中国科学技术大学
University of Science and Technology of China

程序设计II

第5讲 高精度计算

计算机学院 黄章进

zhuang@ustc.edu.cn

内容

- 例题：大整数加法 2981
- 例题：大整数减法
- 例题：大整数乘法 2980
- 例题：大整数除法 2737
- 例题：循环数 2952
- 例题：麦森数 2706

C/C++中数的表示范围

- 整数 limits.h
  - int/long：$-2^{31} \sim 2^{31}-1$，即 -2 147 483 648 ~ 2 147 483 647
  - unsigned：$0 \sim 2^{32}-1$，即 0 ~ 4 294 967 295
  - long long：$-2^{63} \sim 2^{63}-1$，即-9 223 372 036 854 775 808 ~ 9 223 372 036 854 775 807
  - unsigned long long：$0 \sim 2^{64}-1$，即0 ~ 18 446 744 073 709 551 615
- 如何处理一个200位的整数？

C/C++中数的表示范围

- 浮点数：
  - float 单精度，32位，6位精度
    - 最小正值 1.17549×10^{-38}，最大值 3.40282×10^{38}
  - double 双精度，64位，15位精度
    - 最小正值 2.22507×10^{-308}，最大值 1.79769×10^{308}
  - long double 扩展精度，96位
  - 存储遵循IEEE 745标准（即IEC 60559）规范
- 如何精确到小数点后100位

例题：大整数加法

- 问题描述
  - 求两个不超过200位的非负整数的和。
- 输入
  - 有两行，每行是一个不超过200位的非负整数，没有多余的前导0。
- 输出
  - 一行，即相加后的结果。结果里不能有多余的前导0，即如果结果是342，那么就不能输出为0342。

大整数加法

- 输入样例

22222222222222222222

33333333333333333333

- 输出样例

55555555555555555555

大整数加法

- 解题思路
  - 用字符型或整型数组来存放大整数
    - an[0]存放个位数，an[1]存放十位数，an[2]存放百位数……
  - 模拟小学生列竖式做加法，从个位开始逐位相加，超过或达到10则进位。
    - 用int an1[201]保存第一个数，用int an2[200]表示第二个数，然后逐位相加，相加的结果直接存放在an1中。要注意处理进位。

2981 大整数加法


```
#include <stdio.h>
#include <string.h>
#define MAX_LEN 201
int an1[MAX_LEN+10], an2[MAX_LEN+10];
char szLine1[MAX_LEN+10], szLine2[MAX_LEN+10];
int Add(int nMaxLen, int *an1, int *an2)
// 将长度最多为nMaxLen的大整数an1和an2相加，结果放在an1
// an1[0], an2[0]对应于个位
{
   int i, nHighestPos = 0;
   for ( i = 0; i < nMaxLen; i++ ) {
      an1[i] += an2[i];                    //逐位相加
      if ( an1[i] >= 10 ) {      //看是否要进位
         an1[i] -= 10;
         an1[i+1] ++;                              //进位
      }
      if ( an1[i] )          nHighestPos = i; //记录最高位的位置
   }
   return nHighestPos;  }
```

大整数加法


```
int main() {
    scanf("%s", szLine1);       scanf("%s", szLine2);
    // 库函数memset将地址an1开始的sizeof(an1)字节内容置成0
    // sizeof(an1)的值就是an1的长度，memset函数在string.h中声明
    memset(an1, 0, sizeof(an1));       memset(an2, 0, sizeof(an2));
    // 下面将szLine1中存储的字符串形式的整数转换到an1中去，
    // an1[0]对应于个位
    int i, j;
    int nLen1 = strlen(szLine1);
    for (j = 0, i = nLen1 - 1; i >= 0; i--)
        an1[j++] = szLine1[i] - '0';
    int nLen2 = strlen(szLine2);
    for (j = 0, i = nLen2 - 1; i >= 0 ; i--)
        an2[j++] = szLine2[i] - '0';
    int nHighestPos = Add(MAX_LEN, an1, an2);     // 可否改进？
    for (i = nHighestPos; i >= 0; i --)  printf("%d", an1[i]);
    return 0; }
```

