

第八章 高精度算法（1）

1. 高精度算法

1.1 高精度算法

高精度算法法是一种处理超过计算机固定位数范围的大整数的计算方法，它可以用于处理大整数的加减 乘除、比较大 小等运算，以及求解大整数的阶乘、幕运算等问题。当参与运算的数据超出 C++ 数据类型的取值范围时，我们就会 用到高精度算法

C++各数据类型取值范围表：

类型	符号	关键字	占字节数	数据范围
整型	有符号	int	4	-2147483648 ~ 2147483647
		long long	8	-9223372036854775808 ~ 9223372036854775807
	无符号	unsigned int	4	0 ~ 4294967295
		unsigned long long	8	0 ~ 1844674407370955161
浮点型	有符号	float	4	3.4E-38 ~ 3.4E+38
		double	8	1.7E-308 ~ 1.7E+308
字符型	有符号	char	1	-128 ~ 127

1.2 高精度算法的注意事项

高精度运算需要注意的问题

数据的接收和存储：

当需要处理的数据超过了任何数据类型所能容纳的范围，可以采用字符串形式输入，并将其中每一位字符 转化为数 字，存入数组中；

代码实现：数字字符串形式输入， 将其转化为整型数组，并逆序存储

逆序存储的原因：

- 1. 使两个参与运算的数据的个位对齐
- 2. 方便处理最高位运算后的进位
- 高精度位数的确定：
运算规则如同算术运算。由于高精度运算的结果可能使数据长度发生增减， 因此不仅需要用整型数组存储数据， 还需要记录整型数组的元素个数，也就是数据实际长度

```
int init(int a[])
{
    string s; //声明数字串
    cin >> s;
    int len = s.length();//len 存储数字串S 的位数 （数字串长度）
    for(int i =0; i < len; ++i)
    {
        a[i] = s[len - i - 1] - '0';//将数字串S转化为数组 a，并倒序存储。
    }
    return len; //返回数字串长度
}
```

- 除了使用变量来存储数据的长度之外， 还可以将数据的长度存储在整形数组下标0的位置：

```
int init(int a[])
{
    string s; //声明数字串
    cin >> s;
    int a[0] = s.length(); //a[0]存储数字串S 的位数 （数字串长度）
    for(int i = 0; i < len; ++i)
    {
        a[i+1] = s[len - i - 1] - '0'; //将数字串S转化为数组 a, 并倒序存储.
    }
    return len; //返回数字串长度
}
```

- 对于有符号高精度数据的运算，除了记录数据本身以外， 还需要记录数据的正负号， 可以使用结构体存储数据

```
struct Data
{
    int flag; // 记录数据的正负号，flag 为 1表示正，为0表示负
    int a[N]; // 记录数据本身
};
```

2. 高精度加法

2.1 基本思想

高精度加法算法的基本思想是将大整数按位进行计算， 从低位到高位逐个相加， 并考虑进位的情况
具体步骤如下：

1. 将相加的数据以字符串的形式读入
 2. 将字符串翻转存入整型数组中， 两数之和的位数最大为较大数的位数加1；
 3. 从低位到高位依次两两相加后， 再加上上一位的进位， 将和模 10 作为本位结果， 将和除以10 作为下一位的进位；
 4. 重复步骤3， 直到所有位数处理完；
 5. 从后往前找结果数组中第一个非0元素的位置（即去前导0的操作）， 从该位置开始逆序输出数 组中的元素， 即为最终的加法结果；
- 通过以上步骤， 可以实现对大整数的精确加法运算

- 加法竖式计算866 + 265的过程

866

+ 265

11

13

11

1131

个位：6 + 5 = 11，个位结果为 1，进位 1
十位：6 + 6 + 1 = 13，十位结果为 3，进位 1
百位：8 + 2 + 1 = 11，百位结果为 1，进位 1
将最后的进位 1 添加到千位；
故最终答案为 1131。

- 高精度加法步骤
用高精度算法计算 866 + 265 的结果。
1. 先将两个数据以字符串的形式存储；
 string s1 = "866", s2 = "265";
2. 将字符串翻转存入整型数组中；
 int A[100] = {6,6,8} , B[100] = {5,6,2} , C[101] = { }（存储结果数组）；
3. 从低位到高位两两相加，同时处理进位；

