**计算机网络编程**

**实验报告**

**班级：07111707**

**组长：1120171189 崔程远**

**成员：1120172149 吴沁璇**

**1120172153 张澈**

**1120172163 王晓媛**

**1120172736 张鉴昊**

**1120172765 曾煜瑾**

**1120173326 曾紫飞**

**北京理工大学**

**计算机学院**

**2020年5月**

**第三章 实验1 循环冗余校验CRC生成和校验程序**

**1. 实验目的**

模拟数据链路层中差错控制的循环冗余码CRC，加强对CRC原理的认识和计算方法的掌握。

1. **实验内容**

配置文件关键要点：

待发送的数据信息二进制比特串（32位）

InfoString1=0110XXXXXXXXXXXXXXXXXXXX110

收发双方预定的生成多项式采用CRC-CCITT=X16+X12+X5+1，对应的二进制比特串（17位）

GenXString=10001000000100001

接收的数据信息二进制比特串（32位）

InfoString2=0110XXXXXXXXXXXXXXXXXXXX110

程序运行屏幕输出要点：

首先显示待发送的数据信息二进制比特串

然后显示收发双方预定的生成多项式采用CRC-CCITT，对应的二进制比特串

计算循环冗余校验码CRC-Code

显示生成的CRC-Code，以及带校验和的发送帧

显示接收的数据信息二进制比特串，以及计算生成的CRC-Code

计算余数

显示余数，为零表示无错，不为零表示出错

1. **实验原理**

收发双方约定一个生成多项式 G(x)（其最高阶和最低阶系数必须为1），发送方用位串及 G(x)进行某种运算得到校验和，并在帧的末尾加上校验和，使带校验和的帧的多项式能被 G(x) 整除; 接收方收到后，用 G(x) 除多项式，若有余数，则传输有错。

发送端CRC校验和计算方法：

1. 若生成多项式 G(x) 为 r 阶(即r＋1位位串)，原帧为 m 位， 其多项式为 M(x)，则在原帧后面添加 r 个 0，即循环左移r位，帧成为 m+r 位，相应多项式成为 ;
2. 按模2除法用 G(x)对应的位串去除对应于 的位串, 得余数 R(x)；
3. 按模2减法(即模2加)从对应于 的位串中减去(加上)余数 R(x),结果即传送的带校验和的帧多项式T(x)。

接收端CRC检验方法：

·接收方收到后，用 G(x) 除多项式，若有余数，则传输有错。若无余数，则正确传输。

1. **实验环境**

|  |  |  |
| --- | --- | --- |
| 语言 | 集成开发环境 | 编译器 |
| C++ | Visual Studio 2017 | gcc version 4.8.1 |
| Java | Eclipse 2019 | java version "1.8.0\_65" |
| Python | Pycharm 2017 | Python 3.7.0 |

1. **实验步骤**

三份代码的结构完全一致，均采用面向对象的构造方法，首先初始化一个类。

1. 然后在Send函数中初始化待发送的数据信息二进制比特串，作为配置文件，并打印相应输出。之后调用GetRemainderStr函数计算余数，得到CRC校验码，返回CRC校验码。
2. 将CRC检验码添加在数据信息后，传给Receive函数。在Receive函数中提取出数据信息和CRC，打印相应输出。之后调用GetRemainderStr函数计算余数，如果余数为0，表示校验成功，否则表示出错。

下面以C++代码为例，进行分析。

·全局变量

string gxStr = "10001000000100001";

表示收发双方预定的生成多项式采用CRC-CCITT=X16+X12+X5+1，对应的二进制比特串。

·GetRemainderStr函数

string GetRemainderStr(string dividendStr, string divisorStr)

{

int dividendLen = dividendStr.length();

int divisorLen = divisorStr.length();

for (int i = 0; i < divisorLen - 1; i++)

{

dividendStr += "0";

}

for (int i = 0; i < dividendLen; i++)

{

if (dividendStr[i] == '1') //如果该位为1

{

dividendStr[i] = '0';

for (int j = 1; j < divisorLen; j++)

{

if (dividendStr[i + j] == divisorStr[j])

{

dividendStr[i + j] = '0';

}

else

{

dividendStr[i + j] = '1';

}

}

}

}

string remainderStr = dividendStr.substr(dividendLen, dividendLen + divisorLen);

return remainderStr;

}

·Send函数

string Send()

