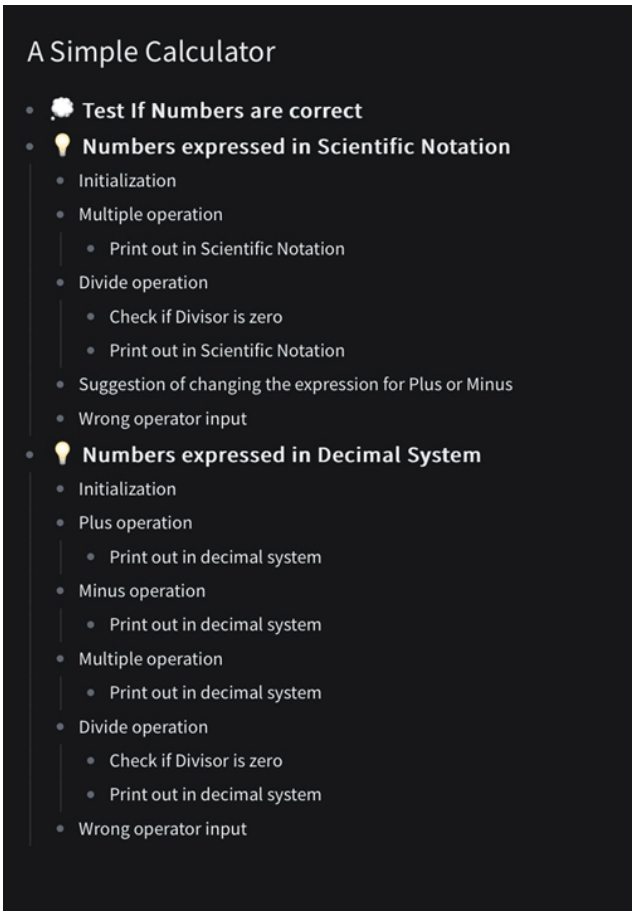
Contents

1 Part 1 - Requirement Analysis

The concise purpose of this project is to implement a calculator which can add, subtract, multiply and divide two numbers.
In order to further satisfy the demand of the user, this project can not only actualize the four arithmetic operations of small numbers but also actualize the four arithmetic operations of large numbers with high precision. Simultaneously, it can not only deal with integers and decimals but also actualize the multiplication and division operations of the numbers in scientific notation.
Moreover, this project can control the precision of its operation during the division (the default number of significant digits is 20), and can also control the number of decimal places output and make it rounded (4 decimal places are reserved by default). The precision of division and the number of decimal places in the output can be modified directly through Marco (Due to the format limit of input requirement, there is no function for the user to input the precision of division or the number of decimal places in the output).

2 Part 2 - Code Implementation

2.1 Overall architecture



2.2 Input the parameters

According to the requirements, we need to input our numbers and the operator through the command line arguments. Thus, we use the main function instead of the scanf function to input. The numbers and the operator will be stored up in a two-dimensional char array as datatype char.

```
1 int main(int argc,char** argv)
```

2.3 Test if the input is appropriate

Test if both the numbers and the operator are correct.

2.3.1 Check the number

For the numbers, check whether the parameter itself is correct, whether the number of each parameter is correct and whether the position of each parameter is correct.
There are four kinds of parameters we can accept: Numbers, Float point, Plus and Minus signs and the character ‘e’ or ‘E’. Moreover, if the operation is a division, the divisor cannot be zero.
The numbers of Plus and Minus signs, decimal points and character ‘e’ or ‘E’ are limited. We check the number of character ‘e’ or ‘E’ first because its occurrence will affect the maximum number of the other two kinds of parameters.

