МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ФАКУЛЬТЕТ АВТОМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ  
  
Кафедра вычислительной техники

Курсовая работа по дисциплине

«Программирование»

**Коллекция объектов произвольных типов в памяти на основе структуры двусвязного списка**

Вариант 6.1

Группа: АВТ-615

Студент: Стрепельцова В. Е.

Преподаватель: Васюткина И. А.

Новосибирск, 2017

Оглавление

[1. Развернутое описание задания 3](#_Toc502161012)

[2. Структурное описание разработки 4](#_Toc502161013)

[3. Функциональное описание 6](#_Toc502161014)

[4. Тестирование 7](#_Toc502161015)

[5. ПРИЛОЖЕНИЕ 11](#_Toc502161016)

# Развернутое описание задания

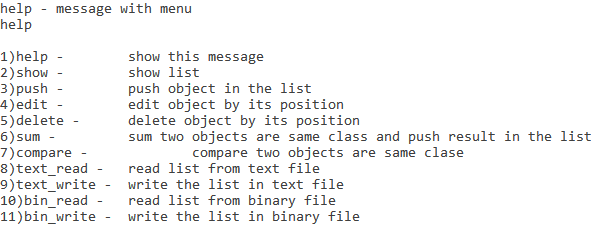
Разработать интерфейс для объединения в структуру данных множества объектов различных классов - абстрактный базовый класс объектов object, для которого предусмотреть виртуальные методы: загрузки объекта из текстовой строки, выгрузки объекта в текстовую строку в динамической памяти, добавления объекта в последовательный двоичный файл, чтения объекта из последовательного двоичного файла, возврата уникального идентификатора класса, возврата указателя на строку с именем класса, сравнения двух объектов, “сложения “(объединения) двух объектов, создание динамической копии объекта. Сделать классы хранимых объектов производными от абстрактного базового класса object (например, класс Float, String и класс из лабораторной работы).

Коллекция представляет собой структуру данных, хранящую указатели на объекты произвольного типа - объекты базового класса. Программа должна обеспечивать просмотр коллекции, добавление объекта выбранного типа, редактирование и удаление объекта, сложение и сравнение объектов при условии совпадения типов с помещением результата в ту же коллекцию, сохранение и загрузку объектов из текстового (или двоичного) файла.

Программа должна реализовывать указанные выше действия. Протестировать структуру данных на простом типе (например, Int или String). Программа тестирования должна содержать меню, обеспечивающее выбор операций.

Коллекция объектов реализуется на двусвязном списке.

Описание основных операций списка.

Меню с перечислением основных операций:

1) help – вывод сообщения с перечислением основных операций списка.

2)show, функция-метод класса List void show() – вывод списка на консоль.

3)push, функция-метод класса List void push() – помещение объекта в конец списка.

4)edit, функция-метод класса List void edition() – редактура объекта по заданному порядковому номеру элемента списка.

5)delete, функция-метод класса List List\* deleting() – удаление объекта по порядковому номеру элемента списка.

6)sum, функция-метод класса List void summation() – сложение двух объектов одного типа и запись результата в конец списка.

7)compare, функция-метод класса List void comparing() сравнение двух объектов одного типа.

8)text\_read, функция-метод класса List void text\_read() – чтение списка из текстового файла.

9)text\_write, функция-метод класса List void text\_write() – запись списка в текстовый файл.

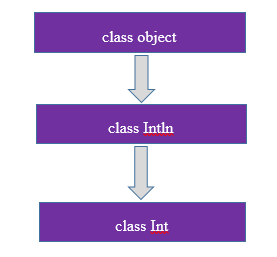
10)bin\_read, функция-метод класса List void bin\_read() – чтение списка из бинарного файла.

11)bin\_write, функция-метод класса List void bin\_write() - запись списка в бинарный файл

# Структурное описание разработки

Для реализации поставленной задачи были разработаны классы: абстрактный класс object, его наследники Int (класс целых чисел) и Intln (класс целых чисел произвольной точности) и класс List (двусвязный список). Указатель на объект абстрактного класса object является элементом класса List, что позволяет осуществить принцип полиморфизма при реализации двусвязного списка.

Схема наследования классов:



1)class object, абстрактный базовый класс.

функции-методы класса:

object() –конструктор по умолчания

virtual ~object() - деструктор

virtual void show() = 0 – функция вывода объекта на консоль

virtual void get() = 0 – считывание объекта с консоли

virtual int idetificate() – функция, возвращающая идентификационные номер класса

virtual char\* class\_name() = 0 – функция, возвращающая строку с именем класса

virtual int obj\_cmp(object\*, object\*) = 0 – сравнение двух объектов

virtual object\* sum(object\*, object\*) = 0 – сложение двух объектов

virtual void bin\_read(std::ifstream&) {} – чтение из бинарного файла

virtual void bin\_write(std::ofstream&) {} – запись в бинарный файл

virtual void text\_read(std::ifstream&) {} – чтение из текстового файла

virtual void text\_write(std::ofstream&) {} – запись в текстовый файл

2)class Intln, класс целых чисел произвольной точности, идентификационный номер 1.

