A Simple Calculator

12112609 刘一郴

1. Code Explanation

1.1 Struct BigNum

1.1.1BigNum is struct to store big num having basic parameters.

```
struct BigNum{    //a struct to store big num
    int size;     //size of a[N]
    bool pos;    //if the value is positive
    int a[N];    //store value bit by bit like 0 1 2 3 equal to 3210
    ll e;
```

1.1.2 Defined function optimize can adjust size and a[n] to delete the pre and suf zeros. **And if it is all zeros, which means its value is 0, we just manipulate it manually.**

```
    void optimize(){ //adjust size a[n] to delete the pre and suf zero
        while(a[size- 1]== 0){
            size--;
            if(size== 0){
                break;
            }
        }
        int num= 0;
        while(a[num]== 0&& num< size){
            num++;
        }
        For(i, num, size- 1){
            a[i- num]= a[i];
        }
        e+= num;
        size-= num;

        if(size== 0){   //avoid there is null in a[n]
            pos= 1;
            a[size++]= 0;
            e= 0;
        }
```

1.1.3Reloaded operator * which return a new

BigNum equal to BigNum1 and BigNum2.

```
BigNum operator* (const BigNum o){  //define * operation
    BigNum ans;
    ans.size= size+ o.size;
    ans.pos= !(pos^ o.pos);
    ans.e= e+ o.e;
    For(i, 0, ans.size- 1){
        ans.a[i]= 0;
    }
    For(i, 0, size- 1){
        For(j, 0, o.size- 1){
            ans.a[i+ j]+= a[i]* o.a[j];
        }
    }
    For(i, 1, ans.size- 1){
        ans.a[i]+= ans.a[i- 1]/ 10;
        ans.a[i- 1]%= 10;
    }
    ans.optimize();
    return ans;
}
```

1.1.4 Defined function read can intialize the parameters to equal to given string s (the input must be valid).

```
    void read(string s){  //intialize BigNum with string s (input must be valid)
        //find the length of s
        int len= s.length();
        //find the 'e''s position
        int ePos= -1;
        int n= s.length();
        For(i, 0, n- 1){
            if(s[i]== 'e'){
                ePos= i;
            }
        }
        //find if it is positive or not
        pos= s[0]!= '-';
        //determine e from give string's suffix if having
        if(ePos== -1){
            e= 0;
        }else {
            e= readString(s.substr(ePos+ 1, len- ePos- 1) );
```

```
        }
        //dertermine size a[n], adjust e form string's prefix
        len= (ePos== -1)? len: ePos; //omit the exxx part
        size= 0;
        int dotPos= -1;
        For(i, 0, len- 1){
            if(s[i]== '-'){
                continue;
            }
            if(s[i]== '.'){
                dotPos= i;
                continue;
            }
            a[size++]= (s[i]- '0');
        }
        if(dotPos!= -1){//update e
            e-= len- 1- dotPos;
        }
```

```
        For(i, 1, size/ 2){ //reverse a[n]
            swap(a[i- 1], a[size- i]);
        }

        optimize();//delete pre and suf zero
    }
```

1.1.5 Defined function print can print the value in the format like 213e12321 or just 21300 or or 0.0123. **Only when the normal print is to long, we print it with 'e' format like 2130000000000 into 213e10, also like 0.0000000000213 into 213e-13.**

```
    void print(){ //print the value
        if(!pos){
            cout<< '-';
        }

        if(e>= 0&& e<= 5){            //like 123000
            Ford(i, size- 1, 0){
                cout<< a[i];
            }
            For(i, 1, e){
                cout<< 0;
            }
        }else{
            if(-e>= size&& -e<= size+ 5){ //like 0.0123
                cout<< "0.";
                For(i, 1, -e- size){
                    cout<< 0;
                }
                Ford(i, size- 1, 0){
                    cout<< a[i];
```

```
                }
            }else if(-e< size&& -e> 0){ //like 1.23
                Ford(i, size- 1, -e){
                    cout<< a[i];
                }
                cout<< '.';
                Ford(i, -e- 1, 0){
                    cout<< a[i];
                }
            }else{                      //like 123e123
                Ford(i, size- 1, 0){
                    cout<< a[i];
                }
                cout<< 'e'<< e;
            }
        }
    }
};
```

1.1.5 A functional function readString, just used to assign value to e, can transform a string-like long long integer into long long integer type.

```
ll readString(string s){  //turn string into Long Long Integer
    ll n= 0, b= 1;
    int len= s.length();
    int i= 0;
    char c= s[0];
    while(!isdigit(c) ){
        if(c== '-') b= -1;
        c= s[++i];
    }
    while(isdigit(c) ){
        n= n* 10;
        n+= c- 48;
        c= s[++i];
    }
    return n* b;
}
```

1.2 Valid judgements

1.2.1 Defined function isPureNum can return ture **IFF** all char of given string are digits.

```
bool isPureNum(string s){ //return ture IFF all char of given string are digits
    int len= s.length();
    //boundary1 length= 0;
    if(len== 0){
        return false;
    }
    //boundary2 all is digit
    int flag= true;
    For(i, 0, len- 1){
        flag&= isdigit(s[i]);
    }
    if(flag){
        return true;
    }else{
        return false;
    }
}
```

