INF 112 – Programação 2

Aula: Bits

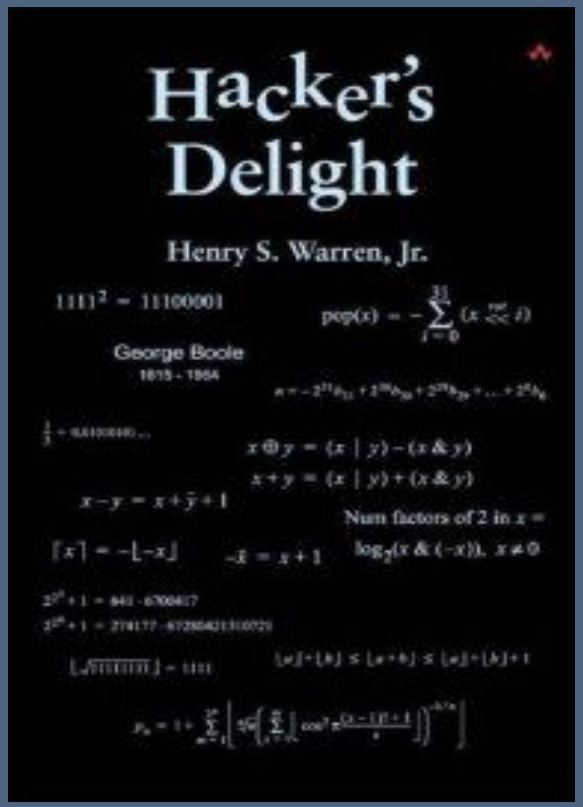
Operadores de bits

C++ oferece várias ferramentas para se manipular bits.

O acesso a bits é importante em várias situações:

- Softwares com alto desempenho.
- Economizar memória com valores booleanos.
- Desenvolvimento de SO.
- Softwares para redes.
- Etc.

Operadores de bits



Obs: esta aula foi baseada no livro texto e nas seguintes referências:

- “A bit of fun: fun with bits” , Topcoder (http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=bitManipulation)
- “Hacker's Delight”, Henry S. Warren

Operadores de bits

As principais ferramentas para manipulação de bits são os operadores de bits:

Operador	Nome	Descrição
&	“E” sobre bits	Os bits resultantes valem 1 se os bits correspondentes valem 1 e 0 caso contrário.
|	“Ou” sobre bits	Os bits resultantes valem 1 se pelo menos um dos bits correspondentes valem 1 e 0 caso contrário.
^	“Ou exclusivo” sobre bits.	Os bits resultantes valem 1 se exatamente um dos bits correspondentes valem 1 e 0 caso contrário.
<<	Deslocamento de bits para a esquerda.	Desloca os bits do primeiro operando para a esquerda pelo número de bits especificados no segundo operando. Preenche a partir da direita com bits 0.
>>	Deslocamento de bits para a direita.	Desloca os bits do primeiro operando para a direita pelo número de bits especificados no segundo operando. O preenchimento à esquerda depende da máquina.
~	Complemento de bits.	Os bits 0 são transformados em 1 e os bits 1 são transformados em 0.

Operadores de bits

Tabela verdade

A	B	A&B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A|B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A^B
0	0	0
0	1	1
1	0	1
1	1	0

A	~A
0	1
1	0

Operadores de bits

Tabela verdade

A	B	A&B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A|B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A^B
0	0	0
0	1	1
1	0	1
1	1	0

A	~A
0	1
1	0

```
 38 & 40

00100110
00101000
--------
00100000

   32
```

Operadores de bits

Tabela verdade

A	B	A&B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A|B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A^B
0	0	0
0	1	1
1	0	1
1	1	0

A	~A
0	1
1	0

```
 38 & 40

00100110
00101000
--------
00100000

   32
```

```
 38 | 40

00100110
00101000
--------
00101110

   46
```

Operadores de bits

Tabela verdade

A	B	A&B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A|B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A^B
0	0	0
0	1	1
1	0	1
1	1	0

A	~A
0	1
1	0

```
 38 & 40

00100110
00101000
--------
00100000

   32
```

```
 38 | 40

00100110
00101000
--------
00101110

   46
```

```
 38 ^ 40

00100110
00101000
--------
00001110

   14
```

Operadores de bits

Tabela verdade

A	B	A&B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A|B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A^B
0	0	0
0	1	1
1	0	1
1	1	0

A	~A
0	1
1	0

```
 38 & 40

 00100110
 00101000
--------
 00100000

    32
```

```
 38 | 40

 00100110
 00101000
--------
 00101110

    46
```

```
 38 ^ 40

 00100110
 00101000
--------
 00001110

    14
```

```
   ~38

 00100110
--------
 11011001

217 ou -39
```

