

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 | 孙陈宇科 | 学号 | U202413452 |

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 2024.12 | 地点 | 华中科技大学 |

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

# 实验目的

完成大数计算系列代码。

# 实验环境

操作系统：Windows 10

编程工具：CodeBlocks 16.01

# 实验一

## 实验任务

* 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
* 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
* 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
* 打印两个长整数求和的竖式计算过程

## 实验步骤

Main函数编程思路如下图：

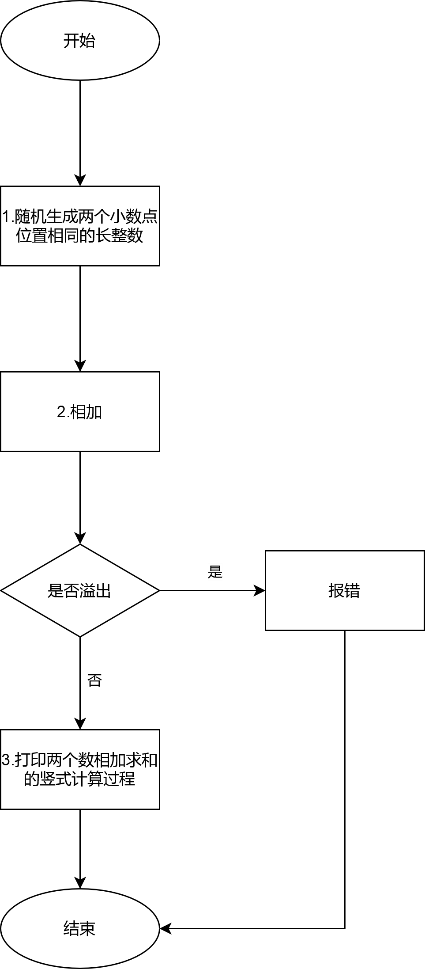


图3.1.1

* 1. 随机生成两个小数点位置相同的长整数

通过结构体存储长整数。定义宏POINT\_POSITION代表小数点位置。

1. **typedef** **struct**{
2. **char** digits[ MAX\_REAL\_WIDTH ];
3. **int** length;
4. }Real;

通过函数Real generateLongReal( void )；生成随机长整数。（具体代码见附录）

* 1. 相加

通过函数Real addTowLongReal(Real r1, Real r2);相加并返回结构体。

函数编写思路如下：

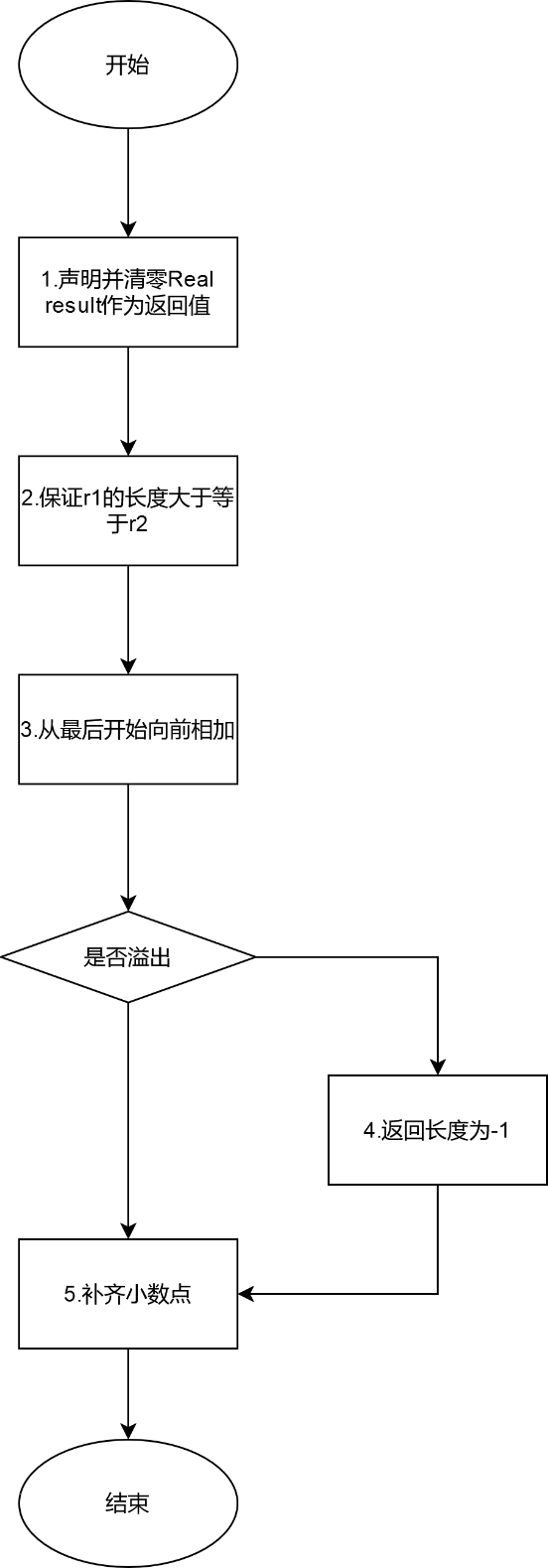


图3.1.2

保证r1的长度大于等于r2是为了便于下一步计算字符‘0’到‘9’与‘/0’相加。

从后向前逐位相加，并用整型carry记录进位。遇到小数点位时跳过。

如果最后carry为1，则首位有进位。溢出返回的结构体（result）中长度（length）为-1，便于main函数的报错。

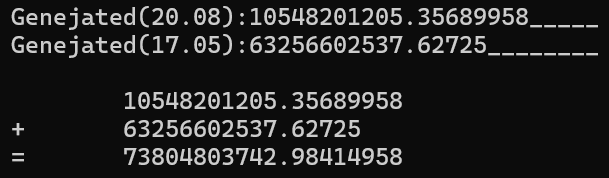
* 1. 打印两个数相加求和的竖式计算过程

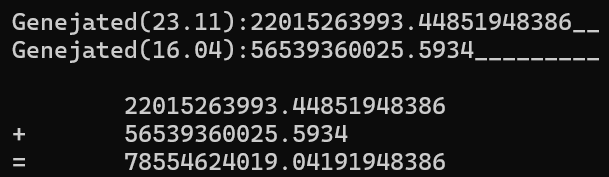
如果没有溢出（首位进位），则通过函数void displayLongReal( Real r );打印竖式计算过程。

