附件2：



课程设计报告书

**题目：长整数运算、高精度运算**

**学 院 计算机科学与工程学院**

**专 业 计算机科学与技术**

**学生姓名**

**学生学号**

**指导教师**

**课程编号 145241**

**课程学分 2**

**起始日期 2021-02-01至2021-03-01**

|  |  |
| --- | --- |
| 教  师  评  语 | 教师签名：  日期： |
| 成  绩  评  定 |  |
| 备  注 |  |

**长整数运算、高精度运算**

**一、选题背景**

**【实验目的及要求】**

解决进行加减运算时C++中整型变量和浮点型变量的大小限制，实现任意大小、任意精度的整型、浮点型数据的加减运算。

**【实验原理】**

C++中有建立链表的功能，可以使用链表逐位存储大的、高精度的数据，通过操作链表模拟加减运算。

**【实验环境】**

硬件：ThinkPad e590笔记本电脑、CPU i5-8265U、12G内存

软件：Win10家庭中文版64位操作系统、Visual Studio 2019

**二、方案论证(设计理念)**

说明设计原理（理念）并进行方案选择，阐明为什么要选择这个设计方案以及所采用方案的特点。

包括：重点说明要实现的功能及其要求、系统的安全性、数据的完整性、应用的运行环境及其性能等要求。

**【实验方案设计】**

**1、要求和规格的说明**

**描述问题：**设计一个程序实现两个任意长的整数（包括正数和负数）、任意精度实数的算术运算。

**2、设计**

**设计思想：**

存储结构：

用动态链表存贮数据，每个结点含一个整型变量，表示一位数字

主要算法思想：

模拟竖式算法，对应数位加减，加法满10进1，减法不够借1，根据输入的运算符、输入大数/高精度数的符号调用相应的函数（加法或减法），根据实际情况判断是否需要输出负号。

**设计表示：**

每个函数头和规格说明：

|  |
| --- |
| Node::Node(int d = 0, Node\* a = NULL)--------结点的构造函数  Node::Node(const Node& temp)--------结点的复制构造函数  LongInt::LongInt()-------长整数类对象的构造函数  LongInt::~LongInt() {}-------长整数类对象的构造函数  void LongInt::insert(int d)-------在长整数前端添加一个结点  void LongInt::input()--------输入大数，构造链表  LongInt LongInt::operator+ (LongInt& other)----两个正的长整数相加，结果以长整数形式返回  LongInt LongInt::operator- (LongInt& other)----一个正的较大的长整数减去一个正的较小的长整数，结果以长整数形式返回  bool LongInt::operator<= (LongInt other)------和数学上的<=符号意义相同，返回一个bool值  bool LongInt::operator< (LongInt other) ------------------和数学上的<符号意义相同，返回一个bool值  void LongInt::format()-------去除运算过程中产生的没有意义的0  void LongInt::display()----------把一个长整数对象按人的习惯打印在屏幕上  LongInt LongInt::PutUpsideDown()-------将存储长整数的链表颠倒  HighDegreeNum::HighDegreeNum() -----高精度数类对象的构造函数  HighDegreeNum::~HighDegreeNum()-----高精度数类对象的析构函数  void HighDegreeNum::input()------输入大数，构造两个链表，分别用于存储整数部分和小数部分  HighDegreeNum HighDegreeNum::operator+ (HighDegreeNum& other)---两个正的高精度整数相加，结果以高精度整数形式返回  HighDegreeNum HighDegreeNum::operator-(HighDegreeNum& other)---- 一个正的较大的高精度整数减去一个正的较小的高精度整数，结果以高精度整数形式返回  bool HighDegreeNum::operator>(HighDegreeNum other)--- 和数学上的>符号意义相同，返回一个bool值  bool HighDegreeNum::operator== (HighDegreeNum other)--- 判断两个正的高精度整数是否相等，返回一个bool值  bool HighDegreeNum::operator>= (HighDegreeNum other)----和数学上的>=符号意义相同，返回一个bool值  void HighDegreeNum::format()-------去除运算过程中产生的没有意义的0  void add()------------当运算符为‘+’时调用的函数  void subtract()------------当运算符为‘-’时调用的函数  int main()-------------------------主函数 |

