Laboratorio di Programmazione 1

Docente: dr. Damiano Macedonio

Lezione 4 – 24/10/2013

Programmi e Algoritmi

- La risoluzione di un *problema* attraverso un computer richiede di elaborare un algoritmo ed *implementarlo* in un programma.

- Un *algoritmo* è un metodo (ricetta) utilizzato per risolvere un determinato problema.

- Un *programma* è un insieme di istruzioni che permettono di risolvere un dato problema.

Linguaggi di Alto e Basso Livello

- Un computer è in grado di comprendere solo un linguaggio binario ed un insieme limitato di operazioni elementari (*istruzioni macchina*).

- Linguaggi di *basso livello*
  - Esiste una corrispondenza uno-a-uno tra ciascuna istruzione del linguaggio ed una specifica istruzione macchina.
  - Il programma *non* è portabile: non può essere eseguito su un processore diverso.
  - Es. linguaggio Assembly

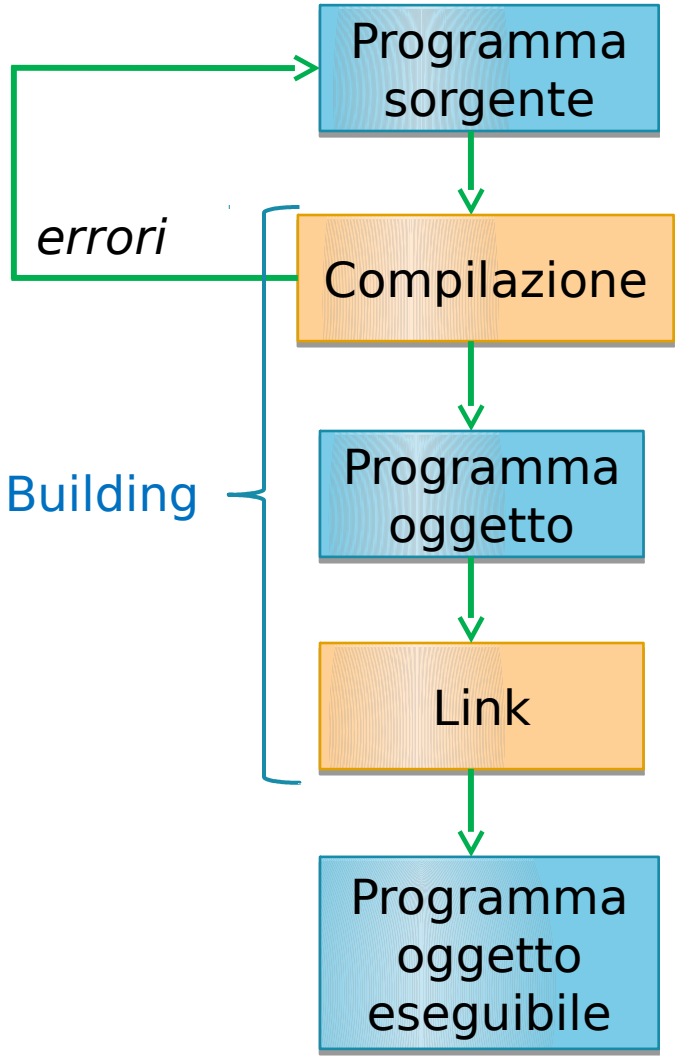
- Linguaggi di *alto livello*
  - Ogni istruzione del linguaggio può essere tradotta in varie istruzioni macchina.
  - I programmi sono indipendenti dalla macchina sottostante.
  - Es. C, C++, Java, ecc...
  - Richiedono la presenza di un *compilatore* o *interprete*.

Compilatore e Interprete

- Un *compilatore* è un programma che analizza un programma scritto in un particolare linguaggio di programmazione e lo *traduce* nel linguaggio macchina adatto ad un certo computer.
  - Es. linguaggio C.

- Un *interprete* è un programma che analizza ed esegue contemporaneamente le istruzioni di un programma.
  - I linguaggi interpretati sono tipicamente più lenti.
  - Es. shell bash ed altri linguaggi di script.

Compilazione di un Programma C



prog.c

Programma scritto nel linguaggio C

gcc prog.c

prog.o

Programma tradotto in linguaggio macchina

Collegamento ad altri programmi compilati o librerie di sistema

prog.out/prog.exe

Primo Programma C

```
#include <stdio.h>
int main( void ){
  printf( “Hello world!\n” );
  return 0;
}
```

- `#include <stdio.h>`
  - Direttiva preprocessore.
  - Inclusione libreria standard I/O.

