

16 | 数据结构（上）：突破基本类型的限制，存储更大的整数

2020-02-20 胡光

人人都能学会的编程入门课

[进入课程 >](#)



讲述：胡光

时长 14:07 大小 11.33M



你好，我是胡光，咱们又见面了。

上两节呢，我们讲了素数筛这个算法，并且用素数筛算法演示了程序设计过程中的框架思维。其中提到了欧拉筛法，不知道勤奋的你有没有课后自己去学习一下呢？如果你学习了欧拉筛法以后，你会对我所说的框架思维有更深刻的体会。

在之前的文章中，我们介绍过算法和数据结构的作用。当时我讲到，算法的作用是做数据的计算，并且它对于编程的重要意义，不止是停留在那些叫得上来名字的具体算法上面，☆ 我们称之为的算法思维。

算法思维的具体表现，就是我们处理得到相同信息时，所采用的不同的流程方法。这些方法呢，有好坏高低的比较，而评价的标准，主要就是**从时空复杂度方面来考量**。由于本专栏主要是教会你掌握编程思维，所以，即使你对时空复杂度不是很了解，也不用担心它会影响你的入门编程学习。你只需要知道，这是我们衡量算法好坏的重要指标即可。

前两篇文章呢，其实更多的就是给大家展示算法思维对于程序设计的重要性，并且，我还要在这里提醒一句，算法的底层是数学，适当的补充数学基础，对于算法的学习是有奇效的。

数据结构和算法，前者负责“表示数据”，后者负责“处理数据”。接下来，我将给你讲讲数据结构的重要性。

今日任务

表示数据到底是什么呢？为什么表示数据很重要？通过今天的 10 分钟任务，你就能明白其中的重要意义。这个任务很简单，就是请你实现一个程序，输出 2 的 1000 次方的结果是多少。

关于这个问题，你可能会意识到，C 语言中给我们提供的 int 类型，肯定是无法完成这个任务的，因为它表示不了这么大的数字。你可能想用 long long 类型来进行解决，那你这就要犯低级错误了。long long 是 64 位整型，也就是占 64 个 2 进制位，它顶多能表示 2 的 64 次方减 1 的结果，相对于 2 的 1000 次方来说，小太多了。

你可能又想到，既然 long long 表示不了，那就使用 double，不是说 double 是浮点数类型，可以表示很大很大的数字么？对，double 作为双精度浮点型，确实可以表示很大很大的数字，2 的 1000 次方这个数字，对于 double 的表示范围来说，也是不足挂齿的。

可这里面存在一个严重的问题，就是 double 是有精度损失的。什么意思呢？请耐心听我给你解释。

其实也很好理解，不管是 long long 类型，还是 double 类型，它们都是 64 位的信息，也就是说，它们都可以准确表示 2 的 64 次方个数量的数字。但是，即使 double 类型表示数字的范围比 long long 要大很多，可这个当中很多数字 double 是没有办法准确表示的。

至于 double 的表示精度，一般来说是有效数字 15 位，就是一个数字，由左向右，从第一个不为零的数字起，向后 15 位都是准确的。因此 double 类型实际上也没有办法，准确表

示 2 的 1000 次方的计算结果。

那究竟应该如何来解决今天这个问题呢？带着这个疑问，让我们正式开始今天的释疑之行吧。

必知必会，查缺补漏

前面讲了这么多，我就是想让你明确一点，就是在我们所认识的 C 语言中，是没有任何一种数据类型，可以表示得下我们今天想要计算 2 的 1000 次方的结果。也就是说，基础类型表示不了我们今天所要计算的这个结果，那该怎么办呢？

还记得我讲过的关于结构体的相关知识么？当时我们使用结构体，创造了一个新的代表坐标点的数据类型。按照创造类型的思路去思考现在这个问题，也就是，如果我们能采用一种能够表示更大范围的整数的数字表示法，那今天这个问题，就可以解决了。这就是我们今天要学习的内容，它的大类名字叫做**高精度表示法**，更具体的叫做**大整数表示法**。

1. 大整数表示法

为了完成今天这个任务，我们需要从数据的表示上下功夫。其实，数据的表示绝不是只有一种方法，就好像你想表达数字 1 的一半，你既可以用 0.5 来表示，也可以用 $1/2$ 来表示。所以，今天我们想要表示很大很大的整数，其实也有很多方法，下面就看看我要给你介绍的方法吧。