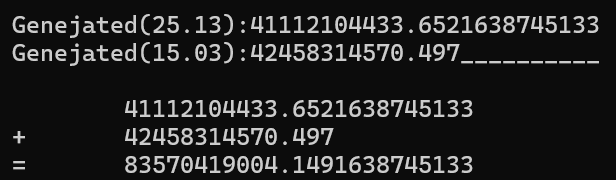
## 代码测试

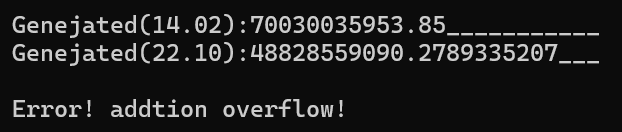
### 测试点1的测试结果

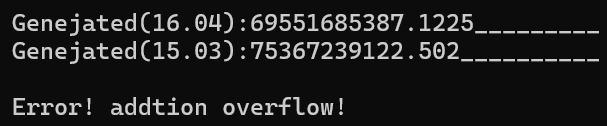
运行数次代码，期望得到的加法结果和竖式计算表达无误。

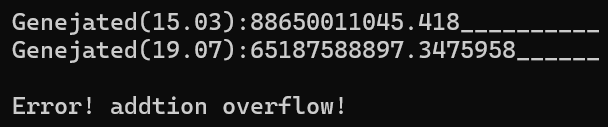












结论：可以正确进行加法计算；可以正确表达竖式计算过程。

## 实验结论

代码达到功能目标

## 实验总结

经过数次调试，通过补充了部分步骤，例如在加法函数中补小数点等。最终达到实验目标

# 实验二

## 实验任务

* 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
* 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
* 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
* 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

main函数编程思路如实验一（图3.1.1）。

1、增加结构体成员，以记录随机的小数点位置。

1. **typedef** **struct**{
2. **char** digits[ MAX\_REAL\_WIDTH ];
3. **int** length;
4. **int** pointPos;
5. }Real;