**三、过程论述**

**【实验过程】**

输入的数据存储为链表时，采用的是反向存储，即头结点存储的是末位数字，这是因为，根据竖式思想，无论是整数部分还是小数部分，首先进行运算的都是末位，而为了便于输出，加减运算的结果是得到一个正向的链表，即头结点存储的是首位数字。

定义了一个Symbol枚举类，有两个取值：positive和negative，用于表示正负号。

LongInt类对象有一个Node\*类型的成员head,指向一个链表的头结点，因此，每个LongInt类对象对应一个链表，链表只存储数字。LongInt还包含一个int类型数据length,用于存储长整数的长度。

HighDegreeNum类对象包含两个LongInt对象——分别表示整数部分和小数部分——和一个表示符号的Symbol类型书局sym（默认值位positive）。

输入过程：当读入大数的第一个字符时，判断是负号还是数字，如果是负号，将sym的值改为negative，继续读入下一个字符。如果是数字，将该数字作为结点插入表示整数部分的链表。接下来每遇到一个字符，判断，如果是“.”则转入小数部分的输入，如果是数字则插入链表，如果是换行符就结束输入，如果是其他字符则忽略。如果换行符出现前没有小数点，则小数部分自动存储为0.

运算过程：结合输入的运算符和两个运算数的符号决定具体调用的函数（如运算符为减号，第一个数为正，第二个数为负，则实际调用的是加法函数），先通过在末尾补0的方法把两个运算数的小数部分补成相同长度，再逐位加减，用int类型数据cache存储“进点”和“借点”数据，从末位运算到较长的数的首位，如果仍有“进点”,在存储结果的HighDegreeNum类对象result的前端插入数据1。

输出过程：逐位输出链表数据，根据实际情况在适当的位置输出负号、逗号、小数点。

**1）调试报告：**

当运算结果形如000.00的时候，由于去零的操作，输出的只有“.”，最终通过通过添加“去零后，如果代表整数部分的链表为空，则补上一个0，如果代表小数部分的链表为空，则不输出点”的语句使问题得以解决。

(-0)+(-0)=-0的问题——设置了当结果为0时不加负号的效果。

**2）测试数据：**

|  |
| --- |
| 7  +  1,111,111,111,111,111,111,111,111,111,111,111,111  222,222,222,222,222,222,222,222,222,222,222,222  +  -12,345,678,901,234,567,890,123,456,789,012,345,678,901,234,567,890  98,765,432,198,765,432,198,765,432,198,765,432,198,765,432,198,765  +  3.14159265358979384626  6.66666666666666666666666  -  -41,829,471,284,124,414.49104801248981284  -64,580,569,452,374,277.12783183148912947  +  -0.00000  -00,000.00  -  000123.99  -0.0100000  -  143.99  -11.01 |