Operadores de bits

Deslocamento de Bits (<<)

```
int x = 10;
//00001010
x = x << 2;
//00101000 (=40, ou seja multiplicamos 2x por 2)
```

Operadores de bits

Deslocamento de Bits (>>)

```
int x = 42;
//00101010
x = x >> 1;
//00010101 (=21)
```

```c
int main() {
    unsigned int a = 3;
    unsigned int b = 5;

    imprimeBits(b);
    imprimeBits(a);
    imprimeBits(~a);
    imprimeBits(a | b);
    imprimeBits(a & b);
    imprimeBits(a ^ b);
    imprimeBits(a << 1);
    imprimeBits(a << 2);
    imprimeBits(a >> 1);
    imprimeBits(a >> 2);

    return 0;
}
```

```
0000000 00000000 00000000 00000101
0000000 00000000 00000000 00000011
1111111 11111111 11111111 11111100
0000000 00000000 00000000 00000111
0000000 00000000 00000000 00000001
0000000 00000000 00000000 00000110
0000000 00000000 00000000 00000110
0000000 00000000 00000000 00001100
0000000 00000000 00000000 00000001
0000000 00000000 00000000 00000000
```

Operadores de bits

Ligando e desligando bits

2^0	=	1	=	00000001_2
2^1	=	2	=	00000010_2
2^2	=	4	=	00000100_2
2^3	=	8	=	00001000_2
2^4	=	16	=	00010000_2
2^5	=	32	=	00100000_2
2^6	=	64	=	01000000_2
2^7	=	128	=	10000000_2

```
int x = 0;
x = x | 4;
x = x | 8;
x = x | 128;
```

```
00000000
```

Operadores de bits

Ligando e desligando bits

2^0	=	1	=	00000001_2
2^1	=	2	=	00000010_2
2^2	=	4	=	00000100_2
2^3	=	8	=	00001000_2
2^4	=	16	=	00010000_2
2^5	=	32	=	00100000_2
2^6	=	64	=	01000000_2
2^7	=	128	=	10000000_2

```
int x = 0;
x = x | 4;
x = x | 8;
x = x | 128;
```

```
00000000

00000000 | 00000100
```

Operadores de bits

Ligando e desligando bits

2^0	=	1	=	00000001_2
2^1	=	2	=	00000010_2
2^2	=	4	=	00000100_2
2^3	=	8	=	00001000_2
2^4	=	16	=	00010000_2
2^5	=	32	=	00100000_2
2^6	=	64	=	01000000_2
2^7	=	128	=	10000000_2

```
int x = 0;
x = x | 4;
x = x | 8;
x = x | 128;
```

```
00000100

00000100 | 00001000
```

Operadores de bits

Ligando e desligando bits

2^0	=	1	=	00000001_2
2^1	=	2	=	00000010_2
2^2	=	4	=	00000100_2
2^3	=	8	=	00001000_2
2^4	=	16	=	00010000_2
2^5	=	32	=	00100000_2
2^6	=	64	=	01000000_2
2^7	=	128	=	10000000_2

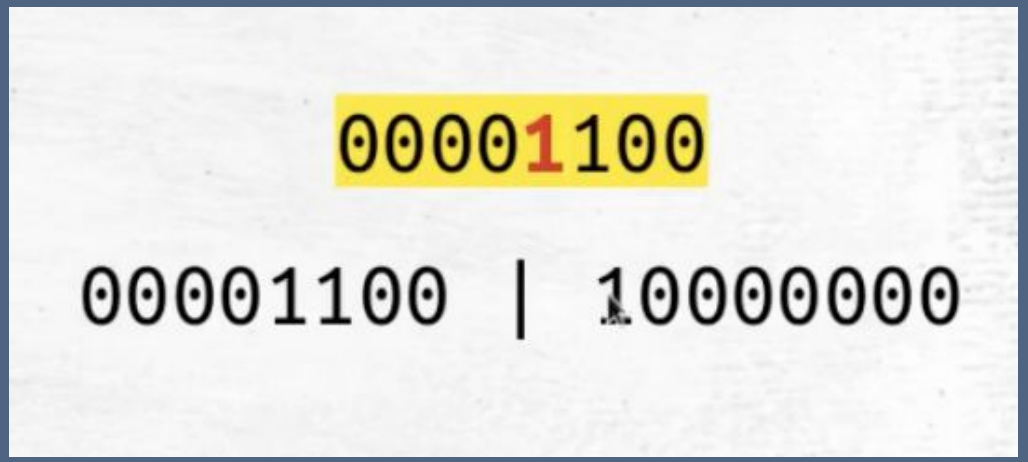
```
int x = 0;
x = x | 4;
x = x | 8;
x = x | 128;
```