- `int main( void ){...}`
  - Dichiarazione di funzione
  - `main` funzione speciale: indica dove inizia l’esecuzione del programma.
  - `int`: tipo valore di ritorno.
  - `( void )`: la funzione non ha argomenti.

Primo Programma C

```
#include <stdio.h>
int main( void ){
  printf( “Hello world!\n” );
  return 0;
}
```

- `printf( “Hello world!\n” );`
  - Funzione che stampa o visualizza il suo argomento.
  - Argomento di tipo stringa, `\n` carattere di escape (*newline*).

- `return 0;`
  - Termina l’esecuzione del programma.
  - Produce lo stato 0 (conclusione senza errori).

Compilazione ed Esecuzione

- Compilazione
  - `gcc prog.c`
  - Produce un file di nome `a.out`
- Esecuzione
  - `./a.out`
- È possibile specificare un nome diverso per il programma compilato usando l'opzione `-o`:
  - Compilazione: `gcc prog.c –o prog`

Esecuzione: `./prog`

Esercizio 1

- Compilare ed eseguire il programma C della slide 7 che stampa a video la stringa “Hello world”.

La Funzione Printf

- La funzione `printf` accetta un numero variabile di argomenti (separati da virgola) e permette di stampare anche il valore delle variabili.

```
#include <stdio.h>
int main( void ){
  int sum;
  sum = 10 + 23;
  printf( “10 + 23 = %i\n”, sum );
}
```

Dichiarazione variabile intera
Operazione aritmetica su interi

Argomento 1 Argomento 2

- *Argomento 1*: stringa da visualizzare.
  - La stringa contiene il carattere speciale `%` seguito dal tipo di valore da visualizzare in quel punto (`i` significa intero).
  - `%i` viene sostituito con il successivo argomento della `printf`.

- La stringa può contenere *n* valori da sostituire, dove *n* corrisponde al numero di argomenti successivi della `printf`.

La Funzione Printf

```
#include <stdio.h>
int main( void ){
  int value1;
  int value2;
  int sum;
  value1 = 10;
  value2 = 23;
  sum = value1 + value2;
  printf( “La somma di %i e %i e’:%i\n”,
  value1, value2, sum );
}
```

Esercizio 2

- Compilare ed eseguire il programma C della slide precedente che calcola la somma di due numeri e stampa a video il risultato.

I Commenti

- In C è possibile specificare due tipi di commenti:
  - Commento multilinea racchiusi tra i caratteri `/*` e `*/`

```
/* Questo programma somma due
   numeri interi e visualizza il risultato */
#include <stdio.h>
int main( void ){
  int value1;
  int value2;
  int sum;
```

  - Commento in linea preceduto dai caratteri `//`

```
  // assegnamento valori alle variabili e calcolo somma
   value1 = 10;
   value2 = 23;
   sum = value1 + value2;
   printf( “La somma di %i e %i e’:%i\n”,
       value1, value2, sum );
}
```

La Funzione Scanf

- La funzione `scanf` permette di richiedere all'utente l'inserimento di valori dal terminale.

```
int number;
scanf( "%i", &number );
```

- Il primo argomento "`%i`" è una *stringa di formato* che indica quali tipi di valore devono essere letti dal terminale.

- Il secondo argomento `&number` specifica dove deve essere memorizzato il valore digitato dall'utente.
  - Il carattere `&` prima del nome della variabile `number` è necessario (vedremo più avanti la sua funzione!).

Esercizio 3

- Modificare il programma precedente in modo che richieda in input due valori, ne calcoli la somma, e stampi a video il risultato.

Esercizio 4

- Scrivere un programma C che:
  - Richiede all’utente un numero intero.
  - Stampa a video la sua tabellina da 0 a 10.

Esercizio 5

Scrivere un programma C che accetta in input un numero intero n e stampa:

1) il resto della divisione per due;

   (in C si calcola con l'istruzione n%2)

2) il valore 2n se n è pari, 3n se n è dispari.

Il problema deve essere risolto senza far uso di costrutti di controllo.

Esercizio 6

- Supponendo che i sorgenti C siano stati salvati nella directory `sources`, scrivere uno script bash compile.sh che:
  - Crea la directory `binaries` se non esiste già.
  - Compila un programma contenuto nella directory `sources` passato come primo parametro, usando per l'eseguibile un nome passato come secondo parametro.
  - Sposta il file eseguibile ottenuto nella directory `binaries`.
- Esempio se abbiamo il file ./sources/prova.c e lanciamo ./compile.sh prova.c prova otteniamo l'eseguibile ./binaries/prova