

电子信息与通信学院

实 验 报 告

|  |  |
| --- | --- |
| 实验名称 | 课程综合练习 |
| 课程名称 | 计算机基础  与程序设计(C) |

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 郑蕊 | 学号 | U202410367 |

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 2025.2.19 | 地点 | 华中科技大学 |

|  |  |  |  |
| --- | --- | --- | --- |
| 成绩 |  | 教师 | 刘威 |

# 实验目的

完成大叔计算系列代码。

# 实验环境

操作系统：Windows 10

编程工具：CodeBlocks 16.01

# 实验一：长整数相加

## 实验任务

1.用一维字符数组记录长整数的每个数位，通过ASCII字符值转换获取每个数位代表的数值

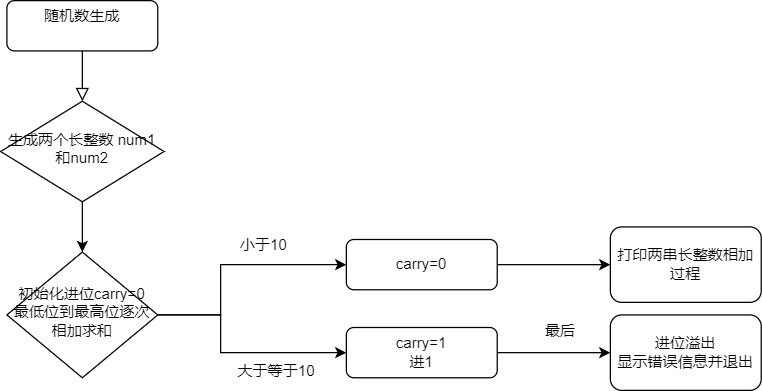
2.随机产生两个长整数，约定其最高位不得为0

3.对两个相同长度的长整数进行求和，有加法进位导致存储位数不足时显示报错信息

4.打印两个长整数求和的竖式计算过程

## 实验步骤

实验思路：



关键代码：

溢出时报错

1. **else** **if** ( isOverflow == 1 )
2. {
3. printf("\nError! addition overflow!\n");
4. }
5. //如果溢出，则显示报错信息

从后逐次相加，以及进位的问题

1. **for** (**int** i = INT\_WIDTH - 1; i >= 0; i--)
2. {
3. **int** digit\_1 = longInt1[i] - '0';
4. **int** digit\_2 = longInt2[i] - '0';
5. **int** sum = digit\_1 + digit\_2 + add;
6. **if** (sum > 9)
7. {
8. add = 1;
9. sum -= 10;
10. }
11. **else**
12. {
13. add = 0;
14. }
15. longIntSum[i] = sum + '0';
16. }

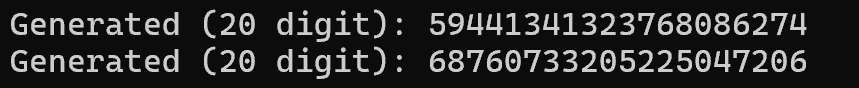
## 代码测试

### void generateLongInt( char longInt[ ] )函数测试点的测试结果

测试步骤：在 main 函数中调用该函数生成多个长整数，观察生成的长整数是否符合预期

预期测试结果：生成随机的长整数，且最高位不为0

实际测试结果（截图）：



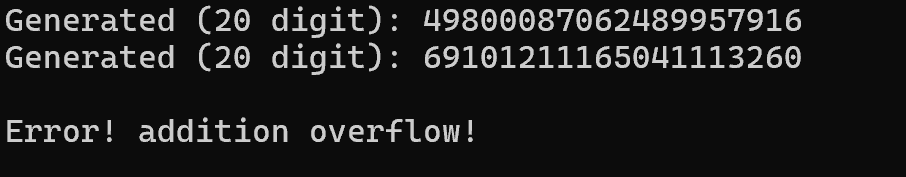
测试结论：测试通过

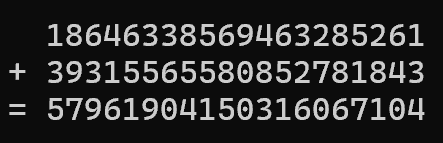
### addTwoLongInts函数测试点的测试结果

测试步骤：调用该函数检查输出结果是否是两个长整数的和

预期测试结果：若进位不溢出，则正确输出两长整数的和；若进位溢出，则显示报错信息

实际测试结果（截图）：





测试结论：测试通过

## 实验结论

代码达到功能目标

## 实验总结

1. 此次实验并不是字符串，没有’\0’作为结尾的规定，全部数位都用于存储数值
2. 应当注意加法进位的正确处理方法
3. 注意有加法进位导致存储位数不足时显示错误信息的情况

# 实验二：长实数相加

## 实验任务

1.用一维字符数组记录长实数的每个数位，通过ASCII字符值转换获取每个数位代表的数值，用一个数位来存储小数点’.’

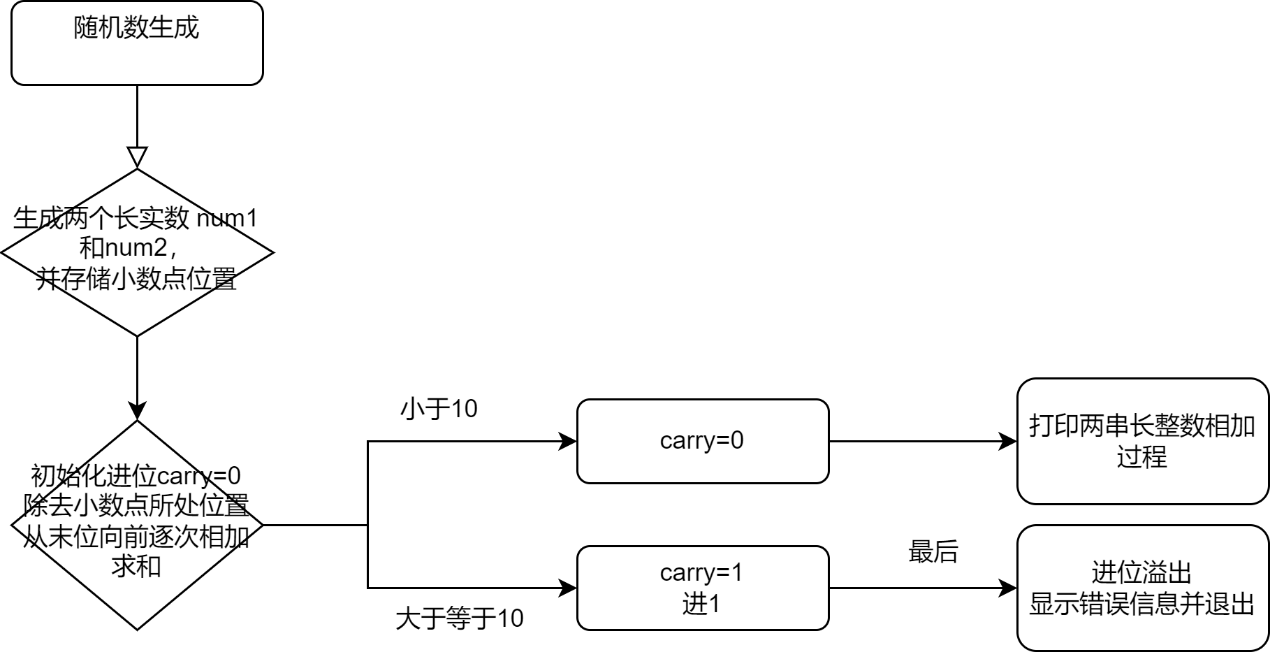
2.随机产生两个长实数，约定其最高位不得为'0'

3.对两个相同长度的长实数进行求和，有加法进位时显示报错信息

4.打印两个长实数求和的竖式计算过程

## 实验步骤

实验思路：



关键代码：求和时除去存储小数点的数位

1. **if** (i == point\_position - 1)
2. {
3. longrealsum[i] = '.';
4. }

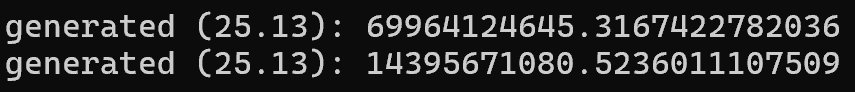
## 代码测试

### generatelongreal函数测试点的测试结果

测试步骤：在 main 函数中调用该函数生成多个长实数，观察生成的长实数是否符合预期

预期测试结果：生成的长实数最高位不为0，小数点位置固定

实际测试结果（截图）：

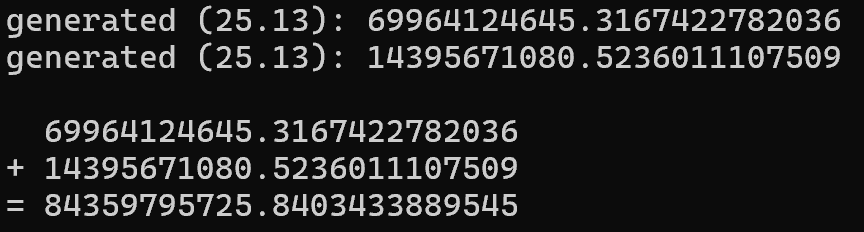


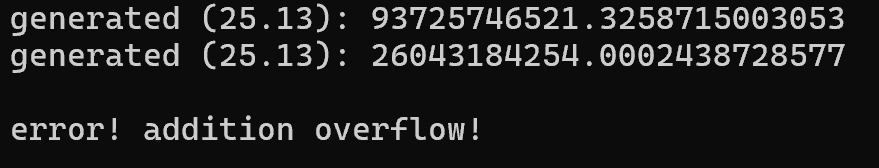
测试结论：测试通过

### Addtwolongreals函数测试点的测试结果

测试步骤：调用该函数检查输出结果是否是两个长实数的和

预期测试结果：若进位不溢出，则正确输出两长实数的和；若进位溢出，则显示报错信息

实际测试结果（截图）：



测试结论：测试通过

## 实验结论

代码达到功能目标

## 实验总结

1．此实验是在实验一的基础上改为用字符数组存储数值和小数点，应当注意到用字符数组来表达长实数是更为有效的方法

2. 存储结构与显示信息的设计，模仿printf( )函数的格式输出符“m.n”的表达，用一个数位来存储小数点

3. 相比实验一需要注意此实验求和应除去小数点所在数位并从末尾依次向前求和

# 实验三：固定点小数相加

## 实验任务

1.结构体内，用一维字符数组记录长实数的每个数位，通过ASCII字符值转换获取每个数位，用一个数位来存储小数点’.’，用一个整数记录长实数的长度。

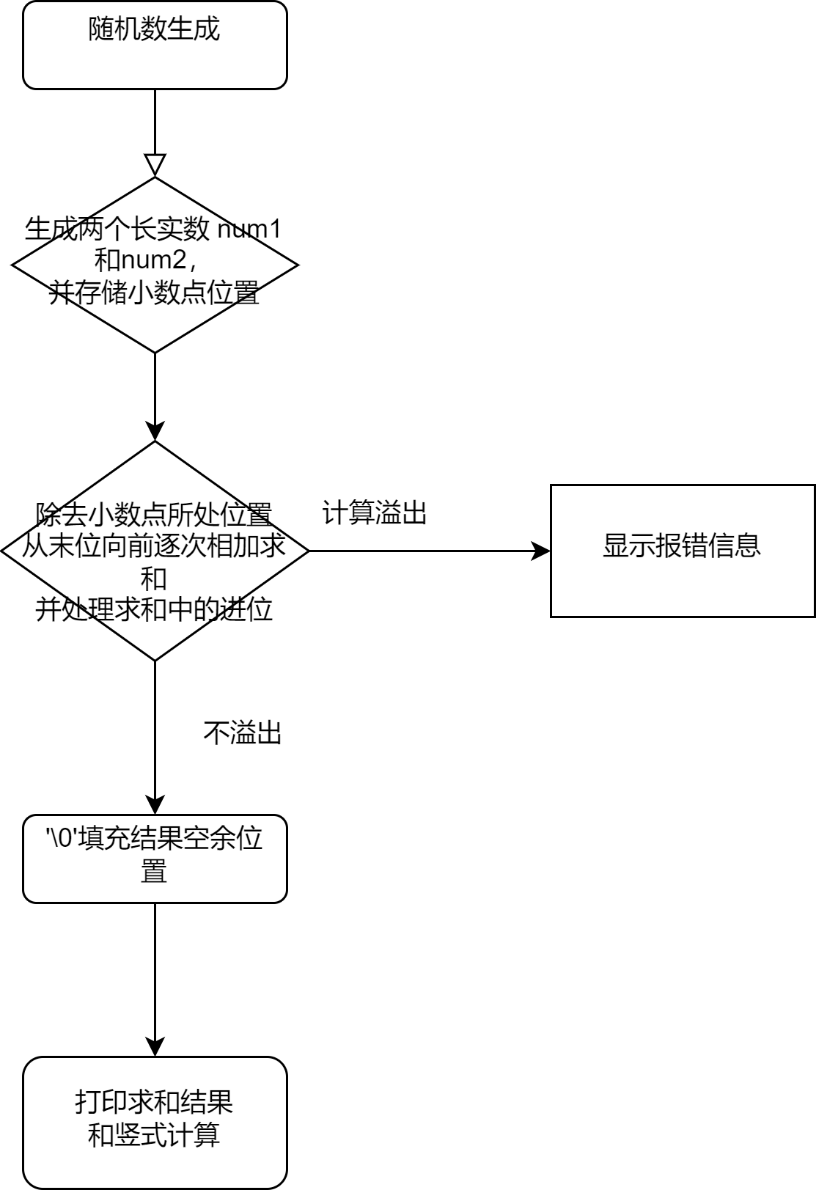
2.随机产生两个长整数，约定其最高位不得为’0’，其长度可以不同，但小数点位置相同

3.对两个整数部分长度相同的长实数进行求和，有加法进位时显示报错信息

4.打印两个长整数求和的竖式计算过程

## 实验步骤

实验思路：



关键代码：求和并判断是否溢出

1. **for**(i = maxLen - 1 ; i >= 0; i--)
2. {
3. **if**( r1.digits[i] == '.' )
4. {
5. r.digits[i] = '.';
6. **continue**;
7. }
8. **else**
9. {
10. sum = carry;
11. **if**(r1.digits[i] != '\0' )
12. {
13. sum += r1.digits[i] - '0';
14. }
15. **if**( r2.digits[i] != '\0' )
16. {
17. sum += r2.digits[i] - '0';
18. }
19. }
20. carry = sum / 10;
21. r.digits[i] = '0' + sum % 10;
23. }
25. **if**( carry > 0 )
26. {
27. //溢出判断
28. r.length = 0 ;
29. **return** r;
30. }
31. **else**
32. {
33. r.length = maxLen;
34. }

关键代码：填充未填充的数组位置为’\0’

1. **for**( i=r.length; i < MAX\_REAL\_WIDTH ; i++ )
2. {   // clear the rest of the memory
3. //生成随机数时，没有用到的数位填充'\0';
4. //显示正常数值时，无需打印；显示全部数宽的时候，空闲数位用字符'\_'代替
5. r.digits[i] = '\0';
6. }
7. r.digits[ MAX\_REAL\_WIDTH - 1 ] = '\0';

