**《程序设计基础课程设计》第1周实验报告**

班级：1803012

姓名：杨煜

学号：18030100204

所做题目：一、数值处理

**第1题**

**1.原始题目及要求；**

**（1）高精度计算**

要求：用整型数组表示10进制大整数（超过2^32的整数），数组的每个元素存储大整数的一位数字，实现大整数的加减法。

**2.题目的分析**

（1）涉及知识点

数组、流程控制、函数等

（2）题目功能理解

由于数组的每个元素仅存储一位数字，所以所需要的数组类型只需 使用为char型；为了表示便利，可以在存储过程中强制每一个数都以有符号数的形式存储；整数的加法可以分为符号位相同的加法和符号位相反的加法，前者只需要按位相加，而后者可以用绝对值较大者的绝对值减去较小者的绝对值，并最终取前者符号位；整数的减法可以用整数大加法来实现，这位编程实现提供了一种简单的方式。

**3. 题目的总体设计：设计思路、算法描述**

（1）程序模块：

（A）初始化模块

初始化一个大整数, 包括分配地址空间与初始赋值。

（B）销毁模块

销毁一个大整数，释放空间。

（C）赋值模块

为一个大整数赋值。

（D）大于算符模块

无符号大整数大于运算。

（E）加法模块

实现有符号大整数加法

（F）减法模块

通过加法模块实现减法。

（2）模块调用关系

减法模块调用加法模块；加法模块调用大于运算符模块和赋值模块。

（3）输入输出数据说明

输入数据：计算功能选择，计算功能功能所需的两个操作数。

输出数据：计算结果。

（4）总体流程

1）显示功能选择界面，提示用户选择功能，转2。

2）用户输入功能序号，用户选择功能。输入若为‘1’，则转3,；若为‘2’则转4

若为‘3’，则转5；若为其他，则提示输入错误，转2。

3）用户输入两操作数，相加后显示运算结果，转1。

4）用户输入两操作数，想减后显示运算结果，转1。

5）退出程序。

**4. 各功能模块/函数的设计说明：**

（1）big\_int big\_int\_initialize( const char\* val )

函数名称: big\_int\_initialize

函数功能: 初始化一个大整数, 包括分配地址空间与初始赋值

参数: big\_int obj( 保存分配地址的指针 ), const char\* val( 表示初始值的字符串 )

返回值: big\_int( 表示结果 )

执行流程：

1）计算初始值字符串长，转2。

2）若初始值有符号分配长度为串长加1的地址空间，否则分配加2 的地址空间

若成功转4，失败转3。

3）返回NULL

4）若初始值无符号位则添加符号位，拷贝初始值，转5。

5）返回分配到的地址空间的首地址。

可能结果：

1）分配失败，返回NULL

2）分配成功，返回大整数首地址

（2）void big\_int\_destory( big\_int obj )

函数名称: big\_int\_destory

函数功能: 销毁一个大整数

参数: big\_int obj( 要销毁的大整数的首地址 )

返回值: void

执行流程：

1）释放地址空间，转2。

2）结束运行。

可能结果：

1）完成销毁

（3）big\_int big\_int\_assign( big\_int dst, const big\_int src )

函数名称: big\_int\_assign

函数功能: 为一个大整数赋值

参数: const big\_int src( 赋值来源 )

返回值: big\_int( 表示结果 )

执行流程：

1）计算源整数的长度并加1（多出的一位用于存储终结符），转2。

2）申请之前计算出的长度的地址空间，若申请成功转3，否则返回NULL。

3）将源整数的内容拷贝到目的数组，转4.

4）返回新的地址空间的首地址表示的大整数。

可能结果：

1）赋值失败，返回NULL

2）赋值成功，返回大整数首地址

（4）bool big\_int\_unsigned\_greater( const big\_int a, const big\_int b )

函数名称: big\_int\_unsigned\_greater

函数功能: 无符号大整数大于运算

参数: const big\_int a( 大于运算左侧操作数 ), const big\_int b( 大于运算右侧

操作数 )

返回值: bool( 表示是否大于 )

执行流程：

1）计算a的长度，保存为sz\_a，计算b的长度，保存为sz\_b，转2。

2）如果sz\_a等于sz\_b则返回sz\_a>sz\_b的运算结果，否则转3。

3）由数字的高位到低位比较数字的大小，一旦出现数字不一样的位置，返回

该位置数字比较的结果，否则转4.

