codice

```
int verificaOrdine (FILE *fp) {
  const int MAXC=50;
  char riga0[MAXC+1], riga1[MAXC+1];

  fgets(riga0,MAXC,fp);
  while (fgets(riga1,MAXC,fp)!=NULL) {
    if (strcmp(riga1,riga0)<0)
      return 0;
    strcpy(riga0,riga1);
  }
  return 1;
}
```

Verifica di congruenza dati

- Formulazione:
  - un file testo contiene una sequenza di temperature (in gradi Celsius) rilevate da un sensore termico nell'arco di una giornata (a distanza di 5 minuti l'una dall'altra). Le temperature (numeri reali) sono separate da spazi o a-capo
  - si scriva una funzione C che, ricevuto come parametro il puntatore al file (già aperto), verifichi che, in ogni intervallo di 10 minuti, la temperatura non vari di più di 5 gradi, ritornando 1 in caso affermativo (0 in caso negativo, da interpretare come malfunzionamento di sensore o controllo termico)

- Soluzione:
  - analizzare iterativamente i dati, verificando progressivamente gli ultimi tre dati acquisiti
  - criterio di accettazione:
    - condizione booleana sui primi 2 dati
    - quantificazione sui restanti
  - verdetto: valore di ritorno della funzione di verifica
  - uscita anticipata non strutturata.

- Struttura dati:
  - 3 variabili reali (float) per le ultime tre letture: t0, t1, t2 (t2 è l'ultimo dato acquisito)
- Algoritmo:
  - si leggono iterativamente i dati, controllando gli ultimi tre acquisiti (i primi due sono letti a parte)
    - non appena |t2-t0|>5 e |t2-t1|>5 (|t1-t0|>5 è già stata verificata), si ritorna 0
    - se viene rispettato il criterio di ordinamento, aggiornando t0 e t1
    - se non è mai stato violato il criterio si ritorna 1.

Codice

```
int verificaTemperature (FILE *fp) {
  float t0, t1, t2;

  fscanf(fp, "%f%f",&t0,&t1);
  if (abs(t1-t0)>5)
    return 0;
  while (fscanf(fp, "%f",&t2)==1) {
    if (abs(t2-t0)>5 || abs(t2-t1)>5)
      return 0;
    t0=t1;
    t1=t2;
  }
  return 1;
}
```

Selezione di dati

- Contestualmente alla verifica di più dati (o sequenze/insiemi) di dati, è possibile discriminare i dati che corrispondono al criterio di verifica, rispetto agli altri
- La selezione può essere vista come una variante della verifica:
  - i dati vengono dapprima verificati
  - quelli che corrispondono al criterio di accettazione vengono scelti
- La selezione è solitamente un processo iterativo:
  - più dati (o insiemi di dati) vengono verificati
  - una parte di questi viene selezionata.

Borse di studio

- Formulazione:
  - un file testo contiene un elenco di studenti, per ognuno dei quali
    - una riga riporta il numero di matricola (preceduto da ‘#’)
    - la riga successiva riporta cognome e nome
    - le righe successive riportano i voti di esami superati, uno per riga
  - si scriva un programma C che
    - legga il file e acquisisca da tastiera due numeri mmin (reale) e nmin (intero)
    - selezioni gli studenti con media non inferiore a mmin e numero di esami superati non inferiore a nmin
    - scriva nomi e numeri di matricola degli studenti selezionati su un secondo file (i nomi dei file sono acquisiti da tastiera).

- Soluzione:
  - analizzare iterativamente i dati, considerando di volta in volta i dati relativi a uno studente, del quale occorre calcolare numero di esami e media
- Struttura dati: bastano variabili scalari:
  - 2 stringhe per nomi di file, cognome e nome e matricola: s0, s1
  - 2 puntatori a file (da aprire contemporaneamente, uno in lettura e uno in scrittura): fin, fout
  - 2 interi per il conteggio degli esami (ne), e soglia minima (nmin)
  - 3 reali per voto corrente (voto), media dei voti (media) e media minima (mmin)

- Algoritmo:
  - la verifica dei dati consiste in:
    - lettura dei dati relativi a uno studente
    - calcolo di numero esami e media
    - confronto con le soglie minime previste
  - la soluzione consiste in una doppia iterazione:
    - esterna (per ogni studente): acquisizione di matricola, cognome e nome
    - interna (per ogni voto): calcolo numero esami e media
  - la separazione tra le informazioni relative a due studenti successivi si ottiene ricoscendo il carattere ‘#’ (inizio sezione relativa a nuovo studente).

