Project1 report

前言

在这第一次的project中，我作为非CS专业0基础入门C/C++的新手，首先根据网上别人开源的代码进行学习和改编，完成了project要求的计算器的任务，并且添加了一些额外的功能。对我自己而言，这次project学习到了很多知识，也了解了C的一些独特的不同于JAVA的用法（比如指针、控制内存等）。虽然写代码和debug花费了大量的时间和精力，但project做完的时候还是成就感满满。

参考代码

由于本身0基础，我首先在网上找到了一个开源的计算器的半成品，并在读懂该大数计算器的基本思路之后，在其之上加入自己的内容，使之完整。

以下是该博客的完整代码（C语言实现大数计算器_大整数计算器c语言_oxuzhenyi的博客-CSDN博客）

```
/*
* file name : calculatorv4.c
* desp : calculator version 4
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define BASE (10)
#define MAX(x,y) ((x)>(y)?(x):(y))

// 大数数据结构
typedef struct bignumber_s{
    char sign; // 代表大数的符号，1为负数，0为正数
    int len;   // 代表大数的位数
    char data[]; // 大数的数据内容，data[0]代表个位，data[1]代表十位，data[2]代表百位....
} bignumber_s;

bignumber_s *calc_add(bignumber_s *a, bignumber_s *b);
bignumber_s *calc_sub(bignumber_s *a, bignumber_s *b);

// 构造大数模板，len指向大数的数据位数，sign表示该大数的符号
bignumber_s *make_bignumber_temp(int len, int sign)
{
    // 分配bignumber_s及其所代表数据 所需的内存
    bignumber_s *temp = malloc(sizeof(bignumber_s)+len);
    if (NULL == temp) {
        perror("Malloc");
        exit(-1);
    }
    temp->sign = sign;
    temp->len  = len;
    memset(temp->data, 0, len);
    return temp;
}

// 处理数字字符串冗余的0，即"00123" -> "123"
const char *strip_str(const char *str)
{
    int i = 0;
        int len = strlen(str);
        for (i=0; i<len-1&&str[i]=='0'; i++) ;
        return str+i;
}
// 将字符串数据填充进模板中
```

```
void fill_data_fromstr(bignumber_s *n, const char *str)
{
    int i = 0;
    int len = n->len;
    for (i=0; i<len; i++) {
        int d = str[len-1-i]-'0';
        if (d>=0 && d<=9)
            n->data[i] = d;
        else {
          fprintf(stderr, "Invalid Number:%s\n", str);
          exit(-1);
        }
    }
}
// 从字符串构造一个大数
bignumber_s *make_bignumber_fromstr(const char *str)
{
    // 处理负数字符串，即"-123"
    int sign = 0;
    if (str[0]=='-') {
        sign = 1;
        str++;
    }
    // 处理数字字符串冗余的0，即"00123" -> "123"
    const char * striped_str = strip_str(str);

    int len = strlen(striped_str);
    // 指定数据位长度即符号，创建一个大数模板
    bignumber_s *temp = make_bignumber_temp(len ,sign);
    // 将字符串数据填充进模板中，完成大数创建
    fill_data_fromstr(temp, striped_str);
    return temp;
}

// 以字符串的形式打印输出一个大数
void print_bignumber(bignumber_s* b)
{
   int len = b->len;
   char *str = malloc(len+1);
   int i = 0;

   for (i=0; i<len; i++) {
      str[i] = b->data[len-i-1]+'0';
   }
   str[len] = '\0';

   fprintf(stdout,"%s%s\n", b->sign==1?"-":"", strip_str(str));
   free(str);
}

void usage(const char *s)
{
    fprintf(stderr, "Usage:%s number1 +-x/ number2.\n",s);
    exit(-1);
}

// 实现无符号加法，a和b为加数，r为和
void add_impl(bignumber_s *a, bignumber_s *b, bignumber_s *r)
{
    int i = 0;
    char carry = 0;
    int len = r->len;
    for (i=0; i<len; i++) {
       if (i<a->len)carry += a->data[i];
       if (i<b->len)carry += b->data[i];
       r->data[i] = carry%BASE;
       carry /= BASE;
    }
}
```

```
bignumber_s *calc_add(bignumber_s *a, bignumber_s *b)
{
    // 情况一和情况四
    if (a->sign==b->sign) {
        // n位数+m位数，其和最大为max(n,m)+1位数
        int len = MAX(a->len, b->len) + 1;
        bignumber_s *result = make_bignumber_temp(len,a->sign);

        add_impl(a,b,result);
        return result;
    } else if (a->sign == 0 && b->sign == 1) {//情况二
        b->sign = 0; //b数去绝对值
        return calc_sub(a,b);
    } else if (a->sign == 1 && b->sign == 0) {//情况三
        a->sign = 0; //a数去绝对值
        return calc_sub(b,a);
    }
}

// 实现无符号减法，a-b , r为差
void sub_impl(bignumber_s *a, bignumber_s *b, bignumber_s *r)
{
        int i=0;
        int borrow = 0;
        int len = r->len;
        int temp = 0;
        for (i=0; i<len; i++) {
            temp = a->data[i]+BASE-borrow-((i<b->len)?b->data[i]:0);
            r->data[i] = temp%BASE;
            borrow = temp/BASE?0:1;
        }
}

int valid_len(bignumber_s *a)
{
    int len = a->len;
    int i = len-1;
    for (i=len-1; i>=0; i--) {
        if (a->data[i]==0) len--;
        else break;
    }
    return len;
}
// 判断两个大数的大小
// a > b 返回1 , a<b 返回 -1 , a==b 返回0