If E or e exit	Plus Minus Sign	Float point
Yes(only 1)	2	2
No	1	1

The positions of the parameters are very strict. Firstly, the position of the Plus and Minus signs should be the front place of the number or the next place after the character ‘e’ or ‘E’. Secondly, the position of the Float

point should be behind the place of the Plus and Minus sign or behind the character ‘e’ or ‘E’. Thirdly, both before and after the character ‘e’ or ‘E’ should exist numbers.

```
int TestIfNum (const char* str)
{
    int sign=1,flag=0;
    if(strspn(str,"0123456789.+-Ee")!=strlen(str))//Check whether there are any undesirable characters
    return sign=0;
    char E_e[2]={'\0'};
    if(strstr(str,"e"))//Check whether it is Scientific notation (flag==0 means yes)
    strcpy(E_e,"e");
    else if(strstr(str,"E"))
    strcpy(E_e,"E");
    else//Detect whether the number entered in decimal is correct
    {
        int k=0,cnt=0;
        while(str[k]!='\0')
        {
            //The presence of plus and minus signs affects the position of the decimal point
            int signOfPM=0;
            if(str[k]=='+'||str[k]=='-')//Check whether the plus/minus sign is correct
            {
                signOfPM=1;
                if(k!=0)//Detection of the position and number of plus and minus signs
                return sign=0;
            }
            if(str[k]=='.')//Check whether the decimal point is correct
            {
                if(signOfPM==0)//Check whether the decimal point position is correct
                {
                    if(k==0)
                    return sign=0;
                }
                else if(signOfPM=1)
                {
                    if(k==1)
                    return sign=0;
                }
                cnt++;
                if(cnt>1)//Check whether the decimal point position is correct
                return sign=0;
            }
            k++;
        }
        flag=1;
    }
    if(flag==0)
    {
        int jmax=strspn(str,"0123456789+-."),cnt=0;
        if(jmax==0)
        return sign=0;
        for(int j=0;j<jmax;j++)//Check whether the string before e or E is correct
        {
            int signOfPM=0;
            if(str[j]=='+'||str[j]=='-')//Check whether the plus/minus sign is correct
            {
                signOfPM=1;
                if(j!=0)//Detection of the position and number of plus and minus signs
                return sign=0;
            }
            if(str[j]=='.')//Check whether the decimal point is correct
            {
                if(signOfPM==0)//Check whether the decimal point position is correct
                {
                    if(j==0)
                    return sign=0;
                }
```

```
                else if(signOfPM=1)
                {
                    if(j==1)
                    return sign=0;
                }
                cnt++;
                if(cnt>1)//Check whether the number is correct
                return sign=0;
            }
        }
        char *strr=(char*)calloc(100,sizeof(char));//Get the string after e or E
        strcpy(strr,strstr(str,E_e));
        if(strlen(strr) > 1)//Check whether there is a number after e or E
        {
            int i=0,cnt=0;
            while(strr[i] != '\0')
            {
                int signOfPM=0;
                if(strr[i]=='+'||strr[i]=='-')//Check whether the plus/minus sign is correct
                {
                    signOfPM=1;
                    if(i!=1)//Detection of the position and number of plus and minus signs
                    return sign=0;
                }
                if(strr[i]=='.')
                {
                    if(signOfPM==0)//Check whether the decimal point position is correct
                    {
                        if(i==0)
                        return sign=0;
                    }
                    else if(signOfPM=1)
                    {
                        if(i==1)
                        return sign=0;
                    }
                    cnt++;
                    if(cnt>1)//Check whether the number of decimal points is correct
                    return sign=0;
                }
                if(i>0 && (strr[i]=='e'||strr[i]=='E'))//Check whether the number of e or E is correct
                return sign=0;
                i++;
            }
        }
        else
        return sign=0;
    }
    return sign;
}
```

2.3.2 Check the operator

There are four kinds of appropriate operators: “+”, “-”, “*” and “/”. The function strcmp is used in the loop of If to check. If no one is matched, then print “The input cannot be interpreted as an operator!”.

2.4 Store the Numbers

There are two types of numbers: Numbers expressed in the decimal system and Numbers expressed in scientific notation. In order to handle the big numbers, the numbers are stored in a structure. As long as we can handle big numbers, we can handle small numbers.

2.4.1 Number expressed in decimal system

We store the information of the number in four sections: Sign, Dot position, Length and the Data (in the right order).

```
typedef struct bignum//Store decimal digits
{
    int Sign;
    int Dot_position;//Relative to the value in Data
    int Length;
    int* Data;
}Bignum;
```

2.4.2 Number expressed in scientific notation

We store the information of the number in three sections: Digit before the character 'e' or 'E', Digit after the character 'e' or 'E' and Sign. (Assuming the digit before and after the character 'e' or 'E' are not too large)

```
typedef struct Scentificnum//Storing numbers from Scientific notation
{
    int Sign;
    double Data1;//Data before E or e
    double Data2;//Data after E or e
}SNnum;
```

2.5 Carry out the four operations

The way to carry out the four operations is depending on the type of number.

