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。

2. 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

3. 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。

4. 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

5. 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);}
```

6. 运行结果

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

• C++

```
D:\desktop\C++>CRC.exe
待发送的数据信息二进制比特串为：01100000000000111110000000000110
CRC-CCITT对应的二进制比特串为：10001000000100001
生成的CRC-Code为：0111000011110110
带校验和的发送帧为：01100000000000111110000000000110011100001111011O

接收的数据信息二进制比特串为：01100000000000111110000000000110
生成的CRC-Code为：0111000011110110
余数为：0
校验成功
```

• Java

```
Console  Debug Shell  Problems
<terminated> CRC [Java Application] E:\Program Software\java-2019-09\eclipse\jr
待发送的数据信息二进制比特串为：01100000000000111110000000000110
CRC-CCITT对应的二进制比特串为：10001000000100001
生成的CRC-Code为：0111000011110110
带校验和的发送帧为：01100000000000111110000000000110011100001111011O

接收的数据信息二进制比特串为：01100000000000111110000000000110
生成的CRC-Code为：0111000011110110
余数为：0
校验成功
```

• Python

```
D:\desktop\Python>python CRC.py
待发送的数据信息二进制比特串为：01100000000000111110000000000110
CRC-CCITT对应的二进制比特串为：10001000000100001
生成的CRC-Code为：0111000011110110
带校验和的发送帧为：01100000000000111110000000000110011100001111011O

接收的数据信息二进制比特串为：01100000000000111110000000000110
生成的CRC-Code为：0111000011110110
余数为：  0
校验成功
```

7. 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。