int cmp(bignumber_s *a, bignumber_s *b)
{

        if (a->sign==0&&b->sign==1) return 1;
        if (a->sign==1&&b->sign==0) return -1;

        int sign = a->sign;
        int alen = valid_len(a);
        int blen = valid_len(b);
        if (alen>blen) return (sign==1?-1:1);
        else if (alen<blen) return (sign==1?1:-1);
        else {
                int i = 0;
                int len = alen;
                for (i=len-1; i>=0; i--) {
                 if (a->data[i]>b->data[i])return (sign==1?-1:1);
                 else if (a->data[i]<b->data[i]) return (sign==1?1:-1);
            }
            return 0;
        }
}

bignumber_s *calc_sub(bignumber_s *a, bignumber_s *b)
```

```
{
    // 情况一
    if(a->sign==0 && b->sign==0) {
        if (cmp(a,b)>=0) { // 被减数大于等于减数，即差大于等于0时
                int len = a->len;
                bignumber_s *result = make_bignumber_temp(len,0);
                sub_impl(a,b,result);
                return result;
        } else { // 被减数小于减数，即差小于0时
                int len = b->len;
                bignumber_s *result = make_bignumber_temp(len,1);
                sub_impl(b,a,result);
                return result;
        }
    } else if (a->sign==1 && b->sign==1) { //情况四
        b->sign=0;
        a->sign=0;
        return calc_sub(b,a); //调换减数和被减数，变成情况一
    }else if (a->sign==0 && b->sign==1) { // 情况二
        b->sign=0;
        bignumber_s *result = calc_add(a,b);
        result->sign = 0;
        return result;
    } else if (a->sign==1 && b->sign==0) { // 情况三
        a->sign=0;
        bignumber_s *result = calc_add(a,b);
        result->sign = 1;
        return result;
    }
}

// 实现无符号乘法，x * y , z为积
void mul_impl(bignumber_s *x, bignumber_s *y, bignumber_s *z)
{
    int n = x->len;
    int m = y->len;
    int i = 0;
    int j = 0;
    int carry = 0;
    for (i=0; i<m; i++) {
        // y的每一位乘以x，即计算y[i] * x[0...n] 并累加到z[i...i+n]中
        for (j=0; j<n; j++) {
            carry += y->data[i]*x->data[j] + z->data[i+j];
            z->data[i+j] = carry%BASE;
            carry /= BASE;
        }

        // 将剩余的进位,继续在z[i+n...n+m]中累加，从而完成y[i]乘以x[0...n]其积累加到z[i...m+n]中
        for (; j+i<n+m; j++) {
            carry += z->data[i+j];
            z->data[i+j] = carry % BASE;
            carry /= BASE;
        }
    }
}

bignumber_s *calc_mul(bignumber_s *a, bignumber_s *b)
{
    int len = a->len + b->len;
    bignumber_s *result = make_bignumber_temp(len,a->sign==b->sign?0:1); //a->sign==b->sign为情况一，否则为情况二。
    mul_impl(a,b,result);
    return result;
}

// 大数加一操作
void plusone(bignumber_s *a)
{
   int len = a->len ;
   int i;
```

```
   int carry = 1;
   for (i=0; i<len; i++) {
      carry += a->data[i];
      a->data[i] = carry%BASE;
      carry /= BASE;
   }
}
// 大数除法实现
bignumber_s *calc_div(bignumber_s *a, bignumber_s *b)
{
    bignumber_s *zero = make_bignumber_temp(1,0);
    if (cmp(b,zero)==0) {// 除数为0 报错
        fprintf(stderr,"Integer division by zero\n");
        exit(-1);
    }else if (cmp(a,zero)==0) { // 被除数为0 ，结果为0
        return zero;
    }

    int len = a->len;
    bignumber_s *result = make_bignumber_temp(len,a->sign==b->sign?0:1);
    a->sign = 0;
    b->sign = 0;
    bignumber_s *temp = make_bignumber_temp(len, 0);
    bignumber_s *aa = a;
    while(1) {
         if (cmp(aa, b)>=0) {
             sub_impl(aa, b, temp);
             plusone(result);
             aa = temp;
         } else {
             free(temp);
             return result;
         }
    }
}

int main(int argc, char *argv[])
{
    bignumber_s *a = make_bignumber_fromstr(argv[1]);
    bignumber_s *b = make_bignumber_fromstr(argv[3]);
    if (argc!=4) usage(argv[0]);

    if (0 == strcmp(argv[2],"+"))
        print_bignumber(calc_add(a,b));
    else if (0 == strcmp(argv[2],"-"))
        print_bignumber(calc_sub(a,b));
    else if (0 == strcmp(argv[2],"x"))
        print_bignumber(calc_mul(a,b));
    else if (0 == strcmp(argv[2],"/"))
        print_bignumber(calc_div(a,b));
    else usage(argv[0]);
    return 0;
}
```

