中南大学

数据结构试验报告

题 目 实验一

学生姓名 王云鹏

学 号 8213180228

指导老师 郑 瑾

学 院 计算机学院

专业班级 物联网1802

完成时间 2020.6

指导老师评定 签名

实验一

1. 需求分析

1．单向链表操作的实现（验证性实验）

问题描述

(1) 用头插法（或尾插法）建立带头结点的单向链表。

(2) 对已建立的单向链表实现插入、删除、查找等基本操作。

2．城市链表（设计性实验）

问题描述

将若干城市的信息存入一个带头结点的单向链表。结点中的城市信息包括城市名、城市的位置坐

标。要求能够利用城市名和位置坐标进行有关查找、插入、删除、更新等操作。

基本要求

(1) 给定一个城市名，返回其位置坐标。

(2) 给定一个位置坐标P和一个距离D，返回所有与P的距离小于等于D的城市。

测试数据

由读者依据软件工程的测试技术自己确定。注意测试边界数据

4．长整数运算（综合性实验）

问题描述

设计一个程序实现两个任意长的整数求和运算。

基本要求

利用双向循环链表实现长整数的存储，每个结点含一个整型变量。任何整型变量的范围是

−(215−1)～(215−1)。输入和输出形式：每4位一组，组间用逗号隔开。

测试数据

(1) 0和0，应输出“0”。

(2) −2345,6789，−7654,3211，应输出“−1,0000,0000”。

(3) −9999,9999，1,0000,0000,0000，应输出“9999,0000,0001”。362 附 录

(4) 1,0001,000，−1,0001,0001，应输出“0”。

(5) 1,0001,0001，−1,0001,0000，应输出“1”。

实现提示

(1) 每个结点中可以存放的最大整数为215−1＝32 767，这样才能保证两数相加不会溢出。但若按

32 768进制数存放，在十进制数与32768进制数之间的转换十分不方便，故可以在每个结点中仅存放十

进制数的4位，即不超过9 999的非负整数，整个链表视为万进制数。

(2) 可以利用头结点数据域的符号代表长整数的符号。用其绝对值表示元素结点数目。相加过程

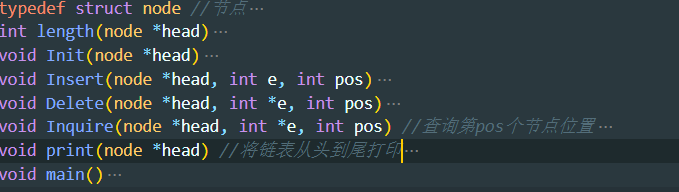
中不要破坏两个操作数链表。两操作数的头指针存于指针数组中是简化程序结构的一种方法。不能给