4）执行到此处证明两数相等，放回false，结束运行。

可能结果：

1）true，表示a>b；

2）false，表示a<=b；

（5）big\_int big\_int\_add( const big\_int src1, const big\_int src2 )

函数名称: big\_int\_add

函数功能: 大整数加法

参数: const big\_int src1( 操作数1 ), const big\_int src2( 操作数2 )

返回值: big\_int( 表示结果 )

执行流程：

1）计算src1的长度，保存为sz1，计算src2的长度，保存为sz2，保存两者中

较大者的长度加2（多余的两位中一位是终结符，另一位表示两数相加的

结果的位数不会超过两者中位数大者的长度加1）为len，转2。

2）分配长度为len的地址空间，若失败则返回NULL，否则在最后一位添加终

结符后转3。

3）若src1的符号位等于src2的符号位，转4，否则转6。

4）从低位到高位依次完成两操作数相加，结果存入新的地址空间，转5。

5）去除多余的地址空间，返回结果，结束运行。

6）判断两数绝对值大小，转7。

7）从低位到高位用绝对值大的数的绝对值减去绝对值较小的数的绝对值，转

8。

8）结果符号位设置为绝对值较大者的符号位，去除多余的地址空间，转9

9）返回结果，结束运行。

可能结果：

1）计算失败，返回NULL。

2）计算成功，返回大整数首地址。

（5）big\_int big\_int\_sub( const big\_int src1, const big\_int src2 )

函数名称: big\_int\_sub

函数功能: 大整数减法

参数: const big\_int src1(被减数 ), const big\_int src2( 减数 )

返回值: big\_int( 表示结果 )

执行流程：

1）计算src2的长度加1，保存为为len，转2。

2）分配长度为len的地址空间，若失败则返回NULL，否则转3。

3）将src2的内容拷贝到新的地址空间，转4。

4）翻转新整数的符号位，转5。

5）调用big\_int\_add计算原被减数和新整数加法结果，转6。

6）返回结果，结束。

可能结果：

1）计算失败，返回NULL

2）计算成功，返回大整数首地址

**5. 源程序：**

（1）big\_int\_test.c

#include<stdio.h>

#include<stdbool.h>

#include"big\_int.h"

int main(){

int choice;

big\_int a, b, res;

char t\_a[512], t\_b[512];

bool is\_end = false;

while( !is\_end ){

printf("请选择功能\n");

printf("1.加法\n2.减法\n3.退出程序\n");

scanf("%d", &choice );

switch( choice ){

case 1:{

printf("请输入两个操作数\n");

scanf("%s %s", t\_a, t\_b );

a = big\_int\_initialize(t\_a);

b = big\_int\_initialize(t\_b);

res = big\_int\_assign( big\_int\_add(a,b) );

printf("结果为：%s\n", res );

free( a );

free( b );

free( res );

break;

};

case 2:{

printf("请输入两个操作数\n");

scanf("%s %s", t\_a, t\_b );

a = big\_int\_initialize(t\_a);

b = big\_int\_initialize(t\_b);

res = big\_int\_assign( big\_int\_sub(a,b) );

printf("结果为：%s\n", res );

free( a );

free( b );

free( res );

break;

};

case 3:{

printf("退出程序\n");

is\_end = true;

break;

};

default:{

printf("不存在此选项,请重新输入\n");

break;

};

}

}

return 0;

}

（2）big\_int.h

#ifndef \_\_BIG\_INT\_H

#define \_\_BIG\_INT\_H

#include<string.h>

#include<malloc.h>

#include<stdbool.h>

typedef char\* big\_int;// 采用字符数组存储大整数, 大整数首地址即字符指针

/\*

\* 函数名称: big\_int\_initialize

\* 函数功能: 初始化一个大整数, 包括分配地址空间与初始赋值

\* 参数: big\_int obj( 保存分配地址的指针 ), const char\* val( 表示初始值的字符串 )

\* 返回值: big\_int( 表示结果 )

\*/

big\_int big\_int\_initialize( const char\* val ){

size\_t len = strlen( val ) + 1; // 字符数组需要多一个字符存储终结符

bool flag = val[0] !='-' && val[0] != '+';// 标识初始值是否有符号

if( flag ){

++len;

}// 如果输入的无符号, 则添加上'+', 从而数组长度需要加1

big\_int temp = (big\_int)malloc( sizeof(char)\*(len) );

if( !temp ){

return NULL;

}// 分配空间, 若成功则继续执行, 失败则返回错误

if( flag ){

strcpy( temp+1, val );

temp[0] = '+';