以上参考代码仅仅实现了+-x/的整数四则运算，我在其之上添加了e指数和小数的内容。

代码解释

大数构建

```
typedef struct bignumber_s{
    char sign; // 代表大数的符号，1为负数，0为正数
```

```
    int len;   // 代表大数的位数
    int point;  // 代表小数点位于第几位
    int sci;    //代表科学计数法后面0的数量
    char data[]; // 大数的数据内容，data[0]代表个位，data[1]代表十位，data[2]代表百位....
} bignumber_s;
```

在本项目中，需要完成大数的计算。由于数字本身的长度受内存限制，需要把一长串数字以字符串的形式保存，并保留其中e指数和小数点的相关信息。于是，我定义了新的结构体bignumber_s，来保存数字符号，位数，小数点位数，和是否启用科学计数法，以及数字内容。

```
int main(int argc, char *argv[])
{
    int judge = 0;//判断加减乘除，加1减2乘3除4
    if(0 == strcmp(argv[2],"+\0"))  judge = 1;
    if(0 == strcmp(argv[2],"-\0"))  judge = 2;
    else if (0 == strcmp(argv[2],"x\0")) judge = 3;
    else if (0 == strcmp(argv[2],"/\0"))  judge = 4;
    bignumber_s *b = make_bignumber_fromstr(argv[3]);
    bignumber_s *a = make_bignumber_fromstr(argv[1]);
    if (argc!=4) usage(argv[0]);  //判断输入不是四个字符，报错
    if (judge == 1)  print_bignumber(calc_add(a,b));
    else if (judge == 2)  print_bignumber(calc_sub(a,b));
    else if (judge == 3)  print_bignumber(calc_mul(a,b));
    else if (judge == 4)  print_bignumber(calc_div(a,b));
    else usage(argv[0]);  //+-x/以外的情况，报错
    return 0;
}
```

