

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 | 徐诚 | 学号 | U202413383 |

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

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

# 实验目的

完成xxx系列代码（日历系列、大数计算系列，选择其中之一）。

# 实验环境

操作系统：Windows 10

编程工具：CodeBlocks 16.01

# 实验一

## 实验任务

浮动点小数相加（指针操作动态内存），实现无最大位宽参数的动态数组

需要完成的函数：

//创建结构体长实数

Real \*generateLongReal ( void );

//释放内存

void destroyLongReal ( Real \*rPtr );

//将两实数相加

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

//打印长实数

void displayAssignedLongReal ( const Real \*rPtr );

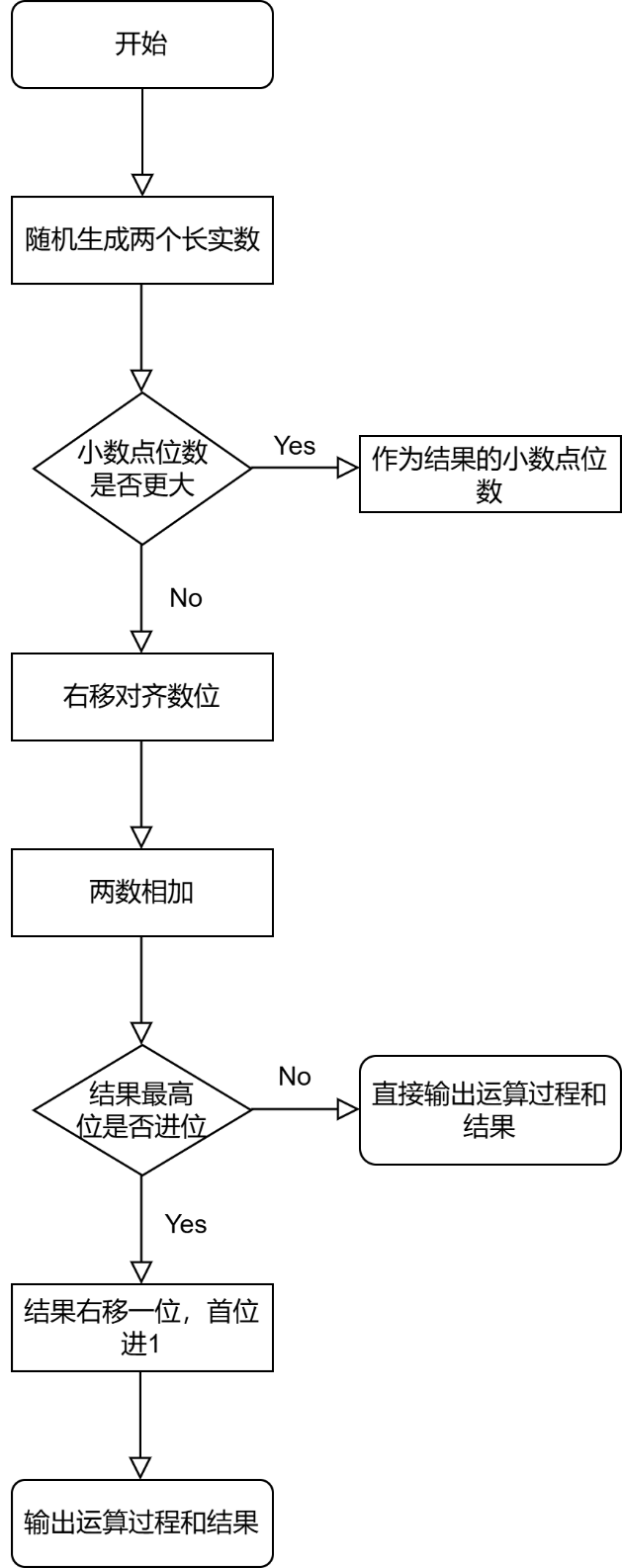
//对长实数进行右对齐操作

void shiftDigitsToRight ( Real \*rPtr, int shiftLength );

//对长实数进行左对齐操作

void shiftDigitsToLeft ( Real \*rPtr, int shiftLength );

