04 - 编码与位运算

C++ 程序设计进阶

SOJ 信息学竞赛教练组

2024 年 7 月 21 日

目录

进制

- 进制是人为定义的带进位的计数方法

进制

- 进制是人为定义的带进位的计数方法
- X 进制

进制

- 进制是人为定义的带进位的计数方法
- X 进制
  - 逢 X 进一，由 X 个**数码**组成，分别代表 $0 \sim X-1$ 这 X 个数字

进制

- 进制是人为定义的带进位的计数方法
- X 进制
  - 逢 X 进一，由 X 个**数码**组成，分别代表 $0 \sim X-1$ 这 X 个数字
  - 从低位到高位的权重依次是 $X^0, X^1, X^2, X^3, ...$

二进制转十进制

- 按权展开求和

$$(10011)_2 = ?$$

二进制转十进制

- 按权展开求和

$$
\begin{aligned}
(10011)_2 &= 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\
&= 19
\end{aligned}
$$

二进制转十进制

```cpp
// #include ...

// 函数功能：返回 n 位二进制数 b[0 ~ n-1] 的十进制数值
int bin2dec(int b[], int n) {
  int sum = 0, w = 1;
  for (int i = n - 1; i >= 0; i--) {
    sum += b[i] * w;
    w *= 2;
  }
  return sum;
}

int bin[35];

int main() {
  int n;
  cin >> n;
  for (int i = 0; i < n; i++) cin >> bin[i];
  cout << bin2dec(bin, n) << endl;
  return 0;
}
```

十进制转二进制

- 除二取余法
  - 对一个数值 `n` 在循环中重复 `n % 2, n /= 2;` 的操作，可以得到这个数值二进制逆序的每一位
  - 取余二除以二，逆序输出

十进制转二进制

```cpp
// #include ...

int bin[35];
// 函数功能：输出十进制 x 的二进制表示
void dec2bin(int x) {
  int siz = 0; // 记录二进制的位数
  do {
    bin[siz] = x % 2;
    siz++;
    x /= 2;
  } while (x);
  // 逆序输出
  for (int i = siz - 1; i >= 0; i--) cout << bin[i];
  cout << endl;
}

int main() {
  int x;
  cin >> x;
  dec2bin(x);
  return 0;
}
```

十进制转二进制

```cpp
// #include ...

int bin[35];
// 函数功能：输出十进制 x 的二进制表示
void dec2bin(int x) {
  int siz = 0; // 记录二进制的位数
  do {
    bin[siz] = x % 2;
    siz++;
    x /= 2;
  } while (x);
  // 逆序输出
  for (int i = siz - 1; i >= 0; i--) cout << bin[i];
  cout << endl;
}

int main() {
  int x;
  cin >> x;
  dec2bin(x);
  return 0;
}
```

目录

数据储存单位

- 比特 (Bit)
  - 比特是数据储存的最小单位
  - 二进制的一位，叫做 $1\ Bit$

数据储存单位

- 比特 (Bit)
  - 比特是数据储存的最小单位
  - 二进制的一位，叫做 $1\ Bit$
- 字节 ($Byte$)
  - 字节是用于计量存储容量的一种计量单位
  - $1\ Byte = 8\ Bit$

变量的数据范围

- $1\ Bit$ 可以表示出 0 和 1 这 2 个二进制码
- $2\ Bit$ 可以表示出 $00, 01, 10, 11$ 这 4 个二进制码
- $3\ Bit$ 可以表示出 $000, 001, 010, 011, 100, 101, 110, 111$ 这 8 个二进制码
- $32\ Bit$ 可以表示多少个不同数字?
  - 2^{32} 个
  - 可表示的数的范围是 ?

变量的数据范围

- $1\ Bit$ 可以表示出 0 和 1 这 2 个二进制码
- $2\ Bit$ 可以表示出 $00, 01, 10, 11$ 这 4 个二进制码
- $3\ Bit$ 可以表示出 $000, 001, 010, 011, 100, 101, 110, 111$ 这 8 个二进制码
- $32\ Bit$ 可以表示多少个不同数字?
  - 2^{32} 个
  - 可表示的数的范围是 $0 \sim 2^{32} - 1$

编码

负数如何转换为二进制码存储?