首先我们先来思考一个事情，如果我想要存储一个 100 位的十进制数字，为什么现有的 int 数据类型做不到？本质上是因为这个数字的位数，超过了 int 能够表示数字的位数上限。int 能够表示的数字大小的上限，是一个以 2 开头的 10 位数字，而我们想要存储的，却是一个 100 位的数字。

看到了这个本质问题后，其实也就找到了解决问题的方向，那就是我们要创造的这种数字的表示方法，能够有足够的空间去容纳更多位数的数字。提起空间，你想到了什么？是不是我们之前讲到的数组？也就是说，我们开辟一个整型数组空间，让这个数组的每个位置存储一位数字，这样是不是就可以很轻松地存储 100 位数字了。

下面就来看看这种大整数表示法，是如何存储数字 3526 的吧：

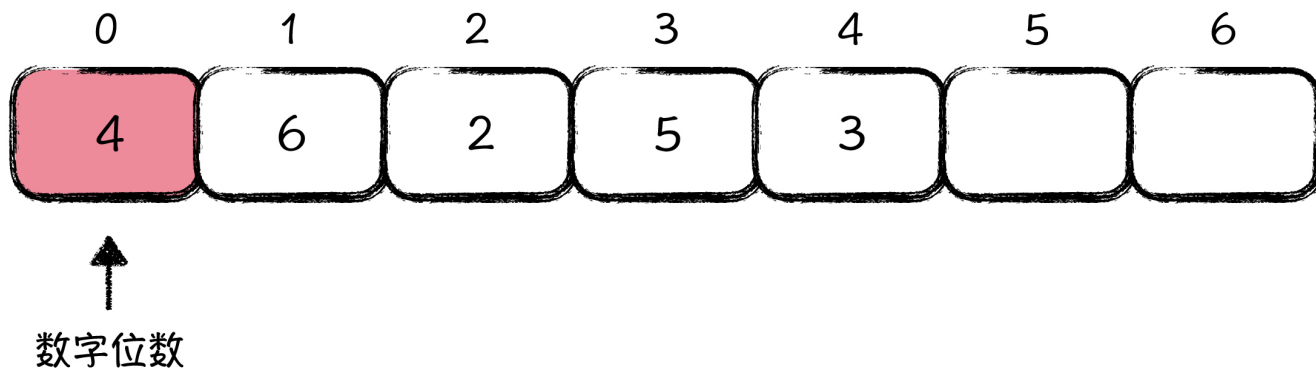


图1：大整数表示示意图

正如你所看到的，这种表示法中，使用数组的第 0 位存储数字的位数，因为 3526 有 4 位，所以数组的第 0 位就设置成了 4 这个值。接下来，数组从第 1 位到第 4 位记录的就是原数字 3526，可是你有没有发现，这个数字是好像是倒着放置的，数字的最高位，也放在数组的最高位中，在图上看着感觉怪怪的。

你可能会觉得别扭，可我要告诉你，这种存储方式，可是凝结了前人的智慧，最直接的一个好处，就是当你拿着两个这样的大整数做加法，产生一个新的大整数的时候，这个新产生的大整数会涉及到进位问题。

例如： $95 + 12 = 107$ ，两个两位的大整数相加，产生一个三位的大整数。在这种从右到左的倒着存储表示法中，是向着数组高位去进位，去扩充位数，这是便利可行的。可你要要是从左到右去正着存储，你会发现一旦最高位产生进位，就很难处理。

2. 如何计算大整数加法

你可能还是不太理解，这种大整数表示法的好处，下面我们就拿“大整数加法”来举个例子。顺便也向你展示一下，我们究竟是如何操作这种大整数。

大整数加法，顾名思义就是利用大整数表式法，做加法运算。具体怎么做，你应该还记得小学时候，老师教给我们的加法竖式吧？其实大整数加法，本质上就是参考这种竖式计算法，把每一位对齐，然后按位相加，加完以后再统一处理进位。下面，我用一张图说明大整数加法，是如何计算 $445 + 9667$ 的：