2.5.1 Add operation for the number expressed in decimal system

Assuming two non-negative numbers adding.
Firstly, align the decimals of two numbers and store them in num1 and num2 in reverse order and create the memory to store the result. Secondly, starting from the first place, we add the two inverse forms of the numbers, getting the temporary number. The digit of the place should be the reminder of the ten for the temporary number and the digit of the next place should add the quotient of the temporary number divided by ten. Finally, if the length of the result is bigger than the memory, create a larger memory to store the data in the right order.

```
Bignum BigNumAdd(Bignum* pnum1,Bignum* pnum2)//By default, two positive numbers are added
{
    Bignum result;
    int *num1,*num2,*tempp,*sum;
    int Len=0;
    /*Align the decimals of the two numbers, store them in reverse order in num1 and num2,
      create sum to store the results, and initially update the position of the decimal point
      of the results*/
    if((pnum1->Length-pnum1->Dot_position)>=(pnum2->Length-pnum2->Dot_position))
    {
        num1=(int*)calloc(pnum1->Length,sizeof(int));
        num2=(int*)calloc(pnum2->Length+(pnum1->Length-pnum1->Dot_position)-(pnum2->Length
        -pnum2->Dot_position),sizeof(int));
        for(int i=pnum1->Length-1,j=0;i>=0;i--,j++)
        num1[j]=pnum1->Data[i];
        for(int i=pnum2->Length-1,j=(pnum1->Length-pnum1->Dot_position)-(pnum2->Length
        -pnum2->Dot_position);i>=0;i--,j++)
        num2[j]=pnum2->Data[i];
        if(pnum1->Dot_position>=pnum2->Dot_position)
        {
            Len=pnum1->Length;
            tempp=(int*)calloc(Len,sizeof(int));
            for(int i=0;i<pnum2->Length+(pnum1->Length-pnum1->Dot_position)-(pnum2->Length
            -pnum2->Dot_position);i++)
```

```
            tempp[i]=num2[i];
            num2=tempp;
            result.Dot_position=pnum1->Dot_position;
        }
        else
        {
            Len=pnum2->Length+(pnum1->Length-pnum1->Dot_position)-(pnum2->Length
            -pnum2->Dot_position);
            tempp=(int*)calloc(Len,sizeof(int));
            for(int i=0;i<pnum1->Length;i++)
            tempp[i]=num1[i];
            num1=tempp;
            result.Dot_position=pnum2->Dot_position;
        }
        sum=(int*)calloc(Len,sizeof(int));
    }
    else
    {
        num1=(int*)calloc(pnum1->Length+(pnum2->Length-pnum2->Dot_position)-(pnum1->Length
        -pnum1->Dot_position),sizeof(int));
        num2=(int*)calloc(pnum2->Length,sizeof(int));
        for(int i=pnum1->Length-1,j=(pnum2->Length-pnum2->Dot_position)-(pnum1->Length
        -pnum1->Dot_position);i>=0;i--,j++)
        num1[j]=pnum1->Data[i];
        for(int i=pnum2->Length-1,j=0;i>=0;i--,j++)
        num2[j]=pnum2->Data[i];
        if(pnum1->Dot_position>=pnum2->Dot_position)
        {
            Len=pnum1->Length+(pnum2->Length-pnum2->Dot_position)-(pnum1->Length
            -pnum1->Dot_position);
            tempp=(int*)calloc(Len,sizeof(int));
            for(int i=0;i<pnum2->Length;i++)
            tempp[i]=num2[i];
            num2=tempp;
            result.Dot_position=pnum1->Dot_position;
        }
        else
        {
            Len=pnum2->Length;
            tempp=(int*)calloc(Len,sizeof(int));
            for(int i=0;i<pnum1->Length+(pnum2->Length-pnum2->Dot_position)-(pnum1->Length
            -pnum1->Dot_position);i++)
            tempp[i]=num1[i];
            num1=tempp;
            result.Dot_position=pnum2->Dot_position;
        }
        sum=(int*)calloc(Len,sizeof(int));
    }
    int temp=0,carry=0;
    for(int i=0;i<Len;i++)//Add two numbers and complete carry
    {
        temp=num1[i]+num2[i]+carry;
        sum[i]=temp%10;
        carry=temp/10;
    }
    int* data;
    if(carry>0)//If the number of results increases, store the results in a larger space
    {
        result.Length=Len+1;//Update result data length
        result.Dot_position++;//Update the position of the decimal point of the result
        data=(int*)calloc(Len+1,sizeof(int));//Store data in positive order
        data[0]=carry;
        for(int i=1,j=Len-1;j>=0;i++,j--)
        data[i]=sum[j];
    }
    else
    {
```

```
        result.Length=Len;//Update result data length
        data=(int*)calloc(Len,sizeof(int));//Update the position of the decimal point of the result
        for(int i=0,j=Len-1;j>=0;i++,j--)//Store data in positive order
        data[i]=sum[j];
    }
    result.Data=data;
    result.Sign=1;
    return result;
}
```