## 实验步骤



## 代码测试

### 测试点 xxx的测试结果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试点 | 测试步骤 | 预期结果 | 实际结果 | 测试结论 |
| 随机数长度和小数点位数是否随机 | 编制generate函数并运行 | 长度和位数均随机 |  | 测试通过 |
| shiftRight函数能否正常对齐 | 打印对齐后实数与前面数字比较 | 小数点正常对齐 |  | 测试通过 |
| 相加结果是否正确（包括首位进位） | 生成多组实数查看相加结果 | 求和正确，进位正确，首位能自动扩展数位 |  | 测试通过 |
|  |  |  |  |  |
|  |  |  |  |  |

## 实验结论

代码能够达到功能目标

## 实验总结

1. 在generateLongReal函数中忘记返回结构体指针，打印出来都是乱码。（不是第一次犯了）
2. 一开始两数相加的时候每次循环直接令SumDigits = digit1 + digi2 –‘0’，导致记录了进位操作但是打印结果都没有进位，后面改为SumDigits += digit1 + digit2 – ‘0’.
3. 检验shiftLeft函数之后忘记右移回去，打印计算过程的时候小数点又没有对齐
4. 思路上，一开始没有右移实数，想直接通过加减每个字符的数位来对齐求和，后来打出一堆乱码放弃了。

# 本课程学习总结

以前从来没有接触过编程语言，学了本学期的C语言，感觉就像打怪升级，捡装备闯关一样，从第一节课只会printf输出到后面能输入，控制循环，数组结构体指针，每次看到窗口里面正确的结果就成就满满，但碰到一个小错误调试半个小时都没看出来也是蛮烦人的，（尤其是if语句里面“=”和“==”，感觉错了无数次，每次都要找半天才找出来，包括这个大数相加的实验也错了一遍）。

然后就是编程卡壳的时候，有时候想上网查一下又想自己多想一下，后面就觉得卡壳也有知识上的卡壳和逻辑上的卡壳，知识上的卡壳就比如说求两个数最大质因数，我连用自然语言怎么表达都不会，那编起来就无从下手，我觉得这种可以搜一下，记住思路以后就直接用了；逻辑上的卡壳就比如我编个月历，什么时候要换行，换行之后要记录，周数要加1这些思路很清楚能说出来，但一次两次编不成功，总有小问题，这种就自己还是要多想想，就跟排序一样，知道是找出最小的拎出来，逻辑上还是要自己过一遍。我最开始编程的时候就无论怎样不搜，自己想个一两个小时都想不出来，后面就顺从了。

代码规范上面还是得多敲多练，这个感觉得形成肌肉记忆才行，敲多了这个大拇指就随时准备按空格了，空行还是经常会忘记，注释几乎都是逼着自己憋出来的。每次功能实现完了都要从头看到尾再规范一遍。代码调试的时候我这个习惯不太好，喜欢从第一行逐行的看，每次功能出错了都从头调试，没什么重点，有时候能检查出更多的错误，大部分情况还是浪费了很多时间。

复习的时候再看前面学过的知识理解还是会更深一些，有些当时不怎么懂的，回头再看看也许就懂了。代码也是常看常新，翻到自己没学数组时候编的日历编了两百多行。

最后学习C语言之路还是任重道远，之前让AI用C语言编了一下贪吃蛇，俄罗斯方块这些，复制之后直接跑通了，感觉挺厉害的，自己要学的东西还是挺多的。

# 附录

1. 大数相加实验
2. #include <stdio.h>
3. #include <stdlib.h>
4. #include <time.h>
6. #define MAX\_REAL\_WIDTH 25
8. **typedef** **struct**{
9. **char** \*digits;
10. **int** pointPos;
11. **int** length;
12. }Real;
14. //创建结构体长实数
15. Real \*generateLongReal ( **void** );
17. //释放内存
18. **void** destroyLongReal ( Real \*rPtr );
20. //将两实数相加
21. Real \*addTwoLongReals ( **const** Real \*rPtr1, **const** Real \*rPtr2 );
23. //打印长实数
24. **void** displayAssignedLongReal ( **const** Real \*rPtr );
26. //对长实数进行右对齐操作
27. **void** shiftDigitsToRight ( Real \*rPtr, **int** shiftLength );
29. //对长实数进行左对齐操作
30. **void** shiftDigitsToLeft ( Real \*rPtr, **int** shiftLength );

