Esercizi su

Notazione posizionale

Istruzioni iterative e di scelta

Varie

- Tracce extra
  - Sul sito del corso

Libreria matematica

- Se utilizzate la libreria matematica
  - Aggiungete *#include <math.h>* se usate il C
    - 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

Basi e cifre

- Rappresentazione numeri naturali
- *Base*: numero (naturale) di valori possibili per ciascuna cifra
- *Cifra*: simbolo rappresentante un numero
- 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

.

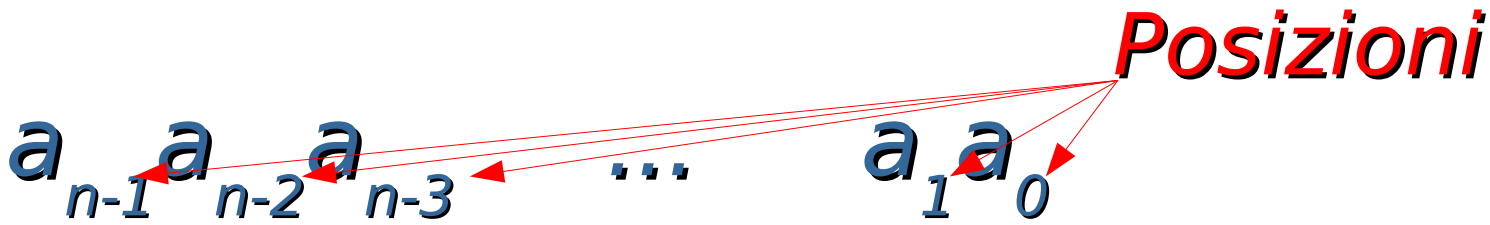
.

.

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

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*):

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

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

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

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

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:

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

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

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

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: www.youtube.com/watch?v=gL9slHxN6EM
- 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

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

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

Uccisione di un processo

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

  `kill -9 <pid>`

- Invocato da un altro terminale
- Se proprio non ci riuscite, chiudete il terminale stesso in cui sta girando il processo fuori controllo ...
- In casi ancora più complicati, chiudere una finestra non basterà, mentre la **`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_vuoto.cc*
- Traccia e soluzione in *quadrato_pieno_un_ciclo.cc*

Esercizi con *break* e *continue*

- Traccia e soluzione in *somma_e_max_2.cc*
- 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

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!

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

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