Laboratorio di Algoritmi e Strutture Dati

Camillo Fiorentini

Anno accademico 2007/08

Testi consigliati

- *C Didattica e Programmazione, 4ª edizione* (A book on C, 4^{th} edition),
  Al Kelley, Ira Pohl. Addison-Wesley, Italia, 2004.

oppure

- *Il linguaggio C. Principi di programmazione e manuale di riferimento. Seconda edizione*.
  B.W. Kernighan, D.M. Ritchie. Pearson Prentice Hall 2004.

Per entrambi i testi vanno bene anche le edizioni precedenti, oppure altri manuali del C (purché si riferiscano all'ANSI C)

Compilatore

- Compilatore di riferimento: `gcc`
  Per Windows:
  `http://www.cs.colorado.edu/~main/cs1300`

- Per ulteriori informazioni consultare la pagina del corso
  `http://homes.dsi.unimi.it/~fiorenti/labalg07.html`
  da cui è anche possibile scaricare il materiale visto a lezione, che **non** sostituisce ma integra i testi consigliati.

Esame

Svolgimento **individuale** di un progetto da svolgere in C.
Essendo un corso di *laboratorio* di Algoritmi e Strutture Dati (e non un semplice corso di programmazione in C) è richiesto l'utilizzo di strutture dati e algoritmi che implementino in modo eficciente le operazioni richieste.
Per questo, prima di svolgere il progetto, è *necessaria* una piena conoscenza degli argomenti svolti nel corso di Algoritmi.

Prerequisiti per il corso

- Programmazione in Java.
- Nozioni fondamentali di Architettura degli elaboratori (rappresentazione binaria dell'informazione, linguaggio macchina e linguaggio assemblativo, la catena di programmazione: compilatore, linker e loader).

Analisi della complessità

Permette il confronto di algoritmi.
Esempio
Supponiamo di voler ordinare n interi generati casualmente e memorizzati in un array. Consideriamo tre algoritmi e la relativa complessità del tempo di esecuzione.

- Quicksort
  Tempo medio: $O(n.\log n)$
- Quicksort randomizzato
  Tempo medio: $O(n.\log n)$
- Insertion sort
  Tempo medio: $O(n^2)$

Gli algoritmi più efficienti sono i primi due.

Confronto sperimentale delle prestazioni

Nella tabella vengono riportati i tempi di esecuzione in secondi dei tre algoritmi in funzione del numero n di elementi da ordinare.

n	**10.000**	**20.000**	**30.000**	**40.000**	**50.000**	**60.000**
Quick.	0,01	0,01	0,02	0,02	0,03	0,04
Q. rand.	0,01	0,01	0,02	0,02	0,03	0,04
Ins. sort	0,23	0,83	1,84	3,25	5,04	7,24

Utilità dell'analisi della complessità

- L'esperimento conferma che l'insertion sort è un pessimo algoritmo rispetto agli altri due (e questo *indipendentemente* dal linguaggio di programmazione usato per implementare l'algoritmo).
- La teoria degli algoritmi permette di conoscere la complessità di un algoritmo, ossia il tempo di esecuzione e le risorse di memoria impiegate in funzione della dimensione dei dati di input.
- Per scrivere programmi efficienti, prima di scrivere il codice occorre valutare la complessità degli algoritmi che risolvono il problema assegnato e scegliere quello più efficiente.

Il C: cenni storici

- 1972: Prima implementazione del linguaggio C su elaboratore PDP11 da parte di D. M. Ritchie presso i laboratori AT&T Bell.
  È introdotto per poter riscrivere in un linguaggio di alto livello il codice del sistema operativo UNIX.
  È un'evoluzione del B e del BCPL in cui sono aggiunti i tipi e le strutture di controllo.
- Anni '80: Sviluppo del "C tradizionale".
- 1983: Inizia la definizione dello standard "ANSI C" da parte dell'American National Standards Institute.
- 1990: L'International Standardization Organization (ISO) approva l'ANSI C (o "ANSI/ISO C").
  È a questo standard che fa riferimento il corso e i libri consigliati (e quindi anche il progetto dell'esame).