长整数位数规定上限。

1. 概要设计

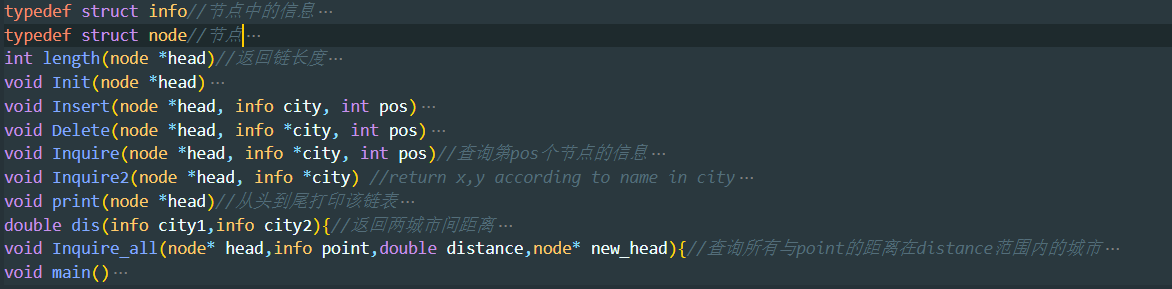
* 验证性实验

函数：



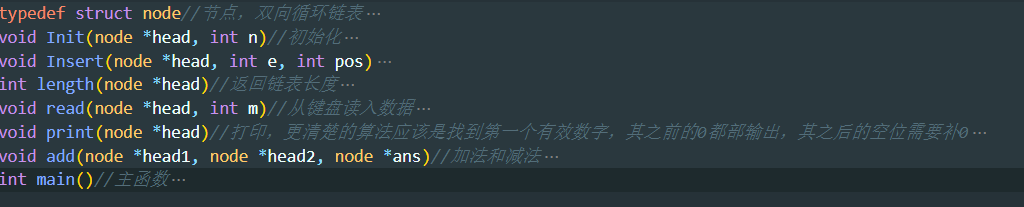
* 设计性实验

函数：



* 综合性实验

函数：



1. 详细设计

* 验证性实验

|  |
| --- |
| *#include* <stdio.h>  *#include* <stdlib.h>  typedef struct node *//节点*  {      int data;      struct node \*next;  } node;  int length(node \*head)  { *//返回链表长度，算上头节点*      int len = 0;      node \*p = head;  *while* (p != NULL)      {          len++;          p = p->next;      }  *return* len;  }  void Init(node \*head)  {                                        *//带头节点*  *if* (head == NULL)      {          printf("malloc fail in main\n");  *return*;      }      head->data = 0; *//头节点存链长，且头节点不计入链长*  *// printf("%d",head->data);*      head->next = NULL;  }  void Insert(node \*head, int e, int pos)  {                                      *//在第pos个节点后插入新节点*  *if* (pos > length(head) || pos < 0) *//不合法位置*      {          printf("illegal position in Insert\n");  *return*;      }      node \*p = head;  *for* (int i = pos; i > 1; i--) *//找到插入位置*      {          p = p->next;      }      node \*x = (node \*)malloc(sizeof(node)); *//要插入的节点*  *if* (x == NULL)      {          printf("malloc fail in Insert");  *return*;      }      x->data = e;      x->next = p->next;      p->next = x;      head->data++;  }  void Delete(node \*head, int \*e, int pos)  {                                      *//删除第pos个节点*  *if* (pos > length(head) || pos < 0) *//非法位置*      {          printf("illegal position in Delete\n");  *return*;      }      node \*p = head;*//当前指针*      node \*p\_pre;*//当前指针的前驱*  *for* (int i = pos; i >= 1; i--) *//找到位置*      {          p\_pre = p;          p = p->next;      }      p\_pre->next = p->next;*//将当前节点从链表中断开*      \*e = p->data;*//用e带出删掉的值*      free(p);      head->data--;  }  void Inquire(node \*head, int \*e, int pos) *//查询第pos个节点位置*  {  *if* (pos > length(head) || pos < 0)      {          printf("illegal position in Inquire\n");  *return*;      }      node \*p = head;  *for* (int i = pos; i >= 1; i--)*//找位置*      {          p = p->next;*//下一个*      }      \*e = p->data;*//用e带出查询结果*      printf("查询结果————第%d个节点的值为：%d",pos,\*e);  }  void print(node \*head) *//将链表从头到尾打印*  {      node \*p = head;      printf("当前链表有节点：%d个\n", head->data);      printf("链表节点从头到尾分别为：\n");  *while* (p->next != NULL)      {          p = p->next;          printf("%d\n", p->data);      }  }  void main()  {      node\* head = (node \*)malloc(sizeof(node)); *//在主函数中申请头节点*      Init(head); *//初始化*      int \*e;      Insert(head, 10, 1); *//插入*      Insert(head, 28, 2);      Insert(head, 30, 1);  *// printf("hh");*      print(head);      Delete(head, e, 2); *//删除*      print(head);      Inquire(head, e, 2); *//查询*  } |