**3）实现注释：**

实现了允许带有逗号的输入、输出时附加逗号

实现了正数负数混合加减法

实现了去除整数部分前置0、小数部分后置0

实现了(-0)+(-0)=0

**4）用户手册：**

第一步输入数据组数n，表示将进行n次运算

第二步输入运算符，可以是“+”或“-”，表示要进行的是那种运算

第三步输入第一个数（加数或被减数），以换行符结束，可正可负，可整数可小数

第四步输入第二个数（加数或减数），以换行符结束，可正可负，可整数可小数

两个数都输入后，即显示运算结果，重复第二、三、四步，可以再次进行运算，直至运算了n次，程序运行结束，按任意键退出。

**5）附：**

|  |
| --- |
| 源.cpp |
| /\*  本程序实现了含小数的大数加减运算  \*/  #include"类定义.h"  using std::cin;  using std::cout;  using std::endl;  //运算符为"+"时调用的函数  void add()  {      //提示      cout << "大数加法(含小数):" << endl;      for (int i = 0; i < 45; ++i) cout << "+";cout << endl;      HighDegreeNum num\_one, num\_two, result,Zero;        //Zero表示0      Zero.intPart.insert(0); Zero.decPart.insert(0);      //输入      cout << "请输入加数一:";      num\_one.input();      cout << "请输入加数二:";      num\_two.input();        //正数加正数      if (num\_one.sym == Symbol::positive          && num\_two.sym == Symbol::positive)result = num\_one + num\_two;        //负数加负数      else if (num\_one.sym == Symbol::negative          && num\_two.sym == Symbol::negative)      {          result = num\_one + num\_two; if (!(result==Zero))result.sym = Symbol::negative;      }      //小正数加大负数      else if (num\_one.sym == Symbol::positive          && num\_two.sym == Symbol::negative          && num\_two>=num\_one) {          result = num\_two - num\_one; if (!(result==Zero))result.sym = Symbol::negative;      }      //大正数加小负数      else if (num\_one.sym == Symbol::positive          && num\_two.sym == Symbol::negative          && num\_one> num\_two)result = num\_one - num\_two;      //小负数加大正数      else if (num\_one.sym == Symbol::negative          && num\_two.sym == Symbol::positive          && num\_two>= num\_one) result = num\_two - num\_one;      //大负数加小正数      else if (num\_one.sym == Symbol::negative          && num\_two.sym == Symbol::positive          && num\_one>num\_two) {          result = num\_one - num\_two; if (!(result==Zero))result.sym = Symbol::negative;      }      //去除无意义的零      result.format();      //输出      cout << "程序运行完毕，两数的和为:";      result.display();      cout << endl;      for (int i = 0; i < 45; ++i) cout << "+";      cout << endl << endl;  }  //运算符为"-"时调用的函数  void subtract()  {      //提示      cout << "大数减法(含小数):" << endl;      for (int i=0; i < 45; ++i)cout << "-";cout << endl;      HighDegreeNum num\_one, num\_two, result,Zero;      //Zero表示零      Zero.intPart.insert(0); Zero.decPart.insert(0);      //输入      cout << "请输入被减数:";      num\_one.input();      cout << "请输入减数:";      num\_two.input();      //大正数减小正数      if (num\_one.sym == Symbol::positive          && num\_two.sym == Symbol::positive          && num\_one>= num\_two) result = num\_one - num\_two;      //小正数减大正数      else if (num\_one.sym == Symbol::positive          && num\_two.sym == Symbol::positive          && num\_two> num\_one)      {          result = num\_two - num\_one;          if (!(result == Zero))result.sym = Symbol::negative;      }      //负数减正数      else if (num\_one.sym == Symbol::negative          && num\_two.sym == Symbol::positive)      {          result = num\_one + num\_two;          if (!(result==Zero))result.sym = Symbol::negative;      }      //正数减负数      else if (num\_one.sym == Symbol::positive          && num\_two.sym == Symbol::negative)result = num\_one + num\_two;        //小负数减大负数      else if (num\_one.sym == Symbol::negative          && num\_two.sym == Symbol::negative          && num\_two>= num\_one)  result = num\_two - num\_one;        //大负数减小负数      else if (num\_one.sym == Symbol::negative          && num\_two.sym == Symbol::negative          && num\_one> num\_two)      {  result = num\_one - num\_two; if (!(result == Zero))result.sym = Symbol::negative;      }      //输出      cout << "程序运行完毕，两数的差为:";      result.display();      cout << endl;      for (int i = 0; i < 45; ++i) cout << "-";      cout<< endl << endl;  }  //主函数  int main()  {      cout << "请问你有多少组运算数据:";      int n; cin >> n;      if (n <= 0) { cout << "error"; return n; }      for (int i = 0; i < 45; ++i)cout << "\*";cout << endl;      cout << "若要做加法，请输入加号；若要做减法，请输入减号"<<endl<<endl;      for (int i = 0; i < n; ++i)      {          cout << "请输入运算符:";          char op; cin >> op;          if (op == '+') add();          if (op == '-') subtract();      }      return 0;  } |