2、按需要修改了addTowLongReal函数。

增加了shiftDigitsToRight和shiftDigitsToLeft函数。

相加函数的思路如下图：

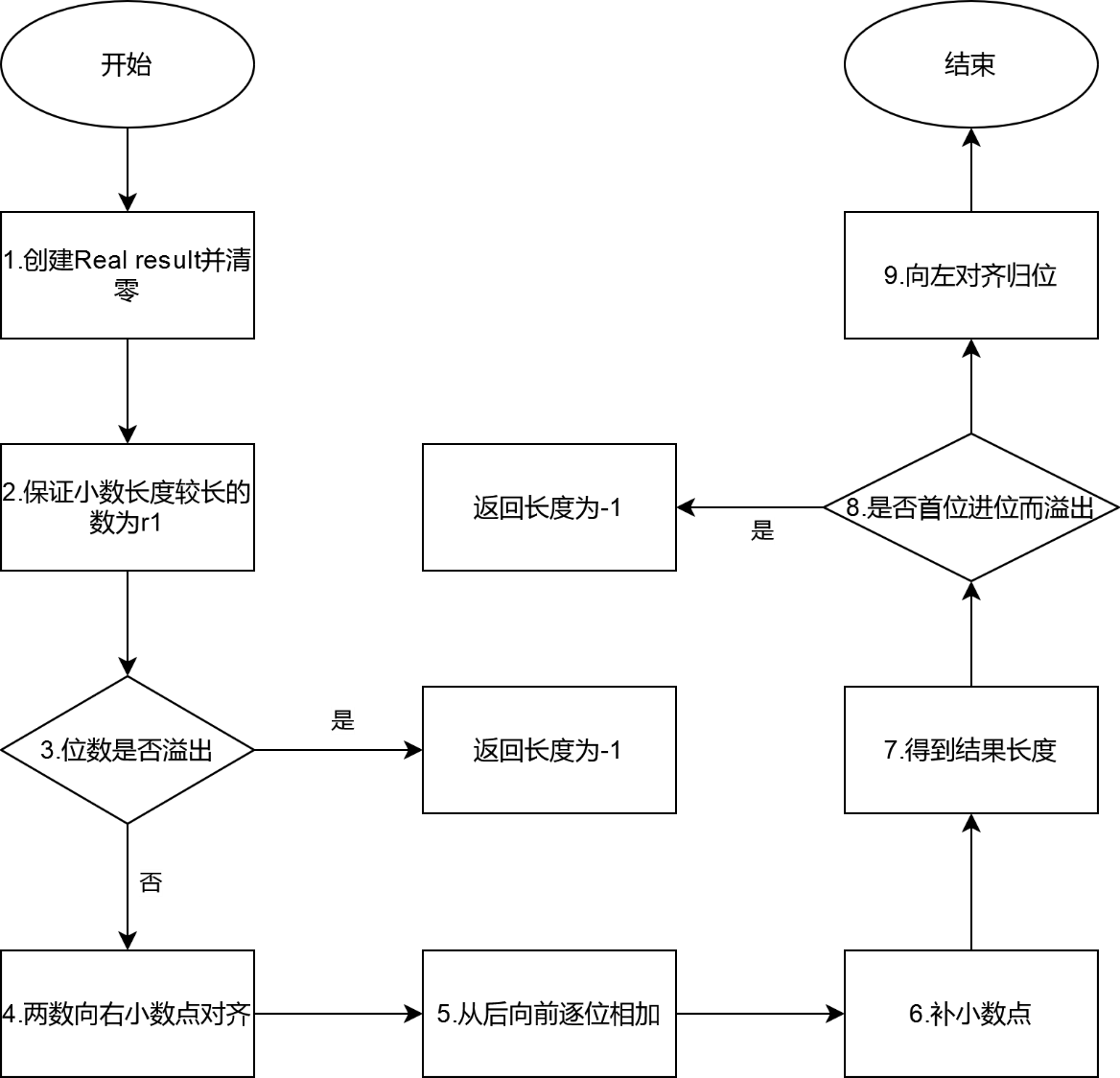


图3.1.2

2、如果r1的小数长度比让r2短，则r1、r2交换，便于计算。

3、新增溢出可能。如果结果的小数部分长度加上整数部分长度超过最大长度则报错。

4、用新增的函数使两数右移对齐小数点，便于逐位计算。

5、逐位计算时，相比于实验一，需要多考虑一些首位和末尾的计算。

7、右移使得结果长度的获得不那么直接。这里选择了计数首位前的’\0’的个数来计算。

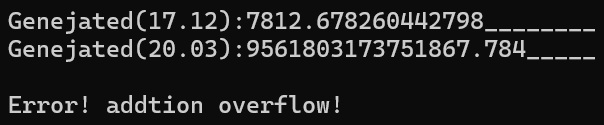
9、最后左移显示结果。

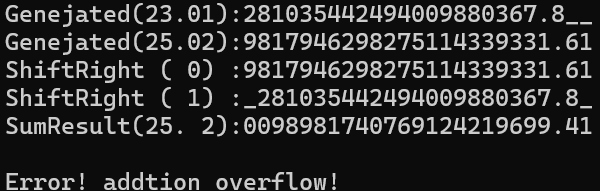
## 代码测试

以下每种情况都有3次以上的测试，下面仅列出一部分测试截图。

### 测试点1的测试结果

运行数次代码，期望两种种情况下的溢出判断正确。即首位进位的溢出和位数不足的溢出。



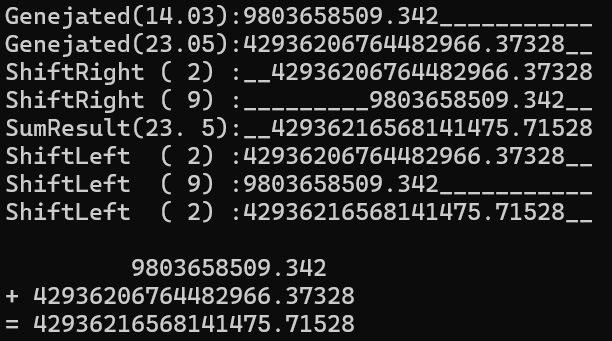


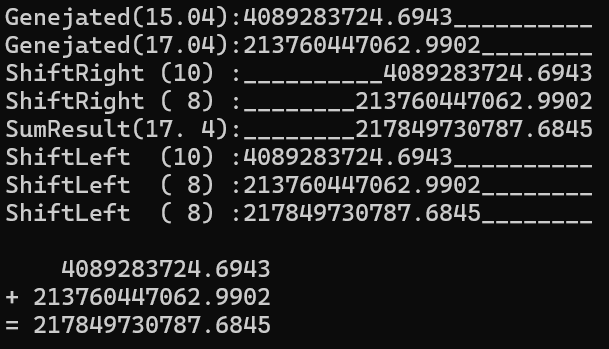
结论：可以正确进行溢出判断。

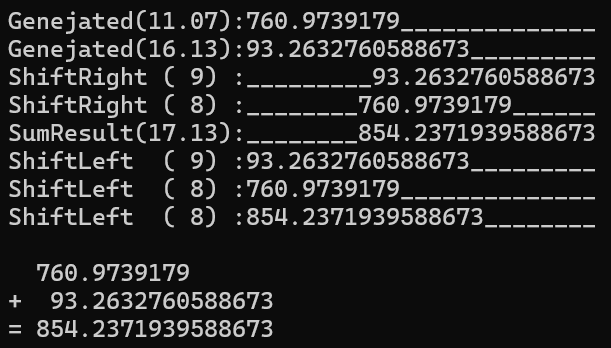
### 测试点2的测试结果

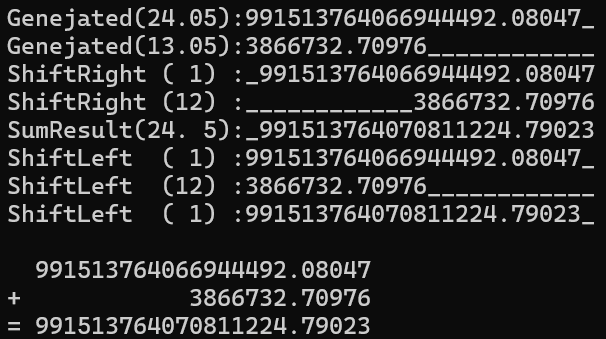
运行数次代码，期望在各种情况下得到的加法结果和竖式计算表达无误。

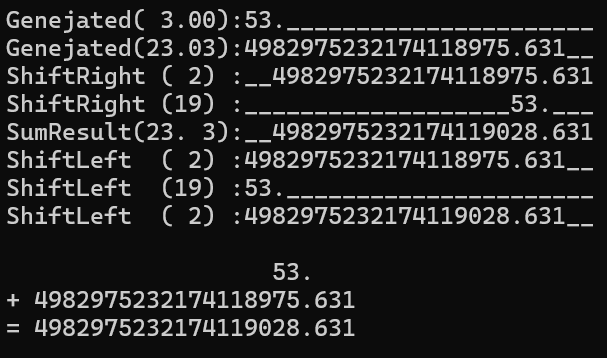
情况包括末尾不对齐、首位不对齐，首位进位遇9连续进位、对齐的首位进位等。











结论：可以正确进行加法计算；可以正确表达竖式计算过程。

## 实验结论

代码达到功能目标

## 实验总结

经过数次调试，通过补充了多个情况的分类讨论以及调整步骤顺序。最终达到实验目标

# 实验三

## 实验任务

* 预期功能与上一个实验相同
* 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
* 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

Main函数编程思路基本如实验一，仅在main函数的最后增加释放动态内存的步骤。

1. // 释放内存
2. free(real1);
3. free(real2);
4. free(realSum);

此实验中结构体的创建过程由函数申请动态内存后由返回值带回。

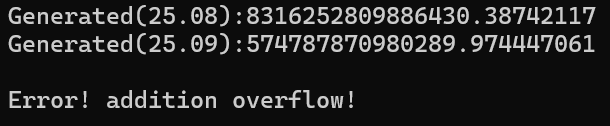
1. Real\* addTowLongReal(Real\* r1, Real\* r2) {
2. Real\* result = (Real\*)malloc(**sizeof**(Real));
3. ……
4. }