编码

- 编码是信息从一种形式或格式转换为另一种形式的过程，一般是指用预先规定的方法将文字、数字或其它对象编成数码
- 整数编码的发展过程：原码、反码、补码
  - 原码、反码、补码的形式都是 **符号位 + 数值位**
  - 规定二进制最高位是符号位，0 表示正，1 表示负

原码

- 原码表示法
  - 先写出数值的绝对值对应的二进制表示
  - 在数值位前面（最高位）增加一位符号位，0 表示正，1 表示负
- 9 的二进制表示：1001
- $+9$ 的 8 位原码为：00001001
- -9 的 8 位原码为：10001001

```
00001001
10001001
```

原码

- 原码表示法
  - 先写出数值的绝对值对应的二进制表示
  - 在数值位前面（最高位）增加一位符号位，0 表示正，1 表示负
- 9 的二进制表示：1001
- +9 的 8 位原码为：00001001
- −9 的 8 位原码为：10001001

```
1 00001001
2 10001001
```

符号位

原码

- 原码表示法
  - 先写出数值的绝对值对应的二进制表示
  - 在数值位前面（最高位）增加一位符号位，0 表示正，1 表示负
- 9 的二进制表示：1001
- +9 的 8 位原码为：00001001
- −9 的 8 位原码为：10001001

```
1 00001001
2 10001001
```

符号位　数值位

原码

- 优点
  - 解决了负数的存储问题
- 缺点
  - 0 存在两种形式: $(00000000)_2$ 和 $(10000000)_2$
  - 加减数值运算不符合竖式运算的规律
  - 例如:

$$
\begin{aligned}
(1)_{10} + (-1)_{10} &= (00000001)_2 + (10000001)_2 \\
&= (10000010)_2 \\
&= (-2)_{10}
\end{aligned}
$$

原码

- 优点
  - 解决了负数的存储问题
- 缺点
  - 0 存在两种形式: $(00000000)_2$ 和 $(10000000)_2$
  - 加减数值运算不符合竖式运算的规律
  - 例如:

$$
\begin{aligned}
(1)_{10} + (-1)_{10} &= (00000001)_2 + (10000001)_2 \\
&= (10000010)_2 \\
&= (-2)_{10}
\end{aligned}
$$

运算结果是错误的

反码

- 反码表示法
  - 正数的反码与原码相同
  - 负数的反码与原码相比，符号位相同，其他位相反
- $+9$ 的 8 位原码为：　　，反码为
- -9 的 8 位原码为：　　，反码为

反码

- 反码表示法
  - 正数的反码与原码相同
  - 负数的反码与原码相比，符号位相同，其他位相反
- $+9$ 的 8 位原码为：00001001，反码为
- -9 的 8 位原码为：　　，反码为

反码

- 反码表示法
  - 正数的反码与原码相同
  - 负数的反码与原码相比，符号位相同，其他位相反
- $+9$ 的 8 位原码为：00001001，反码为 00001001
- -9 的 8 位原码为：　　，反码为

反码

- 反码表示法
  - 正数的反码与原码相同
  - 负数的反码与原码相比，符号位相同，其他位相反
- $+9$ 的 8 位原码为：00001001，反码为 00001001
- -9 的 8 位原码为：10001001，反码为

反码

- 反码表示法
  - 正数的反码与原码相同
  - 负数的反码与原码相比，符号位相同，其他位相反
- $+9$ 的 8 位原码为：00001001，反码为 00001001
- -9 的 8 位原码为：10001001，反码为 11110110

反码

- 优点
  - 加减数值运算部分符合竖式运算的规律
  - 例如：

$$\begin{aligned}(1)_{10} + (-1)_{10} &= (00000001)_2 + (11111110)_2 \\ &= (11111111)_2 \\ &= (-0)_{10}\end{aligned}$$

- 缺点
  - 0 存在两种形式：$(00000000)_2$ 和 $(11111111)_2$
  - 加减数值运算不完全符合竖式运算的规律

反码计算的规律

- 不符合竖式运算规律

$$
\begin{aligned}
(5)_{10} + (-4)_{10} &= (00000101)_2 + (11111011)_2 \\
&= (100000000)_2 \text{ 9 位数，溢出} \\
&= (00000000)_2 \\
&= (0)_{10}
\end{aligned}
$$

$$
\begin{aligned}
(4)_{10} + (-2)_{10} &= (00000100)_2 + (11111101)_2 \\
&= (100000001)_2 \text{ 9 位数，溢出} \\
&= (00000001)_2 \\
&= (1)_{10}
\end{aligned}
$$

- 计算结果比正确答案少了 1
- 负数在反码的基础上加 1，可解决这个问题

补码

- 补码表示法
  - 正数的补码、反码、原码都相同
  - 负数的补码是它的反码 $+1$
- $+9$ 的 8 位原码为 00001001，反码为 00001001，
  补码为 00001001
- -9 的 8 位原码为 10001001，反码为 11110110，
  补码为 11110111

补码

- 优点
  - 每个数值均有唯一一种表示
  - 加减数值运算符合竖式运算规律，可以将符号位和数值位统一处理
- 在计算机系统中，通常使用补码来表示和存储整数
  - `int` 数据范围是 $-2^{31} \sim 2^{31} - 1$
  - `long long` 数据范围是 $-2^{63} \sim 2^{63} - 1$

