

CoderFarm - Corso base

Lezione 1

Carlo Collodel, Francesco Cerroni

24 ottobre 2022

Hello World!

Il vostro (forse) secondo programma in C++



```
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello World!" << endl;
}
```

https://www.onlinegdb.com/online_c++_compiler

Hello World!

Il vostro (forse) secondo programma in C++

- ▶ Provate a cambiare il testo dentro le doppie virgolette!
- ▶ Provate a inserire dei valori *speciali*: `"\n"`, `"\t"`, `"\r"`, ...
- ▶ Provate a inserire altri valori *speciali*: `"\x61"`, `"\x6d"`, `"\x6f"`, `"\x67"`, `"\x75"`, `"\x73"`, `"\x0a"`, ...
- ▶ Provate a inserire ancora altri valori *speciali*: `"\163"`, `"\151"`, `"\165"`, `"\155"`, ...

Basi di programmazione

Variabili

Una **variabile** può essere vista come una *scatola* (con una dimensione) che può contenere un valore qualsiasi.

In matematica avrete sicuramente già incontrato le variabili, ad esempio nell'espressione

$$3x^2 + 2x - 5$$

x è un numero (una variabile) che può assumere qualsiasi(!) valore. Come in Matematica le variabili possono appartenere a insiemi diversi, in Informatica le variabili possono essere di *tipi* diversi: esistono i numeri interi, i numeri decimali, i caratteri, le stringhe (insiemi di caratteri), ecc.

Basi di programmazione

Tipi primitivi

In C++ esistono vari tipi detti *primitivi*, perchè costituiscono lo "scheletro" per costruire dei tipi più complessi. Questi tipi sono:

- ▶ **Intero**: come suggerisce il nome, rappresentano numeri interi.
- ▶ **Float**: servono a rappresentare i numeri decimali.
- ▶ **Booleano**: una variabile *booleana* rappresenta un valore che può essere solo vero (true) o falso (false).
- ▶ **Carattere**: rappresenta un singolo carattere ASCII.

Basi di programmazione

Tipi numerici interi

Keyword	Numero di bit	Intervallo
short (int)	16	$[-2^{15}, 2^{15} - 1]$
unsigned short (int)	16	$[0, 2^{16} - 1]$
int	32	$[-2^{31}, 2^{31} - 1]$
unsigned (int)	32	$[0, 2^{32} - 1]$
long long (int)	64	$[-2^{63}, 2^{63} - 1]$
unsigned long long (int)	64	$[0, 2^{64} - 1]$
__int128_t	128	$[-2^{127}, 2^{127} - 1]$
__uint128_t	128	$[0, 2^{128} - 1]$

Basi di programmazione

Perché usare valori unsigned?

Per gli interi e i caratteri si può aggiungere prima del nome la parola "unsigned" per imporre che questi rappresentino solamente valori non negativi.

Non sono usati molto in programmazione competitiva, però permettono di ottenere:

- ▶ Il doppio dei valori rappresentabili.
- ▶ ~~La soddisfazione di essere dei bravi programmatori.~~
- ▶ Se bisogna verificare che dei valori appartengano a un range $[0, k]$, allora si può omettere il confronto " > 0 ".

Basi di programmazione

Tipi numerici decimali

Keyword	Numero di bit
float	32
double	64
long double	64/80/128

Basi di programmazione

Booleani

Le variabili booleane prendono il nome da *George Boole*, che per primo creò un'algebra dove le variabili potevano assumere solamente due valori: vero o falso. In C++, per utilizzare questa tipologia di variabili si usa la *keyword* **bool**, che occupa 8 bit di spazio in memoria(!).

Basi di programmazione

Caratteri

Per memorizzare variabili di tipo carattere si usa la *keyword* **char** e, come la variabile di tipo `bool`, anche questo tipo di variabile occupa 8 bit di spazio.

I tipi di carattere rappresentabili sono consultabili nella tabella ASCII (<https://www.asciitable.com/> oppure, su Linux, `man ascii`).

P.S.

Esistono anche i tipi **wchar_t**, **char16_t** e **char32_t** che permettono di codificare una quantità maggiore di caratteri (non richiesto in programmazione competitiva).

Basi di programmazione

Dichiarazione di variabili



```
#include <iostream>

using namespace std;

int main() {
    int x;
    char c;
    __int128_t big;
    unsigned long long y;
}
```

Basi di programmazione

Inizializzazione di variabili



```
#include <iostream>

using namespace std;

int main() {
    int x = 42;
    char c = 'E';
    __int128_t big = (__int128_t)69;
    unsigned long long y = 420ULL;
}
```

Basi di programmazione

Operazioni tra numeri

Le 5 principali operazioni aritmetiche supportate sono:

Operatore	Nome
+	Somma
-	Sottrazione
*	Prodotto
/	Divisione
%	Modulo (resto della divisione)

Basi di programmazione

Somma



```
#include <iostream>

using namespace std;

int main() {
    int a = 2;
    int b = 3;
    int c = a + b; // c = 5
    cout << c << "\n";
}
```

