**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра САПР**

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ**

**по дисциплине «Программирование»**

**Тема: Длинная арифметика**

Студент гр. 2302 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Николаев В.Ю.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Калмычков В.А.

Санкт-Петербург

2023 г.

**ЗАДАНИЕ**

**НА КУРСОВУЮ РАБОТУ**

| Студент Николаев Всеволод Юрьевич | | |
| --- | --- | --- |
| Группа 2302 | | |
| Тема работы: Длинная арифметика | | |
| Исходные данные:  На вход поступает файл in.txt, в котором находятся координаты точек. | | |
| Содержание пояснительной записки:  Исходная формулировка, Цель работы, Организация UI, Особенности системы программирования, Структуры данных, Способ хранения данных, Описание основных функций, Представление алгоритма решения задачи, Текст программы, Подготовка контрольных примеров, Анализ полученных результатов, Выводы о проделанной работе | | |
| Предполагаемый объем пояснительной записки:  Не менее 10 страниц. | | |
| Дата выдачи задания: 20.02.2023 | | |
| Дата сдачи реферата: | | |
| Дата защиты реферата: | | |
| Студент |  | Николаев В.Ю. |
| Преподаватель |  | Калмычков В.А. |

**АННОТАЦИЯ**

У каждого человека имеется набор больших чисел, с которыми он хочет выполнять различные математические операции: сложение, вычитание, умножение, деление и взятие остатка от деления. В каждом наборе существуют числа, которые имеют больше 9 и даже 18 знаков, что больше максимально возможного количества знаков во всех типах данных. Производить данные операции в ручную не хочется, а многие калькуляторы не умеют делать это в столбик. Что делать? Автор данной курсовой работы решил эту проблему! Теперь любой человек может найти сумму, разность, произведение, частное двух чисел и не только!

**SUMMARY**

Each person has a set of large numbers with which he wants to perform various mathematical operations: addition, subtraction, multiplication, division, and taking the remainder of the division. In each set, there are numbers that have more than 9 and even 18 characters, which is more than the maximum possible number of characters in all data types. I do not want to perform these operations manually, and many calculators do not know how to do this in a column. What to do? The author of this course work solved this problem! Now anyone can find the sum, difference, product, quotient of two numbers and more!

# 

[**Исходная формулировка 5**](#_wukrm6x68ot2)

[**Цель работы 5**](#_xi82tyx4x9ci)

[**Организация UI 5**](#_cnvquz9mm7f)

[**Особенности системы программирования 7**](#_sjnoustmz58i)

[**Структуры данных 7**](#_9zvuyy1peu81)

[**Способ хранения данных 8**](#_fojvabamf37a)

[**Описание основных функций 8**](#_6ch2ifyjg7de)

[**Представление алгоритма решения задачи 10**](#_ik0vsjaar38u)

[**Текст программы 10**](#_xgevv17qc4r2)

[**Подготовка контрольных примеров 28**](#_iiw88igvg9du)

[**Анализ полученных результатов 28**](#_hdwafdtr7nrf)

[**Выводы о проделанной работе 28**](#_aldurkp3t2c)

# 

# Исходная формулировка

Школьная арифметика с визуализацией действий «в столбик», реализованная на основе “длинной арифметики” (учесть размер экрана и т. п.). Списочное представление в динамической памяти. Операции: +, , \*, div, mod.

# Цель работы

Демонстрация навыков приобретённых в ходе выполнения лабораторных работ, а именно: работа с файлами, создание классов, работа с односвязными и двусвязными списками, работа со строками

# **Организация UI**

Макет O1:

Choose what do you want to do:\n

\t1) Add a number to the list;\n

\t2) Remove a number from the list;\n

\t3) Display a list of numbers on the screen;\n

\t4) Add 2 numbers;\n

\t5) Subtract another from one number;\n

\t6) Multiply 2 numbers;\n

\t7) Divide one number by another;\n

\t8) Get the remainder from dividing one number by another;\n

\t9) Save the list to a file;\n

\t10) Load list from a file;\n

\t11) End the program.\n

Макет O2:

Enter the number what you want to add to the list:\n

Макет O3:

List is clear.\n

Макет O4:

Enter the index of the number you want to remove:\n

Макет O5:

Do you want to save the list to a file before exiting (y / n)?\n

Макет O6:

Invalid input. Try again:\n

Макет O7:

Goodbye!\n

Макет O8:

Do you want to save this number in the list (y / n)?\n

Макет O9:

Error: file not found.\n

Макет O10:

List saved.\n

Макет O11:

List is empty.\n

Макет O12:

Size of the list: " << list.size << ".\n

Макет O13:

List:\n

Макет O14:

n\_n

Макет O15:

List loaded.\n

Макет O16:

Enter the first number:\n

Макет O17:

Choose how you want to enter the first number:\n

\t1) Number from the list;\n

\t2) Number from the keyboard.\n

Макет O18:

Enter the index of the first number:\n

Макет O19:

Enter the first number:\n

Макет O20:

Enter the second number:\n

Макет O21:

Choose how you want to enter the second number:\n

\t1) Number from the list;\n

\t2) Number from the keyboard.\n

Макет O22:

Enter the index of the second number:\n

Макет O23:

Enter the second number:\n

| Макет AIC1:  /---\  |n\_n|  |n\_n|  |---|  |n\_n|  \---/  Макет AIC3:  /------\  | n\_n|  | \*n\_n|  |------|  | n\_n|  | n\_n |  | ... |  |n\_n |  |------|  | n\_n|  \------/  Макет I1:  n\_n | Макет AIC2:  /----\  |\_n\_n|  | n\_n|  |----|  | n\_n|  \----/  Макет AIC4:  /----------\  |\_n\_n |n\_n|  | n\_n |---|  | ---- |n\_n|  | \_n\_n |  | n\_n |  | ---- |  | \_n\_n |  | ... |  | --- |  | n\_n |  \----------/  Макет I2:  c |
| --- | --- |

# Особенности системы программирования

Проект собран на g++ (Ubuntu 11.3.0-1ubuntu1~22.04.1) 11.3.0. Для сборки используется GNU Make 4.3 Built for x86\_64-pc-linux-gnu.

# Структуры данных

| Строка | Число | Список чисел |
| --- | --- | --- |
| class String  {  private:  StringNode \*root = nullptr;  public:  int size = 0;  public:  String();  String(const String &);  ~String();  void pushBack(char);  void pushBack(char \*);  void pushBack(String);  void clear();  bool isInt();  int toInt();  friend bool operator==(const String &, const String &);  friend std::ostream &operator<<(std::ostream &, const String &);  friend std::istream &operator>>(std::istream &, String &);  char &operator[](const int) const;  String operator=(const String &);  }; | class Number  {  private:  NumberNode \*lEnd = nullptr;  NumberNode \*rEnd = nullptr;  bool sign = true;  public:  int size = 0;  private:  bool isEmpty() const;  void deleteLeftZeroes();  void pushLeft(int, int);  void pushRight(int);  void shiftRight();  Number multiplyByDigit(const int &) const;  public:  Number();  Number(const int &);  Number(const Number &);  ~Number();  String debugOutput() const;  void clear();  friend std::ostream &operator<<(std::ostream &, const Number &);  friend std::istream &operator>>(std::istream &, Number &);  bool operator==(const Number &) const;  bool operator!=(const Number &) const;  bool operator<(const Number &) const;  bool operator>(const Number &) const;  bool operator<=(const Number &) const;  bool operator>=(const Number &) const;  Number operator-() const;  Number operator=(const Number &);  Number operator-(const Number &) const;  Number operator+(const Number &) const;  Number operator-=(const Number &);  Number operator+=(const Number &);  Number operator\*(const Number &) const;  Number operator/(const Number &) const;  Number operator\*=(const Number &);  Number operator/=(const Number &);  Number operator%(const Number &) const;  Number operator%=(const Number &);  }; | class List  {  private:  ListNode \*begin = nullptr;  ListNode \*end = nullptr;  public:  int size = 0;  public:  List();  List(const List &);  ~List();    void pushBack(const Number &);    void pushFront(const Number &);  void remove(int);  void clear();  bool isEmpty() const;    Number &operator[](int) const;  List operator=(const List &);  }; |
| StringNode \*root = nullptr - корень строки  int size = 0 - длина строки | NumberNode \*lEnd = nullptr - левый конец числа  NumberNode \*rEnd = nullptr - правый конец числа  bool sign = true - знак числа  int size = 0 - размер числа | ListNode \*begin = nullptr - начало списка  ListNode \*end = nullptr - конец списка  int size = 0 - размер списка |
|  |  |  |