Operadores de bits

Ligando e desligando bits

2^0	=	1	=	00000001_2
2^1	=	2	=	00000010_2
2^2	=	4	=	00000100_2
2^3	=	8	=	00001000_2
2^4	=	16	=	00010000_2
2^5	=	32	=	00100000_2
2^6	=	64	=	01000000_2
2^7	=	128	=	10000000_2

```
int x = 0;
x = x | 4;
x = x | 8;
x = x | 128;
```



Operadores de bits

Ligando e desligando bits

2^0	=	1	=	00000001_2
2^1	=	2	=	00000010_2
2^2	=	4	=	00000100_2
2^3	=	8	=	00001000_2
2^4	=	16	=	00010000_2
2^5	=	32	=	00100000_2
2^6	=	64	=	01000000_2
2^7	=	128	=	10000000_2

```
int x = 0;
x = x | 4;
x = x | 8;
x = x | 128;
```

Operadores de bits

Ligando e desligando bits

2^0	=	1	=	00000001_2
2^1	=	2	=	00000010_2
2^2	=	4	=	00000100_2
2^3	=	8	=	00001000_2
2^4	=	16	=	00010000_2
2^5	=	32	=	00100000_2
2^6	=	64	=	01000000_2
2^7	=	128	=	10000000_2

```
int x = 39;
x = x & ~ 4;
x = x & ~ 2;
x = x & ~ 1;
```

Operadores de bits

Ligando e desligando bits

2^0	=	1	=	00000001_2
2^1	=	2	=	00000010_2
2^2	=	4	=	00000100_2
2^3	=	8	=	00001000_2
2^4	=	16	=	00010000_2
2^5	=	32	=	00100000_2
2^6	=	64	=	01000000_2
2^7	=	128	=	10000000_2

```
int x = 39;
x = x & ~ 4;
x = x & ~ 2;
x = x & ~ 1;
```

00100111

Operadores de bits

Ligando e desligando bits

2^0	=	1	=	00000001_2
2^1	=	2	=	00000010_2
2^2	=	4	=	00000100_2
2^3	=	8	=	00001000_2
2^4	=	16	=	00010000_2
2^5	=	32	=	00100000_2
2^6	=	64	=	01000000_2
2^7	=	128	=	10000000_2

```
int x = 39;
x = x & ~ 4;
x = x & ~ 2;
x = x & ~ 1;
```

```
00100111

00100111 & 11111011
```

Operadores de bits

Ligando e desligando bits

2^0	=	1	=	00000001_2
2^1	=	2	=	00000010_2
2^2	=	4	=	00000100_2
2^3	=	8	=	00001000_2
2^4	=	16	=	00010000_2
2^5	=	32	=	00100000_2
2^6	=	64	=	01000000_2
2^7	=	128	=	10000000_2

```
int x = 39;
x = x & ~ 4;
x = x & ~ 2;
x = x & ~ 1;
```

Operadores de bits

Ligando e desligando bits

2^0	=	1	=	00000001_2
2^1	=	2	=	00000010_2
2^2	=	4	=	00000100_2
2^3	=	8	=	00001000_2
2^4	=	16	=	00010000_2
2^5	=	32	=	00100000_2
2^6	=	64	=	01000000_2
2^7	=	128	=	10000000_2

```
int x = 39;
x = x & ~ 4;
x = x & ~ 2;
x = x & ~ 1;
```

00100011

00100011 & 11111101

Operadores de bits

Ligando e desligando bits

2^0	=	1	=	00000001_2
2^1	=	2	=	00000010_2
2^2	=	4	=	00000100_2
2^3	=	8	=	00001000_2
2^4	=	16	=	00010000_2
2^5	=	32	=	00100000_2
2^6	=	64	=	01000000_2
2^7	=	128	=	10000000_2