开启一次计算，我们要先从命令行得到输入的两个数字和操作符号，读取argv[2]为操作符号，并判断输入是否异常。这里不直接用argv[2]而是用judge来判断的原因是：后续的处理可能会在内存里改变argv[2]的数值，因此先赋值给judge防止改变。

```
bignumber_s *make_bignumber_fromstr(const char *str)
{
  char *strnd = malloc(strlen(str)+1);  //为strnd创建一块地址，并把str复制给他
  strcpy(strnd, str);
```

读取字符串，并重新开启一块内存存放该字符串

```
int sign = 0;
    if (strnd[0]=='-') {  //判断有无负号
        sign = 1;
        strnd++;
    }
```

判断输入是否有负号

```
//找到字符串里e指数的位置
    int sci_pos = 0;  //表示e的位置
    int multi_sci = 0;
    for(int j = currentLen; j > 0; j--){
        if(strnd[j-1] == 'e' && multi_sci == 0){
            sci_pos = currentLen - j;
            multi_sci = 1;
        }
        else if(strnd[j-1] == 'e' && multi_sci == 1){
            fprintf(stderr,"you have input multiple e, please check your input number!");
            exit(-1);
        }
    }
```

查看字符串里是否有e指数，并确认e指数最多只有一个（如果有两个会报错）

```
//找到字符串里小数点的位置，如果超过一个小数点会报错
    int point = 0;
    int multiPoint = 0;
    for(int j = currentLen; j > 0; j--){
        if(strnd[j-1] == '.' && multiPoint == 0){
            if(sci_pos != 0){
                point = currentLen - j -1 - sci_pos;  //倘若有科学计数法的情况
            }
            else{
                point = currentLen - j;  //没有科学技术法的情况
            }
            multiPoint = 1;
        }
        else if(strnd[j-1] == '.' && multiPoint == 1){  // 出现多个.的情况striped str
            fprintf(stderr,"you have input multiple points, please check your input number!");
            exit(-1);
        }
    }
```

查看字符串里是否有小数点，并确认小数点最多只有一个

```
// 处理数字字符串冗余的0，即"00123" -> "123",同时去除小数点和e指数
    unsigned char * striped_str = strip_str(str, point, sci_pos)
```

去除数组中多余的0、小数点和e指数，如001.234e6 → 1234

```
if(sci_pos!=0){  //启动科学计数法时
        for(int i = strlen(str_origin) - sci_pos; i <= strlen(str_origin) - 1; i++){

            sci = 10*sci + str_origin[i] - '0'; //计算科学计数法0数量
            // printf("sci=%d,i=%d\n",sci,i);
        }
        // currentLen = currentLen - sci_pos - 1; //修正e指数去除之后的长度
    }
```

首先判断是否启用科学计数法，若启动，则计算科学计数法中0的数量，保存到sci中

```
//将数字填充完整
if(point <= sci){
        sci = sci - point;
        len = len + sci;  //长度变更
        point = 0;
    }
    else{
        point = point - sci;
        sci = 0;
    }
    int i = 0;
    for(i = 0; i < sci; i++){
        striped_str[len-sci+i] = '0';
    }
    striped_str[len] = '\0';
```

