Esercizi su

Notazione posizionale

Istruzioni iterative e di scelta

Libreria matematica

- Se utilizzate la libreria matematica
  - Aggiungete *#include <math.h>* se usate il C
    - Tipicamente non serve nessuna direttiva in C++
  - Passate anche l'opzione *-lm* al *g++*

Abbreviazioni e prefissi

- b bit
- B byte
- K Kilo 1000 oppure 1024
- M Mega 1000*K oppure 1024*K
- G Giga 1000*Moppure 1024*M
- Esempi del secondo uso:
  - 3 KB = 3 * 1024 byte
  - 4 Mb = 4 * 1024 * 1024 bit

Ambiguità

- C'è spesso ambiguità sull'uso dei prefissi K, M, G, ...
- Per indicare le dimensioni della memoria principale si usano nel significato informatico di potenze di 2
- Per altre grandezze (dimensioni dischi, velocità di trasferimento) si usano quasi sempre nel significato scientifico di potenze di 10

Prefissi binari

- E' stato proposto l'uso di **prefissi binari** dedicati, proprio per distinguerli da quelli scientifici
- 1024 Ki kibi
- 1024*Ki Mi mebi
- 1024*Mi Gi gibi
- 1024*Gi Ti tebi
- 1024*Ti Pi pebi
- 1024*Pi Ei exbi
- 1024*Ei Zi zebi
- 1024*Zi Yi yobi

Base 2

- Per capire fino in fondo come sono rappresentate le informazioni in un calcolatore occorre conoscere la rappresentazione dei numeri in base 2
- Il motivo è che le informazioni sono rappresentate come sequenze di bit, ossia cifre con due soli possibili valori

Basi e cifre 1/2

- Partiamo dalla rappresentazione di un numero in una generica base
- Cominciamo dalla rappresentazione dei numeri naturali

Basi e cifre 2/2

- Rappresentazione di un numero in una data base: sequenza di cifre
- *Cifra*: simbolo rappresentante un numero
- *Base*: numero (naturale) di valori possibili per ciascuna cifra
- In base $b > 0$ si utilizzano b cifre distinte, per rappresentare i valori $0,\ 1,\ 1+1,\ 1+1+1,\ \ldots,\ b-1$

Cifre e numeri in base 10

- Es: in base 10 le cifre sono

0 che rappresenta il valore 0

1 che rappresenta il valore 1

2 che rappresenta il valore 1+1

3 che rappresenta il valore 1+1+1

Simbolo grafico

Concetto astratto di numero naturale

.

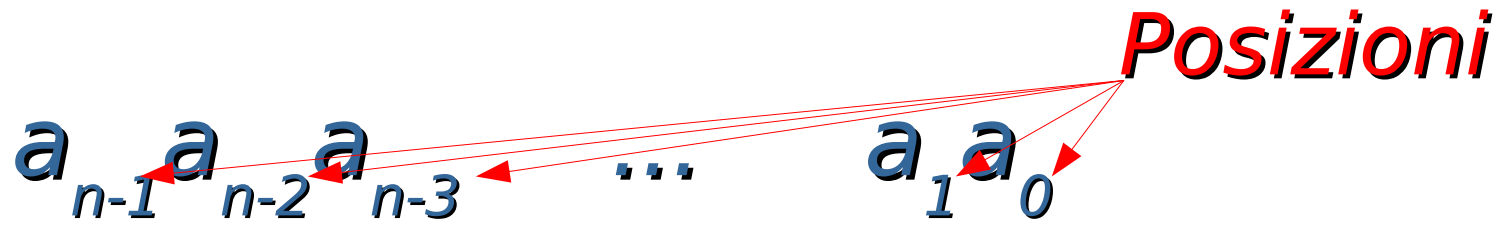
.

.

9 che rappresenta il valore 1+1+1+1+1+1+1+1+1

Notazione posizionale 2/3

- Rappresentazione di un numero su *n* cifre in base *b*:



$$a_i \in \{0, 1, \ldots, b - 1\}$$

- Es: Notazione decimale:

$b = 10$, $a_i \in \{0, 1, 2, \ldots, 9\}$

$345 \Rightarrow a_2 = 3, a_1 = 4, a_0 = 5$

Notazione

- Per rendere esplicita la base utilizzata, si può utilizzare la notazione

$$[\, x \,]_b$$

$$a_i \in \{0, 1, \ldots, b - 1\}$$

dove *x* è una qualsiasi espressione, ed il cui significato è che ogni numero presente nell'espressione è rappresentato in base *b*

