Aula 2 - Escovando bits com C#

Docupedia Export

Author:Balem Luis (CtP/ETS)
Date:29-Mar-2023 17:22

Table of Contents

- Operadores: Uma visão em binário
- Estruturas Básicas
- Funções
- Exercícios Propostos 1

1 Operadores: Uma visão em binário

A compreensão de números binários é importante para compreender algumas operações, bem como permitir que certas coisas sejam feitas mais eficientemente.
Desta forma, vamos ver algumas das operações mostradas no final da Aula 1 olhando para números binários.
Primariamente, precisamos entender bem o que são números binários.
Um número binário é uma sequência de bits, que são nada mais, nada menos, que 0's e 1's. Vamos usar bastante a noção de byte daqui para frente, que é o conjunto de 8 bits, indo de 0 (00000000) a 255 (11111111) bem como o tipo definido em C#.
Assim como em decimal, que tem os digitos de 0 a 9, ao olhar o próximo número apenas somamos 1 ao digito mais a direita:
34 -> 35
O sucessor de 34 é 35, pois 4 + 1 é 5. Note que quando passamos do último digito (9) voltamos para o primeiro (0) e somamos 1 a casa da esquerda:
49 -> 50
O sucessor de 49 é 50, pois 9 + 1 = 10, assim o 9 torna-se 0 e somamos 1 ao 4 que torna-se 5. Da mesma forma:
9199 -> 9200
Em binário, os mesmos mecânismos acontecem. Porém, só temos 2 digitos, assim:
000 -> 001 001 -> 010 010 -> 011 100 -> 101 101 -> 110 110 -> 111
Pela tabela podemos ver como isso ocorre:

Binário	Decimal
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Note que existem formas mais fáceis de realizar conversões binário-decimal após compreender seu funcionamento. Então vejamos elas: Vamos tentar converter 10100101 de binário para decimal. Basta ignorar os 0's e para cada 1 você soma uma potência de 2. Essa potência deve ser correspondente a posição de cada 1 no binário. Assim a contribuição dos 0's é zero e a dos 1's depende da sua posição no binário. Observe:

digito	**7°**	**6º**	**5°**	**4º**	**3°**	**2º**	**1°**	**0º**	**resultado**
binário	1	0	1	0	0	1	0	1	
potência	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
resultado	128	64	32	16	8	4	2	1	
contribuição	128	0	32	0	0	4	0	1	175

Da mesma forma, podemos converter de um decimal para um binário apenas buscando os bits onde a conversão de correta. Interessantemente, só existe uma combinação de 0's e 1's para cada decimal. Um truque para realizar essa operação é ler da esquerda para direita, assim se a soma do bit não extrapolar o decimal, consideramos o mesmo como 1. Observe a conversão de 202 para binário:

passo	**binário**	**conversão**	**é maior que 202**	**ação**
1	1_____	128	não	próximo bit
2	11_____	196	não	próximo bit
3	111____	228	sim	reverte
4	110____	196	não	próximo bit
5	1101___	204	sim	reverte
6	1100___	196	não	próximo bit
7	11001__	200	não	próximo bit

passo	binário	conversão	é maior que 202	ação
8	110011_	202	não	próximo bit
9	1100111	203	sim	reverte
10	1100110	202	não	fim

Podemos observar agora algumas das operações do C# com um pouco de distinção e pensar nos números binários a partir disto. Vamos começar com o Shift ou deslocamento binário:
00000001 << 1 = 00000010
Basicamente, temos o deslocamento uma casa a esquerda. Outro exemplo:
01010101 << 1 = 10101010
Isso ocorre a todos os bits. Note que um deslocamento maior que 1 pode ocorrer:
00000001 << 7 = 10000000
Outra noção importante é que se um número sai para fora dos bits (depende se for um byte, short, int ou long) ele simplesmente desaparece:
11110000 << 2 = 11000000
O Shift também pode ser aplicado a direita:
11110000 >> 2 = 00111100
Ele também segue as mesmas regras que o outro shift:
11111111 >> 3 = 00011111
Você ainda pode escrever em código C#:

```
byte b = 4;
b <<= 2;
```

Da mesma forma como se usa +=, -=, *=, /= em C# e outras linguagens.
Você também pode usar operações lógicas fazendo operações bit-a-bit:
11100000 | 00000111 = 11100111 (Ou, aplicado bit-a-bit)
11111000 & 00011111 = 00011000 (E, aplicado bit-a-bit)
11110000 ^ 11001100 = 00111100 (Ou exclusivo, aplicado bit-a-bit)
~00001111 = 11110000 (Não, aplicado bit-a-bit)
É importante salientar que os números com sinal são representados na forma complemento de 2. Isso significa, entre outras coisas, que o primeiro bit representa se o número é negativo ou não. Logo, caso alguma operação que você realizar jogue um 1 no primeiro bit o número ficará negativo.
Outro ponto rápido: existe uma diferença entre && e || de & e |. A repetição do símbolo denota circuito-curto, o que significa que a depender do primeiro valor, o segundo não é considerado. Assim true || false não considera a segunda parte da expressão enquanto o circuito-completo true | false considera a expressão como um todo. Com isso &&/|| não pode ser usado em tipos binários, mas pode evitar cálculos desnecessários e é bom ser usado em condicionais.