Поля-данные класса:

int exponent – длина числа

char\* mantissa – массив цифр числа

Функции-методы класса:

Intln() – конструктор по умолчанию

Intln(int e, char\* m) – конструктор с параметрами

Intln(const Intln &n) – конструктор копирования

~Intln() – деструктор

void show() - вывод объекта классса

void get() – считывание объекта с консоли

int indificate() - получение уникального идентификатора класса

char\* class\_name() – функция, возвращающая строку с именем класса

int obj\_cmp(object\*, object\*) - сравнение двух объектов

object\* sum(object\*, object\*) - сложение двух объектов

void text\_read(std::ifstream&) - чтение из текстового файла

void text\_write(std::ofstream&) - запись в текстовый файл

void bin\_read(std::ifstream&) - чтение из бинарного файла

void bin\_write(std::ofstream&) - запись в бинарный файл

Перегрузки необъодимых операций:

Intln& operator=(const Intln&);

Intln operator+(const Intln&);

friend Intln operator-(const Intln&, const Intln&);

Intln& operator++();

Intln operator++(int);

friend std::ostream& operator<<(std::ostream&, Intln&);

friend std::istream& operator >> (std::istream&, Intln&);

friend std::ofstream& operator<<(std::ofstream&, Intln&);

friend std::ifstream& operator >> (std::ifstream&, Intln&);

3)class Int, класс целых чисел

Поля-данные класса:

int dat – целое число

Функции-методы идентичны методам класса Intln.

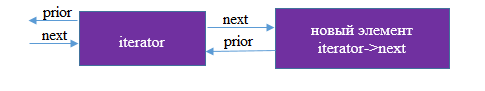
# Функциональное описание

1. Считывание объекта класса Intln

Каждый очередной символ считывается в переменную char ch функцией getchar() пока не прервется символьный поток. С каждым новым считанным символом объем памяти, выделенной для массива цифр числа, увеличивается функцией realloc(). После считывания символов полученная строка завершается битовым нулем и записывается в поле данных char\* mantissa. Количество полученных символов записывается в поле данных int exponent. Реализация в функции istream& operator>> (std::istream& is, Intln& numb).

1. Добавление элемента списка.

Запрашивается тип добавляемого элемента, в соответствии с ним поле данных элемента списка object\* obj инициализируется необходимым типом (Int или Intln), затем с помощью функции-метода get() класса object принимается значение объекта того класса, тип которого был задан. С помощью условия if (obj == nullptr) происходит проверка, первый ли элемент записывается в список. В случае, если список уже не пуст, последующий элемент записывается в конец список с направлением указателей на предыдущий и следующий элементы, что обеспечивает двунаправленную структуру списка. Реализация в функции void List::push().

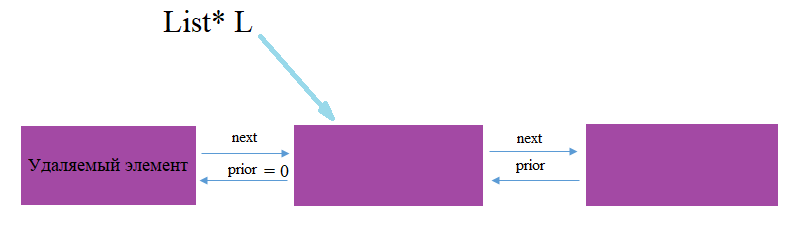
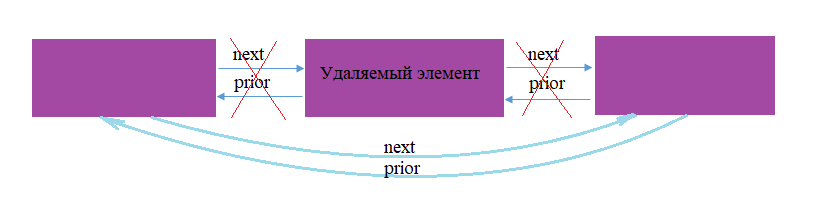


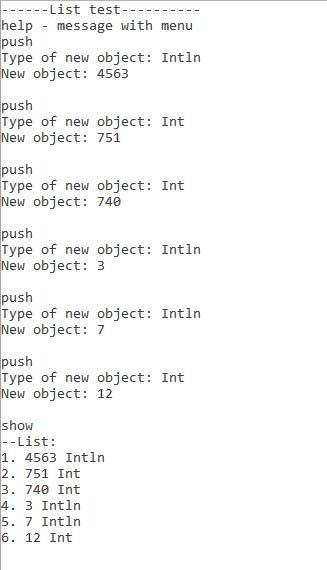
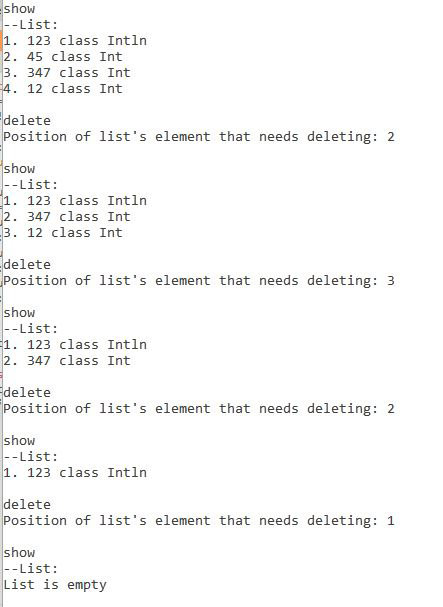
1. Суммирование элементов списка.