将之前的001.234e6 → 1234变成1234000成为可运算的字符串

```
// 指定数据位长度即符号，创建一个大数模板
    bignumber_s *temp = make_bignumber_temp(len, sign, point,sci);

// 构造大数模板，len指向大数的数据位数，sign表示该大数的符号,point表示小数点在第几位,sci表科学计数法
bignumber_s *make_bignumber_temp(int len, int sign, int point, int sci)
{
    // 分配bignumber_s及其所代表数据 所需的内存
    bignumber_s *temp = malloc(sizeof(bignumber_s)+len);  //定义temp为bignumber_s的指针

    if (NULL == temp) {
        perror("Malloc"); //把一个描述性错误消息输出到标准错误 stderr，相当于出错后会在前面打上Malloc
        exit(-1); //退出程序，同时将-1写入环境变量ERRORLEVEL
    }
    temp->sign = sign;  //提取结构体变量bignumber_s的sign值
    temp->len  = len;
    temp->point = point;
    temp->sci = sci;
    memset(temp->data, 0, len); //将temp里data的值的前len位填充0字符
    return temp;
}
```

按照给定输入，创建一个大数模板，并传入参数。

```
void fill_data_fromstr(bignumber_s *n, const char *str)
{
    int i = 0;
    int len = n->len;
    for (i=0; i<len; i++) {
        // printf("str[%d]=%s\n",len-1-i,str[len-1-i]);
        int d = str[len-1-i]-'0';  //这是反着填充的，str的最后一位填在大数的data的第一位
        if (d>=0 && d<=9)
            n->data[i] = d;
        else {
          fprintf(stderr, "Invalid Number:%s\n", str);
          exit(-1);
        }
    }
}
```

往模板里填充入字符，为了后续计算方便，这里把字符串反过来填充。

这样，我们就完成了一个大数的构建了。在两个数字字符串都构建完成后，需要进行对应的加减乘除操作。

加法运算

```
bignumber_s *calc_add(bignumber_s *a, bignumber_s *b)
{// 情况一和情况四
    if (a->sign==b->sign) {
        // n位数+m位数，其和最大为max(n,m)+1位数
        int point = MAX(a->point,b->point); //a和b较大的一个数字的小数位数是和的小数位数
        int len = MAX(a->len-a->point, b->len-b->point) + 1 + point;  //如果超过10，就要进1
        bignumber_s *result = make_bignumber_temp(len,a->sign,point,0);  //符号与a保持一致
        result->sci = judgeUseSci(a,b);
        add_impl(a,b,result);
        return result;
    } else if (a->sign == 0 && b->sign == 1) {//情况二，变成减法
        b->sign = 0; //b数去绝对值
        return calc_sub(a,b);
    } else if (a->sign == 1 && b->sign == 0) {//情况三，变成减法
        a->sign = 0; //a数去绝对值
        return calc_sub(b,a);
    }
}
```

正数加正数和负数加负数为情况一和四，进行add_impl计算。

正数加负数和负数加正数为情况二和三，需要进行salc_sub减法计算.

```
// 实现无符号加法，a和b为加数，r为和
void add_impl(bignumber_s *a, bignumber_s *b, bignumber_s *r)
{
    int i = 0;
    char carry = 0;
    int len = r->len;
    int point = r->point;
    int times = add_times(a,b);
    //带小数进行计算
    for (i=0; i<times; i++) {
        if((a->point + i - point) < 0 || a->point + i - point > (a->len-1)){
            carry += b->data[b->point+i-point];        }
        else if(b->point + i - point < 0 || b-> point + i - point > (b->len-1)){
            carry += a->data[a->point + i - point];        }
        else{
            carry += a->data[a->point+i-point] + b->data[b->point+i-point];
        }
        if(i == times - 1 && carry >= BASE) r->data[times] = 1;
        r->data[i] = carry%BASE;
        carry /= BASE;
    //计算整数
    }
}
```