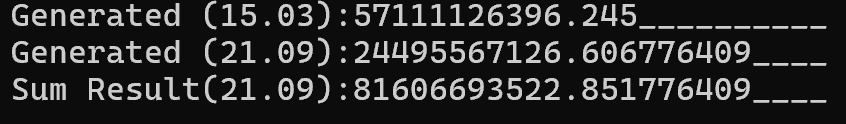
## 代码测试

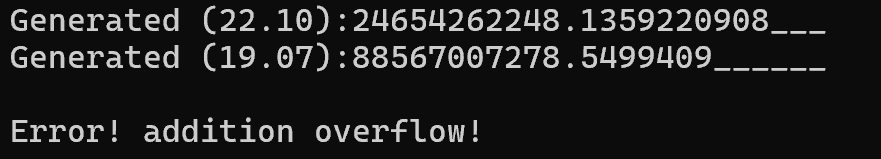
### addTwoLongReals中求和测试点的测试结果

测试步骤：在main函数中调用该函数，观察输出的sum result是否符合预期测试结果

预期测试结果：计算溢出时会显示报错信息，不溢出时正确显示预期结果

实际测试结果（截图）：





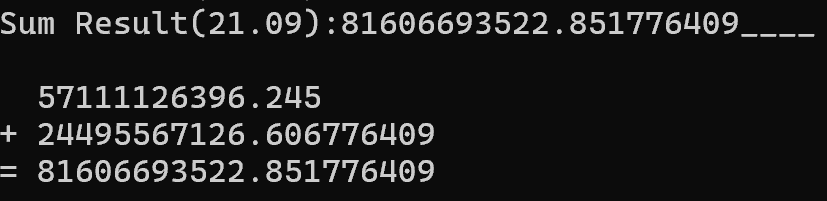
测试结论：测试通过

### addTwoLongReals中填充实数测试点的测试结果

测试步骤：观察比较竖式计算和此函数输出的sum result的非数位的显示区别

预期测试结果：显示位宽时空余位置会用’-‘显示，显示实数加法时会不显示

实际测试结果（截图）：



测试结论：测试通过

## 实验结论

代码达到功能目标

## 实验总结

1. 此实验是实验二的结构体版，现有知识只支持通过数组名实现基于引用的传递，当数据结构变为结构体时，需要重新设计基于值传递的系列函数；
2. 注意将结构体作为函数返回值带回
3. 确保在数组末尾添加字符串结束符 '\0'，以正确处理字符串操作；
4. 将未使用的数组位置填充为 '\0'，以便在正常打印时不显示，而在显示全部位宽时用字符 '\_' 代替。

# 实验四：浮动点小数相加

## 实验任务

1.结构体内，用一维字符数组记录长实数的每个数位，通过ASCII字符值转换获取每个数位，用一个数位来存储小数点’.’，用一个整数记录长实数的长度，用一个整数记录小数点位置

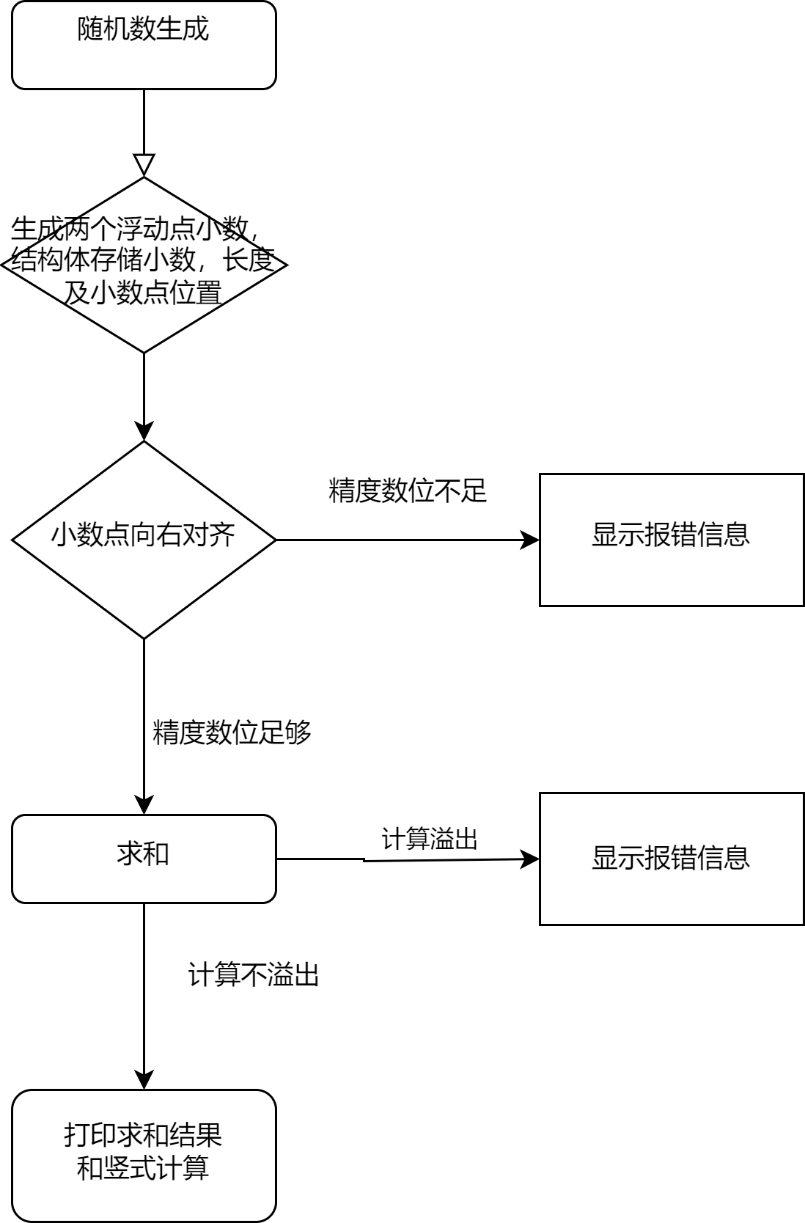
2.随机产生两个长整数，约定其最高位不得为‘0’，其长度、小数点位置都可以不同

3.对两个整数部分长度不同的长实数进行求和，有加法进位时显示报错信息

4.打印两个长实数求和的竖式计算过程，需要根据小数点位置不同调整输出格式

## 实验步骤

实验思路：



关键代码：生成浮动点小数

1. Real generateLongReal( **void** )
2. {
3. Real r;
4. **int** i;
5. **int** len = rand() % ( MAX\_REAL\_WIDTH + 1 );
6. r.pointPos = rand() % len  + 1 ;
7. r.length = len;
9. **for**( i = 0; i < len ; i++ )
10. {
11. **char** digit = rand() % 10;
12. **if**( i == r.pointPos-1 )
13. {
14. r.digits[i] = '.';
15. }
16. **else**
17. {
18. r.digits[i] = '0' + digit;
19. }
20. }
21. r.digits[0] = '1' + rand() % 9;
22. r.digits[ r.length - 1 ] = '1' + rand() % 9;
23. **if**( r.pointPos == 1 )
24. {
25. r.digits[0] = '.';
26. }
28. **if**( r.pointPos == r.length )
29. {
30. r.digits[ r.length - 1 ] = '.';
31. }
33. **for**( i = len; i < MAX\_REAL\_WIDTH; i++ )
34. {
35. r.digits[i] = '\0';
36. }
38. r.digits[ MAX\_REAL\_WIDTH - 1 ] = '\0';
39. printf("Generated  (%2d.%02d):", r.length, r.length - r.pointPos );
40. displayFixedwidthNumber( r.digits, MAX\_REAL\_WIDTH );
42. **return** r;
43. }

关键代码：小数点位置向右/左移动(基本同理)

1. **void** shiftDigitsToRight( **char** number[],**int** shiftLength)
2. {
3. **int** i;
4. **for**( i = MAX\_REAL\_WIDTH - 1; i >= shiftLength; i-- )
5. {
6. number[i] = number[i-shiftLength];
7. }
8. **for**( i = 0; i < shiftLength; i++ )
9. {
10. number[i] = '\0';
11. }
12. printf("shiftRight  (%02d)  :",shiftLength);
13. displayFixedwidthNumber( number, MAX\_REAL\_WIDTH );
14. **return**;
15. }

关键代码：浮动点小数向右对齐

1. //向右对齐
2. **int** r1right = MAX\_REAL\_WIDTH - r1.length ;
3. **int** r2right = MAX\_REAL\_WIDTH - r2.length ;
4. **int** shiftLength = (r1right < r2right) ? r1right : r2right ;
6. //将两个数的小数点向右移动对齐并求和
7. **if**( r1right < r2right )
8. {    //判断表达精度的数位是否足够
9. //是否在移动过程中向右溢出
10. **if**(r2.length+shiftLength +r1.pointPos-r2.pointPos > MAX\_REAL\_WIDTH
11. //是否向左溢出
12. ||r2.pointPos - r1.pointPos > shiftLength )
13. {
14. r.length = 0;
15. **return** r;
16. }
17. **else**
18. {
19. shiftDigitsToRight( r1.digits, shiftLength );
20. shiftDigitsToRight( r2.digits, shiftLength + r1.pointPos-r2.pointPos );
21. }
22. }
23. **else** **if**( r1right > r2right )
24. {
25. **if**(r1.length+shiftLength + r2.pointPos-r1.pointPos > MAX\_REAL\_WIDTH
26. || r1.pointPos - r2.pointPos > shiftLength )
27. {
28. r.length = 0;
29. **return** r;
30. }
31. **else**
32. {
33. shiftDigitsToRight( r1.digits, shiftLength + r2.pointPos-r1.pointPos );
34. shiftDigitsToRight( r2.digits, shiftLength  );
35. }
36. }
37. **else** **if**( r1right == r2right )
38. {
39. **if**(r1.pointPos!=r2.pointPos)
40. {
41. r.length = 0;
42. **return** r;
43. }
44. **else**
45. {
46. shiftDigitsToRight( r1.digits, r1right + abs(r2.pointPos-r1.pointPos) );
47. shiftDigitsToRight( r2.digits, r2right + abs(r2.pointPos-r1.pointPos) );
48. }
49. }

关键代码：求和

1. //求和
2. **for**(i = MAX\_REAL\_WIDTH - 1 ; i >= 0 ; i--)
3. {
4. **if**( r1.digits[i] == '.' )
5. {
6. r.digits[i] = '.';
7. r.pointPos = r.length - (MAX\_REAL\_WIDTH - ( i + 1 ) );
8. **continue**;
9. }
10. **else**
11. {
12. sum = carry;
13. **if**(r1.digits[i] != '\0' )
14. {
15. sum += r1.digits[i] - '0';
16. }
17. **if**( r2.digits[i] != '\0' )
18. {
19. sum += r2.digits[i] - '0';
20. }
21. }
22. carry = sum / 10;
23. r.digits[i] = '0' + sum % 10;
24. }
25. **if**( carry > 0 )
26. {
27. //加法运算有进位，计算溢出
28. r.length = 0 ;
29. **return** r;
30. }
32. **for**( i = MAX\_REAL\_WIDTH - r.length - 1; i >=0 ; i-- )
33. {   // clear the rest of the memory
34. //生成随机数时，没有用到的数位填充'\0';
35. //显示正常数值时，无需打印；显示全部数宽的时候，空闲数位用字符'\_'代替
36. r.digits[i] = '\0';
37. }

关键代码：去除前置零

1. //求向右对齐后长实数
2. **int** r1left = 0,r2left = 0;
3. **while**(r1.digits[r1left] =='\0')
4. {
5. r1left++;
6. }
7. **while**(r2.digits[r2left] =='\0')
8. {
9. r2left++;
10. }
11. **int** rleft = (r1left < r2left) ? r1left : r2left ;
12. r.length = MAX\_REAL\_WIDTH - rleft ;
14. shiftDigitsToLeft( r1.digits, r1left  );
15. shiftDigitsToLeft( r2.digits, r2left );
16. shiftDigitsToLeft( r.digits, rleft );

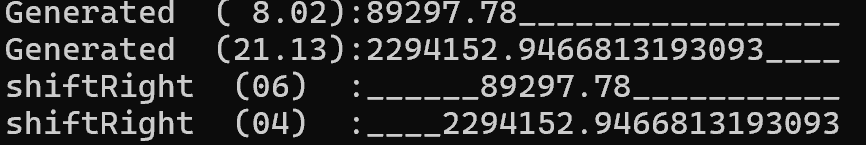
## 代码测试

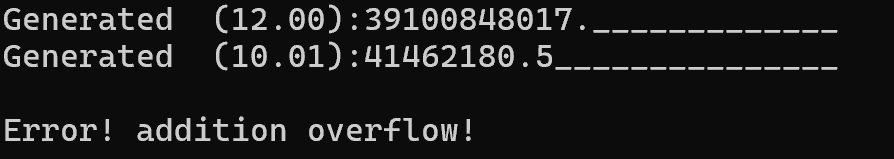
### shiftDigitsToRight测试点的测试结果

测试步骤：在addTwoLongReal里面调用该函数，观察输出结果

预期测试结果：数位精度足够时，两浮动点小数的小数点向右对齐；数位精度不够时显示报错信息

实际测试结果（截图）：





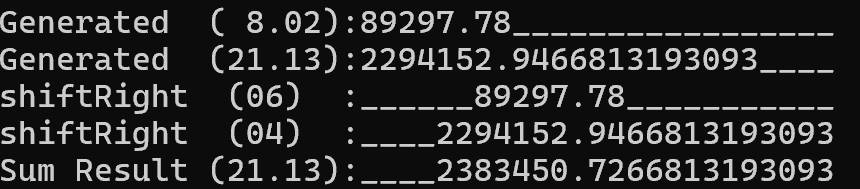
测试结论：测试通过

### addTwoLongReal求和测试点的测试结果

测试步骤：在main函数里调用函数，观察输出结果

预期测试结果：计算溢出时显示报错信息，计算不溢出时输出求和结果

实际测试结果（截图）：



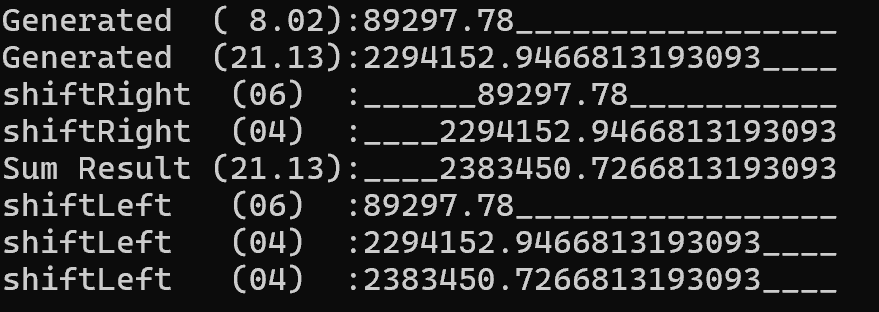
测试结论：测试通过

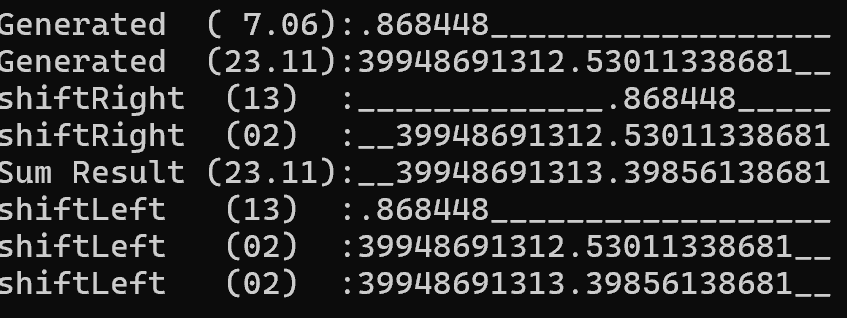
### shiftDigitsToLeft测试点的测试结果

测试步骤：在addTwoLongReal函数里调用函数，观察结果

预期测试结果：两个生成的随机浮动点小数及求和结果向左去除前置的0

实际测试结果（截图）：





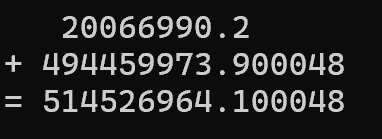
测试结论：测试通过

### displayAssignedLongReal测试点的测试结果

测试步骤：在main函数调用该函数观察是否能打印竖式计算过程

预期测试结果：根据小数点位置调整输出格式打印竖式计算过程

实际测试结果（截图）：



测试结论：测试通过

## 实验结论

代码达到功能目标

## 实验总结

1. 本实验是在实验三的基础上将固定位置的小数点改为浮动点，小数位置随机产生，因此结构体成员增加一个描述小数点的位置
2. 注意在运算正常情况下，程序需要将生成的两个随机数的小数点位置向右对齐并求和，求和完毕后，最后将三个数的小数点位置向左移动，去除前置的0
3. 小数点向右对齐的过程中可能出现前越位和后越位两种情况，需要明确表达精度的数位不足的条件
4. 对齐后求和时可能会出现进位导致的计算溢出
5. 如果不溢出需要通过新增函数实现小数点对齐输出打印算式