小结

	正数	负数
原码	符号位为 0, 数值位为正常二进制表示	符号位为 1, 数值位为正常二进制表示
反码	与原码相同	数值位在原码的基础上取反
补码	与原码相同	数值位在反码的基础上 $+1$

随堂练习

选择题

- 十进制数 -14 的 8 位二进制补码是

  A. 10001110

  B. 11110001

  C. 11110010

  D. 01110010

随堂练习

选择题

- 十进制数 -14 的 8 位二进制补码是

  A. 10001110

  B. 11110001

  C. 11110010

  D. 01110010

随堂练习

选择题

- 十进制数 -14 的 8 位二进制补码是

  A. 10001110

  B. 11110001

  C. 11110010

  D. 01110010

- 解法
  - 原码：
  - 反码：
  - 补码：

随堂练习

选择题

- 十进制数 -14 的 8 位二进制补码是

  A. 10001110

  B. 11110001

  C. 11110010

  D. 01110010

- 解法
  - 原码：10001110 (注意负数符号位是 1)
  - 反码：
  - 补码：

随堂练习

选择题

- 十进制数 -14 的 8 位二进制补码是

  A. 10001110

  B. 11110001

  C. 11110010

  D. 01110010

- 解法
  - 原码：10001110 (注意负数符号位是 1)
  - 反码：11110001
  - 补码：

随堂练习

选择题

- 十进制数 -14 的 8 位二进制补码是

  A. 10001110

  B. 11110001

  C. 11110010

  D. 01110010

- 解法
  - 原码：10001110 (注意负数符号位是 1)
  - 反码：11110001
  - 补码：11110010

原码和补码的转换

如何将给定的补码，转换回原码的形式?

原码和补码的转换

- 正数的原码、反码、补码都相同
- 负数原码转补码
  - 符号位不变、数值位取反（0 变 1，1 变 0）转为反码，反码 $+1$ 转为补码
- 负数补码转原码
  - 补码 -1 转为反码，反码符号位不变、数值位取反转为原码

原码和补码的转换

选择题

- 在 8 位二进制补码中，00101011 表示的数是十进制下的

  A. 43

  B. 85

  C. -43

  D. -84

原码和补码的转换

选择题

- 在 8 位二进制补码中，00101011 表示的数是十进制下的

  A. 43

  B. 85

  C. -43

  D. -84

原码和补码的转换

选择题

- 在 8 位二进制补码中，00101011 表示的数是十进制下的

  A. 43

  B. 85

  C. -43

  D. -84

- 解法

原码和补码的转换

选择题

- 在 8 位二进制补码中，00101011 表示的数是十进制下的

  A. 43

  B. 85

  C. -43

  D. -84

- 解法

  1. 观察到符号位是 0 所以是正数

原码和补码的转换

选择题

- 在 8 位二进制补码中，00101011 表示的数是十进制下的

  A. 43

  B. 85

  C. -43

  D. -84

- 解法

  1. 观察到符号位是 0 所以是正数
  2. 正数的原码、反码、补码一致

原码和补码的转换

选择题

- 在 8 位二进制补码中，00101011 表示的数是十进制下的

  A. 43

  B. 85

  C. -43

  D. -84

- 解法

  1. 观察到符号位是 0 所以是正数
  2. 正数的原码、反码、补码一致
  3. 将原码 01010101 按权展开求和为十进制即可

原码和补码的转换

选择题

- 在 8 位二进制补码中，10101011 表示的数是十进制下的

A. 43

B. -85

C. -43

D. -84

原码和补码的转换

选择题

- 在 8 位二进制补码中，10101011 表示的数是十进制下的

A. 43

B. -85

C. -43

D. -84

原码和补码的转换

选择题

- 在 8 位二进制补码中，10101011 表示的数是十进制下的

  A. 43

  B. −85

  C. −43

  D. −84

- 解法
  - 补码：
  - 反码：
  - 原码：

原码和补码的转换

选择题

- 在 8 位二进制补码中，10101011 表示的数是十进制下的

  A. 43

  B. −85

  C. −43

  D. −84

- 解法
  - 补码：10101011
  - 反码：
  - 原码：

原码和补码的转换

选择题

- 在 8 位二进制补码中，10101011 表示的数是十进制下的

  A. 43

  B. −85

  C. −43

  D. −84

- 解法
  - 补码：10101011
  - 反码：10101010
  - 原码：

原码和补码的转换

选择题

- 在 8 位二进制补码中，10101011 表示的数是十进制下的

  A. 43

  B. −85

  C. −43

  D. −84

- 解法
  - 补码：10101011
  - 反码：10101010
  - 原码：11010101

原码和补码的转换

选择题

- 在 8 位二进制补码中，10101011 表示的数是十进制下的

  A. 43

  B. -85

  C. -43

  D. -84

- 解法
  - 补码：10101011
  - 反码：10101010
  - 原码：11010101
  - 对原码的数值位进行按权展开求和，得到 85，符号位是 1，所以是负数

目录

字符

计算机只能存储二进制内容，
那如何存储字母、符号等文本信息呢?