1.2.2 Defined function isNum can return true **IFF** string is in '-123' or '-1.23'-like format.

```
bool isNum(string s){ //return true IFF string is in '-123' or '-1.23'-like format
    if(s[0]== '-'){  //omit the '-' in 0 index;
        s= s.substr(1, s.length()- 1);
    }

    int len= s.length();
    For(i, 0, len- 1){  //if there is one '.', then ture IFF two parts are purenum(all is digit)
        if(s[i]== '.'){
            return isPureNum(s.substr(0, i) )&& isPureNum(s.substr(i+ 1, len- i) );
        }
    }

    //no '.', then ture IFF this is purenum(all is digit)
    return isPureNum(s);
}
```

1.2.3 Defined function isValid can return true **IFF** string is in '-123' or '-1.23' or '-1.23e-12'-like format.

```
bool isvalid(string s){  //check if s is valid
    int len= s.length();
    int eNum= 0;
    int ePos= -1;
    For(i, 0, len- 1){
        if(s[i]== 'e'){     //if number of 'e' >0 return false;
            eNum++;
            ePos= i;
            if(eNum> 1){
                return false;
            }
        }
    }

    if(eNum== 0){  //if no 'e' return if s is num, if not return if two parts are nums
        return isNum(s);
    }else{
        return isNum(s.substr(0, ePos) )&& isIntNum(s.substr(ePos+ 1, len- ePos- 1) );
    }
}
```

1.2.4 Defined function isIntNum can return true **IFF** string is in '-123'-like integer format.

```
bool isIntNum(string s){//return true IFF string is in '-123'-like format
    if(s[0]== '-'){  //omit the '-' in 0 index;
        s= s.substr(1, s.length()- 1);
    }
    return isPureNum(s);
}
```

1.3 Main

For two string, check if it is valid. Then transform two string into BigNum. Then multiply this two BigNum and print the result BigNum.

```
int main(int argc, char* argv[])
{
        string s1, s2;
        if(argc< 3){
            cout<< "please input two string!"<< endl;
            return 0;
        }
        s1= argv[1];
        s2= argv[2];
        if(!isvalid(s1)|| !isvalid(s2) ){  //determine if it is valid num
            cout<< "The input cannot be interpret as numbers!"<< endl;
            return 0;
        }
        BigNum bn1, bn2, bn3;
        bn1.read(s1);
        bn2.read(s2);
        bn3= bn1* bn2;
        cout<< s1;
        cout<< " * ";
        cout<< s2;
        cout<< " = ";
        bn3.print();
        cout<< endl;

    return 0;
}
```

2. Requirements

2.1 When you run the program as follows, it will output the expression and the result. The two numbers should be input through the command line arguments. If the two numbers are integers, the program will multiply them in integer format.

```
C:\Users\86182\Desktop
./mul 2 3
2 * 3 = 6
./mul 232 -231
232 * -231 = -53592
./mul 0 -123
0 * -123 = 0
```

2.2 If the input contains some non-integer numbers, the program will try to interpret the input as floatingpoint numbers.

(it can also identify 0002 wisely)

```
C:\Users\86182\Desktop\sundry workshop\source.
./mul 3.1416 2
3.1416 * 2 = 6.2832
./mul 3.1415 2.0e-2
3.1415 * 2.0e-2 = 0.06283
./mul 2e123211 3.123e-123211
2e123211 * 3.123e-123211 = 6.246
./mul 0002e0000000 00.03e123
0002e0000000 * 00.03e123 = 6e121
```

2.3 It can tell that the input is not a number.

return true **IFF in format (-)123(.)123(e(-)123).**

```
./mul a 2
The input cannot be interpret as numbers!
./mul 12e 2
The input cannot be interpret as numbers!
./mul -11-2e-1 2
The input cannot be interpret as numbers!
./mul -1.e1 2
The input cannot be interpret as numbers!
./mul 1.-1e1 2
The input cannot be interpret as numbers!
./mul 1.1.1e1 2
The input cannot be interpret as numbers!
./mul 1e1.1 2
The input cannot be interpret as numbers!
```

2.4 If you input some big integers, what will happen? Please describe some possible solutions, and try to

implement them.

```
选择 C:\Users\86182\Desktop\sundry workshop\source.exe
./mul 1234567890 1234567890
1234567890 * 1234567890 = 1524157875019052100
./mul 12300000000000 234000000000000000
12300000000000 * 234000000000000000 = 28782e26
```

2.5 If you input some big floating-point numbers, what will happen? Please describe some possible solutions, and try to implement them.

```
C:\Users\86182\Desktop\sundry workshop\so
./mul 1.0e200 1.0e200
1.0e200 * 1.0e200 = 1e400
./mul 2e123211 3.123e-123211
2e123211 * 3.123e-123211 = 6.246
./mul 0002e0000000 00.03e123
0002e0000000 * 00.03e123 = 6e121
```

2.6 Some others which can improve the program. like the highlight of 2.2 and 2.3.

We can enlarge the parameter range in BigNum. Like the size of a[n]. also we can use a array to store the number of e. but in practice is unnecessary.

In the main function. We can check if there is two or more string. If not, print “please input two string!”.