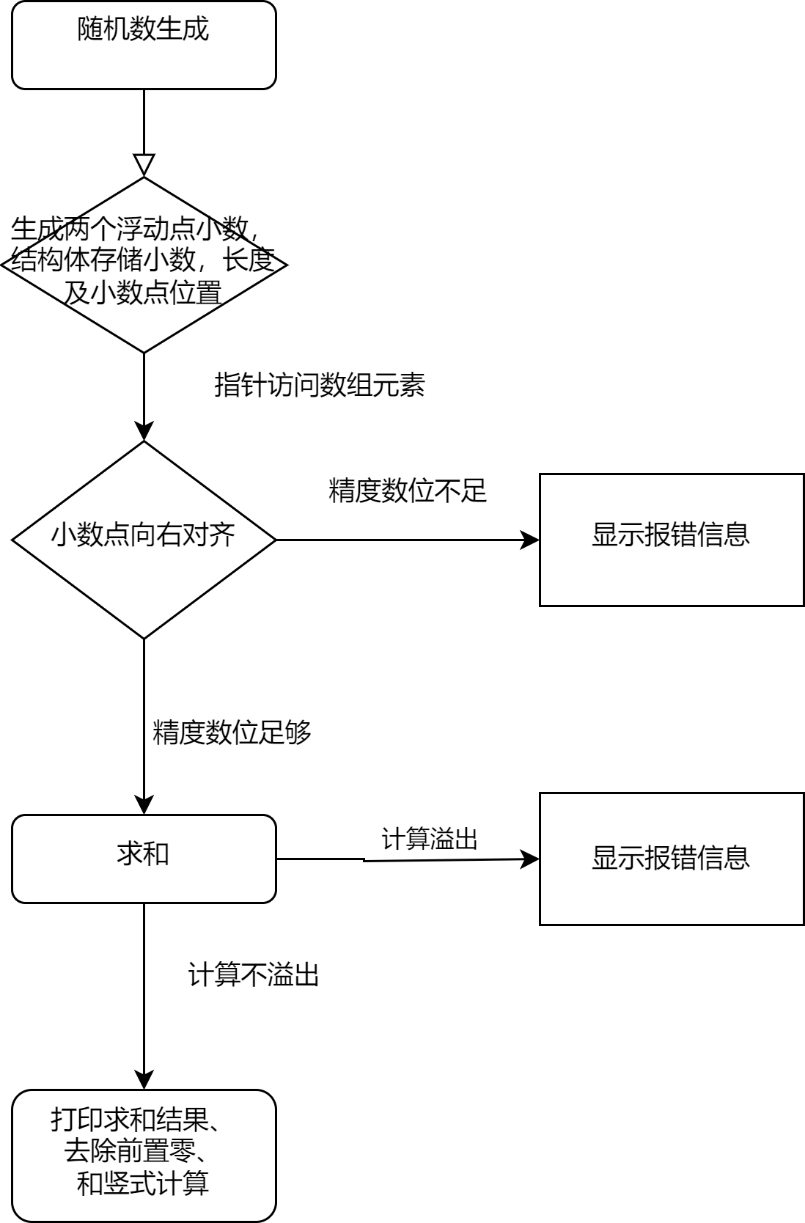
# 实验五：指针操作数组元素

## 实验任务

预期功能与实验四相同，要求修改对应的数组操作的语句，通过指针访问数组元素。

## 实验步骤

实验思路：



关键代码：通过指针访问数组元素打印显示位宽的小数

1. **char** \*cPtr = number;
2. **int** i;
3. **for**( i = 0; i < width; i++ )
4. {
5. **if**( \*(cPtr + i) == '\0' )
6. {
7. printf("\_");
8. }
9. **else**
10. {
11. printf("%c", \*(cPtr + i) );
12. }
13. }
14. printf("\n");

关键代码：指针访问数组元素使小数点向右/左移动

1. **char** \*end = number + MAX\_REAL\_WIDTH - 1;
2. **char** \*start = number + MAX\_REAL\_WIDTH - 1 - shiftLength;
3. **while**( start >= number )
4. {
5. \*end = \*start;
6. end--;
7. start--;
8. }
9. **while**( end >= number )
10. {
11. \*end = '\0';
12. end--;
13. }
15. printf("shiftRight  (%02d)  :",shiftLength);
16. displayFixedwidthNumber( number, MAX\_REAL\_WIDTH );
17. **return**;

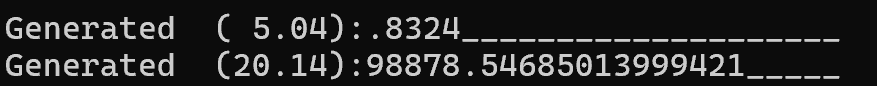
## 代码测试

### displayFixedwidthNumber测试点的测试结果

测试步骤：在main函数调用该函数观察是否能打印显示所有位宽的小数

预期测试结果：是否遍历数组多余位置用’\_’表示

实际测试结果（截图）：



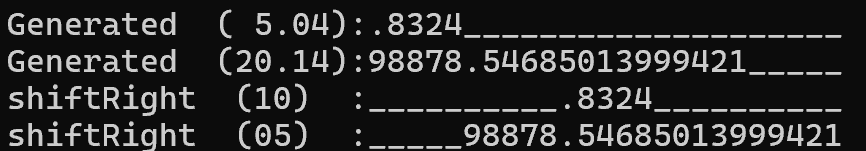
测试结论：测试通过

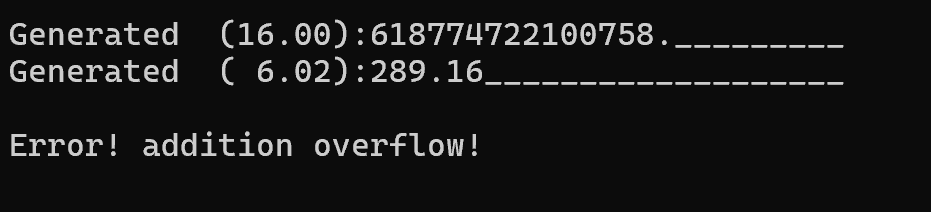
### shiftDigitsToRight测试点的测试结果

测试步骤：在addTwoLongReal里面调用该函数，观察输出结果

预期测试结果：数位精度足够时，两浮动点小数的小数点向右对齐；数位精度不够时显示报错信息

实际测试结果（截图）：





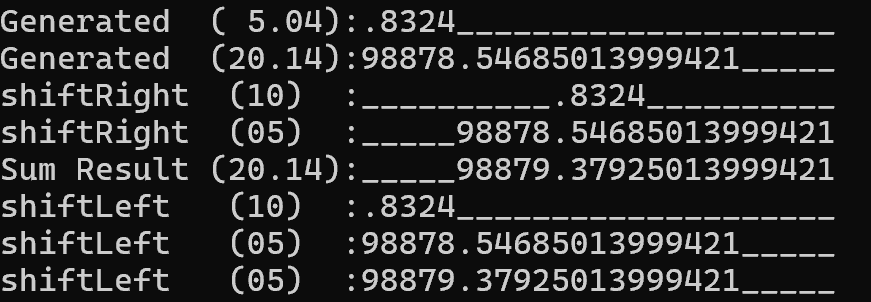
测试结论：测试通过

### shiftDigitsToLeft测试点的测试结果

测试步骤：在addTwoLongReal函数里调用函数，观察结果

预期测试结果：两个生成的随机浮动点小数及求和结果向左去除前置的0

实际测试结果（截图）：



测试结论：测试通过

## 实验结论

代码达到功能目标

## 实验总结

1. 此实验是实验四的指针访问数组元素版，要求修改对应的数组操作的语句，通过指针访问数组元素。因此除求和函数外都需要用指针法修改

2. 实验四中显示正常数值的函数需要更改为基于字符数组指针的地址偏移计算

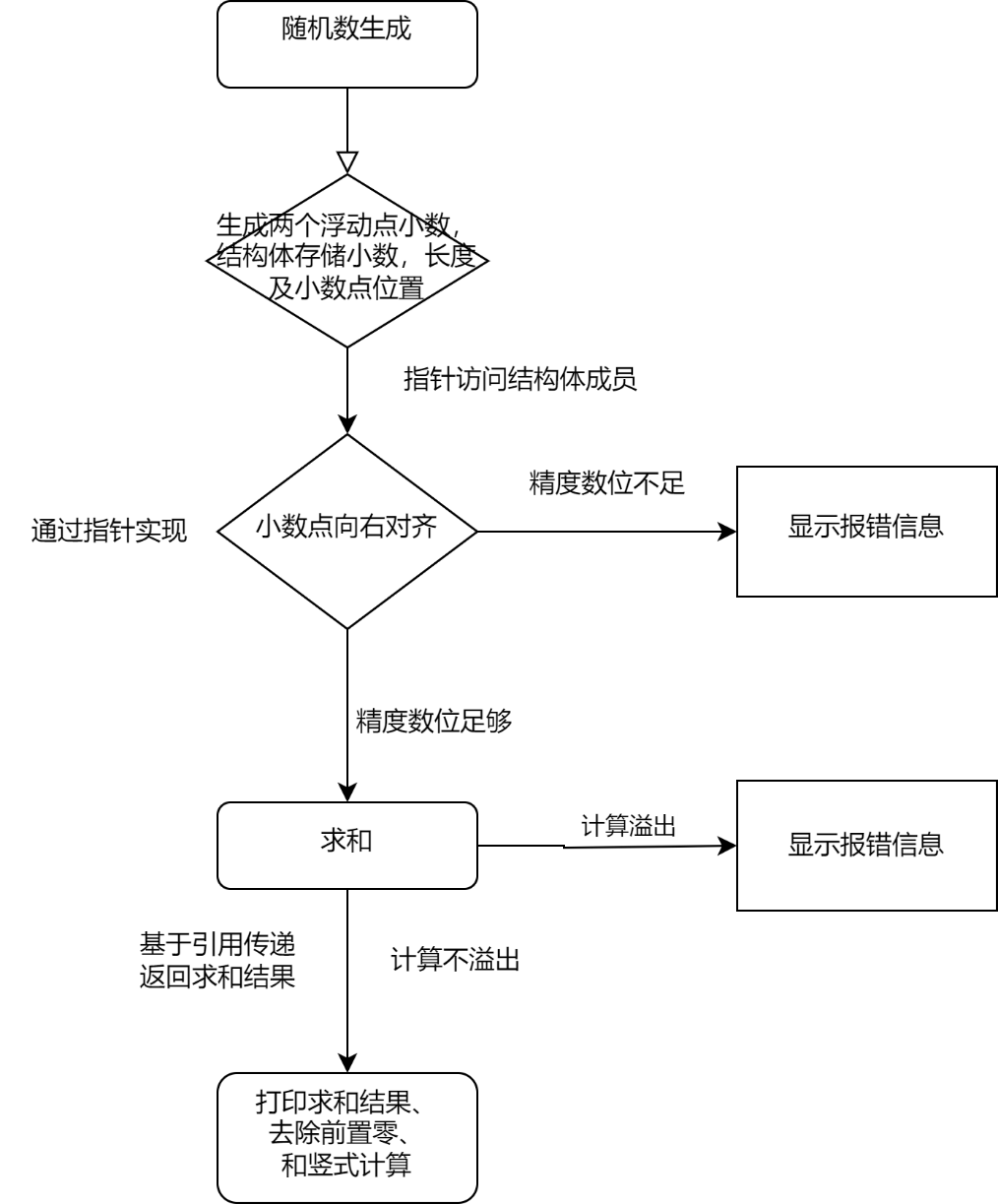
# 实验六：指针操作数组和结构体

## 实验任务

修改函数，通过指针操作相关数组、结构体的函数，实现基于引用的传递

## 实验步骤

实验思路：



关键代码：通过指针实现向右对齐

void shiftDigitsToRight(const char\* longNumber, int shiftLength)

{

    char \*end = longNumber + MAX\_REAL\_WIDTH - 1;

    char \*start = longNumber + MAX\_REAL\_WIDTH - 1 - shiftLength;

    while (start >= longNumber)

    {

        \*end = \*start;

        end--;

        start--;

    }

    while (end >= longNumber)

    {

        \*end = '\0';

        end--;

    }

    printf("shiftRight  (%02d)  :", shiftLength);

    displayFixedWidthNumber(longNumber, MAX\_REAL\_WIDTH);

    return ;

}

关键代码：通过结构体指针基于函数引用传递求和结果

for(i = MAX\_REAL\_WIDTH - 1 ; i >= 0 ; i--)

    {

        if( rPtr1->digits[i] == '.' )

        {

            rPtrSum->digits[i] = '.';

            rPtrSum->pointPos = rPtrSum->length - (MAX\_REAL\_WIDTH - ( i + 1 ) );

            continue;

        }

        else

        {

            sum = carry;

            if(rPtr1->digits[i] != '\0' )

            {

                sum += rPtr1->digits[i] - '0';

            }

            if( rPtr2->digits[i] != '\0' )

            {

                sum += rPtr2->digits[i] - '0';

            }

        }

        carry = sum / 10;

        rPtrSum->digits[i] = '0' + sum % 10;

    }

    if( carry > 0 )

    {

        //加法运算有进位，计算溢出

        rPtrSum->length = 0 ;

        return ;

    }

    for( i = MAX\_REAL\_WIDTH - rPtrSum->length - 1; i >=0 ; i-- )

    {   // clear the rest of the memory

        //生成随机数时，没有用到的数位填充'\0';

        //显示正常数值时，无需打印；显示全部数宽的时候，空闲数位用字符'\_'代替

        rPtrSum->digits[i] = '\0';

    }

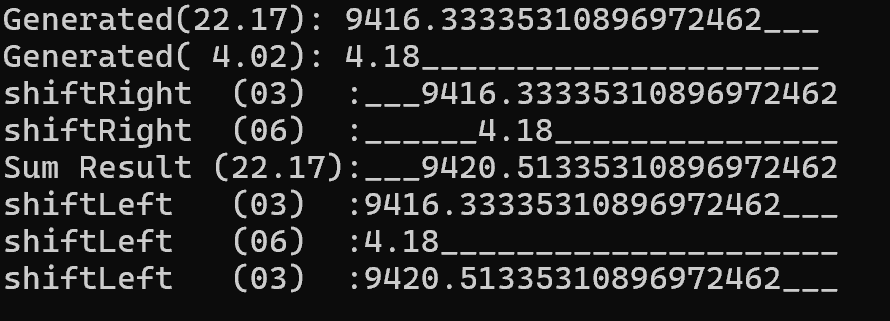
## 代码测试

### addTwoLongReals测试点的测试结果

测试步骤:在main函数里调用该函数，是否通过结构体的指针实现基于引用的传递

预期测试结果：小数点向右对齐正确，求和结果符合预期，且前面0都被除去；有溢出或精度不足会报错

实际测试结果（截图）：



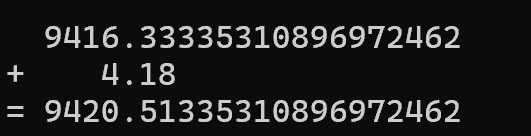
测试结论:测试通过

### displayFixedWidthNumber测试点的测试结果

测试步骤:在main函数里调用

预期测试结果:通过指针实现竖式计算过程

实际测试结果（截图）:



测试结论:测试通过

## 实验结论

代码达到功能目标

## 实验总结

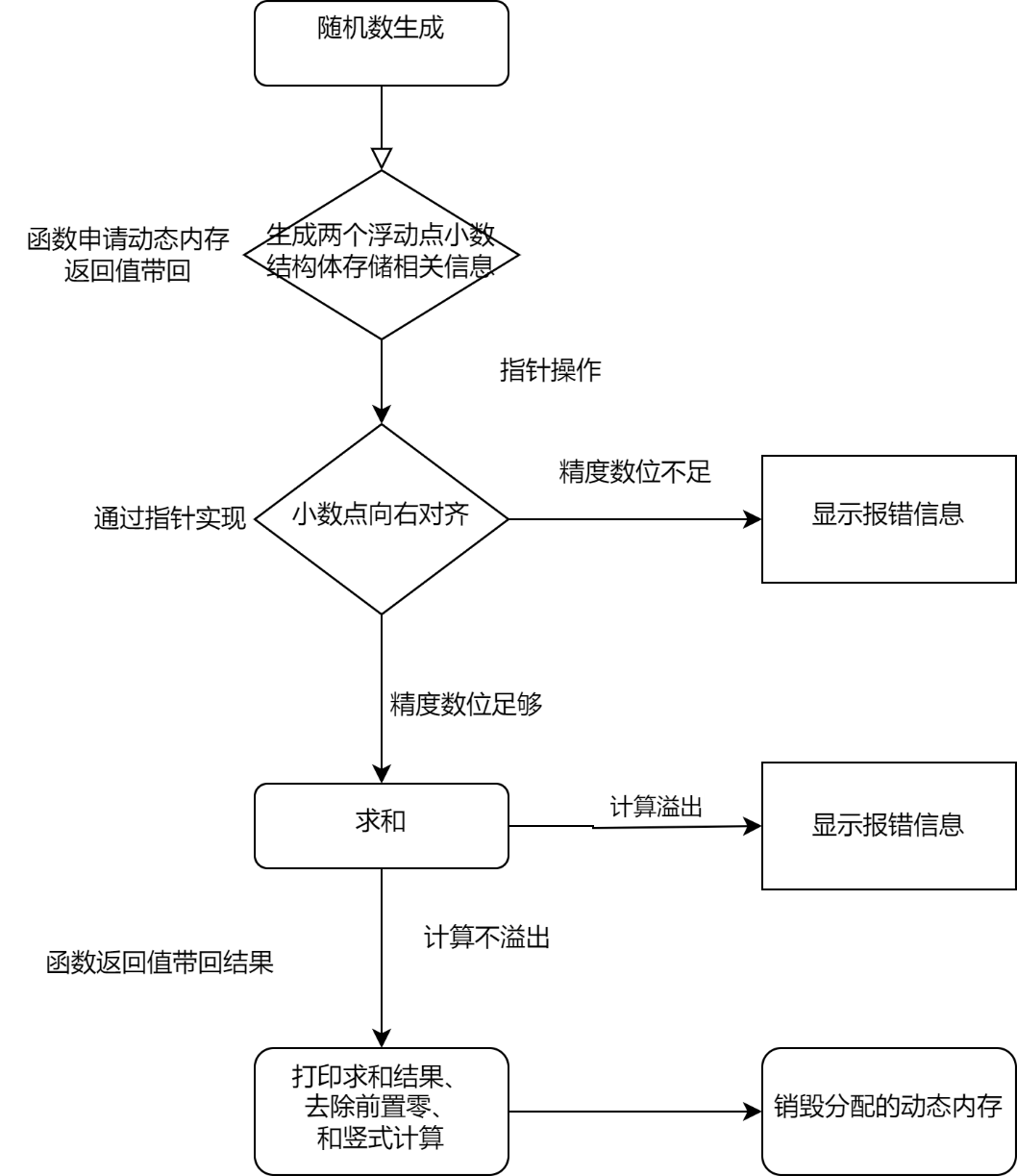
* 1. 与实验五相比，数据结构保持不变，通过指针实现基于引用的函数传递，修改相应的函数；
  2. 原来返回结构体的函数改为通过基于指针的基于引用的函数传递；
  3. 原来通过数组名实现基于引用传递的函数改写为通过指针来实现。

# 实验七：指针操作函数返回值

## 实验任务

修改函数，通过指针申请内存创建结构体对象，通过返回结构体指针带回

## 实验步骤

实验思路：  
关键代码：销毁函数，释放所申请的动态内存

void destoryLongReal(Real\* rPtr)

{

    free(rPtr);

    return;

}

关键代码：求和结果通过函数返回值带回

Real\* addTwoLongReals(const Real\* rPtr1, const Real\* rPtr2)

{

Real\* rPtrSum = (Real\*)malloc(sizeof(Real));

…… …… …… ……

return rPtrSum;

}

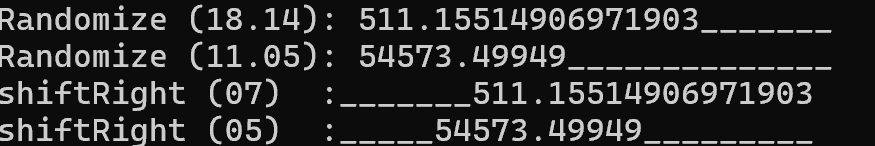
## 代码测试

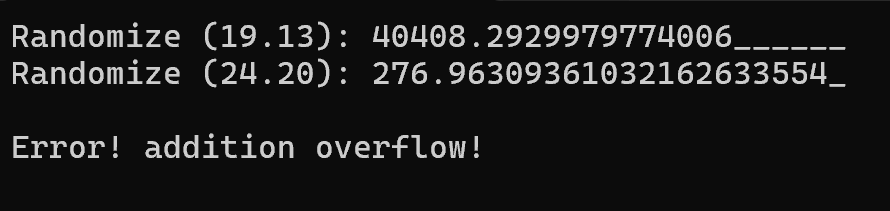
### shiftDigitsToRight测试点的测试结果

测试步骤：在addTwoLongReal里面调用该函数，观察输出结果

预期测试结果：数位精度足够时，两浮动点小数的小数点向右对齐；数位精度不够时显示报错信息

实际测试结果( 截图 )：





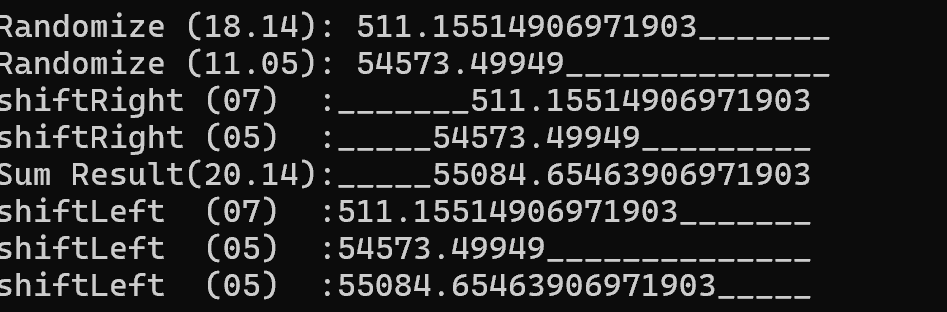
测试结论:测试通过

### addTwoLongReals测试点的测试结果

测试步骤:在main函数里调用该函数，是否通过结构体的指针实现基于引用的传递

预期测试结果：求和结果符合预期有溢出或精度不足会报错，函数返回值带回被动态内存创建的新对象

实际测试结果（截图）：



测试结论：测试通过

## 实验结论

代码达到功能目标

## 实验总结

* 1. 与实验七相比，数据结构保持不变，通过指针实现从函数返回结构体对象，修改部分函数
  2. 结构体对象由generateLongReal函数申请动态内存创建，通过指针返回值带回给main函数进行操作
  3. 函数返回值用于带回被动态内存创建的新对象
  4. 通过指针申请动态内存创建的结构体对象，无法自动销毁，需要通过代码释放动态内存

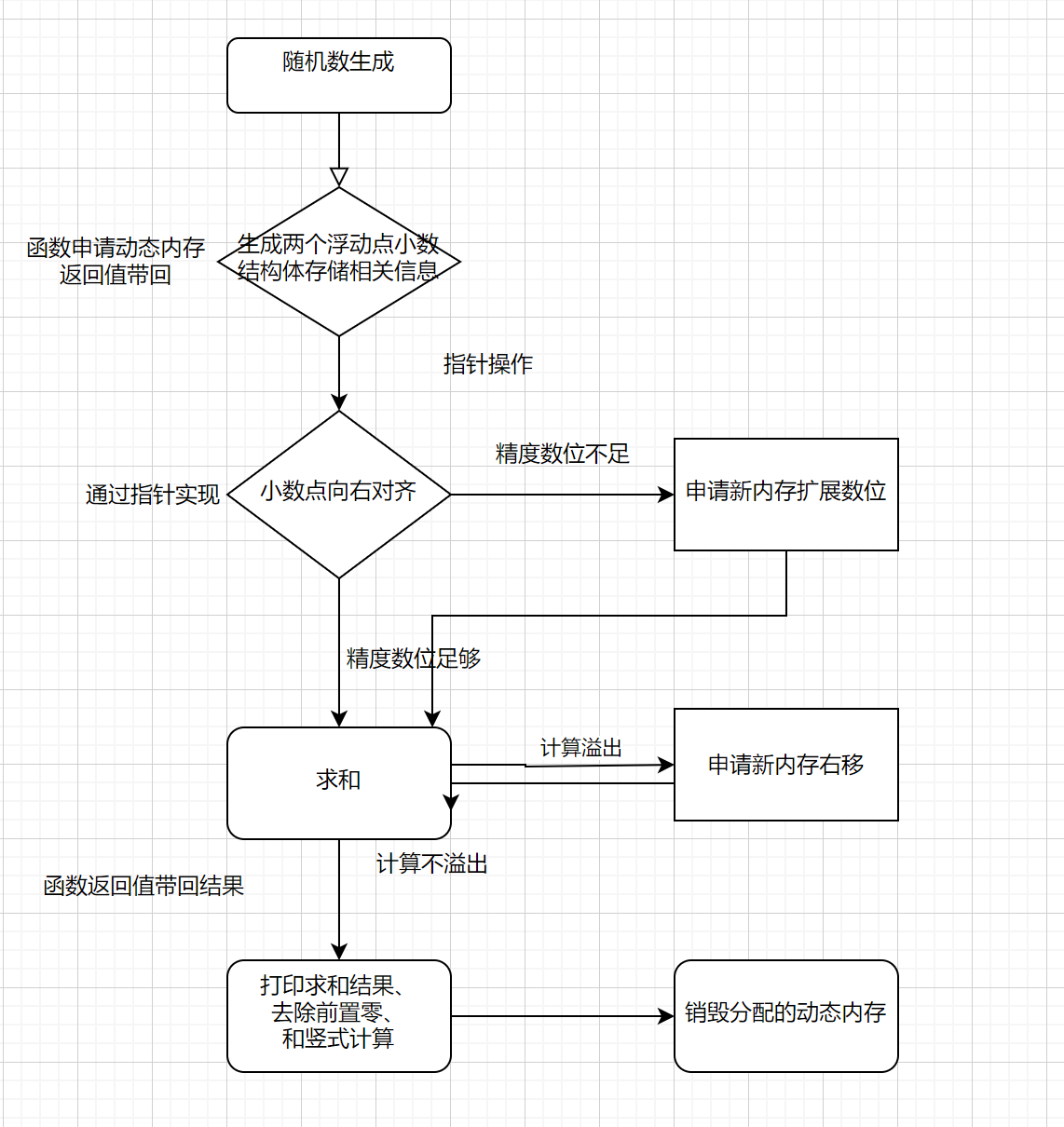
# 实验八：指针操作动态内存

## 实验任务

前一个实验中需要固定宽度数组的部分，改写为无最大位宽参数的动态数组

## 实验步骤

实验思路：



关键代码：向右对齐小数点

if( rPtr1->pointPos > rPtr2->pointPos )

    {

        shiftDigitsToRight( rPtr2, rPtr1->pointPos - rPtr2->pointPos );

        rPtrSum->pointPos = rPtr1->pointPos;

    }

    else if( rPtr1->pointPos < rPtr2->pointPos )

    {

        shiftDigitsToRight( rPtr1, rPtr2->pointPos - rPtr1->pointPos );

        rPtrSum->pointPos = rPtr2->pointPos;

    }

    else if( rPtr1->pointPos == rPtr2->pointPos )

    {

        rPtrSum->pointPos = rPtr1->pointPos;

    }

关键代码：求和

//从后向前逐位相加

    for(i = rPtrSum->length - 1 ; i >= 0 ; i--)

    {

        if( i== rPtrSum->pointPos - 1 )

        {

            rPtrSum->digits[i] = '.';

            continue;

        }

        else

        {

            sum = carry;

            if( rPtr1->digits[i] >= '0' && rPtr1->digits[i] <= '9' )

            {

                sum += rPtr1->digits[i] - '0';

            }

            if( rPtr2->digits[i] >= '0' && rPtr2->digits[i] <= '9' )

            {

                sum += rPtr2->digits[i] - '0';

            }

        }

        carry = sum / 10;

        rPtrSum->digits[i] = '0' + sum % 10;

    }

    if( carry > 0 )

    {

        ;

        //存储数位不足时向右移动申请新内存，支持拓展数位，不再存在溢出的问题

        rPtrSum->length++;

        rPtrSum->digits = (char\*)realloc(rPtrSum->digits, rPtrSum->length \* sizeof(char));

        shiftDigitsToRight( rPtrSum, 1 );

        rPtrSum->digits[0] = '0' + carry;