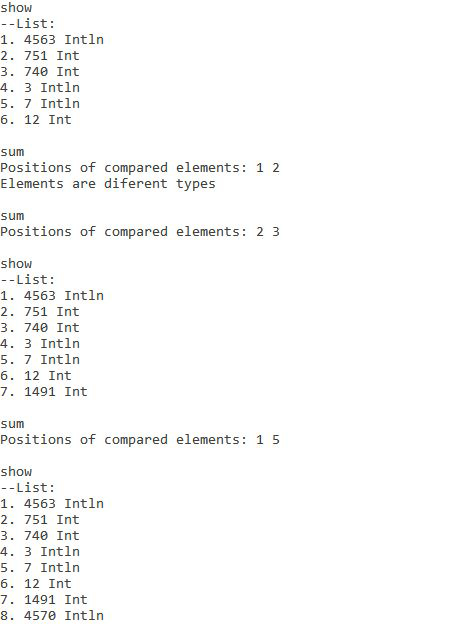
Запрашиваются порядковые номера суммируемых элементов, происходит проверка на равенство их типов. При равенстве объекты складываются с помощью метода-функции object\* sum(object\*, object\*) абстрактного класса object. Поле данных object\* obj элемента списка инициализируются в соответствии с типом объекта-результата суммы, получает результат суммы.

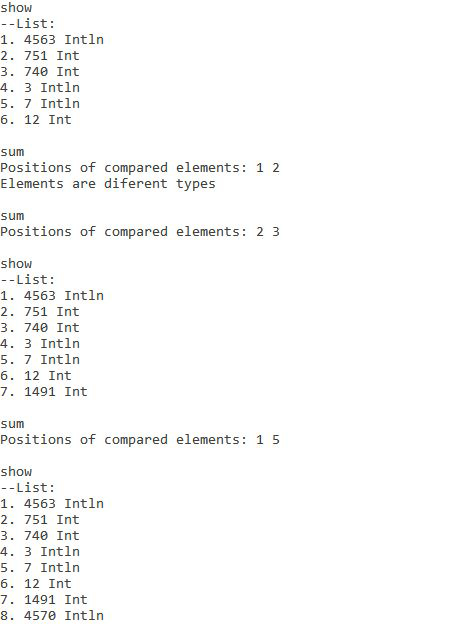
1. Удаление элемента

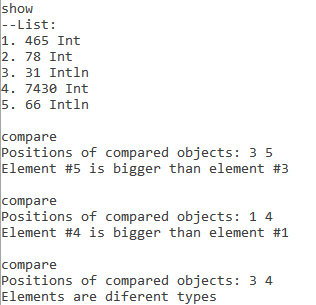
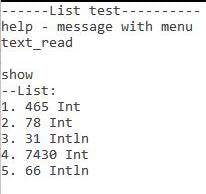
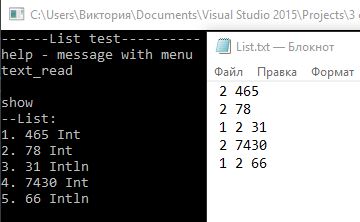
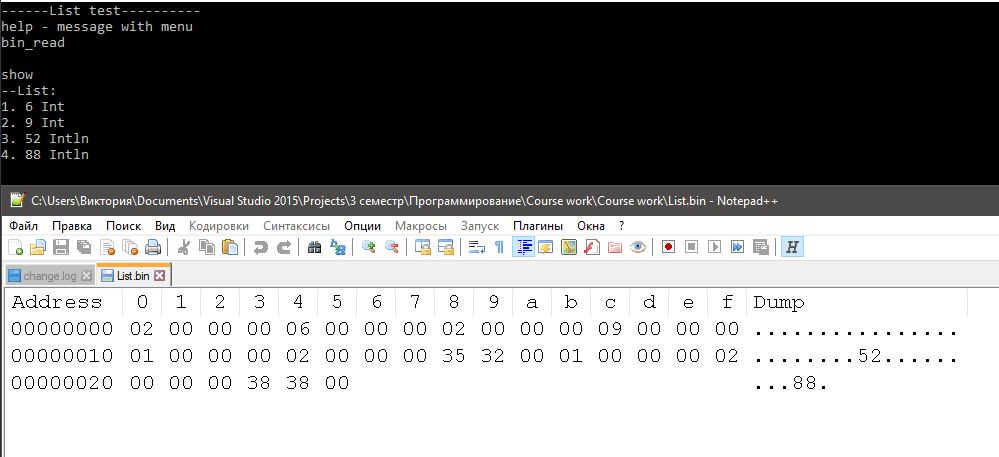
При удалении элемента по позиции выполняются 4 условия:

