DiST
Dipartimento di Scienze e Tecnologie

Laurea triennale in Informatica

modulo (CFU 6) *di*

Programmazione II e Lab.

prof. Mariarosaria Rizzardi

Centro Direzionale di Napoli – Isola C4
stanza: n. 423 – IV piano Lato Nord
tel.: 081 547 6545
email: mariarosaria.rizzardi@uniparthenope.it

- **Operatori logici in C++ (il tipo bool)**

- **Operatori bitwise in C++11 (la classe bitset)**

provare...

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

int main()
{
    cout << "sizeof(char)      = " << sizeof(char)  << endl;
    cout << "sizeof(short)     = " << sizeof(short) << endl;
    cout << "sizeof(long)      = " << sizeof(long)  << endl;
    cout << "sizeof(long long) = " << sizeof(long long)  << endl;
    cout << "sizeof(char*)     = " << sizeof(char*) << endl;
    return 0;
}
```

output

MinGW
Windows

```
sizeof(char)      = 1
sizeof(short)     = 2
sizeof(long)      = 4
sizeof(long long) = 8
sizeof(type*)     = 8
```

lunghezza in byte

1 byte = 8 bit

Tipo “puntatore”

perché?

Variabili logiche in C++ (logical operators)

In **C++** esiste il tipo predefinito **bool** (tipo booleano o logico) con valori {**false**, **true**}, corrispondenti ai valori interi {0, 1}, e gli stessi operatori del **C** (and, or, not).

```
#include<iostream>
using namespace std;
int main()
{   bool x = 0;      // false
    bool y = 100;    // true
    bool z = 15.75;  // true
    cout << "x = " << x << endl;
    cout << "y = " << y << endl;
    cout << "z = " << z << endl;
return 0;
}
```

```
x = 0
y = 1
z = 1
```

```
#include<iostream>
using namespace std;
int main()
{   int x1=10, x2=20, m=2;
    bool b1, b2;
    b1 = x1 == x2; // false
    b2 = x1 < x2;  // true
    cout << "b1 = " << b1 << "\n";
    cout << "b2 = " << b2 << "\n";
    bool b3 = true;
    if (b3)
        cout << "Yes" << "\n";
    else
        cout << "No" << "\n";
    int x3 = false + 5*m - b3;
    cout << x3 << "\n";
    return 0;
}
```

```
b1 = 0
b2 = 1
Yes
9
```

```
int x3 = false + 5*m - b3;
            0       10    1
```

Visualizzare "**true**" e "**false**" in C++ invece di 1 e 0

```
#include<iostream>
using namespace std;
int main()
{    bool x = 0;       // false
     bool y = 100;     // true
     bool z = 15.75;  // true
     cout << "x = " << boolalpha << x << endl;
     cout << "y = " << boolalpha << y << endl;
     cout << "z = " << boolalpha << z << endl;
return 0;
}
```

```
x = false
y = true
z = true
```

format flag

```
#include<iostream>
using namespace std;
int main()
{    int x1=10, x2=20, m=2;
     bool b1, b2;
     b1 = x1 == x2; // false
     b2 = x1 < x2;  // true
     cout << "b1 = " << boolalpha << b1 << "\n";
     cout << "b2 = " << boolalpha << b2 << "\n";
     bool b3 = true;
     int x3 = false + 5*m - b3;
     cout << x3 << boolalpha << "\n";
     return 0;
}
```

```
b1 = false
b2 = true
9
```

Operatori bitwise in C++

logical operators

!	
F	V
V	F

&&	F	V
F	F	F
V	F	V

||	F	V
F	F	V
V	V	V

bitwise operators

~	
0	1
1	0

&	0	1
0	0	0
1	0	1

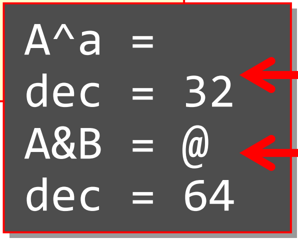
|	0	1
0	0	1
1	1	1

^	0	1
0	0	1
1	1	0

```
#include<iostream>
using namespace std;
int main()
{   char A='A', a='a', B='B', b='b', r;
    r = A^a;
    cout << "A^a = " << r << endl;
    cout << "dec = " << (int)r << endl;
    r = A&B;
    cout << "A&B = " << r << endl;
    cout << "dec = " << (int)r << endl;
    return 0;
}
```

come intero signed a 8 bit

char	ASCII	binario
A	65	0100 0001
a	97	0110 0001
B	66	0100 0010
b	98	0110 0010



```
A^a = 
dec = 32
A&B = @
dec = 64
```

00100000

01000000

Operatori bitwise in C++: bitset

Il modo migliore per gestire bit in C++ è usare la **classe template** **bitset<N>**, dove N è noto al tempo della compilazione.

La classe emula un array of bit, ma ottimizza l'allocazione di spazio: ogni elemento occupa solo un bit.