    }

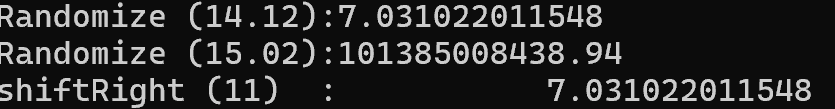
## 代码测试

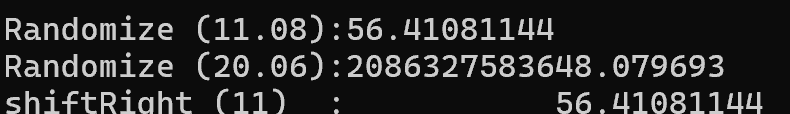
### shiftDigitsToRight测试点的测试结果

测试步骤：在addTwoLongReals里调用该函数使小数点对齐

预期测试结果：小数点位置靠前的向右与小数点位置靠后的对齐

实际测试结果（截图）：





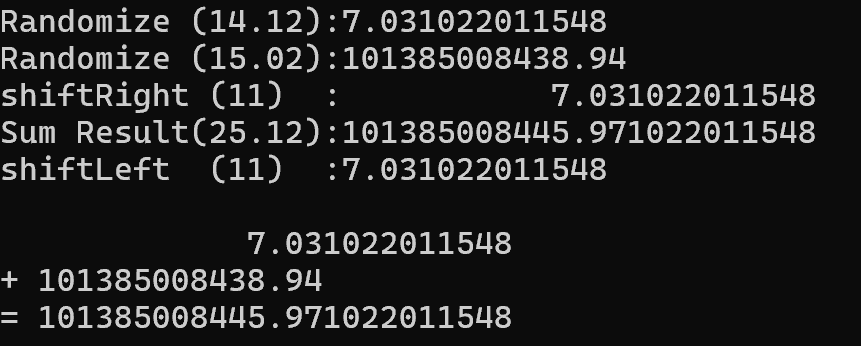
测试结论：测试通过

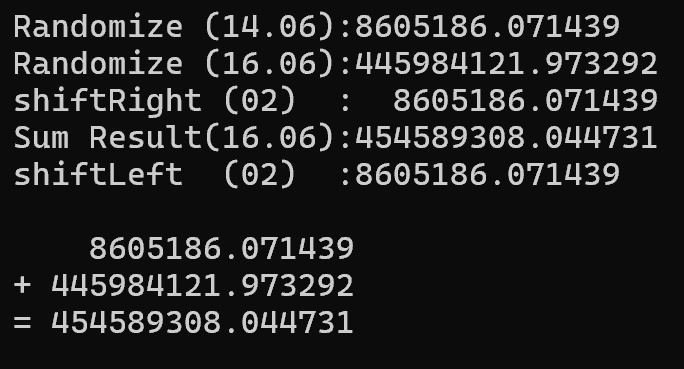
### addTwoLongReals测试点的测试结果

测试步骤：main函数调用，观察返回值是否是两小数相加结果

预期测试结果：返回值适量小数相加结果

实际测试结果（截图）：





测试结论：测试通过

## 实验结论

代码达到功能目标

## 实验总结

* + 1. 实验七中需要固定宽度数组的部分，改写为无最大位宽参数的动态数组
    2. 存储数位不足时向右移申请新内存，支持扩展数位，避免溢出的问题
    3. 注意到通过动态内存分配获得的结构体变量内存和结构体成员数组的内存不相同，要分别处理
    4. 左移和右移的函数不再是对固定内存区内不同数位的操作，而是直接调整动态内存区的大小和内容

# 本课程学习总结

1. 学习C语言的历程和总结（心酸血泪史）

一开始学条件语句时对于条件的判断等不太清楚，稍微懂了点后又被递归函数绕晕，紧接着就是数组各种细节和访问及运用，结构体顺势而上，刚开始没弄明白他到底是什么用处，逐渐清晰后又迎来了那个最最重要的东西——指针，无孔不入，数组结构体都有它的身影。最后还有动态内存，使用的可以称的上是一个乱用。只能打开企微看课件，慢慢琢磨各种细节。遇到大代码看懂以后觉得自己可以了，一到实操就完蛋。

二. 程序中出现的问题和改进

1、字符串操作时忘掉’\0’数组时又想着’\0’ 改进：铭记在心编完检查

2、指针使用时忘记初始化 改进：使用时注意正确初始化指针

3、排序函数基本只用效率最慢的冒泡 改进：勤背勤用，还可以自己去查找快速排序等等排序函数

三. 代码规范与调试技巧

1. 多打点空格代码不要ugly
2. 函数名，变量名等等要符合规范，不能自己瞎起
3. 注释也是代码的一部分，不能编爽了，之后再看又完蛋
4. 调试可以使用调试工具逐步执行代码，

四. 学习记录和心得

1、每次一编码就编到底了，应该要仔细琢磨题干要求后一步一步来

2、ai在c语言学习这块还是发挥着很大的作用的，真编不出来一点或怎么调试都报错的时候可以问问ai让他分析

3、一些比较普遍的代码可以啃啃分析，然后再自己上手

4、什么菜鸟教程什么CSDN什么github啊，有具体的知识点不会可以去上面翻翻多看看

# 附录

## 一、长整数相加

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <time.h>
5. #define INT\_WIDTH 20
6. // Generate a random long integer
7. **void** generateLongInt( **char** longInt[ ] );
8. //基于引用的传递，由该函数为字符数组longInt[]每个数位产生随机数
10. // Sum two long integers
11. **int** addTwoLongInts( **char** longInt1[ ], **char** longInt2[ ], **char** longIntSum[ ] );
12. //基于引用的传递，由longInt1[]和longInt2[]为加数，longSum[]存放求和结果，返回值int标记是否计算溢出
14. // Display the long integer in a single line
15. **void** displayLongInt( **char** longInt[ ] );
17. **int** main(**void**)
18. {
19. srand( time(NULL) );
20. **char** longInt1[ INT\_WIDTH ] = { 0 };
21. generateLongInt(longInt1);
22. **char** longInt2[ INT\_WIDTH ] = { 0 };
23. generateLongInt(longInt2);
25. **char** longIntSum[ INT\_WIDTH ] = { 0 };
26. **int** isOverflow = addTwoLongInts( longInt1, longInt2, longIntSum );
27. //该函数返回值表达是否计算溢出
29. **if** ( isOverflow == 0 )
30. {
31. printf("\n  ");
32. displayLongInt( longInt1 );
33. printf("+ ");
34. displayLongInt( longInt2 );
35. printf("= ");
36. displayLongInt( longIntSum );
37. }
38. //如果不溢出，则由main函数控制输出竖式计算过程
40. **else** **if** ( isOverflow == 1 )
41. {
42. printf("\nError! addition overflow!\n");
43. }
44. //如果溢出，则显示报错信息
46. printf("\n");
47. **return** 0;
48. }
50. **void** generateLongInt( **char** longInt[] )
51. {
52. **int** i;
53. longInt[0] = '1' + rand() % 9;
54. **for** ( i = 1; i < INT\_WIDTH; i++ )
55. {
56. longInt[i] += '0' + rand() % 10;
57. }
59. printf( "Generated (%d digit): ", INT\_WIDTH);
60. displayLongInt( longInt );
62. **return**;
63. }
65. **int** addTwoLongInts( **char** longInt1[], **char** longInt2[], **char** longIntSum[] )
66. {
67. **int** add = 0;
68. **for** (**int** i = INT\_WIDTH - 1; i >= 0; i--)
69. {
70. **int** digit\_1 = longInt1[i] - '0';
71. **int** digit\_2 = longInt2[i] - '0';
72. **int** sum = digit\_1 + digit\_2 + add;
73. **if** (sum > 9)
74. {
75. add = 1;
76. sum -= 10;
77. }
78. **else**
79. {
80. add = 0;
81. }
82. longIntSum[i] = sum + '0';
83. }
84. **return** add;
85. }
87. **void** displayLongInt( **char** longInt[] )
88. {
89. **int** i;
90. **for** ( i = 0; i < INT\_WIDTH; i++ )
91. {
92. printf( "%c", longInt[i] );
93. }
94. printf("\n");
96. **return**;
97. }

## 二、长实数相加

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <time.h>
5. #define real\_width 25 // the width of real number includes the point/dot
6. #define point\_position 12
8. // generate a random long real number
9. **void** generatelongreal( **char** longreal[] );
10. // 基于引用的传递，由该函数为字符数组 longint[]每个数位产生随机数
12. // sum two long real numbers
13. **int** addtwolongreals( **char** longreal1[], **char** longreal2[], **char** longrealsum[]);
14. //基于引用的传递，由longlnt1[]和longlnt2[]为加数，longsum[]存放求和结果，返回值int标记是否计算溢出
16. // display the long integer in a single line
17. **void** displaylongreal( **char** longreal[] );
19. **int** main( **void** )
20. {
21. srand(time(null));
22. **char** longreal1[ real\_width ] = { 0 };
23. generatelongreal( longreal1 );
24. **char** longreal2[real\_width] = { 0 };
25. generatelongreal( longreal2 );
27. **char** longrealsum[real\_width] = { 0 };
28. //该函数返回值表达是否计算溢出
29. **int** isoverflow = addtwolongreals(longreal1, longreal2, longrealsum);
31. //如果不溢出，则由main函数控制输出竖式计算过程
32. **if** (isoverflow == 0)
33. {
34. printf("\n  ");
35. displaylongreal(longreal1);
36. printf("+ ");
37. displaylongreal(longreal2);
38. printf("= ");
39. displaylongreal(longrealsum);
40. }
41. //如果溢出，则显示报错信息
42. **else** **if** (isoverflow == 1)
43. {
44. printf("\nerror! addition overflow!\n");
45. }
47. printf("\n");
48. **return** 0;
50. }
51. **void** generatelongreal( **char** longreal[] )
52. {
53. **int** i;
54. longreal[0] = '1' + rand() % 9;
55. **for** (i = 1; i < real\_width; i++)
56. {
57. longreal[i] = '0' + rand() % 10;
58. }
59. longreal[point\_position - 1] = '.';
60. printf("generated (%d.%d): ", real\_width, real\_width - point\_position);
61. displaylongreal( longreal );
63. **return**;
64. }
66. **int** addtwolongreals(**char** longreal1[], **char** longreal2[], **char** longrealsum[])
67. {
68. **int** add = 0;
69. **for** (**int** i = real\_width - 1; i >= 0; i--)
70. {
72. **if** (i == point\_position - 1)
73. {
74. longrealsum[i] = '.';
75. }
76. **else**
77. {
78. **int** dig1 = longreal1[i] - '0';
79. **int** dig2 = longreal2[i] - '0';
80. **int** sum = dig1 + dig2 + add;
81. **if** (sum > 9)
82. {
83. add = 1;
84. sum -= 10;
85. }
86. **else**
87. {
88. add = 0;
89. }
90. longrealsum[i] = sum + '0';
91. }
92. }
93. **return** add;
94. }
96. **void** displaylongreal(**char** longreal[])
97. {
98. **int** i;
99. **for** (i = 0; i < real\_width; i++)
100. {
101. printf("%c", longreal[i]);
102. }
103. printf("\n");
105. **return**;
107. }

## 三、固定小数点相加

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <time.h>
5. #define MAX\_REAL\_WIDTH 25
6. #define POINT\_POSITION 12
8. **typedef** **struct**{
9. **char** digits[ MAX\_REAL\_WIDTH ];
10. **int** length;
11. } Real;
13. // Generate a random long real number
14. //修改生成函数的设计，将结构体作为返回值带回
15. Real generateLongReal( **void**);
17. // Sum two long real numbers
18. //修改计算函数的设计，将计算结果结构体作为返回值带回
19. Real addTwoLongReals( Real r1,Real r2);
21. // Display the real number in a single line
22. //新增一个函数，支持对空闲数位的打印显示
23. **void** displayLongReal( Real r);
25. // Display the number in a fixed width
26. **void** displayFixedwidthNumber( **char** number[], **const** **int** width );
28. **int** main( **void** )
29. {
30. srand( time(NULL));
31. Real real1 = generateLongReal();
32. Real real2= generateLongReal();
33. Real realSum=addTwoLongReals( real1,real2 );
35. //判断结构体变量是否有效
36. **if**( realSum.length > 0 )
37. {   //如果不溢出，输出竖式计算过程
38. printf("\n  ");
39. displayLongReal( real1 );
40. printf("+ ");
41. displayLongReal( real2 );
42. printf("= ");
43. displayLongReal( realSum );
44. }
45. **else**
46. {   //如果溢出，则显示报错信息
47. printf("\nError! addition overflow!\n");
48. }
49. printf("\n");
50. **return** 0;
51. }
53. Real generateLongReal( **void** )
54. {
55. Real r;
56. **int** fractionLength = rand() % ( MAX\_REAL\_WIDTH - POINT\_POSITION + 1 );
57. r.length = POINT\_POSITION + fractionLength;
59. **int** i;
60. r.digits[0] = '1' + rand() % 9;
61. **for**( i = 1; i < r.length - 1; i++ )
62. {
63. r.digits[i] = '0' + rand() % 10;
64. }
65. r.digits[ r.length - 1 ] = '1' + rand() % 9;
66. r.digits[ POINT\_POSITION - 1 ] = '.';
67. **for**( i = r.length; i < MAX\_REAL\_WIDTH; i++ )
68. {   // clear the rest of the memory
69. //生成随机数时，没有用到的数位填充'\0';
70. //显示正常数值时，无需打印；显示全部数宽的时候，空闲数位用字符'\_'代替
71. r.digits[i] = '\0';
72. }
73. r.digits[ MAX\_REAL\_WIDTH - 1 ] = '\0';
74. printf("Generated (%2d.%02d):", r.length, r.length - POINT\_POSITION );
75. displayFixedwidthNumber( r.digits, MAX\_REAL\_WIDTH );
77. **return** r;
78. }
80. **void** displayLongReal( Real r)
81. {
82. **int** i;
83. **for**(i=0; i<r.length; i ++ )
84. {
85. printf("%c", r.digits[i]);
86. }
87. printf("\n");
88. **return**;
89. }
91. **void** displayFixedwidthNumber( **char** number[], **const** **int** width )
92. {
93. **int** i;
94. **for**(i=0; i< width; i ++ )
95. {
96. **if**( number[i] == '\0' )
97. {
98. printf("\_");
99. }
100. **else**
101. {
102. printf("%c", number[i]);
103. }
104. }
105. printf("\n");
106. **return**;
107. }
109. Real addTwoLongReals( Real r1,Real r2)
110. {
111. Real r;
112. **int** i;
113. **int** carry = 0;
114. **int** sum = 0;
115. **int** maxLen = ( r1.length > r2.length ) ? r1.length : r2.length ;
117. //判断是否溢出
118. **if**( maxLen >= MAX\_REAL\_WIDTH )
119. {
120. r.length = 0;
121. **return** r;
122. }
124. **for**(i = maxLen - 1 ; i >= 0; i--)
125. {
126. **if**( r1.digits[i] == '.' )
127. {
128. r.digits[i] = '.';
129. **continue**;
130. }
131. **else**
132. {
133. sum = carry;
134. **if**(r1.digits[i] != '\0' )
135. {
136. sum += r1.digits[i] - '0';
137. }
138. **if**( r2.digits[i] != '\0' )
139. {
140. sum += r2.digits[i] - '0';
141. }
142. }
143. carry = sum / 10;
144. r.digits[i] = '0' + sum % 10;
146. }
148. **if**( carry > 0 )
149. {
150. //溢出判断
151. r.length = 0 ;
152. **return** r;
153. }
154. **else**
155. {
156. r.length = maxLen;
157. }
159. **for**( i=r.length; i < MAX\_REAL\_WIDTH ; i++ )
160. {   // clear the rest of the memory
161. //生成随机数时，没有用到的数位填充'\0';
162. //显示正常数值时，无需打印；显示全部数宽的时候，空闲数位用字符'\_'代替
163. r.digits[i] = '\0';
164. }
165. r.digits[ MAX\_REAL\_WIDTH - 1 ] = '\0';
166. printf("Sum Result(%2d.%02d):", r.length, r.length - POINT\_POSITION );
167. displayFixedwidthNumber( r.digits, MAX\_REAL\_WIDTH );
168. **return** r;
169. }