Codice

```
#define MAXL 100
#include <stdio.h>
int main(void) {
  char s0[MAXL+1], s1[MAXL+1];
  FILE *fin, *fout;
  int nmin, ne;
  float mmin, media, voto;

  printf("File ingresso: "); scanf("%s", s0);
  printf("File uscita: "); scanf("%s", s1);
  fin = fopen(s0, "r"); fout = fopen(s1,"w");
  printf("soglie min. n. esami e media: ");
  scanf("%d%f", &nmin, &mmin);
  ...
```

```
  while (fgets(s0,MAXL,fin)!=NULL) {
    fgets(s1,MAXL,fin);
    ne=0; media=0.0;
    while (fscanf(fin, "%f",&voto)==1) {
      media += voto; ne++;
    }
    media = media/ne;
    if (media >= mmin && ne>=nmin) {
      fprintf(fout, "%s%s",s0,s1);
    }
  }
  fclose(fin); fclose(fout);
}
```

Codice

```
#define MAXL 100
#include <stdio.h>
int main(void) {
  char s0[MAXL+1], s1[MAXL+1];
  FILE *fin, *fout;
  int nmin, ne;
  float mmin, media, voto;

  printf("File ingress
  printf("File uscita
  fin = fopen(s0, "r"
  printf("soglie min. n. esami e media: ");
  scanf("%d%f", &nmin, &mmin);
  ...
```

```
  while (fgets(s0,MAXL,fin)!=NULL) {
    fgets(s1,MAX ,fin);
    ne=0; media  ;
    while (fsc     "%f",&voto)==1) {
      media +=      ++;
    }
    media = r
  }
  fclose(fin); fclose(fout);
}
```

Legge riga contenente matricola (senza verificare presenza di '#')

Codice

```
#define MAXL 100
#include <stdio.h>
int main(void) {
  char s0[MAXL+1], s1[MAXL+1];
  FILE *fin, *fout;
  int nmin, ne;
  float mmin, media, voto;

  printf("File ingress
  printf("File uscita
  fin = fopen(s0, "r");
  printf("soglie min. n. esami e media: ");
  scanf("%d%f", &nmin, &mmin);
  ...
```

```
  while (fgets(s0,MAXL,fin)!=NULL) {
    fgets(s1,MAXL,fin);
    ne=0;  media=0.0;
    wh     (fscanf(fin, "%f",&voto)==1) {
           ia += voto; ne++;

           a = media/ne;
           media >= mmin && ne>=nmin) {
           fout, "%s%s",s0,s1);
    }
  fclose(fin); fclose(fout);
}
```

Legge cognome e nome

Codice

```
#define MAXL 100
#include <stdio.h>
int main(void) {
  char s0[MAXL+1], s1[MAXL+1];
  FILE *fin, *fout;
  int nmin, ne;
  float mmin, media, voto;

  printf("File ingresso: "); scanf("%s", s0);
  printf("File uscita: "); scanf("%s", s1);
  fin = fopen(s0, "r"); fout = fopen(s1, "w");
  printf("soglie m
  scanf("%d%f", &nm
  ...
```

```
  while (fgets(s0,MAXL,fin)!=NULL) {
    fgets(s1,MAXL,fin);
    ne=0; media=0.0;
    while (fscanf(fin, "%f",&voto)==1) {
      media += voto; ne++;
    }
    media = media/n
    if (media >=          e>=nmin) {
      fprint                ",s0,s1);
    }
}
```

