C++中的二进制

以下是C++中二进制整型数据的存储格式说明、十进制与二进制的转换方法及演示代码：

一、二进制整型存储格式

1. **无符号整数**
   直接以二进制形式存储，所有位表示数值。
   示例：`unsigned char 5` → `00000101`
2. **有符号整数**
   采用**补码**存储：
   - **正数**：与原码相同。
   - **负数**：原码取反后加1。
     示例：`char -5` → 原码 `10000101` → 反码 `11111010` → 补码 `11111011`
3. **字节顺序（Endianness）**
   数据在内存中的存储顺序分为大端序（高位在前）和小端序（低位在前），但转换时无需关心。

二、转换方法

1. 十进制转二进制

- **无符号数**：逐次除以2取余，逆序拼接余数。
- **有符号数**：转换为补码形式，通过位运算直接提取每一位。

2. 二进制转十进制

- **无符号数**：按权展开求和，即每位值乘以2的幂次。
- **有符号数**：若最高位为1，先取补码得到绝对值，再添加负号。

三、演示代码

```cpp
#include <iostream>
#include <string>
#include <bitset> // 用于验证结果

using namespace std;

// 十进制转二进制（补码形式，指定位数）
string decimalToBinary(int num, int bits = 8) {
    string binary;
    for (int i = bits - 1; i >= 0; i--) {
        binary += (num & (1 << i)) ? '1' : '0';
    }
    return binary;
```

```cpp
}

// 二进制转十进制（支持补码）
int binaryToDecimal(const string& binary, int bits) {
    if (binary.length() != bits) {
        cerr << "错误：二进制位数应为 " << bits << endl;
        return 0;
    }
    int value = 0;
    bool isNegative = (binary[0] == '1');
    string processed = binary;

    // 负数处理：补码转原码
    if (isNegative) {
        // 取反
        for (char& c : processed) c = (c == '0') ? '1' : '0';
        // 加1
        int carry = 1;
        for (int i = bits - 1; i >= 0 && carry; i--) {
            if (processed[i] == '1') {
                processed[i] = '0';
            } else {
                processed[i] = '1';
                carry = 0;
            }
        }
    }

    // 计算十进制值
    for (int i = 0; i < bits; i++) {
        value = value * 2 + (processed[i] - '0');
    }
    return isNegative ? -value : value;
}

int main() {
    // 示例1：十进制转二进制
    int a = 13, b = -5;
    cout << "十进制 " << a << " → 二进制：" << decimalToBinary(a, 8) << endl;
    cout << "十进制 " << b << " → 二进制：" << decimalToBinary(b, 8) << endl;

    // 示例2：二进制转十进制
    string bin1 = "00001101", bin2 = "11111011";
    cout << "二进制 " << bin1 << " → 十进制：" << binaryToDecimal(bin1, 8) << endl;
    cout << "二进制 " << bin2 << " → 十进制：" << binaryToDecimal(bin2, 8) << endl;

    // 验证（使用<bitset>）
    cout << "\n验证（使用<bitset>）：" << endl;
    cout << "二进制 11111011 → 十进制：" << (int)bitset<8>("11111011").to_ulong() <<
endl;
    return 0;
}
```

输出结果：

复制

```
十进制 13 → 二进制：00001101
十进制 -5 → 二进制：11111011
二进制 00001101 → 十进制：13
二进制 11111011 → 十进制：-5

验证（使用<bitset>）：
二进制 11111011 → 十进制：251
```

四、注意事项

1. **负数的验证问题**：
   `<bitset>` 将二进制视为无符号数，因此 `11111011` 输出为251。实际转换需通过补码逻辑处理符号。
2. **位数一致性**：
   转换时必须明确位数（如8位、32位），否则补码计算会出错。