## 四、浮动点小数相加

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <time.h>
5. #define MAX\_REAL\_WIDTH 25
7. **typedef** **struct**{
8. **char** digits[ MAX\_REAL\_WIDTH ];
9. **int** pointPos;
10. **int** length;
11. } Real;
13. // Generate a random long real number
14. Real generateLongReal( **void**);
16. // Sum two long real numbers
17. Real addTwoLongReals( Real r1,Real r2);
19. //为支持浮动小数点的计算和显示，新增若干函数
20. // Display the real number in a single line
21. **void** displayAssignedLongReal( **int** pointPos,Real r);
23. // Shift the digits to right direction
24. **void** shiftDigitsToRight( **char** number[],**int** shiftLength);
26. // Shift the digits to left direction
27. **void** shiftDigitsToLeft( **char** number[],**int** shiftLength);
29. // Display the number in a fixed width
30. **void** displayFixedwidthNumber( **char** number[], **const** **int** width );
32. **int** main( **void** )
33. {
34. srand( time(NULL));
35. Real real1 = generateLongReal();
36. Real real2 = generateLongReal();
37. Real realSum = addTwoLongReals( real1,real2 );
39. //判断结构体变量是否有效
40. **if**( realSum.length > 0 )
41. {   //如果不溢出，输出竖式计算过程
42. printf("\n  ");
43. displayAssignedLongReal( realSum.pointPos,real1 );
44. printf("+ ");
45. displayAssignedLongReal( realSum.pointPos,real2 );
46. printf("= ");
47. displayAssignedLongReal( realSum.pointPos,realSum );
48. }
49. **else**
50. {   //如果溢出，则显示报错信息
51. printf("\nError! addition overflow!\n");
52. }
53. printf("\n");
54. **return** 0;
55. }
57. **void** displayAssignedLongReal(**int** pointPos,Real r)
58. {
59. **int** i;
60. **for**( i=0 ; i < pointPos-r.pointPos ; i++ )
61. {
62. printf(" ");
63. }
64. **for**( i=0 ; i < r.length ; i ++ )
65. {
66. printf("%c",r.digits[i]);
67. }
68. printf("\n");
70. **return**;
71. }
73. **void** shiftDigitsToRight( **char** number[],**int** shiftLength)
74. {
75. **int** i;
76. **for**( i = MAX\_REAL\_WIDTH - 1; i >= shiftLength; i-- )
77. {
78. number[i] = number[i-shiftLength];
79. }
80. **for**( i = 0; i < shiftLength; i++ )
81. {
82. number[i] = '\0';
83. }
84. printf("shiftRight  (%02d)  :",shiftLength);
85. displayFixedwidthNumber( number, MAX\_REAL\_WIDTH );
86. **return**;
87. }
89. **void** shiftDigitsToLeft( **char** number[],**int** shiftLength)
90. {
91. **int** i;
92. **for**( i = 0; i < MAX\_REAL\_WIDTH - shiftLength; i++ )
93. {
94. number[i] = number[i+shiftLength];
95. }
96. **for**( i = MAX\_REAL\_WIDTH - shiftLength; i < MAX\_REAL\_WIDTH; i++ )
97. {
98. number[i] = '\0';
99. }
100. printf("shiftLeft   (%02d)  :",shiftLength);
101. displayFixedwidthNumber( number, MAX\_REAL\_WIDTH );
102. **return**;
103. }
105. Real addTwoLongReals( Real r1,Real r2)
106. {
107. Real r;
108. **int** i;
109. **int** carry = 0;
110. **int** sum = 0;
112. //向右对齐
113. **int** r1right = MAX\_REAL\_WIDTH - r1.length ;
114. **int** r2right = MAX\_REAL\_WIDTH - r2.length ;
115. **int** shiftLength = (r1right < r2right) ? r1right : r2right ;
117. //将两个数的小数点向右移动对齐并求和
118. **if**( r1right < r2right )
119. {    //判断表达精度的数位是否足够
120. //是否在移动过程中向右溢出
121. **if**(r2.length+shiftLength +r1.pointPos-r2.pointPos > MAX\_REAL\_WIDTH
122. //是否向左溢出
123. ||r2.pointPos - r1.pointPos > shiftLength )
124. {
125. r.length = 0;
126. **return** r;
127. }
128. **else**
129. {
130. shiftDigitsToRight( r1.digits, shiftLength );
131. shiftDigitsToRight( r2.digits, shiftLength + r1.pointPos-r2.pointPos );
132. }
133. }
134. **else** **if**( r1right > r2right )
135. {
136. **if**(r1.length+shiftLength + r2.pointPos-r1.pointPos > MAX\_REAL\_WIDTH
137. || r1.pointPos - r2.pointPos > shiftLength )
138. {
139. r.length = 0;
140. **return** r;
141. }
142. **else**
143. {
144. shiftDigitsToRight( r1.digits, shiftLength + r2.pointPos-r1.pointPos );
145. shiftDigitsToRight( r2.digits, shiftLength  );
146. }
147. }
148. **else** **if**( r1right == r2right )
149. {
150. **if**(r1.pointPos!=r2.pointPos)
151. {
152. r.length = 0;
153. **return** r;
154. }
155. **else**
156. {
157. shiftDigitsToRight( r1.digits, r1right + abs(r2.pointPos-r1.pointPos) );
158. shiftDigitsToRight( r2.digits, r2right + abs(r2.pointPos-r1.pointPos) );
159. }
160. }

163. //判断是否溢出
164. **if**( r1.length > MAX\_REAL\_WIDTH || r2.length > MAX\_REAL\_WIDTH )
165. {
166. r.length = 0;
167. **return** r;
168. }
170. //求向右对齐后长实数
171. **int** r1left = 0,r2left = 0;
172. **while**(r1.digits[r1left] =='\0')
173. {
174. r1left++;
175. }
176. **while**(r2.digits[r2left] =='\0')
177. {
178. r2left++;
179. }
180. **int** rleft = (r1left < r2left) ? r1left : r2left ;
181. r.length = MAX\_REAL\_WIDTH - rleft ;
183. //求和
184. **for**(i = MAX\_REAL\_WIDTH - 1 ; i >= 0 ; i--)
185. {
186. **if**( r1.digits[i] == '.' )
187. {
188. r.digits[i] = '.';
189. r.pointPos = r.length - (MAX\_REAL\_WIDTH - ( i + 1 ) );
190. **continue**;
191. }
192. **else**
193. {
194. sum = carry;
195. **if**(r1.digits[i] != '\0' )
196. {
197. sum += r1.digits[i] - '0';
198. }
199. **if**( r2.digits[i] != '\0' )
200. {
201. sum += r2.digits[i] - '0';
202. }
203. }
204. carry = sum / 10;
205. r.digits[i] = '0' + sum % 10;
206. }
207. **if**( carry > 0 )
208. {
209. //加法运算有进位，计算溢出
210. r.length = 0 ;
211. **return** r;
212. }
214. **for**( i = MAX\_REAL\_WIDTH - r.length - 1; i >=0 ; i-- )
215. {   // clear the rest of the memory
216. //生成随机数时，没有用到的数位填充'\0';
217. //显示正常数值时，无需打印；显示全部数宽的时候，空闲数位用字符'\_'代替
218. r.digits[i] = '\0';
219. }
221. printf("Sum Result (%2d.%02d):", r.length, r.length - r.pointPos );
222. displayFixedwidthNumber( r.digits, MAX\_REAL\_WIDTH );
224. //最后将三个数的小数点向左移动，去除前置的0
225. shiftDigitsToLeft( r1.digits, r1left  );
226. shiftDigitsToLeft( r2.digits, r2left );
227. shiftDigitsToLeft( r.digits, rleft );
229. **return** r;
230. }
232. **void** displayFixedwidthNumber( **char** number[], **const** **int** width )
233. {
234. **int** i;
235. **for**( i = 0; i < width; i++ )
236. {
237. **if**( number[i] == '\0' )
238. {
239. printf("\_");
240. }
241. **else**
242. {
243. printf("%c", number[i]);
244. }
245. }
246. printf("\n");
247. **return**;
248. }
250. Real generateLongReal( **void** )
251. {
252. Real r;
253. **int** i;
254. **int** len = rand() % ( MAX\_REAL\_WIDTH + 1 );
255. r.pointPos = rand() % len  + 1 ;
256. r.length = len;
258. **for**( i = 0; i < len ; i++ )
259. {
260. **char** digit = rand() % 10;
261. **if**( i == r.pointPos-1 )
262. {
263. r.digits[i] = '.';
264. }
265. **else**
266. {
267. r.digits[i] = '0' + digit;
268. }
269. }
270. r.digits[0] = '1' + rand() % 9;
271. r.digits[ r.length - 1 ] = '1' + rand() % 9;
272. **if**( r.pointPos == 1 )
273. {
274. r.digits[0] = '.';
275. }
277. **if**( r.pointPos == r.length )
278. {
279. r.digits[ r.length - 1 ] = '.';
280. }
282. **for**( i = len; i < MAX\_REAL\_WIDTH; i++ )
283. {
284. r.digits[i] = '\0';
285. }
287. r.digits[ MAX\_REAL\_WIDTH - 1 ] = '\0';
288. printf("Generated  (%2d.%02d):", r.length, r.length - r.pointPos );
289. displayFixedwidthNumber( r.digits, MAX\_REAL\_WIDTH );
291. **return** r;
292. }

## 五、指针操作数组元素

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <time.h>
5. #define MAX\_REAL\_WIDTH 25
7. **typedef** **struct**{
8. **char** digits[ MAX\_REAL\_WIDTH ];
9. **int** pointPos;
10. **int** length;
11. } Real;
13. //需要用指针法改写的函数，通过指针操作数组元素
14. // Generate a random long real number
15. Real generateLongReal( **void**);
17. // Sum two long real numbers
18. Real addTwoLongReals( Real r1,Real r2);
20. //需要用指针法改写的函数，通过指针操作数组元素
21. // Display the real number in a single line
22. **void** displayAssignedLongReal( **int** pointPos,Real r);
24. // Shift the digits to right direction
25. **void** shiftDigitsToRight( **char** number[],**int** shiftLength);
27. // Shift the digits to left direction
28. **void** shiftDigitsToLeft( **char** number[],**int** shiftLength);
30. // Display the number in a fixed width
31. **void** displayFixedwidthNumber( **char** number[], **const** **int** width );
33. **int** main( **void** )
34. {
35. srand( time(NULL));
36. Real real1 = generateLongReal();
37. Real real2 = generateLongReal();
38. Real realSum = addTwoLongReals( real1,real2 );
40. //判断结构体变量是否有效
41. **if**( realSum.length > 0 )
42. {   //如果不溢出，输出竖式计算过程
43. printf("\n  ");
44. displayAssignedLongReal( realSum.pointPos,real1 );
45. printf("+ ");
46. displayAssignedLongReal( realSum.pointPos,real2 );
47. printf("= ");
48. displayAssignedLongReal( realSum.pointPos,realSum );
49. }
50. **else**
51. {   //如果溢出，则显示报错信息
52. printf("\nError! addition overflow!\n");
53. }
54. printf("\n");
55. **return** 0;
56. }
58. **void** displayAssignedLongReal(**int** pointPos,Real r)
59. {
60. **int** i;
61. **for**( i=0 ; i < pointPos-r.pointPos ; i++ )
62. {
63. printf(" ");
64. }
65. **char** \*cPtr = NULL ;
66. **for**( cPtr = r.digits ; cPtr < r.digits + r.length ; cPtr ++ )
67. {
68. printf("%c",\*cPtr);
69. }
70. printf("\n");
72. **return**;
73. }
75. **void** shiftDigitsToRight( **char** number[],**int** shiftLength)
76. {
77. **char** \*end = number + MAX\_REAL\_WIDTH - 1;
78. **char** \*start = number + MAX\_REAL\_WIDTH - 1 - shiftLength;
79. **while**( start >= number )
80. {
81. \*end = \*start;
82. end--;
83. start--;
84. }
85. **while**( end >= number )
86. {
87. \*end = '\0';
88. end--;
89. }
91. printf("shiftRight  (%02d)  :",shiftLength);
92. displayFixedwidthNumber( number, MAX\_REAL\_WIDTH );
93. **return**;
94. }
96. **void** shiftDigitsToLeft( **char** number[],**int** shiftLength)
97. {
98. **char** \*start = number;
99. **char** \*end = number + shiftLength;
100. **while**( end < number + MAX\_REAL\_WIDTH )
101. {
102. \*start = \*end;
103. start++;
104. end++;
105. }
106. **while**( start < number + MAX\_REAL\_WIDTH )
107. {
108. \*start = '\0';
109. start++;
110. }
112. printf("shiftLeft   (%02d)  :",shiftLength);
113. displayFixedwidthNumber( number, MAX\_REAL\_WIDTH );
114. **return**;
115. }
117. Real addTwoLongReals( Real r1,Real r2)
118. {
119. Real r;
120. **int** i;
121. **int** carry = 0;
122. **int** sum = 0;
124. //向右对齐
125. **int** r1right = MAX\_REAL\_WIDTH - r1.length ;
126. **int** r2right = MAX\_REAL\_WIDTH - r2.length ;
127. **int** shiftLength = (r1right < r2right) ? r1right : r2right ;
129. //将两个数的小数点向右移动对齐并求和
130. **if**( r1right < r2right )
131. {    //判断表达精度的数位是否足够
132. //是否在移动过程中向右溢出
133. **if**(r2.length+shiftLength + r1.pointPos-r2.pointPos > MAX\_REAL\_WIDTH
134. //是否向左溢出
135. ||r2.length-r1.length > shiftLength )
136. {
137. r.length = 0;
138. **return** r;
139. }
140. **else**
141. {
142. shiftDigitsToRight( r1.digits, shiftLength );
143. shiftDigitsToRight( r2.digits, shiftLength + r1.pointPos-r2.pointPos );
144. }
145. }
146. **else** **if**( r1right > r2right )
147. {
148. **if**(r1.length+shiftLength + r2.pointPos-r1.pointPos > MAX\_REAL\_WIDTH
149. || r1.length-r2.length > shiftLength )
150. {
151. r.length = 0;
152. **return** r;
153. }
154. **else**
155. {
156. shiftDigitsToRight( r1.digits, shiftLength + r2.pointPos-r1.pointPos );
157. shiftDigitsToRight( r2.digits, shiftLength  );
158. }
159. }
160. **else** **if**( r1right == r2right )
161. {
162. **if**(r1.pointPos!=r2.pointPos)
163. {
164. r.length = 0;
165. **return** r;
166. }
167. **else**
168. {
169. shiftDigitsToRight( r1.digits, r1right + abs(r2.pointPos-r1.pointPos) );
170. shiftDigitsToRight( r2.digits, r2right + abs(r2.pointPos-r1.pointPos) );
171. }
172. }