Basi di programmazione

Sottrazione



```
#include <iostream>

using namespace std;

int main() {
    int a = 5;
    int b = 2;
    int c = a - b; // c = 3
    cout << c << "\n";
}
```

Basi di programmazione

Moltiplicazione



```
#include <iostream>

using namespace std;

int main() {
    int a = 4;
    int b = 3;
    int c = a * b; // c = 12
    cout << c << "\n";
}
```


Basi di programmazione

Divisione



```
#include <iostream>

using namespace std;

int main() {
    /*
     * la divisione tra interi approssima il risultato
     * sempre per difetto, infatti  $10 / 3 = 3.333333...$ 
     * che approssimato per difetto è 3.
     */
    int a = 10;
    int b = 3;
    int c = a / b; // c = 3
    cout << c << "\n";
}
```

Basi di programmazione

Modulo



```
#include <iostream>

using namespace std;

int main() {
    /*
       c è uguale al resto della divisione
       tra 27 e 5
    */
    int a = 27;
    int b = 5;
    int c = a % b; // c = 2
    cout << c << "\n";
}
```

Basi di programmazione

Operazione e assegnazione



```
#include <iostream>

using namespace std;

int main() {
    // funziona con tutte le operazioni:
    // +=, -=, *=, /=, %=
    int a = 5;
    a += 2; // a = a + 2, a = 7
}
```

Basi di programmazione

Incremento e decremento

Altre operazioni utili sono l'**incremento** e il **decremento**

```
● ● ●  
  
#include <iostream>  
  
using namespace std;  
  
int main() {  
    // a++ incrementa di 1 il valore di a dopo l'istruzione  
    int a = 5;  
    int b = a++; // b = 5, a = 6  
  
    // ++a incrementa di 1 il valore di a prima dell'istruzione  
    a = 5;  
    int c = ++a; // c = 6, a = 6  
  
    // esiste l'operatore -- che funziona allo stesso modo  
    // ma decrementa di 1  
    a = 5;  
    int d = a--; // d = 5, a = 4  
  
    a = 5;  
    int e = --a; // e = 4, a = 4  
}
```

Basi di programmazione

Operazioni tra numeri

Precisazioni:

- ▶ Se dividete un qualsiasi valore per 0 il programma terminerà.
- ▶ Il punto precedente vale anche se cercate di calcolare il resto della divisione per 0 (internamente viene effettuata una divisione).
- ▶ L'operatore modulo è un'operazione che "ha senso" solo tra due numeri interi.

Basi di programmazione

Operazioni binarie tra interi

Dato che ogni numero intero viene memorizzato in base due, esistono operazioni che agiscono *bit a bit* tra due numeri. Queste sono dette operazioni **bitwise**.

Nome	Simbolo	Spiegazione
and	&	associa 1 se entrambi i bit sono 1
or		associa 1 se almeno uno dei 2 bit è uguale a 1
xor	^	associa 1 se i due bit hanno valore diverso

Basi di programmazione

Operazioni binarie su singoli interi

Esistono 3 operazioni binarie che si applicano a un singolo valore:

Nome	Simbolo	Spiegazione
Inverso	\sim	cambia il valore di tutti i bit
Shift sinistro	$<<$	aggiunge un certo numero di zeri a inizio numero
Shift destro	$>>$	elimina un certo numero di bit da inizio numero

Basi di programmazione

and



```
#include <iostream>
```

```
using namespace std;
```

```
int main() {
```

```
    int a = 13;    // 1101
```

```
    int b = 11;    // 1011
```

```
    int c = a & b; // 1001
```

```
}
```


Basi di programmazione

or



```
#include <iostream>
```

```
using namespace std;
```

```
int main() {  
    int a = 9;      // 1001  
    int b = 11;     // 1011  
    int c = a | b;  // 1011  
}
```

Basi di programmazione

xor



```
#include <iostream>
```

```
using namespace std;
```

```
int main() {  
    int a = 12;    // 1100  
    int b = 10;    // 1010  
    int c = a ^ b; // 0110  
}
```

Basi di programmazione

Inverso



```
#include <iostream>
```

```
using namespace std;
```

Basi di programmazione

Shift sinistro



```
#include <iostream>

using namespace std;

int main() {
    // aggiunta di 2 zeri all'inizio di a
    int a = 5;          // 101
    int b = 2;
    int c = a << b; // 10100
}
```

Basi di programmazione

Shift destro



```
#include <iostream>

using namespace std;

int main() {
    // rimozione dei primi 2 bit di a
    int a = 21;      // 10101
    int b = 2;
    int c = a >> b; // 101
}
```

Basi di programmazione

Osservazioni sugli shift

Le operazioni di shift hanno le seguenti proprietà:

- ▶ Shift sinistro di n bit, è come moltiplicare per 2^n .
- ▶ Shift destro di n bit è come dividere per 2^n (approssimando per difetto).
- ▶ Attenzione ai numeri negativi!

Basi di programmazione

Operazioni binarie

Esercizio

Come impostare il terzo bit di un numero a 1 usando solo operazioni binarie?
(AND, OR, XOR, Inverso, Shift sinistro e Shift Destro).

