Uso pratico degli operatori BIT a BIT

Gabriele Frassi

```
#include <iostream>
using namespace std;

int main() {
    int size = 8*sizeof(unsigned int); // "(vedi 1)"

    unsigned int M1 = 1u << size-1; // "(32-1) (vedi 2)"

    unsigned int x;
    cout<<"Digita un numero senza segno: "; // "(vedi 3)"
    cin>>x;

    // "(vedi 4 e 5)"

    cout<<"La codifica binaria di 5 e': ";
    while(M1 != 0) {
        if((M1&x) != 0)
            cout<<1;
        else
            cout<<0;

        M1 = M1 >> 1;
    }
}
```

Note

1. Calcolo il numero di bit disponibili per la mia variabile. In questo caso sono 32. sizeof indica il numero di byte, quindi moltiplico per 8 ottenendo i bit totali.
2. Creo una maschera di bit che mi rappresenta l'1 binario.
    - Sposto l'uno nella prima casella a sinistra che ho a disposizione.
    - Utilizzo per questo scopo l'operatore bit a bit "traslazione a sinistra".
    - Se ho 32 bit per spostare l'1 dall'ultima alla prima casella da sinistra ho bisogno di aggiungere 31 zeri.
    - **Prima**: 00000000000000000000000000000001
    - **Dopo**: 10000000000000000000000000000000
3. Richiedo il numero che voglio convertire in base binaria

4. $M1! = 0$ significa che ripeterò il corpo del while finchè la maschera M1 non corrisponderà a zero.

   - Se ho 32 bit la maschera che mi rappresenta zero è costituita esclusivamente da zeri.
   - Con l'operatore "traslazione a destra", pertanto, sposto l'1 da sinistra verso destra fino a farlo uscire.
   - **Inizio**: M1 = 10000000000000000000000000000000
   - **Sposto**:
     - 00100000000000000000000000000000
     - 00000000000000001000000000000000
     - 00000000000000000000000000000001
   - **Fine**: M1 = 00000000000000000000000000000000

5. Con l'operatore AND bit a bit confronto la maschera M1 con la maschera che rappresenta un numero x.

   Utilizzare l'operatore AND bit a bit significa confrontare bit per bit come con un AND logico, ottenendo una nuova maschera. la presenza di un solo 1 in M1, spostato ogni volta di una posizione verso destra, fa si che ciò che è espresso da $M1\&x$ dipenda esclusivamente dal confronto tra due bit (di cui uno sempre 1).
   Se c'è corrispondenza tra questi due bit (1 AND 1), $M1\&x$ esprimerà in base dieci un numero diverso da zero.
   Se il numero è diverso da zero significa che nella nuova maschera il bit sarà 1, altrimenti 0.

   Poniamo, per esempio, x = 9, che in base binaria corrisponde a 1001.

   Ho le due maschere:
   M1 = 10000000000000000000000000000000
   x = 00000000000000000000000000001001

   Nelle prime 28 operazioni la maschera risultante sarà: 00000000000000000000000000000000
   cout << 0; (Non ho avuto corrispondenza, 1 AND 0 : false)

   Nella 29esima operazione la maschera risultante sarà: 00000000000000000000000000001000
   cout << 1; (Ho avuto corrispondenza, 1 AND 1 : true)

   Nella 30esima e nella 31esima operazione avrò: 00000000000000000000000000000000
   cout << 0; (Non ho avuto corrispondenza, 1 AND 0 : false)

   Nella 32esima e ultima operazione avrò come maschera: 00000000000000000000000000000001
   cout << 1; (Ho avuto corrispondenza, 1 AND 1 : true)

   Dopo 32 cout avrò epresso la maschera: 00000000000000000000000000001001