175. //判断是否溢出
176. **if**( r1.length > MAX\_REAL\_WIDTH || r2.length > MAX\_REAL\_WIDTH )
177. {
178. r.length = 0;
179. **return** r;
180. }
182. //求向右对齐后长实数
183. **int** r1left = 0,r2left = 0;
184. **while**(r1.digits[r1left] =='\0')
185. {
186. r1left++;
187. }
188. **while**(r2.digits[r2left] =='\0')
189. {
190. r2left++;
191. }
192. **int** rleft = (r1left < r2left) ? r1left : r2left ;
193. r.length = MAX\_REAL\_WIDTH - rleft ;
195. //求和
196. **for**(i = MAX\_REAL\_WIDTH - 1 ; i >= 0 ; i--)
197. {
198. **if**( r1.digits[i] == '.' )
199. {
200. r.digits[i] = '.';
201. r.pointPos = r.length - (MAX\_REAL\_WIDTH - ( i + 1 ) );
202. **continue**;
203. }
204. **else**
205. {
206. sum = carry;
207. **if**(r1.digits[i] != '\0' )
208. {
209. sum += r1.digits[i] - '0';
210. }
211. **if**( r2.digits[i] != '\0' )
212. {
213. sum += r2.digits[i] - '0';
214. }
215. }
216. carry = sum / 10;
217. r.digits[i] = '0' + sum % 10;
218. }
219. **if**( carry > 0 )
220. {
221. //加法运算有进位，计算溢出
222. r.length = 0 ;
223. **return** r;
224. }
226. **for**( i = MAX\_REAL\_WIDTH - r.length - 1; i >=0 ; i-- )
227. {   // clear the rest of the memory
228. //生成随机数时，没有用到的数位填充'\0';
229. //显示正常数值时，无需打印；显示全部数宽的时候，空闲数位用字符'\_'代替
230. r.digits[i] = '\0';
231. }
233. printf("Sum Result (%2d.%02d):", r.length, r.length - r.pointPos );
234. displayFixedwidthNumber( r.digits, MAX\_REAL\_WIDTH );
236. //最后将三个数的小数点向左移动，去除前置的0
237. shiftDigitsToLeft( r1.digits, r1left  );
238. shiftDigitsToLeft( r2.digits, r2left );
239. shiftDigitsToLeft( r.digits, rleft );
241. **return** r;
242. }
244. **void** displayFixedwidthNumber( **char** number[], **const** **int** width )
245. {
246. **char** \*cPtr = number;
247. **int** i;
248. **for**( i = 0; i < width; i++ )
249. {
250. **if**( \*(cPtr + i) == '\0' )
251. {
252. printf("\_");
253. }
254. **else**
255. {
256. printf("%c", \*(cPtr + i) );
257. }
258. }
259. printf("\n");
260. **return**;
261. }
263. Real generateLongReal( **void** )
264. {
265. Real r;
266. **int** i;
267. **int** len = rand() % ( MAX\_REAL\_WIDTH + 1 );
268. r.pointPos = rand() % len  + 1 ;
269. r.length = len;
271. **char** \*cPtr = r.digits;
272. **for**( i = 0; i < len ; i++ )
273. {
274. **char** digit = rand() % 10;
275. **if**( i == r.pointPos-1 )
276. {
277. \*(cPtr + i) = '.';
278. }
279. **else**
280. {
281. \*(cPtr + i) = '0' + digit;
282. }
283. }
284. \*(cPtr + 0) = '1' + rand() % 9;
285. \*(cPtr + r.length - 1 ) = '1' + rand() % 9;
287. **if**( r.pointPos == 1 )
288. {
289. \*(cPtr + 0) = '.';
290. }
292. **if**( r.pointPos == r.length )
293. {
294. \*(cPtr + r.length - 1 ) = '.';
295. }
297. **for**( i = len; i < MAX\_REAL\_WIDTH; i++ )
298. {
299. \*(cPtr + i) = '\0';
300. }
302. \*(cPtr + MAX\_REAL\_WIDTH - 1 ) = '\0';
303. printf("Generated  (%2d.%02d):", r.length, r.length - r.pointPos );
304. displayFixedwidthNumber( r.digits, MAX\_REAL\_WIDTH );
306. **return** r;
307. }

## 六、指针操作数组和结构体

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <time.h>
5. #define MAX\_REAL\_WIDTH 25
7. **typedef** **struct** {
8. **char** digits[MAX\_REAL\_WIDTH];
9. **int** pointPos;
10. **int** length;
11. } Real;
13. //通过指针实现基于引用的函数传递，修改相应的函数
15. //原来返回结构体的函数改为，通过结构体指针的基于引用的函数传递
16. **void** generateLongReal( Real\* rPtr );
18. **void** addTwoLongReals( **const** Real\* rPtr1, **const** Real\* rPtr2, Real\* rPtrSum );
20. **void** displayAssignedLongReal( **int** pointPos, Real\* rPtr );
22. //原来通过数组名实现基于引用传递的函数，改为通过指针实现
23. **void** shiftDigitsToRight(**const** **char**\* longNumber, **int** shiftLength);
25. **void** shiftDigitsToLeft(**const** **char**\* longNumber, **int** shiftLength);
27. **void** displayFixedWidthNumber(**const** **char**\* number, **const** **int** width);
29. **int** main(**void**)
30. {
31. srand(time(NULL));
32. Real r1, r2, sum;
33. Real\* rPtr1 = &r1, \* rPtr2 = &r2, \* rPtrSum = ∑
35. //通过结构体指针rPtrSum实现基于引用的传递，相应的内存区sum对象被函数直接改写，无需通过函数返回值带回，节省了内存复制过程
36. generateLongReal(rPtr1);
37. generateLongReal(rPtr2);
38. addTwoLongReals(rPtr1, rPtr2, rPtrSum);
40. **if** (rPtrSum->length > 0)
41. {
42. printf("\n  ");
43. displayAssignedLongReal(rPtrSum->pointPos, rPtr1);
44. printf("+ ");
45. displayAssignedLongReal(rPtrSum->pointPos, rPtr2);
46. printf("= ");
47. displayAssignedLongReal(rPtrSum->pointPos, rPtrSum);
48. }
49. **else**
50. {
51. printf("\nError! addition overflow!\n");
52. }
53. printf("\n");
54. }
56. //通过结构体指针来操作结构体变量的成员
57. **void** generateLongReal(Real\* rPtr)
58. {
59. rPtr->length = 2 + rand() % (MAX\_REAL\_WIDTH - 1);
60. rPtr->pointPos = 1 + rand() % (rPtr->length);
62. **char**\* cPtr = rPtr->digits;
63. \*cPtr = '1' + rand() % 9;
64. **for** (cPtr++; cPtr < rPtr->digits + rPtr->length - 1; cPtr++)
65. {
66. \*cPtr = '0' + rand() % 10;
67. }
68. **for** (; cPtr < rPtr->digits + MAX\_REAL\_WIDTH; cPtr++)
69. {
70. // clear the rest of the memory
71. \*cPtr = '\0';
72. }
74. rPtr->digits[rPtr->length - 1] = '1' + rand() % 9;
75. rPtr->digits[rPtr->pointPos - 1] = '.';
77. printf("Generated(%2d.%02d): ", rPtr->length, rPtr->length - rPtr->pointPos);
78. displayFixedWidthNumber(rPtr->digits, MAX\_REAL\_WIDTH);
79. **return**;
80. }
82. **void** addTwoLongReals(**const** Real\* rPtr1, **const** Real\* rPtr2, Real\* rPtrSum)
83. {
84. **int** i;
85. **int** carry = 0;
86. **int** sum = 0;
88. //向右对齐
89. **int** r1right = MAX\_REAL\_WIDTH - rPtr1->length ;
90. **int** r2right = MAX\_REAL\_WIDTH - rPtr2->length ;
91. **int** shiftLength = (r1right < r2right) ? r1right : r2right ;
93. //将两个数的小数点向右移动对齐并求和
94. **if**( r1right < r2right )
95. {    //判断表达精度的数位是否足够
96. //是否在移动过程中向右溢出
97. **if**(rPtr2->length+shiftLength + rPtr1->pointPos-rPtr2->pointPos > MAX\_REAL\_WIDTH
98. ||rPtr2->pointPos - rPtr1->pointPos > shiftLength )
99. {
100. rPtrSum->length = 0;
101. **return**;
102. }
103. **else**
104. {
105. shiftDigitsToRight( rPtr1->digits, shiftLength );
106. shiftDigitsToRight( rPtr2->digits, shiftLength + rPtr1->pointPos-rPtr2->pointPos );
107. }
108. }
109. **else** **if**( r1right > r2right )
110. {
111. **if**(rPtr1->length+shiftLength + rPtr2->pointPos-rPtr1->pointPos > MAX\_REAL\_WIDTH
112. || rPtr1->pointPos - rPtr2->pointPos > shiftLength )
113. {
114. rPtrSum->length = 0;
115. **return** ;
116. }
117. **else**
118. {
119. shiftDigitsToRight( rPtr1->digits, shiftLength + rPtr2->pointPos-rPtr1->pointPos );
120. shiftDigitsToRight( rPtr2->digits, shiftLength  );
121. }
122. }
123. **else** **if**( r1right == r2right )
124. {
125. **if**(rPtr1->pointPos!=rPtr2->pointPos)
126. {
127. rPtrSum->length = 0;
128. **return** ;
129. }
130. **else**
131. {
132. shiftDigitsToRight( rPtr1->digits, r1right  );
133. shiftDigitsToRight( rPtr2->digits, r2right  );
134. }
135. }

138. //判断是否溢出
139. **if**( rPtr1->length > MAX\_REAL\_WIDTH || rPtr2->length > MAX\_REAL\_WIDTH )
140. {
141. rPtrSum->length = 0;
142. **return** ;
143. }
145. //求向右对齐后长实数
146. **int** r1left = 0,r2left = 0;
147. **while**(rPtr1->digits[r1left] =='\0')
148. {
149. r1left++;
150. }
151. **while**(rPtr2->digits[r2left] =='\0')
152. {
153. r2left++;
154. }
155. **int** rleft = (r1left < r2left) ? r1left : r2left ;
156. rPtrSum->length = MAX\_REAL\_WIDTH - rleft ;
158. //求和
159. **for**(i = MAX\_REAL\_WIDTH - 1 ; i >= 0 ; i--)
160. {
161. **if**( rPtr1->digits[i] == '.' )
162. {
163. rPtrSum->digits[i] = '.';
164. rPtrSum->pointPos = rPtrSum->length - (MAX\_REAL\_WIDTH - ( i + 1 ) );
165. **continue**;
166. }
167. **else**
168. {
169. sum = carry;
170. **if**(rPtr1->digits[i] != '\0' )
171. {
172. sum += rPtr1->digits[i] - '0';
173. }
174. **if**( rPtr2->digits[i] != '\0' )
175. {
176. sum += rPtr2->digits[i] - '0';
177. }
178. }
179. carry = sum / 10;
180. rPtrSum->digits[i] = '0' + sum % 10;
181. }
182. **if**( carry > 0 )
183. {
184. //加法运算有进位，计算溢出
185. rPtrSum->length = 0 ;
186. **return** ;
187. }
189. **for**( i = MAX\_REAL\_WIDTH - rPtrSum->length - 1; i >=0 ; i-- )
190. {   // clear the rest of the memory
191. //生成随机数时，没有用到的数位填充'\0';
192. //显示正常数值时，无需打印；显示全部数宽的时候，空闲数位用字符'\_'代替
193. rPtrSum->digits[i] = '\0';
194. }
196. printf("Sum Result (%2d.%02d):", rPtrSum->length, rPtrSum->length - rPtrSum->pointPos );
197. displayFixedWidthNumber( rPtrSum->digits, MAX\_REAL\_WIDTH );
199. //最后将三个数的小数点向左移动，去除前置的0
200. shiftDigitsToLeft( rPtr1->digits, r1left  );
201. shiftDigitsToLeft( rPtr2->digits, r2left );
202. shiftDigitsToLeft( rPtrSum->digits, rleft );
204. **return**  ;
205. }
207. **void** displayAssignedLongReal(**int** pointPos, Real\* rPtr)
208. {
209. **int** i;
210. **for** (i = 0; i < pointPos - rPtr->pointPos; i++)
211. {
212. printf(" ");
213. }
214. **for** (i = 0; i < rPtr->length; i++)
215. {
216. printf("%c", rPtr->digits[i]);
217. }
218. printf("\n");
220. **return**;
221. }
223. **void** shiftDigitsToRight(**const** **char**\* longNumber, **int** shiftLength)
224. {
225. **char** \*end = longNumber + MAX\_REAL\_WIDTH - 1;
226. **char** \*start = longNumber + MAX\_REAL\_WIDTH - 1 - shiftLength;
227. **while** (start >= longNumber)
228. {
229. \*end = \*start;
230. end--;
231. start--;
232. }
233. **while** (end >= longNumber)
234. {
235. \*end = '\0';
236. end--;
237. }
239. printf("shiftRight  (%02d)  :", shiftLength);
240. displayFixedWidthNumber(longNumber, MAX\_REAL\_WIDTH);
241. **return** ;
242. }
244. **void** shiftDigitsToLeft(**const** **char**\* longNumber, **int** shiftLength)
245. {
246. **char** \*start = longNumber;
247. **char** \*end = longNumber + shiftLength;
248. **while**( end < longNumber + MAX\_REAL\_WIDTH )
249. {
250. \*start = \*end;
251. start++;
252. end++;
253. }
254. **while**( start < longNumber + MAX\_REAL\_WIDTH )
255. {
256. \*start = '\0';
257. start++;
258. }
260. printf("shiftLeft   (%02d)  :",shiftLength);
261. displayFixedWidthNumber( longNumber, MAX\_REAL\_WIDTH );
262. **return**;
263. }
265. **void** displayFixedWidthNumber(**const** **char**\* number, **const** **int** width)
266. {
267. **int** i;
268. **for** (i = 0; i < width; i++)
269. {
270. **if** (number[i] == '\0')
271. {
272. printf("\_");
273. }
274. **else**
275. {
276. printf("%c", number[i]);
277. }
278. }
279. printf("\n");
280. **return**;
281. }

## 七、指针操作函数返回值

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <time.h>
5. #define MAX\_REAL\_WIDTH 25
7. **typedef** **struct** {
8. **char** digits[MAX\_REAL\_WIDTH];
9. **int** pointPos;
10. **int** length;
11. } Real;
13. //原来的结构体创建过程，交给函数通过申请动态内存方式来实现，通过返回值带回
14. Real\* generateLongReal(**void**);
16. //新建一个函数销毁函数，释放所申请的动态内存
17. **void** destoryLongReal(Real\* rPtr);
19. //计算结果的结构体通过函数返回值带回
20. Real\* addTwoLongReals(**const** Real\* rPtr1, **const** Real\* rPtr2);
22. **void** displayAssignedLongReal(**int** pointPos, Real\* rPtr);
24. **void** shiftDigitsToRight(**const** **char**\* longNumber, **int** shiftLength);
26. **void** shiftDigitsToLeft(**const** **char**\* longNumber, **int** shiftLength);
28. **void** displayFixedWidthNumber(**const** **char**\* number, **const** **int** width);
30. **int** main(**void**)
31. {
32. srand(time(NULL));
33. //结构体对象由generateLongReal函数申请动态内存创建，通过指针返回值带回给main()进行操作
34. Real\* rPtr1 = generateLongReal( );
35. Real\* rPtr2 = generateLongReal( );
36. //函数返回值用于带回被动态内存创建的新对象
37. Real\* rPtrSum = addTwoLongReals(rPtr1, rPtr2);
39. **if** (rPtrSum->length > 0)
40. {
41. printf("\n  ");
42. displayAssignedLongReal(rPtrSum->pointPos, rPtr1);
43. printf("+ ");
44. displayAssignedLongReal(rPtrSum->pointPos, rPtr2);
45. printf("= ");
46. displayAssignedLongReal(rPtrSum->pointPos, rPtrSum);
47. }
48. **else**
49. {
50. printf("\nError! addition overflow!\n");
51. }
52. printf("\n");
54. //通过指针申请动态内存创建的结构体对象，无法自动销毁，需要通过代码释放内存
55. destoryLongReal(rPtr1);
56. destoryLongReal(rPtr2);
57. destoryLongReal(rPtrSum);
58. }
60. //结构体对象由generateLongReal函数申请动态内存创建，通过指针返回值带回给main()进行操作
61. Real \*generateLongReal(**void**)
62. {
63. Real\* rPtr = (Real\*)malloc(**sizeof**(Real));
65. rPtr->length = 2 + rand() % (MAX\_REAL\_WIDTH - 1);
66. rPtr->pointPos = 1 + rand() % (rPtr->length);
68. **char**\* cPtr = rPtr->digits;
69. \*cPtr = '1' + rand() % 9;
70. **for** (cPtr++; cPtr < rPtr->digits + rPtr->length - 1; cPtr++)
71. {
72. \*cPtr = '0' + rand() % 10;
73. }
74. **for** ( ; cPtr < rPtr->digits + MAX\_REAL\_WIDTH; cPtr++)
75. {
76. \*cPtr = '\0';
77. }
78. rPtr->digits[rPtr->length - 1] = '1' + rand() % 9;
79. rPtr->digits[rPtr->pointPos - 1] = '.';
81. printf("Randomize (%2d.%02d): ", rPtr->length, rPtr->length - rPtr->pointPos);
82. displayFixedWidthNumber(rPtr->digits, MAX\_REAL\_WIDTH);
84. **return** rPtr;
85. }
87. **void** destoryLongReal(Real\* rPtr)
88. {
89. free(rPtr);
91. **return**;
92. }
94. Real\* addTwoLongReals(**const** Real\* rPtr1, **const** Real\* rPtr2)
95. {
96. Real\* rPtrSum = (Real\*)malloc(**sizeof**(Real));
98. **int** i;
99. **int** carry = 0;
100. **int** sum = 0;
102. //向右对齐
103. **int** r1right = MAX\_REAL\_WIDTH - rPtr1->length ;
104. **int** r2right = MAX\_REAL\_WIDTH - rPtr2->length ;
105. **int** shiftLength = (r1right < r2right) ? r1right : r2right ;


