Informazioni Generali Esercitazioni di Informatica

Gerardo Pelosi

e-mail(s): gerardo.pelosi@unibg.it,
gerardo.pelosi@polimi.it
phone: (+39) 02 2399 3476
web: http://home.deib.polimi.it/pelosi

Dip. di Elettronica, Informazione e Bioingegneria (DEIB),
Politecnico di Milano,
via G. Ponzio 34/5, 20133 Milano
Ed. 18, Ufficio n°127

Sito del Corso

www.unibg.it
> Didattica
> Indice insegnamenti
> Insegnamenti attivi
> 22010 - Informatica (modulo di programmazione + basi di dati)(Gestionale) (12 crediti)

Materiale didattico

Online (sul sito del corso)

- Slide delle esercitazioni dell'intero corso
- Esercizi aggiuntivi (senza soluzioni)
- Comunicazioni, date esami

Libri consigliati:

V. Moriggia, G. Psaila: "*Concetti fondamentali di informatica*" ed. Esculapio (Bologna: Progetto Leonardo), 2007

G. Psaila, "*Esercizi ragionati in C/C++*" ed. Esculapio (Bologna: Progetto Leonardo), 2001

P. Cremonesi, G. Psaila, "*Introduzione ragionata al C/C++*" ed. Esculapio (Bologna: Progetto Leonardo), 2000

Altri testi, eserciziari e manuali vari di C:

- B. Kernighan, D. Ritchie: "Linguaggio C [ANSI C]", ed. Jackson Libri [il testo di chi il C lo ha inventato – insegna tutto sul C, ma non insegna a programmare !]

- Harvey M. Deitel, Paul J. Deitel, "C++ Fondamenti di programmazione", ed. Apogeo. [testo didattico di C++ molto completo, ma non insegna a programmare !]

- Altri testi... vanno bene tutti, e nessuno è perfetto !

Ricevimento studenti

NON c'è orario di ricevimento per le esercitazioni
MA il modo migliore (e più sicuro) per essere ricevuti e ascoltati è fissare un appuntamento (via e-mail)

- Utilizzare la mail dell'università *XX@studenti.unibg.it* (da tenere sempre "sott'occhio", *per evitare il riempimento della casella di posta !*)
- Preferibilmente, non utilizzare indirizzi e-mail come: *masterOfWarCraft@isp.com* (*non è professionale !*)
- Scrivere in modo sintetico ma chiaro
  - Identificandosi precisamente [*nome, matricola, corso di laurea*]
  - Spiegare brevemente lo scopo dell'incontro: chiarimenti didattici su..., problemi amministrativi per...
  - Dare all'e-mail un **Oggetto** sintetico ma preciso, che inizi preferibilmente come **INFORMATICA ….** Es: "**INFORMATICA MODULO Programmazione Richiesta colloquio per chiarimenti (Rossi, Matricola: 1001111)**"
- Se i quesiti ammettono risposta via e-mail... ... l'interazione più efficace è proprio via e-mail !

  … (*non allegare codice con richieste del tipo: "non compila,... dà un risultato non corretto e non capisco perché …*")
- il Mercoledì, (durante e) al termine delle esercitazioni, è un ottimo momento per fare domande !

Organizzazione

Le esercitazioni si svolgeranno nel modo seguente:

1. Sul sito del corso troverete le tracce degli esercizi commentati e risolti in aula.

2. Degli esercizi proposti soltanto alcuni verranno risolti subito alla lavagna per ricapitolare i concetti su cui si focalizzerà l'esercitazione oppure per introdurre qualche caratteristica del linguaggio.

3. Almeno un esercizio, tipicamente due, saranno svolti in autonomia da ciascuno studente, o coppia di studenti, utilizzando il compilatore in dotazione all'aula. Il tempo previsto per ciascun esercizio è stimato in circa 20-25 minuti.

4. Le soluzioni degli esercizi saranno disponibili solo al termine dell'esercitazione sul sito del corso.

   Si raccomanda vivamente di NON rimandare l'occasione di fare pratica ai soli momenti di studio individuale ma di usare la sessione di laboratorio per chiarire i propri dubbi !

   Sulla base delle statistiche fatte negli anni precedenti si raccomanda anche a coloro che ritengono di avere una certa padronanza della programmazione di NON sottovalutare il corso.