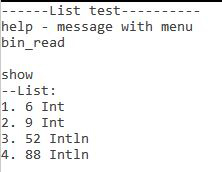
* удаляемый элемент (List\* iterator) – единственный в списке (iterator->prior == nullptr && iterator->next == nullptr). При этом данные, хранимые в этом элементе стираются с помощью деструктора своего класса, указатель зануляется.
* удаляемый элемент – первый в списке (iterator->prior == nullptr). В этом случае указатель на начало списка перенаправляется на второй элемент, первый удаляется.
* удаляемый элемент – последний в списке (iterator->next == nullptr). Данные, хранящиеся в объекте, стираются, указатель на него зануляется.
* удаляемый элемент находится между двумя элементами списка

1. Тестирование
2. Заполнение списка функцией push.
3. Удаление элементов
4. Суммирование элементов





1. Сравнение элементов списка
2. Ввод списка из текстового файла
3. Ввод списка из бинарного файла



# Вывод

# В ходе выполнения курсовой работы была спроектирована и реализована структура данных коллекция объектов, основой которой послужил двунаправленный список. Элементы списка хранят указатели на абстрактный класс, что позволяет осуществить одну из основных парадигм объектно-ориентированного программирования – полиморфизм. Благодаря этому структура применима для хранения объектов сразу нескольких типов данных.

# ПРИЛОЖЕНИЕ

1. Abstract class.h

#include<fstream>

#include<iostream>

#include<cmath>

class object

{

public:

object() {};

virtual ~object() {};

virtual void show() = 0;

virtual void get() = 0;

virtual int idetificate() { return(0); };

virtual char\* class\_name() = 0;

virtual int obj\_cmp(object\*, object\*) = 0;

virtual object\* sum(object\*, object\*) = 0;

virtual void bin\_read(std::ifstream&) {};

virtual void bin\_write(std::ofstream&) {};

virtual void text\_read(std::ifstream&) {};

virtual void text\_write(std::ofstream&) {};

};

1. Intln.h

#pragma once

#include"Abstract Class.h"

class Intln : public object

{

protected:

int exponent;

char\* mantissa;

public:

Intln();

Intln(int e, char\* m);

Intln(const Intln &n);

~Intln();

void show(); //вывод объекта классса, виртуальная ф

void get();

int indificate(); //получение уникального идентификатора класса

char\* class\_name();//строка с именем класса

int obj\_cmp(object\*, object\*);//сравнение двух объектов

object\* sum(object\*, object\*);//сложение двух объектов

void text\_read(std::ifstream&);///чтение из текстового файла

void text\_write(std::ofstream&);//запись в текстовый файл

void bin\_read(std::ifstream&);//чтение из бинарного файла

void bin\_write(std::ofstream&);//запись в бинарный файл

Intln& operator=(const Intln&);

Intln operator+(const Intln&);

friend Intln operator-(const Intln&, const Intln&);

Intln& operator++();

Intln operator++(int);

friend std::ostream& operator<<(std::ostream&, Intln&);

friend std::istream& operator >> (std::istream&, Intln&);

friend std::ofstream& operator<<(std::ofstream&, Intln&);

friend std::ifstream& operator >> (std::ifstream&, Intln&);

};

Intln::Intln()

{

exponent = 1;

mantissa = NULL;

}

Intln::Intln(int e, char\* m)

{

exponent = e;

mantissa = new char[e + 1];

for (int i = 0; i < exponent; i++)

{

mantissa[i] = m[i];

}

mantissa[exponent] = '\0';

}

Intln::Intln(const Intln &n)

{

exponent = n.exponent;

mantissa = new char[exponent + 1];

for (int i = 0; i < exponent + 1; i++)

{

mantissa[i] = n.mantissa[i];

}

}

Intln::~Intln()

{

if (mantissa)

delete[] mantissa;

}

void Intln::show()

{

for (int i = 0; i < exponent; i++)

std::cout << mantissa[i];

std::cout << " ";

std::cout << class\_name() << " " << std::endl;

}

void Intln::get()

{

std::cin >> \*this;

}

int Intln::indificate()

{

return 1;

}

inline char\* Intln::class\_name()

{

char\* name = new char[6];

name = "Intln";

return name;

}

int Intln::obj\_cmp(object\* n1, object\* n2)

{

//object::obj\_cmp(object\* n1, object\* n2);

return strcmp(((Intln\*)n1)->mantissa, ((Intln\*)n2)->mantissa);

}

object\* Intln::sum(object\* n1, object\* n2)

{

//object::sum(object\*, object\*);

object\* res = new Intln;

\*((Intln\*)res) = \*((Intln\*)n1) + \*((Intln\*)n2);

return res;

}

inline void Intln::text\_read(std::ifstream& text\_f)

{

text\_f >> \*this;

}

inline void Intln::text\_write(std::ofstream& text\_f)