A_2

A_1

A_0

$+ B_2$

B_1

B_0

C_3

C_2

C_1

C_0

$A[0] + B[0] = 6 + 5 = 11$

$A[1] + B[1] + x = 6 + 6 + 1 = 13$

$A[2] + B[2] + x = 8 + 2 + 1 = 11$

当前位 $C_0 = 1$ ，进位 $x = 1$

当前位 $C_1 = 3$ ，进位 $x = 1$

当前位 $C_2 = 1$ ，进位 $x = 1$

当前位 $C_3 = 1$

4. 把存储结果的数组从高位到低位依次输出。

代码示例：

```

#include<iostream>
#include<string>
using namespace std;
int A[201], B[201], C[202];
string s1, s2;
int init(int a[], string &s)
{
    cin >> s; //读入字符串
    int len = s.size(); //计算数字串位数
    for(int i = 0; i < len; i++)
    {
        //将数字串逐位转换为数字并逆序存放
        a[i] = s[len - 1 - i] - '0';
    }
    return len; // 返回数字串长度
}

int main()
{
    int lena = init(A, s1); //计算A数字串长度
    int lenb = init(B, s2); //计算B数字串长度
    int lenc = max(lena, lenb); //查找最长数字串长度
    int x = 0; //表示进位
    for(int i = 0; i < lenc; ++i)
    {
        C[i] = A[i] + B[i] + x;
        x = C[i] / 10;
        C[i] %= 10;
    }
    C[lenc] = x; // 最高位的进位放到C[lenc]
    while(C[lenc] == 0 && lenc > 0) lenc--; //去前导0
    for(int i = lenc; i >= 0; i--) cout << C[i]; // 逆序输出结果
    return 0;
}

```

3. 高精度减法

3.1 高精度减法

高精度减法的基本思想是将大整数按位进行计算，从低位到高位逐个相减，并考虑借位的情况
具体步骤如下

1. 将参与运算的数据以字符串的形式读入
2. 将字符串翻转存入整型数组中，两数之差的位数最大为较大数的位数；
3. 如果是小数减大数，则交换两数位置，并输出负号
4. 从低位到高位逐位求差，如果某位不够减需借位，那么高位减1，本位加 10 再相减
5. 重复步骤4直到所有位数减完；
6. 从后往前找结果数组中第一个非0元素的位置（即去前导0 操作），从该位置开始逆序输出数组中的元素，即为最终的减法结果

- 减法竖式计算866 - 775的过程

1

866

- 775

91

个位：6 - 5 = 1，个位结果为 1；
十位：不够减，向百位借一当十，16 - 7 = 9，十位结果为 9，百位减 1；
百位：(8 - 1) - 7 = 0，百位结果为 0；
故最终答案为 91。

- 高精减法步骤

用高精度算法计算866 - 775 的结果。

1. 先将两个数据以字符串的形式存储；

```
string s1 = "866", s2 = "775";
```

2. 将字符串翻转存入整型数组中；

```
int A[100] = {6,6,8} , B[100] = {5,7,7} , C[101] = { } (存储结果数组) ;
```

3. 从低位到高位逐位求差，如果差小于 0，向高位借一当十，高位减 1；

A₂

A₁

A₀

- B₂

B₁

B₀

C₂

C₁

C₀

A[0] - B[0] = 6 - 5 = 1 当前位 C₀ = 1

A₁ - B₁ 小于0, 向高位借一当十, A₁ += 10, 则

A[1] - B[1] = 16 - 7 = 9 当前位 C₁ = 9

A[2] - B[2] = (8 - 1) - 7 = 0 当前位 C₂ = 0

4. 把存储结果的数组从高位到低位依次输出（减法最终结果要去前导 0 后输出）。

代码示例：

file:///Users/silhouetteof/Library/CloudStorage/OneDrive-个人/CppTeaching/01algorithm&datastructure/8高精度算法1.md

5/6

```

#include<iostream>
#include<string>
using namespace std;
int A[201], B[201], C[202];
string s1, s2;
int init(int a[], string &s)
{
    cin >> s; //读入字符串
    int len = s.size(); //计算数字串位数
    for(int i = 0; i < len; i++)
    {
        //将数字串逐位转换为数字并逆序存放
        a[i] = s[len - 1 - i] - '0';
    }
    return len; // 返回数字串长度
}
void Sub() //减法运算过程
{
    for(int i=0; i < lenc; ++i)
    {
        C[i] += A[i] - B[i];
        // 如果不够减
        if(C[i] < 0)
        {
            C[i+1]--;
            C[i] += 10;
        }
    }
}
int main()
{
    int lena = init(A, s1); //计算A数字串长度
    int lenb = init(B, s2); //计算B数字串长度
    int lenc = max(lena, lenb); //查找最长数字串长度 int X

    if(lenc < lenb || (lena == lenb && s1 < s2))
    {
        // 如果是小数减大数 交换
        swap(A, B);
        cout << "-";
    }
    Sub();
    while(C[lenc] == 0 && lenc > 0) lenc--; //去前导0
    for(int i = lenc; i >= 0; i--) cout << C[i]; // 逆序输出结果
    return 0;
}

```