5. Ogni 2 settimane (circa), come materiale complementare, verranno pubblicate tracce di problemi (Homeworks) senza soluzione, che si invita a risolvere e commentare assieme al docente.

Linguaggi di programmazione

Linguaggi di programmazione (formali, per la codifica)

Consentono di scrivere gli algoritmi sotto forma di programmi eseguibili dal calcolatore

- Linguaggi macchina: ogni loro istruzione (vocabolo) è composta come una sequenza cifre binarie direttamente decodificabile dalla specifica macchina (CPU) per essere attuata.
- Linguaggi di alto livello: linguisticamente più vicini al linguaggio naturale
  (1 istruzione equivale a 2+, o anche 10+, istruzioni in linguaggio macchina)
- Linguaggi *assembler*: linguisticamente più vicini alle istruzioni eseguite direttamente dalla macchina (CPU)
  (1 istruzione *assembler* equivale a 1 istruzione in linguaggio macchina)

La “Babele” dei linguaggi

– Nello sviluppo di software i linguaggi *assembler* soffrono di problemi di compatibilità fra diverse tipologie di architetture hardware dei calcolatori

– I linguaggi *assembler* danno l'opportunità di maggior controllo sulle caratteristiche della macchina e di maggior efficienza nell'esecuzione del software

– Inizialmente il software era scritto direttamente nel linguaggio della macchina, ma già nella seconda metà degli anni '50 si iniziò a realizzare programmi che traducevano automaticamente programmi scritti nei linguaggi di più alto livello nel linguaggio di “basso” livello di macchine diverse (... *compiler* + *linker*)

Componenti di un linguaggio

- ***Vocabolario****:* parole chiave del linguaggio
  - riconosciute dal ***parser*** (**analizzatore lessicale**)
- ***Sintassi****:* regole per comporre i simboli del vocabolario
  - Il controllo della sintassi avviene tramite l'**analizzatore sintattico**
- ***Semantica***: significato delle espressioni
  - Il controllo della semantica è il più difficile
  - Un errore semantico può essere rilevato solo dall'utilizzatore del software o dal programmatore, in genere, solo a tempo di esecuzione

Alcuni linguaggi

- I primi e tradizionali linguaggi
  - Fortran, Cobol, Ada, ...
- Linguaggi che non "mimano" l'architettura della macchina
  - Lisp
- Linguaggi speciali
  - SQL: per interrogazione di database, …
- Linguaggi moderni
  - Basic, Pascal, Modula2, C, C++, Java, C#, Python, Ruby …
  - linguaggi di *scripting* o orientati al Web: PHP, Perl, Tcl/Tk, JavaScript

Compilatori e Interpreti

- I ***compilatori*** sono *software* che **traducono** i programmi scritti in un linguaggio d'alto livello in **codice macchina**
  - il programma una volta che è stato *interamente* tradotto viene eseguito dal calcolatore
  - il maggior vantaggio della compilazione è senz'altro l'efficienza in termini di prestazioni,
  - il codice tradotto in linguaggio macchina è valido solo per una piattaforma specifica (*combinazione di architettura hardware e sistema operativo*)

- Gli ***interpreti*** sono programmi che **traducono ed eseguono** ciascuna istruzione del programma scritto in un linguaggio d'alto livello in modo sequenziale.
  - un programma scritto in un linguaggio interpretato non ha, in linea di massima, dipendenze dalla specifica piattaforma su cui viene eseguito
  - ma è più lento e richiede più memoria in fase di esecuzione

Esempi di linguaggi *interpretati*

– Lisp, Prolog (usati nell'intelligenza artificiale)
– Basic, PHP, LaTeX, JavaScript (ad eccezione del motore Javascript V8 in Google-Chrome)

Esempi di linguaggi *compilati*

– Cobol, C, C++, Pascal, Fortran

Bytecode

Una soluzione intermedia fra compilazione e interpretazione è stata introdotta

- con il concetto di **bytecode** nelle prime versioni del linguaggio *Pascal* e successivamente adottata nei linguaggi *Java* e *Python*, *Visual Basic (VB)* e *.NET* di Microsoft.

