

结构体中使用 `sort()` 函数:

1、`sort`内写函数重建比较规则

2、结构体内重载比较运算符 改变比较规则

第二种比第一种效率要高



实验舱
青少年编程
走近科学 走进名校

实验舱蛟龙三班

Day9 高精度 加减运算

zlj

2022

为什么要用高精度

类 型 名	类型的大小(字节)	值 域
signed char	1	-128~127
short short int	2	-256~255
int	4	-2 147 483 648~+2 147 483 647
long long int	4	-2 147 483 648~+2 147 483 647
long long long long int	8	-9 223 372 036 854 775 808~9 223 372 036 854 775 807

为什么要用高精度

型标识符	类型说明	长度（字节）	大小（范围）	备注
char	字符型	1	-128 ~ 127	-27 ~ (27 - 1)
unsigned char	无符字符型	1	0 ~ 255	0 ~ (28 - 1)
short int	短整型	2	-32768 ~ 32767	2-15 ~ (215 - 1)
unsigned short int	无符短整型	2	0 ~ 65535	0 ~ (216 - 1)
int	整型	4	-2147483648 ~ 2147483647	-231 ~ (231 - 1)
unsigned int	无符整型	4	0 ~ 4294967295	0 ~ (232-1)
float	实型（单精度）	4	$1.18 \times 10^{-38} \sim 3.40 \times 10^{38}$	7位有效位
double	实型（双精度）	8	$2.23 \times 10^{-308} \sim 1.79 \times 10^{308}$	15位有效位
long double	实型（长双精度）	10	$3.37 \times 10^{-4932} \sim 1.18 \times 10^{4932}$	19位有效位

一、高精度加法

思考：现有两个不超过100位的正整数，要求你将它们加起来的和输出。

【输入】

39827384

3048503945903485

【输出】

3048503985730869

$$\begin{array}{rcccc} & & 5 & 3 & 7 \\ + & & 5 & 4 & 3 \\ \hline 1 & 0 & 8 & 0 & \end{array}$$

高精度加法

$$\begin{array}{r} 5 6 7 \\ + 6 2 8 9 \\ \hline 6 7 14 16 \end{array}$$

问题：

- 1、如何存数？
- 2、按位相加，结果如何存？
- 3、如何进位？
- 4、如何输出结果？

高精度加法

	5	6	7	
+	6	2	8	9

6	7	14	16
---	---	----	----

【算法分析】

- 1、输入并转换为int 数组
- 2、 `int a[500], b[500], c[500]`
- 3、按位相加：`c[i]=a[i]+b[i]`
- 4、如何进位？
`jw=(c[i]+jw)/10;`
`c[i]=(c[i]+jw)%10;`
- 5、按位输出结果。

```
3  const int N=10005;
4  int a[N]= {0},b[N]= {0},c[N]= {0};
5  int main() { // 字符串模拟高精度加法
6      string x,y;
7      cin>>x>>y; // 字符串输入加数
8      int len=max(x.size(),y.size());
9      reverse(x.begin(),x.end());
10     reverse(y.begin(),y.end());
11     for(int i=0; i<x.size(); i++) {
12         a[i]=x[i]-'0'; // 分离数字
13     }
14     for(int i=0; i<y.size(); i++) {
15         b[i]=y[i]-'0';
16     }
17     for(int i=0; i<len; i++) {
18         c[i]=a[i]+b[i]; // 按位相加
19     }
20 }
```



```
21 int  jw=0; //处理进位
22 for(int i=0; i<len; i++) {
23     c[i]=c[i]+jw;
24     jw=c[i]/10;
25     c[i]=c[i]%10;
26 }
27 if(jw) { //处理最高位
28     c[len++]=jw;
29 }
30 for(int i=len-1; i>=0; i--) { //输出
31     cout<<c[i];
32 }
33 cout<<endl;
34 return 0;
```

