

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 | 谭盛晞 | 学号 | U202411086 |

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 2024.1.17 | 地点 | 华中科技大学 |

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

# 实验目的

完成大数据计算系列代码附加一个日历代码

# 实验环境

操作系统：Windows 10

编程工具：CodeBlocks 16.01

# 实验一

## 实验任务

– **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**

– **随机产生两个长整数，约定其最高位不得为‘0’**

– **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**

– **打印两个长整数求和的竖式计算过程**

**完成addTwoLonInts函数**

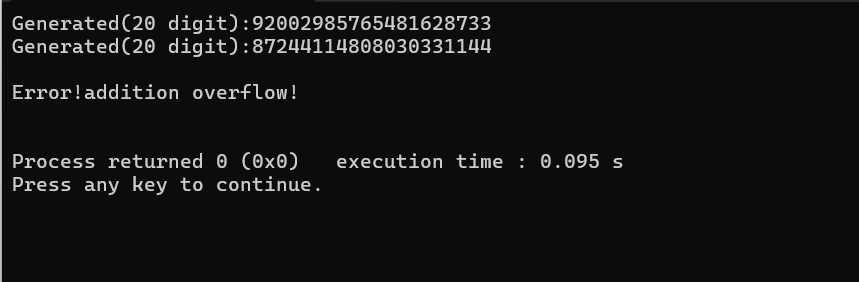
## 实验步骤

用isOverflow储存相加结果是否溢出

用int型的sum变量储存两个长整数每一位相加的和，用carry记录是否进位，并最后通过carry判断是否溢出

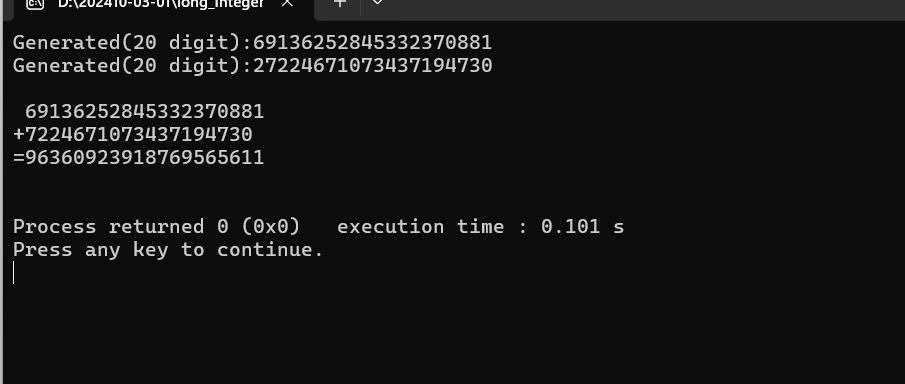
## 代码测试

### 测试点相加溢出（预期报错）的测试结果



正确报错

测试点相加未溢出（预期显示相加结果）的测试结果



正确相加

## 实验结论

代码达到预期功能

## 实验总结

良好

# 实验二

## 实验任务

– **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‘.’**

– **随机产生两个长实数，约定其最高位不得为‘0’**

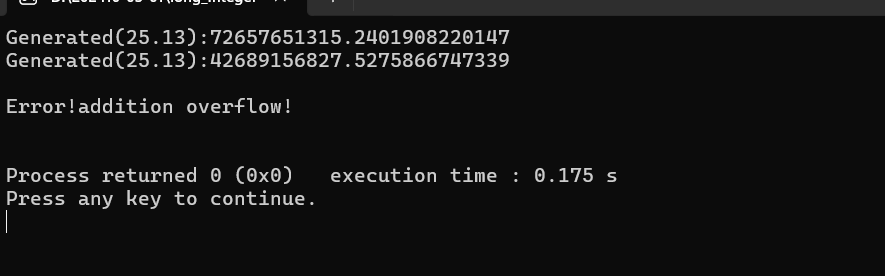
– **对两个相同长度的长实数进行求和，有加法进位时显示报错信息**

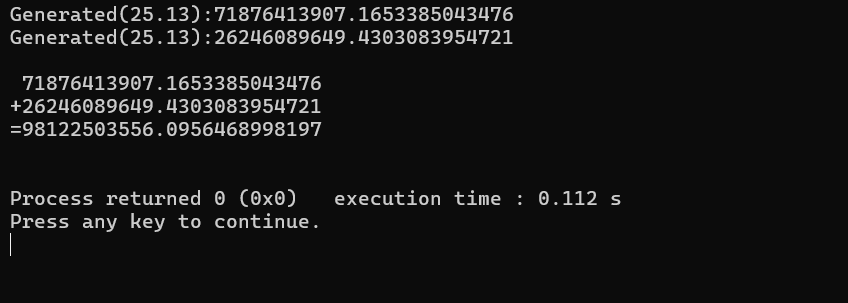
– **打印两个长实数求和的竖式计算过程**

## 实验步骤

步骤与实验一类似，只是需要分别相加小数部分与整数部分，相当于进行了两次两个长整数的相加。

4.3 代码测试





## 4.4实验结论

合格

## 4.5 实验总结

良好

# 实验三

## 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**

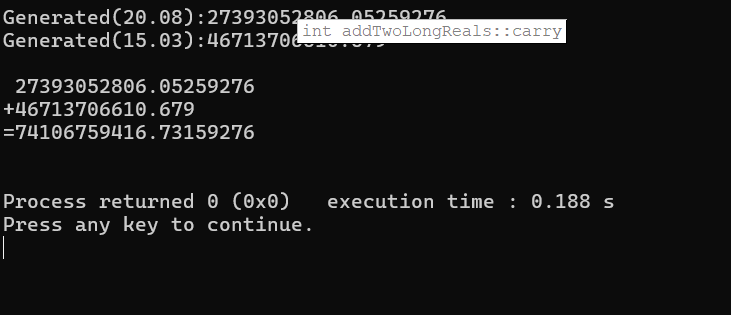