字符

- 字符是字母、数字符号、标点符号等的统称
- 计算机储存字符，通常是将字符编码成一个整数，再将这个整数转换为对应的二进制进行储存
- C++ 中用 `char` 类型储存字符，将字符编码成整数的方式是 ASCII 码（定义了 128 个字符，用 $0 \sim 127$ 表示）

字符编码 - ASCII 码

- `'0'~'9'`: $48 \sim 57$
- `'A'~'Z'`: $65 \sim 90$
- `'a'~'z'`: $97 \sim 122$
- 储存字符时，本质上是储存了它对应的整数

字符编码 - ASCII 码

- 代码示例

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6    char ch = 'A';
7    cout << (int)ch << endl;
8    cout << (char)66 << endl;
9    return 0;
10 }
```

- 输出

字符编码 - ASCII 码

- 代码示例

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6    char ch = 'A';
7    cout << (int)ch << endl;
8    cout << (char)66 << endl;
9    return 0;
10 }
```

- 输出

  65

字符编码 - ASCII 码

- 代码示例

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6    char ch = 'A';
7    cout << (int)ch << endl;
8    cout << (char)66 << endl;
9    return 0;
10 }
```

- 输出

```
65
B
```

字符类型的运算

- char 类型本质上也是整数类型，因此也支持算术运算、布尔运算
  - char 类型数据参与运算时，会隐式转换为 int 类型，然后进行计算
  - 注意 char 类型变量占 1 字节，能储存的数据范围是 $-128 \sim 127$，计算结果超过这个范围不能储存在 char 变量中

字符类型的运算

- char 类型本质上也是整数类型，因此也支持算术运算、布尔运算
  - char 类型数据参与运算时，会隐式转换为 int 类型，然后进行计算
  - 注意 char 类型变量占 1 字节，能储存的数据范围是 $-128 \sim 127$，计算结果超过这个范围不能储存在 char 变量中
- 常用场景
  - 字符类型的判断
  - 字符与整数之间的转换
  - 大小写字母转换

字符类型的判断

- 大写字母 $A \sim Z$ 的 ASCII 码是**连续的** $65 \sim 90$，所以可以方便地判断一个 char 类型变量 ch 是否为大写字母类型

字符类型的判断

- 大写字母 $A \sim Z$ 的 ASCII 码是**连续的** $65 \sim 90$，所以可以方便地判断一个 char 类型变量 ch 是否为大写字母类型
  - `if (ch >= 'A' && ch <= 'Z')`
  - `if (ch >= 65 && ch <= 90)`

字符类型的判断

- 大写字母 $A \sim Z$ 的 ASCII 码是**连续的** $65 \sim 90$，所以可以方便地判断一个 char 类型变量 ch 是否为大写字母类型
  - `if (ch >= 'A' && ch <= 'Z')`
  - `if (ch >= 65 && ch <= 90)`
- 判断 ch 是否为小写字母类型

字符类型的判断

- 大写字母 $A \sim Z$ 的 ASCII 码是**连续的** $65 \sim 90$，所以可以方便地判断一个 char 类型变量 ch 是否为大写字母类型
  - `if (ch >= 'A' && ch <= 'Z')`
  - `if (ch >= 65 && ch <= 90)`
- 判断 ch 是否为小写字母类型
  - `if (ch >= 'a' && ch <= 'z')`

字符类型的判断

- 大写字母 $A \sim Z$ 的 ASCII 码是**连续的** $65 \sim 90$，所以可以方便地判断一个 char 类型变量 ch 是否为大写字母类型
  - `if (ch >= 'A' && ch <= 'Z')`
  - `if (ch >= 65 && ch <= 90)`
- 判断 ch 是否为小写字母类型
  - `if (ch >= 'a' && ch <= 'z')`
- 判断 ch 是否为数字类型

字符类型的判断

- 大写字母 $A \sim Z$ 的 ASCII 码是**连续的** $65 \sim 90$，所以可以方便地判断一个 char 类型变量 ch 是否为大写字母类型
  - `if (ch >= 'A' && ch <= 'Z')`
  - `if (ch >= 65 && ch <= 90)`
- 判断 ch 是否为小写字母类型
  - `if (ch >= 'a' && ch <= 'z')`
- 判断 ch 是否为数字类型
  - `if (ch >= '0' && ch <= '9')`

目录