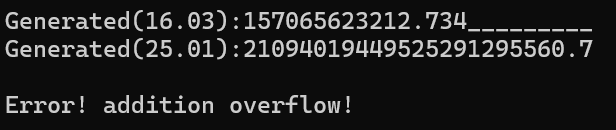
相加的函数也通过申请动态内存创建结果的结构体后通过返回值带回地址。

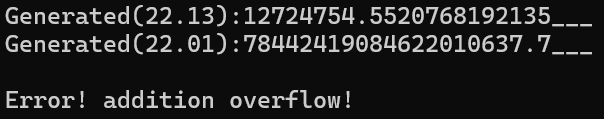
其他函数按需改为通过指针传递结构体等。函数直接根据指针在对应地址进行操作。

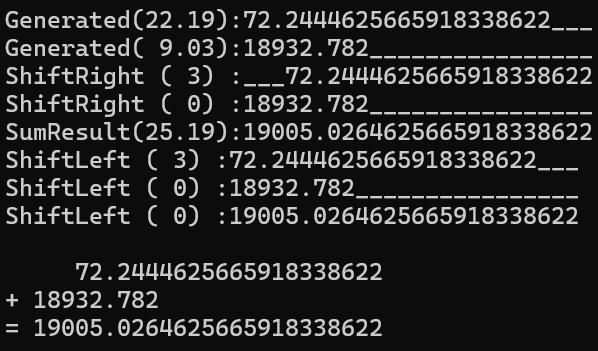
## 代码测试

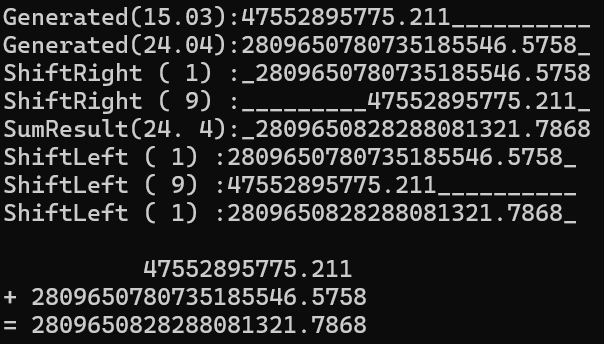
运行数次代码，期望得到的加法结果和竖式计算表达无误。

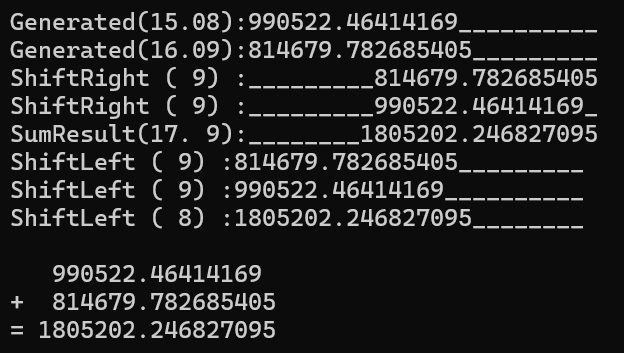


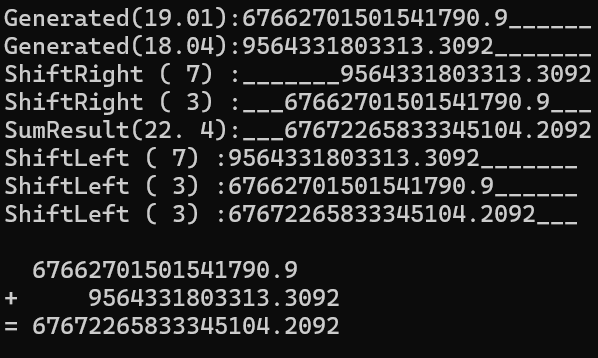












结论：可以正确进行加法计算；可以正确表达竖式计算过程。

## 实验结论

代码达到功能目标

## 实验总结

通过修改代码掌握了函数返回指针类型的方法，通过指针返回值带回当前函数可访问内存对象以及通过指针参数申请、释放动态结构体对象的方法

# 本课程学习总结

一、学习C语言的历程和总结

（一）控制语句与基本数据类型

初识C语言，是从控制语句和基本数据类型开始的。控制语句就像是程序的骨头，让我能够根据不同的条件和循环要求，控制程序的执行流程。记得第一次用if语句实现一个简单的判断程序，输入一个数字判断它是正数、负数还是零，那种看到程序按照预期运行的成就感，让我对C语言充满了好奇和热情，也因为渴望实现更加复杂的功能而充满了继续学习的动力。基本数据类型则是搭建程序的“基石”，整型、浮点型、字符型……每一种类型都有其独特的用途和特点。在学习过程中，我通过编写各种小程序来熟悉它们，比如看看浮点型到哪一位开始计算不精确等，这些基础的学习为后续更复杂的编程打下了一定的基础。

（二）函数与数组

当学到函数时，我意识到这是代码复用和模块化的关键。将一段具有特定功能的代码封装成函数，不仅让程序结构更加清晰，还能在需要的时候随时调用，大大提高了编程效率。我尝试着把之前写的一些重复代码片段改写成函数，比如打印字符串的代码封装成一个displayStr函数，然后在主函数中多次调用它，程序的可读性和可维护性瞬间提升。数组的学习则让我能够处理多个相同类型数据的集合。一开始，我用数组来存储和操作一组整数，通过下标来访问和修改数组元素，实现了诸如求一组数的平均值、最大值等功能。随着学习的深入，我还学会了二维数组，用它来处理矩阵运算等更复杂的问题，这让我对数据的组织和处理有了更深刻的理解。

（三）结构体与指针

结构体的出现，让我对数据的组织方式有了全新的认识。它可以将不同类型的数据组合在一起，形成一个有机的整体。比如定义一个学生结构体，包含学号、姓名、成绩等信息，这样在处理学生数据时就更加方便和直观。指针的学习则是一个挑战，它让我对内存有了更深入的了解。一开始，我对指针的概念很模糊，总是搞不清楚指针变量和普通变量的区别。但在老师的耐心讲解和多次练习后，我逐渐明白了指针是指向内存地址的变量，通过指针可以间接访问和修改变量的值。指针在函数传参、动态内存分配等方面有着重要的应用，它让程序变得更加灵活和高效。

二、程序中出现的问题与改进

（一）逻辑错误

在编写程序时，经常会遇到逻辑错误。比如在用循环语句实现一个数列求和的程序时，我错误地将循环条件设置为i <= n，导致多计算了一项。后来发现问题所在，将条件改为i < n，才得到了正确的结果。为了避免这类错误，我学会了在编写代码前先在纸上画出流程图，明确程序的逻辑结构，然后再进行编码。