33. **int** main ( **void** )
34. {
35. srand ( time(NULL) );
36. Real \*rPtr1 = generateLongReal();
37. Real \*rPtr2 = generateLongReal();
38. Real \*rPtrSum = addTwoLongReals( rPtr1, rPtr2 );
40. printf("\n  ");
41. displayAssignedLongReal( rPtr1 );
42. printf("+ ");
43. displayAssignedLongReal( rPtr2 );
44. printf("= ");
45. displayAssignedLongReal( rPtrSum );
46. printf("\n");
48. destroyLongReal ( rPtr1 );
49. destroyLongReal ( rPtr2 );
50. destroyLongReal ( rPtrSum );
52. **return** 0;
53. }
55. Real \*generateLongReal ( **void** )
56. {
57. //为结构体分配内存
58. Real \*rPtr = ( Real\* )malloc( **sizeof** ( Real ) );
60. //随机产生长度和小数点位置
61. rPtr->length = 2 + rand() % ( MAX\_REAL\_WIDTH - 1 );
62. rPtr->pointPos = 1 + rand() % ( rPtr->length );
64. //为字符成员分配内存
65. rPtr->digits = (**char**\*) calloc( rPtr->length, **sizeof**(**char**) );
66. rPtr->digits[0] = '1' + rand() % 9;
68. //随机产生字符
69. **int** i;
70. **for** ( i = 1; i < rPtr->length -1; i ++ )
71. {
72. rPtr->digits[i] = '0' + rand() % 10;
73. }
74. rPtr->digits[rPtr->length - 1] = '1' + rand() % 9;
75. rPtr->digits[rPtr->pointPos - 1] = '.';
77. printf("Randomize (%2d.%2d): ",rPtr->length,rPtr->length - rPtr->pointPos );
78. displayAssignedLongReal( rPtr );
80. **return** rPtr;
81. }
83. **void** displayAssignedLongReal ( **const** Real \*rPtr )
84. {
85. **int** i;
86. **for** ( i = 0; i < rPtr->length; i ++ )
87. {
88. printf("%c",rPtr->digits[i]);
89. }
90. printf("\n");
91. }
93. Real \*addTwoLongReals ( **const** Real \*rPtr1, **const** Real \*rPtr2 )
94. {
95. Real \*sum = ( Real\* ) malloc( **sizeof**(Real) );
97. //判断小数点位数最大的实数作为结果的小数点位数
98. sum->pointPos = ( rPtr1->pointPos > rPtr2->pointPos ) ? rPtr1->pointPos : rPtr2->pointPos;
99. sum->length = sum->pointPos + ( ( rPtr1->length - rPtr1->pointPos ) > ( rPtr2->length - rPtr2->pointPos ) ?
100. ( rPtr1->length - rPtr1->pointPos ) : ( rPtr2->length - rPtr2->pointPos ) );
101. sum->digits = ( **char**\* ) calloc( sum->length , **sizeof**(**char**) );
103. **int** i;
105. //定义长实数右移或左移的长度
106. **int** shift1 = sum->pointPos - rPtr1->pointPos,
107. shift2 = sum->pointPos - rPtr2->pointPos,
108. shift0 = 0;//标记，用来判断最高位是否进位
110. //对需要右移的实数进行操作
111. **if** ( shift1 > 0 )
112. {
113. shiftDigitsToRight ( rPtr1, shift1 );
114. printf("shiftRight(%2d)   : ", shift1);
115. displayAssignedLongReal( rPtr1 );
116. }
117. **if** ( shift2 > 0 )
118. {
119. shiftDigitsToRight ( rPtr2, shift2 );
120. printf("shiftRight(%2d)   : ", shift2);
121. displayAssignedLongReal( rPtr2 );
122. }
124. //记录两实数对应数位的数字，相加
125. **char** digit1,digit2;
127. //从最后一位开始相加
128. **for** ( i = sum->length - 1; i >= 0; i -- )
129. {
130. //若长实数在该位没有数字，记为‘0’
