Corso Programmazione 2009-2010

(docente)
Fabio Aiolli
E-mail: aiolli@math.unipd.it
Web: www.math.unipd.it/~aiolli

(docenti laboratorio)
A. Burattin, E. Caniato, A. Ceccato

Dipartimento di Matematica Pura ed Applicata
Torre Archimede, Via Trieste 63

Orario delle lezioni e esercitazioni

~36 ore di lezioni in aula P200

- Martedi'
  - Ore 11:30 - 13:15
- Venerdi'
  - Ore 10:30 – 12:15

~32 ore di esercitazioni in laboratorio

- Proposte.. Martedi', Mercoledi', ore 14:00 – 17:00
- Aula Informatica C dell' ex Dipartimento di Matematica Pura ed Applicata (Paolotti)
- Prima esercitazione in laboratorio il 20/04/09

Risorse per il corso

A. Kelley, I. Pohl. "C Didattica e programmazione", IV edizione, Pearson, 2004.

Altri libri consigliati, slide ed esercizi saranno disponibili sul sito web del corso:

http://www.math.unipd.it/~aiolli/corsi/0910/prgxmat/prg.html

Il GOOGLE GROUP del corso:

http://groups.google.it/group/prxmat10atunipd

Il GOOGLE GROUP del corso:

http://groups.google.it/group/prxmat10atunipd

ISTRUZIONI x GLI STUDENTI

- Nickname: <Nome><InizialeCognome> (per esempio, FabioA)

- Inserire indirizzo di posta elettronica

Esame Scritto

Prima Parte

- Domande riguardanti la sintassi del linguaggio C e semplici programmi

Seconda Parte

- Analisi e implementazione di algoritmi in C

Colloquio (per esame da 8 crediti)

- Contenuti di Introduzione alla Programmazione

Contenuti del corso

- Panoramica sul C
- Strutture dati ed algoritmi
- Programmi x il calcolo scientifico

Molto Importante

Imparare a programmare NON e' facile e NON si impara sui libri ma necessita di MOLTA pratica (e curiosita')

RENDETEMI IL COMPITO + FACILE

Qualsiasi tipo di interazione con lo studente e' ben accetta da parte mia

Richiedete spesso chiarimenti e partecipate attivamente alle lezioni e ai laboratori

Iniziamo..

PARTE 1

Definizioni Fondamentali

Algoritmo

DEFINIZIONE

- Insieme completo delle regole che permettono la soluzione di un determinato problema

DEFINIZIONE OPERATIVA

- Procedura effettiva che indica le istruzioni (passi) da eseguire per ottenere i risultati voluti a partire dai dati di cui si dispone

Algoritmo: Esempi

Esempio Culinario:

- Ricetta x cucinare gli spaghetti

Esempi sui numeri:

- Insieme di passi per verificare se un numero è dispari, pari, primo, ecc.

Altri esempi: MCD(a,b), mcm(a,b)

- Per esempio l'algoritmo di Euclide per il calcolo del MCD (che può essere usato anche per il calcolo del mcm!)

Algoritmo: Caratteristiche

- Esprimibile con un numero finito di istruzioni
- Istruzioni eseguibili da un elaboratore
- Insieme di istruzioni di cardinalità finita
- Tempo di esecuzione di ogni istruzione finito
- Elaboratore ha una memoria
- Calcolo per passi discreti
- Non esiste limite alla lunghezza dei dati di ingresso
- Non c'e' un limite alla memoria disponibile
- Numero di passi esecuzione eventualmente illimitato

Linguaggi

LINGUAGGI NON FORMALI (ambigui)

- Linguaggio Naturale
- Linguaggio della musica
- Linguaggio del corpo
- Ecc.