* 设计性实验

|  |
| --- |
| *#include* <stdio.h>  *#include* <stdlib.h>  *#include* <string.h>  *#include*<math.h>  typedef struct info*//节点中的信息*  {      char \*name;      double x;      double y;  } info;  typedef struct node*//节点*  {      info city;      struct node \*next;  } node;  int length(node \*head)*//返回链长度*  {      int len = 0;      node \*p = head;  *while* (p != NULL)      {          len++;          p = p->next;      }  *return* len;  }  void Init(node \*head)  {                     *//带头节点*      head->next = NULL;  }  void Insert(node \*head, info city, int pos)  { *//在第pos个节点后插入新节点*  *if* (pos > length(head) || pos < 0)*//非法位置*      {          printf("illegal position in Insert\n");  *return*;      }      node \*p = head;  *for* (int i = pos; i > 1; i--)*//找到位置*      {          p = p->next;      }      node \*x = (node \*)malloc(sizeof(node));*//申请节点*  *if* (x == NULL)      {          printf("malloc fail in Insert");  *return*;      }      x->city = city;      x->next = p->next;      p->next = x;  }  void Delete(node \*head, info \*city, int pos)  { *//删除第pos个节点*  *if* (pos > length(head) || pos < 0)*//非法位置*      {          printf("illegal position in Delete\n");  *return*;      }      node \*p = head;      node \*p\_pre = head;  *for* (int i = pos; i > 0; i--)*//找到位置*      {          p\_pre = p;          p = p->next;      }      p\_pre->next = p->next;      \*city = p->city;      free(p);  }  void Inquire(node \*head, info \*city, int pos)*//查询第pos个节点的信息*  {  *if* (pos > length(head) || pos < 0)*//非法位置*      {          printf("illegal position in Inquire\n");  *return*;      }      node \*p = head;  *for* (int i = pos; i > 0; i--)*//找到位置*      {          p = p->next;      }      \*city = p->city;      printf("the result of Inquire is: name:%s  x:%f  y:%f\n",      city->name,city->x,city->y);  }  void Inquire2(node \*head, info \*city) *//return x,y according to name in city*  {      node \*p = head;  *while* (p != NULL)*//遍历链表*      {  *if* (strcmp(city->name, p->city.name) == 0)*//判断当前节点城市名与传入参数city中的城市名是否相同*          {              city->x = p->city.x;              city->y = p->city.y;              printf("the address of the %s is (%f,%f)\n", city->name,                     city->x, city->y);  *return*;          }          p = p->next;      }      printf("fail to find %s\n",city->name);*//还没返回说明没找到*  }  void print(node \*head)*//从头到尾打印该链表*  {      node \*p = head;      printf("当前链表有节点：%d个\n", length(head)-1);      printf("链表节点从头到尾分别为：\n");  *while* (p->next != NULL)*//遍历*      {          p = p->next;          printf("name:%s   ", p->city.name);          printf("x:%f    ", p->city.x);          printf("y:%f\n", p->city.y);      }  }  double dis(info city1,info city2){*//返回两城市间距离*  *return* sqrt(pow((city1.x-city2.x),2)+pow((city1.y-city2.y),2));  }  void Inquire\_all(node\* head,info point,double distance,node\* new\_head){*//查询所有与point的距离在distance范围内的城市*      node\* p=head;  *while*(p->next!=NULL){*//遍历*          p=p->next;          double disReal=dis(point,p->city);  *if*(disReal<distance){*//判断是否在distance范围内*              Insert(new\_head,p->city,1);          }      }  }  void main()  {      node \*head = (node \*)malloc(sizeof(node));*//在主函数内申请头节点*  *if* (head == NULL)      {          printf("malloc fail in main\n");  *return*;      }      Init(head);*//初始化*      info city1 = {"hefei", 1, 1};      Insert(head, city1, 1);*//插入*      info city2 = {"nanjing", 2, 2};      Insert(head, city2, 2);      info city3 = {"shanghai", 3, 3};      Insert(head, city3, 3);      print(head);*//打印*      info city4;      Delete(head, &city4, 2);*//删除*      print(head);      Inquire(head, &city4, 2);*//查询*      info city = {"hefei"};      Inquire2(head, &city);*//根据城市名查询*      info point={NULL,0,0};      node\* new\_head=(node\*)malloc(sizeof(node));*//用头节点为new\_head的链表带出所有满足条件的城市*  *if*(new\_head==NULL){          printf("malloc fail in main\n");  *return*;      }      Init(new\_head);      Inquire\_all(head,point,2,new\_head);*//查询在distance范围内的城市*      print(new\_head);  } |