2.5.2 Subtract operation for the number expressed in decimal system

Assuming two non-negative numbers subtracting. Firstly, recognize which of the number is larger and assign the sign to the result. Secondly, align the two numbers with decimals, store the large numbers in reverse order in minuend, and the small numbers in reverse order in subtrahend, so as to subtract the small numbers from the large numbers. Thirdly, starting from the first place, we subtract the subtrahend from the If the minuend, getting the temporary number. If the temporary number is negative, the digit of this place should be the temporary number adding ten and the digit of the next place should subtract one from the original number. The data of the result is stored in the array minuend. Finally, create new memory to store the data in the right order which deletes the zeros at both ends.

```
Bignum BigNumSubtract(Bignum* pnum1,Bignum* pnum2)//Subtract two positive numbers by default
{
    Bignum result;
    Bignum *big,*small;//Distinguish the size of two numbers and judge the result symbol
    if(pnum1->Dot_position>pnum2->Dot_position)
    {
        result.Sign=1;
        big=pnum1;
        small=pnum2;
    }
    else if(pnum1->Dot_position<pnum2->Dot_position)
    {
        result.Sign=-1;
        big=pnum2;
        small=pnum1;
    }
    else
    {
        int sign=0,i=0;
        while(pnum1->Data[i]!='\0'||pnum2->Data[i]!='\0')
        {
            if(pnum1->Data[i]>pnum2->Data[i])
            {
                result.Sign=-1;
                big=pnum1;
                small=pnum2;
                sign=1;
                break;
            }
            else if(pnum1->Data[i]<pnum2->Data[i])
            {
                result.Sign=-1;
                big=pnum2;
                small=pnum1;
                sign=1;
                break;
            }
            i++;
        }
        if(sign==0)
        {
            if(pnum1->Length>pnum2->Length)
            {
                result.Sign=1;
                big=pnum1;
```

```
                small=pnum2;
            }
            else if(pnum1->Length<pnum2->Length)
            {
                result.Sign=-1;
                big=pnum2;
                small=pnum1;
            }
            else
            {
                int num[1]={0};
                result.Data=num;
                result.Sign=1;
                result.Length=1;
                return result;
            }
        }
    }
    /*Align the two numbers with decimals, store the large numbers in reverse order in minuend
    and the small numbers in reverse order in subtrahend, so as to subtract the small numbers
    from the large numbers*/
    int *minuend,*subtrahend;
    int small_len=0,temp=0,borrow=0,big_len=0;
    if((big->Length-big->Dot_position)<(small->Length-small->Dot_position))
    {
        minuend=(int*)calloc((small->Length-small->Dot_position)+big->Dot_position,sizeof(int));
        for(int i=(small->Length-small->Dot_position)-(big->Length-big->Dot_position),j=big->Length-1;
        j>=0;j--,i++)
        minuend[i]=big->Data[j];
        subtrahend=(int*)calloc(small->Length,sizeof(int));
        for(int i=0,j=small->Length-1;j>=0;j--,i++)
        subtrahend[i]=small->Data[j];
        small_len=small->Length;
        big_len=(small->Length-small->Dot_position)+big->Dot_position;
    }
    else
    {
        minuend=(int*)calloc(big->Length,sizeof(int));
        for(int i=0,j=big->Length-1;j>=0;j--,i++)
        minuend[i]=big->Data[j];
        subtrahend=(int*)calloc((big->Length-big->Dot_position)+small->Dot_position,sizeof(int));
        for(int i=(big->Length-big->Dot_position)-(small->Length-small->Dot_position),j=small->Length-1;
        j>=0;j--,i++)
        subtrahend[i]=small->Data[j];
        small_len=(big->Length-big->Dot_position)+small->Dot_position;
        big_len=big->Length;
    }
    //Subtract decimals from large numbers, complete the borrowing, and store the results in minuend
    for(int i=0;i<small_len;i++)
    {
        temp=minuend[i]-subtrahend[i]-borrow;
        if(temp<0)
        {
            minuend[i]=temp+10;
            borrow=1;
        }
        else
        {
            minuend[i]=temp;
            borrow=0;
        }
    }
    if(borrow==1)
    minuend[small_len]-=borrow;
    //Optimize the result output and delete the redundant zero
    int findex=0,bindex=big_len-1,Length=0,dot_position=0;//Find the index in reverse order
    while(minuend[findex]==0)
```

```
    findex++;
    while(minuend[bindex]==0)
    bindex--;
    //Derive the positive order index from the reverse order index
    int Findex=big_len-1-bindex,Bindex=big_len-1-findex;
    if(Findex>=big->Dot_position)//Determine the position of the decimal point
    {
        dot_position=1;
        bindex++;
    }
    else
    dot_position=big->Dot_position-Findex;
    Length=bindex-findex+1;//Length of confirmation result
    int* data=(int*)calloc(Length,sizeof(int));//Store the optimized data in positive order
    for(int i=bindex,j=0;i>=findex;i--,j++)
    data[j]=minuend[i];
    result.Data=data;
    result.Length=Length;
    result.Dot_position=dot_position;
    return result;
}
```

