《程序设计进阶与实践》实验报告

姓名	王昱	学号	PB21030814	日期	2022.3.19
实验名称	梅森素数的判定(p<=1000)				

实验环境:
CPU: Intel(R) Core(TM) i7-10870H CPU @ 2.20GHz 2.21 GHz
内存: 16.0 GB (15.8 GB 可用)
操作系统: Windows 11 家庭中文版
软件平台: Visual Studio Code 1.65.2

一、问题分析与求解思路

题目要求是利用 C 语言以高精度数的方式实现 1000 以内的梅森素数的求解，我的思路如下:

①利用试除法求出 1000 以内的素数存放在一个数组当中。

②将指数 p 对应的梅森素数以二进制的形式先存放在数组中，然后再将二进制转化成对应的十进制，存放在另一个数组中。

③利用 Lucas－Lehmer 算法，同时利用高精度乘法、求余运算，算出卢卡斯·莱默余数,存放在一个数组中。

令梅森数 [1] $M_p=2^p-1$作为检验对象,定义数列{L_n}:$L_0=4$, $L_n=L_{n-1}^2-2$,n>0.这个数列的开始几项是4,14,194,37634,1416317954.....**那么M_p是质数当且仅当$L_{p-2}\equiv 0\,(mod M_p)$**.否则$M_p$是合数。$s_{p-2}$模$M_p$的余数叫做$M_p$的**卢卡斯－莱默余数。**

④判断卢卡斯·莱默余数是否为零，为零则指数 p 对应的梅森素数是质数，否则为合数。

二、核心代码说明

```
//求余函数，余数存放在第一个参数里
void Remainder(int *an1, int *an2, int *tmpAn2)
{
    while (Max(MAX_LEN, an1, an2) == an1)
    {
        int nShiftLen = Length(MAX_LEN, an1) - Length(MAX_LEN, an2) - 1;
        ShiftLeft(MAX_LEN, an2, tmpAn2, nShiftLen);
        while (Max(MAX_LEN, an1, tmpAn2) == an1)
        {
            Substract(MAX_LEN, an1, tmpAn2);
        }
    }
}
```

求余函数：先调用 Max 函数，判断第一个参数和第二个参数哪个大，如果第二个参数大就直接退出循环否则继续循环。然后调用 Length 函数求大整数的长度。接着调用左移函数 ShiftLen，将左移后的结果存放在一个中间数组中（也就是第三个参数中）。最后判断第一个参数和第三个参数哪个大，如果第一个大就调用 Substract 函数让第一个参数一直减第三个参数，否则退出第二个 while 循环。

```
void L_L_Remainder(int *DecimalismMersenne, int p, int Result[], int L_L_Expression[], int Tmp[])
{
    for (int k = 0; k < p - 2; k++)
    {
        memset(L_L_Expression, 0, MAX_LEN * sizeof(int));
        if (k != 0)
        {
            for (int m = 0; m < MAX_LEN; m++)
            {
                L_L_Expression[m] = Result[m];
            }
            memset(Result, 0, 2 * MAX_LEN * sizeof(int));
        }
        else
        {
            memset(Result, 0, 2 * MAX_LEN * sizeof(int));
            L_L_Expression[0] = 4;
        }
        Multiply(Length(MAX_LEN, L_L_Expression), L_L_Expression,

        Length(MAX_LEN, L_L_Expression), L_L_Expression, Result);
        Remainder(Result, DecimalismMersenne, Tmp);
    }
}
```