二、高精度减法

现有两个10000位的正整数a和b，要求你将它们做减法后输出。

【输入】

3048503945903485

39827384

【输出】

3048503906076101

$$\begin{array}{r} 5 3 3 \\ - 3 4 7 \\ \hline 1 8 6 \end{array}$$

高精度减法

$$\begin{array}{r} 5 \quad 3 \quad 3 \\ - 3 \quad 4 \quad 7 \\ \hline 2 \quad -1 \quad -4 \end{array}$$

1 1

【算法分析】

- 1、字符串转int 数组a, b, c
- 2、按位相减: $c[i] = a[i] - b[i]$
- 3、如何借位:
如果 $c[i] < 0$:
 $c[i] = c[i] + 10$;
 $a[i+1] = a[i+1] - 1$;
- 4、按位输出
考虑前序 0;
- 5、考虑结果负数情况

字符串模拟减法

```
4 int a[N]= {0},b[N]= {0},c[N]= {0};
5 int main() { //字符串模拟高精度减法
6     string x,y;
7     cin>>x>>y; //字符串输入
8     int len1=x.size(),len2=y.size(),pd=0;
9     if(len1<len2 || len1==len2&& x<y){ //大数放前, 判断负数
10         swap(x,y);pd=1;
11     }
12     int len=max(x.size(),y.size());
13     reverse(x.begin(),x.end());
14     reverse(y.begin(),y.end());
15     for(int i=0; i<x.size(); i++) {
16         a[i]=x[i]-'0'; //分离数字
17     }
18     for(int i=0; i<y.size(); i++) {
19         b[i]=y[i]-'0';
20     }
21     for(int i=0; i<len; i++) {
22         c[i]=a[i]-b[i]; //按位相减
23     }
24 }
```

字符串模拟减法

```
25 // //处理借位
26 for(int i=0; i<len; i++) {
27     if(c[i]<0){
28         c[i]=c[i]+10;c[i+1]--;
29     }
30 }
31 while(!c[len-1]&&len>1){
32     len--;
33 }
34 if(pd)cout<<' - ';
35 for(int i=len-1; i>=0; i--) { //输出
36     cout<<c[i];
37 }
38 cout<<endl;
39 return 0;
```

例6:喜加一

- 小A今天在路上捡到了一块钱，于是他准备把钱存进自己的银行卡里（拾金不昧才是良好品德），他想知道存入这一块钱后，卡里有多少钱，虽然问题很简单，但是因为小A是个土豪，所以他的银行卡里的钱比较多，所以心算比较复杂，想请你帮他算一算
- 文件输入
- 第一行一个正整数K，表示小A银行卡里的钱数
- 文件输出
- 输出小A银行卡里的钱数+1后的结果
- 输入样例
- 123
- 输出样例
- 124
- 数据规模
- 对于前20%的数据， $K \leq 1000$
- 对于前30%的数据， $K \leq 10^{12}$
- 对于前100%的数据， $K \leq 10^{9999}$

例7： 很大的斐波那契数

输入一个整数N，求fib(N)的值， $n \leq 50000$;

样例：

100

输出：

354224848179261915075

```

2  using namespace std; //很大的斐波那契数
3  string add(string a, string b) { //高精加法
4      string ans; int i=0, jw=0;
5      if(a.length() < b.length()) (1); //大的在前
6      reverse(a.begin(), a.end());
7      reverse(b.begin(), b.end());
8      while(i < b.length()) { //处理长度相等部分
9          int a1=a[i]-'0', b1=b[i]-'0';
10         int cur=(2);
11         ans+='0'+cur%10;
12         jw=cur/10;
13         i++;
14     }
15     while(i < (3)) { //处理较长的部分
16         int cur=a[i]-'0'+jw;
17         ans+='0'+cur%10;
18         jw=cur/10;
19         i++;
20     }
21     if((4)) // 处理进位
22         ans+='0'+jw;
23     reverse(ans.begin(), ans.end());
24     return ans; // 返回相加结果
25 }