2.5.3 Multiple operation for the number expressed in decimal system

Firstly, create two new memory to store the data in the inverse form and create a memory for the result with the length of the sum of the length of the two data. Secondly, multiply each digit of the first number with each digit of the second number and store each result in the place of the sum of the place of the two digits. Thirdly, starting from the first place of the result, the digit in each place equals the reminder of ten for the original digit and the digit in the next place adds the integer quotient of the original digit divided by ten. Fourthly, if there are zeros on the last few places of the result, create a new memory to store the data without the zeros on the last few places. Finally, update the Dot position, and length sign of the result.

```
Bignum BigNumMultiple(Bignum* pnum1,Bignum* pnum2)
{
    Bignum result;
    int* num1=(int*)calloc(pnum1->Length,sizeof(int));//Store data in reverse order
    int* num2=(int*)calloc(pnum2->Length,sizeof(int));
    for(int i=0,j=pnum1->Length-1;i<pnum1->Length;i++,j--)
    num1[i]=pnum1->Data[j];
    for(int i=0,j=pnum2->Length-1;i<pnum2->Length;i++,j--)
    num2[i]=pnum2->Data[j];
    int* data=(int*)calloc(pnum1->Length+pnum2->Length,sizeof(int));//Create result array
    for(int i=0;i<pnum1->Length;i++)//Multiply
    {
        for(int j=0;j<pnum2->Length;j++)
        data[i+j]+=num1[i]*num2[j];
    }
    for(int i=0;i<pnum1->Length+pnum2->Length-1;i++)//Carry forward
    {
        if(data[i]>9)
        data[i+1]+=data[i]/10;
        data[i] = data[i] % 10;
    }
    int index=0;//Optimize the result output and delete the redundant zero
    for(index=pnum1->Length+pnum2->Length-1;index>=0;index--)
    {
        if(data[index]==0)
        continue;
        else
        break;
    }
    int* final_data=(int*)calloc(index+1,sizeof(int));//Store the optimized results in positive order
    for(int i=0,j=index;i<=index;i++,j--)
    final_data[i]=data[j];
    result.Data=final_data;
    result.Sign=pnum1->Sign*pnum2->Sign;//Update symbols of data
```

```
    result.Length=index+1;//Update data length
    //Update decimal point position
    result.Dot_position=index+1-(pnum1->Length-pnum1->Dot_position)-(pnum2->Length-pnum2->Dot_position);
    return result;
}
```