– **对两个整数部分长度相同的长实数进行求和，有加法进位时显示报错信息**

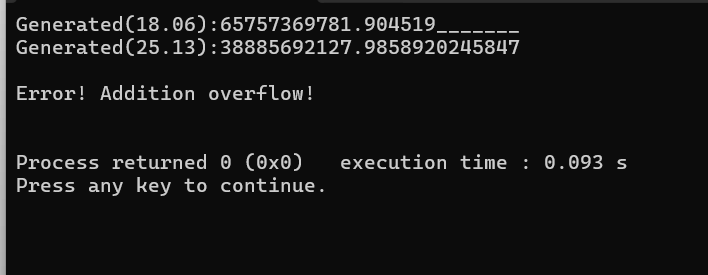
– **打印两个长整数求和的竖式计算过程**

## 实验步骤

步骤依然与实验一相似，只是要考虑随机生成小数部分的位数。

5.3 实验测试





## 5.4实验结论

合格

## 5.5 实验总结

良好

# 实验四

## 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**

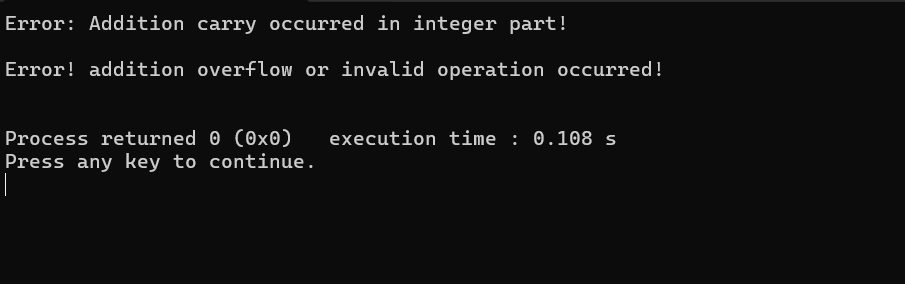
– **对两个整数部分长度不同的长实数进行求和，有加法进位时显示报错信息**

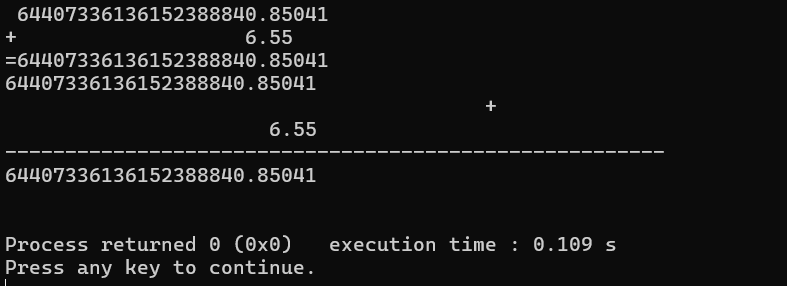
– **打印两个长实数求和的竖式计算过程，需要根据小数点位置不同调整输出格式**

## 实验步骤

先将小数对整齐再相加，利用对其函数是浮动小数点变成固定小数点的情况。

6.3 实验测试





Add函数编写错误导致结果不正确

# 实验五

# 7.1实验目的

**预期功能与前一个实验相同，要求修改对应的数组操作的语句，通过指针访问数组元素**

7.2实验步骤

引入指针变量操作数组，但遇到了与上个实验相同的问题，这一次代码无法运行。

7.3实验测试

# 实验六

# 8.1实验目的

**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**

8.2实验步骤

创建结构体，步骤与数组类似。

8.3实验测试

代码无法运行

# 实验七

# 9.1实验目的

**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**

9.2实验步骤

在上一个实验的基础上加入malloc函数，提高内存利用率。

9.3实验测试

代码无法运行。

# 实验八

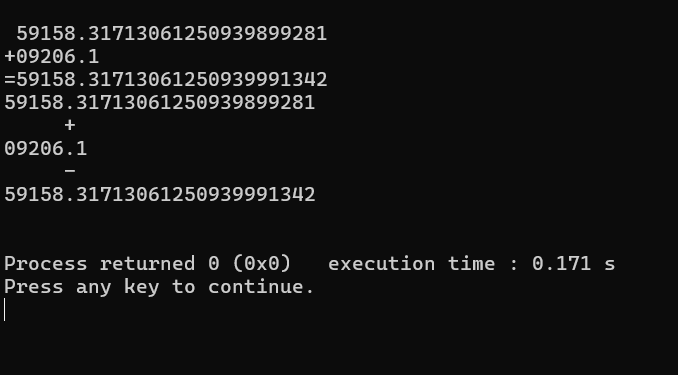
# 10.1实验目的

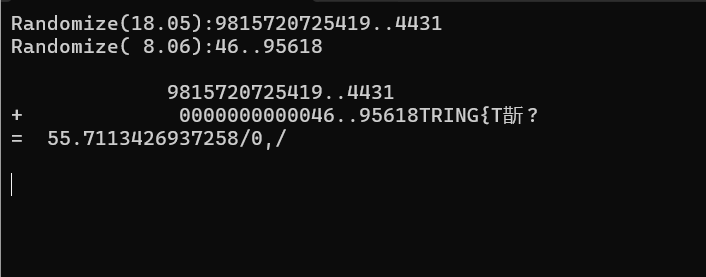
**前一个实验中需要固定宽度数组的部分，改写为无最大位宽参数的动态数组**

10.2实验步骤

彻底解决溢出的问题，**存储数位不足时自动右移申请新内存，支持自动扩展数位。**

10.3实验测试





Add函数编写失败导致结果出现乱码

# 本课程学习总结

学习C语言的历程和总结（心酸血泪史）

其实整个学期的C语言课都没有听懂，并且学到结构体的时候实在是太痛苦了，就欠下了很多作业，后来问了高中室友，她推荐我找一本入门的书看完，并自己多敲敲代码，我照做后补起来一些，但是还是感觉很多地方都不会。感觉学好C语言的秘诀就是多敲代码，可惜我没有这个毅力，所以始终在挣扎，希望寒假每天编一编能让手感变好。