Calcoli

- Si utilizzano degli algoritmi
- Esattamente quelli imparati alle elementari per la base 10
- Esempio: per sommare due numeri, si sommano le cifre a partire da destra e si utilizza il riporto

Esempi in base 10

$[\ 345\ ]_{10}$

$[\ 2 * 10 + 5 * 1\ ]_{10}$

Notazione posizionale

$$[\ a_{n-1}a_{n-2}a_{n-3}\ldots a_1a_0\ ]_b =$$

$$[a_0*1 + a_1*b + a_2*b^2 + a_3*b^3 + \ldots + a_{n-1}*b^{n-1}\ ]_b$$

$$= [\ \sum_{i=0,\,1,\,\ldots,\,n-1} a_i * b^i\ ]_b$$

Peso cifra i-esima

- Es: $b = 10$, $a_i \in \{0, 1, 2, \ldots, 9\}$

$$[\ 345\ ]_{10} = [3 * 10^2 + 4 * 10 + 5 * 1]_{10}$$

Notazione binaria

- Base 2, 2 cifre:
  - 0, 1
- La cifra nella posizione *i-esima* ha peso 2^i
- Esempi (*configurazioni di bit*):

$[0]_{10} = [0]_2$

$[1]_{10} = [1]_2$

$[2]_{10} = [10]_2 = [1*2 + 0*1]_{10}$

$[3]_{10} = [11]_2 = [1*2 + 1*1]_{10}$

Base 16 1/2

- Una base che risulta spesso molto conveniente è la base 16
- Perché ogni cifra in base sedici corrisponde ad una delle possibili combinazioni di 4 cifre in base 2
- Quindi, data la rappresentazione in base 2 di un numero naturale, la sua rappre-sentazione in base 16 si ottiene dividendo la sequenza in base in sotto-sequenze consecutive da 4 cifre ciascuna, partendo da destra, e convertendo ciascuna sotto-sequenza di quattro cifre binarie nella corrispondente cifra in base 16

Base 16 2/2

- Viceversa, data la rappresentazione in base 16 di un numero naturale, il corrispondente numero in base 2 si ottiene convertendo semplicemente ciascuna cifra della rappresentazione in base 16 nella corrispondente sequenza di 4 cifre in base 2

Notazione esadecimale

- Base 16, 16 cifre:
  - 0, 1, 2, …, 9, A, B, C, D, E, F
- Valore cifre in decimale:
  - 0, 1, 2, …, 9, 10, 11, 12, 13, 14, 15
- La cifra nella posizione *i-esima* ha peso 16^i
- Esempi:

$[0]_{10} = [0]_{16}$

$[10]_{10} = [A]_{16}$

$[18]_{10} = [12]_{16} = [1*16 + 2*1]_{10}$

Rappresentazione naturali

- In una cella di memoria o in una sequenza di celle di memoria si può memorizzare con facilità un numero naturale memorizzando la configurazione di bit corrispondente alla sua rappresentazione in base 2
- Questa è la tipica modalità con cui sono memorizzati i numeri naturali
- Coincide con gli esempi che abbiamo già visto in lezioni precedenti

Rappresentazione interi 1/2

- Come rappresentare però numeri con segno?
- Non esiste un elemento all'interno delle celle, che sia destinato a memorizzare il segno
- Come potremmo cavarcela?

Rappresentazione interi 2/2

- Un'idea sarebbe quella di utilizzare uno dei bit per il segno
  - 0 per i valori positivi
  - 1 per i valori negativi
- Il problema è che sprechiamo una configurazione di bit, perché avremmo due diverse rappresentazioni per il numero 0
  - Una col segno positivo
  - Una col segno negativo

Complemento a 2

- Per ovviare a questo problema, i numeri con segno sono tipicamente rappresentati in complemento a 2
- Se *n* è un numero minore di 0, allora, anziché memorizzare il numero originale *n*, si memorizza il risultato della somma algebrica
  $2^N + n$
  dove *N* è il numero di bit su cui si intende memorizzare il numero

Vantaggi

- C'è una sola rappresentazione per lo 0
- Gli algoritmi di calcolo delle operazioni di somma, sottrazione, moltiplicazione e divisione sono gli stessi dei numeri naturali rappresentati in base 2