|  |
| --- |
| 成员函数实现.cpp |
| #include"类定义.h"  using std::cin;  using std::cout;  using std::endl;  //Node（结点）构造函数  Node::Node(int d = 0, Node\* a = NULL) { data = d; next = a; }  Node::Node(const Node& temp) { data = temp.data; next = temp.next; }  //LongInt（长整数）构造函数、析构函数  LongInt::LongInt() { head = new Node; }  LongInt::~LongInt() {};  //在链表前端插入一个数，成为新的头节点  void LongInt::insert(int d)  {      if (length == 0)head = new Node(d);      else {          Node\* temp = new Node(d, head);          head = temp;      }      length++;  }  //输入长整数，并存储链表（反向链表：低位在前，高位在后）  void LongInt::input() {      char cache;      cin >> cache;      while (cache < '0' || cache>'9')      {          if (cache == '-')sym = Symbol::negative; cin >> cache;      }      while (true)      {          if (cache == '0')cache = getchar(); else break;      }      if (cache == '\n' || cache == '.') { insert(0); return; }      insert(cache - '0');      while (true)      {          cache = getchar();          if (cache == '\n' || cache == '.') break;          if (cache < '0' || cache>'9')continue;          insert(cache - '0');      }  }  //长整数的加法函数  LongInt LongInt::operator+ (LongInt& other)  {      LongInt result;      int temp = 0, i, m, n;      n = length;      m = other.length;      Node\* p = head, \* q = other.head;      for (i = 0; i < n && i < m; i++)      {          int d = p->data + q->data + temp;          temp = d / 10;          result.insert(d % 10);          p = p->next;          q = q->next;      }      for (; i < n; i++)      {          int d = p->data + temp;          temp = d / 10;          result.insert(d % 10);          p = p->next;      }      for (; i < m; i++)      {          int d = q->data + temp;          temp = d / 10;          result.insert(d % 10);          q = q->next;      }      if (temp != 0)          result.insert(temp);      result.format();      return result;  }  //长整数的减法函数  LongInt LongInt::operator- (LongInt& other)  {      LongInt result;      int i, n, m;      n = length;      m = other.length;      Node\* p = head, \* q = other.head;      for (i = 0; i < m; i++)      {          int cache = p->data - q->data;          if (cache < 0)          {              p->next->data -= 1; cache += 10;          }          result.insert(cache);          p = p->next;          q = q->next;      }      for (; i < n; i++)      {          int cache;          cache = p->data;          if (cache < 0 && p->next)          {              p->next->data--;              cache += 10;          }          if (p->next == NULL && p->data == 0)break;          result.insert(cache);          p = p->next;      }      result.format();      return result;  }  //将链表输出为长整数（带逗号）  void LongInt::display() {      int k = 0;      if (sym == Symbol::negative)cout << "-";      Node\* temp = head;      while (temp)      {          cout << temp->data;          temp = temp->next;          k++;          if (k % 3 == length % 3 && temp != NULL) cout << ",";      }  }  //LongInt对运算符<=的重载  bool LongInt::operator<= (LongInt other)  {      if (length < other.length)return 1;      if (length > other.length)return 0;      Node\* p1 = head, \* p2 = other.head;      while (p1 != NULL)      {          if (p1->data < p2->data)return 1;          if (p1->data > p2->data)return 0;          p1 = p1->next;          p2 = p2->next;      }      return 1;  }  //LongInt对运算符<的重载  bool LongInt::operator< (LongInt other)  {      if (other <= \*this)return 0;      else return 1;  }  //去除链表开头的连续的0（适用于正向链表）  void LongInt::format()  {      while (head->data == 0 && head->next != NULL)      {          Node\* x = head; head = head->next; delete x; length--;      };      if (length == 0) insert(0);      return;  }  //颠倒链表，在正向链表（高位在前，低位在后）和反向链表（低位在前，高位在后）之间转换  LongInt LongInt::putUpsideDown()  {      LongInt result; Node\* p = head;      while (p != NULL)      {          result.insert(p->data); p = p->next;      }      return result;  }  //HighDegreeNum（高精度数）构造、析构函数  HighDegreeNum::HighDegreeNum() {}  HighDegreeNum::~HighDegreeNum() {}  //输入高精度数，存储为两个“长整数”（整数部分和小数部分）  void HighDegreeNum::input()  {      char cache;      cin >> cache;      while (cache < '0' || cache>'9')      {          if (cache == '-')sym = Symbol::negative; cin >> cache;      }      while (cache == '0')cache = getchar();      if (cache == '\n') { intPart.insert(0); decPart.insert(0); return; }      if (cache == '.') intPart.insert(0);      else      {          if (cache >= '0' && cache <= '9')intPart.insert(cache - '0');          while (true)          {              cache = getchar();              if (cache == '.') break;              if (cache == '\n') { decPart.insert(0); return; }              if (cache < '0' || cache>'9')continue;              intPart.insert(cache - '0');          }      }        cin >> cache;      while (cache < '0' || cache>'9')cin >> cache;      if (cache == '\n') { decPart.insert(0); return; }      while (true)      {          if (cache == '\n')break;          if (cache < '0' || cache>'9')continue;          decPart.insert(cache - '0');          cache = getchar();      }      //remove meaningless "0"      while (decPart.head->data == 0)      {          Node\* x = decPart.head;          decPart.head = decPart.head->next;          delete x;          decPart.length--;          if (decPart.length == 0) { decPart.insert(0); break; }      }  }  //输出高精度数  void HighDegreeNum::display()  {      if (sym == Symbol::negative)cout << "-";      intPart.display();      if (decPart.length == 1 && decPart.head->data == 0) return;      else      {          cout << ".";          Node\* temp = decPart.head;          while (temp)          {              cout << temp->data;              temp = temp->next;          }      }  }  //高精度数的加法函数  HighDegreeNum HighDegreeNum::operator+ (HighDegreeNum& other)  {      HighDegreeNum result;      int a = decPart.length;      int b = other.decPart.length;      int n, m;      LongInt longer, shorter;      if (a < b) { n = b; m = a; longer = other.decPart; shorter = decPart; }      else { n = a; m = b; longer = decPart; shorter = other.decPart; }      Node\* p = longer.head, \* q = shorter.head;      for (int i = 0; i < n - m; ++i)      {          result.decPart.insert(p->data);          p = p->next;      }      int cache = 0;      for (int i = 0; i < m; ++i)      {          cache = cache + p->data + q->data;          if (cache > 9) { cache -= 10; result.decPart.insert(cache); cache = 1; }          else { result.decPart.insert(cache); cache = 0; }          p = p->next;          q = q->next;      }      result.intPart = intPart + other.intPart;      LongInt temp = result.intPart.putUpsideDown();      if (cache == 1)      {          Node\* x = temp.head;          while (true)          {              if (x == NULL) { x = new Node(1);  temp.length++; break; }              x->data++;              if (x->data == 10) { x->data = 0; x = x->next; continue; }              else { cache = 0; break; }          }      }      result.intPart = temp.putUpsideDown();      result.format();      return result;  }  //高精度数的减法函数  HighDegreeNum HighDegreeNum::operator-(HighDegreeNum& other)  {      HighDegreeNum result;      //decPart      int m = decPart.length;      int n = other.decPart.length;      for (int i = 0; i < n - m; ++i)decPart.insert(0);      for (int i = 0; i < m - n; ++i)other.decPart.insert(0);      int a = decPart.length;      int cache = 0;      Node\* p = decPart.head;      Node\* q = other.decPart.head;      for (int i = 0; i < a; ++i)      {          cache = p->data - cache - q->data;          if (cache < 0) { cache += 10; result.decPart.insert(cache); cache = 1; }          else { result.decPart.insert(cache); cache = 0; }          p = p->next;          q = q->next;      }      if (cache == 1)      {          Node\* x = intPart.head;          while (true)          {              if (x->data != 0) { x->data--; break; }              else x->data = 9;              x = x->next;          }      }      result.intPart = intPart - other.intPart;      result.format();      return result;  }  //HighDegreeNum对运算符>的重载  bool HighDegreeNum::operator>(HighDegreeNum other)  {      if (intPart < other.intPart)return 0;      if (other.intPart < intPart)return 1;      LongInt x1 = decPart, x2 = other.decPart;      int m = x1.length;      int n = x2.length;      for (int i = 0; i < n - m; ++i)x1.insert(0);      for (int i = 0; i < m - n; ++i)x2.insert(0);      if (x1 <= x2)return 0;      return 1;  }  //HighDegreeNum对运算符==的重载  bool HighDegreeNum::operator== (HighDegreeNum other)  {      if (\*this > other)return 0;      if (other > \*this)return 0;      else return 1;  }  //HighDegreeNum对运算符>=的重载  bool HighDegreeNum::operator>= (HighDegreeNum other)  {      if (\*this > other)return 1;      if (\*this == other)return 1;      return 0;  }  //去除整数部分前导0和小数部分后导0  void HighDegreeNum::format()  {      intPart.format();      LongInt temp = decPart.putUpsideDown();      temp.format();      decPart = temp.putUpsideDown();      return;  } |