程序中出现的问题和改进

一些简单的语法错误似乎是最难检查的，如for循环中条件要不要取等，==是否被误写为=，我经常犯这一类错误，现在习惯在草稿纸上演算一下再敲代码。

代码规范与调试技巧

每写完一个函数都尽量设置一个调试节点，这样方便最后查看，这样做比最后写完却始终de不出bug要划算得多。

学习记录和心得

说实话大一上的C学的非常痛苦也非常迷茫，总有种无可奈何的感觉，说是现在已经喜欢上这门课当然是谎话，但是不能放弃的事情，只好调节心态让自己持续输出能量，可能某一天就会好起来吧。

# 附录

完整实验代码附在此处

1. 实验1
2. main.c
3. #include <stdio.h>
4. #include <stdlib.h>
5. #include <time.h>
6. #define INT\_WIDTH 20
7. //Generate a random long integer
8. void generateLongInt(char longInt[]);
9. //Sum two long integers
10. int addTwoLongInts(char longInt1[],char longInt2[],char longIntSum[]);
11. //Display the long integer in a single line
12. void displayLongInt(char longInt[]);
13. int main(void)
14. {
15. srand(time(NULL));
16. char longInt1[INT\_WIDTH ]={0};generateLongInt(longInt1);
17. char longInt2[INT\_WIDTH ]={0};generateLongInt(longInt2 );
18. char longIntSum[INT\_WIDTH ]={0};
19. int isOverflow =addTwoLongInts(longInt1,longInt2,longIntSum);
20. if(isOverflow ==0){
21. printf("\n ");
22. displayLongInt(longInt1);printf("+");
23. displayLongInt(longInt2 );printf("=");
24. displayLongInt(longIntSum );
25. }
26. else if(isOverflow ==1)
27. {
28. printf("\nError!addition overflow!\n");
29. }
30. printf("\n");
31. return 0;
32. }
33. void generateLongInt(char longInt[]){
34. int i;
35. longInt[0]='1'+rand()%9;
36. for(i=1;i<INT\_WIDTH;i++)
37. {
38. longInt[i]+='0'+rand()%10;
39. }
40. printf("Generated(%d digit):",INT\_WIDTH);
41. displayLongInt(longInt );
42. return;
43. }
44. int addTwoLongInts(char longInt1[], char longInt2[], char longIntSum[])
45. {
46. int isOverflow = 0;
47. int carry = 0;
48. int i;
49. for (i = INT\_WIDTH - 1; i >= 0; i--)
50. {
51. int sum = longInt1[i] - '0' + longInt2[i] - '0' + carry;
52. longIntSum[i] = (sum % 10) + '0';
53. carry = sum / 10;
54. }
55. if (carry > 0)
56. {
57. isOverflow = 1;
58. }
59. longIntSum[INT\_WIDTH] = '\0'; // Null terminator
60. return isOverflow;
61. }
62. void displayLongInt(char longInt[])
63. {
64. int i;
65. for(i=0;i<INT\_WIDTH;i++){
66. printf("%C",longInt[i]);
67. }
68. printf("\n");
69. return;
70. }
71. 实验2

（1）main.c

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define REAL\_WIDTH 25

#define POINT\_POSITION 12

//Generate a random long real number

void generateLongReal(char longReal[]);

//sum two long real numbers

int addTwoLongReals(char longReal1[],char longReal2[],char longRealSum[]);

//Display the long integer in a single line

void displayLongReal(char longReal[]);

int main(void)

{

srand(time(NULL));

char longReal1[REAL\_WIDTH]={0};

generateLongReal(longReal1);

char longReal2[REAL\_WIDTH]={0};

generateLongReal(longReal2);

char longRealSum[REAL\_WIDTH ]={0};

int isOverflow =addTwoLongReals(longReal1,longReal2,longRealSum );

if(isOverflow == 0)

{

printf("\n ");

displayLongReal(longReal1);printf("+");

displayLongReal(longReal2);printf("=");

displayLongReal(longRealSum );

}

else if(isOverflow ==1)

{

printf("\nError!addition overflow!\n");

}

printf("\n");

return 0;

}

void generateLongReal(char longReal[])

{

int i;

longReal[0]='1'+rand()%9;

for(i=1;i<REAL\_WIDTH;i++)

{

longReal[i]='0'+rand()%10;

}

longReal[POINT\_POSITION-1]='.';

printf("Generated(%d.%d):",REAL\_WIDTH,REAL\_WIDTH-POINT\_POSITION);

displayLongReal(longReal );

return;

}

int addTwoLongReals(char longReal1[], char longReal2[], char longRealSum[])

{

int i;

int sum;

int carry = 0;

int isOverflow = 0;

// Initialize longRealSum with zeros

for (i = 0; i < REAL\_WIDTH; i++)

{

longRealSum[i] = '0';

}

// Add the fractional part

for (i = REAL\_WIDTH - 1; i > POINT\_POSITION; i--)

{

sum = (longReal1[i] - '0') + (longReal2[i] - '0') + carry;

longRealSum[i] = (sum % 10) + '0';

carry = sum / 10;

}

// Add the integer part

for (i = POINT\_POSITION - 1; i >= 0; i--)

{

sum = (longReal1[i] - '0') + (longReal2[i] - '0') + carry;

longRealSum[i] = (sum % 10) + '0';

carry = sum / 10;

}

// Handle the carry after the last digit

if (carry == 1)

{

isOverflow = 1;

}

// Place the decimal point

longRealSum[POINT\_POSITION - 1] = '.';

return isOverflow;

}

void displayLongReal(char longReal[])

{

int i;

for(i=0;i<REAL\_WIDTH;i++){printf("%c",longReal[i]);

}

printf("\n");

return;

}

三.实验3

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define MAX\_REAL\_WIDTH 25

#define POINT\_POSITION 12

typedef struct {

char digits[MAX\_REAL\_WIDTH];

int length;

} Real;

// Generate a random long real number

Real generateLongReal(void);