{

int identificator = indificate();

text\_f << " " << identificator << " ";

text\_f << \*this;

}

void Intln::bin\_read(std::ifstream& bin\_f)

{

if (mantissa)

delete[] mantissa;

bin\_f.read((char\*)&exponent, sizeof(int));

mantissa = new char[exponent + 1];

bin\_f.read(mantissa, exponent + 1);

}

void Intln::bin\_write(std::ofstream& bin\_f)

{

int identificator = indificate();

bin\_f.write((char\*)&identificator, sizeof(int));

bin\_f.write((char\*)&exponent, sizeof(int));

bin\_f.write(mantissa, exponent + 1);

bin\_f.flush();

}

Intln& Intln::operator=(const Intln& n)

{

exponent = n.exponent;

if (mantissa) {

delete[] mantissa;

}

mantissa = new char(exponent + 1);

for (int i = 0; i < exponent + 1; i++)

{

mantissa[i] = n.mantissa[i];

}

return \*this;

}

Intln Intln::operator+(const Intln& right)

{

Intln result;

bool reg = false;

int position;

if (this->exponent >= right.exponent)

{

position = this->exponent - 1;

result.exponent = this->exponent;

}

else

{

position = right.exponent - 1;

result.exponent = right.exponent;

}

char\* ptr;

if ((ptr = (char\*)realloc(result.mantissa, position + 2)) != NULL)

result.mantissa = ptr;

int i, j;

for (i = this->exponent - 1, j = right.exponent - 1; position >= 0; i--, j--, position--)

{

if (i < 0)

while (j >= 0)

{

result.mantissa[position] = right.mantissa[j] + reg;

reg = false;

if (result.mantissa[position] > '9')

{

reg = true;

result.mantissa[position] -= 10;

}

j--;

position--;

}

if (j < 0)

while (i >= 0)

{

result.mantissa[position] = this->mantissa[i] + reg;

reg = false;

if (result.mantissa[position] > '9')

{

reg = true;

result.mantissa[position] -= 10;

}

i--;

position--;

}

if (i >= 0 && j >= 0)

{

result.mantissa[position] = this->mantissa[i] + right.mantissa[j] - '0' + reg;

reg = false;

if (result.mantissa[position] > '9')

{

reg = true;

result.mantissa[position] -= 10;

}

}

}

if (reg)

{

if ((ptr = (char\*)realloc(result.mantissa, result.exponent \* 2 + 1)) != NULL)

result.mantissa = ptr;

for (int k = result.exponent - 1; k >= 0; k--)

result.mantissa[k + 1] = result.mantissa[k];

result.mantissa[0] = '1';

result.exponent++;

}

result.mantissa[result.exponent] = '\0';

return result;

}

Intln operator-(const Intln& left, const Intln& right)

{

Intln result;

bool reg = false;

int position;

if (left.exponent >= right.exponent)

{

position = left.exponent - 1;

result.exponent = left.exponent;

}

else

{

position = right.exponent - 1;

result.exponent = right.exponent;

}

char\* ptr;

if ((ptr = (char\*)realloc(result.mantissa, position + 2)) != NULL)

result.mantissa = ptr;

int i, j;

for (i = left.exponent - 1, j = right.exponent - 1; position >= 0; i--, j--, position--)

{

if (i < 0)

while (j >= 0)

{

result.mantissa[position] = right.mantissa[j] - reg;

reg = false;

if (result.mantissa[position] < '0')

{

reg = true;

result.mantissa[position] += 10;

}

j--;

position--;

}

if (j < 0)

while (i >= 0)

{

result.mantissa[position] = left.mantissa[i] - reg;

reg = false;

if (result.mantissa[position] < '0')

{

reg = true;

result.mantissa[position] += 10;

}

i--;

position--;

}

if (i >= 0 && j >= 0)

{

result.mantissa[position] = left.mantissa[i] - right.mantissa[j] + '0' - reg;

reg = false;

if (result.mantissa[position] < '0')

{

reg = true;

result.mantissa[position] += 10;

}

}

}

int nnull = 0;

int k = 0;

while (result.mantissa[k] == '0')

{

nnull++;

k++;

}

for (k = 0; k < result.exponent - nnull; k++)

{

result.mantissa[k] = result.mantissa[k + nnull];

}

result.exponent -= nnull;

result.mantissa[result.exponent] = '\0';

return result;

}

Intln& Intln::operator++()

{

int i;

bool reg = true;

for (i = exponent - 1; i >= 0; i--)

{

mantissa[i] += reg;

reg = false;

if (mantissa[i] > '9')

{

reg = true;

mantissa[i] -= 10;

}

}

if (reg)

{

char\* ptr;

if ((ptr = (char\*)realloc(mantissa, exponent \* 2 + 1)) != NULL)

mantissa = ptr;

for (i = exponent - 1; i >= 0; i--)

mantissa[i + 1] = mantissa[i];

mantissa[0] = '1';

exponent++;

}

mantissa[exponent] = '\0';

return \*this;

}

Intln Intln::operator++(int)

{

Intln temp = \*this;