|  |
| --- |
| 类定义.h |
| #pragma once  #include<iostream>  //符号类  enum class Symbol :bool { negative, positive };  //结点类  //包含一个数位  struct Node  {  public:      int data;      Node\* next;      Node(int, Node\*);      Node(const Node&);  };  //长整数类  //存储一个符号  //和一个链表  class LongInt  {      friend class HighDegreeNum;  private:      Node\* head;      Symbol sym = Symbol::positive;      int length = 0;  public:      LongInt();      ~LongInt();      void insert(int);      void display();      void input();      LongInt operator+ (LongInt&);      LongInt operator- (LongInt&);      bool operator< (LongInt);      bool operator<= (LongInt);      void format();      LongInt putUpsideDown();  };  //高精度数类  //存储一个符号  //和两个LongInt(整数部分和小数部分)  class HighDegreeNum  {      friend void add();      friend void subtract();  private:LongInt intPart; LongInt decPart; Symbol sym = Symbol::positive;  public:      HighDegreeNum();      ~HighDegreeNum();      void input();      void display();      HighDegreeNum operator+ (HighDegreeNum&);      HighDegreeNum operator-(HighDegreeNum&);      void format();      bool operator> (HighDegreeNum);      bool operator== (HighDegreeNum);      bool operator>= (HighDegreeNum);  }; |

