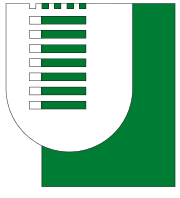
Tutoraggio 2

Gabriele Russo Russo

Corso di Calcolatori Elettronici
A.A. 2018/19



TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Alcune buone pratiche

Any fool can write code that a computer can understand. Good programmers write code that humans can understand.
Martin Fowler

- Un programma non dovrebbe semplicemente "funzionare" (e.g., `bad_code.c`)
- Tanti fattori influenzano la qualità del codice
- Vediamo alcune delle tante buone pratiche da seguire

Indentazione

Nel linguaggio C l'indentazione del codice non ha un valore sintattico. Tuttavia, è fondamentale per la leggibilità del codice.

```
while (i < n) {
    if (i != 2) {
for (j = 1; j < 10; ++j) {
if (j + a > n) {
    do_something(); b = 0;
}
}
}
i++;
}
```

```
while (i < n) {
    if (i != 2) {
        for (j = 1; j < 10; ++j) {
            if (j + a > n) {
                do_something();
                b = 0;
            }
        }
    }
    ++i;
}
```

Commenti

```
// Questo e' un commento
/* Questo e' un commento che
   si estende su piu' righe */
```

E' molto importante commentare il codice, per spiegare le scelte fatte dal programmatore. Commenti che non aggiungono alcuna informazione sono invece inutili. Esempio:

```
// incrementa la variable i di 2 (inutile)
i = i + 2;
```

```
// salta il valore successivo della sequenza perche' ....
i = i + 2;
```

Nel caso delle funzioni, è utile indicare il compito svolto dalla funzione e (se significativo) cosa restituisce la funzione. Esempio:

```
/* Crea un file. Restituisce -1 in caso di errore. */
int create_file() {
    ...
```

Nomi

Una buona scelta di nomi per variabili e funzioni può rendere superflua l'aggiunta di commenti.

```
n = leggi_input1();
if (verifica(n) > 0) {
    c = leggi_input2();
    stampa(n, c);
}
```

```
nome = chiedi_nome();
if (calcola_lunghezza(nome) > 0) {
    cognome = chiedi_cognome();
    stampa(nome, cognome);
}
```

Funzioni

- È bene organizzare il codice in funzioni
- Ogni funzione dovrebbe svolgere un compito ben preciso
- Evitare per quanto possibile i *side effects*
- Per programmi medio-grandi, è utile suddividere il codice anche in più file sorgenti (vedremo come farlo nelle prossime esercitazioni)

Esercizio: operatori bitwise (e non solo)

```
int n = 1030; int z = 0; char c = 34;
int x;
```

Qual è il contenuto di `x` dopo ciascuna di queste istruzioni?

```
1) x = n && c;
2) x = n && z;
3) x = n & c;
4) x = n & z;
5) x = n & 0xFF;
6) x = c & 0xFF;
7) x = n & ~c;
8) x = n >> 1;
9) x = c << 1;
10) x = c ^ (0xFF & n);
11) x = n % (c % 30);
```

Codice: esempio_bitwise.c

Esercizio

Scrivere un programma C che riceve in input un intero $0 \leq N \leq 10$ e calcola il valore della funzione $f(n)$ per ogni intero $0 \leq n \leq N$.

$$f(n) = \sum_{i=1}^{n^2} \sum_{k=1}^{\lceil \frac{i}{2} \rceil} Fib(k \ mod \ 10)$$

dove $Fib(n)$ indica l'n-esimo numero di Fibonacci:

$$Fib(n) = \begin{cases} Fib(n-1) + Fib(n-2), & n \geq 2 \\ n, & n \in \{0, 1\} \end{cases}$$

Codice: `esempio_fib_loops.c`