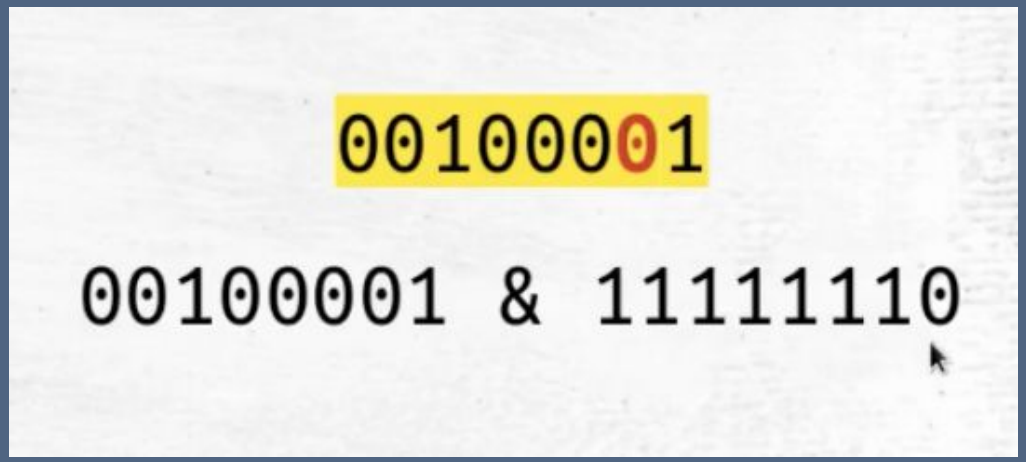
```
int x = 39;
x = x & ~ 4;
x = x & ~ 2;
x = x & ~ 1;
```

Operadores de bits

Ligando e desligando bits

2^0	=	1	=	00000001_2
2^1	=	2	=	00000010_2
2^2	=	4	=	00000100_2
2^3	=	8	=	00001000_2
2^4	=	16	=	00010000_2
2^5	=	32	=	00100000_2
2^6	=	64	=	01000000_2
2^7	=	128	=	10000000_2

```
int x = 39;
x = x & ~ 4;
x = x & ~ 2;
x = x & ~ 1;
```



Operadores de bits

Ligando e desligando bits

2^0	=	1	=	00000001_2
2^1	=	2	=	00000010_2
2^2	=	4	=	00000100_2
2^3	=	8	=	00001000_2
2^4	=	16	=	00010000_2
2^5	=	32	=	00100000_2
2^6	=	64	=	01000000_2
2^7	=	128	=	10000000_2

```
int x = 39;
x = x & ~ 4;
x = x & ~ 2;
x = x & ~ 1;
```

Operadores de bits

Como descobrir então se um bit está ligado?

2^0	=	1	=	00000001_2
2^1	=	2	=	00000010_2
2^2	=	4	=	00000100_2
2^3	=	8	=	00001000_2
2^4	=	16	=	00010000_2
2^5	=	32	=	00100000_2
2^6	=	64	=	01000000_2
2^7	=	128	=	10000000_2

00100100

Operadores de bits

Ligando e desligando bits

2^0	=	1	=	00000001_2
2^1	=	2	=	00000010_2
2^2	=	4	=	00000100_2
2^3	=	8	=	00001000_2
2^4	=	16	=	00010000_2
2^5	=	32	=	00100000_2
2^6	=	64	=	01000000_2
2^7	=	128	=	10000000_2

00100100

bit-0 está ligado?

00100100 & 00000001 ↔ 0

Operadores de bits

Ligando e desligando bits

2^0	=	1	=	00000001_2
2^1	=	2	=	00000010_2
2^2	=	4	=	00000100_2
2^3	=	8	=	00001000_2
2^4	=	16	=	00010000_2
2^5	=	32	=	00100000_2
2^6	=	64	=	01000000_2
2^7	=	128	=	10000000_2

```
00100100
```

bit-0 está ligado?

```
00100100 & 00000001 ↔ 0
```

bit-1 está ligado?

```
00100100 & 00000010 ↔ 0
```

Operadores de bits

Ligando e desligando bits

2^0	=	1	=	00000001_2
2^1	=	2	=	00000010_2
2^2	=	4	=	00000100_2
2^3	=	8	=	00001000_2
2^4	=	16	=	00010000_2
2^5	=	32	=	00100000_2
2^6	=	64	=	01000000_2
2^7	=	128	=	10000000_2

```
00100100
```

bit-0 está ligado?

```
00100100 & 00000001 ↔ 0
```

bit-1 está ligado?

```
00100100 & 00000010 ↔ 0
```

bit-2 está ligado?

```
00100100 & 00000100 ↔ 1
```

Operadores de bits

Ligando e desligando bits

2^0	=	1	=	00000001_2
2^1	=	2	=	00000010_2
2^2	=	4	=	00000100_2
2^3	=	8	=	00001000_2
2^4	=	16	=	00010000_2
2^5	=	32	=	00100000_2
2^6	=	64	=	01000000_2
2^7	=	128	=	10000000_2

```
00100100
```

bit-0 está ligado?

```
00100100 & 00000001 ↔ 0
```

bit-1 está ligado?

```
00100100 & 00000010 ↔ 0
```

bit-2 está ligado?

```
00100100 & 00000100 ↔ 1
```

bit-3 está ligado?

```
00100100 & 00001000 ↔ 0
```

Operadores de bits

Dado um número inteiro *x*, como podemos saber se o “i-ésimo” bit de *x* vale 1?