|  |
| --- |
| *#include* <stdio.h>  *#include* <stdlib.h>  *#include* <math.h>  typedef struct node*//节点，双向循环链表*  {      int num;      struct node \*pre;      struct node \*next;  } node;  void Init(node \*head, int n)*//初始化*  {  *//n是num的位数，其符号代表num的符号，位数*      head->next = head;      head->pre = head;      head->num = n;  }  void Insert(node \*head, int e, int pos)  { *//插在第pos个节点之后*      node \*s = (node \*)malloc(sizeof(node));*//申请节点*  *if* (s == NULL)      {          printf("malloc fail in Insert");  *return*;      }      s->num = e;      node \*p = head;  *for* (int i = pos; i > 1; i--)*//找位置*      {          p = p->next;      }      s->next = p->next;*//插入*      p->next = s;      s->next->pre = s;      s->pre = p;  }  int length(node \*head)*//返回链表长度*  {      node \*p = head;      int len = 1;  *while* (p->next != head)*//遍历*      {          len++;          p = p->next;      }  *return* len;  }  void read(node \*head, int m)*//从键盘读入数据*  {      int e;      int n = abs(m);*//去掉符号*  *switch* (n % 4)      {  *case* 1:*//若是1/5/9之类的*          scanf("%1d", &e);*//首位只存一位数字*          Insert(head, e, length(head));  *for* (int i = abs(n - 1) / 4; i > 0; i--)*//假如是9，需要三个节点存储，（除首位外)每个节点存储一个四位数*          {              scanf("%4d", &e);              Insert(head, e, length(head));          }  *break*;  *case* 2:*//若是2/6/10之类的*          scanf("%2d", &e);*//首位存两位数字*          Insert(head, e, length(head));  *for* (int i = abs(n - 1) / 4; i > 0; i--)          {              scanf("%4d", &e);              Insert(head, e, length(head));          }  *break*;  *case* 3:          scanf("%3d", &e);          Insert(head, e, length(head));  *for* (int i = abs(n - 1) / 4; i > 0; i--)          {              scanf("%4d", &e);              Insert(head, e, length(head));          }  *break*;  *case* 0:*//若是四的整数倍*  *for* (int i = abs(n - 1) / 4 + 1; i > 0; i--)*//每个节点存一个四位数*          {              scanf("%4d", &e);              Insert(head, e, length(head));          }  *break*;      }  }  void print(node \*head)*//打印，更清楚的算法应该是找到第一个有效数字，其之前的0都部输出，其之后的空位需要补0*  {      node \*p = head;*//头指针*      p = p->next;*//head不需要打印*  *if*(length(head)==2){*//若是只有一个节点，即在四位数以内，直接打印即可，不需要补0*          printf("%d", p->num);      }  *else* *if*(length(head)>2 && p->num!=0){*//若不只四位数，在首位不为0的情况下，才打印*          printf("%d", p->num);      }      int flag=0;*//用来判断是否到达有效数字（数电的思想），看其从开头到现在是否一直为0，是则flag>0*  *if*(p->num==0 && length(head)>2){*//若不只四位数，在首位为0的情况下，flag==1*          flag=1;      }  *while* (p->next != head)*//遍历链表*      {          p = p->next;  *if*(flag>0 && p->num==0){*//如果当前位为0，且之前一直没到有效数字*              flag++;*//flag加一，说明已经有新的一位为0了*  *if*(flag==abs(head->num)){*//如果从开头到结尾都是0，那至少打印一位数字*                  printf("%d",p->num);  *return*;              }  *continue*;*//尚未找到有效数字时，就不打印*          }  *else* *if*(flag>0 && p->next==head){*//到了最后一位尚未打印，那么至少打印最后一位*              printf("%d",p->num);  *return*;          }  *else*{*//找到有效数字就将flag置0，其后的数字需要补0*              flag=0;          }  *if* (abs(p->num) >= 0 && abs(p->num) <= 9)*//一位数字补三个0*          {              printf("000%d", p->num);          }  *else* *if* (abs(p->num) >= 10 && abs(p->num) <= 99)*//两位数字补两个0*          {              printf("00%d", p->num);          }  *else* *if* (abs(p->num) >= 100 && abs(p->num) <= 999)          {              printf("0%d", p->num);          }  *else*          {              printf("%d", p->num);          }      }      printf("\n");  }  void add(node \*head1, node \*head2, node \*ans)*//加法和减法*  {  *//对齐数字*      int m = (abs(head1->num) - 1) / 4 + 1;*//计算以head1位头节点的链有几个节点*      int n = (abs(head2->num) - 1) / 4 + 1;      int sub = m - n;  *if* (sub > 0)*//如果head1比较长，就给head2从首位开始补0*      {  *for* (int i = sub; i > 0; i--)          {              Insert(head2, 0, 1);          }      }  *else* *if* (sub < 0)      {  *for* (int i = abs(sub); i > 0; i--)          {              Insert(head1, 0, 1);          }      }      int count = m > n ? m : n;*//计算次数，看比较高的位数*      ans->num=count;*//答案的位数*  *//计算*      int flag = 0;*//进位与借位标志*      node \*p1 = head1->pre;*//head1的末尾*      node \*p2 = head2->pre;      int answer = 0;*//每个节点计算结果*  *//两数皆为正数，加法*  *if* (head1->num > 0 && head2->num > 0)      {  *for* (int i = count; i > 0; i--)          {              answer = p1->num + p2->num + flag;*//直接加，再加上进位标志*  *if* (answer > 9999)*//溢出*              {                  flag = 