// Sum two long real numbers

Real addTwoLongReals(Real r1, Real r2);

// Display the real number in a single line

void displayLongReal(Real r);

// Display the number in a fixed width

void displayFixedwidthNumber(char number[], const int width);

int main(void) {

srand(time(NULL));

Real real1 = generateLongReal();

Real real2 = generateLongReal();

Real realSum = addTwoLongReals(real1, real2);

if (realSum.length > 0) {

printf("\n");

displayLongReal(real1);

printf("+");

displayLongReal(real2);

printf("=");

displayLongReal(realSum);

} else {

printf("\nError! Addition overflow!\n");

}

printf("\n");

return 0;

}

Real generateLongReal(void) {

Real r;

int fractionLength = rand() % (MAX\_REAL\_WIDTH - POINT\_POSITION + 1);

r.length = POINT\_POSITION + fractionLength;

int i;

r.digits[0] = '1' + rand() % 9;

for (i = 1; i < r.length - 1; i++) {

r.digits[i] = '0' + rand() % 10;

}

r.digits[r.length - 1] = '1' + rand() % 9;

r.digits[POINT\_POSITION - 1] = '.';

for (i = r.length; i < MAX\_REAL\_WIDTH; i++) {

r.digits[i] = '\0'; // Clear the rest of the memory

}

printf("Generated(%2d.%02d):", r.length, r.length - POINT\_POSITION);

displayFixedwidthNumber(r.digits, MAX\_REAL\_WIDTH);

return r;

}

void displayLongReal(Real r) {

int i;

for (i = 0; i < r.length; i++) {

printf("%c", r.digits[i]);

}

printf("\n");

return;

}

void displayFixedwidthNumber(char number[], const int width) {

int i;

for (i = 0; i < width; i++) {

if (number[i] == '\0') {

printf("\_");

} else {

printf("%c", number[i]);

}

}

printf("\n");

return;

}

Real addTwoLongReals(Real r1, Real r2) {

Real result;

int maxLength = (r1.length > r2.length) ? r1.length : r2.length;

result.length = maxLength; // Initialize result length to the maximum length of the two numbers

int carry = 0;

int i;

// Initialize the result array with zeros

for (i = 0; i < MAX\_REAL\_WIDTH; i++) {

result.digits[i] = '0';

}

// Start adding from the end

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

int digit1 = (i < r1.length) ? r1.digits[i] - '0' : 0;

int digit2 = (i < r2.length) ? r2.digits[i] - '0' : 0;

int sum = digit1 + digit2 + carry;

result.digits[i] = (sum % 10) + '0';

carry = sum / 10;

}

// Handle the carry

if (carry > 0) {

// If there is a carry and no space to store it, report overflow

if (result.length >= MAX\_REAL\_WIDTH) {

result.length = 0; // Set length to 0 to indicate overflow

} else {

// Shift all digits to the right to make space for the carry

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

result.digits[i] = result.digits[i - 1];

}

result.digits[0] = carry + '0';

result.length++; // Increase the length of the result

}

}

// Determine the position of the decimal point

if (result.length > POINT\_POSITION) {

result.digits[POINT\_POSITION - 1] = '.';

}

return result;

}

四.实验4

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#define MAX\_REAL\_WIDTH 25

typedef struct {

char digits[MAX\_REAL\_WIDTH];

int pointPos;

int length;

} Real;

// Generate a random long real number

Real generateLongReal(void);

// Sum two long real numbers

Real addTwoLongReals(Real r1, Real r2);

// Display the real number in a single line

void displayAssignedLongReal(int pointPos, Real r);

// Shift the digits to right direction

void shiftDigitsToRight(char number[], int shiftLength, int length);

// Shift the digits to left direction

void shiftDigitsToLeft(char number[], int shiftLength, int length);

// Display the number in a fixed width

void displayFixedwidthNumber(char number[], const int width);

// Display the vertical addition of two long reals

void displayVerticalAddition(Real r1, Real r2, Real result);

// Calculate the integer part length

int integerPartLength(Real r);

// Remove leading zeros from a number

void removeLeadingZeros(Real \*r);

int main(void)

{

srand(time(NULL));

Real real1 = generateLongReal();

Real real2 = generateLongReal();

Real realSum = addTwoLongReals(real1, real2);

if (realSum.length > 0) {

printf("\n ");

displayAssignedLongReal(realSum.pointPos, real1);

printf("+");

displayAssignedLongReal(realSum.pointPos, real2);

printf("=");

displayAssignedLongReal(realSum.pointPos, realSum);

displayVerticalAddition(real1, real2, realSum);

} else {

printf("\nError! addition overflow or invalid operation occurred!\n");

}

printf("\n");

return 0;

}

void displayAssignedLongReal(int pointPos, Real r)

{

int i;

for (i = 0; i < pointPos - r.pointPos; i++) {

printf(" ");

}

for (i = 0; i < r.length; i++) {

if (i == r.pointPos) {

printf("."); // Print the decimal point

}

printf("%c", r.digits[i]);

}

printf("\n");

return;

}

// Generate a random long real number

Real generateLongReal(void)

{

Real r;

int i;

int len;

do {

len = rand() % MAX\_REAL\_WIDTH + 1; // Randomly generate the length

} while (len == 0); // Ensure the length is not zero

r.length = len;

r.pointPos = rand() % (len + 1); // Randomly generate the position of the decimal point

for (i = 1; i < len; i++) {

r.digits[i] = '0' + rand() % 10; // Randomly generate each digit

}

r.digits[0] = '1' + rand() % 9;

r.digits[len] = '\0'; // Null terminator for the string

return r;

}

// Sum two long real numbers

Real addTwoLongReals(Real r1, Real r2)

{

Real result;

int maxLen = (r1.length > r2.length)? r1.length : r2.length;

int carry = 0;

int sum, i;

int decimalShift = r1.pointPos - r2.pointPos;

// Align the numbers based on the decimal point

if (decimalShift > 0) {

shiftDigitsToRight(r2.digits, decimalShift, r2.length);