131. **if** ( i > rPtr1->length - 1 )
132. {
133. digit1 = '0';
134. }
135. **else**
136. {
137. digit1 = rPtr1->digits[i];
138. **if** ( digit1 == ' ' )
139. {
140. digit1 = '0';
141. }
142. }
144. **if** ( i > rPtr2->length - 1 )
145. {
146. digit2 = '0';
147. }
148. **else**
149. {
150. digit2 = rPtr2->digits[i];
151. **if** ( digit2 == ' ' )
152. {
153. digit2 = '0';
154. }
155. }
157. sum->digits[i] += digit1 + digit2 - '0';
159. //进行进位操作
160. **if**(sum->digits[i] > '9')
161. {
162. sum->digits[i] -= 10;
163. **if**( i != sum->pointPos )
164. {
165. sum->digits[i-1] += 1;
166. }
167. **else** **if**( i == sum->pointPos )
168. {
169. sum->digits[i-2] += 1;
170. }
171. //记录最高位进位，方便后续操作
172. **if** ( i == 0 )
173. {
174. shift0 ++;
175. }
176. }
177. }
178. sum->digits[sum->pointPos - 1] = '.';
180. //检测到最高位进位，将结果右移，将最高位赋为‘1’
181. **if** ( shift0 != 0 )
182. {
183. shiftDigitsToRight( sum, shift0 );
184. sum->digits[0] = '1';
185. }
187. printf("Sum Result(%2d.%2d): ", sum->length,sum->pointPos);
188. displayAssignedLongReal( sum );
190. //将右移过的长实数左移回来，检测shiftDigitsToLeft函数功能是否正常
191. **if** ( shift1 > 0 )
192. {
193. shiftDigitsToLeft ( rPtr1, shift1 );
194. printf("shiftLeft (%2d)   : ", shift1);
195. displayAssignedLongReal( rPtr1 );
197. //左移后右移回去，避免影响加法过程的展示，下同
198. shiftDigitsToRight ( rPtr1, shift1 );
199. }
200. **if** ( shift2 > 0 )
201. {
202. shiftDigitsToLeft ( rPtr2, shift2 );
203. printf("shiftLeft (%2d)   : ", shift2);
204. displayAssignedLongReal( rPtr2 );
205. shiftDigitsToRight ( rPtr2, shift2 );
206. }
208. **return** sum;
209. }
211. **void** shiftDigitsToRight ( Real \*rPtr, **int** shiftLength )
212. {
213. //调整内存区的大小
214. rPtr->digits = (**char** \*)realloc(rPtr->digits,(rPtr->length + shiftLength)\***sizeof**(**char**));
216. //右移赋值操作
217. **int** i;
218. **for** ( i = rPtr->length - 1; i >= 0; i -- )
219. {
220. rPtr->digits[i+shiftLength] = rPtr->digits[i];
221. }
223. //将前面空缺的数位赋为‘ ’
224. **for** ( i = 0; i < shiftLength; i ++ )
225. {
226. rPtr->digits[i] = ' ';
227. }
229. //调整实数的长度和小数点位置
230. rPtr->length += shiftLength;
231. rPtr->pointPos += shiftLength;
233. }
235. **void** shiftDigitsToLeft ( Real \*rPtr, **int** shiftLength )
236. {
237. **int** i;
238. **for** ( i = 0; i < rPtr->length - shiftLength; i ++ )
239. {
240. rPtr->digits[i] = rPtr->digits[i+shiftLength];
241. }
242. **for** ( i = rPtr->length - shiftLength; i < rPtr->length; i ++ )
243. {
244. rPtr->digits[i] = ' ';
245. }
246. rPtr->length -= shiftLength;
247. rPtr->pointPos -= shiftLength;
249. }
251. **void** destroyLongReal ( Real \*rPtr )
252. {
253. **if** ( rPtr->digits != NULL )
254. {
255. free ( rPtr->digits );
256. }
257. free( rPtr );
259. }