POLITECNICO
DI TORINO

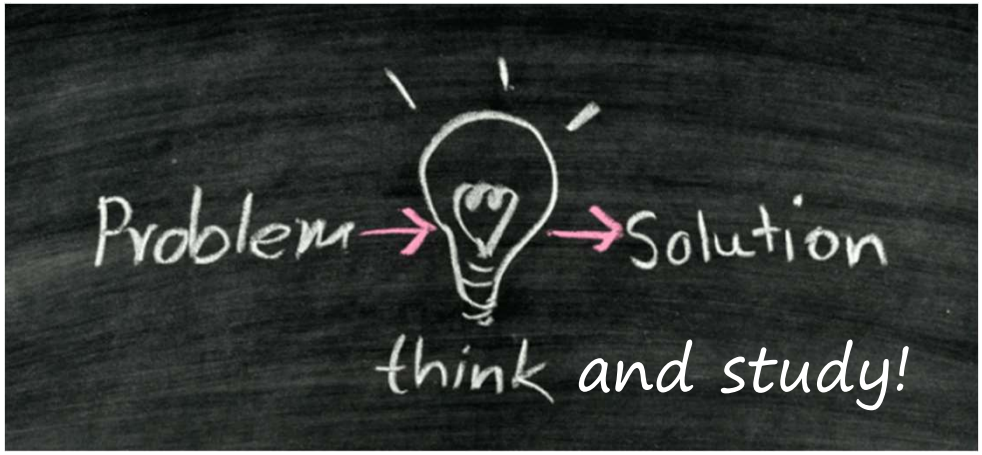
Dipartimento
di Automatica e Informatica

Capitolo 3: Problem-solving su dati scalari

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

Dal C ai programmi

- Costrutti e regole del linguaggio
  - Dati per scontati !!! (quasi)
- Programmare → "dal problema alla soluzione" (utilizzando il linguaggio C)
  - Strategia → problem solving
  - Esperienza e abilità personale
  - Imparare da soluzioni proposte
  - NOVITA': Classificazione di problemi



Classi di problemi

	Senza vettori	Con vettori/matrici
Numerici	Equaz. 2° grado Serie e successioni numeriche …	Statistiche per gruppi Operazioni su insiemi di numeri Generazione numeri primi Somme/prodotti matriciali
Codifica	Conversioni di base (es. binario/decimale) Crittografia di testo …	Conversioni tra basi numeriche Ricodifica testi utilizzando tabelle di conversione
Elab. Testi	Manipolazione stringhe Menu con scelta Grafico di funzione (asse X verticale)	Conteggio caratteri in testo Grafico funzione (asse X orizzontale) Formattazione testo (centrare, eliminare spazi)
Verifica/ selezione	Verifica di ordinamento/congruenza di dati Verifica mosse di un gioco Filtro su elenco di dati Ricerca massimo o minimo Ordinamento parziale	Verifica di unicità (o ripetizione) di dati Selezione di dati in base a criterio di accettazione Ricerca di dato in tabella (in base a nome/stringa) Ordinamento per selezione

Classi di problemi

	Senza vettori	Con vettori/matrici
Numerici	Equaz. 2° grado Serie e successioni numeriche …	Statistiche per gruppi Operazioni su insiemi di numeri Generazione numeri primi Somme/prodotti matriciali
Codifica	Conversioni di base (es. binario/decimale) Crittografia di testo …	Conversioni tra basi numeriche Ricodifica testi utilizzando tabelle di conversione
Elab. Testi	Manipolazione stringhe Menu con scelta Grafico di funzione (asse X verticale)	Conteggio caratteri in testo Grafico funzione (asse X orizzontale) Formattazione testo (centrare, eliminare spazi)
Verifica/ selezione	Verifica di ordinamento/congruenza di dati Verifica mosse di un gioco Filtro su elenco di dati Ricerca massimo o minimo Ordinamento parziale	Verifica di unicità (o ripetizione) di dati Selezione di dati in base a criterio di accettazione Ricerca di dato in tabella (in base a nome/stringa) Ordinamento per selezione

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

Attenzione!

- 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
  - Allora non devo salvare tutto lo storico, e posso lavorare senza salvare tutti i dati
  - Non ho bisogno di memorizzare i dati in vettori e matrici (farlo sarebbe sbagliato ed inefficiente!)

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

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

Equazione di II grado

- 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 soluzioni complesse
- Algoritmo: selezione tra 5 casi mediante schemi di if … else per distinguere impossibile, indeterminata, I grado, II grado
  - NB: non è consigliabile il costrutto switch (effettua la selezione su valori interi)

CONDIZIONE	TIPOLOGIA	EQUAZIONE
a == 0 && b==0 && c == 0	Indeterminata	-
a == 0 && b==0 && c != 0	Impossibile	-
a == 0 && b!=0	Eq. I grado	x = -c/b
a != 0 && Δ == 0	2 soluzioni reali coincidenti	x0 = x1 = -b/2a
a != 0 && Δ > 0	2 soluzioni reali distinte	$X_{0,1}$ = (-b ± √Δ)/2a
a != 0 && Δ < 0	2 soluzioni immaginarie	Re(x) = -b/2a Im(x) = √-Δ/2a