In tutti questi casi il codice sorgente dei programmi non viene compilato in linguaggio macchina, ma viene tradotto in un **codice intermedio** "ibrido" destinato a venire interpretato al momento dell'esecuzione del programma.

Il motivo di questo doppio passaggio è di

- avere la portabilità dei linguaggi interpretati
- grazie alla pre-compilazione, un interprete più semplice e quindi più veloce.

Il codice intermedio è più facile sia da interpretare che da compilare: per questo motivo sia per *Java, Python* che per i linguaggi *.NET* (es. C#) sono stati sviluppati i

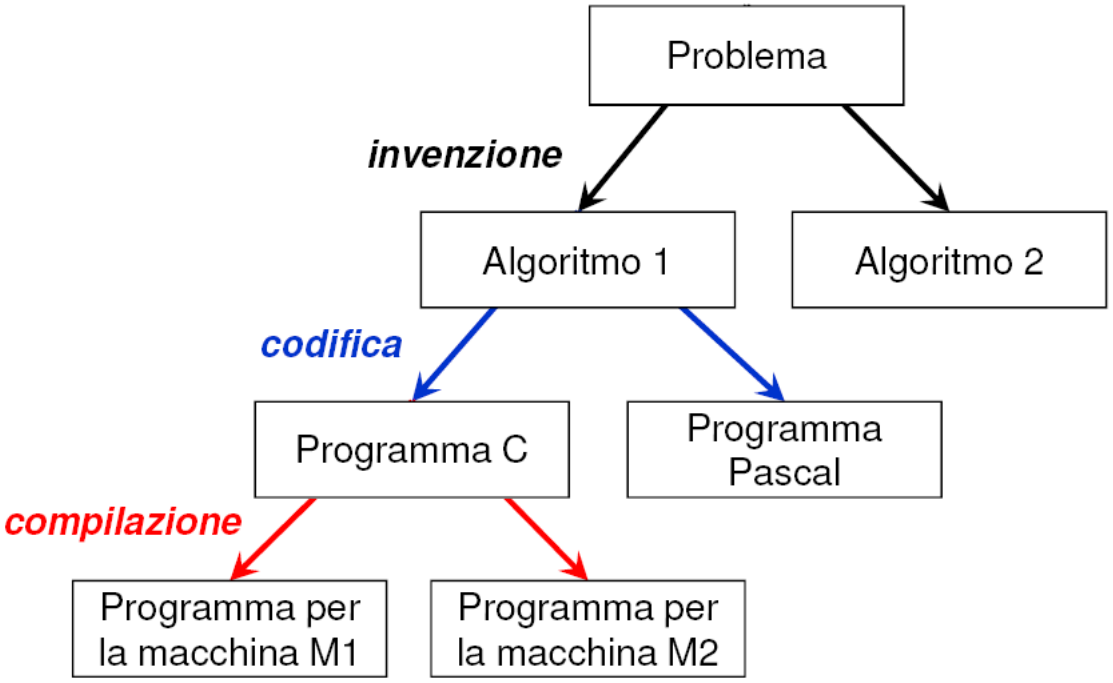
compilatori JIT (Just In Time)

- che al momento del lancio di un programma *Java, VB* o *.NET* **compilano al volo il codice intermedio e mandano in esecuzione un normale codice macchina nativo**, eliminando completamente la necessità dell'interprete e rendendo i programmi scritti in questi linguaggi veloci quasi quanto i normali programmi compilati

Problemi, Algoritmi, Programmi

Compito dell'informatico è inventare (*creare*) **algoritmi** cioè

1. escogitare e formalizzare le sequenze di passi che risolvono un problema
2. e codificarli, con un linguaggio di programmazione, in **programmi**



La catena di programmazione (linguaggi ***compilati***)

Si parte dalla codifica di un algoritmo in un linguaggio di programmazione scelto dallo sviluppatore che prende il nome di **programma sorgente.**
Si tratta di un documento elettronico (*file*) contenente caratteri stampabili che rappresentano parole di un linguaggio che può essere sia

- di basso livello: *Assembly*
- sia di alto livello: C, C++, Java, …

Tramite opportuni strumenti si elabora il programma sorgente tramite due passaggi (in stretta sequenza)