```

```

26 string x[1000005];
27 int main() {
28     int n;
29     cin >> n;
30     x[1]='1', x[2]='1';
31     for(int i=3; i<=n; i++) { //递推求fib(n)
32         x[i]=add(x[i-1], x[i-2]);
33     }
34     cout << x[n];
35     return 0;
36 }

```


三、高精度（加减）之运算符重载

高精度 用重载运算符写，较为直观与方便

重载运算符 '+' '-' 重新定义运算符的含义，让它具有大整数加减的规则。

知识准备：

结构体的使用，包括定义、初始化和自定义内部函数
字符串的输入与转化

```
Struct bigNum{  
    int len, x[N] = {0} ;  
    void pint() {  
        for (int i= len-1;i>=0;i--) printf("%d",x[i]) ;puts("") ;  
    }  
} a,b,c;
```

重载运算符（高精度+高精度）

```
inline bigNum operator +(bigNum a , bigNum b) {  
    bigNum c ;  
    c.len= max(a.len, b.len) ;  
    for (inti= 0;i< c.len;++ i) c.x[i] = a.x[i] + b.x[i] ;  
    （进位代码）  
    return C ;  
} // inline 定义内置函数
```

进位调整：加法

进位代码：

```
int i, jw=0;
for ( i= 0;i< c.len; i++ ) {
    jw= ( jw+c.x[i] ) /10;
    c.x[i] =c.x[i]%10;
}
if (jw)c.len++, c.x[c.len]=jw;

}
```

重载运算符（高精度 - 高精度）

```
inline bigNum operator - (bigNum a , bigNum b) {  
    bigNum c ;  
    c.len= max(a.len, b.len) ;  
    for (inti= 0;i< c.len;++ i) c.x[i] = a.x[i] - b.x[i] ;  
    return fix(c) ;  
}
```

进位调整：减法

```
inline bigNum fix(bigNum a) {  
    for (int i= 0;i< a.len;++ i) {  
        if (a.x[i] <0) a.x[i]+= 10 , a.x[i+1] --; //借位  
    }  
    while (!a.x[a.len] && a.len>1) a.len--;  
    return a ;  
}
```

读入大整数 a,b;

```
32  bignum read(){  
33      string s;  
34      cin>>s;  
35      reverse(s.begin(),s.end());  
36      bignum a ;  
37      a.len=s.size();  
38      for(int i=0;i<s.size();++i){  
39          a.x[i]=s[i]-'0';  
40      }  
41      return a;  
42  }
```

```
3 bool check(string a,string b){
4     if(a==b)return true;
5     if(a.size()!=b.size())return false;
6     if(a.size()%2)return false;
7     string a1,a2,b1,b2;int len=a.size()/2;
8     a1=a.substr(0,len),a2=a.substr(len,len);
9     b1=b.substr(0,len),b2=b.substr(len,len);
10    return check(a1,b1)&&check(a2,b2)||check(a1,b2)&&check(a2,b1);
11 }
```

字符串弱等于

```
int main() {
    int t;cin>>t;
    while(t--){
        string a,b;
        cin>>a>>b;
        if( check(a,b)==true ){
            cout<<"YES"<<endl;
        }else{
            cout<<"NO"<<endl;
        }
    }
}
```



```

3 char a[205][205];
4 int main() {
5     int n; string s;
6     cin >> n;
7     cin >> s;
8     int row = s.size() / n, k = 0;
9     // cout << row << endl;
10    // 还原矩阵
11    for (int i = 1; i <= row; i++) {
12        if (i % 2 == 1) {
13            for (int j = 1; j <= n; j++) {
14                a[i][j] = s[k++];
15            }
16        } else {
17            for (int j = n; j >= 1; j--) {
18                a[i][j] = s[k++];
19            }
20        }
21    }
22 }

```

反反复复

```

// 输出
for (int j = 1; j <= n; j++) {
    for (int i = 1; i <= row; i++) {
        cout << a[i][j];
    }
}
return 0;

```