2 Estruturas Básicas

Como você deve ter aprendido em outras linguagens, para programar é necessário de algumas estruturas para controlar o fluxo de execução. Agora, vamos ver como produzi-las em C#:

- if

```
1  using static System.Console;
2
3  int idade = int.Parse(ReadLine());
4  if (idade > 17)
5  {
6      WriteLine("É maior de idade.");
7  }
8  else if (idade > 15)
9  {
10     WriteLine("Não é maior de idade. Mas em alguns países já pode dirigir");
11 }
12 else
13 {
14     WriteLine("Menor de idade.");
15 }
```

Então, caso a idade digitada seja 18 para cima a condição no primeiro escopo é executada. Note que a segunda condição também seria verdadeira, porém, apenas uma cláusula é executada por vez. Caso nem o 'if', nem o 'else if' seja executado, executamos o 'else'.

- switch

```
1  using static System.Console;
2
3  int idade = int.Parse(ReadLine());
4  switch (idade)
5  {
6      case 18:
7          WriteLine("É maior de idade.");
8      break;
9
10     case 19:
11         WriteLine("Está ficando velho!");
12     goto case 18;
13
```

```
    case 16:
    case 17:
        WriteLine("Não é maior de idade. Mas em alguns países já pode dirigir.");
    break;

    default:
        WriteLine("Menor de idade.");
    break;
}
```

O switch usa uma forma diferente de execução, tornando-o mais rápido a depender da situação. Se existem poucos intervalos e muitas opções distintas, o switch é uma boa opção. Você precisa fechar todos os casos com break, return ou goto. O mais comum é se utilizar break, mas como pode observar, você pode usar goto para que 2 casos diferentes executem. Você também pode usar 2 cases seguidos para indicar o mesmo comportamento para diferentes valores da variável.

- while

```
using static System.Console;

WriteLine("Fatorial de qual valor você irá querer?");
int n = int.Parse(ReadLine());

int fatorial = 1;

while (n > 1)
{
    fatorial *= n;
    n--;
}
using static System.Console;

int numeroSecreto = 540;
int numero;

do
{
    WriteLine("Tente adivinhar o número secreto...");
    numero = int.Parse(ReadLine());

    if (numero > numeroSecreto)
        WriteLine("O numero secreto é menor");
```

```
25      else if (numero < numeroSecreto)
26          WriteLine("O numero secreto é maior");
27
28  } while (numero != numeroSecreto);
```

No While e Do...While temos as mais básicas estruturas de repetição. O bloco é executado apenas enquanto a condição seja verdadeira. O que muda é apenas o momento em que a condição é considerada.

- for

```
1  using static System.Console;
2
3  int[] vetor = new int[50];
4
5  WriteLine("Digite 50 valores");
6  for (int i = 0; i < vetor.Lenth; i++)
7  {
8      vetor[i] = int.Parse(ReadLine());
9  }
```

Também temos o For que é ótimo para acessar vetores pois o mesmo tem inicialização, condição e incremento no final de cada loop. É bom lembrar que também temos coisas como break, que interrompe um loop e continue, que pula para o próximo loop.

3 Funções

A partir de agora, vamos parar de fazer tantas operações fora de funções. Funções são uma das estruturas mais básicas e importantes dentro da programação e por isso vamos aprender a utilizá-las em C#. A estrutura básica é:
retorno nome(parâmetros) { implementação }
Em C# você não precisa declarar a função antes de usá-la (ela pode estar depois no código), mas você precisa seguir algumas regras:

- Se a função pede algum retorno, você deve retorná-lo, caso contrário você terá um erro de compilação.
- Você deve garantir que todos os caminhos retornam algo.
- Cada parâmetro é como uma variável que deve ser enviado na chamada, a não ser que seja opicional.
- Se você declarar uma parâmetro com mesmo nome de outra variável no programa, ao usar a variável estará usando a declarada dentro da função.

A seguir exemplos:

```
1   int value = modulo(-3);
2
3   int modulo(int i)
4   {
5       if (i < 0)
6           return -i;
7       return i;
8   }
9   using static System.Console;
10
11  WriteLine("Fatorial de qual valor você irá querer?");
12  int n = int.Parse(ReadLine());
13
14  WriteLine("O resultado é: " + fatorial(n));
15
16  int fatorial(int n)
17  {
18      if (n < 2)
19          return 1;
20
21      return n * fatorial(n - 1);
22  }
23  using static System.Console;
24
25  mostreUmTexto("Hello World!");
26  mostreUmTexto(); // mostrará 'Olá mundo'
```

```
27
28  void mostreUmTexto(string texto = "Olá Mundo!") // não retorna nada
29  {
30      WriteLine("Texto:");
31      WriteLine(texto);
32      //return; // pode ter um return aqui, mas o mesmo não é obrigatório
33  }
```

4 Exercícios Propostos 1

- a. Converta os seguintes valores:
  - a) 101 para binário
  - b) 102 para binário
  - c) 103 para binário
  - d) 254 para binário
  - e) 11111100 para decimal
  - f) 10000000 para decimal
  - e) 11000001 para decimal
- a. Escreva uma função C# que converta um número decimal para binário (retornando uma string).
- a. Faça um programa que dado dois números dados pelo usuário a e b, desenhe na tela, em binário, o resultado de a << b;