例题：大整数减法

- 问题描述
  - 求2个大的正整数相减的差
- 输入
  - 第1行是测试数据的组数n，每组测试数据占2行，第1行是被减数a，第2行是减数b(a > b)。每组测试数据之间有一个空行，每行数据不超过100个字符
- 输出要求
  - n行，每组测试数据有一行输出是相应的整数差

大整数减法

- 输入样例

2

99999999999999999999999999999999999

9999999999999

54096567750978508956870567980689709345465465756767686784354353451

- 输出样例

99999999999999999999999900000000000000

54096567750978508956870567980689709345465465756767686784354353444

大整数减法


```
int main() {
    int n;
    scanf("%d", &n);
    while (n--) {
        scanf("%s", szLine1);          scanf("%s", szLine2);
        int i, j;
        memset(an1, 0, sizeof(an1));          memset(an2, 0, sizeof(an2));
        //下面将szLine1中存储的字符串形式的整数转换到an1中去，
        //an1[0]对应于个位
        int nLen1 = strlen( szLine1);
        for (j = 0, i = nLen1 - 1; i >= 0; i--)
            an1[j++] = szLine1[i] - '0';
        int nLen2 = strlen(szLine2);
        for( j = 0, i = nLen2 - 1; i >= 0; i--)
            an2[j++] = szLine2[i] - '0';
        int nStartPos = Substract(MAX_LEN, an1, an2);
        for ( i = nStartPos; i >= 0; i -- )              printf("%d", an1[i]);
        printf("\n");
    }
    return 0; }
```

大整数减法


```
#include <stdio.h>
#include <string.h>
#define MAX_LEN 110
int an1[MAX_LEN], an2[MAX_LEN];
char szLine1[MAX_LEN], szLine2[MAX_LEN];

int Substract(int nMaxLen, int *an1, int *an2)
{   // 两个最多nMaxLen位的大整数an1减去an2, an1保证大于an2
    int i, nStartPos = 0;
    for ( i = 0; i < nMaxLen; i++ )  {  // 请补全循环体代码




        }
        if ( an1[i] )            nStartPos = i; //记录最高位的位置
    }
    return nStartPos;  // 返回值是结果里面最高位的位置
}
```

例题：大整数乘法

- 问题描述
  - 求两个不超过200位的非负整数的积。
- 输入
  - 有两行，每行是一个不超过200位的非负整数，没有多余的前导0。
- 输出
  - 一行，即相乘后的结果。结果里不能有多余的前导0，即如果结果是342，那么就不能输出为0342

大整数乘法

- 输入样例

12345678900

98765432100

- 输出样例

1219326311126352690000

大整数乘法

- 解题思路
  - 用unsigned an1[200]和unsigned an2[200]分别存放两个乘数，用aResult[400]来存放积。计算的中间结果也都存在aResult中。
    - aResult长度取400是因为两个200位的数相乘，积最多有400位
    - an1[0], an2[0], aResult[0]都表示个位。
- 一个数的第i位和另一个数的第j位相乘所得的数，一定是要累加到结果的第i+j位上。
  - 这里i, j都是从右往左，从0开始数。
- 计算的过程基本上和小学生列竖式做乘法相同。
  - 为编程方便，并不急于处理进位，而将进位问题留待最后统一处理。

大整数乘法

现以835×49为例来说明程序的计算过程。

- 先算835×9。5×9得到45个1，3×9得到27个10，8×9得到72个100。由于不急于处理进位，所以835×9算完后，aResult如下：

下标		5	4	3	2	1	0
aResult	- - - -	0	0	0	72	27	45

- 接下来算5×4。此处5×4的结果代表20个10，因此要 aResult[1] += 20，变为：

下标		5	4	3	2	1	0
aResult	- - - -		0	0	72	**47**	45

大整数乘法

- 835×49

下标		5	4	3	2	1	0
aResult	----		0	0	72	**47**	45

- 再下来算3×4。此处3×4的结果代表12个100，因此要 aResult[2] += 12，变为：

下标		5	4	3	2	1	0
aResult	----	0	0	0	**84**	47	45

- 最后算 8×4。此处8×4的结果代表 32个1000，因此要aResult[3] += 32，变为：

下标		5	4	3	2	1	0
aResult	----	0	0	**32**	84	47	45

大整数乘法

- 835×49 乘法过程完毕：

下标		5	4	3	2	1	0
aResult	----	0	0	32	84	47	45

- 接下来从 aResult[0]开始向高位逐位处理进位问题
- aResult[0]留下5，把4加到aResult[1]上，aResult[1]变为51后，应留下1，把5加到aResult[2]上……最终使得aResult里的每个元素都是1位数，结果就算出来了：