Acquisisce numeri fino a quando il carattere ‘#’ lo impedisce

Codice

```c
#define MAXL 100
#include <stdio.h>
int main(void) {
  char s0[MAXL+1], s1[MAXL+1];
  FILE *fin, *fout;
  int nmin, ne;
  float mmin, media, voto;

  printf("File ingresso: "); scanf("%s", s0);
  printf("File uscita: "); scanf("%s", s1);
  fin = fopen(s0, "r"); fout = fopen(s1,"w");
  printf("soglie min. n. esami e media: ");
  scanf("%d%f", &nmin, &mmin);
  ...
```

```c
  while (fgets(s0,MAXL,fin)!=NULL) {
    fgets(s1,MAXL,fin);
    ne=0; media=0.0;
    while (fscanf(fin, "%f",&voto)==1) {
      media += voto; ne++;
    }
    media = media/ne;
    if (media >= mmin && ne>=nmin) {
      fprintf(fout, "%s%s",s0,s1);
    }
  }
  fclose(fin); fclose(fout);
}
```

Problemi di ordinamento

- Un problema di ordinamento consiste nella richiesta di permutare una sequenza di dati, in modo tale che (dopo la permutazione) sia verificato un criterio di ordinamento
- Per ordinare dei dati, occorre un operatore (o una funzione) di confronto tra coppie di dati, tale da decidere quale dei due precede l'altro secondo il criterio di ordinamento (è ammessa l'uguaglianza tra dati, che consente ad ognuno di precedere l'altro): di solito si usano operatori relazionali (<, <=, >, >=).

Ordinamenti totali e parziali

- Ordinamento “totale”:
  - in base al criterio scelto, un dato precede (nella sequenza ordinata) tutti i successivi. Ogni dato è confrontabile con tutti gli altri.
  - Esempio: ordinamento alfabetico tra nomi
- Ordinamento “parziale”:
  - il criterio di confronto scelto non definisce una relazione di precedenza tra alcune coppie di dati.
  - Esempi:
    - ordinamento alfabetico di un insieme di nomi e numeri (l’ordinamento alfabetico non è definito sui numeri);
    - ordinamento di nomi in base alla lettera iniziale (non è definito l’ordine relativo tra nomi con la stessa iniziale)

Algoritmi di ordinamento “parziale”

- La maggioranza degli algoritmi di ordinamento “totale” richiede l’utilizzo di vettori (per i passaggi intermedi). Tali algoritmi saranno trattati con i vettori
- Semplici problemi di ordinamento “parziale” sono risolvibili mediante la ripetizione di problemi di selezione
- Siccome sarà necessario esaminare più volte i dati, per evitare l’uso di vettori, si utilizzerà un file, letto più volte (soluzione poco efficiente).

Ordinamento di punti per quadrante

- Formulazione:
  - un file testo contiene un elenco di punti del piano, ognuno rappresentato su una riga del file mediante due numeri reali (le coordinate)
  - si scriva una funzione C che riscriva i punti su un secondo file (i nomi di entrambi i file sono ricevuti come parametri), dopo averli ordinati in base al quadrante di appartenenza
  - NB: i punti appartenenti agli assi cartesiani possono essere attribuiti indifferentemente ad uno dei quadranti adiacenti.

- Soluzione:
  - leggere 4 volte il file di ingresso, selezionando ogni volta (e scrivendo sul file in uscita) i punti di un quadrante
- Struttura dati e algoritmo:
  - risulta opportuno scrivere una funzione di selezione dei dati di un quadrante. La funzione viene chiamata 4 volte. Il quadrante viene identificato mediante due parametri interi (uno per le ascisse e uno per le ordinate): +1 indica valori positivi, -1 indica valori negativi. I quadranti sono denotati dalle coppie (1,1), (-1,1), (-1,-1), (1,-1).