Caratteristiche del linguaggio

- Linguaggio imperativo con numerosi tipi di dato e strutture di controllo
- Caratteristiche di basso livello (gestione delle memoria, puntatori, operazioni bit a bit, ...)
- Codice efficiente e compatto
- Ricche librerie (standard) per operazioni non definite nel linguaggio (lettura, scrittura, allocazione dinamica della memoria, . . .)
- Permette lo sviluppo di un programma su più file compilabili separatamente (utile per lo sviluppo di grossi progetti)

C e Java

La *sintassi* è simile.

Ad esempio

```
int i,x,y,z;
for(i=0 ; i<=100 ; i++){
  if(x>0 && (y< 5 || y > 10))
      x = y*6;
  else
      x = y + 2*y;
  y = z++;
}
```

si comporta come in Java.

Tuttavia:

- In C non si può fare

  ```
  for(int i=0 ; i<=100 ; i++){
    ...
  }
  ```

  in quanto con l'ANSI C le variabili possono essere dichiarate solo all'inizio di una funzione.
- Alcune parole hanno nei due linguaggi significati diversi (es., la parola chiave `static`).

Differenze

È richiesto uno *stile* di programmazione completamente diverso.

- Il C è orientato alla programmazione strutturata (top-down): per risolvere un problema, lo si scompone in sottoproblemi, ognuno dei quali può essere risolto in modo indipendente (raffinamento per passi successivi).
- Il linguaggio non definisce meccanismi per la gestione di *oggetti*.
- Un programma è un insieme di funzioni. Una funzione può essere vista come un programma che, dati dei valori in ingresso, compie delle operazioni e restituisce un risultato.

  Nota: Le funzioni in C non vanno confuse con i metodi Java (servizi forniti da oggetti). Sono analoghe ai *metodi statici*.

- Non fornisce una visione astratta della macchina sottostante, ma occorre preoccuparsi dei dettagli di basso livello (gestione della memoria, controllo valori iniziali delle variabili, controllo sugli indici degli array, puntatori,...).
- Non ci sono meccanismi per la gestione delle eccezioni.

Esempio di programma

```
/* file prog1.c */


  #include <stdio.h> /*  linea trattata
                        dal preprocessore */

   int main(void){
     printf(``Questo e' il mio primo programma\n'');
     return 0;
   }
```

Per poter eseguire il programma occorre compilare il file `prog1.c`.
Il programma stampa sullo schermo

```
Questo e' il mio primo programma
```


Struttura del programma

Un programma C è costituito da un insieme di definizioni di funzioni; la prima ad essere eseguita è la funzione `main()`.
Nell'esempio, `main()` è l'unica funzione definita.

```
/* definizione della funzione main() */

 int main(void){ /* inizio definizione  */
    printf(``Questo e' il mio primo programma\n'');
    return 0;
 } /* termine definizione */
```

Generalmente la funzione `main()` non ha argomenti (si usa il tipo nullo `void`) ed è di tipo `int` (intero).

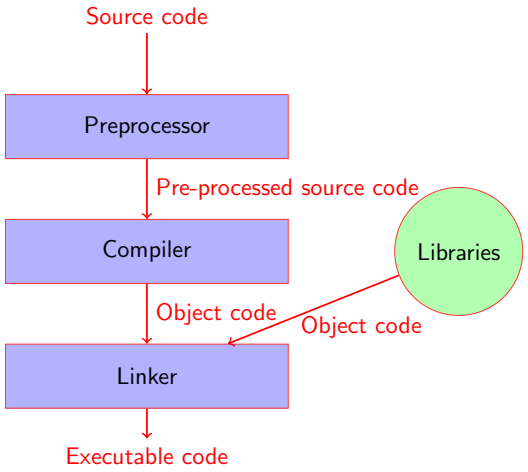
Nota
Seguendo la convenzione del libro, il nome di una funzione è seguito dalla coppia di parentesi '`()`'. Quindi, le notazioni '`printf()`', '`main()`' indicano che '`printf`' e '`main`' sono nomi di funzione.