数字字符转换为整数

- 如何将数字字符 `ch` 换算为对应的整数?
  - 即 `'0'`(48) 换算为 0，`'1'`(49) 换算为 1，`'2'`(50) 换算为 2 ……

数字字符转换为整数

- 如何将数字字符 `ch` 换算为对应的整数?
  - 即 `'0'`(48) 换算为 0，`'1'`(49) 换算为 1，`'2'`(50) 换算为 2 ……
  - `(int)ch`

数字字符转换为整数

- 如何将数字字符 `ch` 换算为对应的整数?
  - 即 `'0'`(48) 换算为 0，`'1'`(49) 换算为 1，`'2'`(50) 换算为 2 ……
  - ~~`(int)ch`~~

数字字符转换为整数

- 如何将数字字符 `ch` 换算为对应的整数?
  - 即 `'0'`(48) 换算为 0，`'1'`(49) 换算为 1，`'2'`(50) 换算为 2 ……
  - ~~`(int)ch`~~
  - `ch - '0'`
  - `ch - 48`

字母转换为整数

- 如何计算大写字母字符 `ch` 是第几个字母?
  - 即 `'A'` 换算为 0，`'B'` 换算为 1，`'C'` 换算为 2 ... `'Z'` 换算为 25

字母转换为整数

- 如何计算大写字母字符 `ch` 是第几个字母?
  - 即 `'A'` 换算为 0，`'B'` 换算为 1，`'C'` 换算为 2 ... `'Z'` 换算为 25
  - `ch - 'A'`

字母转换为整数

- 如何计算大写字母字符 `ch` 是第几个字母?
  - 即 `'A'` 换算为 0，`'B'` 换算为 1，`'C'` 换算为 2 ... `'Z'` 换算为 25
  - `ch - 'A'`
- 如何计算小写字母字符 `ch` 是第几个字母?

字母转换为整数

- 如何计算大写字母字符 ch 是第几个字母?
  - 即 'A' 换算为 0，'B' 换算为 1，'C' 换算为 2 ... 'Z' 换算为 25
  - `ch - 'A'`
- 如何计算小写字母字符 ch 是第几个字母?
  - `ch - 'a'`

整数转换为字母

- 如何计算第 x 个大写字母是什么?
  - 即 0 换算为 `'A'`(65)，1 换算为 `'B'`(66)，2 换算为 `'C'`(67) ...

整数转换为字母

- 如何计算第 x 个大写字母是什么?
  - 即 0 换算为 `'A'`(65)，1 换算为 `'B'`(66)，2 换算为 `'C'`(67) ...
  - `x + 'A'`
  - `x + 65`

整数转换为字母

- 如何计算第 x 个大写字母是什么?
  - 即 0 换算为 `'A'`(65)，1 换算为 `'B'`(66)，2 换算为 `'C'`(67) ...
  - `x + 'A'`
  - `x + 65`
- 如何计算第 x 个小写字母是什么?

整数转换为字母

- 如何计算第 x 个大写字母是什么?
  - 即 0 换算为 `'A'`(65)，1 换算为 `'B'`(66)，2 换算为 `'C'`(67) ...
  - `x + 'A'`
  - `x + 65`
- 如何计算第 x 个小写字母是什么?
  - `x + 'a'`

大小写字母换算

- 如何将大写字母 `ch` 换算为对应的小写字母?
  - 即 `'A'` 换算为 `'a'`，`'B'` 换算为 `'b'`，`'C'` 换算为 `'c'` ...

大小写字母换算

- 如何将大写字母 ch 换算为对应的小写字母?
  - 即 'A' 换算为 'a'，'B' 换算为 'b'，'C' 换算为 'c' …

1. 先计算出 ch 是第几个字母：`int x = ch - 'A';`

大小写字母换算

- 如何将大写字母 ch 换算为对应的小写字母?
  - 即 'A' 换算为 'a'，'B' 换算为 'b'，'C' 换算为 'c' ...

1. 先计算出 ch 是第几个字母：`int x = ch - 'A';`
2. 再计算第 x 个小写字母是什么：`x + 'a';`

大小写字母换算

- 如何将大写字母 ch 换算为对应的小写字母?
  - 即 'A' 换算为 'a'，'B' 换算为 'b'，'C' 换算为 'c' ...

1. 先计算出 ch 是第几个字母：`int x = ch - 'A';`
2. 再计算第 x 个小写字母是什么：`x + 'a';`

- 结合上述两步：`ch - 'A' + 'a'`

大小写字母换算

- 如何将大写字母 ch 换算为对应的小写字母?
  - 即 'A' 换算为 'a'，'B' 换算为 'b'，'C' 换算为 'c' ...

  1. 先计算出 ch 是第几个字母：`int x = ch - 'A';`
  2. 再计算第 x 个小写字母是什么：`x + 'a';`

  - 结合上述两步：`ch - 'A' + 'a'`
- 如何将小写字母 ch 换算为对应的大写字母?

大小写字母换算

- 如何将大写字母 ch 换算为对应的小写字母?
  - 即 'A' 换算为 'a'，'B' 换算为 'b'，'C' 换算为 'c' ...

  1. 先计算出 ch 是第几个字母：`int x = ch - 'A';`
  2. 再计算第 x 个小写字母是什么：`x + 'a';`

  - 结合上述两步：`ch - 'A' + 'a'`