} else if (decimalShift < 0) {

shiftDigitsToRight(r1.digits, -decimalShift, r1.length);

}

result.length = maxLen;

result.pointPos = (r1.pointPos > r2.pointPos)? r1.pointPos : r2.pointPos;

// Check for integer part carry

int intPartLen = integerPartLength(r1) > integerPartLength(r2)? integerPartLength(r1) : integerPartLength(r2);

int carryDetected = 0;

for (i = 0; i < intPartLen; i++) {

int digit1 = (i < r1.length)? (r1.digits[r1.length - intPartLen + i] - '0') : 0;

int digit2 = (i < r2.length)? (r2.digits[r2.length - intPartLen + i] - '0') : 0;

sum = digit1 + digit2 + carry;

if (sum >= 10 && i == intPartLen - 1) {

carryDetected = 1;

}

carry = sum / 10;

}

if (carryDetected) {

printf("Error: Addition carry occurred in integer part!\n");

result.length = 0;

return result;

}

// Add digits from right to left

for (i = 0; i < maxLen; i++) {

int digit1 = (i < r1.length)? (r1.digits[r1.length - 1 - i] - '0') : 0;

int digit2 = (i < r2.length)? (r2.digits[r2.length - 1 - i] - '0') : 0;

sum = digit1 + digit2 + carry;

carry = sum / 10;

result.digits[result.length - 1 - i] = (sum % 10) + '0';

}

// Handle carry if there is one

if (carry > 0) {

if (result.length >= MAX\_REAL\_WIDTH) {

printf("Error: Result length exceeds maximum limit!\n");

result.length = 0; // Indicate overflow

return result;

}

// Shift digits to the right to make room for carry

for (i = result.length; i >= 1; i--) {

result.digits[i] = result.digits[i - 1];

}

result.digits[0] = carry + '0';

result.length++;

} else {

// Remove leading zeros

removeLeadingZeros(&result);

}

result.digits[result.length] = '\0'; // Null terminator for the string

return result;

}

// Shift the digits to the right

void shiftDigitsToRight(char number[], int shiftLength, int length)

{

memmove(number + shiftLength, number, length);

memset(number, '0', shiftLength);

}

// Shift the digits to the left

void shiftDigitsToLeft(char number[], int shiftLength, int length)

{

memmove(number, number + shiftLength, length - shiftLength);

memset(number + (length - shiftLength), '0', shiftLength);

}

// Display the number in a fixed width

void displayFixedwidthNumber(char number[], const int width)

{

int len = strlen(number);

int i;

for (i = 0; i < width - len; i++) {

printf(" ");

}

for (i = 0; i < len; i++) {

printf("%c", number[i]);

}

printf("\n");

}

// Display the vertical addition of two long reals

void displayVerticalAddition(Real r1, Real r2, Real result)

{

int maxLen = (r1.length > r2.length)? r1.length : r2.length;

int maxDecimal = (r1.pointPos > r2.pointPos)? r1.pointPos : r2.pointPos;

int padding1 = maxLen - r1.length;

int padding2 = maxLen - r2.length;

int i;

// Print the first number

for (i = 0; i < padding1; i++) {

printf(" ");

}

for (i = 0; i < r1.length; i++) {

if (i == r1.pointPos) {

printf(".");

}

printf("%c", r1.digits[i]);

}

printf("\n");

// Print the plus sign

for (i = 0; i < maxLen + 1; i++) {

if (i == maxDecimal) {

printf("+ ");

} else {

printf(" ");

}

}

printf("\n");

// Print the second number

for (i = 0; i < padding2; i++) {

printf(" ");

}

for (i = 0; i < r2.length; i++) {

if (i == r2.pointPos) {

printf(".");

}

printf("%c", r2.digits[i]);

}

printf("\n");

// Print the horizontal line

for (i = 0; i < maxLen + 2; i++) {

if (i == maxDecimal) {

printf("---");

} else {

printf("--");

}

}

printf("\n");

// Print the result

for (i = 0; i < result.length; i++) {

if (i == result.pointPos) {

printf(".");

}

printf("%c", result.digits[i]);

}

printf("\n");

}

// Calculate the integer part length

int integerPartLength(Real r)

{

return r.pointPos;

}

// Remove leading zeros from a number

void removeLeadingZeros(Real \*r)

{

int leadingZeroCount = 0;

while (leadingZeroCount < r->length && r->digits[leadingZeroCount] == '0') {

leadingZeroCount++;

}

// Adjust pointPos without making it negative

if (r->pointPos >= leadingZeroCount) {

r->pointPos -= leadingZeroCount;

} else {

r->pointPos = 0;

}

// Move non-zero digits to the front

int i;

for (i = 0; i < r->length - leadingZeroCount; i++) {

r->digits[i] = r->digits[i + leadingZeroCount];

}

r->length -= leadingZeroCount;

r->digits[r->length] = '\0';

}

六.实验6

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#define MAX\_REAL\_WIDTH 25

typedef struct {

char digits[MAX\_REAL\_WIDTH];

int pointPos;

int length;

} Real;

// Generate a random long real number

void generateLongReal(Real \*rPtr);

// sum two long real numbers

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

// Display the real number in a single line

void displayAssignedLongReal(int pointPos, Real \*rPtr);

// shift the digits to right direction

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

// shift the digits to left direction

void shiftDigitsToLeft(const char \*longNumber, int shiftLength);

// Display the number in a fixed width

void displayFixedwidthNumber(const char \*number, const int width);

int main(void) {

srand(time(NULL));

Real r1, r2, sum;

Real \*rPtr1 = &r1, \*rPtr2 = &r2, \*rPtrSum = &sum;

generateLongReal(rPtr1);

generateLongReal(rPtr2);

addTwoLongReals(rPtr1, rPtr2, rPtrSum);

if (rPtrSum->length > 0) {

printf("\n");

displayAssignedLongReal(rPtrSum->pointPos, rPtr1);

printf("+");

displayAssignedLongReal(rPtrSum->pointPos, rPtr2);

printf("=");

displayAssignedLongReal(rPtrSum->pointPos, rPtrSum);