Como podemos “ligar” o “i-ésimo” bit de um número *x*?

Como podemos “desligar” o “i-ésimo” bit de um número *x*?

Como podemos “imprimir” os bits de um número?

Operadores de bits

Dado um número inteiro *x*, como podemos saber se o "i-ésimo" bit de *x* vale 1?

Basta verificar se "x & (1<<i)" é diferente de 0.

Como podemos "ligar" o "i-ésimo" bit de um número *x*?

Basta fazer: x = x | (1<<i); (ou: x |= 1<<i;)

Como podemos "desligar" o "i-ésimo" bit de um número *x*?

Basta fazer: x = x & ~(1<< i);

Como podemos "imprimir" os bits de um número *x*?

Basta varrer os bits e imprimir 1 se o teste "(x & (1<<i)) != 0" for verdadeiro e 0 caso contrário. Veja o código a seguir.

```
void imprimeBits(unsigned int n) {
    for(int i=31;i>=0;i--) {
        if (i!=0 && i%8 ==0) cout << " ";
        if ( (n & (1<<i)) != 0)
            cout << 1;
        else
            cout << 0;
    }
    cout << endl;
}
int main() {
    unsigned int a = 3;
    unsigned int b = 5;

    imprimeBits(b);
    imprimeBits(a);
    imprimeBits(~a);
// ...
```

```
0000000 00000000 00000000 000000101
0000000 00000000 00000000 000000011
1111111 11111111 11111111 111111100
```

Operadores de bits

Alguns “truques”:

Como podemos “desligar” o bit mais à direita de um número *x*?

Como podemos descobrir se um número é da forma 2^n?

Operadores de bits

Como podemos “desligar” o bit mais à direita de um número *x*?

Basta fazer: x & (x-1)

Ex: x = 10100 , x-1 = 10011 $\longrightarrow$ x & (x-1) = 10000

Como podemos descobrir se um número é da forma 2^n?

Basta “desligar” o bit mais à direita e ver se o resultado é 0 (além disso, é necessário ver se o número original era 0).

Ex: x=01000 , x-1 = 001111 $\longrightarrow$ x & (x-1) = 000000

Operadores de bits

Como podemos descobrir se um número é da forma 2^n-1?

Como podemos isolar o bit mais à direita de um número x ?

Operadores de bits

Como podemos descobrir se um número é da forma 2^n-1?

Basta fazer o teste: x & (x+1)

Ex: x = 000111 , x+1 = 001000 $\rightarrow$ x&(x+1) = 000000

Como podemos isolar o bit mais à direita de um número x ?

Basta fazer: x & ~(x-1)

Ex: x= 001010, x-1=001001 $\rightarrow$ 000010

Operadores de bits

Exemplos de aplicação: armazenar “conjuntos”, vetores de booleanos, etc.

Por exemplo, o número binário 9 poderia ser utilizado para indicar que o elemento “0” e o elemento “3” estão no conjunto.

$9_{10} \rightarrow 00000000\ 00000000\ 00000000\ 00001001_2$

Com isso, pode-se, por exemplo, utilizar um número inteiro de 32 bits para representar 32 possíveis elementos em um conjunto.

Quais seriam as principais vantagens disso? Alguma desvantagem?

Como poderíamos representar conjuntos com capacidade maior?

Operadores de bits

Com isso, pode-se, por exemplo, utilizar um número inteiro de 32 bits para representar 32 possíveis elementos em um conjunto.

Quais seriam as principais vantagens disso? Alguma desvantagem?

- Economia de memória.
  - Por exemplo, um único inteiro (4 bytes) poderia substituir um array com 32 bools (32 bytes).
- Eficiência em algumas situações.
  - Por exemplo, pode-se verificar se o conjunto está vazio simplesmente o comparando com o número 0.
- Em algumas situações pode ser ineficiente:
  - Por exemplo, para verificar se um determinado número pertence ao conjunto seria necessário realizar algumas operações sobre bits.

Como poderíamos representar conjuntos com capacidade maior?

- R: basta utilizar um tipo que utiliza mais bits ou então utilizar um array de inteiros.