```
#include<iostream>
#include<bitset>
using namespace std;
int main()
{   bitset<8> A='A', a='a', B='B', b='b', r;

    r = A^a; cout << "A^a = " << r << endl;
    cout << "dec = " << r.to_ulong() << endl;

    r = A&B; cout << "A&B = " << r << endl;
    cout << "dec = " << r.to_ulong() << endl;

    r=A<<2; cout << "dec = " << r.to_ulong() << endl;
    r=a>>3; cout << "dec = " << r.to_ulong() << endl;

    cout << "r.size() = " << r.size() << endl;
    cout << "r.count() = " << r.count() << endl;
return 0;
}
```

visualizza in decimale

```
A   = 01000001
a   = 01100001
A^a = 00100000
dec = 32

A   = 01000001
B   = 01000010
A&B = 01000000
dec = 64

A    = 01000001
A<<2 = 00000100
dec = 4

a    = 01100001
a>>3 = 00001100
dec = 12
```

```
r.size()  = 8
r.count() = 2
```

numero totale di bit
numero di bit pari a 1

Operatori bitwise in C++: bitset

```
#include<iostream>
#include <bitset>
using namespace std;
int main()
{   unsigned short N = 125;
    bitset<8> Bits (N); // oppure Bits = N;
    cout << "Bits = " << Bits << endl;
    bitset<8> Bits2 = 0x0F;
    cout << Bits << " & 00001111 = " << (Bits2 & Bits) << endl;

    cout << Bits << " << 4 = " << (Bits << 4) << endl;
                                   // uguale a Bits <<= 4
    return 0;
}                                  // uguale a Bits=Bits << 4
```

```
Bits = 01111101
```

```
01111101 & 00001111 = 00001101
```

```
01111101 << 4 = 11010000
```

```
#include<iostream>
#include <bitset>
using namespace std;
int main()
{   bitset<8> Bits; cout << "Bits = " << Bits << '\n';
    Bits.set(3);    cout << "Bits = " << Bits << '\n';
    Bits.set();     cout << "Bits = " << Bits << '\n';
    Bits.flip(2);   cout << "Bits = " << Bits << '\n';
    Bits.flip();    cout << "Bits = " << Bits << '\n';
    return 0;
}
```

```
Bits = 00000000
Bits = 00001000
Bits = 11111111
Bits = 11111011
Bits = 00000100
```

anche i metodi .set() e .reset() possono agire su un solo bit

Operatori bitwise in C++: bitset

Si può accedere ad ogni singolo bit di un **bitset<N>** mediante bitset::reference, come in un array.
L'indice `0` corrisponde al bit meno significativo (più a destra), l'indice **N-1** corrisponde al bit più significativo (più a sinistra).

```
#include<iostream>
#include <bitset>
#define N 4

using namespace std;

int main()
{   bitset<N> Bits;
    Bits[1]=1;          // 0010
    Bits[2]=Bits[1]; // 0110
    cout << "Bits: " << Bits << '\n';

    int I = (int)Bits.to_ulong();
    cout << "I=Bits: " << I << endl;
    return 0;
}
```

```
Bits: 0110
```

```
I=Bits: 6
```

Esercizi C++

Scrivere una function C++ per invertire l'ordine dei bit di una variabile intera **short** A mediante **bitset**.

b_{15}	…	b_5	b_4	b_3	b_2	b_1	b_0

b_0	b_1	b_2	b_3	b_4	b_5	…	b_{15}

Idea

```
#include<iostream>
#include <bitset>
using namespace std;
int main()
{   short A=-13;   bitset<16> Bits (A);
    cout << "A = -13 = Bits:  " << Bits << '\n';
    short B=0;
    for (char k=0; k<Bits.size(); k++)
    {
        B <<= 1;
        B |= Bits[k];
    }
    Bits = B; cout << "B = " << B << " bits: " << Bits << endl;
    return 0;
}
```

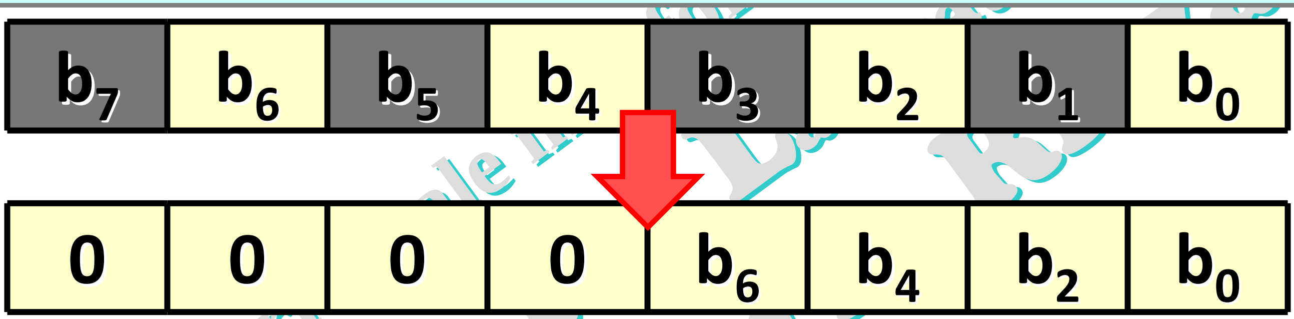
inizializza al valore di A

```
A = -13 = Bits:
1111111111110011
```

```
B = -12289 bits:
1100111111111111
```

Esercizi C++

Scrivere una function C++ che estragga da una variabile di tipo intero char A i suoi bit di posto pari mediante bitset.



b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	0	0	b_6	b_4	b_2	b_0

Idea

```
#include<iostream>
#include <bitset>
using namespace std;
int main()
{   char A=-13;   bitset<8> Bits (A);
    cout << "Bits:  " << Bits << '\n';
    char B=0;     bitset<8> Bits2;
    for (char k=0; k<Bits.size(); k+=2)
        Bits2[k/2] = Bits[k];
    B=Bits2.to_ulong();
    cout << "Bits: " << Bits2 << " bits: " << Bits << endl;
    return 0;
}
```

```
Bits: 11110011
k=0      Bits[k] = 1
k=2      Bits[k] = 0
k=4      Bits[k] = 1
k=6      Bits[k] = 1
Bits: 00001101
```