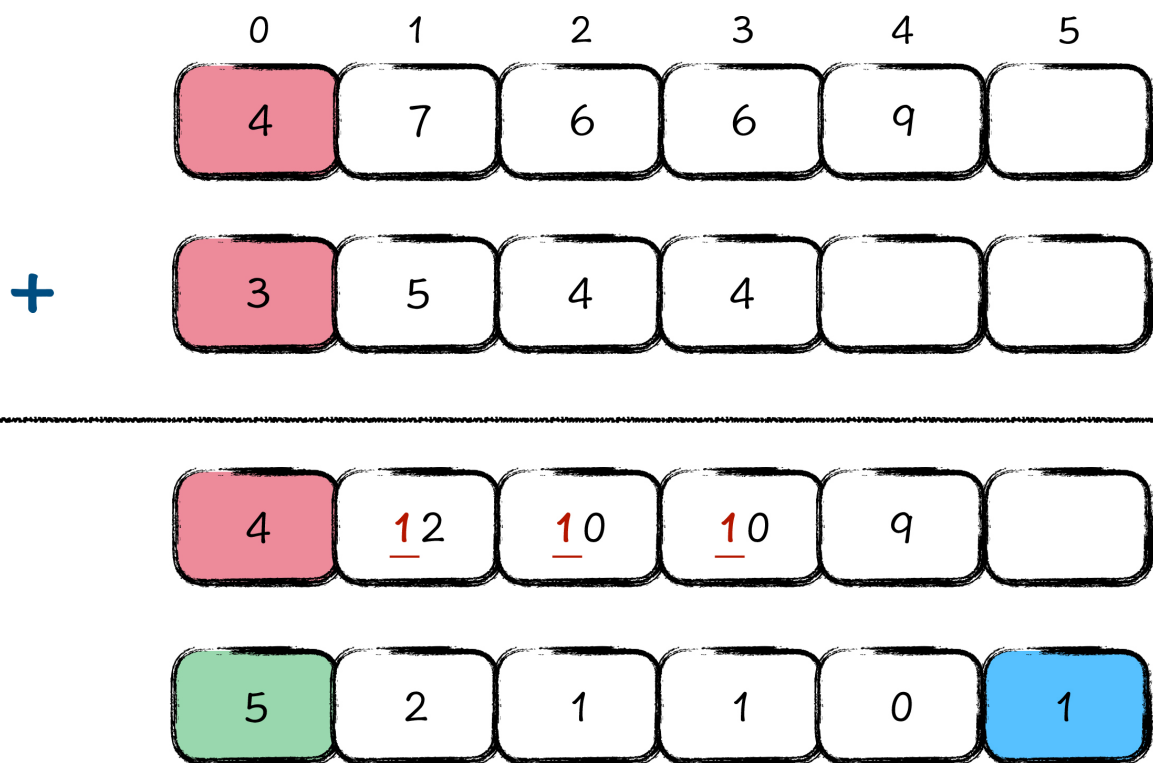


图2：大整数加法示意图

正如你所看到的，首先我们用大整数表示法，分别表示 445 和 9667 这两个数字；然后以位数最长的那个大整数，作为计算结果大整数的基础位数，445 和 9667 按位相加，得到一个 4 位的结果大整数，4 位分别是，9、10、10、12；最后我们再依次处理进位，就得到了底下那一行的结果：10112。

在这个过程中，你会看到最高位的 9 产生了进位，最终变成了一个 5 位的大整数，产生的新最高位，我们只需要继续向后放即可。这就是我刚刚所说的，这种大整数表示法，能够非常方便地处理进位。

看完了大整数加法的过程后，不可缺少的，就是代码的实现过程。下面我给你准备了一份代码，代码中有相关注释，这是需要你自己拿出时间，来进行自学的内容。

[复制代码](#)

```
1 // 定义一个交换两个变量值的宏 swap
2 #define swap(a, b) { \
3     __typeof(a) _t = a; \
4     a = b, b = _t; \
5 }
6 // 实现大整数加法 a + b 的结果，存放在 c 中
7 void plus_big_integer(int *a, int *b, int *c) {
8     // 让 a 指向位数较长的那个数字
9     if (a[0] < b[0]) swap(a, b);
10    // 大整数 c 的位数以 a 的位数为基准
```



```

11     c[0] = a[0];
12     // 循环模拟按位做加法
13     for (int i = 1; i <= a[0]; i++) {
14         if (i <= b[0]) c[i] = a[i] + b[i];
15         else c[i] = a[i];
16     }
17     // 处理每一位的进位过程
18     for (int i = 1; i <= c[0]; i++) {
19         if (c[i] < 10) continue;
20         // 判断是不是最高位产生了进位
21         // 如果是最高位产生进位，就进行初始化
22         if (i == c[0]) c[++c[0]] = 0;
23         c[i + 1] += c[i] / 10;
24         c[i] %= 10;
25     }
26     return ;
27 }