下标		5	4	3	2	1	0
aResult	----	0	4	0	9	1	5

2980 大整数乘法

中国科学技术大学
University of Science and Technology of China

```
#include <stdio.h>
#include <string.h>
#define MAX_LEN 200
unsigned an1[MAX_LEN+10], an2[MAX_LEN+10];
unsigned aResult[MAX_LEN * 2 + 10];
char szLine1[MAX_LEN+10], szLine2[MAX_LEN+10];
int main() {
    gets(szLine1);        gets(szLine2);
    memset(an1, 0, sizeof(an1));      memset(an2, 0, sizeof(an2));
    memset(aResult, 0, sizeof(aResult));
    int i, j;
    int nLen1 = strlen(szLine1);
    for (j = 0, i = nLen1 - 1; i >= 0; i--)  // 把字符串转换为整数
        an1[j++] = szLine1[i] - '0';
    int nLen2 = strlen(szLine2);
    for (j = 0, i = nLen2 - 1; i >= 0; i--)
        an2[j++] = szLine2[i] - '0';
```

下标		5	4	3	2	1	0
aResult	- - - -	0	0	**32**	84	47	45

```
    //每一轮都用an2的一位，去和an1各位相乘，从an2的个位开始
    for ( i = 0; i < nLen2; i++ )    {
        for( j = 0; j < nLen1; j++ )
            //两数第i, j位相乘，累加到结果的第i+j位
            aResult[i+j] += an2[i] * an1[j];
    }
    //下面的循环统一处理进位问题
    int nHighestPos = 0;
    for ( i = 0; i < MAX_LEN * 2; i++ )   {  // 请补全循环体代码




        if ( aResult[i] )            nHighestPos = i;
    }

    for ( i = nHighestPos; i >= 0; i -- )        printf("%d", aResult[i]);
    return 0;
}
```

例题：大整数除法

- 问题描述
  - 求2个大的正整数相除的商
- 输入
  - 第1行是测试数据的组数n，每组测试数据占2行，第1行是被除数，第2行是除数。每组测试数据之间有一个空行，每行数据不超过100个字符
- 输出
  - n行，每组测试数据有一行输出是相应的整数商

大整数除法

- 输入样例

3

240533731296337335900926045774205743923049649393035559579766079108273964629871925853187017525844299311608703729070792489710950125097905508837931978941000000000000000000000000000000000000000010000000000

5409656775097850895687056798068970934546546575676768678435435345

1

- 输出样例

0

10000000000000000000000000000000

5409656775097850895687056798068970934546546575676768678435435345

大整数除法

- 基本思想是反复做减法，看看从被除数里最多能减去多少个除数，商就是多少。
- 一个一个减显然太慢，如何减得更快一些呢？
- 以7546除以23为例来看一下：开始商为0
  - 先减去23的100倍，就是2300，发现够减3次，余下646。于是商的值就增加300
  - 然后用646减去230，发现够减2次，余下186。于是商的值增加20，变为320
  - 最后用186减去23，够减8次，因此最终商就是328
- 本题的核心是要写一个大整数的减法函数，然后反复调用该函数进行减法操作

2737 大整数除法


```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

int Substract(int nMaxLen, int *an1, int * an2)
//大整数an1减去an2。两者最多nMaxLen位，an1必须不小于an2, 差放在an1里
//返回差的最高非0位的位置
{
   int i, nStartPos = 0;
   for ( i = 0; i < nMaxLen ; i ++ ) {
      an1[i] -= an2[i];                    //逐位相减
      if ( an1[i] < 0 ) {          //看是否要借位
         an1[i] += 10;
         an1[i+1] --;              //借位
      }
      if ( an1[i] )
         nStartPos = i; //记录最高位的位置
   }
   return nStartPos;   }
```