1;*//进位*                  Insert(ans, answer % 10000, 1);              }  *else* *if* (answer >= 0)*//未溢出*              {                  flag = 0;                  Insert(ans, answer, 1);              }              p1 = p1->pre;*//计算更高位*              p2 = p2->pre;          }  *if* (flag == 1)*//如果最高位的加法还有进位*          {              Insert(ans, 1, 1);*//增加一个更高位*          }      }  *//两数皆为负数，算完加法，添个负号即可*  *else* *if* (head1->num < 0 && head2->num < 0)      {  *for* (int i = count; i > 0; i--)          {              answer = abs(p1->num) + abs(p2->num) + flag;*//当加法算*  *if* (answer > 9999)              {                  flag = 1;                  Insert(ans, answer % 10000, 1);              }  *else* *if* (answer >= 0)              {                  flag = 0;                  Insert(ans, answer, 1);              }              p1 = p1->pre;              p2 = p2->pre;          }  *if* (flag == 1)*//最高位有进位*          {              Insert(ans, -1, 1);*//只需要在意最高位的符号，其为负即整个长数为负数（输出的时候）*          }  *else*          {              ans->next->num = -ans->next->num;*//最高位置为负数*          }      }  *//一正一负，减法，需考虑借位*  *else*      {  *for* (int i = count; i > 0; i--)          {              answer = ((head1->num) / abs(head1->num)) \* p1->num + ((head2->num) / abs(head2->num)) \* p2->num + flag;*//带符号的加法*  *if* (answer >= 0)*//结果为非负数*              {                  flag = 0;*//不用借位*                  Insert(ans, answer, 1);              }  *if* (answer < 0)*//结果为负数，插入的其实为补码*              {                  flag = -1;*//借位*  *if* (i == 1)*//算到最后一次（最高位）时，插入对应位数的补码*                  {  *if* (answer == 0)*//零直接插0*                          Insert(ans, 0, 1);  *if* (answer > 0 && answer < 10)*//一位数*                          Insert(ans, 10 + answer, 1);  *if* (answer >= 10 && answer < 100)*//两位数*                          Insert(ans, 100 + answer, 1);  *if* (answer >= 100 && answer < 1000)                          Insert(ans, 1000 + answer, 1);  *if* (answer >= 1000 && answer < 10000)                          Insert(ans, 10000 + answer, 1);                  }  *else//其他位，直接补上用10000的补*                  {                      Insert(ans, 10000 + answer, 1);                  }              }              p1 = p1->pre;*//计算更高位*              p2 = p2->pre;          }  *if* (flag == -1)*//算到最高位还需借位的话，说明之前的（包括最高位）都是补码，需要计算补码的原码*          {              node \*p = ans;  *for* (int i = count; i > 0; i--)              {                  p = p->next;  *if* (i == 1)*//最低位没有被更低位借位，直接用10000求补*                      p->num = 10000 - p->num;  *if* (i == count)*//最高位，用对应位数的来求补码*                  {  *if* (p->num > 0 && p->num < 10)                          p->num =  p->num-9;*//因为是负数，所以减9*  *if* (p->num >= 10 && p->num < 100)                          p->num = p->num-99;  *if* (p->num >= 100 && p->num < 1000)                          p->num = p->num-999;  *if* (p->num >= 1000 && p->num < 10000)                          p->num = p->num-9999;                  }  *else//其他位数被更低位借去了1，so用9999减*                      p->num = 9999 - p->num;              }          }      }  }  int main()*//主函数*  {      node \*head1 = (node \*)malloc(sizeof(node));*//第一个数字的头节点*  *if* (head1 == NULL)      {          printf("malloc fail in main");  *return* 0;      }      printf("please enter the number of digits you will enter,whose symbol stands for the number's symbol: ");      int n;      scanf("%d", &n);*//读入数字的位数，其符号代表第一个数字的符号*  *if*(n==0){          printf("illegal number");  *return* 0;      }      Init(head1, n);*//初始化*      printf("please enter the first number: ");      read(head1, n);*//read读入第一个数字*  *// print(head1);*      node \*head2 = (node \*)malloc(sizeof(node));*//第二个数字的头节点*  *if* (head2 == NULL)      {          printf("malloc fail in main");  *return* 0;      }      printf("please enter the number of digits you will enter,whose symbol stands for the number's symbol: ");      int m;      scanf("%d", &m);  *if*(m==0){          printf("illegal number");  *return* 0;      }      Init(head2, m);      printf("please enter the second number: ");      read(head2, m);  *// print(head2);*      node \*ans = (node \*)malloc(sizeof(node));*//结果的头节点*  *if* (ans == NULL)      {          printf("malloc fail in ans of main");  *return* 0;      }      Init(ans, 0);*//初始化*      add(head1, head2, ans);*//计算结果*      printf("the sum of these numbers is: ");      print(ans);*//打印结果*  } |