} else {

printf("\nError! addition overflow!\n");

}

printf("\n");

}

void generateLongReal(Real \*rPtr) {

rPtr->length = 2 + rand() % (MAX\_REAL\_WIDTH - 1);

rPtr->pointPos = 1 + rand() % (rPtr->length);

char \*cPtr = rPtr->digits;

\*cPtr = '1' + rand() % 9;

for (cPtr++; cPtr < rPtr->digits + rPtr->length - 1; cPtr++) {

\*cPtr = '0' + rand() % 10;

}

// Set the decimal point

rPtr->digits[rPtr->pointPos] = '.';

// Set the last digit

rPtr->digits[rPtr->length - 1] = '1' + rand() % 9;

// Null-terminate the string

rPtr->digits[rPtr->length] = '\0';

printf("Generated(%2d.%02d):", rPtr->length, rPtr->length - rPtr->pointPos);

displayFixedwidthNumber(rPtr->digits, MAX\_REAL\_WIDTH);

return;

}

// sum two long real numbers

void addTwoLongReals(const Real \*rPtr1, const Real \*rPtr2, Real \*rPtrSum) {

int decimalShift = rPtr1->pointPos - rPtr2->pointPos;

int maxLen;

int carry = 0;

int sum, i;

// Align the numbers based on the decimal point

if (decimalShift > 0) {

shiftDigitsToRight(rPtr2->digits, decimalShift);

rPtr2->pointPos += decimalShift;

rPtr2->length += decimalShift;

} else if (decimalShift < 0) {

shiftDigitsToRight(rPtr1->digits, -decimalShift);

rPtr1->pointPos -= decimalShift;

rPtr1->length -= decimalShift;

}

maxLen = (rPtr1->length > rPtr2->length)? rPtr1->length : rPtr2->length;

rPtrSum->length = maxLen;

rPtrSum->pointPos = rPtr1->pointPos;

// Add digits from right to left

for (i = 0; i < maxLen; i++) {

int digit1 = (i < rPtr1->length)? (rPtr1->digits[rPtr1->length - 1 - i] - '0') : 0;

int digit2 = (i < rPtr2->length)? (rPtr2->digits[rPtr2->length - 1 - i] - '0') : 0;

sum = digit1 + digit2 + carry;

carry = sum / 10;

rPtrSum->digits[rPtrSum->length - 1 - i] = (sum % 10) + '0';

}

// Handle carry if there is one

if (carry > 0) {

if (rPtrSum->length >= MAX\_REAL\_WIDTH) {

rPtrSum->length = 0;

return;

}

// Shift digits to the right to make room for carry

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

rPtrSum->digits[i] = rPtrSum->digits[i - 1];

}

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

rPtrSum->length++;

rPtrSum->pointPos++;

} else {

// Remove leading zeros

int leadingZeroCount = 0;

while (leadingZeroCount < rPtrSum->length && rPtrSum->digits[leadingZeroCount] == '0') {

leadingZeroCount++;

}

// Adjust pointPos without making it negative

if (rPtrSum->pointPos >= leadingZeroCount) {

rPtrSum->pointPos -= leadingZeroCount;

} else {

rPtrSum->pointPos = 0;

}

// Move non-zero digits to the front

for (i = 0; i < rPtrSum->length - leadingZeroCount; i++) {

rPtrSum->digits[i] = rPtrSum->digits[i + leadingZeroCount];

}

rPtrSum->length -= leadingZeroCount;

}

rPtrSum->digits[rPtrSum->length] = '\0'; // Null terminator for the string

}

// Display the real number in a single line

void displayAssignedLongReal(int pointPos, Real \*rPtr) {

int i;

for (i = 0; i < pointPos - rPtr->pointPos; i++) {

printf(" ");

}

for (i = 0; i < rPtr->length; i++) {

if (i == rPtr->pointPos) {

printf(".");

}

printf("%c", rPtr->digits[i]);

}

printf("\n");

}

// shift the digits to right direction

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

memmove((char \*)longNumber + shiftLength, (const char \*)longNumber, strlen(longNumber));

memset((char \*)longNumber, '0', shiftLength);

}

// shift the digits to left direction

void shiftDigitsToLeft(const char \*longNumber, int shiftLength) {

memmove((char \*)longNumber, (const char \*)longNumber + shiftLength, strlen(longNumber) - shiftLength);

memset((char \*)longNumber + (strlen(longNumber) - shiftLength), '0', shiftLength);

}

// Display the number in a fixed width

void displayFixedwidthNumber(const char \*number, const int width) {

int len = strlen(number);

int i;

for (i = 0; i < width - len; i++) {

printf(" ");

}

for (i = 0; i < len; i++) {

printf("%c", number[i]);

}

printf("\n");

}

七.实验7

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#define MAX\_REAL\_WIDTH 25

typedef struct {

char digits[MAX\_REAL\_WIDTH];

int pointPos;

int length;

} Real;

// create a random long real number, allocate memory for Real

Real \*generateLongReal(void);

// Release the memory for Real

void destroyLongReal(Real \*rPtr);

// sum two long real numbers

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

// Display the real number in a single line

void displayAssignedLongReal(int pointPos, Real \*rPtr);

// shift the digits to right direction

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

// shift the digits to left direction

void shiftDigitsToLeft(const char \*longNumber, int shiftLength);

// Display the number in a fixed width

void displayFixedwidthNumber(const char \*number, const int width);

int main(void) {

srand(time(NULL));

Real \*rPtr1 = generateLongReal();

Real \*rPtr2 = generateLongReal();

Real \*rPtrSum = addTwoLongReals(rPtr1, rPtr2);

if (rPtrSum->length > 0) {

printf("\n");

displayAssignedLongReal(rPtrSum->pointPos, rPtr1);

printf("+");

displayAssignedLongReal(rPtrSum->pointPos, rPtr2);

printf("=");

displayAssignedLongReal(rPtrSum->pointPos, rPtrSum);