- La funzione `printf()` serve per la stampa (*formatted print*).
- Le *istruzioni* devono terminare con `;`.
- L'istruzione
  `printf("Questo e' il mio primo programma\n");`
  stampa sullo schermo la stringa fra apici; i due caratteri `\n` rappresentano il carattere di *newline* e forzano il cursore a iniziare una nuova linea.
- La seconda istruzione
  `return 0;`
  restituisce 0 al sistema operativo (non è necessario farlo, ma è il modo standard di terminare un programma).

Nota
In ambiente Linux, il valore restituito da un programma è assegnato alla variabile '?' ('`echo $?`' stampa il valore di '?)'.

```
>  a.out
Questo e' il mio primo programma
> echo ?$
0
```

La compilazione



Compilazione

La compilazione è un processo di traduzione del codice sorgente in linguaggio macchina. Viene generato un file binario eseguibile.

```
gcc  prog1.c -o prog1
```

- gcc è il comando che richiama il compilatore.
- L'opzione `-o` serve per specificare il nome dell'eseguibile. Se omesso, vene assegnato un nome di default (es., `a.out` in Unix, `a.exe` in Windows).

Il file prodotto dal compilatore è ora eseguibile, è quindi possibile dare il comando:

```
prog1
```


Compilazione

Il processo di compilazione avviene in tre passi.

1. Il preprocessor (preprocessore) elimina i commenti e tratta le linee che iniziano con # (*direttive al preprocessore*).

Esempio

```
#include "file.h"
```

Il preprocessore *sostituisce* tale linea con il contenuto del file di nome `file.h` contenuto nella directory corrente. Se il file va ricercato altrove (in locazioni note al sistema) il nome del file va scritto fra le parantesi angolari < e >, esempio:

```
#include <stdio.h>
/* stdio.h e' un file presente nel sistema */
```

Il preprocessore produce un file ASCII che verrà tradotto in codice oggetto dal compilatore.

2. Il compiler (compilatore) traduce il programma sorgente in codice oggetto (file binario).
3. Il linker (o loader) carica e "mette insieme" i file oggetto prodotti al passo 2 e quelli già presenti nel sistema (ad esempio, il file oggetto che contiene `printf()`) e produce un unico file eseguibile.

Con il comando

```
gcc  prog1.c -o prog1
```

vengono eseguiti tutti e tre i passi. L'eseguibile si chiama `prog1`.

> Si raccomanda di leggere attentamente il libro per una descrizione più dettagliata

Funzioni di libreria

- Le *funzioni di libreria* sono contenute in particolari file oggetto (quindi, file già compilati) posti in directory note al sistema. Sono inglobate nel programma dal linker nel passo 3.
- Per poter eseguire il passo 2 il compilatore deve però conoscere il prototipo delle funzioni usate, ossia:
  - numero e tipo dei parametri della funzione
  - tipo del valore restituito dalla funzione.
- Queste informazioni sono contenute negli header file (*file di intestazione*).

Esempio

`printf()` non è una parola chiave del C, ma una funzione di libreria. Il prototipo di `printf()` è contenuto nello header file (`stdio.h`). Quindi, in ogni file in cui `printf()` è chiamata va incluso lo header file `stdio.h`.

Header file

- Sono file ASCII che contengono le definizioni che servono al compilatore per tradurre il codice contenuto in un file (prototipi di funzione, definizioni, ...)
  Per convenzione hanno suffisso `.h`.
- Gli header file vanno inclusi nel sorgente mediante `#include` (quindi sono inclusi dal preprocessore prima della traduzione in codice oggetto). Vengono inclusi all'*inizio* del file in modo che le definizioni contenute in essi siano visibili in tutto il file.

Esempio

In `prog1.c`, viene usata la funzione della libreria standard `printf()`:

```
...
printf("Questo e' il mio primo programma\n");
...
```

- Il prototipo di `printf()` è nel file `stdio.h` che va quindi incluso in `prog1.c`.

  ```
  #include <stdio.h>
  ```

  Come esercizio si provi a cercare `stdio.h` e a leggere ciò che riguarda `printf()`. Si trova una definizione del tipo

  ```
  extern int printf (__const char *__restrict __format, ...);
  ```

- Il codice di `printf()` non è in `stdio.h`, ma è già compilato in un file oggetto.

