CS205 Project1 Report

11911702 郭一潼
Studentid:11911702

CS205 Project1 Report

1. Introduction

1.1 Project Description

This Project designs a calculator for valid input number(including integer/float in valid decimal system or scientific notation form).

1. The calculator should get input throught command line
2. It can tell that the input is not a number
3. It can still get the right result although the input is big

1.2 Development Environment

- x86_64
  - vscode (version 1.71)
  - WSL (version 2)
  - Ubuntu(22.04)
  - g++(11.2.0)

2.Analysis

2.1 Input Protection

The program should tell whether the input is valid or not.
Eg: **3.145a1, a.3145, 1.0ea , a.0e10** are not valid input.

By contrast, **3.1415, 2.0e10 , 1e10** are valid input.
Overall, there are four kinds of valid input:

- Valid input
  - 1.Each bit of the input is digital number.Eg:114514
  - 2.The bits before and after 'e' are all digital number.Eg:1e10
  - 3.The bits except '.' are all digital number.Eg:3.1415
  - 4.The bits before 'e' fit rule 3 , the bits after 'e' fit rule 1. Eg:2.0e10

2.2 Array Multiply Method

Considering there are big number inputs sometimes. Using '*' operator may cause overflow, so we use **char array** to represent the number.

- Procedure
  - 1.Reverse the input number and store it into char input1 [ ] and char input2 [ ]
  - 2.Using a char res [ ] (res.length=input1.length+input2.length)to store the result
  - 3. For each index i in input1 and for each index j in input2,res[i+j]=input1[i]*input2[j]
  - 4.If res[i+j]>9 then res[i+j+1]+=res[i+j]/10 , res[i+j]=res[i+j]%10.
  - 5.From the higher index of res to lower index of res, if find the first unzero bit, then output all the remaining digital bits.

Eg:Taking 341*61 as example.

2.3 Overall Multiply Method

Considering there exists exponent form integer and float. The exponent **can be added** , so in my program , i use a int variable **exponent_all** to represent the final exponent of the res array and output the result **considering the relationship of exponent_all and the number of un-zero bits of res**.

- Procedure
  - 1.For exponent integer,we can directly get their exponent(after 'e' part)
    Eg:1e10 the exponent is 10.
  - 2.For all numeric float form, we can get their exponent (after '.' part)
    Eg:3.1415=31415*10^-4, the exponent is -4
  - 3. For exponent form float, the exponent is added by the previous part of 'e' and the part after 'e'.
    Eg:3.1415e10 the exponent of 3.1415 is -4 and that of exponent part is 10, overall the exponent of 3.1415e10 is -4+10=6.
  - The overall result should consider the relationship of valid un-zero digital bits of res and exponent.
    Eg:3.1415*1.0e-1 the exponent_all=31415*10*10^(-4-1-1) so res is 314150 and exponent is -6 , so the overall result is 0.31415.

3.Code

3.1 Integer Input Protection

```
//It is used to tell whether there is non digital number in the [start,end]
interval.
bool IntervalAreAllNumerics(char input_num[],int start,int end){
    if(start>end){
        return false;
    }
    bool res=true;
    int temp;
    for(int i=start;i<=end;i++){
       temp=input_num[i];
       if(temp<48||temp>57){
        res=false;
        break;
       }
    }
    return res;
}
bool IsValidIntegerAllNumeric(char input_num[]){
    bool res=true;
    int len=strlen(input_num);
    if(input_num[0]=='-'){
        return IntervalAreAllNumerics(input_num,1,len-1);
    }
    else{
         return IntervalAreAllNumerics(input_num,0,len-1);
    }
}
bool IsValidIntegerExponent(char input_num[]){
    int len=strlen(input_num);
    for(int i=0;i<len;i++){
        if(input_num[i]=='e'){
            if(input_num[0]=='-'){
                if(IntervalAreAllNumerics(input_num,1,i-
1)&&IntervalAreAllNumerics(input_num,i+1,len-1)){
                return true;
              }
              else{
                return false;
              }

            }
            else{
                if(IntervalAreAllNumerics(input_num,0,i-
1)&&IntervalAreAllNumerics(input_num,i+1,len-1)){
                return true;
              }
              else{
                return false;
              }
            }
```

```
        }
    }
    return false;
}
bool IsValidInteger(char input_num[]){
    return IsValidIntegerAllNumeric(input_num)||IsValidIntegerExponent(input_num);
}
```