* 综合性实验

4.调试分析

1. 采用IDE中自带的调试功能进行调试，手动添加断点和查看程序。
2. 对设计和编码的讨论和分析。该程序实现了顺序栈的操作。分析程序代码的质量，主要从以下几个方面考虑。

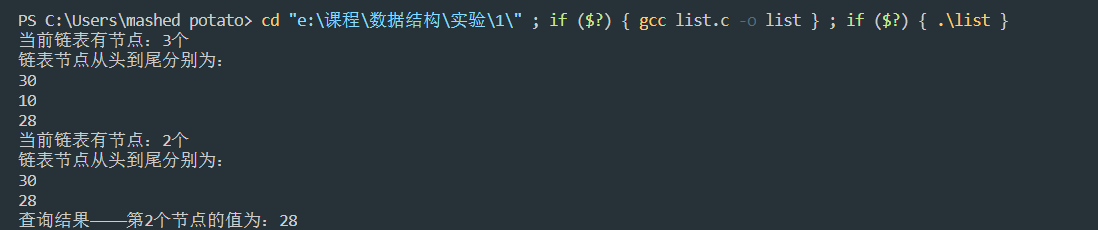
* 正确性。在一定的数据范围内，该程序能实现所需功能，所以正确性是没有问题的。
* 健壮性。在一定的数据输入范围内，该程序能较好的实现操作。但是如果输入数 据非法，该程序还是可能会产生一些预想不到的输出结构，或是不做任何处理。所以， 该程序的健壮性有待进一步的提高。要综合考虑一些情况，当输入有误时，应返回一个 表示错误的值，并中止程序的执行，以便在更高的抽象层次上进行处理。