1. Compilazione (*Compiling*)
2. Collegamento (*Linking*)

Si genera un altro documento elettronico (*file*) contenente anche caratteri NON stampabili con una precisa codifica binaria che verranno acquisiti dalla macchina preposta all'esecuzione.
Tale documento si dice essere scritto in *linguaggio macchina* ed è chiamato **programma eseguibile.**

1. Videoscrittura (*editing*)

- Il testo del programma sorgente, costituito da una sequenza di caratteri, viene **composto e modificato** usando uno specifico programma: l'***editor***
- Così otteniamo un ***File*** (documento testuale) con il **Programma Sorgente** memorizzato in memoria di massa con nome:
  - **xxxx.asm** *per programmi in assembly*
  - **xxxx.c** *per programmi in C*
  - **xxxx.cpp** *per programmi in C++*

2. Traduzione (*compiling*)

Il testo del *Programma sorgente* (es. xxxx.cpp), costituito da una sequenza di caratteri, viene tradotto usando uno specifico programma: il **compilatore**, in un *Programma in codice macchina o* ***Programma Oggetto***
(file: xxxx.obj) costituito da una sequenza di byte che è molto vicino a quanto direttamente acquisibile dal calcolatore.

Il *compilatore* è specifico per ciascun tipo di *architettura hardware e sistema operativo* del calcolatore dove il Programma sarà eseguito.

• Durante questa fase si riconoscono i simboli, le parole e i costrutti del linguaggio utilizzato nel programma sorgente
  - eventuali messaggi diagnostici segnalano **errori lessicali e sintattici** nel programma sorgente e **alcuni errori semantici** (es., *type checking*,oppure l' uso di un simbolo prima della sua definizione)

3. Collegamento (*linking*)

- Il programma *collegatore* (***linker***) deve collegare fra loro differenti *programmi oggetto*
- I compilatori vengono forniti con ***librerie*** (cioè collezioni di *file oggetto*) che consentono di utilizzare sotto-programmi che eseguono un insieme di operazioni comuni a quasi tutti i software che è possibile scrivere con il linguaggio scelto
- Si genera un **Programma Eseguibile** (XXX.exe), un file che contiene sequenza di byte che è direttamente acquisibile dal calcolatore
  - Messaggi di errore possono essere dovuti ad errori nel citare i nomi delle funzioni da collegare

4. Caricamento (*loading*)

- Il programma caricatore (***loader)*** individua una porzione **della memoria centrale del calcolatore** dove copiare il contenuto del file **xxxx.exe** (che si trova sulla memoria di massa)
  - Eventuali messaggi rivolti all'utente possono segnalare che non c'è abbastanza spazio in memoria

I programmi: Editor, Compilatore, Linker, Loader sono eseguiti dal calcolatore tramite comandi imposti dal programmatore al Sistema Operativo (SO)

Il SO è a sua volta un programma eseguibile che ha il compito di attuare i comandi dell'utilizzatore del calcolatore utilizzando l'hardware a disposizione dello stesso (il SO è mandato in esecuzione dal *BootLoader* che è a sua volta avviato dal *BIOS* che è impostato dal costruttore della "scheda madre" del calcolatore e si auto-avvia all'accensione)

5. Esecuzione

Per eseguire il programma occorre fornire in ingresso i dati richiesti e in uscita riceveremo i risultati (su video o file o stampante)

– Durante l'esecuzione possono verificarsi degli errori (detti "*errori di run-time*"), quali:
  - calcoli con risultati scorretti (per esempio un *overflow!*)
  - calcoli impossibili (divisioni per zero, logaritmo di un numero negativo, radice quadrata di un numero negativo,….)
  - errori nella concezione dell'algoritmo (l'algoritmo non risolve il problema dato)

Quelli citati sono tutti esempi di ***errori semantici***

Quindoi anche nel caso dei linguaggi C/C++ le fasi sono:

1. Videoscrittura
   produzione del programma, svolta dal programmatore tramite un programma di videoscrittura (*editor*)
2. Traduzione (compilazione)
   svolta dal *compilatore* (*compiler*)
3. Collegamento (linking)
   svolto dal *collegatore* (*linker*)
4. Caricamento (loading)
   svolto dal *caricatore* (*loader*)