3.2 Float Input Protection

```
bool IsValidFloatAllNumeric(char input_num[]){
    bool res=true;
    int len=strlen(input_num);
    for(int i=0;i<len;i++){
        if(input_num[i]=='.'){
             if(input_num[0]=='-'){
                return IntervalAreAllNumerics(input_num,1,i-
1)&&IntervalAreAllNumerics(input_num,i+1,len-1);
             }
             else{
                return IntervalAreAllNumerics(input_num,0,i-
1)&&IntervalAreAllNumerics(input_num,i+1,len-1);
             }

        }
    }
   return false;
}
bool IsValidFloatExponent(char input_num[]){
    int len=strlen(input_num);
    for(int i=0;i<len;i++){
        if(input_num[i]=='e'){
              char temp[200]={0};
              strncpy(temp,input_num+0,i);
              if(IsValidFloatAllNumeric(temp)){
                if(input_num[i+1]=='-'){
                   return IntervalAreAllNumerics(input_num,i+2,len-1);
                }
                else{
                    return IntervalAreAllNumerics(input_num,i+1,len-1);
                }
              }
              else{
                return false;
              }
        }
    }
    return false;
}
bool IsValidFloat(char input_num[]){
```

```
    return IsValidFloatAllNumeric(input_num)||IsValidFloatExponent(input_num);
}
```

3.3 Char Array Multiplication

```
//The function below is used to reverse the array for example 12345 should be
reversed to 54321 to calculate the right answer.
void CharArrayReverse(char a[]){
    char temp;
    int index=strlen(a);
    for(int i=0;i<index/2;i++){
            temp=a[i];
            a[i]=a[index-i-1];
            a[index-i-1]=temp;
    }
}
void MultiplyCharArrays(char a[],char b[],char c []){
    CharArrayReverse(a);
    CharArrayReverse(b);
    int len_a=strlen(a);
    int len_b=strlen(b);
    int len_c=strlen(a)+strlen(b);
    for(int i=0;i<len_a;i++){
        if(a[i]=='-'){
            break;
        }
        else{
            for(int j=0;j<len_b;j++){
                if(b[j]=='-'){
                    break;
                }
                c[i+j]+=(a[i]-'0')*(b[j]-'0');
                if(c[i+j]>9){
                    c[i+j+1]+=c[i+j]/10;
                    c[i+j]=c[i+j]%10;
                }
            }
        }
    }
    for(int i=0;i<len_c;i++){
        c[i]+='0';
    }
}
```

3.4 Output Result

```
void ShowResult(char res[],int exponent){
    int len=strlen(res);
    bool whether_first=false;
```

```
    int have_exponent=-1;
    for(int i=len-1;i>=0;i--){
        if(!whether_first&&res[i]!='0'){
            whether_first=true;
            have_exponent=i;
            break;
        }
    }
    if(exponent==0){
        for(int i=have_exponent;i>=0;i--){
            cout<<res[i];
        }
        return;
    }
    if(have_exponent+exponent==0){
        cout<<res[have_exponent]<<".";
        for(int i=have_exponent-1;i>=0;i--){
            cout<<res[i];
        }
    }
    else if(have_exponent+exponent<0){
            cout<<"0.";
            for(int i=have_exponent+exponent+1;i<0;i++){
                cout<<"0";
            }
            for(int i=have_exponent;i>=0;i--){
                cout<<res[i];
            }
    }
    else{
           if(exponent>=0){
            for(int i=have_exponent;i>=0;i--){
                cout<<res[i];
            }
            for(int i=0;i<exponent;i++){
                cout<<"0";
            }
           }
           else{
             for(int i=have_exponent;i>=0;i--){
                cout<<res[i];
                if(i+exponent==0){
                    cout<<".";
                }
             }
           }
    }
}
```

4.Result and Verification

5.Difficulties and Solution

1.The first is use which type to get the input. **to represent the number.**
2.The second is input protection.**Its solution is to test whether the input is among four kinds of valid input above**.
3.The third is to find a general solution to all kinds of data like (integer* integer, integer* float, float*float). **The solution is using an exponent_all variable to get the overall exponent(in 2.3 Overall Multiply Method Part) and output the result according to the relationship of un-zero bits of res and exponent_all.**