# **Способ хранения данных**

| Тип данных | Название переменной | Описание переменной |
| --- | --- | --- |
| fstream | lout | Видение протокола |
| List | list | Список используемых чисел |
| int | choice | Выбор того, что человек хочет сделать |
| Number | number | Число, введённое пользователем |
| number1 | Первое число, введённое или выбранное пользователем |
| number2 | Второе число, введённое или выбранное пользователем |

# Описание основных функций

| Тип | Название | Передаваемые переменные | Описание переменных | Описание функции |
| --- | --- | --- | --- | --- |
| int | inputNumber | int min | минимальное значение | Ввод числа от минимального до максимального значения |
| int max | максимальное значение |
| Number | inputNumber | - | - | Ввод длинного числа |
| Number | inputNumber | List &list | список чисел | Ввод длинного числа и возможная запись этого числа в список |
| void | inputNumber | List &list | список чисел | Ввод и/или выбор из списка пары чисел |
| Number &number1 | первое число |
| Number &number2 | второе число |
| void | saveNumber  InList | List &list | список чисел | Сохранение числа в список |
| const Number &number | число |
| void | saveListIn  File | List &list | список чисел | Сохранение списка в файл |
| void | loadList  FromFile | List &list | список чисел | Запись чисел из файла в список |
| void | printList | List &list | список чисел | Вывод списка на экран |
| void | addition  IntoAColumn | const Number &number1 | первое число | Сложение двух чисел в столбик |
| const Number &number2 | второе число |
| List &list | список чисел |
| void | subtraction  IntoAColumn | const Number &number1 | первое число | Вычитание двух чисел в столбик |
| const Number &number2 | второе число |
| List &list | список чисел |
| void | multiplication  IntoAColumn | const Number &number1 | первое число | Умножение двух чисел в столбик |
| const Number &number2 | второе число |
| List &list | список чисел |
| void | division  IntoAColumn | const Number &number1 | первое число | Деление с остатком двух чисел в столбик. В результат записывается частное |
| const Number &number2 | второе число |
| List &list | список чисел |
| void | remainder  IntoAColumn | const Number &number1 | первое число | Деление с остатком двух чисел в столбик. В результат записывается остаток |
| const Number &number2 | второе число |
| List &list | список чисел |

# Текст программы

| MakeFile  PROJECT\_ROOT := $(shell pwd)  all:  cd build/; mkdir -p Number/Node; mkdir -p String/Node; mkdir -p List/Node; mkdir -p UserInterface; cd $(PROJECT\_ROOT)  cd build/; rm -f \*/\*.o \*/\*/\*.o; cd $(PROJECT\_ROOT)  cd build/Number/Node/; g++ -c -Wall $(PROJECT\_ROOT)/lib/Number/Node/\*.cpp; cd $(PROJECT\_ROOT)  cd build/Number/; g++ -c -Wall $(PROJECT\_ROOT)/lib/Number/\*.cpp; cd $(PROJECT\_ROOT)  cd build/String/Node/; g++ -c -Wall $(PROJECT\_ROOT)/lib/String/Node/\*.cpp; cd $(PROJECT\_ROOT)  cd build/String/; g++ -c -Wall $(PROJECT\_ROOT)/lib/String/\*.cpp; cd $(PROJECT\_ROOT)  cd build/List/Node/; g++ -c -Wall $(PROJECT\_ROOT)/lib/List/Node/\*.cpp; cd $(PROJECT\_ROOT)  cd build/List/; g++ -c -Wall $(PROJECT\_ROOT)/lib/List/\*.cpp; cd $(PROJECT\_ROOT)  cd build/UserInterface/; g++ -c -Wall $(PROJECT\_ROOT)/lib/UserInterface/\*.cpp; cd $(PROJECT\_ROOT)  g++ -Wall build/\*/\*.o build/\*/\*/\*.o src/Main.cpp -o build/build.out  ./build/build.out < files/in |
| --- |