加法计算方法：找到小数位数对齐，逐位相加，若另一个数字该位没有值则只加一个数字，超过10进1

减法计算

```
//先判断大小，再进行无符号的减法，用大数减小数
bignumber_s *calc_sub(bignumber_s *a, bignumber_s *b)
{
    // 情况一
    if(a->sign==0 && b->sign==0) {
        if (cmp(a,b)>=0) { // 被减数大于等于减数，即差大于等于0时
                int len = a->len;
                int point = MAX(a->point,b->point);
                bignumber_s *result = make_bignumber_temp(len,0,point,0);
                sub_impl(a,b,result);
                return result;
        } else { // 被减数小于减数，即差小于0时
                int len = b->len;
                int point = MAX(a->point,b->point);
                bignumber_s *result = make_bignumber_temp(len,1,point,0);
                sub_impl(b,a,result);  //先确认结果是负号后，直接调换输入顺序，进行正的无符号减法
                return result;
        }
    } else if (a->sign==1 && b->sign==1) { //情况四，负减负等于正减正，调换顺序，变成情况一
        b->sign=0;
        a->sign=0;
        return calc_sub(b,a); //调换减数和被减数，变成情况一
    }else if (a->sign==0 && b->sign==1) { // 情况二，正减负等于加法
        b->sign=0;
        bignumber_s *result = calc_add(a,b);
        result->sign = 0;
        return result;
    } else if (a->sign==1 && b->sign==0) { // 情况三，负减正等于负数加法
        a->sign=0;
        bignumber_s *result = calc_add(a,b);
        result->sign = 1;
        result->sci = judgeUseSci(a,b);
        return result;
    }
}
```

情况一和四是正数减正数和负数减负数，需要进行减法运算。由于是无符号减法运算，首先需要对比cmp输入数字的大小，把大数作为前项输入，小一点的数作为后一项输入，然后确定输出的符号，进行sub_impl计算。如果是情况二三（正数减负数和负数减正数）则前往加法运算。

```
// 实现无符号减法，a-b , r为差
void sub_impl(bignumber_s *a, bignumber_s *b, bignumber_s *r)
{
        int i=0;
        int borrow = 0;
        int len = r->len;
        int point = r->point;
        int temp = 0;
        int times = add_times(a,b);
        for (i=0; i<times; i++) {
            if(a->point+i-point<0 || a->point+i-point>(a->len-1))
                temp = 0 + BASE - borrow  - b->data[b->point+i-point];
            else if(b->point+i-point<0 || b->point+i-point>(b->len-1))
                temp = a->data[a->point + i - point] + BASE - borrow;
            else{
                temp = a->data[a->point + i -point] + BASE - borrow - b->data[b->point + i - point];
            }
            r->data[i] = temp%BASE;
            borrow = temp / BASE ? 0 : 1;
```

```
        }
}
```

无符号减法的具体实现过程：按照小数点对齐，并逐位相减，保留余数，保留借位。

乘法计算

```
bignumber_s *calc_mul(bignumber_s *a, bignumber_s *b)
{
    int len = a->len + b->len;
    int point = a->point + b->point;
    bignumber_s *result = make_bignumber_temp(len,a->sign==b->sign?0:1,point,0); //a->sign==b->sign为情况一，否则为情况二。
    mul_impl(a,b,result);
    result->sci = judgeUseSci(a,b);
    return result;
}
```

先判断输出结果的正负号：如果输入的符号不一致则负号，输入符号一致则等于输入符号。

```
// 实现无符号乘法，x * y , z为积
void mul_impl(bignumber_s *x, bignumber_s *y, bignumber_s *z)
{
    int n = x->len;
    int m = y->len;
    int i = 0;
    int j = 0;
    x->point + y->point;
    int carry = 0;
    for (i=0; i<m; i++) {
        // y的每一位乘以x，即计算y[i] * x[0...n] 并累加到z[i...i+n]中
        for (j=0; j<n; j++) {
            carry += y->data[i]*x->data[j] + z->data[i+j];
            z->data[i+j] = carry%BASE;
            carry /= BASE;
        }

        // 将剩余的进位,继续在z[i+n...n+m]中累加，从而完成y[i]乘以x[0...n]其积累加到z[i...m+n]中
        for (; j+i<n+m; j++) {
            carry += z->data[i+j];
            z->data[i+j] = carry % BASE;
            carry /= BASE;
        }
    }
}
```