++\*this;

return temp;

}

std::ostream& operator<<(std::ostream& os, Intln& numb)

{

for (int i = 0; i < numb.exponent; i++)

os << numb.mantissa[i];

return os;

}

std::istream& operator>> (std::istream& is, Intln& numb)

{

char ch;

int size = 2;

char\* num = (char\*)calloc(size, sizeof(char));

int index = 0;

ch = getchar();

while (ch = getchar())

{

if (ch == '\n')

break;

num[index] = ch;

index++;

if (index >= size)

{

char\* ptr;

if ((ptr = (char\*)realloc(num, (size \* 2) + 1)) != NULL)

{

size += 1;

num = ptr;

}

}

}

num[index] = '\0';

numb.exponent = strlen(num);

numb.mantissa = new char[numb.exponent + 1];

for (int i = 0; i <= numb.exponent; i++)

{

numb.mantissa[i] = num[i];

}

return is;

}

std::ofstream& operator<<(std::ofstream& os, Intln& numb)

{

os << numb.exponent << " " << numb.mantissa;

return os;

}

std::ifstream& operator >> (std::ifstream& is, Intln& numb)

{

if (numb.mantissa)

delete[] numb.mantissa;

is.get();

numb.exponent = is.get() - '0';

is.get();

numb.mantissa = new char[numb.exponent + 1];

int i;

for (i = 0; i < numb.exponent + 1; i++)

numb.mantissa[i] = is.get();

numb.mantissa[i - 1] = '\0';

return is;

}

1. Int.h

#pragma once

#include"Intln.h"

class Int : public object

{

private:

int dat;

public:

Int();

Int(int);

//Int(const Int &n);

~Int();

void show(); //вывод объекта классса, виртуальная ф

void get();

int indificate(); //получение уникального идентификатора класса

char\* class\_name();//строка с именем класса

int obj\_cmp(object\*, object\*);//сравнение двух объектов

object\* sum(object\*, object\*);//сложение двух объектов

void text\_read(std::ifstream&);///чтение из текстового файла

void text\_write(std::ofstream&);//запись в текстовый файл

void bin\_read(std::ifstream&);//чтение из бинарного файла

void bin\_write(std::ofstream&);//запись в бинарный файл

Int& operator=(const Int& n);

Int operator+(const Int& right);

friend Int operator-(const Int& left, const Int& right);

Int& operator++();

Int operator++(int);

friend std::ostream& operator<<(std::ostream& os, Int& numb);

friend std::istream& operator >> (std::istream& is, Int& numb);

friend std::ofstream& operator<<(std::ofstream& os, Int& numb);

friend std::ifstream& operator >> (std::ifstream& is, Int& numb);

};

Int::Int()

{

dat = 0;

}

Int::Int(int n)

{

dat = n;

}

Int::~Int()

{

dat = 0;

}

void Int::show()

{

std::cout << dat << " ";

std::cout << class\_name() << " " << std::endl;

}

void Int::get()

{

std::cin >> \*this;

}

int Int::indificate()

{

return 2;

}

char\* Int::class\_name()

{

char\* name = new char[4];

name = "Int";

return name;

}

void Int::text\_read(std::ifstream& text\_f)

{

text\_f >> \*this;

}

void Int::text\_write(std::ofstream& text\_f)

{

int identificator = indificate();

text\_f << " " << identificator << " ";

text\_f << \*this;

}

void Int::bin\_read(std::ifstream& bin\_f)

{

bin\_f.read((char\*)&dat, sizeof(int));

}

void Int::bin\_write(std::ofstream& bin\_f)

{

int identificator = indificate();

bin\_f.write((char\*)&identificator, sizeof(int));

bin\_f.write((char\*)&dat, sizeof(int));

}

Int& Int::operator=(const Int& n)

{

dat = n.dat;

return \*this;

}

Int Int::operator+(const Int & right)

{

Int res;

res.dat = this->dat + right.dat;

return res;

}

Int& Int::operator++()

{

this->dat = this->dat + 1;

return \*this;

}

Int Int::operator++(int)

{

Int temp = \*this;

++\*this;

return temp;

}

int Int::obj\_cmp(object\* n1, object\* n2)

{

if (((Int\*)n1)->dat > ((Int\*)n2)->dat) return 1;

if (((Int\*)n1)->dat < ((Int\*)n2)->dat) return -1;

else return 0;

}

object\* Int::sum(object\* n1, object\* n2)

{

object\* res = new Int(0);

((Int\*)res)->dat += ((Int\*)n1)->dat;

((Int\*)res)->dat += ((Int\*)n2)->dat;

dat = ((Int\*)res)->dat;

return res;

}

Int operator-(const Int& left, const Int& right)

{

Int res;

res.dat = left.dat - right.dat;

return res;

}

std::ostream& operator<<(std::ostream& os, Int& numb)

{

os << numb.dat;

return os;

}

std::istream& operator >> (std::istream& is, Int& numb)

{

is >> numb.dat;

return is;

}

std::ofstream& operator<<(std::ofstream& os, Int& numb)