Codice

```
void ordinaPunti (char nin[], char nout[]) {
  FILE *fin, *fout;
  fout = fopen(nout, "w");
  fin = fopen(nin, "r");
  selezionaPunti(fin,fout,1,1);
  fclose(fin); fin = fopen(nin, "r");
  selezionaPunti(fin,fout,-1,1);
  fclose(fin); fin = fopen(nin, "r");
  selezionaPunti(fin,fout,-1,-1);
  fclose(fin); fin = fopen(nin, "r");
  selezionaPunti(fin,fout,1,-1);
  fclose(fin);
  fclose(fout);
}
```

```
void selezionaPunti (
  FILE *fi, FILE *fo, int sx, int sy) {
  float x,y;
  int xOK, yOK;
  while (fscanf(fi, "%f%f",&x,&y)==2) {
    xOK = x*sx>0.0 || (x==0.0 && sx>0);
    yOK = y*sy>0.0 || (y==0.0 && sy>0);
    if (xOK && yOK)
      fprintf(fo, "%f %f\n",x,y);
  }
}
```

Codice

```
void ordinaPunti (char nin[], char nout[]) {
  FILE *fin, *fout;
  fout = fopen(nout, "w");
  fin = fopen(nin, "r");
  selezionaPunti(fin,fout,1,1);
  fclose(fin); fin = fopen(nin, "r");
  selezionaPunti(fin,fout,-1,1);
  fclose(fin); fin = fopen(nin, "r");
  selezionaPunti(fin,fout,-1,-1);
  fclose(fin); fin = fopen(nin, "r");
  selezionaPunti(fin,fout,1,-1);
  fclose(fin);
  fclose(fout);
}
```

```
void selezionaPunti (
  FILE *fi, FILE *fo, int sx, int sy) {
  float x,y;
  int xOK, yOK;
  while (fscanf(fi, "%f%f",&x,&y)==2) {
    xOK = x*sx>0.0 || (x==0.0 && sx>0);
    yOK = y*sy>0.0 || (y==0.0 && sy>0);
    if (xOK && yOK)
      fprintf(fo, "%f %f\n",x,y);
  }
}
```

Codice

```
void ordinaPunti (char nin[], char nout[]) {
  FI
  fo
  fi
  selezionaPunti(fin,fout,1,1);
  fclose(fin); fin = fopen(nin, "r");
  selezionaPunti(fin,fout,-1,1);
  fclose(fin); fin = fopen(nin, "r");
  selezionaPunti(fin,fout,-1,-1);
  fclose(fin); fin = fopen(nin, "r");
  selezionaPunti(fin,fout,1,-1);
  fclose(fin);
  fclose(fout);
}
```

Ascissa(ordinata) OK
se concorde con sx(sy)

```
void selezionaPunti (
  FILE *fi, FILE *fo, int sx, int sy) {
  float x,y;
  int xOK, yOK;
  while (fscanf(fi, "%f%f",&x,&y)==2) {
    xOK = x*sx>0.0 || (x==0.0 && sx>0);
    yOK = y*sy>0.0 || (y==0.0 && sy>0);
    if (xOK && yOK)
      fprintf(fo, "%f %f\n",x,y);
  }
}
```

Codice

```
void ordinaPunti (char nin[], char nout[]) {
  FI
  fo
  fi
  selezionaPunti(fin,fout,1,1);
  fclose(fin); fin = fopen(nin, "r");
  selezionaPunti(fin,fout,-1,1);
  fclose(fin); fin = fopen(nin, "r");
  selezionaPunti(fin,fout,-1,-1);
  fclose(fin); fin = fopen(nin, "r");
  selezionaPunti(fin,fout,1,-1);
  fclose(fin);
  fclose(fout);
}
```

```
void selezionaPunti (
  FILE *fi, FILE *fo, int sx, int sy) {
  float x,y;
  int xOK, yOK;
  while (fscanf(fi, "%f%f",&x,&y)==2) {
    xOK = x*sx>0.0 || (x==0.0 && sx>0);
    yOK = y*sy>0.0 || (y==0.0 && sy>0);
    if (xOK && yOK)
      fprintf(fo, "
```

Ascissa(ordinata) OK se concorde con sx(sy)

Ascissa(ordinata) OK se nulla e sx(sy) è positivo (convenzionalmente 0 viene interpretato come positivo)