**四、结果分析**

经过对算法的分析，得出以下结论：由于链表只有一个入口，访问任何结点都需要通过这个入口，当需要倒序访问链表中的结点时，这增加了大量时间消耗，所以，我认为应当使用更便捷的方式，如数组或string类。以降低时间复杂度。

另外，我发现，功能如此单一的小型程序，竟然也有这么高的时间复杂度，难以想象一个大型程序如何做到快速运行，这让我意识到精简算法的重要性，不能只顾实现功能而忽略时间消耗。

**五、课程设计总结**

程序是要边调试边修改的，应该做到每实现一项功能就做一次调试，这样比较容易发现问题产生的位置，一项功能完善后再进行下一项的编写，这样才不至于到全部代码写好后对莫名其妙的结果手足无措。

注释是代码可读性的至上法宝，勤于写注释、改注释不仅有利于他人读代码，也有利于自己查找错误，是应该养成的好习惯。

计算机最根本的用途就是计算，可以说，计算机所有的应用都是建立在计算的基础上的，这次的《大数加减》大作业，让我更加深刻地体会到了这一点。多做类似的基础小程序，有利于加深我对C++语法和计算机科学原理的理解，我深知这还远不算计算机的真正“应用”，但我想，我在这条路上的第一步，已经迈出去了。