}

else{

strcpy( temp, val );

}// 初始输入确定拷贝起始位置

return temp;

}

/\*

\* 函数名称: big\_int\_destory

\* 函数功能: 销毁一个大整数

\* 参数: big\_int obj( 要销毁的大整数的首地址 )

\* 返回值: void

\*/

void big\_int\_destory( big\_int obj ){

free( obj );

return;

}

/\*

\* 函数名称: big\_int\_assign

\* 函数功能: 为一个大整数赋值

\* 参数: const big\_int src( 赋值来源 )

\* 返回值: big\_int( 表示结果 )

\*/

big\_int big\_int\_assign( const big\_int src ){

size\_t len = strlen( src )+1;// 多一位存储终结符

big\_int temp = (big\_int)malloc( sizeof(char)\*(len) );

if( !temp ){

return NULL;

}

strcpy( temp, src );

return temp;

}

/\*

\* 函数名称: big\_int\_unsigned\_greater

\* 函数功能: 无符号大整数大于运算

\* 参数: const big\_int a( 大于运算左侧操作数 ), const big\_int b( 大于运算右侧操作数 )

\* 返回值: bool( 表示是否大于 )

\*/

bool big\_int\_unsigned\_greater( const big\_int a, const big\_int b ){

size\_t sz\_a = strlen( a ), sz\_b = strlen( b );

if( sz\_a != sz\_b){

return sz\_a>sz\_b;

}// 长度不同时, 长度长的数字大

for( size\_t i=1; i<sz\_a; --i ){

if( a[i] != b[i] ){

return a[i] > b[i];

}

}// 长度相同时, 最高不等位的比较结果即最终结果

return false;// 每一位均相等, 即两数相等, 即不大于

}

/\*

\* 函数名称: big\_int\_add

\* 函数功能: 大整数加法

\* 参数: const big\_int src1( 操作数1 ), const big\_int src2( 操作数2 )

\* 返回值: big\_int( 表示结果 )

\*/

big\_int big\_int\_add( const big\_int src1, const big\_int src2 ){

size\_t sz1 = strlen( src1 ), sz2 = strlen( src2 ),

len = (sz1>sz2? sz1: sz2) + 2;

// 加法结果的位数不会超过两个操作数长度最大者加一

big\_int temp = (big\_int)malloc( sizeof(char)\*(len) );

if( !temp ){

return NULL;

}

temp[len-1] = '\0';// 添加终结符

if( src1[0] == src2[0] ){

// 两数符号相同时, 绝对值相加, 最后添上符号位即可

int carry\_in = 0;// 进位值

size\_t index1=sz1-1, index2=sz2-1, index = len-2;

while( index1>=1 && index2>=1 ){

// 从低位到高位按位相加

int sum = ( src1[index1]-'0' ) + ( src2[index2]-'0' ) + carry\_in;

carry\_in = sum/10;

temp[index] = '0' + sum%10;

--index1;

--index2;

--index;

}

// 将长度更长的数的剩余位和进位加到结果中

while( index1>=1 ){

int sum = ( src1[index1]-'0' ) + carry\_in;

carry\_in = sum/10;

temp[index] = '0' + sum%10;

--index1;

--index;

}

while( index2>=1 ){

int sum = ( src2[index2]-'0' ) + carry\_in;

carry\_in = sum/10;

temp[index] = '0' + sum%10;

--index2;

--index;

}

if( carry\_in ){

temp[index] = '1';

--index;

}

temp[index--] = src1[0];// 确定符号位

size\_t gap = index + 1;// 表示最终结果相对于首地址的偏移

if( gap ){

//若有空余空间, 填满空间的前端

size\_t f\_len = len - gap;

for( size\_t i=0; i<f\_len; ++i ){

temp[i] = temp[i+gap];

}

}

temp = ( big\_int )realloc( temp, len - gap );// 释放需要的地址空间

return temp;// 赋值并返回结果

}

else{

// 两数符号不同时, 用绝对值大的数的绝对值减绝对值小的数的绝对值,

// 结果符号位为绝对值大的数的符号位

int carry\_in = 0;// 借位值

bool flag = big\_int\_unsigned\_greater( src1+1, src2+1 );