Nota 1/2

- Senza entrare in ulteriori dettagli
  - una sequenza di *N* bit che rappresenta un numero negativo ha sempre il bit più a sinistra uguale ad 1
  - la rappresentazione in complemento a due di un numero positivo su *N* bit è uguale alla sua rappresentazione in base 2

Nota 2/2

- Quindi una configurazione di bit con il bit più a sinistra ad 1 rappresenta
  - un valore positivo se sta rappresentando un numero naturale in base 2
  - un valore negativo se sta rappresentando un numero in complemento a 2

Rappresentazione **int**

- Gli oggetti di tipo **int** sono tipicamente rappresentati in complemento a 2

Manipolatori

- Oggetti, che se passati allo *stream* di uscita, ne modificano il comportamento
- Possono essere *persistenti*: influenzano tutte le successive scritture

Modifica base uscita

- Manipolatore *hex*
  - Persistente, se passato all'oggetto cout, da quel momento in poi tutti i numeri saranno stampati in base 16
- Come tornare alla base 10?
  - Manipolatore *dec*

Stampa numeri in base 16

- Esercizio (*stampa_hex.cc*):
  - Scrivere un programma che legge da *stdin* un intero non negativo in notazione decimale e lo stampa in esadecimale
  - Se il numero inserito è negativo il programma non stampa nulla

Comando sleep

- Provate ad invocare il comando **`sleep 2`**
- Il comando aspetta 2 secondi e poi termina
- In generale, la sintassi (semplificata) è

  `sleep` <*numero_secondi*>

Invocazione comandi 1/2

- Si possono invocare i comandi anche da dentro un programma C/C++
- Basta scrivere
  `system("<riga_di_comando>");`
- Ad esempio
  `system("g++ ciao_mondo.cc");`
- Il programma rimane fermo finché il comando non è terminato

Invocazione comandi 2/2

- Per utilizzare la **`system`**, bisogna aggiungere la direttiva

```
#include <stdlib.h>
```

Esercizio

- Scrivere un programma che stampa l'elenco dei file contenuti nella cartella corrente

Soluzione

```
#include <stdlib.h>
main()
{
    system("ls") ;
}
```

Esercizio su while

- Mettendo assieme quanto detto nelle precedenti slide, svolgere il seguente esercizio
  - *stampa_secondi.cc*
- Idea per un cliclo infinito: esiste una espressione costante che vale sempre **`true`**?

Esperimento

- Provate a togliere l'istruzione di invocazione del comando *sleep*
- Come si fa per fermare il programma?

Cattiva soluzione :)

- Probabilmente avrete avuto voglia di agire come in:
  http://www.youtube.com/watch?v=hBhlQgvHmQ0
- Ma è meglio non utilizzare questo metodo se ci tenete al vostro computer (o al vostro lavoro) ...
- Vediamo come possiamo cavarcela in modo meno distruttivo

Come si termina ...

- ... un programma in esecuzione (***processo***)?
  - Ctrl + C
- In UNIX ci si basa sul concetto di *terminale*
- Anche da GUI, quello che si apre è un terminale (Terminal, Konsole, xterm, ...)
- In seguito a determinate combinazioni di caratteri il terminale spedisce speciali segnali ai processi

Segnale di interruzione

- Ctrl + C manda il segnale SIGINTR (interruzione) al processo in esecuzione
- A volte tale segnale può non bastare per interrompere il processo
- Possiamo agire in modi più efficaci

Chiusura del terminale

- Chiudere il terminale in cui sta girando un processo fuori controllo di norma causa anche la terminazione del processo
- In casi ancora più complicati, chiudere il terminale non basta, mentre la soluzione illustrata nelle seguenti slide funziona sempre

Altri dettagli sui processi

- Ad ogni processo è associato un identificatore numerico: *pid*
- Per elencare i propri processi:

ps x

- Per elencare tutti i processi:

ps ax

Uccisione di un processo

- Comando *kill*: spedisce segnali ai processi. Per uccidere un processo:

  `kill -9 <`*`pid`*`>`

- Invocato da un altro terminale
- **`kill -9`** funziona sempre

Miglioramenti e variazioni 1/2

- Come avrete notato, spesso il programma *stampa_secondi.cc* salta un secondo nella stampa
  - Non vedremo come correggere questo difetto
- Ci piacerebbe inoltre che il numero di secondi venisse scritto sempre sulla stessa riga, dando l'effetto di un orologio digitale