2.5.4 Divide operation for the number expressed in decimal system

The length of the significant number of the result is twenty.
Firstly, create a new memory named divisor to store the data of the second number in the right order. Create a new memory named dividend with the length of the sum of divide precision minus one and the length of the first number to store all the digits of the first number in the right order and fill the extra place with zeroes. Create a memory for the result with the length as same as the length of the dividend. Secondly, constantly subtract the divisor from the dividend until the dividend is less than the divisor and store the times into the result. Thirdly, if there are zeros on the first few places of the result, create a new memory to store the digit without the zeros on the first few places. Finally, update the Sign, Dot position, Length and Data of the result.

```
Bignum BigNumDivide(Bignum* pnum1,Bignum* pnum2)
{
    Bignum result;
    int* divisor=(int*)calloc(pnum2->Length,sizeof(int));//Default divisor length is less than 20
    for(int i=0;i<pnum2->Length;i++)
    divisor[i]=pnum2->Data[i];
    //Store data of divisor and dividend in positive order
    int* dividend=(int*)calloc(divideprecision-1+pnum1->Length,sizeof(int));
    for(int i=0;i<pnum1->Length;i++)
    dividend[i]=pnum1->Data[i];
    int* data=(int*)calloc(divideprecision-1+pnum1->Length,sizeof(int));
    int cnt=0,borrow=0,left=0,sign=0,k=0,t=0,w=0,i=0,j=0;
    //Convert division into subtraction for operation
    for(i=pnum2->Length-1,w=0;i<divideprecision-1+pnum1->Length;i++,w++)
    {
        cnt=0,sign=0;
        while(1)
        {
            if(left==0)
            {
                for(k=w,j=0;k<pnum2->Length+w;k++,j++)
                {
                    if(dividend[k]>divisor[j])
                    break;
                    else if(dividend[k]<divisor[j])
                    {
                        left=dividend[w];
                        sign=1;
                        break;
                    }
                }
            }
            if(sign==1)
            break;
            borrow=0;
            for(j=pnum2->Length-1,t=pnum2->Length-1+w;j>=0;j--,t--)
            {
                int temp=dividend[t]-divisor[j]-borrow;
                if(temp<0)
                {
                    dividend[t]=temp+10;
                    borrow=1;
                }
                else
                {
                    dividend[t]=temp;
                    borrow=0;
                }
```

```
            }
            cnt++;
            left-=borrow;
        }
        data[i]=cnt;
    }
    result.Dot_position=pnum1->Dot_position+pnum2->Length-pnum2->Dot_position;
    int index=0;
    for(index=0;index<result.Dot_position;index++)
    {
        if(data[index]!=0)
        break;
    }
    if(index==result.Dot_position)
    index--;
    int* final_data=(int*)calloc(divideprecision,sizeof(int));
    for(i=index,j=0;i<index+divideprecision;i++,j++)
    final_data[j]=data[i];
    result.Data=final_data;
    result.Length=divideprecision;
    result.Sign=pnum1->Sign*pnum2->Sign;
    result.Dot_position-=index;
    return result;
}
```

2.5.5 Multiple operation for the number expressed in scientific notation

Firstly, we multiply the digits of the two numbers stored in data1. Secondly, we add the digits of the two numbers stored in data2. Finally, we update data1, data2 and the sign of the result.

```
SNnum SNnumMultiple(SNnum* pnum1,SNnum* pnum2)
{
    SNnum result;
    result.Sign=pnum1->Sign*pnum2->Sign;
    result.Data1=pnum1->Data1*pnum2->Data1;
    result.Data2=pnum1->Data2+pnum2->Data2;
    while(result.Data1>=10)
    {
        result.Data1/=10;
        result.Data2++;
    }
    return result;
}
```

2.5.6 Divide operation for the number expressed in scientific notation

Firstly, we divide the digits of the second number stored in data1 from those stored in the first number. Secondly, we subtract the digits of the second number stored in data2 from those stored in the first number. Finally, we update data1, data2 and the sign of the result.

```
SNnum SNnumDivide(SNnum* pnum1,SNnum* pnum2)
{
    SNnum result;
    result.Sign=pnum1->Sign*pnum2->Sign;
    result.Data1=pnum1->Data1/pnum2->Data1;
    result.Data2=pnum1->Data2-pnum2->Data2;
    while(result.Data1<1)
    {
        result.Data1*=10;
        result.Data2--;
    }
    return result;
}
```