5.使用说明

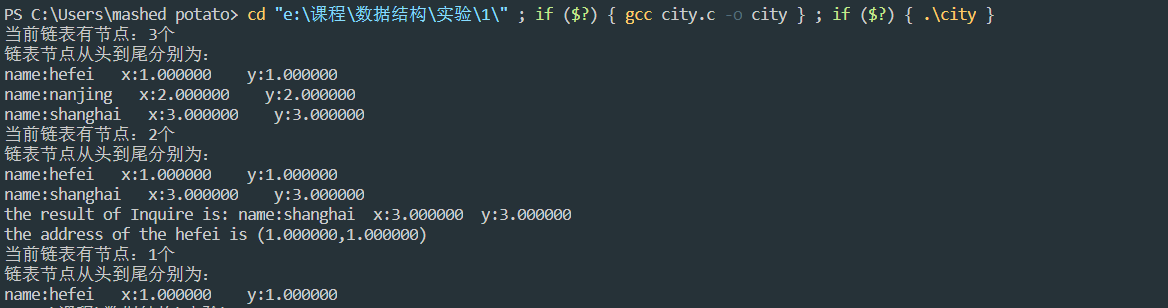
按照屏幕提示，选择想要的功能并输入对应数字，按下ENTER键后，根据屏幕提示进行输入，即可得到想要的结果。

6.测试程序运行结果

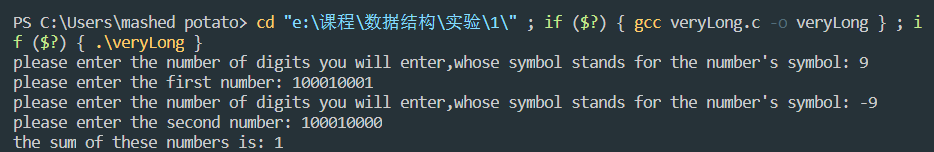
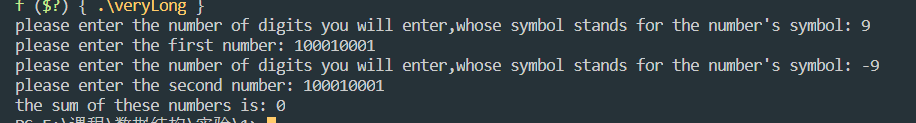
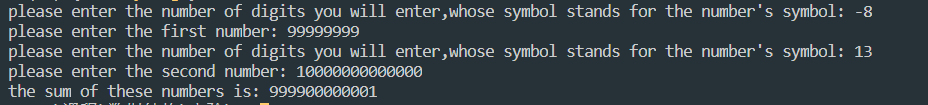
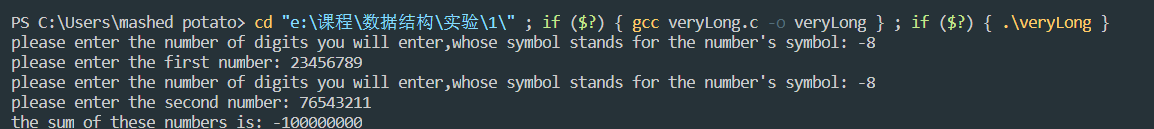
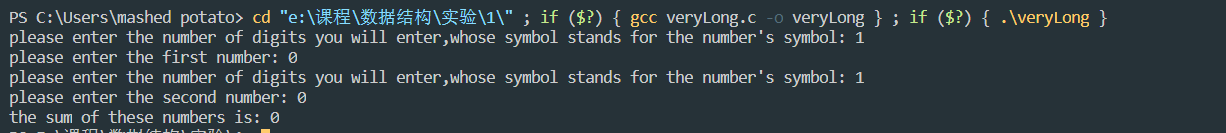
* 验证性实验



* 设计性实验



* 综合性实验



1. **心得体会**

通过本次实验，使我对数据结构有了更深的理解，对指针的运用更加熟练，熟悉了对函数的定义和操作。