Miglioramenti e variazioni 2/2

- Inoltre vogliamo provare a far stampare solo quanti secondi sono trascorsi dalla partenza del programma
- Infine vorremmo che il programma si fermasse da solo quando è trascorso un numero di secondi deciso a tempo di scrittura del programma stesso

Esercizio

- Tutti queste caratteristiche, tranne la correzione del salto di più di un secondo, vanno implementate nel seguente esercizio:
  - *stampa_secondi_trascorsi.cc*

Esercizio su **for** e **while**

- Traccia e soluzione in *somma_e_max_1.cc*

Esercizi per casa

- Estendere l'esercizio *somma_e_max_1.cc* effettuando anche il controllo di *overflow* sul valore della somma
- Svolgere una variante di *somma_e_max_1.cc* in cui si calcola il prodotto tra gli elementi al posto della somma
  - Estendere anche questo esercizio controllando che non vi sia *overflow*

Fattoriale

- Traccia e soluzione in *fattoriale.cc*
- Sfida:
  - Quanto vale 11! ?

Risposta

- 39916800

Esercizio per casa

- Il calcolo del fattoriale può portare facilmente ad *overflow*, utilizzare la stessa soluzione adottata per il precedente esercizio per casa per controllare lo stato di *overflow* nel calcolo del fattoriale

Esercizi con cicli annidati

- Traccia e soluzione in *quadrato_pieno.cc*
- Traccia e soluzione in *quadrato_pieno_un_ciclo.cc*
- Per casa:
  - Traccia e soluzione in *quadrato_vuoto.cc*

Esercizi con *break* e *continue*

- Traccia e soluzione in *somma_e_max_2.cc*
- Per casa:
  - Traccia e soluzione in *somma_e_max_3.cc*

(Ri)Cominciamo bene 1/3

- Approfittiamo di nuovo del prossimo esercizio per applicare delle prime regole di buona programmazione
- **NON INSERIAMO NUMERI SENZA NOME NEL NOSTRO PROGRAMMA!**
  - Sono i cosiddetti **numeri magici**, perché appaiono magicamente in un programma
  - Chi legge il programma di norma non capisce da dove spuntano
- Al contrario, utilizziamo sempre costanti con nome
  - Il nome della costante fa capire a chi legge di cosa si tratta
  - Se dobbiamo cambiare un numero utilizzato più volte nel programma, basta cambiare valore alla costante una sola volta

(Ri)Cominciamo bene 2/3

- **USIAMO NOMI SIGNIFICATIVI PER LE VARIABILI**
  - L'idea è che leggendo il nome di una variabile si dovrebbe capire cosa contiene e qual è il suo obiettivo
  - Il compromesso di norma è tra la lunghezza e la chiarezza del nome
    - Se mettiamo troppe informazioni nel nome, quest'ultimo diventerà molto lungo, rendendo faticosa la scrittura/modifica del programma

(Ri)Cominciamo bene 3/3

- **NON REPLICHIAMO IL CODICE!**
  - Se in due punti del programma scriviamo due pezzi di codice (quasi) identici, cerchiamo il modo di ripensare la logica di quelle parti del programma in maniera tale da scrivere quel pezzo di codice una sola volta
    - Rendendolo magari un po' più generale per gestire in un sol colpo i due casi gestiti dai due pezzi di codice di partenza
  - Eliminare la replicazione non conviene solo nel caso in cui farlo comporterebbe complicazioni nel codice maggiori della replicazione stessa
- La replicazione rende il codice più lungo e spesso più difficile da capire, infine aumenta la probabilità di commettere errori e di dimenticare di correggerli in tutti i punti in cui le stesse istruzioni sono replicate

Esercizio con menu

- Traccia e soluzione in *catena_omogenea.cc*
  - Solo menu: *catena_omogenea_solo_menu.c*
- Questo esercizio è propedeutico per la prima mini-prova di programmazione di autovalutazione

Altri esercizi: stampa di figure

- Stampare un rettangolo (pieno e vuoto) di lati *m* ed *n*
  - Scrivere sia una soluzione con due cicli che una con un solo ciclo
- Stampare un triangolo (pieno e vuoto) di lato *n*
- Stampare un rombo (pieno e vuoto) di lato *n*

Compiti per casa

- Tracce e soluzioni nella cartella *Compiti per Casa*
- Fateli!

Mini-prova di programazione

- Traccia e soluzione in *catena.cc*
  - Solo traccia: *traccia_catena.txt*
- Assumendo di aver già svolto *catena_omogena.cc*, il tempo a disposizione per completare la prova è un'ora