109. //将两个数的小数点向右移动对齐并求和
110. **if**( r1right < r2right )
112. {    //判断表达精度的数位是否足够
113. //是否在移动过程中向右溢出
114. **if**(rPtr2->length+shiftLength + rPtr1->pointPos - rPtr2->pointPos > MAX\_REAL\_WIDTH
115. //是否向左溢出
116. || rPtr2->pointPos - rPtr1->pointPos > shiftLength)
117. {
118. rPtrSum->length = 0;
119. **return** rPtrSum;
120. }
121. **else**
122. {
123. shiftDigitsToRight( rPtr1->digits, shiftLength );
124. shiftDigitsToRight( rPtr2->digits, shiftLength + rPtr1->pointPos-rPtr2->pointPos );
125. }
126. }
127. **else** **if**( r1right > r2right )
128. {
129. **if**(rPtr1->length+shiftLength + rPtr2->pointPos - rPtr1->pointPos > MAX\_REAL\_WIDTH
130. || rPtr1->pointPos - rPtr2->pointPos > shiftLength)
131. {
132. rPtrSum->length = 0;
133. **return** rPtrSum;
134. }
135. **else**
136. {
137. shiftDigitsToRight( rPtr1->digits, shiftLength + rPtr2->pointPos-rPtr1->pointPos );
138. shiftDigitsToRight( rPtr2->digits, shiftLength  );
139. }
140. }
141. **else** **if**( r1right == r2right )
142. {
143. **if**(rPtr1->pointPos!=rPtr2->pointPos)
144. {
145. rPtrSum->length = 0;
146. **return** rPtrSum;
147. }
148. **else**
149. {
150. shiftDigitsToRight( rPtr1->digits, r1right  );
151. shiftDigitsToRight( rPtr2->digits, r2right  );
152. }
153. }

156. //判断是否溢出
157. **if**( rPtr1->length > MAX\_REAL\_WIDTH || rPtr2->length > MAX\_REAL\_WIDTH )
158. {
159. rPtrSum->length = 0;
160. **return** rPtrSum;
161. }
163. //求向右对齐后长实数
164. **int** r1left = 0,r2left = 0;
165. **while**(rPtr1->digits[r1left] =='\0')
166. {
167. r1left++;
168. }
169. **while**(rPtr2->digits[r2left] =='\0')
170. {
171. r2left++;
172. }
173. **int** rleft = (r1left < r2left) ? r1left : r2left ;
174. rPtrSum->length = MAX\_REAL\_WIDTH - rleft ;
176. //求和
177. **for**(i = MAX\_REAL\_WIDTH - 1 ; i >= 0 ; i--)
178. {
179. **if**( rPtr1->digits[i] == '.' )
180. {
181. rPtrSum->digits[i] = '.';
182. rPtrSum->pointPos = rPtrSum->length - (MAX\_REAL\_WIDTH - ( i + 1 ) );
183. **continue**;
184. }
185. **else**
186. {
187. sum = carry;
188. **if**(rPtr1->digits[i] != '\0' )
189. {
190. sum += rPtr1->digits[i] - '0';
191. }
192. **if**( rPtr2->digits[i] != '\0' )
193. {
194. sum += rPtr2->digits[i] - '0';
195. }
196. }
197. carry = sum / 10;
198. rPtrSum->digits[i] = '0' + sum % 10;
199. }
200. **if**( carry > 0 )
201. {
202. //加法运算有进位，计算溢出
203. rPtrSum->length = 0 ;
204. **return** rPtrSum;
205. }
207. **for**( i = MAX\_REAL\_WIDTH - rPtrSum->length - 1; i >=0 ; i-- )
208. {   // clear the rest of the memory
209. //生成随机数时，没有用到的数位填充'\0';
210. //显示正常数值时，无需打印；显示全部数宽的时候，空闲数位用字符'\_'代替
211. rPtrSum->digits[i] = '\0';
212. }
214. printf("Sum Result(%2d.%02d):", rPtrSum->length, rPtrSum->length - rPtrSum->pointPos );
215. displayFixedWidthNumber( rPtrSum->digits, MAX\_REAL\_WIDTH );
217. //最后将三个数的小数点向左移动，去除前置的0
218. shiftDigitsToLeft( rPtr1->digits, r1left  );
219. shiftDigitsToLeft( rPtr2->digits, r2left );
220. shiftDigitsToLeft( rPtrSum->digits, rleft );
222. **return** rPtrSum;
223. }
225. **void** displayAssignedLongReal(**int** pointPos, Real\* rPtr)
226. {
227. **int** i;
228. **for** (i = 0; i < pointPos - rPtr->pointPos; i++)
229. {
230. printf(" ");
231. }
232. **for** (i = 0; i < rPtr->length; i++)
233. {
234. printf("%c", rPtr->digits[i]);
235. }
236. printf("\n");
238. **return**;
239. }
241. **void** shiftDigitsToRight(**const** **char**\* longNumber, **int** shiftLength)
242. {
243. **char** \*end = longNumber + MAX\_REAL\_WIDTH - 1;
244. **char** \*start = longNumber + MAX\_REAL\_WIDTH - 1 - shiftLength;
245. **while** (start >= longNumber)
246. {
247. \*end = \*start;
248. end--;
249. start--;
250. }
251. **while** (end >= longNumber)
252. {
253. \*end = '\0';
254. end--;
255. }
257. printf("shiftRight (%02d)  :", shiftLength);
258. displayFixedWidthNumber(longNumber, MAX\_REAL\_WIDTH);
259. **return** ;
260. }
262. **void** shiftDigitsToLeft(**const** **char**\* longNumber, **int** shiftLength)
263. {
264. **char** \*start = longNumber;
265. **char** \*end = longNumber + shiftLength;
266. **while**( end < longNumber + MAX\_REAL\_WIDTH )
267. {
268. \*start = \*end;
269. start++;
270. end++;
271. }
272. **while**( start < longNumber + MAX\_REAL\_WIDTH )
273. {
274. \*start = '\0';
275. start++;
276. }
278. printf("shiftLeft  (%02d)  :",shiftLength);
279. displayFixedWidthNumber( longNumber, MAX\_REAL\_WIDTH );
280. **return** ;
281. }
283. **void** displayFixedWidthNumber(**const** **char**\* number, **const** **int** width)
284. {
285. **int** i;
286. **for** (i = 0; i < width; i++)
287. {
288. **if** (number[i] == '\0')
289. {
290. printf("\_");
291. }
292. **else**
293. {
294. printf("%c", number[i]);
295. }
296. }
297. printf("\n");
298. **return**;
299. }

## 八、指针操作动态内存

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <time.h>
5. #define MAX\_REAL\_WIDTH 25
7. **typedef** **struct** {
8. //数据结构有变化，结构体成员char\* digits是只想动态内存区的指针
9. **char**\* digits;
10. **int** pointPos;
11. **int** length;
12. } Real;
14. Real\* generateLongReal(**void**);
16. **void** destoryLongReal(Real\* rPtr);
18. Real\* addTwoLongReals(Real\* rPtr1,Real\* rPtr2);
20. //左移和右移的函数不再是对固定内存区不同数位的操作，但是直接调整动态内存区的大小和内容
21. **void** shiftDigitsToRight(Real \*rPtr, **int** shiftLength);
23. **void** shiftDigitsToLeft(Real \*rPtr, **int** shiftLength);
25. **void** displayAssignedLongReal(**int** assignedLength, **const** Real\* rPtr);

28. **int** main(**void**)
29. {
30. srand(time(NULL));
31. Real\* rPtr1 = generateLongReal( );
32. Real\* rPtr2 = generateLongReal( );
33. Real\* rPtrSum = addTwoLongReals(rPtr1, rPtr2);
35. //结构体可以动态调整所需的字符数组区域大小，无需再判断是否溢出
36. printf("\n  ") ;
37. displayAssignedLongReal(rPtrSum->pointPos-rPtr1->pointPos, rPtr1);
38. printf("+ ");
39. displayAssignedLongReal(rPtrSum->pointPos-rPtr2->pointPos, rPtr2);
40. printf("= ");
41. displayAssignedLongReal(0, rPtrSum);

44. destoryLongReal(rPtr1);
45. destoryLongReal(rPtr2);
46. destoryLongReal(rPtrSum);
47. }
49. Real \*generateLongReal(**void**)
50. {
51. //当前情况，结构体对象被创建时，指针类型的成员被一起创建，但是并未为其分配内存，还需要再操作
52. Real\* rPtr = (Real\*)malloc(**sizeof**(Real));
54. rPtr->length = 2 + rand() % (MAX\_REAL\_WIDTH - 1);
55. rPtr->pointPos = 1 + rand() % (rPtr->length);
57. //allocate memory for rPtr->digits array
58. rPtr->digits = (**char**\*)calloc(rPtr->length , **sizeof**(**char**));
59. rPtr->digits[0] = '1' + rand() % 9;
61. **int** i;
62. **for**( i = 1; i < rPtr->length - 1; i++ )
63. {
64. rPtr->digits[i] = '0' + rand() % 10;
65. }
66. rPtr->digits[rPtr->length - 1] = '1' + rand() % 9;
67. rPtr->digits[rPtr->pointPos - 1] = '.';
69. printf("Randomize (%2d.%02d):", rPtr->length, rPtr->length - rPtr->pointPos);
70. displayAssignedLongReal(0, rPtr);
72. **return** rPtr;
73. }
75. **void** destoryLongReal(Real\* rPtr)
76. {
77. **if**( rPtr->digits != NULL )
78. {
79. free(rPtr->digits);
80. }
81. free(rPtr);
83. **return**;
84. }
86. Real\* addTwoLongReals(Real\* rPtr1, Real\* rPtr2)
87. {
88. Real\* rPtrSum = (Real\*)malloc(**sizeof**(Real));
90. **int** i;
91. **int** carry = 0;
92. **int** sum = 0;
94. //向右对齐小数点
95. **if**( rPtr1->pointPos > rPtr2->pointPos )
96. {
97. shiftDigitsToRight( rPtr2, rPtr1->pointPos - rPtr2->pointPos );
98. rPtrSum->pointPos = rPtr1->pointPos;
99. }
100. **else** **if**( rPtr1->pointPos < rPtr2->pointPos )
101. {
102. shiftDigitsToRight( rPtr1, rPtr2->pointPos - rPtr1->pointPos );
103. rPtrSum->pointPos = rPtr2->pointPos;
104. }
105. **else** **if**( rPtr1->pointPos == rPtr2->pointPos )
106. {
107. rPtrSum->pointPos = rPtr1->pointPos;
108. }
110. //结果长度
111. rPtrSum->length = (rPtr1->length > rPtr2->length) ? rPtr1->length : rPtr2->length;
112. rPtrSum->digits = (**char**\*)calloc(rPtrSum->length , **sizeof**(**char**));
113. rPtrSum->digits[rPtrSum->length] = '\0';
115. //从后向前逐位相加
116. **for**(i = rPtrSum->length - 1 ; i >= 0 ; i--)
117. {
118. **if**( i== rPtrSum->pointPos - 1 )
119. {
120. rPtrSum->digits[i] = '.';
121. **continue**;
122. }
123. **else**
124. {
125. sum = carry;
126. **if**( rPtr1->digits[i] >= '0' && rPtr1->digits[i] <= '9' )
127. {
128. sum += rPtr1->digits[i] - '0';
129. }
130. **if**( rPtr2->digits[i] >= '0' && rPtr2->digits[i] <= '9' )
131. {
132. sum += rPtr2->digits[i] - '0';
133. }
134. }
135. carry = sum / 10;
136. rPtrSum->digits[i] = '0' + sum % 10;
137. }
138. **if**( carry > 0 )
139. {
140. ;
141. //存储数位不足时向右移动申请新内存，支持拓展数位，不再存在溢出的问题
142. rPtrSum->length++;
143. rPtrSum->digits = (**char**\*)realloc(rPtrSum->digits, rPtrSum->length \* **sizeof**(**char**));
144. shiftDigitsToRight( rPtrSum, 1 );
145. rPtrSum->digits[0] = '0' + carry;
146. }
148. printf("Sum Result(%2d.%02d):", rPtrSum->length, rPtrSum->length - rPtrSum->pointPos );
149. displayAssignedLongReal(0, rPtrSum);
151. **if**(rPtr1->pointPos > rPtr2->pointPos)
152. {
153. shiftDigitsToLeft( rPtr2, rPtr1->pointPos - rPtr2->pointPos );
154. }
155. **else** **if**(rPtr1->pointPos < rPtr2->pointPos)
156. {
157. shiftDigitsToLeft( rPtr1, rPtr2->pointPos - rPtr1->pointPos );
158. }
160. **return** rPtrSum;
161. }
163. **void** shiftDigitsToRight(Real \*rPtr, **int** shiftLength)
164. {
165. rPtr->digits = (**char**\*)realloc(rPtr->digits, (rPtr->length+shiftLength+1) \* **sizeof**(**char**));
166. **int** i;
167. **for** ( i = rPtr->length - 1; i >= 0; i--)
168. {
169. rPtr->digits[i + shiftLength] = rPtr->digits[i];
170. }
171. **for** ( i = 0; i < shiftLength; i++)
172. {
173. rPtr->digits[i] = ' ';
174. }
175. rPtr->length += shiftLength;
176. rPtr->digits[rPtr->length] = '\0';
178. printf("shiftRight (%02d)  :",shiftLength);
179. displayAssignedLongReal( 0,rPtr);
180. **return**;
181. }
183. **void** shiftDigitsToLeft(Real \*rPtr, **int** shiftLength)
184. {
185. **int** i;
186. **for** ( i = 0; i < rPtr->length - shiftLength; i++)
187. {
188. rPtr->digits[i] = rPtr->digits[i + shiftLength];
189. }
190. rPtr->length -= shiftLength;
191. rPtr->digits = (**char**\*)realloc(rPtr->digits, (rPtr->length + 1) \* **sizeof**(**char**));
192. rPtr->digits[rPtr->length] = '\0';
194. printf("shiftLeft  (%02d)  :",shiftLength);
195. displayAssignedLongReal( 0, rPtr );
196. **return**;
197. }
199. **void** displayAssignedLongReal(**int** assignedLength, **const** Real\* rPtr)
200. {
201. **int** i;
202. **for** (i = 0; i < assignedLength  ; i++)
203. {
204. printf(" ");
205. }
206. **for** (i = 0; i < rPtr->length; i++)
207. {
208. printf("%c", rPtr->digits[i]);
209. }
210. printf("\n");
212. **return**;
213. }