- 如何将小写字母 ch 换算为对应的大写字母?
  - `ch - 'a' + 'A'`

字符换算

填空题

- 字符 ch 换算为整数
  - '0'~'9' 换算为 $0 \sim 9$，公式为
- 字符 ch 大小写字母换算
  - 'A'~'Z' 换算为 'a'~'z'，公式为
  - 'a'~'z' 换算为 'A'~'Z'，公式为

字符换算

填空题

- 字符 ch 换算为整数
  - '0'~'9' 换算为 $0 \sim 9$，公式为 `ch - '0'`
- 字符 ch 大小写字母换算
  - 'A'~'Z' 换算为 'a'~'z'，公式为
  - 'a'~'z' 换算为 'A'~'Z'，公式为

字符换算

填空题

- 字符 ch 换算为整数
  - '0'~'9' 换算为 $0 \sim 9$，公式为 `ch - '0'`
- 字符 ch 大小写字母换算
  - 'A'~'Z' 换算为 'a'~'z'，公式为 `ch - 'A' + 'a'`
  - 'a'~'z' 换算为 'A'~'Z'，公式为

字符换算

填空题

- 字符 ch 换算为整数
  - '0'~'9' 换算为 $0 \sim 9$，公式为 `ch - '0'`
- 字符 ch 大小写字母换算
  - 'A'~'Z' 换算为 'a'~'z'，公式为 `ch - 'A' + 'a'`
  - 'a'~'z' 换算为 'A'~'Z'，公式为 `ch - 'a' + 'A'`

目录

位运算

- 位运算是基于整数的二进制补码进行的运算
  - 计算机内部就是以二进制补码来存储整数数据，因此位运算速度极快
  - 进行位运算时程序会自动按照二进制进行运算，无需把十进制数字转换为二进制
  - 通常只对非负整数进行位运算
- 位运算共 6 种
  - 与 (`&`) , 或 (`|`) , 异或 (`^`) , 取反 (`~`) , 左移 (`<<`) , 右移 (`>>`)

位运算

- 与 (&) , 或 (|) , 异或 (^) 这三种位运算都是两数间的位运算
- 两数之间的位运算：将两个整数的二进制补码中的每一位逐一进行运算，得到结果的每一位

运算符	解释
&	只有两个对应位都为 1 时才为 1
|	只要两个对应位中有一个 1 时就为 1
^	只有两个对应位不同时才为 1

按位与 &

$$
\begin{array}{rr}
 & 0010101 \\
\& & 0000000 \\
\hline
\end{array}
$$

按位与 &

$$\begin{array}{rr} & 0010101 \\ \& & 0000000 \\ \hline & 0000000 \end{array}$$

- `x & 0` 结果为 0

按位与 &

$$\begin{array}{rr} & 0010101 \\ \& & 0000000 \\ \hline & 0000000 \end{array} \qquad \begin{array}{rr} & 0010101 \\ \& & 0000001 \\ \hline & \end{array}$$

- `x & 0` 结果为 0

按位与 &

$$\begin{array}{rr} & 0010101 \\ \& & 0000000 \\ \hline & 0000000 \end{array} \qquad \begin{array}{rr} & 0010101 \\ \& & 0000001 \\ \hline & 0000001 \end{array}$$

- `x & 0` 结果为 0
- `x & 1` 结果为 0 或 1

按位与 &

$$\begin{array}{rr} & 0010101 \\ \& & 0000000 \\ \hline & 0000000 \end{array} \qquad \begin{array}{rr} & 0010101 \\ \& & 0000001 \\ \hline & 0000001 \end{array}$$

- `x & 0` 结果为 0
- `x & 1` 结果为 0 或 1
  - 二进制补码中的最低位，可用于判断 `x` 的奇偶性

按位与 &

$$\begin{array}{rr} & 0010101 \\ \& & 0000000 \\ \hline & 0000000 \end{array} \qquad \begin{array}{rr} & 0010101 \\ \& & 0000001 \\ \hline & 0000001 \end{array} \qquad \begin{array}{rr} & 0010101 \\ \& & 0010101 \\ \hline & \end{array}$$

- `x & 0` 结果为 0
- `x & 1` 结果为 0 或 1
  - 二进制补码中的最低位，可用于判断 `x` 的奇偶性

按位与 &

$$\begin{array}{rr} & 0010101 \\ \& & 0000000 \\ \hline & 0000000 \end{array} \qquad \begin{array}{rr} & 0010101 \\ \& & 0000001 \\ \hline & 0000001 \end{array} \qquad \begin{array}{rr} & 0010101 \\ \& & 0010101 \\ \hline & 0010101 \end{array}$$

