Lezione 5 –Introduzione al Linguaggio C

Corso — Programmazione
Programmazione imperativa – Linguaggio C

Marco Anisetti
e-mail: marco.anisetti@unimi.it
web: http://homes.di.unimi.it/anisetti/

Storia del Linguaggio C

- Fu inventato nel 1972 da Dennis Ritchie presso i laboratori della AT&T Bell
- Serviva a scrivere il codice del sistema operativo UNIX per un processore DEC PDP-11
- Nel 1989 l'American National Standards Institute (ANSI) ha approvato lo standard C89, noto anche come ANSI C
- L'anno seguente ISO ha adottato lo standard ANSI C
    - Revisioni in 1999 (C99), 2011 (C11) e 2018 (C18)

Struttura di un programma

- Un programma è come una **funzione** che prende degli ingressi (x) e ritorna delle uscite (y) ovvero y<-f(x)
- Possiamo iniziare a pensare che tale **funzione** debba essere esplicitata quando scriviamo del codice
    - Formato: <output> <nomefunzione> ‘‘(’’ <input> ‘‘)’’…

- Il programma potrebbe essere fatto da sequenze di funzioni a sua volta

- La funziona principale, quella che ‘‘contiene’’ tutto il programma, viene per convenzione chiamata **main**

- Fisicamente un programma scritto in un linguaggio di programmazione risiede su uno o più file di testo
    - Di solito ogni file contiene una o più funzioni differenti

Struttura di un programma

- Questi file che contengono il programma si chiamano anche ***sorgenti***
- Un programma può essere fatto da più file sorgente che sono tutti necessari per la sua esecuzione
    - Utile per leggere il codice e per riusarlo

- Prima di eseguire il programma quindi vengono in modo automatico riuniti i sorgenti in un'unica entità che poi viene tradotta nel codice eseguibile

- Questo processo si chiama ***compilazione***

- La traduzione può avvenire in più fasi avvicinandosi al vero linguaggio del calcolatore che è chiamato **linguaggio macchina**

Assembler

- Il linguaggio più vicino al linguaggio macchina che non richiede passaggi intermedi nel suo processo di traduzione è l'assembler
- Gli altri linguaggi ad alto livello vengono prima tradotti in assembler e poi in linguaggio macchina
- Alto livello vuol dire che il linguaggi si avvicina di più al modo di pensare delle persone che al modo di agire del computer

Il primo programma in C

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

int main(int argc, char *argv[]) {

printf("Hello");

return 0;
}
```

Il primo programma in C

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

int main(int argc, char *argv[]) {

printf("Hello");

return 0;
}
```

Direttive per il preprocessor (#include inclusione di altri file contenenti codice)

Il primo programma in C

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

int main(int argc, char *argv[]) {

printf("Hello");

return 0;
}
```

Funzione principale con relativi input e output

Per l'output si specifica solo il tipo

Per l'input anche un nome della variabile

L'input è opzionale può essere omesso

```
int main ()
```

Il primo programma in C

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

int main(int argc, char *argv[]) {

printf("Hello");

return 0;
}
```

{ } racchiudono un **blocco di codice** in questo caso il blocco che fa parte della funzione main

Il primo programma in C

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

int main(int argc, char *argv[]) {

printf("Hello");

return 0;
}
```

Una funzione che stampa del testo

il “;” delimita la fine di una linea di codice definendo la **sequenza**

Il primo programma in C

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

int main(int argc, char *argv[]) {

printf("Hello");

return 0;
}
```

Istruzione che ritorna l'output della funzione in questo caso torna l'intero 0

Dichiarazione di variabili

- In C le variabili vanno dichiarate prima di essere usate
- La dichiarazione prevede di esprimere il tipo il nome ed un eventuale valore di inizializzazione attraverso un assegnamento

```
int  somma=0;
```

- L'assegnamento in C è codificato dal simbolo "="
- Assegna il valore che sta alla destra al contenitore (variabile di norma) che sta alla sinistra
- *Proviamo ad alterare l'esempio delle printf dichiarando delle variabili che poi stamperemo*

Overflow e Underflow

- Essendo i numeri rappresentati in uno spazio finito di memoria con un numero prefissato di bit, può succedere che una operazione porti ad un numero non rappresentabile da quella quantità di bit

- Es. se i numeri interi (int) fossero a 3 bit

- int a=8;
- a=a+1 */*con interi a 3 bit porta a sforare il massimo numero rappresentabile*/*

- Si parala di **overflow**

The number glitch that can lead to catastrophe - BBC Future

- **Underow** e l'incapacità di rappresentare un valore reale troppo piccolo, che viene quindi rappresentato come zero.

Input dei dati

- E’ un argomento molto complesso da affrontare al momento

- Per fare esercizio assumeremo di utilizzare delle funzioni senza chiederci molto come mai sono fatte in un certo modo

getchar(): leggere un carattere
scanf(): legge char int float e double

Es.
```
char c;
c=getchar();
```

In realtà quando si usa getchar si preme anche invio dopo il carattere inserito da tastiera quindi i caratteri sono due (\n è un carattere)

La selezione

- Costrutto di selezione in C

If (<espressione>) <blocco then>

If (<espressione>) <blocco then> else <blocco else>

- Il blocco sappiamo se è costituito da più di una istruzione è definito da { }

- Di <espressione> si valuta il valore logico (vero o falso)

- In realtà il C per convenzione lavora sempre con i numeri (il tipo boolean è stato introdotto dopo nel C99)

- Per il C il numero 0 è falso e qualsiasi altro numero è vero (per convenzione si intende l'1)

Espressione: una prima approssimazione

- *E' una sequenza di simboli a cui è associato un valore*
- Per il momento ci accontentiamo di supporre che siano valide le regole della matematica per ottenere il valore dell'espressione
- Abbiamo operatori relazionali (<,>,<=, >=, ==, !=)
- Operatori aritmetici (+,-,*,/,%)
- Operatori logici: (!, &&, ||, …)
- Si possono usare le parentesi tonde per chiarire le precedenze anche se esiste un ordine di precedenza