Basi di programmazione

Operazioni binarie

Soluzione

Sia n un numero intero positivo. Allora l'operazione da fare è:

$$n = n \mid (1 \ll 3)$$


Oppure:

$$n = n \mid 8$$

Basi di programmazione

Leggere valori in input

Per leggere un valore inserito dall'utente si utilizza la funzione *cin*:



```
#include <iostream>

using namespace std;

int main() {
    int n; // creazione della variabile
    cin >> n; // leggi il valore di n
    cout << "Hai inserito il numero " << n << "\n";

    int a, b; // creazione di 2 variabili insieme
    cin >> a >> b; // lettura prima di a e poi di b
}
```

Basi di programmazione

Leggere e scrivere da File



```
#include <iostream>

using namespace std;

int main() {
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
}
```

Basi di programmazione

Condizioni

A volte bisogna eseguire certe operazioni in base a delle condizioni. Consideriamo questo esempio:

Una strana richiesta


Sia n un intero positivo.

- ▶ Se n è pari stampare $\frac{n}{2}$
- ▶ Se n è dispari stampare $3n + 1$

Basi di programmazione

Condizioni

In generale un blocco condizionale (if) funziona nel seguente modo:



```
if (condizione) {  
    // codice da eseguire se la condizione è vera  
}  
  
if (condizione) {  
    // codice da eseguire se la condizione è vera  
} else {  
    // codice da eseguire se la condizione è falsa  
}
```

Basi di programmazione

Condizioni

Possiamo anche confrontare dei valori:

Simbolo	Significato
<code>==</code>	Uguale
<code>!=</code>	Diverso
<code>></code>	Maggiore
<code><</code>	Minore
<code>>=</code>	Maggiore o uguale
<code><=</code>	Minore o uguale

Basi di programmazione

Operatori logici

Quando abbiamo a che fare con condizioni logiche (o variabili booleane) esistono 3 operatori che possiamo usare per concatenare diverse condizioni (o variabili):

- ▶ **and** (`&&`) - $A \&\& B$ è vera solo se A e B sono entrambe vere
- ▶ **or** (`||`) - $A || B$ è vera se almeno una tra A e B è vera
- ▶ **not** (`!`) - $!A$ è vera se A è falsa, falsa se A è vera

Basi di programmazione

Condizioni

Scriviamo un programma per l'esercizio precedente:

```
● ● ●  
  
#include <iostream>  
  
using namespace std;  
  
int main() {  
    int n;  
    cin >> n; // leggiamo il valore di n  
  
    if (n % 2 == 0) { // se n è divisibile per 2  
        cout << n / 2 << "\n";  
    } else { // altrimenti n è dispari  
        cout << 3 * n + 1 << "\n";  
    }  
}
```

Basi di programmazione

Cicli

Proviamo a modificare l'esercizio precedente:

Una strana richiesta (pt.2)

Sia n un intero positivo. Vogliamo costruire una successione di numeri in base a questa regola:

- ▶ Se n è pari il prossimo elemento sarà $\frac{n}{2}$
- ▶ Se n è dispari il prossimo elemento sarà $3n + 1$

A questo punto n diventa il nuovo valore aggiunto e si ripete il processo finchè n è diverso da 1.


Esempio:

$$5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$$

Basi di programmazione

Cicli

Per creare un ciclo in C++ possiamo usare un'istruzione **while**:




```
while (condizione){  
    // codice da eseguire  
    // finchè la condizione  
    // è vera  
}  
  
do {  
    // codice da eseguire  
    // finchè la condizione  
    // è vera  
} while (condizione);
```

Basi di programmazione

Cicli

La soluzione dell'esercizio:



```
int main() {
    int n; // diamo per scontato che n >= 1
    cin >> n; // lettura di n
    while (n != 1) { // finchè n è diverso da 1
        cout << n << " ";

        if (n % 2 == 0) { // se n è pari
            n = n / 2;
        } else { // se n è dispari
            n = 3 * n + 1;
        }
    }
    cout << "1 \n";
}
```

Basi di programmazione

Cicli

Per determinati cicli conviene utilizzare un ciclo **for**:

```
for (codice pre - ciclo; condizione; codice alla fine di ogni ciclo) {  
    // codice  
}
```

// esempio: i 2 blocchi seguenti sono equivalenti

```
for (int i = 0; i < 10; i++) {  
    cout << i << "\n";  
}
```

```
int i = 0;  
while (i < 10) {  
    cout << i << "\n";  
    i++;  
}
```

Esercizi!

- ▶ Trova la parità
<https://training.olinfo.it/#/task/pari/statement>
- ▶ Trova il massimo
<https://training.olinfo.it/#/task/easy1/statement>
- ▶ Trova la somma pari massima
<https://training.olinfo.it/#/task/easy2/statement>

Fine

Ci vediamo alla prossima lezione!

- ▶ **E-Mail:** `base@coderfarm.it`
- ▶ **Telegram:** T.B.D.