2.6 Output the result

2.6.1 Number expressed in decimal system

```
void Printout(Bignum result)
{
    if(result.Sign==-1)//Symbol of output result
    printf("-");
    /*The position of the last non-zero number is counted because the integer is output in decimal
    form due to the precision in large number division*/
    int index=result.Length-1;
    while(result.Data[index]==0&&index>=result.Dot_position)
    index--;
    if(index>=result.Dot_position)//Judge if it is a decimal
    {
        int *data=(int*)calloc(result.Dot_position+outputprecision,sizeof(int)),*dataa=NULL;
        int i=0;
        while(i<result.Length&&i<result.Dot_position+outputprecision)//Store data in positive order
        {
            data[i]=result.Data[i];
            i++;
        }
        //When the number after the decimal point of the result is greater than the output digit of
        the decimal point
        if(result.Dot_position+outputprecision<result.Length)
        {
            if(result.Data[result.Dot_position+outputprecision]>=5)//Rounding
            {
                data[result.Dot_position+outputprecision-1]++;
                int carry=0,temp=0;
                for(i=result.Dot_position+outputprecision-1;i>=0;i--)//Carry forward
                {
                    temp=data[i]+carry;
                    data[i]=temp%10;
                    carry=temp/10;
                }
                if(carry!=0)//If the number of results increases, store the results in a larger space
                {
                    dataa=(int*)calloc(result.Dot_position+outputprecision+1,sizeof(int));
                    dataa[0]=carry;
                    for(int i=1;i<result.Dot_position+outputprecision+1;i++)
                    dataa[i]=data[i-1];
                    result.Dot_position++;
                    data=dataa;
                }
            }
        }
        for(i=0;i<result.Dot_position;i++)//Output Decimal
        printf("%d",data[i]);
        printf(".");
        for(i=result.Dot_position;i<result.Dot_position+outputprecision;i++)
        printf("%d",data[i]);
    }
    else
    {
        for(int i=0;i<result.Dot_position;i++)//Output integer
        printf("%d",result.Data[i]);
    }
}
```

2.6.2 For the number expressed in the scientific notation

```
void SNprintout(SNnum result)
{
    if(result.Data1==0)//Determine whether the number is zero
```

```
    printf("0");//If it is zero, no output e or E is required
    else//Output base
    {
        printf("%.2lf",result.Data1);
        if(result.Data2!=0)//Output index
        printf("e%.2lf",result.Data2);
    }
}
```

3 Part 3 - Result & Verification

3.1 Intelligence

It can tell the reason when the input is not a number

Figure 1: Wrong parameter.

Figure 2: Wrong number of the Sign.

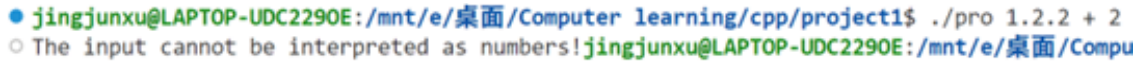


Figure 3: Wrong number of dot.

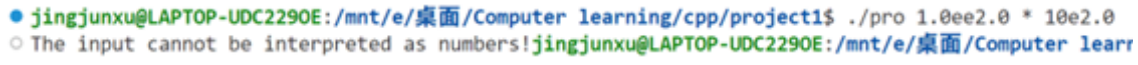


Figure 4: Wrong number of E(e).

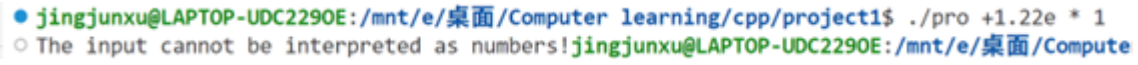


Figure 5: Nothing is behind the E(e).

Figure 6: Wrong position of the sign and dot.

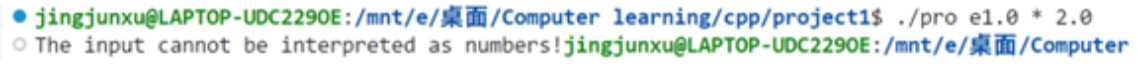


Figure 7: Wrong position of E(e).