Codice versione 1: `if` non annidati

RICALCA ESATTAMENTE LA TABELLA PER LE CONDIZIONI SU a, b, 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);
    }
  }
  return 0;
}
```

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) {
    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 coincidenti: ");
      printf("%f %f\n",x0,x1);
    }
    else /* if delta != 0 */
      if (delta > 0) {
        x0 = (-b-sqrt(delta))/(2*a);
        x1 = (-b+sqrt(delta))/(2*a);
        printf("2 sol. eali dist.: %f %f\n",x0,x1);
      }
      else { /* delta < 0 */
        re = -b/(2*a);
        im = sqrt(-delta)/(2*a);
        printf("2 sol. comp.con.: \n x0=%f-i*%f
                x1=%f+i*%f\n", re, im, re, im);
      }
  }
  return 0;
}
```

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

Area di triangolo rettangolo

- 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

```
#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);
 return 0;
}
```

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 = (float)(b*c)/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

```
...
int main(void) {
  ...
  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 Hn 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 Hn
- Soluzione:
  - Generare iterativamente i termini della successione 1, 1/2, 1/3, ...
  - Calcolare in modo incrementale la sommatoria dei termini generati

Ridotta n-esima di serie armonica

- 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 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

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/transcodifica): 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, ...)
  - Possono essere visualizzati 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, ...)
  - Possono essere visualizzati 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?

Problemi su valori numerici

- 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

Problemi su valori numerici

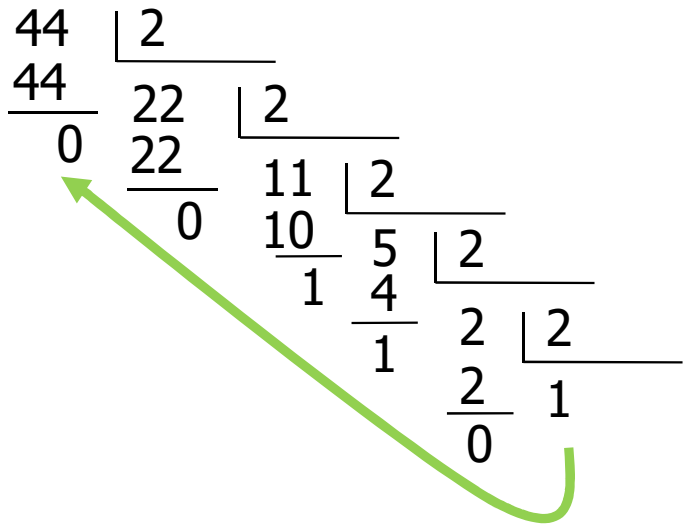
- I problemi di codifica si risolvono spesso mediante algoritmi iterative
  - Ora si trattano solo algoritmi che non richiedono di salvare le cifre
  - Se devo salvare le cifre, devo usare i vettori
- A seconda del formato, può essere necessario gestire cifre numeriche, segno e/o esponente (potenza della 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)
    - Dovrei salvare tutte le cifre
  - 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

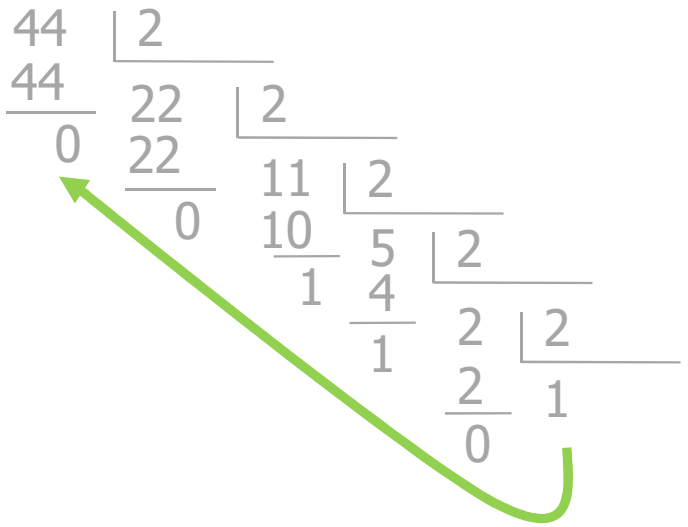
Divisioni successive:
serve un vettore



$44_{10} = 101100_2$

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

Codifica binaria di un intero

- 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 p come 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 p è minore di 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
- altrimenti 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
    - L'acquisizione/conversione termina inserendo una cifra non valida
  - Ogni intero acquisito va convertito dalla base b0 alla base b1 e stampato a video

- Soluzione:
  - Acquisire da tastiera b0 e b1
  - Iterare per acquisire e controllare le cifre
  - Convertire passando attraverso la rappresentazione interna standard (binaria)

Conversione tra basi

- Struttura dati: variabili intere:
  - Due variabili per le basi b0 e b1
  - Una variabile per il numero (n) convertito in base 10 dal formato originariamente espresso nella base b0 (cifre inserite da tastiera)
  - 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

Conversione tra basi

- 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;
  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;
    }
  }
}
```

Codice

```
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");
}
```