} else {

printf("\nError! addition overflow!\n");

}

printf("\n");

destroyLongReal(rPtr1);

destroyLongReal(rPtr2);

destroyLongReal(rPtrSum);

return 0;

}

Real \*generateLongReal(void) {

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

rPtr->length = 2 + rand() % (MAX\_REAL\_WIDTH - 1);

rPtr->pointPos = 1 + rand() % (rPtr->length);

char \*cPtr = rPtr->digits;

\*cPtr = '1' + rand() % 9;

for (cPtr++; cPtr < rPtr->digits + rPtr->length - 1; cPtr++) {

\*cPtr = '0' + rand() % 10;

}

// Set the decimal point

rPtr->digits[rPtr->pointPos] = '.';

// Set the last digit

rPtr->digits[rPtr->length - 1] = '1' + rand() % 9;

// Null-terminate the string

rPtr->digits[rPtr->length] = '\0';

printf("Randomize(%2d.%02d):", rPtr->length, rPtr->length - rPtr->pointPos);

displayFixedwidthNumber(rPtr->digits, MAX\_REAL\_WIDTH);

return rPtr;

}

void destroyLongReal(Real \*rPtr) {

free(rPtr);

}

// sum two long real numbers

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

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

int decimalShift = rPtr1->pointPos - rPtr2->pointPos;

int maxLen;

int carry = 0;

int sum, i;

// Align the numbers based on the decimal point

if (decimalShift > 0) {

shiftDigitsToRight(rPtr2->digits, decimalShift);

rPtr2->pointPos += decimalShift;

rPtr2->length += decimalShift;

} else if (decimalShift < 0) {

shiftDigitsToRight(rPtr1->digits, -decimalShift);

rPtr1->pointPos -= decimalShift;

rPtr1->length -= decimalShift;

}

maxLen = (rPtr1->length > rPtr2->length)? rPtr1->length : rPtr2->length;

rPtrSum->length = maxLen;

rPtrSum->pointPos = rPtr1->pointPos;

// Add digits from right to left

for (i = 0; i < maxLen; i++) {

int digit1 = (i < rPtr1->length)? (rPtr1->digits[rPtr1->length - 1 - i] - '0') : 0;

int digit2 = (i < rPtr2->length)? (rPtr2->digits[rPtr2->length - 1 - i] - '0') : 0;

sum = digit1 + digit2 + carry;

carry = sum / 10;

rPtrSum->digits[rPtrSum->length - 1 - i] = (sum % 10) + '0';

}

// Handle carry if there is one

if (carry > 0) {

if (rPtrSum->length >= MAX\_REAL\_WIDTH) {

destroyLongReal(rPtrSum);

return NULL;

}

// Shift digits to the right to make room for carry

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

rPtrSum->digits[i] = rPtrSum->digits[i - 1];

}

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

rPtrSum->length++;

rPtrSum->pointPos++;

} else {

// Remove leading zeros

int leadingZeroCount = 0;

while (leadingZeroCount < rPtrSum->length && rPtrSum->digits[leadingZeroCount] == '0') {

leadingZeroCount++;

}

// Adjust pointPos without making it negative

if (rPtrSum->pointPos >= leadingZeroCount) {

rPtrSum->pointPos -= leadingZeroCount;

} else {

rPtrSum->pointPos = 0;

}

// Move non-zero digits to the front

for (i = 0; i < rPtrSum->length - leadingZeroCount; i++) {

rPtrSum->digits[i] = rPtrSum->digits[i + leadingZeroCount];

}

rPtrSum->length -= leadingZeroCount;

}

rPtrSum->digits[rPtrSum->length] = '\0'; // Null terminator for the string

return rPtrSum;

}

// Display the real number in a single line

void displayAssignedLongReal(int pointPos, Real \*rPtr) {

int i;

for (i = 0; i < pointPos - rPtr->pointPos; i++) {

printf(" ");

}

for (i = 0; i < rPtr->length; i++) {

if (i == rPtr->pointPos) {

printf(".");

}

printf("%c", rPtr->digits[i]);

}

printf("\n");

}

// shift the digits to right direction

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

memmove((char \*)longNumber + shiftLength, (const char \*)longNumber, strlen(longNumber));

memset((char \*)longNumber, '0', shiftLength);

}

// shift the digits to left direction

void shiftDigitsToLeft(const char \*longNumber, int shiftLength) {

memmove((char \*)longNumber, (const char \*)longNumber + shiftLength, strlen(longNumber) - shiftLength);

memset((char \*)longNumber + (strlen(longNumber) - shiftLength), '0', shiftLength);

}

// Display the number in a fixed width

void displayFixedwidthNumber(const char \*number, const int width) {

int len = strlen(number);

int i;

for (i = 0; i < width - len; i++) {

printf(" ");

}

for (i = 0; i < len; i++) {

printf("%c", number[i]);

}

printf("\n");

}

八.实验8

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#define MAX\_REAL\_WIDTH 25

typedef struct {

char\* digits;

int pointPos;

int length;

} Real;

// Create a random long real number, allocate memory for Real

Real \*generateLongReal(void);

// Release the memory for Real

void destroyLongReal(Real \*rPtr);

// Sum two long real numbers

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

// Shift the digits to right direction, re-allocate memory for rPtr->digits[]

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

// Shift the digits to left direction, re-allocate memory for rPtr->digits[]

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

// Display the real number in a single line

void displayAssignedLongReal(int assignedLength, const Real\* rPtr);

int main(void) {

srand(time(NULL));

Real \*rPtr1 = generateLongReal();

Real \*rPtr2 = generateLongReal();

Real \*rPtrSum = addTwoLongReals(rPtr1, rPtr2);

printf("\n");

displayAssignedLongReal(rPtr1->pointPos, rPtr1);

printf("+");

displayAssignedLongReal(rPtr2->pointPos, rPtr2);

printf("=");

displayAssignedLongReal(rPtrSum->pointPos, rPtrSum);

printf("\n");

destroyLongReal(rPtr1);

destroyLongReal(rPtr2);

destroyLongReal(rPtrSum);

return 0;