乘法的具体实现：逐位相乘，并累加到结果当中

除法计算

```
// 大数除法实现
bignumber_s *calc_div(bignumber_s *a, bignumber_s *b)
{
```

```
    bignumber_s *zero = make_bignumber_temp(1,0,0,0);
    if (!judge_zero(b)) {// 除数为0 报错
        fprintf(stderr,"Integer division by zero\n");
        exit(-1);
    }else if (!judge_zero(a)) { // 被除数为0 ，结果为0
        return zero;
    }
    int alen = a->len;  //记录原始a的长度
    int blen = b->len;
    int p = decimal_digits; //保留p位小数

    //判断a,b谁更长
    int value = b->point - a->point + p;
    a = add_zero(a, value);  //a的长度会发生变化
    int clen = a->len - blen + 1;
    bignumber_s * aa = make_bignumber_temp(blen,0,0,0);
    bignumber_s * bb = make_bignumber_temp(blen,0,0,0);
    bignumber_s * bbb = make_bignumber_temp(blen+1,0,0,0);  //创建一个bb的数位加一的版本，方便与ccc比较大小
    bignumber_s * cc = make_bignumber_temp(blen,0,0,0);
    bignumber_s * ccc = make_bignumber_temp(blen + 1,0,0,0);
    bignumber_s * ccc_medium = make_bignumber_temp(blen + 1,0,0,0);
    bignumber_s * result = make_bignumber_temp(clen,0,p,0);
    int len = blen;
    for(int i = 0; i <= a->len - blen; i++){  //a是补充完0后的
        if(i == 0){
            for(int j = 1; j <= len; j++){
                aa->data[len-j] = a->data[a->len-j];
                bb->data[len-j] = b->data[blen-j];
                bbb->data[bbb->len-j-1] = b->data[blen-j];  //从倒数第二位开始补充
            }
            bbb->data[bbb->len-1] = 0;  //最后一位补上0
            cc = aa;
            int time = 0;
            while(bigger(cc,bb)>=0){
                aa = cc;
                sub_impl(aa,bb,cc);  //输出存储在cc；里
                time++;
            }
            result->data[clen-i-1] = time;
            time = 0;
        }
        else if(i == 1){
            //为ccc赋值，ccc包含了a的下一位
            for(int j = 1; j <= blen; j++){
                ccc->data[blen-j+1] = cc->data[blen-j];
            }
            ccc->data[0] = a->data[a->len-blen-i];//赋值a的下一位
            int time = 0;  //计数

            while(bigger(ccc,bbb)>=0){
                ccc_medium = ccc;
                sub_impl(ccc_medium,bbb,ccc);  //ccc和ccc_medium一起减小bb
                time++;
            }
            result->data[clen-i-1] = time;
            time = 0;
        }
        else{
            //与上式基本相同，把数据前移,ccc长度时blen+1
            for(int j = 1; j <= ccc->len - 1; j++){
                ccc->data[ccc->len-j] = ccc->data[ccc->len-j-1];
            }
            ccc->data[0] = a->data[a->len-blen-i];//赋值a的下一位
            int time = 0;  //计数

            while(bigger(ccc,bbb)>=0){
                ccc_medium = ccc;
                sub_impl(ccc_medium,bbb,ccc);
                time++;
```

```
            }
            result->data[clen-i-1] = time;
            time = 0;
        }
    }
    // print_bignumber(result);
    result->sci = judgeUseSci(a,b);
    if(a->sign == b->sign) result->sign = 0;
    else result->sign = 1;
    return result;
}
```