LINGUAGGI FORMALI (di programmazione)

- Linguaggi NON ambigui, regolati da regole grammaticali precise
- Possono essere classificati in base al loro livello di astrazione

Linguaggi di Alto Livello (LAL)

ESEMPI FAMOSI

- Imperativi: PASCAL, FORTRAN,COBOL, C
- Ad oggetti: CPP, JAVA

Appositi software (compilatori) si occupano di tradurre le istruzioni scritte in questi linguaggi (cosiddetto codice sorgente, un file di testo), nell'equivalente codice eseguibile dalla macchina (cosiddetto codice eseguibile, binario)

Caratteristiche LAL strutturati

SEQUENZA

- Le istruzioni vengono eseguite in sequenza nell'ordine in cui compaiono in un *blocco* di istruzioni

SELEZIONE

- Strutture di controllo decisionali:
  - P.e. Se <espressione> esegui <BloccoIstruzioniV> altrimenti esegui <BloccoIstruzioniF>

ITERAZIONE

- Strutture di controllo iterative:
  - P.e. Fintanto che <espressione> esegui <BloccoIstruzioni>
  - Oppure, Esegui <BloccoIstruzioni> fintanto che <espressione>

Complessita' degli Algoritmi

- Complessita' Polinomial (P) : Il numero di passi e' proporzionale in modo polinomiale alla cardinalita' dell'input

- Complessita' Non-Deterministic Polinomial (NP) : Sono noti algoritmi che terminano in un numero di passi polinomiale usando un numero indeterminato di macchine in parallelo, oppure utilizzando l'algoritmo di Gastone

Algoritmi di Ricerca

- Ricerca MIN e MAX
- Ricerca di un valore in una collezione
- Ricerca di un valore in una collezione ordinata
- Ricerca degli zeri di una funzione

Algoritmi di Ordinamento

- Ordinamento degli elementi in una generica collezione
- Fondere due collezioni ognuna di esse gia' ordinata
- Ordinare senza usare confronti

Algoritmi di Ottimizzazione

- Problema del Commesso Viaggiatore (TSP)
- Problema dei Cammini Minimi (SP)

PARTE 2

Il processo di programmazione

Problemi e algoritmi

- La **programmazione** e' la disciplina che si occupa della risoluzione di problemi tramite programmi scritti in qualche **linguaggio di programmazione**
- Il **problema** e' una definizione sintetica di cio' che il programma deve realizzare dal punto di vista dell'utente
- La **soluzione** fornisce un'idea dell'approccio che deve essere seguito per risolvere il problema, tenendo anche in considerazione quale software gia' esistente potrebbe essere riutilizzato
- L'**algoritmo** e' una descrizione precisa e non ambigua di una sequenza di passi da seguire per risolvere un problema. L'algoritmo e' quindi un raffinamento della soluzione. Gli algoritmi possono essere espressi in molti modi
- Il **programma** e' una (possibile) descrizione dell'algoritmo espressa usando un particolare linguaggio di programmazione. Si intende che il programma debba essere completo ed eseguibile
- Per i banali problemi che vedremo, useremo direttamente il linguaggio C per descrivere gli algoritmi. Quindi, per noi, **algoritmi=programmi**

Il processo di programmazione

- Il **processo di programmazione** comprende tutte le attivita' necessarie per sviluppare dei programmi in modo che siano memorizzati e preparati per l'esecuzione
- Per sviluppare programmi su un computer e' necessario un ambiente di sviluppo che almeno comprenda, oltre al sistema operativo, un **editor** ed un **compilatore**

Editor e compilatori

- Un **programma** e' un testo che consiste in una sequenza di istruzioni scritte in un particolare linguaggio di programmazione. Tale "testo" viene creato e salvato come un file su disco tramite un **editor**.
- Una volta che e' stato scritto e memorizzato, il (testo del) programma viene dato in input al **compilatore**
- Il compilatore ha una duplice funzione:
  - Controlla la **validita' del programma**: segnala gli **errori di sintassi**. Questi errori devono essere corretti tramite l'editor ed il programma corretto deve quindi essere nuovamente compilato
  - Se il programma non contiene errori, il compilatore **traduce il programma** in linguaggio macchina