}

Real \*generateLongReal(void) {

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

rPtr->length = 2 + rand() % (MAX\_REAL\_WIDTH - 1);

rPtr->pointPos = 1 + rand() % (rPtr->length);

// Allocate memory for rPtr->digits array

rPtr->digits = (char\*)calloc(rPtr->length + 1, sizeof(char));

rPtr->digits[0] = '1' + rand() % 9;

int i;

for (i = 1; i < rPtr->length - 1; i++) {

rPtr->digits[i] = '0' + rand() % 10;

}

rPtr->digits[rPtr->length - 1] = '1' + rand() % 9;

rPtr->digits[rPtr->pointPos] = '.';

printf("Randomize(%2d.%02d):", rPtr->length, rPtr->length - rPtr->pointPos);

displayAssignedLongReal(0, rPtr);

return rPtr;

}

void destroyLongReal(Real \*rPtr) {

if (rPtr->digits!= NULL) {

free(rPtr->digits);

}

free(rPtr);

return;

}

// Sum two long real numbers

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

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

int decimalShift = rPtr1->pointPos - rPtr2->pointPos;

int maxLen;

int carry = 0;

int sum, i;

// Align the numbers based on the decimal point

if (decimalShift > 0) {

shiftDigitsToRight(rPtr2, decimalShift);

rPtr2->pointPos += decimalShift;

rPtr2->length += decimalShift;

} else if (decimalShift < 0) {

shiftDigitsToLeft(rPtr1, -decimalShift);

rPtr1->pointPos -= decimalShift;

rPtr1->length -= decimalShift;

}

maxLen = (rPtr1->length > rPtr2->length)? rPtr1->length : rPtr2->length;

rPtrSum->length = maxLen;

rPtrSum->pointPos = rPtr1->pointPos;

// Add digits from right to left

for (i = 0; i < maxLen; i++) {

int digit1 = (i < rPtr1->length)? (rPtr1->digits[rPtr1->length - 1 - i] - '0') : 0;

int digit2 = (i < rPtr2->length)? (rPtr2->length - 1 - i >= 0? rPtr2->digits[rPtr2->length - 1 - i] - '0' : 0) : 0;

sum = digit1 + digit2 + carry;

carry = sum / 10;

rPtrSum->digits[maxLen - 1 - i] = (sum % 10) + '0';

}

// Handle carry if there is one

if (carry > 0) {

if (rPtrSum->length >= MAX\_REAL\_WIDTH) {

destroyLongReal(rPtrSum);

return NULL;

}

// Shift digits to the right to make room for carry

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

rPtrSum->digits[i] = rPtrSum->digits[i - 1];

}

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

rPtrSum->length++;

rPtrSum->pointPos++;

} else {

// Remove leading zeros

int leadingZeroCount = 0;

while (leadingZeroCount < rPtrSum->length && rPtrSum->digits[leadingZeroCount] == '0') {

leadingZeroCount++;

}

// Adjust pointPos without making it negative

if (rPtrSum->pointPos >= leadingZeroCount) {

rPtrSum->pointPos -= leadingZeroCount;

} else {

rPtrSum->pointPos = 0;

}

// Move non-zero digits to the front

for (i = 0; i < rPtrSum->length - leadingZeroCount; i++) {

rPtrSum->digits[i] = rPtrSum->digits[i + leadingZeroCount];

}

rPtrSum->length -= leadingZeroCount;

}

rPtrSum->digits[rPtrSum->length] = '\0'; // Null terminator for the string

return rPtrSum;

}

// Shift the digits to right direction, re-allocate memory for rPtr->digits[]

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

int newLength = rPtr->length + shiftLength;

char \*newDigits = (char \*)realloc(rPtr->digits, newLength + 1);

if (newDigits == NULL) {

fprintf(stderr, "Memory allocation failed\n");

exit(1);

}

rPtr->digits = newDigits;

memmove(rPtr->digits + shiftLength, rPtr->digits, rPtr->length);

memset(rPtr->digits, '0', shiftLength);

rPtr->length = newLength;

}

// Shift the digits to left direction, re-allocate memory for rPtr->digits[]

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

int newLength = rPtr->length - shiftLength;

char \*newDigits = (char \*)realloc(rPtr->digits, newLength + 1);

if (newDigits == NULL) {

fprintf(stderr, "Memory allocation failed\n");

exit(1);

}

rPtr->digits = newDigits;

memmove(rPtr->digits, rPtr->digits + shiftLength, newLength);

memset(rPtr->digits + newLength, '0', shiftLength);

rPtr->length = newLength;

}

// Display the real number in a single line

void displayAssignedLongReal(int assignedLength, const Real\* rPtr) {

int i;

for (i = 0; i < assignedLength; i++) {

printf(" ");

}

for (i = 0; i < rPtr->length; i++) {

if (i == rPtr->pointPos) {

printf(".");

}

printf("%c", rPtr->digits[i]);

}

printf("\n");

}

日历代码

#include <stdio.h>

#include <stdlib.h>

#define YEAR 2021

#define IS\_LEAPYEAR 0

#define JAN1 5

int main(void)

{

int month, day, daySeq;

int first = JAN1;

scanf("%d%d", &month, &day) ;

//get the sequence of that day in that week

int sum = 0;

int i;

int a[12] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31,};

for(i = 0; i < (month - 1); i++){

sum += a[i];

}

day += 4;

first += sum;

first %= 7;

first -= 1;

sum += day;

daySeq = sum % 7;

printf("daySeq = %d\n",daySeq);

printf("Mon Tue Wed The Fri Sat Sun\n");

printf("---------------------------\n");

int j;

for ( j = 0; j<first; j++){

printf(" ");

}

for (j = 1; j <= a[month - 1]; j++) {

printf("%3d ", j);

if ((j + first) % 7 == 0)

printf("\n");

}

return 0;

}