（二）语法错误

语法错误是初学者常犯的错误。比如在定义函数时，忘记写返回类型；在使用指针时，忘记加\*或&等。这些错误会导致编译器报错，无法生成可执行文件。为了减少语法错误，我养成了认真阅读教材和参考文档的习惯，对几个语法点通过反复练习才熟练掌握。同时，在编写代码时，我也尽量遵循代码规范，使用清晰的缩进和注释，这样可以更容易地发现语法错误。

另外，还有初始化的问题，字符串等定义使为清零导致出现随机的字符的情况也比较常见。在编译是还难以发现，多次测试才容易发现问题。所以我养成了声明定义时立刻赋值0的习惯。

三、代码规范与调试技巧

（一）代码规范

良好的代码规范对于程序的可读性和可维护性至关重要。我遵循以下几点规范：

**空格：**在各类运算符前后、分号和逗号后、括号内、关键字后使用空格。使代码更加整洁、易读，也有助于团队协作和代码维护。

**缩进与对齐**：使用统一的缩进方式，通常就是按一下Tab键，让代码层次分明，便于阅读。例如，在if语句和循环语句的代码块中，都进行适当的缩进。

**命名规范**：变量名和函数名采用有意义的命名方式，尽量使用驼峰命名法或下划线分隔命名法。比如，用studentName表示学生姓名，用calculate\_average表示计算平均值的函数。以及迫使自己使用英文。

**注释**：在代码的关键部分添加注释，说明代码的功能和实现方法。对于复杂的函数，会在函数上方添加详细的注释，包括函数的参数、返回值和功能描述。例如：

1. // 计算一组数的平均值
2. // 参数：arr 数组，n 数组长度
3. // 返回值：平均值
4. **double** calculate\_average(**int** arr[], **int** n) {
5. ...
6. }

（二）调试技巧

调试是编程过程中不可或缺的环节，以下是一些常用的调试技巧：

**编译器报错信息**：仔细阅读编译器的报错信息，它通常会指出错误所在的行号和错误类型。根据报错信息，可以快速定位语法错误等简单问题。

**打印调试**：在程序的关键位置添加printf语句，输出变量的值和程序的执行流程。通过观察打印结果，可以判断程序是否按照预期运行，从而发现逻辑错误。例如，在循环语句中打印循环变量的值，检查循环是否正常执行。

**Debug**：使用编译软件的debug功能，可以逐行执行代码，观察每一步的变量变化和程序执行情况。这对于调试复杂的程序和查找隐藏的错误非常有效。

四、学习记录与心得

学习C语言的过程充满了挑战和乐趣。从一开始对控制语句和基本数据类型的懵懂，到后来熟练运用函数、数组、结构体和指针，每一步都离不开优质的课件和作业。老师在课堂上生动的讲解、耐心的答疑，让我对C语言的理解更加深入。他总是能够用简单易懂的例子，以及生动形象的比喻，把复杂的概念讲解得清清楚楚，让我敬佩不已。同时，我也深深感受到了C语言的魅力，它简洁而强大，能够让我实现各种各样的功能，解决实际问题。在学习过程中，我对C语言的兴趣越来越浓厚，每当看到自己编写的程序成功运行，都会有一种巨大的满足感。

然而，在学习过程中，我也犯过一些错误。有一次考试，由于没有好好复习，导致很多知识点都模模糊糊，考试成绩很不理想。这次失败让我深刻认识到了复习的重要性。在后续的学习中，我制定了详细的复习计划，定期回顾学过的知识，做大量的练习题，确保每个知识点都扎实掌握。通过这次经历，我明白了学习不能临时抱佛脚，只有平时努力积累，才能在考试和实际应用中取得好成绩。

回顾这段学习历程，虽然遇到了不少困难，但也在解决问题的过程中不断成长。我感谢老师的辛勤付出，让我在C语言的海洋中畅游；我也感谢这段学习经历，让我对编程有了更深刻的理解和热爱。未来，我将继续努力学习C语言，探索更多的编程知识，用所学的技能解决更多的实际问题，为自己的编程之路奠定坚实的基础。

# 附录

1. 实验一
2. main.c
3. //固定点小数相加
5. #include <stdio.h>
6. #include <stdlib.h>
7. #include <string.h>
8. #include <time.h>
10. #define MAX\_REAL\_WIDTH  25
11. #define POINT\_POSITION  12
13. **typedef** **struct**{
14. **char** digits[ MAX\_REAL\_WIDTH ];
15. **int** length;
16. }Real;
17. //随机生成数字
18. Real generateLongReal( **void** );
19. //相加
20. Real addTowLongReal(Real r1, Real r2);
21. //打印数字
22. **void** displayLongReal( Real r );
23. //用'\0'补位
24. **void** displayFixedWidthNumber(**char** number[], **const** **int** width);
26. **int** main()
27. {
28. srand( time(NULL) );
29. Real real1 = generateLongReal();
30. Real real2 = generateLongReal();
31. Real realSum = addTowLongReal( real1, real2 );
33. **if** ( realSum.length > 0 )
34. {
36. printf("\n ");
37. displayLongReal( real1 );
38. printf("+ ");
39. displayLongReal( real2 );
40. printf("= ");
41. displayLongReal( realSum );
42. }
43. **else**{
44. printf("\nError! addtion overflow!\n");
45. }
47. printf("\n");
49. **return** 0;
50. }
52. Real generateLongReal( **void** )
53. {
54. Real r;
55. **int** fractionLength = rand() % (MAX\_REAL\_WIDTH - POINT\_POSITION + 1);
56. r.length = POINT\_POSITION + fractionLength;
58. **int** i;
59. r.digits[0] = '1' + rand() % 9;
60. **for** ( i = 1; i < r.length - 1;i ++){
61. r.digits[i] = '0' + rand() % 10;
62. }
63. r.digits[r.length - 1] = '1' + rand() % 9;
64. r.digits[POINT\_POSITION - 1] = '.';
65. **for**(i = r.length; i < MAX\_REAL\_WIDTH; i ++ ){
66. r.digits[i] = '\0';
67. }
69. printf("Genejated(%2d.%02d):",r.length, r.length - POINT\_POSITION );
70. displayFixedWidthNumber( r.digits, MAX\_REAL\_WIDTH);
72. **return** r;
73. }
75. Real addTowLongReal(Real r1, Real r2) {
76. Real result;
77. memset(result.digits, 0, MAX\_REAL\_WIDTH);
78. **int** carry = 0;
79. **int** i;
81. **if** (r2.length > r1.length) {
82. Real temp = r1;
83. r1 = r2;
84. r2 = temp;
85. }
87. result.length = r1.length;
88. //从后向前相加
89. **for** (i = r1.length - 1; i >= 0; i--) {
90. **if** (i == POINT\_POSITION - 1){
91. **continue**;
92. }
93. **if** ( r2.digits[i] == 0){
94. result.digits[i] = r1.digits[i];
95. }**else** **if**(r1.digits[i] - '0' + r2.digits[i] - '0' + carry >= 10){
96. result.digits[i] = (r1.digits[i] - '0' + r2.digits[i] - '0' + carry) % 10 + '0';
97. carry = 1;
98. }**else** **if**(r1.digits[i] - '0' + r2.digits[i] - '0' + carry < 10){
99. result.digits[i] = (r1.digits[i] - '0' + r2.digits[i] - '0' + carry) + '0';
100. carry = 0;
101. }
102. }
104. // 溢出判断
105. **if** (carry > 0) {
106. result.length = -1;
107. }
109. // 补小数点
110. result.digits[POINT\_POSITION - 1] = '.';
112. **return** result;
113. }