{

os << numb.dat;

return os;

}

std::ifstream& operator >> (std::ifstream& is, Int& numb)

{

is >> numb.dat;

return is;

}

1. List.h

#include"Int.h"

class List

{

private:

object\* obj;

List\* next;

List\* prior;

public:

List();

~List();

void show();

void push();

void edition();

List\* deleting();

void summation();

void comparing();

void text\_read();

void text\_write();

void bin\_read();

void bin\_write();

};

List::List()

{

obj = nullptr;

next = nullptr;

prior = nullptr;

}

List::~List()

{

List\* iterator;

iterator = this;

while (iterator->next != nullptr)

{

delete obj;

iterator->prior->next = nullptr;

iterator->prior = nullptr;

iterator = iterator->next;

}

}

void List::show()

{

std::cout << "--List:" << std::endl;

if (this->obj == nullptr)

{

std::cout << "List is empty" << std::endl;

return;

}

List\* iterator;

iterator = this;

int i = 1;

while (iterator != nullptr)

{

std::cout << i++ << ". ";

iterator->obj->show();

iterator = iterator->next;

}

}

void List::push()

{

char type[10];

std::cout << "Type of new object: ";

std::cin >> type;

if (obj == nullptr)

{

if (strcmp("Intln", type) == 0)

obj = new Intln;

if (strcmp("Int", type) == 0)

obj = new Int;

if (obj == nullptr)

{

std::cout << "Wrong type" << std::endl;

return;

}

std::cout << "New object: ";

obj->get();

return;

}

List\* iterator = this;

while (iterator->next!= nullptr)

iterator = iterator->next;

iterator->next = new List;

iterator->next->prior = iterator;

iterator = iterator->next;

if (strcmp("Intln", type) == 0)

iterator->obj = new Intln;

if (strcmp("Int", type) == 0)

iterator->obj = new Int;

if (iterator->obj == nullptr)

{

std::cout << "Wrong type" << std::endl;

return;

}

std::cout << "New object: ";

iterator->obj->get();

}

void List::edition()

{

int position;

std::cout << "Position of list's element that needs edition: ";

std::cin >> position;

List\* iterator;

iterator = this;

for (int i = 1; i < position; i++, iterator = iterator->next);

if (iterator == nullptr)

{

std::cout << "Off-list position" << std::endl;

return;

}

std::cout << "Old object: ";

iterator->obj->show();

char type[10];

std::cout << "Type of new object: ";

std::cin >> type;

if (strcmp(iterator->obj->class\_name(), type) != 0)

{

std::cout << "Different types" << std::endl;

return;

}

if (strcmp("Intln", type) == 0)

iterator->obj = new Intln;

if (strcmp("Int", type) == 0)

iterator->obj = new Int;

std::cout << "New object: ";

iterator->obj->get();

}

List\* List::deleting()

{

int position;

std::cout << "Position of list's element that needs deleting: ";

std::cin >> position;

List\* iterator;

iterator = this;

int i = 1;

for (; i < position; ++i, iterator = iterator->next);

if (iterator == nullptr)

{

std::cout << "Off-list position" << std::endl;

return this;

}

if (iterator->prior == nullptr && iterator->next == nullptr)

{

delete iterator->obj;

iterator->obj = nullptr;

iterator = nullptr;

return this;

}

if (iterator->prior == nullptr)

{

delete this->obj;

this->obj = nullptr;

return this->next;

}

if (iterator->next == nullptr)

{

iterator = iterator->prior;

delete iterator->next->obj;

iterator->next->obj = nullptr;

iterator->next = nullptr;

return this;

}

iterator->prior->next = iterator->next;

iterator->next->prior = iterator->prior;

delete iterator->obj;

iterator->obj = nullptr;

iterator = nullptr;

return this;

}

void List::summation()

{

int pos1, pos2;

std::cout << "Positions of compared elements: ";

std::cin >> pos1;

std::cin >> pos2;

List\* iterator1;

iterator1 = this;

for (int i = 1; i < pos1; i++, iterator1 = iterator1->next);

List\* iterator2;

iterator2 = this;

for (int i = 1; i < pos2; i++, iterator2 = iterator2->next);

int eq = strcmp(iterator1->obj->class\_name(), iterator2->obj->class\_name());

if (eq != 0)

{

std::cout << "Elements are diferent types" <<std::endl;

return;

}

List\* iterator = this;

while (iterator->next != nullptr)

iterator = iterator->next;

iterator->next = new List;

iterator->next->prior = iterator;

iterator = iterator->next;

if (strcmp("Intln", iterator1->obj->class\_name()) == 0)

iterator->obj = new Intln;

if (strcmp("Int", iterator1->obj->class\_name()) == 0)

iterator->obj = new Int;

iterator->obj = iterator->obj->sum(iterator1->obj, iterator2->obj);

}

void List::comparing()

{

int pos1, pos2;

std::cout << "Positions of compared objects: ";

std::cin >> pos1;

std::cin >> pos2;

List\* iterator1;

iterator1 = this;

for (int i = 1; i < pos1; i++, iterator1 = iterator1->next);