大整数除法


```
int Length(int nMaxLen, int *an)
// 求大整数的位数，0算0位
{
    int i;
    for ( i = nMaxLen - 1; an[i] == 0 && i >= 0; i-- );
    if ( i >= 0 )          return i + 1;
    return 0;
}

void ShiftLeft(int nMaxLen, int *an1, int *an2, int n)
// 将大整数an1左移n位，即乘以10的n次方，结果放到an2里
{
    int i;
    memcpy(an2, an1, nMaxLen * sizeof(int));
    if ( n <= 0 )          return;
    for ( i = nMaxLen -1; i >= 0; i -- )   {  // 请写出循环体代码




    }
 }
```

大整数除法


```
int * Max(int nMaxLen, int *an1, int *an2)
// 求大整数an1和an2里面大的那个；若an1==an2，返回an1
// 如果都是0，则返回NULL
{
    bool bBothZero = true;
    int i;
    for ( i = nMaxLen -1; i >= 0 ; i -- ) {
        if ( an1[i] > an2[i] )
            return an1;
        else if ( an1[i] < an2[i] )
            return an2;
        else if ( an1[i] )
            bBothZero = false;
    }
    if ( bBothZero)
        return NULL;
    else
        return an1;
}
```

大整数除法


```
#define MAX_LEN 110
int an1[MAX_LEN];    // 存放被除数, an1[0]对应于个位
int an2[MAX_LEN];    // 存放除数
int tmpAn2[MAX_LEN];
int anResult[MAX_LEN];    // 存放商
char szLine1[MAX_LEN];    // 存放被除数的字符串
char szLine2[MAX_LEN];    // 存放除数的字符串
char szNouse[MAX_LEN];

int main()
{
    int n;
    scanf("%d", &n);
    gets(szNouse);
```

大整数除法


```
while(n--) {
    gets(szLine1);
    gets(szLine2);
    gets(szNouse);
    int i, j;
    //库函数memset将地址an1开始的sizeof(an1)字节内容置成0
    memset(an1, 0, sizeof(an1));
    memset(an2, 0, sizeof(an2));
    //下面将szLine1中存储的字符串形式的整数转换到an1中去，
    int nLen1 = strlen( szLine1);
    for (j = 0, i = nLen1 - 1; i >= 0; i--)
        an1[j++] = szLine1[i] - '0';
    int nLen2 = strlen(szLine2);
    for ( j = 0, i = nLen2 - 1; i >= 0; i--)
        an2[j++] = szLine2[i] - '0';
```

大整数除法


```
    int nHighestPos = 0;
    memset(anResult, 0, sizeof(anResult));
    int nShiftLen = Length(MAX_LEN, an1) - Length(MAX_LEN, an2);
    // 只要an1大于an2，就不停相减
    while( Max(MAX_LEN, an1, an2) == an1 ) {
        // 算出an2的10的nShiftLen次方倍
        ShiftLeft(MAX_LEN, an2, tmpAn2, nShiftLen);
        // 重复减去an2的10的nShiftLen次方倍，看能减几次
        while( Max(MAX_LEN, an1, tmpAn2) == an1) {
            Substract(MAX_LEN, an1, tmpAn2);
            anResult[nShiftLen] ++;     // 记录商对应位
        }
        // 记录结果最高位的位置
        if( nHighestPos == 0 && anResult[nShiftLen])
            nHighestPos = nShiftLen;
        nShiftLen--;
    }
    for ( i = nHighestPos; i >= 0; i-- )            printf("%d", anResult[i]);
    printf("\n");
  }
  return 0;
}
```

高精度计算需要注意的几个问题

- 大整数的读入
  - 不能用数值型的读入方式
  - 用字符串的读入方式，再按字符分别转成数字
- 计算过程中考虑进位和借位问题（先不急于处理，在计算结束时处理）
- 输出时注意跳过高位时多余的0
- 数组需要稍微大一些，避免运算时溢出