116. **void** displayLongReal( Real r ){
117. **int** i;
118. printf("\t");
119. **for** (i = 0; i < r.length; i ++ ){
120. printf("%c",r.digits[i]);
121. }
122. printf("\n");
123. **return**;
124. }
126. **void** displayFixedWidthNumber( **char** number[], **const** **int** width ){
127. **int** i;
128. **for**(i = 0; i < width; i ++){
129. **if**( number[i] == '\0'){
130. printf("\_");
131. }
132. **else**{
133. printf("%c",number[i]);
134. }
135. }
137. printf("\n");
138. **return**;
139. }
140. 实验二

（1）main.c

1. //结构体浮动点小数相加
2. //scyk
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7. #include <time.h>
9. #define MAX\_REAL\_WIDTH  25
11. **typedef** **struct**{
12. **char** digits[ MAX\_REAL\_WIDTH ];
13. **int** length;
14. **int** pointPos;
15. }Real;
16. //随机生成数字
17. Real generateLongReal( **void** );
18. //相加
19. Real addTowLongReal(Real r1, Real r2);
20. //打印数字
21. **void** displayAssignedLongReal( **int** pointPos, Real r );
22. //右移
23. Real shiftDigitsToRight( Real r, **int** shiftLength);
24. //左移
25. Real shiftDigitsToLeft( Real r, **int** shiftLength);
26. //用'\0'补位
27. **void** displayFixedWidthNumber(**char** number[], **const** **int** width);
29. **int** main()
30. {
31. srand( time(NULL) );
32. Real real1 = generateLongReal();
33. Real real2 = generateLongReal();
34. Real realSum = addTowLongReal( real1, real2 );
36. **if** ( realSum.length > 0 )
37. {
38. printf("\n  ");
39. displayAssignedLongReal(realSum.pointPos, real1 );
40. printf("+ ");
41. displayAssignedLongReal(realSum.pointPos, real2 );
42. printf("= ");
43. displayAssignedLongReal(realSum.pointPos, realSum );
44. }
45. **else**{
46. printf("\nError! addtion overflow!\n");
47. }
49. printf("\n");
51. **return** 0;
52. }
54. Real generateLongReal( **void** )
55. {
56. Real r;
57. r.pointPos = 2 + rand() % 23;
58. **int** fractionLength = rand() % (MAX\_REAL\_WIDTH - r.pointPos + 1);
59. r.length = r.pointPos + fractionLength;
61. **int** i;
62. r.digits[0] = '1' + rand() % 9;
63. **for** ( i = 1; i < r.length - 1;i ++){
64. r.digits[i] = '0' + rand() % 10;
65. }
66. r.digits[r.length - 1] = '1' + rand() % 9;
67. r.digits[r.pointPos - 1] = '.';
68. **for**(i = r.length; i < MAX\_REAL\_WIDTH; i ++ ){
69. r.digits[i] = '\0';
70. }
72. printf("Genejated(%2d.%02d):",r.length, r.length - r.pointPos );
73. displayFixedWidthNumber( r.digits, MAX\_REAL\_WIDTH);
75. **return** r;
76. }
78. Real addTowLongReal(Real r1, Real r2) {
79. Real result;
80. memset(result.digits, 0, MAX\_REAL\_WIDTH);
82. **if** (r2.length - r2.pointPos > r1.length - r1.pointPos) {
83. Real temp = r1;
84. r1 = r2;
85. r2 = temp;
86. }