| | main.cpp  #include "../lib/UserInterface/UserInterface.hpp"  int main()  {  lout << "Program started." << std::endl;  List list;  lout << "List created." << std::endl;  bool end = false;  lout << "The main loop started." << std::endl;  while (!end)  {  lout << "The main loop iteration started." << std::endl;  std::cout << "Choose what do you want to do:" << std::endl  << "\t1) Add a number to the list;" << std::endl  << "\t2) Remove a number from the list;" << std::endl  << "\t3) Display a list of numbers on the screen;" << std::endl  << "\t4) Add 2 numbers;" << std::endl  << "\t5) Subtract another from one number;" << std::endl  << "\t6) Multiply 2 numbers;" << std::endl  << "\t7) Divide one number by another;" << std::endl  << "\t8) Get the remainder from dividing one number by another;" << std::endl  << "\t9) Save the list to a file;" << std::endl  << "\t10) Load list from a file;" << std::endl  << "\t11) End the program." << std::endl;  int choice = inputNumber(1, 11);  lout << "The user entered the number " << choice << "." << std::endl;  switch (choice)  {  case 1:  {  std::cout << "Enter the number what you want to add to the list:" << std::endl;  Number number = inputNumber();  lout << "The user entered the number [" << number.debugOutput() << "]." << std::endl;  list.pushBack(number);  lout << "The number [" << number.debugOutput() << "] was added to the list." << std::endl;  }  break;  case 2:  {  if (list.size == 0)  {  lout << "The list is empty." << std::endl;  std::cout << "List is clear." << std::endl;  break;  }  std::cout << "Enter the index of the number you want to remove:" << std::endl;  int index = inputNumber(1, list.size); index--;  lout << "The user entered the index = " << index << "." << std::endl;  list.remove(index);  lout << "The number with index = " << index << " was removed from the list." << std::endl;  }  break;  case 3:  {  printList(list);  lout << "The list will be displayed on the screen." << std::endl;  }  break;  case 4:  {  Number number1, number2;  inputNumber(list, number1, number2);  lout << "The user entered the numbers [" << number1.debugOutput() << "] and [" << number2.debugOutput() << "]." << std::endl;  additionIntoAColumn(number1, number2, list);  }  break;  case 5:  {  Number number1, number2;  inputNumber(list, number1, number2);  lout << "The user entered the numbers [" << number1.debugOutput() << "] and [" << number2.debugOutput() << "]." << std::endl;    subtractionIntoAColumn(number1, number2, list);  }  break;    case 6:  {  Number number1, number2;  inputNumber(list, number1, number2);  lout << "The user entered the numbers [" << number1.debugOutput() << "] and [" << number2.debugOutput() << "]." << std::endl;  multiplicationIntoAColumn(number1, number2, list);  }  break;  case 7:  {  Number number1, number2;  inputNumber(list, number1, number2);  lout << "The user entered the numbers [" << number1.debugOutput() << "] and [" << number2.debugOutput() << "]." << std::endl;    divisionIntoAColumn(number1, number2, list);  }  break;  case 8:  {  Number number1, number2;  inputNumber(list, number1, number2);  lout << "The user entered the numbers [" << number1.debugOutput() << "] and [" << number2.debugOutput() << "]." << std::endl;    remainderIntoAColumn(number1, number2, list);  }  break;  case 9:  {  saveListInFile(list);  lout << "The list was saved to a file." << std::endl;  }  break;  case 10:  {  loadListFromFile(list);  lout << "The list was loaded from a file." << std::endl;  }  break;  case 11:  {  std::cout << "Do you want to save the list to a file before exiting (y / n)?" << std::endl;    char answer = 'n';  std::cin >> answer;  while (answer != 'y' && answer != 'n' && answer != 'Y' && answer != 'N')  {  std::cout << "Invalid input. Try again:" << std::endl;  std::cin >> answer;  }    if (answer == 'y')  {  saveListInFile(list);  lout << "The list was saved to a file." << std::endl;  }  std::cout << "Goodbye!" << std::endl;  end = true;  }  break;  }  }  lout << "Program completed." << std::endl;  return 0;  } | | --- | | lib/String/Node/Operator.cpp  #include "StringNode.hpp"  StringNode StringNode::operator=(const StringNode &other)  {  size = other.size;  data = new char[l + 1];  for (int i = 0; i < size; i++)  data[i] = other.data[i];    for (int i = size; i < l; i++)  data[i] = '0';  if (other.next != nullptr)  next = new StringNode(\*other.next);  return \*this;  } | | lib/String/Node/StringNode.cpp  #include "StringNode.hpp"  StringNode::StringNode()  {  data = new char[l + 1];  for (int i = 0; i < l; i++)  data[i] = '0';  }  StringNode::StringNode(const StringNode &node)  {  size = node.size;  data = new char[l + 1];  for (int i = 0; i < size; i++)  data[i] = node.data[i];  for (int i = size; i < l; i++)  data[i] = '0';    if (node.next != nullptr)  next = new StringNode(\*node.next);  }  StringNode::~StringNode()  {  delete next;  next = nullptr;  delete[] data;  data = nullptr;  } | | lib/String/Node/StringNode.hpp  #pragma once  #include "../../Header.hpp"  class StringNode  {  public:  char \*data = nullptr;  static const int l = 10;  int size = 0;  StringNode \*next = nullptr;  public:  StringNode();  StringNode(const StringNode &);  ~StringNode();  StringNode operator=(const StringNode &);  }; | | lib/String/Clear.cpp  #include "String.hpp"  void String::clear()  {  delete root;  root = nullptr;  size = 0;  } | | lib/String/IsInt.cpp  #include "String.hpp"  bool String::isInt()  {  if (size == 0)  return false;    StringNode \*cur = root;    int i = 0;  if (cur->data[0] == '-' || cur->data[0] == '+')  i++;    for (; i < size; i++)  {  char c = cur->data[i % cur->l];  if (c < '0' || c > '9')  return false;  }    return true;  } | | lib/String/Operator.cpp  #include "String.hpp"  bool operator==(const String &left, const String &right)  {  if (left.size == right.size)  {  StringNode \*lcur = left.root;  StringNode \*rcur = right.root;    while (lcur != nullptr)  if (lcur != rcur)  return false;  else  {  lcur = lcur->next;  rcur = rcur->next;  }  }    return left.size == right.size;  }  std::ostream &operator<<(std::ostream &os, const String &str)  {  StringNode \*cur = str.root;  while (cur != nullptr)  {  for (int i = 0; i < cur->size; i++)  if (cur->data[i] != '\r')  os << cur->data[i];    cur = cur->next;  }  return os;  }  std::istream &operator>>(std::istream &is, String &str)  {  is.unsetf(std::ios::skipws);  char c;  while(is >> c && c != '\n')  str.pushBack(c);    return is;  }  char &String::operator[](const int n) const  {  assert((void("E1"), 0 <= n));  assert((void("E1"), n < size));  StringNode \*cur = root;  int i = n;  for (; i > root->l; i -= root->l)  cur = cur->next;  return cur->data[i];  }  String String::operator=(const String &str)  {  size = str.size;  if (str.root != nullptr)  root = new StringNode(\*str.root);  return \*this;  } | | lib/String/PushBack.cpp  #include "String.hpp"  void String::pushBack(char c)  {  size++;  if (root == nullptr)  {  root = new StringNode();  root->data[root->size++] = c;  return;  }  StringNode \*cur = root;  while (cur->next != nullptr)  cur = cur->next;    if (cur->size == root->l)  {  cur->next = new StringNode();  cur = cur->next;  cur->data[cur->size++] = c;  return;  }    cur->data[cur->size++] = c;  }  void String::pushBack(char \*str)  {  for (int i = 0; int(str[i]) != '\0'; i++)  pushBack(str[i]);  }  void String::pushBack(String str)  {  for (int i = 0; i < str.size; i++)  pushBack(str[i]);  } | | lib/String/String.cpp  #include "String.hpp"  String::String()  {}  String::String(const String &string)  {  size = string.size;  if (string.root != nullptr)  root = new StringNode(\*string.root);  }  String::~String()  {  delete root;  root = nullptr;  } | | lib/String/String.hpp  #pragma once  #include "Node/StringNode.hpp"  class String  {  private:  StringNode \*root = nullptr;  public:  int size = 0;  public:  String();  String(const String &);  ~String();  void pushBack(char);  void pushBack(char \*);  void pushBack(String);  void clear();  bool isInt();  int toInt();  friend bool operator==(const String &, const String &);  friend std::ostream &operator<<(std::ostream &, const String &);  friend std::istream &operator>>(std::istream &, String &);  char &operator[](const int) const;  String operator=(const String &);  };  char \*toChar(String); | | lib/String/ToChar.cpp  #include "String.hpp"  char \*toChar(String string)  {  char \*result = new char[string.size];    for (int i = 0; i < string.size; i++)  result[i] = string[i];    return result;  } | | lib/String/ToInt.cpp  #include "String.hpp"  int String::toInt()  {  assert((void("E6"), size == 0 or isInt()));  StringNode \*cur = root;  int res = 0;  int sign = 1;  int i = 0;  if (cur->data[0] == '-')  {  sign = -1;  i++;  }  else if (cur->data[0] == '+')  i++;  for (; i < size; i++)  {  res \*= 10;  res += cur->data[i % cur->l] - '0';  }  return res \* sign;  } | | lib/Number/Node/NumberNode.cpp  #include "NumberNode.hpp"  NumberNode::NumberNode()  {}  NumberNode::NumberNode(int \_data)  {  data = \_data;  size = 1;  }  NumberNode::NumberNode(int \_data, int \_size)  {  data = \_data;  size = \_size;  }  NumberNode::NumberNode(const NumberNode &node)  {  data = node.data;  size = node.size;  }  NumberNode::~NumberNode()  {} | | lib/Number/Node/NumberNode.hpp  #pragma once  #include "../../Header.hpp"  class NumberNode  {  public:  static const int l = 8;  int data = 0;  int size = 0;  NumberNode \*left = nullptr;  NumberNode \*right = nullptr;  public:  NumberNode();  NumberNode(int);  NumberNode(int, int);  NumberNode(const NumberNode &);  ~NumberNode();  NumberNode operator=(const NumberNode &);  }; | | lib/Number/Node/Operator.cpp  #include "NumberNode.hpp"  NumberNode NumberNode::operator=(const NumberNode &node)  {  data = node.data;  size = node.size;  return \*this;  } | | lib/Number/Clear.cpp  #include "Number.hpp"  void Number::clear()  {  NumberNode \*cur = lEnd;  while (cur != nullptr)  {  NumberNode \*tmp = cur;  cur = cur->right;  delete tmp;  }  lEnd = nullptr;  rEnd = nullptr;  size = 0;  sign = true;  } | | lib/Number/DeleteLeftZeroes.cpp  #include "Number.hpp"  void Number::deleteLeftZeroes()  {  if (isEmpty())  return;  if (\*this == 0 && size == 1)  return;  if (-\*this == 0 && size == 1)  {  sign = true;  return;  }  NumberNode \*cur = lEnd;  while (cur != nullptr && cur->data == 0 && cur != rEnd)  {  NumberNode \*tmp = cur;  size -= cur->size;  cur = cur->right;  delete tmp;  }  if (cur == nullptr)  {  clear();  return;  }  lEnd = cur;  lEnd->left = nullptr;    for (int i = cur->size - 1; i >= (cur == rEnd ? 1 : 0); i--)  if (cur->data / (int)pow(10, i) % 10 != 0)  break;  else  {  cur->size--;  size--;  }    if (-\*this == 0)  sign = true;  } | | lib/Number/MultiplyByDigit.cpp  #include "Number.hpp"  Number Number::multiplyByDigit(const int &digit) const  {  Number result;  if (isEmpty())  return result;  if (digit == 0)  return 0;  if (digit == 1)  return \*this;  if (digit == -1)  {  result = \*this;  result.sign = !sign;  return result;  }  result.sign = sign == (digit > 0);  NumberNode \*cur1 = rEnd;  int tenPowerL = (int)pow(10, NumberNode::l);  int carry = 0;  while (cur1 != nullptr)  {  int sum = cur1->data \* digit + carry;  carry = sum / tenPowerL;  sum %= tenPowerL;  cur1 = cur1->left;  result.pushLeft(sum, NumberNode::l);  }  if (carry != 0)  result.pushLeft(carry, 1);  result.deleteLeftZeroes();  return result;  } | | lib/Number/Number.cpp  #include "Number.hpp"  Number::Number()  {}  Number::Number(const int &number)  {  if (number == 0)  {  rEnd = new NumberNode(0);  lEnd = rEnd;    sign = true;  size = 1;  return;  }  sign = number >= 0;  pushRight(sign ? number : -number);  shiftRight();  }  Number::Number(const Number &number)  {  if (number.isEmpty())  return;  sign = number.sign;  size = number.size;  lEnd = new NumberNode(number.lEnd->data, number.lEnd->size);    NumberNode \*cur1 = lEnd;  NumberNode \*cur2 = number.lEnd;  while (cur2->right != nullptr)  {  cur1->right = new NumberNode(cur2->right->data, cur2->right->size);  cur1->right->left = cur1;    cur1 = cur1->right;  cur2 = cur2->right;  }  rEnd = cur1;  }  Number::~Number()  {  clear();  } | | lib/Number/Operator.cpp  #include "Number.hpp"  std::ostream &operator<<(std::ostream &os, const Number &number)  {  if (number.isEmpty())  return os;  if (number.sign == false)  os << '-';  NumberNode \*cur = number.lEnd;  while (cur != nullptr)  {  for (int i = 0; i < std::min(cur->size, 8); i++)  os << cur->data / (int)pow(10, cur->size - i - 1) % 10;  cur = cur->right;  }  return os;  }  std::istream &operator>>(std::istream &is, Number &number)  {  number.clear();  is.unsetf(std::ios::skipws);  char c;  while (is >> c && (c == ' ' || c == '\n' || c == '\r'));  if (c == '-' || c == '+')  {  number.sign = c == '+';  is >> c;  }    do  {  if (c < '0' || '9' < c)  {  is.setstate(std::ios::badbit);  return is;  }  number.pushRight(c - '0');  } while(is >> c && !is.eof() && (c != ' ' && c != '\n' && c != '\r'));  number.shiftRight();  number.deleteLeftZeroes();  is.setf(std::ios::skipws);  return is;  }  bool Number::operator==(const Number &number) const  {  if (isEmpty())  return number.isEmpty();  if (number.isEmpty())  return false;  if (sign != number.sign)  return false;  NumberNode \*cur1 = lEnd;  NumberNode \*cur2 = number.lEnd;  while (cur1 != nullptr && cur2 != nullptr)  {  if (cur1->data != cur2->data)  return false;  cur1 = cur1->right;  cur2 = cur2->right;  }  bool res = cur1 == nullptr && cur2 == nullptr;  cur1 = nullptr;  cur2 = nullptr;  return res;  // return cur1 == nullptr && cur2 == nullptr;  }  bool Number::operator!=(const Number &number) const  {  return !(\*this == number);  }  bool Number::operator<(const Number &number) const  {  if (isEmpty())  return number.isEmpty();  if (number.isEmpty())  return false;  if (sign != number.sign)  return !sign;  if (sign == false)  return number < \*this;  if (size != number.size)  return size < number.size;  NumberNode \*cur1 = lEnd;  NumberNode \*cur2 = number.lEnd;  while (cur1 != nullptr)  {  if (cur1->data != cur2->data)  return cur1->data < cur2->data;  cur1 = cur1->right;  cur2 = cur2->right;  }  return false;  }  bool Number::operator>(const Number &number) const  {  return number < \*this;  }  bool Number::operator<=(const Number &number) const  {  return !(\*this > number);  }  bool Number::operator>=(const Number &number) const  {  return !(\*this < number);  }  Number Number::operator-() const  {  Number result = \*this;  result.sign = !sign;  return result;  }  Number Number::operator=(const Number &number)  {  if (number.isEmpty())  {  clear();  return \*this;  }  sign = number.sign;  size = number.size;  lEnd = new NumberNode(number.lEnd->data, number.lEnd->size);    NumberNode \*cur1 = lEnd;  NumberNode \*cur2 = number.lEnd;  while (cur2->right != nullptr)  {  cur1->right = new NumberNode(cur2->right->data, cur2->right->size);  cur1->right->left = cur1;    cur1 = cur1->right;  cur2 = cur2->right;  }  rEnd = cur1;    return \*this;  }  Number Number::operator-(const Number &number) const  {  Number result;  if (isEmpty() || number.isEmpty())  return result;  if (sign == number.sign)  {  Number n1 = \*this;  Number n2 = number;  n1.sign = true;  n2.sign = true;  if (n1 < n2)  {  result = n2 - n1;  result.sign = !sign;  }  else  {  if (n1 == n2)  return 0;    result.sign = sign;  NumberNode \*cur1 = rEnd;  NumberNode \*cur2 = number.rEnd;  int tenPowerL = (int)pow(10, NumberNode::l);  int carry = 0;  while (cur1 != nullptr)  {  int data1 = cur1->data;  int data2 = cur2 != nullptr ? cur2->data : 0;  int sum = data1 + carry - data2;  if (sum < 0)  {  sum += tenPowerL;  carry = -1;  }  else  carry = 0;    cur1 = cur1->left;  cur2 = cur2 != nullptr ? cur2->left : nullptr;  result.pushLeft(sum, NumberNode::l);  }  result.deleteLeftZeroes();  }  }  else  result = \*this + -number;    // NumberNode \*cur = result.lEnd;  // while (cur != nullptr)  // {  // result.size += cur->size;  // cur = cur->right;  // }  return result;  }  Number Number::operator+(const Number &number) const  {  Number result;  if (isEmpty() || number.isEmpty())  return result;  if (sign == number.sign)  {  result.sign = sign;  NumberNode \*cur1 = rEnd;  NumberNode \*cur2 = number.rEnd;  int tenPowerL = (int)pow(10, NumberNode::l);  int carry = 0;  while (cur1 != nullptr || cur2 != nullptr)  {  int sum = 0;  int data1 = cur1 != nullptr ? cur1->data : 0;  int data2 = cur2 != nullptr ? cur2->data : 0;  sum = data1 + data2 + carry;  carry = sum / tenPowerL;  sum %= tenPowerL;  int size1 = cur1 != nullptr ? cur1->size : 0;  int size2 = cur2 != nullptr ? cur2->size : 0;    int l = 0;  int sum1 = sum;  do  {  l++;  } while (sum1 /= 10);  l = std::max(std::max(size1, size2), l);  if (l > NumberNode::l)  l = NumberNode::l;  cur1 = cur1 != nullptr ? cur1->left : nullptr;  cur2 = cur2 != nullptr ? cur2->left : nullptr;  result.pushLeft(sum, l);  }  if (carry != 0)  result.pushLeft(carry, 1);  }  else  result = \*this - -number;    // NumberNode \*cur = result.lEnd;  // while (cur != nullptr)  // {  // result.size += cur->size;  // cur = cur->right;  // }  return result;  }  Number Number::operator-=(const Number &number)  {  \*this = \*this - number;  return \*this;  }  Number Number::operator+=(const Number &number)  {  \*this = \*this + number;  return \*this;  }  Number Number::operator\*(const Number &number) const  {  Number result = 0;  if (isEmpty() || number.isEmpty())  return result;  if (number == 0 || \*this == 0)  return 0;  if (number == 1)  return \*this;  if (number == -1)  return -\*this;  if (\*this == 1)  return number;  if (\*this == -1)  return -number;  if (\*this < 0 || number < 0)  {  result = (\*this < 0 ? -\*this : \*this) \* (number < 0 ? -number : number);  result.sign = this->sign == number.sign;  return result;  }  NumberNode \*cur = number.rEnd;  int i = 0;  while (cur != nullptr)  {  int data2 = cur->data;  int size2 = cur->size;  for (int j = size2 - 1; j >= 0; j--)  {  int digit = data2 / (int)pow(10, size2 - j - 1) % 10;  Number sum;  sum = multiplyByDigit(digit);  for (int k = 0; k < i; k++)  sum.pushRight(0);  i++;  sum.shiftRight();  result += sum;  }    cur = cur->left;  }  return result;  }  Number Number::operator/(const Number &number) const  {  assert(number != 0);  Number result = 0;  if (isEmpty() || number.isEmpty())  return result;  if (number == 1)  return \*this;  if (number == -1)  return -\*this;  if (\*this == number)  return 1;  if (\*this == -number)  return -1;  if (\*this < 0 || number < 0)  {  Number result = (\*this < 0 ? -\*this : \*this) / (number < 0 ? -number : number);  result.sign = sign == number.sign;  return result;  }  if (\*this < number)  return result;  NumberNode \*cur = lEnd;  Number curNumber;  int tenPowerL = (int)pow(10, NumberNode::l - 1);  int tenPowerCurSize = (int)pow(10, cur->size - 1);  while (cur != nullptr)  {  for (int i = cur->size; i > 0; i--)  {  int digit1 = cur->data / tenPowerCurSize % 10;    curNumber.pushRight(digit1);  curNumber.shiftRight();  curNumber.deleteLeftZeroes();  int digit2 = 0;  if (curNumber >= number)  for (; digit2 < 10; digit2++)  if (number.multiplyByDigit(digit2 + 1) > curNumber)  break;  result.pushRight(digit2);  curNumber -= number.multiplyByDigit(digit2);  tenPowerCurSize /= 10;  }  tenPowerCurSize = tenPowerL;  cur = cur->right;  }  result.shiftRight();  result.deleteLeftZeroes();  return result;  }  Number Number::operator\*=(const Number &number)  {  \*this = \*this \* number;  return \*this;  }  Number Number::operator/=(const Number &number)  {  \*this = \*this / number;  return \*this;  }  Number Number::operator%(const Number &number) const  {  assert(number != 0);  Number result = 0;  if (isEmpty() || number.isEmpty())  return result;  if (number == 1)  return 0;  if (number == -1)  return 0;  if (\*this == number)  return 0;  if (\*this == -number)  return 0;  if (\*this < 0 || number < 0)  {  Number result = (\*this < 0 ? -\*this : \*this) % (number < 0 ? -number : number);  result.sign = sign == number.sign;  return result;  }  if (\*this < number)  return \*this;  result = \*this - (\*this / number) \* number;  return result;  }  Number Number::operator%=(const Number &number)  {  \*this = \*this % number;  return \*this;  } | | | lib/Header.hpp  #pragma once  #include <iostream>  #include <fstream>  #include <cassert>  #include <cmath>  static std::fstream lout("files/log.txt", std::ios::out); | | --- | | lib/UserInterface/ActionIntoAColumn.cpp  #include "UserInterface.hpp"  void additionIntoAColumn(const Number &number1, const Number &number2, List &list)  {  Number result = number1 + number2;    int sizeNumber1 = number1.size + (number1 < 0);  int sizeNumber2 = number2.size + (number2 < 0);  int sizeResult = result.size + (result < 0);  int size = std::max(std::max(sizeNumber1, sizeNumber2), sizeResult);  std::cout << "/";  for (int i = 0; i < size; i++)  std::cout << "-";  std::cout << "\\" << std::endl;  std::cout << "|";  for (int i = 0; i < size - sizeNumber1; i++)  std::cout << " ";  std::cout << number1 << "|" << std::endl;  std::cout << "|";  for (int i = 0; i < size - sizeNumber2; i++)  std::cout << " ";  std::cout << number2 << "|" << std::endl;  std::cout << "|";  for (int i = 0; i < size; i++)  std::cout << "-";  std::cout << "|" << std::endl;  std::cout << "|";  for (int i = 0; i < size - sizeResult; i++)  std::cout << " ";  std::cout << result << "|" << std::endl;  std::cout << "\\";  for (int i = 0; i < size; i++)  std::cout << "-";  std::cout << "/" << std::endl;  saveNumberInList(list, result);    return;  }  void subtractionIntoAColumn(const Number &number1, const Number &number2, List &list)  {  Number result = number1 - number2;    int sizeNumber1 = number1.size + (number1 < 0);  int sizeNumber2 = number2.size + (number2 < 0);  int sizeResult = result.size + (result < 0);  int size = std::max(sizeNumber1, sizeNumber2);  if (size < sizeResult)  size = sizeResult;  else  size++;  std::cout << "/";  for (int i = 0; i < size; i++)  std::cout << "-";  std::cout << "\\" << std::endl;  std::cout << "|";  for (int i = 0; i < size - std::max(sizeNumber1, sizeNumber2) - 1; i++)  std::cout << " ";  std::cout << "\_";  for (int i = 0; i < sizeNumber2 - sizeNumber1; i++)  std::cout << " ";  std::cout << number1 << "|" << std::endl;  std::cout << "|";  for (int i = 0; i < size - sizeNumber2; i++)  std::cout << " ";  std::cout << number2 << "|" << std::endl;  std::cout << "|" << (size == std::max(sizeNumber1, sizeNumber2) + 1 && size != sizeResult ? " " : "");  for (int i = 0; i < size - (size == std::max(sizeNumber1, sizeNumber2) + 1 && size != sizeResult); i++)  std::cout << "-";  std::cout << "|" << std::endl;  std::cout << "|";  for (int i = 0; i < size - sizeResult; i++)  std::cout << " ";  std::cout << result << "|" << std::endl;  std::cout << "\\";  for (int i = 0; i < size; i++)  std::cout << "-";  std::cout << "/" << std::endl;  saveNumberInList(list, result);  return;  }  void multiplicationIntoAColumn(const Number &number1, const Number &number2, List &list)  {  Number result = number1 \* number2;    int sizeNumber1 = number1.size + (number1 < 0);  int sizeNumber2 = number2.size + (number2 < 0);  int sizeResult = result.size + (result < 0);  int size = std::max(sizeNumber1, sizeNumber2);  if (size < sizeResult)  size = sizeResult;  else  size++;  std::cout << "/";  for (int i = 0; i < size; i++)  std::cout << "-";  std::cout << "\\" << std::endl;  std::cout << "|";  for (int i = 0; i < size - sizeNumber1; i++)  std::cout << " ";  std::cout << number1 << "|" << std::endl;  std::cout << "|";  for (int i = 0; i < size - std::max(sizeNumber1, sizeNumber2) - 1; i++)  std::cout << " ";  std::cout << "\*";  for (int i = 0; i < sizeNumber1 - sizeNumber2; i++)  std::cout << " ";  std::cout << number2 << "|" << std::endl;  Number absNumber1 = number1 < 0 ? -number1 : number1;  Number absNumber2 = number2 < 0 ? -number2 : number2;  if (absNumber2 >= 10)  {  Number temp = absNumber1 \* (absNumber2 % 10);  std::cout << "|";  for (int i = 0; i < size - std::max(std::max(sizeNumber1, sizeNumber2), temp.size); i++)  std::cout << " ";  for (int i = 0; i < std::max(std::max(sizeNumber1, sizeNumber2), temp.size); i++)  std::cout << "-";  std::cout << "|" << std::endl;  temp = absNumber2;  temp = temp < 0 ? -temp : temp;  for (int i = 0; temp != 0; i++)  {  Number temp2 = absNumber1 \* (temp % 10);  std::cout << "|";  for (int j = 0; j < size - temp2.size - i; j++)  std::cout << " ";  std::cout << temp2;  for (int j = 0; j < i; j++)  std::cout << " ";  std::cout << "|" << std::endl;  temp /= 10;  }  }  std::cout << "|" << (size == std::max(sizeNumber1, sizeNumber2) + 1 && size != sizeResult ? " " : "");  for (int i = 0; i < size - (size == std::max(sizeNumber1, sizeNumber2) + 1 && size != sizeResult); i++)  std::cout << "-";  std::cout << "|" << std::endl;  std::cout << "|";  for (int i = 0; i < size - sizeResult; i++)  std::cout << " ";  std::cout << result << "|" << std::endl;  std::cout << "\\";  for (int i = 0; i < size; i++)  std::cout << "-";  std::cout << "/" << std::endl;  saveNumberInList(list, result);  return;  }  void divisionIntoAColumn(const Number &number1, const Number &number2, List &list)  {  if (number2 == 0)  {  std::cout << "Division by 0 is not possible." << std::endl;  return;  }  Number absNumber1 = number1 < 0 ? -number1 : number1;  Number absNumber2 = number2 < 0 ? -number2 : number2;  Number result = number1 / number2;  Number absResult = result < 0 ? -result : result;  int sizeAbsNumber1 = number1.size;  // int sizeAbsNumber2 = number2.size;  int sizeNumber1 = number1.size + (number1 < 0);  int sizeNumber2 = number2.size + (number2 < 0);  int sizeResult = result.size + (result < 0);  int leftSize = sizeNumber1 + 1;  int rightSize = std::max(sizeNumber2, sizeResult);  int size = leftSize + 1 + rightSize;    std::cout << "/";  for (int i = 0; i < size; i++)  std::cout << "-";  std::cout << "\\" << std::endl;  std::cout << "|\_" << number1 << "|" << number2;  for (int j = 0; j < rightSize - sizeNumber2; j++)  std::cout << " ";  std::cout << "|" << std::endl;  if (absNumber1 < absNumber2)  {  std::cout << "|";  for (int i = 0; i < leftSize - 1; i++)  std::cout << " ";  std::cout << "0|";  for (int i = 0; i < rightSize; i++)  std::cout << "-";  std::cout << "|" << std::endl;  std::cout << "| ";  for (int i = 0; i < leftSize - 1; i++)  std::cout << "-";  std::cout << "|" << result;  for (int i = 0; i < rightSize - sizeResult; i++)  std::cout << " ";  std::cout << "|" << std::endl;    std::cout << "| " << number1 << "|";  for (int i = 0; i < rightSize; i++)  std::cout << " ";  