例题：循环数

- 问题描述
  - 当一个N位的整数X满足下列条件时，称其为循环数：X与任意一个整数i (1≤ i ≤ N)相乘时，都将产生一个X的“循环”。即：分别将这两个整数(X和X*i)的第1位数字与最后1位数字连在一起，可以得到一个相同的数字循环；当然两个整数在该数字循环中的起始位置不同。例如，142857是一个循环数

142857 *1 = 142857
142857 *2 = 285714
142857 *3 = 428571
142857 *4 = 571428
142857 *5 = 714285
142857 *6 = 857142

循环数

- 输入
  - 输入包括多个长度为 2 到 60 位的整数。注意：每个整数前面的零被看作是该整数的一部分，在计算N时要统计。例如 "01"是 2 位数，"1" 是 1 位数。
- 输出
  - 对于每一个输入的整数，输出一行表明它是否是循环数。

循环数

- 输入样例

142857
142856
142858
01
058823529411764 7

- 输出样例

142857 is cyclic
142856 is not cyclic
142858 is not cyclic
01 is not cyclic
058823529411764 7 is cyclic

循环数

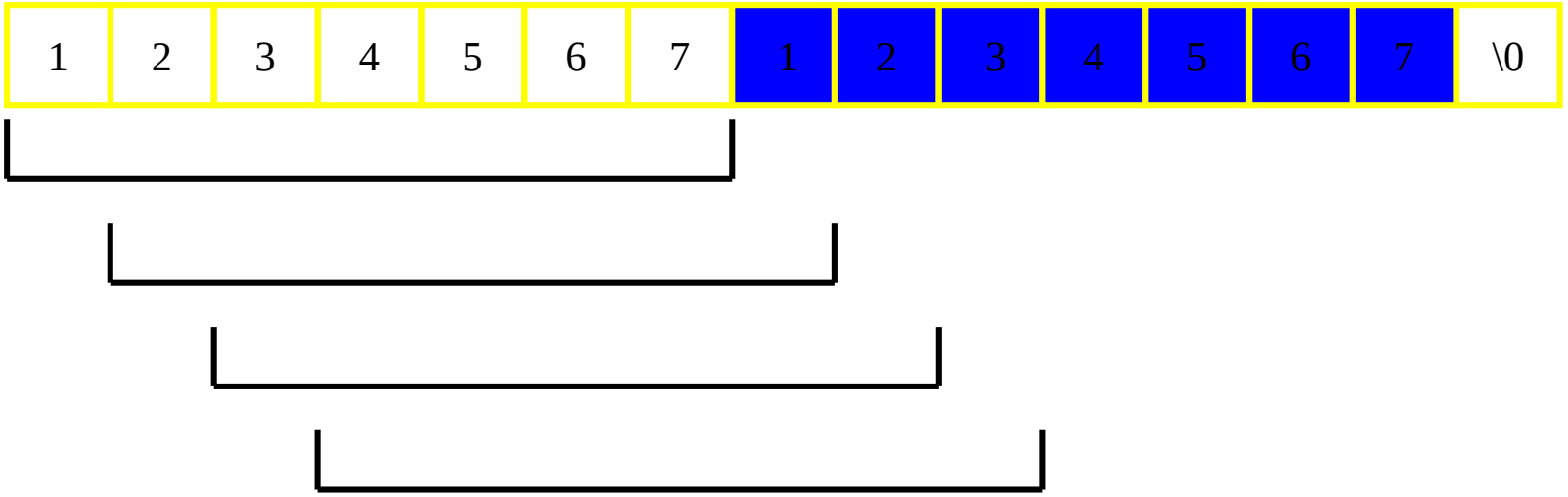
解题思路

- 高精度的乘法：整数可能达60位
  - X*1: $X_1 = X$;
  - X*2: $X_2 = X_1 + X$
  - X*3: $X_3 = X_2 + X$
  - ……
  - X*N: $X_N = X_{N-1} + X$
- X_i是否是X的“循环”？