89. **if**(r1.length - r1.pointPos + r2.pointPos > MAX\_REAL\_WIDTH)
90. {
91. result.length = -1;
92. **return** result;
93. }
95. r1 = shiftDigitsToRight( r1, MAX\_REAL\_WIDTH - r1.length);
96. **int** shiftLength2 = r1.pointPos - r2.pointPos;
97. r2 = shiftDigitsToRight( r2, shiftLength2);
99. result.pointPos = r1.pointPos;
101. **int** carry = 0;
102. **int** i;
104. //从后向前相加
105. **for** (i = MAX\_REAL\_WIDTH - 1; i >= 0; i--) {
106. **if** (i == r1.pointPos - 1){
107. **continue**;
108. }
109. **if** (r1.digits[i] == 0 && r2.digits[i] == 0 && carry == 1){
110. result.digits[i] = '1';
111. carry = 0;
112. }**else** **if**( r2.digits[i] == 0 && carry == 0){
113. result.digits[i] = r1.digits[i];
114. }**else** **if**( r2.digits[i] == 0 && carry == 1 && r1.digits[i] != '9'){
115. result.digits[i] = r1.digits[i] + 1;
116. carry = 0;
117. }**else** **if**( r2.digits[i] == 0 && carry == 1 && r1.digits[i] == '9'){
118. result.digits[i] = '0';
119. carry = 1;
120. }**else** **if**( r1.digits[i] == 0 && carry == 0){
121. result.digits[i] = r2.digits[i];
122. }**else** **if**( r1.digits[i] == 0 && carry == 1 && r2.digits[i] != '9'){
123. result.digits[i] = r2.digits[i] + 1;
124. carry = 0;
125. }**else** **if**( r1.digits[i] == 0 && carry == 1 && r2.digits[i] == '9'){
126. result.digits[i] = '0';
127. carry = 1;
128. }**else** **if**(r1.digits[i] - '0' + r2.digits[i] - '0' + carry >= 10){
129. result.digits[i] = (r1.digits[i] - '0' + r2.digits[i] - '0' + carry) % 10 + '0';
130. carry = 1;
131. }**else** **if**(r1.digits[i] - '0' + r2.digits[i] - '0' + carry < 10){
132. result.digits[i] = (r1.digits[i] - '0' + r2.digits[i] - '0' + carry) + '0';
133. carry = 0;
134. }**else** **if**(r1.digits[i] == 0 && r2.digits[i] == 0 && carry == 1){
135. result.digits[i] = '1';
136. }
137. }
139. // 补小数点
140. result.digits[r1.pointPos - 1] = '.';
142. **for**(i = 0; i < MAX\_REAL\_WIDTH; i ++)
143. {
144. **if**(result.digits[i] != 0)
145. {
146. result.length = MAX\_REAL\_WIDTH - i;
147. **break**;
148. }
149. }
151. printf("SumResult(%2d.%2d):",result.length, MAX\_REAL\_WIDTH - result.pointPos);
152. displayFixedWidthNumber(result.digits,MAX\_REAL\_WIDTH);
154. // 溢出判断
155. **if** (carry > 0 && result.length == MAX\_REAL\_WIDTH) {
156. result.length = -1;
157. **return** result;
158. }
160. r1 = shiftDigitsToLeft( r1, MAX\_REAL\_WIDTH - r1.length);
161. r2 = shiftDigitsToLeft( r2, shiftLength2);
162. result = shiftDigitsToLeft( result, MAX\_REAL\_WIDTH - result.length);
164. **return** result;
165. }

168. **void** displayAssignedLongReal(**int** pointPos, Real r) {
169. **int** i;
170. **for** (i = 0; i < pointPos - r.pointPos; i++) {
171. printf(" ");
172. }
173. **for** (i = 0; i < r.length; i++) {
174. printf("%c", r.digits[i]);
175. }
176. printf("\n");
177. **return**;
178. }
180. **void** displayFixedWidthNumber( **char** number[], **const** **int** width ){
181. **int** i;
182. **for**(i = 0; i < width; i ++){
183. **if**( number[i] == '\0'){
184. printf("\_");
185. }
186. **else**{
187. printf("%c",number[i]);
188. }
189. }
191. printf("\n");
192. **return**;
193. }
195. Real shiftDigitsToRight( Real r, **int** shiftLength)
196. {
197. **if**(shiftLength == 0)
198. {
199. printf("ShiftRight (%2d) :",shiftLength);
200. displayFixedWidthNumber( r.digits, MAX\_REAL\_WIDTH);
201. **return** r;
202. }
203. **int** i;
204. r.pointPos += shiftLength;
205. **for**(i = r.length - 1; i >= 0; i --){
206. r.digits[i + shiftLength] = r.digits[i];
207. r.digits[i] = '\0';
208. }
209. printf("ShiftRight (%2d) :",shiftLength);
210. displayFixedWidthNumber( r.digits, MAX\_REAL\_WIDTH);
211. **return** r;
212. }
214. Real shiftDigitsToLeft( Real r, **int** shiftLength)
215. {
216. **if**(shiftLength == 0)
217. {
218. printf("ShiftLeft  (%2d) :",shiftLength);
219. displayFixedWidthNumber( r.digits, MAX\_REAL\_WIDTH);
220. **return** r;
221. }
222. **int** i;
223. r.pointPos -= shiftLength;
224. **for**(i = shiftLength; i < MAX\_REAL\_WIDTH; i ++){
225. r.digits[i - shiftLength] = r.digits[i];
226. r.digits[i] = '\0';
227. }
228. printf("ShiftLeft  (%2d) :",shiftLength);
229. displayFixedWidthNumber( r.digits, MAX\_REAL\_WIDTH);
230. **return** r;
231. }