// 确定绝对值大的是哪一个操作数

const big\_int t\_src1 = flag? src1: src2;// 绝对值较大的数

const big\_int t\_src2 = flag? src2: src1;// 绝对值较小的数

size\_t t\_sz1 = flag? sz1: sz2;

size\_t t\_sz2 = flag? sz2: sz1;

size\_t index1=t\_sz1-1, index2=t\_sz2-1, index = len-2;

while( index1>=1 && index2>=1 ){

int sum = ( t\_src1[index1]-'0' ) - ( t\_src2[index2]-'0' ) - carry\_in;

carry\_in = sum < 0;// 若结果小于0则表示不够减, 需要借位

temp[index] = carry\_in? sum+10+'0': sum+'0';

--index1;

--index2;

--index;

}

while( index1>=1 ){

int sum = ( t\_src1[index1]-'0' ) - carry\_in;

carry\_in = sum < 0;// 若结果小于0则表示不够减, 需要借位

temp[index] = carry\_in? sum+10+'0': sum+'0';

--index1;

--index;

}

while( index2>=1 ){

int sum = ( t\_src1[index2]-'0' ) - carry\_in;

carry\_in = sum < 0;// 若结果小于0则表示不够减, 需要借位

temp[index] = carry\_in? sum+10+'0': sum+'0';

--index2;

--index;

}// 此运算规则下运算结果一定不会向比最高位更高的位借位

temp[index--] = t\_src1[0];

size\_t gap = index + 1;

if( gap ){

size\_t f\_len = len - gap;

for( size\_t i=0; i<f\_len; ++i ){

temp[i] = temp[i+gap];

}

}

temp = ( big\_int )realloc( temp, len - gap );

return temp;

}

}

/\*

\* 函数名称: big\_int\_sub

\* 函数功能: 大整数减法

\* 参数: const big\_int src1( 被减数 ), const big\_int src2( 减数 )

\* 返回值: bool( 表示是否成功进行了运算 )

\*/

big\_int big\_int\_sub( const big\_int src1, const big\_int src2 ){

size\_t len = strlen( src2 )+1;// 多一位用于存储终结符

big\_int temp = (big\_int)malloc( sizeof(char)\*(len) );

if( !temp ){

return NULL;

}// 分配空间用于临时存储取反后的减数

strcpy( temp, src2 );

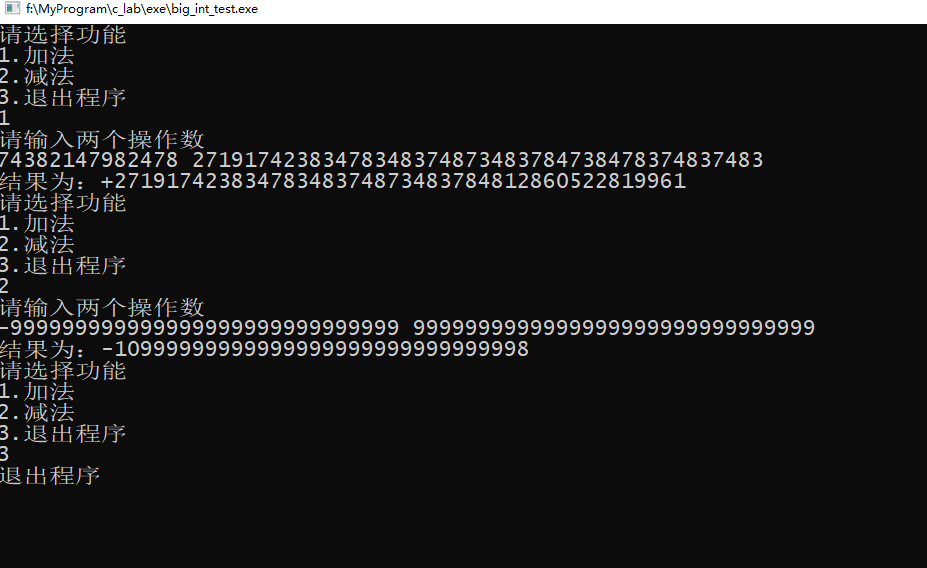
temp[0] = src2[0]=='+' ? '-': '+';// 取反

temp=big\_int\_add( src1, temp );// 减法相当于加上减数的相反数

return temp;

}

#endif

**6. 测试数据（输入、输出）：**

**7. 小结：**

本次实验圆满完成，实验结果符合实验要求。

实验过程中，我复习到了c语言中数组、流程控制、函数等重要知识，收货颇丰。

但此次实验还并不完美，例如如果计算结果中有多个前导0，这些0并不会被自动去除，从而占用了许多内存空间。在后续的优化中，应该对此问题做更完善的处理。