循环数

X_i是否是X的“循环”？

- 穷举：N位整数，循环移位可以有N种可能
- 循环移位方法：
  - "X_i"是否是"XX"的子串？



2952 循环数


```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX_LEN 61
int an1[MAX_LEN+10];    // 存放Xi = X * i
int an2[MAX_LEN+10];    // 存放X
char szLine1[MAX_LEN+10];   // 存放"X"
char szLine2[MAX_LEN+10];   // 存放"Xi"
char szDouble[2 * MAX_LEN + 10];    // 存放"XX"

//将长度最多为 nMaxLen 的大整数 an1和an2 相加，结果放在an1
int Add(int nMaxLen, int *an1, int *an2);
```

循环数


```
int main()
{
    int i, j;
    while( gets( szLine1 ) && szLine1[0] ) {
        memset(an1, 0, sizeof(an1));
        memset(an2, 0, sizeof(an1));
        // 下面将szLine1中存储的字符串形式的整数转换到an1中去，
        // an1[0]对应于个位
        int nLen1 = strlen( szLine1);   // X的位数
        for (j = 0, i = nLen1 - 1; i >= 0; i--) {
            an1[j] = szLine1[i] - '0';
            an2[j] = szLine1[i] - '0';
            j++;
        }

        strcpy(szDouble, szLine1);
        strcat(szDouble, szLine1);  // 构造"XX“
```

循环数


```
    bool bOk = true;    // 是否循环数
    for ( i = 1; i < nLen1; i++ ) {
        int nHighestPos = Add(MAX_LEN, an1, an2);  // an1存放Xi，an2存放X
        if ( nHighestPos >= nLen1 ) { // 长度超出了
            bOk = false;                break;
        }
        // 判断"Xi"是否是"XX"的子串，请写出代码
    }
    if ( bOk)
        printf( "%s is cyclic\n",szLine1);
    else
        printf( "%s is not cyclic\n",szLine1);
}
return 0; }
```

```
int an1[MAX_LEN+10];    // 存放Xi = X * i
int an2[MAX_LEN+10];    // 存放X
char szLine1[MAX_LEN+10];   // 存放"X"
char szLine2[MAX_LEN+10];   // 存放"Xi"
char szDouble[2 * MAX_LEN+10]; //存放"XX"
```

例题：麦森数

- 形如$M_p=2^p-1$的素数称为麦森数(Mersenne prime)，这时P一定也是个素数。但反过来不一定，即如果P是个素数，2^p-1不一定也是素数
  - 头4个麦森数是3、7、31和127，对应的P为2 、3 、5 和7
  - 已发现48个麦森数，已知的最大麦森数为$2^{57,885,161}-1$，有17,425,170位
  - “互联网梅森素数大搜索”(GIMPS)项目
    - http://www.mersenne.org/
  - 美国EFF(Electronic Frontier Foundation)奖金
    - 超过1000万位数，10万美元；超过1亿位数，15万美元；超过10亿位数，25万美元

麦森数

- 总时间限制: 1000ms 内存限制: 65536kB
- 问题描述
  - 任务：从文件中输入P (1000<P<3100000) ,计算2^P-1的位数和最后500位数字（用十进制高精度数表示）
- 输入
  - 只包含一个整数P(1000<P<3100000)
- 输出
  - 第1行：十进制高精度数2^P-1的位数。
  - 第2-11行：十进制高精度数2^P-1的最后500位数字。每行输出50位，共输出10行，不足500位时高位补0
  - 不必验证2^P-1与P是否为素数。

麦森数

- 输入样例

1279

- 输出样例

386

00000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000
00000000000001040793219466439908192524032736408553
86152622472667048053191123504036080596733602980122
39441732324184842421613954281007791383566248323464
90813990660567732076292412950938922034577318334966
15835504729594205476898112116936771475484788669625
01384438260291732348885311160828538416585028255604
66622483189091880184706822220314052102669843548873
29580288780508697361869007147207105557031687290870

麦森数

- 首先考虑2^p-1有多少位
  - 由于2^p-1的个位数只可能是 2, 4, 6, 8，所以2^p-1和2^p的位数相同
  - 使用math.h中声明的，求以10为底的对数的函数double log10(double x)，就能轻松求得2^p-1的位数
  - $\log10(2^p) = p * \log10(2)$