三、实验三

（1）main.c

1. // 浮动点小数相加,指针操作返回值
2. // scyk
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7. #include <time.h>
9. #define MAX\_REAL\_WIDTH  25
11. **typedef** **struct** {
12. **char** digits[MAX\_REAL\_WIDTH];
13. **int** length;
14. **int** pointPos;
15. } Real;
17. // 随机生成数字
18. Real\* generateLongReal(**void**);
20. // 相加
21. Real\* addTowLongReal(Real\* r1, Real\* r2);
23. // 打印数字
24. **void** displayAssignedLongReal(**int** pointPos, Real\* r);
26. // 右移
27. Real\* shiftDigitsToRight(Real\* r, **int** shiftLength);
29. // 左移
30. Real\* shiftDigitsToLeft(Real\* r, **int** shiftLength);
32. // 用 '\0' 补位
33. **void** displayFixedWidthNumber(**char** number[], **const** **int** width);
35. **int** main() {
36. srand(time(NULL));
38. Real\* real1 = generateLongReal();
39. Real\* real2 = generateLongReal();
40. Real\* realSum = addTowLongReal(real1, real2);
42. **if** (realSum->length > 0) {
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. printf("\nError! addition overflow!\n");
52. }
54. // 释放内存
55. free(real1);
56. free(real2);
57. free(realSum);
59. **return** 0;
60. }
62. Real\* generateLongReal(**void**) {
63. Real\* r = (Real\*)malloc(**sizeof**(Real));
64. r->pointPos = 2 + rand() % 23;
65. **int** fractionLength = rand() % (MAX\_REAL\_WIDTH - r->pointPos + 1);
66. r->length = r->pointPos + fractionLength;
68. **int** i;
69. r->digits[0] = '1' + rand() % 9;
70. **for** (i = 1; i < r->length - 1; i++) {
71. r->digits[i] = '0' + rand() % 10;
72. }
73. r->digits[r->length - 1] = '1' + rand() % 9;
74. r->digits[r->pointPos - 1] = '.';
76. **for** (i = r->length; i < MAX\_REAL\_WIDTH; i++) {
77. r->digits[i] = '\0';
78. }
80. printf("Generated(%2d.%02d):", r->length, r->length - r->pointPos);
81. displayFixedWidthNumber(r->digits, MAX\_REAL\_WIDTH);
83. **return** r;
84. }
86. Real\* addTowLongReal(Real\* r1, Real\* r2) {
87. Real\* result = (Real\*)malloc(**sizeof**(Real));
88. memset(result->digits, 0, MAX\_REAL\_WIDTH);
90. **if** (r2->length - r2->pointPos > r1->length - r1->pointPos) {
91. Real\* temp = r1;
92. r1 = r2;
93. r2 = temp;
94. }
96. **if** (r1->length - r1->pointPos + r2->pointPos > MAX\_REAL\_WIDTH) {
97. result->length = -1;
98. **return** result;
99. }
101. r1 = shiftDigitsToRight(r1, MAX\_REAL\_WIDTH - r1->length);
102. **int** shiftLength2 = r1->pointPos - r2->pointPos;
103. r2 = shiftDigitsToRight(r2, shiftLength2);
105. result->pointPos = r1->pointPos;
107. **int** carry = 0;
108. **int** i;
110. **for** (i = MAX\_REAL\_WIDTH - 1; i >= 0; i--) {
111. **if** (i == r1->pointPos - 1) {
112. **continue**;
113. }
114. **if** (r1->digits[i] == 0 && r2->digits[i] == 0 && carry == 1) {
115. result->digits[i] = '1';
116. carry = 0;
117. }
118. **else** **if** (r2->digits[i] == 0 && carry == 0) {
119. result->digits[i] = r1->digits[i];
120. }
121. **else** **if** (r2->digits[i] == 0 && carry == 1 && r1->digits[i] != '9') {
122. result->digits[i] = r1->digits[i] + 1;
123. carry = 0;
124. }
125. **else** **if** (r2->digits[i] == 0 && carry == 1 && r1->digits[i] == '9') {
126. result->digits[i] = '0';
127. carry = 1;
128. }
129. **else** **if** (r1->digits[i] == 0 && carry == 0) {
130. result->digits[i] = r2->digits[i];
131. }
132. **else** **if** (r1->digits[i] == 0 && carry == 1 && r2->digits[i] != '9') {
133. result->digits[i] = r2->digits[i] + 1;
134. carry = 0;
135. }
136. **else** **if** (r1->digits[i] == 0 && carry == 1 && r2->digits[i] == '9') {
137. result->digits[i] = '0';
138. carry = 1;
139. }
140. **else** **if** (r1->digits[i] - '0' + r2->digits[i] - '0' + carry >= 10) {
141. result->digits[i] = (r1->digits[i] - '0' + r2->digits[i] - '0' + carry) % 10 + '0';
142. carry = 1;
143. }
144. **else** **if** (r1->digits[i] - '0' + r2->digits[i] - '0' + carry < 10) {
145. result->digits[i] = (r1->digits[i] - '0' + r2->digits[i] - '0' + carry) + '0';
146. carry = 0;
147. }
148. **else** **if** (r1->digits[i] == 0 && r2->digits[i] == 0 && carry == 1) {
149. result->digits[i] = '1';
150. }
151. }
153. result->digits[r1->pointPos - 1] = '.';
155. **for** (i = 0; i < MAX\_REAL\_WIDTH; i++) {
156. **if** (result->digits[i] != 0) {
157. result->length = MAX\_REAL\_WIDTH - i;
158. **break**;
159. }
160. }
162. printf("SumResult(%2d.%2d):", result->length, MAX\_REAL\_WIDTH - result->pointPos);
163. displayFixedWidthNumber(result->digits, MAX\_REAL\_WIDTH);
165. **if** (carry > 0 && result->length == MAX\_REAL\_WIDTH) {
166. result->length = -1;
167. **return** result;
168. }
170. r1 = shiftDigitsToLeft(r1, MAX\_REAL\_WIDTH - r1->length);
171. r2 = shiftDigitsToLeft(r2, shiftLength2);
172. result = shiftDigitsToLeft(result, MAX\_REAL\_WIDTH - result->length);
174. **return** result;
175. }
177. **void** displayAssignedLongReal(**int** pointPos, Real\* r) {
178. **int** i;
179. **for** (i = 0; i < pointPos - r->pointPos; i++) {
180. printf(" ");
181. }
182. **for** (i = 0; i < r->length; i++) {
183. printf("%c", r->digits[i]);
184. }
185. printf("\n");
186. **return**;
187. }
189. **void** displayFixedWidthNumber(**char** number[], **const** **int** width) {
190. **int** i;
191. **for** (i = 0; i < width; i++) {
192. **if** (number[i] == '\0') {
193. printf("\_");
194. }
195. **else** {
196. printf("%c", number[i]);
197. }
198. }
199. printf("\n");
200. **return**;
201. }
203. Real\* shiftDigitsToRight(Real\* r, **int** shiftLength) {
204. **if** (shiftLength == 0) {
205. printf("ShiftRight (%2d) :", shiftLength);
206. displayFixedWidthNumber(r->digits, MAX\_REAL\_WIDTH);
207. **return** r;
208. }
209. **int** i;
210. r->pointPos += shiftLength;
211. **for** (i = r->length - 1; i >= 0; i--) {
212. r->digits[i + shiftLength] = r->digits[i];
213. r->digits[i] = '\0';
214. }
215. printf("ShiftRight (%2d) :", shiftLength);
216. displayFixedWidthNumber(r->digits, MAX\_REAL\_WIDTH);
217. **return** r;
218. }
220. Real\* shiftDigitsToLeft(Real\* r, **int** shiftLength) {
221. **if** (shiftLength == 0) {
222. printf("ShiftLeft (%2d) :", shiftLength);
223. displayFixedWidthNumber(r->digits, MAX\_REAL\_WIDTH);
224. **return** r;
225. }
226. **int** i;
227. r->pointPos -= shiftLength;
228. **for** (i = shiftLength; i < MAX\_REAL\_WIDTH; i++) {
229. r->digits[i - shiftLength] = r->digits[i];
230. r->digits[i] = '\0';
231. }
232. printf("ShiftLeft (%2d) :", shiftLength);
233. displayFixedWidthNumber(r->digits, MAX\_REAL\_WIDTH);
234. **return** r;
235. }