求卢卡斯·莱默余数函数：第一个参数是待测数的十进制；第二个参数是指数 p；第三个参数是存放结果的数组；第四个参数是卢卡斯莱默算法中数列的表达式;第五个参数是求余的中间数组。每次循环开始之前进行必要的数组清零（如果是第一次循环还要给数列表达式赋予初值 4 否则就将上一次余数的结果赋给 L_L_Expression），然后进行表达式的自乘再减 2（这里减 2 也放在 Multiply 函数中进行了），最后调用取余函数求出卢卡斯·莱默余数放在 Result 数组中。

```
int BinaryNumber[MAX_LEN1] = {0};
int Prime[MAX_LEN] = {0};
int Tmp[MAX_LEN] = {0};
int L_L_Expression[MAX_LEN] = {4};
int Result[MAX_LEN * 2] = {0};
BigInt Big;
IsPrime(Prime); // Prime数组用于存放素数p
for (int i = 0; Prime[i] && i < MAX_LEN; i++)
{
    memset(&Big, 0, sizeof(Big));
    // BinaryNumber用于存放二进制形式
    TransToBinaryNumber(Prime[i], BinaryNumber);
    //此循环用于将二进制转化成十进制
    for (int j = 0; BinaryNumber[j]; j++)
    {
        TransToDecimalism(&Big, BinaryNumber[j] == 1 ? 1 : 0);
    }
    L_L_Remainder(Big.Value, Prime[i], Result, L_L_Expression, Tmp);
    //这里当Prime[0]=2时，它没有执行上述循环直接跳出，Result数组length为0
    //根据现有知识，2^2-1是素数，故这里直接把p=2打印出来
    if (Length(MAX_LEN, Result) == 0)
    {
        printf("P Value:%d\n", Prime[i]);
        printf("MersennePrime:");
        for (int n = Length(MAX_LEN, Big.Value) - 1; n >= 0; n--)
            printf("%d", Big.Value[n]);
        printf("\n");
    }
}
```

主函数：最外层 for 循环是用来遍历 p 为素数的所有情况；内层 for 循环是用来将二进制转化成十进制；卢卡斯·莱默算法不能判断 p=2 时的情况，根据现有知识知道 2^2-1 是素数，而当 p=2 时没有执行内层第二个 for 循环 Result 数组 length 为 0，正好能将 p=2 输出

三、测试、运行与分析

运行结果如图

```
P Value:2
MersennePrime:3
P Value:3
MersennePrime:7
P Value:5
MersennePrime:31
P Value:7
MersennePrime:127
P Value:13
MersennePrime:8191
P Value:17
MersennePrime:131071
P Value:19
MersennePrime:524287
P Value:31
MersennePrime:2147483647
P Value:61
MersennePrime:2305843009213693951
P Value:89
MersennePrime:618970019642690137449562111
P Value:107
MersennePrime:162259276829213363391578010288127
P Value:127
MersennePrime:170141183460469231731687303715884105727
P Value:521
MersennePrime:686479766013060971498190079908139321726943530014330540
0939446345918554318339765605212255964066145455497729631139148085803
7121987999716643812574028291115057151
P Value:607
MersennePrime:531137992816767098689588206552468627329593117727031923
1994441382004035598608522427391625022652292856688893294862465010153
4657933765270723940951997876658735194383127083539321903172812

times:226.748000
请按任意键继续. . .
```

测试结果如图

```
测试数据p的值:5
二进制表示:
11111
十进制表示:
31
卢卡斯·莱默余数长度:0
卢卡斯·莱默余数十进制表示:0

测试数据p的值:83
二进制表示:
111111111111111111111111111111111111111111111111111111111111111111111
11111111111111
十进制表示:
9671406556917033397649407
卢卡斯·莱默余数长度:27
卢卡斯·莱默余数十进制表示:150239943086106615048262034

测试数据p的值:191
二进制表示:
111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111
十进制表示:
9713344461128645354597309534117594533212034195260697606259062048694521
42602604249087
卢卡斯·莱默余数长度:84
卢卡斯·莱默余数十进制表示:7927724779085076844313915906251375916628342
00888934049890582333510724093990120118824

测试数据p的值:307
二进制表示:
111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111
十进制表示:
2532663311084590428779545815241187225959745014796409240720005694391267
5850908863198240399468671287806934801554024052668349579779513011323900
0676726282433860394660533468026791526 3
卢卡斯·莱默余数长度:176
卢卡斯·莱默余数十进制表示:75542256036151397743892127590763606365000 9
5085530072595817194956596681195513784817882948984842295643419108616 26
2475176423363884973012514739720183872928796671554761688387611098 8

测试数据p的值:431
二进制表示:
111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111
十进制表示:
1404447761611184302913519680303925573139044514798677009948672509044786
5297304762744742849790813088751650008896864952606067092950688626298632
2537055187089159670131166738176160372111109063473511030822721056316410
7569048052205800491261514946176100212790338675864723330454999587858894
3727836315262213251892510 71
卢卡斯·莱默余数长度:306
卢卡斯·莱默余数十进制表示:1850895388501953089902400742328203325222553
1033204449005172012234906150058987725593017378865931687107418837893600
0325993596599606071760074225701279605580089197562512603252746233864064
4724002773821872721672647483295969968144989801764162317319284731692537
005083184407511118334722960860028245266014328973744 11

请按任意键继续. . .
```