大数除法我是完全按照笔算除法来做的，可能写的相对复杂。基本思路是：先根据小数点位数，给**被除数a**补充0或者减去0（改变被除数长度），然后**第一次相除**时，使用与**除数b**位数相同的数**bb**与被除数的**对应位数累减**，差保存在**cc**中。直到**cc**小于**bb**，将**cc**的值赋给**ccc**的对应位，并从被除数**a**中取一位出来给**ccc（ccc比cc多一位）**。然后从第二次循环开始，使用**ccc**与**bbb**（比bb多一位数，仅仅为了与ccc位数相同，与bb内容一致）继续上述的累减、判断操作，保留给**ccc**。每次进行累减的时候统计累减的次数（time），该数字则为结果的每一位的值。

输出

```
void print_bignumber(bignumber_s * b)
```

最后，就由该函数输出。

在输出时，会首先判断输入是否有科学计数法，以及最后计算结果是否需要用科学计数法表示。

值得一提的是，在宏定义部分定义了保留小数的位数，用户可以根据自身需要设置保留小数的位数。

```
#define decimal_digits (3)  //这里可修改保留的小数位数
```

task输出展示：（代码里乘法是x，不是*）

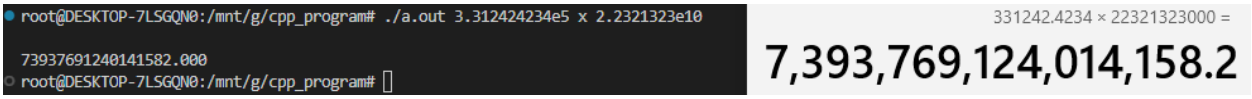
```
root@DESKTOP-7LSGQN0:/mnt/g/cpp_program# ./calculator 2 + 3

5
root@DESKTOP-7LSGQN0:/mnt/g/cpp_program# ./calculator 2 - 3

-1
root@DESKTOP-7LSGQN0:/mnt/g/cpp_program# ./calculator 2 * 3
Invalid Number:2o
root@DESKTOP-7LSGQN0:/mnt/g/cpp_program# ./calculator 2 x 3

6
root@DESKTOP-7LSGQN0:/mnt/g/cpp_program# ./calculator 2 / 3

0.66
root@DESKTOP-7LSGQN0:/mnt/g/cpp_program# ./calculator 3.14 / 0
Integer division by zero
root@DESKTOP-7LSGQN0:/mnt/g/cpp_program# ./calculator a * 2
Invalid Number:2o
root@DESKTOP-7LSGQN0:/mnt/g/cpp_program# ./calculator a x 2
Invalid Number:a
root@DESKTOP-7LSGQN0:/mnt/g/cpp_program# ./calculator 987654321 * 987654321
Invalid Number:2o
root@DESKTOP-7LSGQN0:/mnt/g/cpp_program# ./calculator 987654321 x 987654321

975461057789971041
root@DESKTOP-7LSGQN0:/mnt/g/cpp_program# ./calculator 1e200 x 1e200

1e400
root@DESKTOP-7LSGQN0:/mnt/g/cpp_program#
```

亮点部分

- 完成task要求
- 可以自动抹去输入的无效0位(输入00012 = 12)
- 可自行判断是否启用科学计数法形式输出
- 可根据需要设置保留小数位数
- （仅仅通过自身学习完成了这个项目，没有请教别人和ChatGPT）

问题/未来的目标

虽然这个项目到这里基本上完成了，但还有一些bugs。比如在计算用科学计数法表示的小数的时候，计算器有时会计算错误。比如：



除此之外，目前应该没有其他错误了（也可能是我尝试的次数还不够多）

总体说来，这个项目给了三周时间，但是从配完环境，基本知道c的语句后，能用来做project的时间就不是很多了，在功能上本来想加入e的负数形式的计算的，然而因为时间不太够了估计只能以后再做。除了负的科学计数法之外，未来还可以加入GUI，实现保留sum等操作。

结语

通过这个project，我了解了c语言的许多语句的用法，了解指针的使用方式，学会了基本的计算器的设计思路。同时，也在一次次编译，出错，debug中磨炼了心境、提升了能力。虽然现在自身的水平还不足，但随着这一个个project的完成，自身水平一定会提升。