```

一起动手，搞事情

今天给你留的作业题，和我给你准备的那个大整数加法的代码有关。就是请你完成一个，能够实现读入两个大整数，并且输出两个大整数相加之和的程序。关于这个程序作业，你不需要考虑负数的情况，我们假设所有数字均是正整数。

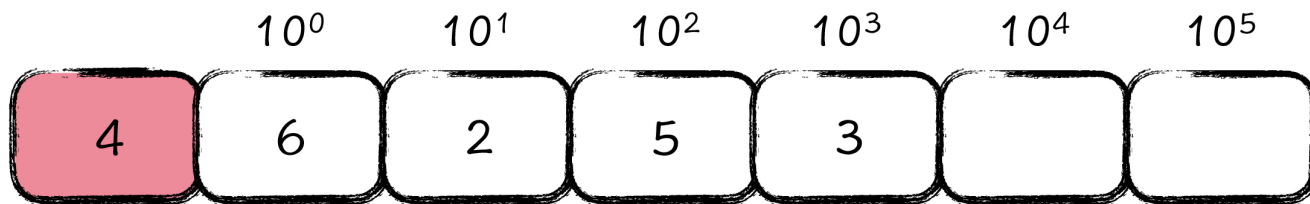
这里给你个提示：在读入两个大整数的时候，你可以按照两个字符串数据进行读入，然后再把字符串数据，转换成我们上面所说的大整数表示法，最后调用上面那个大整数加法的过程。程序的关键提示已经告诉你了，剩下的部分，试试自己完成吧，加油！

突破类型，求解 2^{1000} 的值

最后，我们回到今天的任务。

要计算 2 的 1000 次方的结果，就是要计算 1000 次乘法，最终的结果由于数值太大，我们肯定要使用大整数表示法了。也就是说，我们要在大整数表示法的基础上，操作 1000 次乘法，每次都是乘以 2，那么怎么做大整数乘法呢？

要想理解这个计算过程，我们还是得回到大整数表示法本身，所对应的数学模型理解上，具体请看下图：



$$3 * 10^3 + 5 * 10^2 + 2 * 10^1 + 6 * 10^0$$

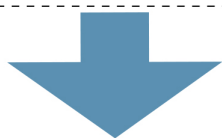
图3：大整数表示法的数学理解

如图所示，我们把大整数表示法中，每一个数字所对应的位权写出来，那么数组中所存储 3、5、2、6 的大整数信息，其实等价于下面那一行数学公式，即

$$3 * 10^3 + 5 * 10^2 + 2 * 10^1 + 6 * 10^0$$

我们对（3526 这个）上面的大整数乘以 2，其实等价于对下面那个数学式子乘以 2，就可以得到如下结果：

$$2 * (3 * 10^3 + 5 * 10^2 + 2 * 10^1 + 6 * 10^0)$$



$$\underline{6} * 10^3 + \underline{10} * 10^2 + \underline{4} * 10^1 + \underline{12} * 10^0$$