这里仅挑选了五组数据作为测试样例，分别是 p=5、83、191、307、431，测试内容为

转二进制、转十进制、求长度、求卢卡斯•莱默余数函数，均给出了正确的结果。由于求卢卡斯•莱默余数函数中调用了大部分函数，故我认为测试求卢卡斯•莱默函数的正确性是主要的也是最重要的（通过测试它也就测试了它调用的函数，也就有了较高的测试代码覆盖率）。通过合理地选用数据，既能够测试出梅森素数又能够测出非梅森素数的卢卡斯•莱默余数，起到检验高精度运算的作用。

四．备注

由于篇幅所限，测试代码以注释形式放在源码后面

总结

遇到的困难：

①不知道该选用什么算法。之前从来未接触过类似的题目，编程基础也仅仅限于大一上学习的 C 语言，所以在寻找算法的时候耗费了很长时间。看到群友说 MR 算法时去看了看，但发现有些复杂，最终选用了 Lucas-Lehmer 算法

②在具体实现的过程中，由于一些粗心导致出现的 bug 较多，又由于经验的欠缺使得 debug 的时间较长。遇到的最大的一个 bug 是我在运行整个程序的时候，在程序把 p=521 判断出来之后就很长时间不出结果。一开始我一直以为是我写的程序复杂度太高没法在正常时间内运行出来，甚至想要用 MR 算法重写程序。但是群友们说 Lucas-Lehmer 算法可以很快得到正确结果，我又不断地尝试 debug。我将 p 分成了一段一段很小的区间，经过测试发现我原来的程序在 p=251、367、541、569、631、727、769、773、863 时程序运行极其缓慢甚至运行不出来，而在去除这些数之后程序能在 5min 之内跑完。于是我单独测试了 251，利用 vscode 的调试功能，发现存放卢卡斯•莱默余数的数组在运行时有数组元素超过 10 的时候，所以我又回头看了乘法函数和求余函数，发现了乘法函数的一个 for 循环判断条件写错了（这个条件本来是想减少循环次数的，没想到意外出错）。修改之后，程序成功运行出来了。

收获：

①VScode 编辑器是现学的，专门找了本讲 vscode 的书看了看，感觉这个编辑器很好用，debug 比较方便，插件也比较好。

②学会利用 C 语言处理高精度数问题，锻炼了自己写代码的能力、debug 的能力和“翻译”算法的能力。

③规范了自己的代码风格。比如之前的变量名、函数名的命名很不规范，经常用 i、j、k 等或者用中文拼音。

④进一步加强了自己的模块化编程能力，主函数中大部分都是直接调用函数，具体功能由具体函数实现。

总而言之，虽然我自身的编程基础很弱，但通过这一次大作业不仅仅学到了知识，也进一步加强了写代码的能力。跑出程序的那一刻总归是令人欣慰的。

最后，感谢助教和老师于百忙之中抽出时间解答。

另附（源代码文件名称）	MersennePrime.c