- `x & 0` 结果为 0
- `x & 1` 结果为 0 或 1
  - 二进制补码中的最低位，可用于判断 `x` 的奇偶性
- `x & x` 结果为 `x`

按位或 |

$$\begin{array}{r} 0010101 \\ | \quad 0000000 \\ \hline 0010101 \end{array}$$

- `x | 0` 结果为 `x`

按位或 |

$$\begin{array}{r} 0010101 \\ | \quad 0000000 \\ \hline 0010101 \end{array} \qquad \begin{array}{r} 0010101 \\ | \quad 0010101 \\ \hline 0010101 \end{array}$$

- `x | 0` 结果为 `x`
- `x | x` 结果为 `x`

按位异或 ^

$$\begin{array}{rr} & 0010101 \\ \wedge & 0000000 \\ \hline & 0010101 \end{array}$$

- `x ^ 0` 结果为 `x`

按位异或 ^

$$\begin{array}{rr} & 0010101 \\ \wedge & 0000000 \\ \hline & 0010101 \end{array} \qquad \begin{array}{rr} & 0010101 \\ \wedge & 0010101 \\ \hline & 0000000 \end{array}$$

- `x ^ 0` 结果为 `x`
- `x ^ x` 结果为 `0`
  - 如果多个数进行异或，出现偶数次的数字相当于被消除掉
  - 例如: `x ^ x ^ y` 的结果为 `y`

随堂练习

填空题

- 执行 `cout << (21 & 36) << endl;` 的输出结果是
- 执行 `cout << (21 | 36) << endl;` 的输出结果是
- 执行 `cout << (21 ^ 36) << endl;` 的输出结果是
- 21 的补码：0...0010101
- 36 的补码：0...0100100
-

随堂练习

填空题

- 执行 `cout << (21 & 36) << endl;` 的输出结果是
- 执行 `cout << (21 | 36) << endl;` 的输出结果是
- 执行 `cout << (21 ^ 36) << endl;` 的输出结果是
- 21 的补码：0...0010101
- 36 的补码：0...0100100
- 按位与：　0...0000100

随堂练习

填空题

- 执行 `cout << (21 & 36) << endl;` 的输出结果是 4
- 执行 `cout << (21 | 36) << endl;` 的输出结果是
- 执行 `cout << (21 ^ 36) << endl;` 的输出结果是
- 21 的补码：0...0010101
- 36 的补码：0...0100100
- 按位与：　0...0000100

随堂练习

填空题

- 执行 `cout << (21 & 36) << endl;` 的输出结果是 4
- 执行 `cout << (21 | 36) << endl;` 的输出结果是
- 执行 `cout << (21 ^ 36) << endl;` 的输出结果是
- 21 的补码：0...0010101
- 36 的补码：0...0100100
- 按位或：　0...0110101

随堂练习

填空题

- 执行 `cout << (21 & 36) << endl;` 的输出结果是 4
- 执行 `cout << (21 | 36) << endl;` 的输出结果是 53
- 执行 `cout << (21 ^ 36) << endl;` 的输出结果是
- 21 的补码：0...0010101
- 36 的补码：0...0100100
- 按位或：　0...0110101

随堂练习

填空题

- 执行 `cout << (21 & 36) << endl;` 的输出结果是 4
- 执行 `cout << (21 | 36) << endl;` 的输出结果是 53
- 执行 `cout << (21 ^ 36) << endl;` 的输出结果是
- 21 的补码：0...0010101
- 36 的补码：0...0100100
- 按位异或：0...0110001

随堂练习

填空题

- 执行 `cout << (21 & 36) << endl;` 的输出结果是 4
- 执行 `cout << (21 | 36) << endl;` 的输出结果是 53
- 执行 `cout << (21 ^ 36) << endl;` 的输出结果是 49
- 21 的补码：0...0010101
- 36 的补码：0...0100100
- 按位异或：0...0110001

例 4.1：找筷子

编程题

- 输入一个整数 n ($1 \leq n \leq 10^7$)，表示有 n 只筷子，接下来输入 n 个整数 x ($1 \leq x \leq 10^9$)，表示筷子的长度。
  这些筷子中只有一只筷子是落单的，其余都成双。
  请找到落单的那只筷子，并输出它的长度。

- 样例输入