std::cout << "|" << std::endl;  }  else  {  Number tenPowerResult = 1;  for (int i = 0; i < result.size - 1; i++)  tenPowerResult \*= 10;  Number tenPowerNumber1 = 1;  for (int i = 0; i < number1.size - 1; i++)  tenPowerNumber1 \*= 10;  int rightShift = 1 + (number1 < 0);  Number temp = 0;  int k = 0;  int l = 0;  bool first = true;  for (int i = 0; i < result.size; i++)  {  for (; k < sizeAbsNumber1 && temp < absNumber2; k++)  {  Number d = (absNumber1 / tenPowerNumber1) % 10;  temp \*= 10;  temp += d;  tenPowerNumber1 /= 10;  if (!first)  {  std::cout << d;  l++;  }  }  int tempSize = temp.size;  Number digit = (absResult / tenPowerResult) % 10;  tenPowerResult /= 10;    if (digit == 0)  continue;  if (!first)  {  for (int j = 0; j < size - l; j++)  std::cout << " ";  l = 0;  std::cout << "|" << std::endl;  }    Number carry = digit \* absNumber2;  std::cout << "|";  for (int j = 0; j < rightShift + temp.size - carry.size; j++)  std::cout << " ";  std::cout << carry;  for (int j = 0; j < leftSize - rightShift - temp.size; j++)  std::cout << " ";  if (!first)  for (int j = 0; j < rightSize + 1; j++)  std::cout << " ";  else  {  std::cout << "|";  for (int j = 0; j < rightSize; j++)  std::cout << "-";  }  std::cout << "|" << std::endl;  std::cout << "|";  for (int j = 0; j < rightShift; j++)  std::cout << " ";  for (int j = 0; j < temp.size; j++)  std::cout << "-";  for (int j = 0; j < leftSize - rightShift - temp.size; j++)  std::cout << " ";  if (!first)  for (int j = 0; j < rightSize + 1; j++)  std::cout << " ";  else  {  std::cout << "|" << result;  for (int j = 0; j < rightSize - sizeResult; j++)  std::cout << " ";  }  std::cout << "|" << std::endl;    temp -= carry;    std::cout << "|";  for (int j = 0; j < rightShift + tempSize - temp.size - 1; j++)  std::cout << " ";  std::cout << (i + 1 == result.size ? (number1 < 0) == (number2 < 0) ? " " : "-" : "\_") << temp;  l += rightShift + tempSize;  rightShift += tempSize - temp.size + (temp == 0);  first = false;  }  if (!first)  {  for (int j = 0; j < size - l; j++)  std::cout << " ";  l = 0;  std::cout << "|" << std::endl;  }  }  std::cout << "\\";  for (int i = 0; i < size; i++)  std::cout << "-";  std::cout << "/" << std::endl;  saveNumberInList(list, result);  return;  }  void remainderIntoAColumn(const Number &number1, const Number &number2, List &list)  {  if (number2 == 0)  {  std::cout << "Division by 0 is not possible." << std::endl;  return;  }  Number absNumber1 = number1 < 0 ? -number1 : number1;  Number absNumber2 = number2 < 0 ? -number2 : number2;  Number result = number1 / number2;  Number realResult = number1 % number2;  Number absResult = result < 0 ? -result : result;  int sizeAbsNumber1 = number1.size;  // int sizeAbsNumber2 = number2.size;  int sizeNumber1 = number1.size + (number1 < 0);  int sizeNumber2 = number2.size + (number2 < 0);  int sizeResult = result.size + (result < 0);  int leftSize = sizeNumber1 + 1;  int rightSize = std::max(sizeNumber2, sizeResult);  int size = leftSize + 1 + rightSize;    std::cout << "/";  for (int i = 0; i < size; i++)  std::cout << "-";  std::cout << "\\" << std::endl;  std::cout << "|\_" << number1 << "|" << number2;  for (int j = 0; j < rightSize - sizeNumber2; j++)  std::cout << " ";  std::cout << "|" << std::endl;  if (absNumber1 < absNumber2)  {  std::cout << "|";  for (int i = 0; i < leftSize - 1; i++)  std::cout << " ";  std::cout << "0|";  for (int i = 0; i < rightSize; i++)  std::cout << "-";  std::cout << "|" << std::endl;  std::cout << "| ";  for (int i = 0; i < leftSize - 1; i++)  std::cout << "-";  std::cout << "|" << result;  for (int i = 0; i < rightSize - sizeResult; i++)  std::cout << " ";  std::cout << "|" << std::endl;    std::cout << "| " << number1 << "|";  for (int i = 0; i < rightSize; i++)  std::cout << " ";  std::cout << "|" << std::endl;  }  else  {  Number tenPowerResult = 1;  for (int i = 0; i < result.size - 1; i++)  tenPowerResult \*= 10;  Number tenPowerNumber1 = 1;  for (int i = 0; i < number1.size - 1; i++)  tenPowerNumber1 \*= 10;  int rightShift = 1 + (number1 < 0);  Number temp = 0;  int k = 0;  int l = 0;  bool first = true;  for (int i = 0; i < result.size; i++)  {  for (; k < sizeAbsNumber1 && temp < absNumber2; k++)  {  Number d = (absNumber1 / tenPowerNumber1) % 10;  temp \*= 10;  temp += d;  tenPowerNumber1 /= 10;  if (!first)  {  std::cout << d;  l++;  }  }  int tempSize = temp.size;  Number digit = (absResult / tenPowerResult) % 10;  tenPowerResult /= 10;    if (digit == 0)  continue;  if (!first)  {  for (int j = 0; j < size - l; j++)  std::cout << " ";  l = 0;  std::cout << "|" << std::endl;  }    Number carry = digit \* absNumber2;  std::cout << "|";  for (int j = 0; j < rightShift + temp.size - carry.size; j++)  std::cout << " ";  std::cout << carry;  for (int j = 0; j < leftSize - rightShift - temp.size; j++)  std::cout << " ";  if (!first)  for (int j = 0; j < rightSize + 1; j++)  std::cout << " ";  else  {  std::cout << "|";  for (int j = 0; j < rightSize; j++)  std::cout << "-";  }  std::cout << "|" << std::endl;  std::cout << "|";  for (int j = 0; j < rightShift; j++)  std::cout << " ";  for (int j = 0; j < temp.size; j++)  std::cout << "-";  for (int j = 0; j < leftSize - rightShift - temp.size; j++)  std::cout << " ";  if (!first)  for (int j = 0; j < rightSize + 1; j++)  std::cout << " ";  else  {  std::cout << "|" << result;  for (int j = 0; j < rightSize - sizeResult; j++)  std::cout << " ";  }  std::cout << "|" << std::endl;    temp -= carry;    std::cout << "|";  for (int j = 0; j < rightShift + tempSize - temp.size - 1; j++)  std::cout << " ";  std::cout << (i + 1 == result.size ? (number1 < 0) == (number2 < 0) ? " " : "-" : "\_") << temp;  l += rightShift + tempSize;  rightShift += tempSize - temp.size + (temp == 0);  first = false;  }  if (!first)  {  for (int j = 0; j < size - l; j++)  std::cout << " ";  l = 0;  std::cout << "|" << std::endl;  }  }  std::cout << "\\";  for (int i = 0; i < size; i++)  std::cout << "-";  std::cout << "/" << std::endl;  saveNumberInList(list, realResult);  return;  } | | lib/UserInterface/InputNumber.cpp  #include "UserInterface.hpp"  int inputNumber(int min, int max)  {    int number;  std::cin >> number;    while (std::cin.fail() || number < min || number > max)  {  std::cin.clear();  std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');    std::cout << "Invalid input. Try again:" << std::endl;  std::cin >> number;  }    return number;  }  Number inputNumber()  {  Number number;  std::cin >> number;  while (std::cin.fail())  {  std::cin.clear();  std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');    std::cout << "Invalid input. Try again:" << std::endl;  std::cin >> number;  }  return number;  }  Number inputNumber(List &list)  {  Number number = inputNumber();  saveNumberInList(list, number);  return number;  }  void inputNumber(List &list, Number &number1, Number &number2)  {  if (list.size == 0)  {  std::cout << "List is clear." << std::endl;  std::cout << "Enter the first number:" << std::endl;  number1 = inputNumber(list);  }  else  {  std::cout << "Size of the list: " << list.size << "." << std::endl;    std::cout << "Choose how you want to enter the first number:" << std::endl  << "\t1) Number from the list;" << std::endl  << "\t2) Number from the keyboard." << std::endl;  int choice = inputNumber(1, 2);    switch (choice)  {  case 1:  {  std::cout << "Enter the index of the first number:" << std::endl;  int index = inputNumber(1, list.size); index--;  number1 = list[index];  }  break;    case 2:  {  std::cout << "Enter the first number:" << std::endl;  number1 = inputNumber(list);  }  break;  }  }  if (list.size == 0)  {  std::cout << "List is clear." << std::endl;    std::cout << "Enter the second number:" << std::endl;  number2 = inputNumber(list);  }  else  {  std::cout << "Size of the list: " << list.size << "." << std::endl;  std::cout << "Choose how you want to enter the second number:" << std::endl  << "\t1) Number from the list;" << std::endl  << "\t2) Number from the keyboard." << std::endl;  int choice = inputNumber(1, 2);  switch (choice)  {  case 1:  {  std::cout << "Enter the index of the second number:" << std::endl;  int index = inputNumber(1, list.size); index--;  number2 = list[index];  }  break;    case 2:  {  std::cout << "Enter the second number:" << std::endl;  number2 = inputNumber(list);  }  break;  }  }  } | | lib/UserInterface/LoadListFromFile.cpp  #include "UserInterface.hpp"  void loadListFromFile(List &list)  {  std::fstream fin("files/list.txt", std::ios::in);  if (!fin.is\_open())  {  std::cout << "Error: file not found." << std::endl;  return;  }  list.clear();  Number number;  while (fin >> number)  list.pushBack(number);  std::cout << "List loaded." << std::endl;  fin.close();  } | | lib/UserInterface/PrintList.cpp  #include "UserInterface.hpp"  void printList(List &list)  {  if (list.isEmpty())  {  std::cout << "List is empty." << std::endl;  return;  }  std::cout << "Size of the list: " << list.size << "." << std::endl;  std::cout << "List:" << std::endl;  for (int i = 0; i < list.size; i++)  std::cout << list[i] << std::endl;  } | | lib/UserInterface/SaveListInFile.cpp  #include "UserInterface.hpp"  void saveListInFile(List &list)  {  std::fstream fout("files/list.txt", std::ios::out);  if (!fout.is\_open())  {  std::cout << "Error: file not found." << std::endl;  return;  }  for (int i = 0; i < list.size; i++)  fout << list[i] << std::endl;  std::cout << "List saved." << std::endl;  fout.close();  } | | lib/UserInterface/SaveNumberInList.cpp  #include "UserInterface.hpp"  void saveNumberInList(List &list, const Number &number)  {  std::cout << "Do you want to save this number in the list (y / n)?" << std::endl;  char answer;  std::cin >> answer;  while (answer != 'y' && answer != 'n')  {  std::cout << "Invalid input. Try again:" << std::endl;  std::cin >> answer;  }  if (answer == 'y')  {  list.pushBack(number);  lout << "The number [" << number.debugOutput() << "] was saved in the list." << std::endl;  }  } | | lib/UserInterface/UserInterface.hpp  #include "../List/List.hpp"  #include "../String/String.hpp"  int inputNumber(int, int);  Number inputNumber();  Number inputNumber(List &);  void inputNumber(List &, Number &, Number &);  void saveNumberInList(List &, const Number &);  void saveListInFile(List &);  void loadListFromFile(List &);  void printList(List &);  void additionIntoAColumn(const Number &, const Number &, List &);  void subtractionIntoAColumn(const Number &, const Number &, List &);  void multiplicationIntoAColumn(const Number &, const Number &, List &);  void divisionIntoAColumn(const Number &, const Number &, List &);  void remainderIntoAColumn(const Number &, const Number &, List &); | | lib/List/Node/ListNode.cpp  #include "ListNode.hpp"  ListNode::ListNode()  {}  ListNode::ListNode(const Number &number)  {  data = number;  }  ListNode::ListNode(const ListNode &node)  {  data = node.data;  if (node.next != nullptr)  next = new ListNode(\*node.next);  }  ListNode::~ListNode()  {  if (next != nullptr)  delete next;  } | | lib/List/Node/ListNode.hpp  #pragma once  #include "../../Number/Number.hpp"  class ListNode  {  public:  Number data;  ListNode \*next = nullptr;  public:  ListNode();  ListNode(const Number &);  ListNode(const ListNode &);  ~ListNode();  ListNode operator=(const ListNode &);  }; | | lib/List/Node/Operator.cpp | | #include "ListNode.hpp"  ListNode ListNode::operator=(const ListNode &node)  {  data = node.data;  if (node.next != nullptr)  next = new ListNode(\*node.next);  return \*this;  } | | lib/List/Clear.cpp  #include "List.hpp"  void List::clear()  {  delete begin;  begin = nullptr;  end = nullptr;  size = 0;  } | | lib/List/IsEmpty.cpp  #include "List.hpp"  bool List::isEmpty() const  {  return begin == nullptr;  } | | lib/List/List.cpp  #include "List.hpp"  List::List()  {  }  List::List(const List &list)  {  size = list.size;  if (list.begin != nullptr)  {  begin = new ListNode(\*list.begin);    ListNode \*cur = begin;  while (cur->next != nullptr)  cur = cur->next;  end = cur;  }  }  List::~List()  {  delete begin;  } | | lib/List/List.hpp  #pragma once  #include "Node/ListNode.hpp"  class List  {  private:  ListNode \*begin = nullptr;  ListNode \*end = nullptr;  public:  int size = 0;  public:  List();  List(const List &);  ~List();    void pushBack(const Number &);    void pushFront(const Number &);  void remove(int);  void clear();  bool isEmpty() const;    Number &operator[](int) const;  List operator=(const List &);  }; | | lib/List/Operator.cpp  #include "List.hpp"  Number &List::operator[](int n) const  {  assert((void("E2"), n < size));  assert((void("E2"), 0 <= n));    ListNode \*cur = begin;  int i = n;  while (i--)  cur = cur->next;    return cur->data;  }  List List::operator=(const List &text)  {  size = text.size;  if (text.begin != nullptr)  {  begin = new ListNode(\*text.begin);    ListNode \*cur = begin;  while (cur->next != nullptr)  cur = cur->next;  end = cur;  }  return \*this;  } | | lib/List/PushBack.cpp  #include "List.hpp"  void List::pushBack(const Number &number)  {  size++;  if (begin == nullptr)  {  begin = new ListNode(number);  end = begin;  return;  }    end->next = new ListNode(number);  end = end->next;  } | | lib/List/PushFront.cpp  #include "List.hpp"  void List::pushFront(const Number &number)  {  ListNode \*newNode = new ListNode();  newNode->data = number;  newNode->next = begin;  begin = newNode;  size++;  } | | lib/List/Remove.cpp  #include "List.hpp"  void List::remove(int i)  {  assert((void("E2"), i < size));  assert((void("E2"), 0 <= i));  size--;    if (i == 0)  {  ListNode \*tmp = begin;  begin = begin->next;  delete tmp;  return;  }    ListNode \*cur = begin;  while (--i)  cur = cur->next;    ListNode \*tmp = cur->next;  cur->next = cur->next->next;  delete tmp;  } | | lib/Number/PushRight.cpp  #include "Number.hpp"  void Number::pushRight(int number)  {  int l = 0;  int tmp = number;  do  l++;  while (tmp /= 10);  int tenPowerL = (int)pow(10, l);  do  {  number %= (tenPowerL);  tenPowerL /= 10;  int d = (number / tenPowerL) % 10;  size++;  if (isEmpty())  {  lEnd = new NumberNode(d);  rEnd = lEnd;  }  else  if (rEnd->size == NumberNode::l)  {  NumberNode \*tmp = new NumberNode(d);  tmp->left = rEnd;  rEnd->right = tmp;  rEnd = tmp;  }  else  {  rEnd->data \*= 10;  rEnd->data += d;  rEnd->size++;  }  } while (tenPowerL != 1);  } | | lib/Number/DebugOutput.cpp  #include "Number.hpp"  String Number::debugOutput() const  {  String result;    if (isEmpty())  return result;    if (!sign)  result.pushBack('-');  else  result.pushBack('+');  NumberNode \*cur = lEnd;  while (cur != nullptr)  {  result.pushBack('[');  for (int i = cur->size; i > 0; i--)  result.pushBack((cur->data % int(pow(10, i))) / int(pow(10, i - 1)) + '0');  result.pushBack(']');  result.pushBack(' ');  result.pushBack('-');  result.pushBack('>');  result.pushBack(' ');  cur = cur->right;  }  result.pushBack('N');  result.pushBack('U');  result.pushBack('L');  result.pushBack('L');  return result;  } | | lib/Number/IsEmpty.cpp  #include "Number.hpp"  bool Number::isEmpty() const  {  return lEnd == nullptr && rEnd == nullptr;  } | | lib/Number/Number.hpp  #pragma once  #include "Node/NumberNode.hpp"  #include "../String/String.hpp"  class Number  {  private:  NumberNode \*lEnd = nullptr;  NumberNode \*rEnd = nullptr;  bool sign = true;  public:  int size = 0;  private:  bool isEmpty() const;  void deleteLeftZeroes();  void pushLeft(int, int);  void pushRight(int);  void shiftRight();  Number multiplyByDigit(const int &) const;  public:  Number();  Number(const int &);  Number(const Number &);  ~Number();  String debugOutput() const;  void clear();  friend std::ostream &operator<<(std::ostream &, const Number &);  friend std::istream &operator>>(std::istream &, Number &);  bool operator==(const Number &) const;  bool operator!=(const Number &) const;  bool operator<(const Number &) const;  bool operator>(const Number &) const;  bool operator<=(const Number &) const;  bool operator>=(const Number &) const;  Number operator-() const;  Number operator=(const Number &);  Number operator-(const Number &) const;  Number operator+(const Number &) const;  Number operator-=(const Number &);  Number operator+=(const Number &);  Number operator\*(const Number &) const;  Number operator/(const Number &) const;  Number operator\*=(const Number &);  Number operator/=(const Number &);  Number operator%(const Number &) const;  Number operator%=(const Number &);  }; | | lib/Number/PushLeft.cpp  #include "Number.hpp"  void Number::pushLeft(int number, int l)  {  int tenPowerL = (int)pow(10, l);  do  {  number %= (tenPowerL);  tenPowerL /= 10;  int d = (number / tenPowerL) % 10;  size++;  if (isEmpty())  {  rEnd = new NumberNode(d);  lEnd = rEnd;  }  else  if (lEnd->size == NumberNode::l)  {  NumberNode \*tmp = new NumberNode(d);  tmp->right = lEnd;  lEnd->left = tmp;  lEnd = tmp;  }  else  {  lEnd->data \*= 10;  lEnd->data += d;  lEnd->size++;  }  } while (tenPowerL != 1);  } | | lib/Number/ShiftRight.cpp  #include "Number.hpp"  void Number::shiftRight()  {  if (isEmpty() || rEnd->size == NumberNode::l)  return;  NumberNode \*cur = rEnd;  while (cur->left != nullptr)  {  int shiftStep = std::min(NumberNode::l - cur->size, cur->left->size);  int tenPowerShiftStep = (int)pow(10, shiftStep);  int tenPowerSize = (int)pow(10, cur->size);  cur->data += (cur->left->data % tenPowerShiftStep) \* tenPowerSize;  cur->left->data /= tenPowerShiftStep;  cur->left->size -= shiftStep;  cur->size += shiftStep;  if (cur->left->size == 0)  {  NumberNode \*tmp = lEnd;  lEnd = lEnd->right;  lEnd->left = nullptr;  delete tmp;  return;  }  cur = cur->left;  }  } | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