List\* iterator2;

iterator2 = this;

for (int i = 1; i < pos2; i++, iterator2 = iterator2->next);

int eq = strcmp(iterator1->obj->class\_name(), iterator2->obj->class\_name());

if (eq != 0)

{

std::cout << "Elements are diferent types" <<std::endl;

return;

}

int result;

result = iterator1->obj->obj\_cmp(iterator1->obj, iterator2->obj);

if (result > 0)

std::cout << "Element #" << pos1 << " is bigger than element #" << pos2 << std::endl;

if (result < 0)

std::cout << "Element #" << pos2 << " is bigger than element #" << pos1 << std::endl;

if (result == 0)

std::cout << "Element #" << pos2 << " and element #" << pos1 << " are equal" << std::endl;

}

void List::text\_read()

{

std::ifstream text\_f("List.txt");

List\* iterator = this;

while (!text\_f.eof())

{

int identificator;

text\_f >> identificator;

if (identificator == 0)

break;

if (obj == nullptr)

{

if (identificator == 1)

iterator->obj = new Intln;

if (identificator == 2)

iterator->obj = new Int;

iterator->obj->text\_read(text\_f);

identificator = 0;

continue;

}

iterator->next = new List;

iterator->next->prior = iterator;

iterator = iterator->next;

if (identificator == 1)

iterator->obj = new Intln;

if (identificator == 2)

iterator->obj = new Int;

iterator->obj->text\_read(text\_f);

identificator = 0;

}

text\_f.close();

}

void List::text\_write()

{

std::ofstream text\_f("List.txt");

List\* iterator = this;

while (iterator != nullptr)

{

iterator->obj->text\_write(text\_f);

text\_f << "\n";

iterator = iterator->next;

}

text\_f.close();

}

void List::bin\_read()

{

std::ifstream bin\_f("List.bin", std::ios\_base::binary);

List\* iterator = this;

bin\_f.seekg(0, std::ios::end);

std::streampos eof = bin\_f.tellg();

bin\_f.seekg(0, std::ios::beg);

while (bin\_f.tellg() != eof) {

int identificator;

bin\_f.read((char\*)&identificator, sizeof(int));

if (obj == nullptr)

{

if (identificator == 1)

iterator->obj = new Intln;

if (identificator == 2)

iterator->obj = new Int;

identificator = 0;

iterator->obj->bin\_read(bin\_f);

continue;

}

iterator->next = new List;

iterator->next->prior = iterator;

iterator = iterator->next;

if (identificator == 1)

iterator->obj = new Intln;

if (identificator == 2)

iterator->obj = new Int;

identificator = 0;

iterator->obj->bin\_read(bin\_f);

}

}

void List::bin\_write()

{

std::ofstream bin\_f("List.bin", std::ios\_base::binary);

List\* iterator = this;

while (iterator != nullptr)

{

iterator->obj->bin\_write(bin\_f);

iterator = iterator->next;

}

bin\_f.flush();

}

1. Main.cpp

#include "List.h"

#include<conio.h>

int main()

{

List\* L = new List;

char com[128];

std::cout << "------List test----------" << std::endl

<< "help - message with menu " << std::endl;

std::cin >> com;

while (strcmp("stop", com) != 0) {

if (strcmp("help", com) == 0)

{

std::cout << "" << std::endl

<< "1)help - show this message" << std::endl

<< "2)show - show list" << std::endl

<< "3)push - push object in the list" << std::endl

<< "4)edit - edit object by its position" << std::endl

<< "5)delete - delete object by its position" << std::endl

<< "6)sum - sum two objects are same class and push result in the list" << std::endl

<< "7)compare - compare two objects are same clase" << std::endl

<< "8)text\_read - read list from text file" << std::endl

<< "9)text\_write - write the list in text file" << std::endl

<< "10)bin\_read - read list from binary file" << std::endl

<< "11)bin\_write - write the list in binary file" << std::endl

<< "" << std::endl;

}

if (strcmp("show", com)==0)

{

L->show();

std::cout << std::endl;

}

if (strcmp("push", com) == 0)

{

L->push();

std::cout << std::endl;

}

if (strcmp("edit", com) == 0)

{

L->edition();

std::cout << std::endl;

}

if (strcmp("delete", com) == 0)

{

L = L->deleting();

std::cout << std::endl;

}

if (strcmp("sum", com) == 0)

{

L->summation();

std::cout << std::endl;

}

if (strcmp("compare", com) == 0)

{

L->comparing();

std::cout << std::endl;

}

if (strcmp("text\_read", com) == 0)

{

L->text\_read();

std::cout << std::endl;

}

if (strcmp("text\_write", com) == 0)

{

L->text\_write();

std::cout << std::endl;

}

if (strcmp("bin\_read", com) == 0)

{

L->bin\_read();

std::cout << std::endl;

}

if (strcmp("bin\_write", com) == 0)

{

L->bin\_write();

std::cout << std::endl;

}

std::cin >> com;

}

system("pause");

return 0;

}