  ```
  5
  2 1 2 3 3
  ```

- 样例输出

  ```
  1
  ```

例 4.1：找筷子

```cpp
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  int ans = 0;
  for (int i = 1; i <= n; i++) {
    int a;
    cin >> a;
    ans ^= a;
  }
  cout << ans << endl;
  return 0;
}
```

例 4.1：找筷子

```cpp
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  int ans = 0;
  for (int i = 1; i <= n; i++) {
    int a;
    cin >> a;
    ans ^= a;
  }
  cout << ans << endl;
  return 0;
}
```

按位取反

- 取反是对一个数进行的计算，即单目运算
- ~x 表示将 x 的二进制补码中包括符号位在内的所有 0 变 1，1 变 0 所得的值

$$\sim 0010101$$

按位取反

- 取反是对一个数进行的计算，即单目运算
- ~x 表示将 x 的二进制补码中包括符号位在内的所有 0 变 1，1 变 0 所得的值

$$\begin{array}{r} \sim 0010101 \\ \hline 1101010 \end{array}$$

按位取反

- 取反是对一个数进行的计算，即单目运算
- ~x 表示将 x 的二进制补码中包括符号位在内的所有 0 变 1，1 变 0 所得的值

$$\begin{array}{r} \sim 0010101 \\ \hline 1101010 \end{array}$$

- 原数值：21
- ~ 21 的值是 -22

左移 <<

- `x << i` 表示将 `x` 的二进制补码向左移动 `i` 位所得的值
  - 左边越界部分被丢弃，右边的空位填充为 0

- 例：00010111　左移 2 位后　01011100

  - `23 << 2` 的结果是 92
  - `x << i` 相当于 $x \times 2^i$

右移 > >

- `x >> i` 表示将 `x` 的二进制补码向右移动 `i` 位所得的值
  - 右边越界部分被丢弃，若 `x` 为非负数，左边的空位填充为 0
- 例：00010111　右移 2 位后　00000101
  - `23 >> 2` 的结果是 5
  - `x >> i` 相当于 $x \div 2^i$

例 4.2：二进制补码形式

编程题

- 输入一个整数 n ($1 \leq n \leq 10^9$)，输出该整数 8 位二进制补码形式。
- 样例输入
  ```
  7
  ```
- 样例输出
  ```
  00000111
  ```

例 4.2：二进制补码形式

- 类似十进制整数分离数位的原理：
  1. 取出最后一位，并存储在数组中
  2. 去除最后一位
  3. 不断重复上述过程，直到整数二进制补码的每一位都已被取出
  4. 逆序输出数组

例 4.2：二进制补码形式

- 类似十进制整数分离数位的原理：
  1. 取出最后一位，并存储在数组中
  2. 去除最后一位
  3. 不断重复上述过程，直到整数二进制补码的每一位都已被取出
  4. 逆序输出数组
- 通过 `& 1` 得到最后一位，通过 `>> 1` 去掉最后一位。

例 4.2：二进制补码形式

```cpp
#include <iostream>

using namespace std;

int bin[35];

int main() {
  int n;
  cin >> n;
  // 从低位到高位依次取出 n 的补码的每一位，存储在 bin 数组中
  for (int i = 1; i <= 8; i++) {
    bin[i] = n & 1;
    n >>= 1; // n = n >> 1;
  }
  // 逆序输出结果
  for (int i = 8; i >= 1; i--) {
    cout << bin[i];
  }
  return 0;
}
```

位运算注意要点

- 同加减乘除运算，位运算本身也不修改变量的值
  - 如 `~x` 不改变 `x` 的值，赋值语句 `x = ~x;` 才能修改 `x` 的值
- 位运算的优先级容易遗忘，注意按照自己希望的计算顺序加括号
  - 如 `if (0 & 1 < 1)`
  - 如果希望 `0 & 1` 先算，应写为 `if ((0 & 1) < 1)`
- 移位
  - 常用 `1 << n` 来计算 2^n，不能写为 `2 << n`

目录

总结

- 编码
  - 整数编码：原码、反码、补码
  - 字符编码：字符类型判断、字符和整数转换、大小写字母转换

总结

- 编码
  - 整数编码：原码、反码、补码
  - 字符编码：字符类型判断、字符和整数转换、大小写字母转换
- 位运算
  - 与 (&) , 或 (|) , 异或 (^)
  - 取反 (~)
  - 左移 (<<) , 右移 (>>)