Esecuzione dei programmi

- Il compilatore naturalmente non e' in grado di scoprire gli **errori logici o di esecuzione** (i ben noti **bug**) del programma: in tali casi, il programma risulta essere sintatticamente corretto ma non si comporta come ci si aspetta
- Il programma viene **eseguito** (o "**fatto girare**") attraverso un comando al sistema operativo che lo carica in memoria e quindi diventa un processo in esecuzione
- Gli errori logici possono essere prevenuti seguendo delle **buone regole di programmazione**
- Spesso gli errori logici si individuano mandando in esecuzione il programma ed osservandone il comportamento tramite degli **insiemi di test affidabili**
- Una volta rilevato e corretto un errore logico, si dovrebbero nuovamente eseguire i test sul suo comportamento, in quanto le presunte "correzioni" potrebbero aver introdotto nuovi errori
- Il processo di rilevazione e correzione degli errori logici di un programma viene chiamato **debugging**

HelloWorld in C

Nome file: helloword.c

```
#include<stdio.h>

main() {
  printf("Hello World!\n");
}
```

gcc helloword.c -o helloword.exe

Token

I token sono unita' sintattiche di base del C

- Parole chiave
- Identificatori
- Costanti
- Costanti stringa
- Operatori
- Simboli di interpunzione

Parole chiave (Keyword) del C

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

La libreria standard

Input e Output	<stdio.h>
Test dei caratteri	<ctype.h>
Funzioni su stringhe	<string.h>
Funzioni Matematiche	<math.h>
Funzioni Utilita'	<stdlib.h>
Funzioni Diagnostiche	<assert.h>
Liste argiomenti variabile	<stdarg.h>
Salti non locali	<setjmp.h>
Segnali	<signal.h>
Funzioni Date e Ore	<time.h>
Limiti dei tipi di variabile	<limits.h>

Tipi di Dato

Un singolo byte	char
Numero Intero	int
Numero Intero corto	short
Numero Intero lungo	long int
Numero in virgola mobile (singola precisione)	float
Numero in virgola mobile (doppia precisione)	double
Numero in virgola mobile (precisione estesa)	long double
Interi con segno	signed {char/int}
Interi senza segno	unsigned {char/int}

Sequenze di Escape

\a bell	\\ backslash
\b backspace	\? punto interrogativo
\f formfeed	\' apice singolo
\n a capo	\" apice doppio
\r ritorno carrello	\ooo carattere ottale
\t tabulazione orizzontale	\xhh carattere esadecimale
\v tabulazione verticale	

Dichiarazioni di variabili

tipo nome_della_variabile [= inizializzazione];

Esempi:

- int a;
- int a = 2;
- int a, b;
- float f = 2.345;
- char c = 'X';
- char messaggio[] = "Questa e' una stringa";

Facendo precedere il tipo dal qualificatore **const**, si ottiene una "variabile costante", il cui valore non potrà essere cambiato

const double pi = 3,141592653;

Operatori

Operatori

- Aritmetici: +, -, *, /, %
- Relazionali: >, >=, <, <=, ==, !=
- Logici: &&, ||, !
- Incremento e decremento: ++, --
- bitwise: &, |, ^, <<, >>, ~
- Assegnazione: =, +=, -=, *=, ecc.
- Condizionale ternario: expr1 ? expr2 : expr

Conversioni di Tipo

Quando un'espressione coinvolge valori di tipo diverso, essi vengono convertiti ad un tipo comune:

- automaticamente ad un tipo "più grande", che quindi non fa perdere informazione
- esplicitamente, mediante un operazione di cast. Obbligatoria se non e' possibile una conversione implicita automatica.

If (else)

```
if (condizione) {
   /* blocco eseguito se la
   condizione e' vera */
} else {
   /* blocco eseguito se la
   condizione e' falsa */
}
```


Operatori Logici

Expr1	Expr2	Expr1 && Expr2	Expr1 || Expr2	! Expr1
Zero	Zero	0	0	1
Zero	Non Zero	0	1	1
Non Zero	Zero	0	1	0
Non Zero	Non Zero	1	1	0

Valutazione short-circuit

La valutazione degli operatori logici && e || avviene in maniera short-circuit, cioè da sinistra verso destra e si interrompe non appena il risultato diventa noto.

(a != 0) && (++i < 10)
(a!=0) || (++i < 10)

Switch Case

```
switch (espressione) {
  case espr-costante1: istruzioni1
  case espr-costante2: istruzioni2
  case espr-costante3: istruzioni3
  case espr-costante4: istruzioni4
  ...
  default: istruzioni_default
}
```