Esercizi su

Istruzioni iterative e di scelta

Imparare dagli esercizi

Notazione posizionale

Mini-prova di programmazione

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("ls ..");`**
- 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>
// non servono altre direttive!


main()
{
    system("ls") ;
}
```

Esercizio

- Scrivere un programma che stampa l'elenco dei file contenuti nella cartella corrente
- Dopodiché stampa
  `Ho finito`

Soluzione

```
#include <stdlib.h>
#include <iostream>
using namespace std ;

main()
{
    system("ls") ;
    cout<<"Ho finito"<<endl ;
}
```

Esercizio su while

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

Domanda

- E’ facile fermare il programma?
- Come mai?
- Cosa succede se si manda il segnale di interruzione mentre è in esecuzione il comando **`sleep`**?

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

- Per trovare il processo relativo al proprio programma cercare il processo il cui nome è uguale a quello dell'eseguibile

Uccisione di un processo

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

- Tutte 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

Infine ...

- **ANALIZZIAMO ATTENTAMENTE LE SPECIFICHE**
  - Anche le affermazioni più ovvie possono nascondere delle ambiguità

https://youtu.be/CeOsRggJO4s

- Da “Cattivissimo Me” (Despicable Me):

Gru: Clearly we need to set some rules.
Rule number one: You will not touch anything.

Margo: Aha. What about the floor?

Gru: Yes, you may touch the floor.

Margo: What about the air?

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

Osservazione e domanda

- La posizione degli anelli va controllata nelle funzionalità di inserimento ed eliminazione
- Perché la traccia me lo chiede
- Ma poi è necessario memorizzare la posizione di ciascun anello?

Risposta

- No
- Perché dal punto 3 capisco che l'unica cosa che viene stampata è un numero di F pari al numero di anelli
- La posizione in cui inserisco un anello non influisce su quello che verrà stampato

Ultima domanda

- Quindi qual è l'unica informazione che serve memorizzare?

Risposta

- Il numero di anelli nella catena

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!

Imparare dagli esercizi 1/3

- Guardare la soluzione solo quando veramente non si sa più come procedere

- **Non assumere** minimamente che dopo aver guardato la soluzione si sia in grado di risolvere lo stesso esercizio, o esercizi simili, in occasioni future

Imparare dagli esercizi 2/3

- Per tutti gli esercizi che non si è riusciti a risolvere da soli
  - Lasciare passare il tempo necessario per dimenticarne la soluzione
  - Tornare poi ad affrontare l'esercizio
  - Con lo stesso approccio utilizzato nel primo tentativo
  - Ripetere questo schema finché non si arriva a risolvere l'esercizio da soli

Imparare dagli esercizi 3/3

- Si è pronti per l'esame **solo se** si è riusciti a risolvere un'alta percentuale degli esercizi **da soli**

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
- Usando tutto il materiale didattico che ritenete opportuno

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

Compiti per casa su base 2

- Svolgere gli esercizi in *base2.txt*

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++*

Quanti mega ci sono ... ?

- https://youtu.be/n89WeAC_6_M

Abbreviazioni e prefissi

- b bit
- B byte
- K Kilo 1000 oppure 1024
- M Mega 1000*K oppure 1024*K
- G Giga 1000*M oppure 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