Importante

Per utilizzare al meglio il proprio compilatore:

- Accertarsi che il compilatore riconosca se il programma è conforme all'ANSI C.
  Con `gcc` usare opzione `-ansi` e `-pedantic`.
- Usare le opzioni che forzano il compilatore a segnalare il maggior numero di avvertimenti (warning).
  Con `gcc` usare opzione `-Wall` (all warning) (raccomandata!).

Si noti che, in presenza di warning, il programma viene comunque compilato. Tuttavia è importante capire il significato degli avvertimenti rilevati dal compilatore perché spesso nascondono errori concettuali di programmazione.

Esempi

Gli esempi qui sotto sono stati ottenuti con la versione 4.1.2... di gcc per Linux. Con altre versioni si possono avere messaggi differenti.

Esempio 1

Sostituire la line

```
#include <stdio.h>
```

con

```
#includ <stdio.h>
```

Viene segnalato l'errore:

```
prog1.c:3:2: error: invalid preprocessing directive #includ
```

Il preprocessore non sa come trattare l'istruzione

```
#includ
```


Esempio 2

Eliminare il `;` dopo `printf(...)`. La compilazione fallisce dando l'errore

```
prog1.c: In function 'main':
prog1.c:8: error: expected ';' before 'return'
```

Significa che nel programma è stato rilevato un errore sintattico (mancanza di `;`) e il compilatore non è in grado di tradurlo.

Esempio 3

Provare a modificare il programma come segue

```
#include <stdio.h>
//  questa e' una linea commentata
....
```

La compilazione ha successo perché gcc (come molti altri compilatori) ammette l'uso della doppia barra per commentare una linea.

Tuttavia facendo

```
gcc -ansi prog1.c
```

la compilazione fallisce rilevando un errore sintattico.

```
prog1.c:3: error: expected identifier or '(' before '/' token
```

Infatti, gli unici commenti ammessi nell'ANSI C sono quelli fra `/*` e `*/` (questo è un utile test da provare con il proprio compilatore).

L'opzione `-pedantic` rivela anomalie di vario tipo presenti nel codice.
Nell'esempio di prima, il programma viene compilato e viene rilevato un *warning*:

```
prog1.c:3:1: warning: C++ style comments are not allowed
 in ISO C89
prog1.c:3:1: warning: (this will be reported only once
 per input file)
```

Un warning segnala una anomalia che non causa il fallimento della compilazione, ma deriva da un uso scorretto del linguaggio che può causare errori durante l'esecuzione del programma.

Esempio 4

Proviamo ora a eliminare la linea `return 0;`

- Il programma è ancora conforme all'ANSI C, in quanto lo standard non richiede che una funzione debba per forza restituire un valore; quindi, anche usando le opzioni `-ansi` e `-pedantic`, il compilatore compila il programma senza lamentarsi.
- Usando invece l'opzione `-Wall` il compilatore, pur compilando il programma, segnala un'anomalia nella funzione `main()`, in quanto viene dichiarata di tipo `int`, ma termina senza restituire alcun valore.

```
prog1.c: In function ‘main’:
prog1.c:7: warning: control reaches end of non-void function
```

  La dimenticanza di istruzioni `return` può provocare errori in esecuzione.

- Consultare attentamente la documentazione del proprio compilatore e vedere le opzioni disponibili
- Imparare a riconoscere gli errori segnalati dal compilatore. Una delle prime difficoltà che si incontrano è proprio il saper capire gli errori segnalati e come correggerli (a volte la dimenticanza di un ";" causa una lunga lista di segnalazioni di errori). L'unico modo di superare l'impatto inziale è di fare esercizi.

Esercizio

In `prog1.c`, sostituire `main` con `man` e compilare il programma. Che errore viene segnalato? In quale fase della compilazione viene rilevato l'errore?