麦森数

- 输出高精度数2^p-1的最后500为数字
  - 做 p 次乘以 2 的操作，结果严重超时
  - 对于任何 p>0，考虑 p 的二进制形式：

$p = a_0 2^0 + a_1 2^1 + a_2 2^2 + \ldots + a_{n-1} 2^{n-1} + 2^n$

这里，a_i要么是1，要么是0。

于是，

$$2^p = 2^{a_0} \times 2^{2a_1} \times 2^{4a_2} \times 2^{8a_3} \times \cdots\cdots \times 2^{a_{n-1} 2^{n-1}} \times 2^{2^n}$$

麦森数

$$2^p = 2^{a_0} \times 2^{2a_1} \times 2^{4a_2} \times 2^{8a_3} \times \cdots\cdots \times 2^{a_{n-1}2^{n-1}} \times 2^{2^n}$$

- 计算2^p的办法
  - 先将结果的值设为1，计算2^1。如果 a_0 值为1，则结果乘以 2^1；
  - 计算 2^2，如果 a_1 为 1，则结果乘以 2^2；
  - 计算 2^4，如果 a_2 为 1，则结果乘以 2^4；
  - 第 i 步 (i 从 0 到 n，a_n是1)就计算 2^{2^i}，如果 a_i 为1，则结果就乘以 2^{2^i}
    - 每次由$2^{2^i} \times 2^{2^i}$就能算出 $2^{2^{i+1}}$
- 由于 p 可能很大，所以上面的乘法都应该使用高精度计算

麦森数

- 在前面的高精度计算中，用数组来存放大整数，数组的一个元素对应于十进制大整数的一位。
  - 本题如果也这样做，就会超时
- 为了加快计算速度，可以用一个数组元素对应于大整数的 4 位，即将大整数表示为 10000 进制，而数组中的每一个元素就存放 10000 进制数的 1 位。
  - 例如，用 int 型数组 a 来存放整数 6373384，那么只需两个数组元素就可以了，a[0]=3384, a[1]=637
- 由于只要求结果的最后 500 位数字，所以我们不需要计算完整的结果，只需算出最后500 位即可。所以本题中的数组最多只需要125 个元素。

2706 麦森数


```
#include <stdio.h>
#include <string.h>
#include <math.h>
#define LEN 125

// Multiply函数功能是计算高精度乘法a*b，结果的末500位放在a中
void Multiply(int* a, int* b)
{
    int i, j, nTmp;
    int nCarry;  //存放进位
    int c[LEN]; //存放结果的末500位
    memset(c, 0, sizeof(int) * LEN);
    for (i = 0; i < LEN; i++) {     // 请写出循环体代码









    }
    memcpy(a, c, LEN*sizeof(int));
}
```

麦森数

$$2^p = 2^{a_0} \times 2^{2a_1} \times 2^{4a_2} \times 2^{8a_3} \times \cdots\cdots \times 2^{a_{n-1}2^{n-1}} \times 2^{2^n}$$

```
int main() {
    int p, i;
    scanf("%d", &p);
    printf("%d\n", (int)(p*log10(2))+1);    // 输出位数
    // 下面将 2 的次幂初始化为 2^(2^0)，最终结果初始化为 1
    int anPow[LEN];    //存放不断增长的2的次幂
    int aResult[LEN];  //存放最终结果的末500位
    anPow[0] = 2;        aResult[0] = 1;
    for (i = 1; i < LEN; i++) {
        anPow[i] = 0;            aResult[i] = 0;
    }
    // 下面计算2的p次方
    while (p > 0)  { // p==0, 则说明p中的有效位都用过了，不需再算下去
        if  ( p & 1 ) //判断此时 p 中最低位是否为 1
            Multiply(aResult, anPow);
        p >>= 1;
        Multiply(anPow, anPow);
    }
    aResult[0]--; //2的p次方算出后减1
```

麦森数


```
//输出结果
   for (i = LEN-1; i >= 0; i--) {    // 每行输出50位
      if (i % 25 == 12)  //当i%25等于12时，第 i 个万进制位会被折行输出
         printf("%02d\n%02d", aResult[i]/100,
               aResult[i]%100);
      else {
         printf("%04d", aResult[i]);
         if (i % 25 == 0)    // 换行
            printf("\n");
      }
   }
   return 0;
}
```