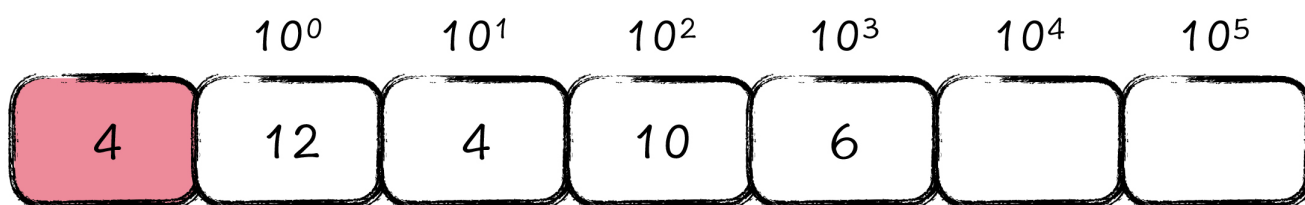



图4：大整数乘法的理解

你会看到，对某个大整数乘 2 的操作，其实，可以看成是对这个大整数的每一位分别乘以 2 的操作，然后再仿照大整数加法的过程，依次处理进位即可。

最后，关于如何完成今天的任务，我给你一个参考程序。当然你也可以选择不看参考程序，自己实现这个过程。

 复制代码

```
1 #include <stdio.h>
2
3 // 将 num 数组初始化成大整数表示的 1
4 // 作用就是做累乘变量
5 int num[400] = {1, 1};
6
7 int main() {
8     // 计算 100 次 2 的 10 次方相乘的结果
9     for (int i = 0; i < 100; i++) {
10         // 对大整数的每一位乘以 2 的 10 次方
11         for (int j = 1; j <= num[0]; j++) num[j] *= 1024;
12         // 处理进位
13         for (int j = 1; j <= num[0]; j++) {
14             if (num[j] < 10) continue;
15             if (j == num[0]) num[++num[0]] = 0;
16             num[j + 1] += num[j] / 10;
17             num[j] %= 10;
18         }
19     }
20     // 输出大整数
21     // 由于大整数是倒着存的，所以输出的时候倒着遍历
22     for (int i = num[0]; i >= 1; --i) printf("%d", num[i]);
23     printf("\n");
24     return 0;
}
```

课程小结

解决了这个任务后，恭喜你，又变强了一点点。今天我们学习了大整数的表示法，以及大整数加法和乘法的基本操作，我希望你记住以下几点：

1. 在大整数的表示法中，数字是从右到左，倒着存放在数组中的。
2. 大整数的表示法，体现的是数据结构对于程序设计的作用。
3. 大整数的加法和乘法过程，体现的则是算法对于程序设计的作用。

同时，你还可以看到，我们在理解大整数乘法的过程中，是从数组的表示法与数学公式的等价性这个角度出发讨论的。其实我就是想再次跟你强调那句话，就是**算法的底层是数学**。

而通过今天的学习，想必你已经对“**数据结构本质是用作数据的表示**”这句话，已经有所感觉了。综合“**算法是做数据的计算**”这句话，说明算法和数据结构是程序中可以独立进行设计的两个部分，关于这点呢，将是下一节咱们讲解的重点。

好了，今天就到这里了，我是胡光，我们下期见。

课程学习计划

关注极客时间服务号 每日学习签到

月领 25+ 极客币

【点击】保存图片，打开【微信】扫码>>>



© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 15 | 框架思维（下）：用筛法求解其他积性函数

下一篇 17 | 数据结构（下）：大整数实战，提升 Shift-And 算法能力

精选留言 (2)

写留言



徐洲更

2020-02-22

对于原来的数据类型而言，我们的数组12345，底层使用64位存储的。而为了突破数据类型本身的限制，存放更大的数字，我们新建了一个数组，数组每个元素大小也是64位的话，也就是说为了表示12345，实际上使用了更多的内存空间。

那么，有没有通过二进制本身，比如说在底层搞一个新的类型，比如说super long, 用128位或者更多位来表示这个大数据类型的方式呢？

展开 ∨

作者回复: 这个是有的。GNU 标准下, 是支持 128 位整型数据的。对于你提出的内存浪费的事情, 其实文中给出的是最基本的一种实现, 真正应用的时候, 你可以尝试每一个整型位置存储若干位的数字,
例如: 123456--> 12 | 34 | 56



😊HappyJoo

2020-02-20

老师您好, 可以麻烦您有空的话帮我解决几个问题吗? 谢谢! :

1, 在标准C中是以 “_” 开头, 所以在标准C中要写成 “_typeof()” 或 “_typeof_()”, 在GNU C 中还支持直接写 “typeof()”, 我看您写的都是 “_typeof()”, 其实三个都是一样的, 但是否为了不造成不必要的麻烦, 才用了标准C的第一个呢?

...

展开 ∨

作者回复: 1、是的, 之前不注意的时候, 踩过坑, 后来习惯养成了, 就顺手写 _typeof了。

2、对于加法来说确实可以, 写成%=只是为了不增加你们的学习负担, 否则加法学一套, 乘法又得学一套。你能发现这其中的问题, 说明你是很认真看了的! 给你点赞!

3、你理解的是对的。

我在手机上操作的, 不方便打开你的链接, 我盲猜一下, cnt是用来记录素数个数的。