It can tell the reason when the input is not an operator

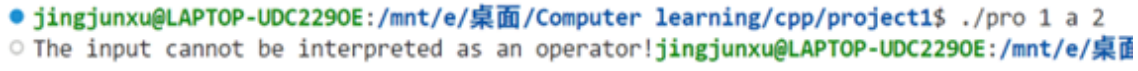


Figure 8: Wrong number of the Sign.

It can tell the reason why the operation cannot be carried out.

```
jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$ ./pro 2 / 0
A number cannot be divied by zero.jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer le
```

Figure 9: In decimal system.

```
jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$ ./pro 1.0e2.0 / 0e1.0
A number cannot be divied by zero.jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/
```

Figure 10: In scientific notation.

```
#define divideprecision 20
#define outputprecision 4
```

It can round the input number

```
jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$ ./pro 5 / 3
5 / 3 = 1.6667jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$
```

Fig.5 can recommend changing the way of expressing the number in order to complete the operations

```
jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$ ./pro 1.0e200 + 1.0e200
Recommend to enter numbers in decimal to compelete the operationjingjunxu@LAPTOP-UDC229OE:/mnt/e
```

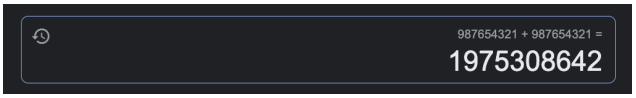
3.2 Correctness

It can do four operations between the small integer numbers

```
jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$ ./pro 2 + 3
2 + 3 = 5jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$ ./pro 2 - 3
2 - 3 = -1jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$ ./pro 2 * 3
2 * 3 = 6jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$ ./pro 2 / 3
2 / 3 = 0.6667jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$
```

It can do four operations between the large integer numbers

```
jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$ ./pro 987654321 + 987654321
987654321 + 987654321 = 1975308642jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$
jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$ ./pro 987654321 - 555555
987654321 - 555555 = 987098766jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$
jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$ ./pro 987654321 * 987654321
987654321 * 987654321 = 975461057789971041jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/projec
t1$
jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$ ./pro 987654321 / 55555
987654321 / 55555 = 17777.9556jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$
```



It can do four operations between the small float number

```
jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$ ./pro 1.1 + 1.2
1.1 + 1.2 = 2.3000jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$ ./pro 1.1 - 1.2
1.1 - 1.2 = -0.1000jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$ ./pro 1.1 * 1.2
1.1 * 1.2 = 1.3200jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$ ./pro 1.1 / 1.2
1.1 / 1.2 = 0.9167jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$
```

It can do four operations between the large float number

```
jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$ ./pro 987654.321 + 555.324
987654.321 + 555.324 = 988209.6450jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$
jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$ ./pro 987654.321 - 555.324
987654.321 - 555.324 = 987098.9970jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$
jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$ ./pro 987654.321 * 555.324
987654.321 * 555.324 = 548468148.1550jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$
jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$ ./pro 987654.321 / 555.324
987654.321 / 555.324 = 1778.5191jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$
```

It can implement multiplication and division when the numbers are expressed in scientific notation.

```
jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$ ./pro 1.0e200 * 1.0e200
1.0e200 * 1.0e200 = 1.00e400.00jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$
jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$ ./pro 6.0e200 / 3.0e100
6.0e200 * 3.0e100 = 2.00e100.00jingjunxu@LAPTOP-UDC229OE:/mnt/e/桌面/Computer learning/cpp/project1$
```

4 Part 4 - Difficulties & Solutions

4.1 Difficulty in actualizing the operations between large numbers

When the number is too large, even if a long double is used to store data, it may sometimes cause an overflow, so this project chooses to use an array to store large data to avoid the problem of overflowing.
When doing the four operations, this project simulates the way humans do the four operations.

4.2 Difficulty in dealing with the numbers expressed in scientific notation

Dealing with the add and subtract operations of the numbers expressed in scientific notation is relatively complicated and we usually express numbers in scientific notation in order to simplify our ways of multiplying and dividing. Thus, this project only implements the functions of multiplication and division in scientific notation. However, we can choose to input the numbers in a decimal system to accomplish the operations of adding and subtracting.