# Подготовка контрольных примеров

Подготовим 8 контрольных примеров, которые проверят работоспособность программы (до ‘:’ идёт то, на что мы проверяем программу, после ‘:’ через запятую идёт порядок команд, которые мы вводим в терминал):

1. Проверка на запуск: 11, n
2. Проверка списка: 3, 1, 1234567890, 3, 1, -1234567890, 3, 2, 1, 3, 11, n
3. Проверка записи в файл и ввода из файла: 3, 1, 1234567890, 3, 1, -1234567890, 9, 2, 1, 2, 1, 3, 10, 3, 11, n
4. Проверка сложения двух чисел: 4, 1234567890, n, -1234567890, n, n, 4, 1234567890, n, 1234567890, n, n, 4, -1234567890, n, -1234567890, n, n, 11, n
5. Проверка вычитания двух чисел: 5, 1234567890, n, -123, n, n, 5, 1234567890, n, 123, n, n, 5, -1234567890, n, -123, n, n, 11, n
6. Проверка произведения двух чисел: 6, 1234567890, n, -1234567890, n, n, 6, 1234567890, n, 1234567890, n, n, 6, -1234567890, n, -1234567890, n, n, 11, n
7. Проверка деления двух чисел: 7, 1234567890, n, -123, n, n, 7, 1234567890, n, 123, n, n, 7, -1234567890, n, -123, n, n, 11, n
8. Проверка деления двух чисел: 8, 1234567890, n, -123, n, n, 8, 1234567890, n, 123, n, n, 8, -1234567890, n, -123, n, n, 11, n

# Анализ полученных результатов

1. Программа завершилась корректно.
2. Программа завершилась корректно.
3. Программа завершилась корректно. В результате работы в файл list.txt были записаны 2 числа: 1234567890, -1234567890.
4. Программа завершилась корректно. В результате работы в терминал были выведены следующие результаты сложения:

| /-----------\  | 1234567890|  |-1234567890|  |-----------|  | 0|  \-----------/ | /----------\  |1234567890|  |1234567890|  |----------|  |2469135780|  \----------/ | /-----------\  |-1234567890|  |-1234567890|  |-----------|  |-2469135780|  \-----------/ |
| --- | --- | --- |

1. Программа завершилась корректно. В результате работы в терминал были выведены следующие результаты вычитания:

| /------------\  |\_ 1234567890|  | -1234567890|  | -----------|  | 2469135780|  \------------/ | /-----------\  |\_1234567890|  | 1234567890|  | ----------|  | 0|  \-----------/ | /------------\  |\_-1234567890|  | -1234567890|  | -----------|  | 0|  \------------/ |
| --- | --- | --- |

1. Программа завершилась корректно. В результате работы в терминал были выведены следующие результаты умножения:

| /-------------\  | 1234567890|  | \* -123|  | ----------|  | 3703703670|  | 2469135780 |  | 1234567890 |  |-------------|  |-151851850470|  \-------------/ | /------------\  | 1234567890|  | \* 123|  | ----------|  | 3703703670|  | 2469135780 |  |1234567890 |  |------------|  |151851850470|  \------------/ | /------------\  | -1234567890|  |\* -123|  | -----------|  | 3703703670|  | 2469135780 |  |1234567890 |  |------------|  |151851850470|  \------------/ |
| --- | --- | --- |

1. Программа завершилась корректно. В результате работы в терминал были выведены следующие результаты деления:

| /---------------------\  |\_1234567890|-123 |  | 123 |---------|  | --- |-10037137|  | \_0456 |  | 369 |  | --- |  | \_877 |  | 861 |  | --- |  | \_168 |  | 123 |  | --- |  | \_459 |  | 369 |  | --- |  | \_900 |  | 861 |  | --- |  | -39 |  \---------------------/ | /--------------------\  |\_1234567890|123 |  | 123 |--------|  | --- |10037137|  | \_0456 |  | 369 |  | --- |  | \_877 |  | 861 |  | --- |  | \_168 |  | 123 |  | --- |  | \_459 |  | 369 |  | --- |  | \_900 |  | 861 |  | --- |  | 39 |  \--------------------/ | /---------------------\  |\_-1234567890|-123 |  | 123 |--------|  | --- |10037137|  | \_0456 |  | 369 |  | --- |  | \_877 |  | 861 |  | --- |  | \_168 |  | 123 |  | --- |  | \_459 |  | 369 |  | --- |  | \_900 |  | 861 |  | --- |  | 39 |  \---------------------/ |
| --- | --- | --- |

1. Программа завершилась корректно. В результате работы в терминал были выведены следующие результаты взятия остатка:

| /---------------------\  |\_1234567890|-123 |  | 123 |---------|  | --- |-10037137|  | \_0456 |  | 369 |  | --- |  | \_877 |  | 861 |  | --- |  | \_168 |  | 123 |  | --- |  | \_459 |  | 369 |  | --- |  | \_900 |  | 861 |  | --- |  | -39 |  \---------------------/ | /--------------------\  |\_1234567890|123 |  | 123 |--------|  | --- |10037137|  | \_0456 |  | 369 |  | --- |  | \_877 |  | 861 |  | --- |  | \_168 |  | 123 |  | --- |  | \_459 |  | 369 |  | --- |  | \_900 |  | 861 |  | --- |  | 39 |  \--------------------/ | /---------------------\  |\_-1234567890|-123 |  | 123 |--------|  | --- |10037137|  | \_0456 |  | 369 |  | --- |  | \_877 |  | 861 |  | --- |  | \_168 |  | 123 |  | --- |  | \_459 |  | 369 |  | --- |  | \_900 |  | 861 |  | --- |  | 39 |  \---------------------/ |
| --- | --- | --- |

# Выводы о проделанной работе

В данной курсовой работы были продемонстрированы навыки, приобретённые в ходе выполнения лабораторных работ, а именно: работа с файлами, создание классов, работа с односвязными и двусвязными списками, работа со строками.