{

string dataStr = "01100000000000111110000000000110";

cout << "待发送的数据信息二进制比特串为：" + dataStr << endl;

cout << "CRC-CCITT对应的二进制比特串为：" + gxStr << endl;

string remainderStr = GetRemainderStr(dataStr, gxStr);

string crcStr = remainderStr;

string sendFrameStr = dataStr + remainderStr;

cout << "生成的CRC-Code为: " << crcStr << endl;

cout << "带校验和的发送帧为: " << sendFrameStr << endl << endl;

return sendFrameStr;

}

·Receive函数

void Receive(string sendFrameStr)

{

int sendFrameLen = sendFrameStr.length();

int gxLen = gxStr.length();

string dataStr = sendFrameStr.substr(0, sendFrameLen - gxLen + 1);

string crcStr = sendFrameStr.substr(sendFrameLen - gxLen + 1);

cout << "接收的数据信息二进制比特串为：" << dataStr << endl;

cout << "生成的CRC-Code为: " + crcStr << endl;

string remainderStr = GetRemainderStr(sendFrameStr, gxStr);

int remainder = atoi(remainderStr.c\_str());

cout << "余数为: " << remainder << endl;

if (remainder == 0)

{

cout << "校验成功" << endl;

}

else

{

cout << "校验错误" << endl;

}

}

·主函数

int main(){

CRC operation;

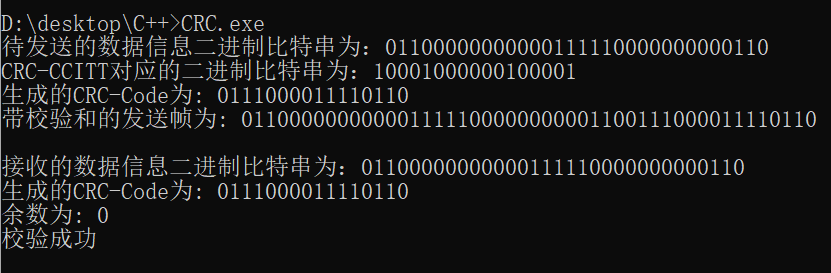
string frameStr = operation.Send();

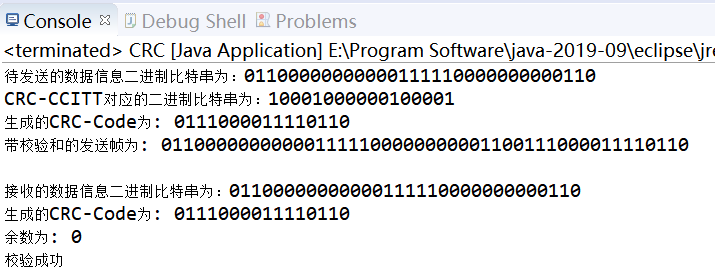
operation.Receive(frameStr);}

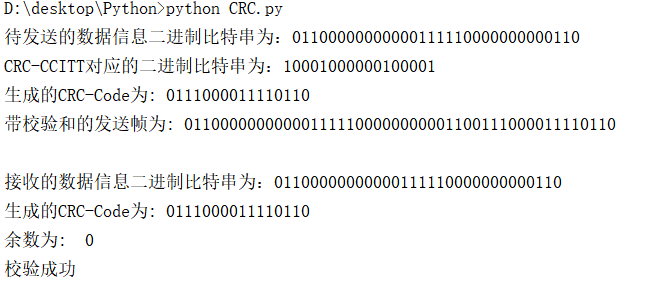
1. **运行结果**

Send函数中将01100000000000111110000000000110作为待发送的数据信息。

**·C++**

**·Java**

**·Python**



1. **实验总结**

这个实验原理相对简单，模拟起来也比较容易，但是在写代码的过程中也遇到了一些问题，调试了许久，因为不同语言方法的调用有些区别，所以A语言的方法对应的B语言的该方法用的时候就会有些问题，这些隐藏的问题通过打log缩小范围才得以解决，比如说将数字字符串和整数的相互转换等。

另外在GetRemainderStr函数求余数的时候，我一开始是将字符串转换为整数进行异或操作，然而经过长时间调试后发现这是有问题的，因为在Receive函数调用GetRemainderStr函数的时候，传入的两个参数分别是32+16、16位的字符串，两者相加就是64位，对应的数值已经超过了long的表示范围，所以在Java和C++中是行不通的，在Python中是可以的。

总的来说，这个实验使我加深了对CRC校验的理解，提高了编程能力。