МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

|  |  |
| --- | --- |
| Кафедра | вычислительной техники |

**Курсовой проект**

**По дисциплине**

**«Программирование»**

|  |  |
| --- | --- |
| Тема: | Коллекция объектов произвольных типов в памяти |

|  |  |
| --- | --- |
| Студент | Павлюк А.С. (вариант 6.5) |

|  |  |
| --- | --- |
| Группа: | АВТ-008 |

|  |  |
| --- | --- |
| Преподаватель: | Романенко Татьяна Александровна |

Новосибирск 2021 г.

Оглавление

[1. Развернутое описание задания 3](#_Toc91462526)

[2. Структурное описание разработки 4](#_Toc91462527)

[3. Функциональное описание 10](#_Toc91462528)

[4. Описание пользовательского интерфейса 21](#_Toc91462529)

[5. Описание работы программы на контрольных примерах 25](#_Toc91462530)

[6. Выводы 27](#_Toc91462531)

[7. Список литературы 28](#_Toc91462532)

[8. Приложение 28](#_Toc91462533)

# Развернутое описание задания

Необходимо 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 (класс Integer и класс из лабораторной работы).

Коллекцию реализовать в виде дерева, вершина которого содержит два указателя на поддеревья, счетчик количества вершин и указатель на объект.

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

# Структурное описание разработки

В данной работе мною были использованы стандартные типы данных и собственные структуры:

* int
* char
* bool
* string
* class Btree
* class Node
* class object
* class Integer
* class Date

**Двоичное дерево** — иерархическая структура данных, в которой каждый узел имеет не более двух потомков (детей). Как правило, первый называется родительским узлом, а дети называются левым и правым наследниками.

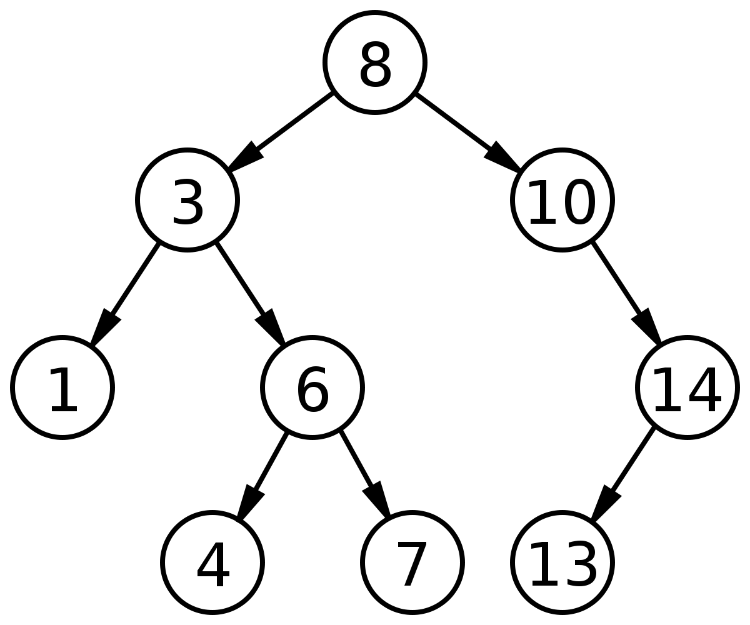


Рисунок 1. Пример двоичного дерева целых чисел

|  |
| --- |
| class Btree |
| Node \*root  int c; |
| Btree();  bool add(object\*);  Node\* insertNode(Node\*, Node\*, bool&);  void search(int);  object\* searchNode(Node\*, int);  void show();  void showNode(Node\*, int);  Node\* minValueObj(Node\*);  bool deleteObj(object\*);  Node\* deleteNodes(Node\*, Node\*, bool&);  void wrBin();  void writeInBinary(ofstream&, Node\*);  void rBin();  void readFromB(ifstream&);  ~Btree(); |

|  |
| --- |
| class Node |
| int key;  object\* o;  Node\* left;  Node\* right;  int count; |
| Node();  Node(object\*);  ~Node(); |

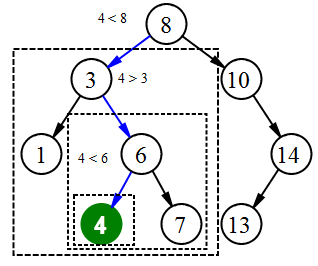
Для решения поставленной задачи была реализована структура данных Btree – двоичное дерево. Также был создан класс узла Node, содержащий необходимые параметры.

В классе Btree все методы можно разделить на пользовательские и структурные. Пользовательские – это те, что вызываются из меню, а структурные – доступные пользователю в неявном виде, в основном рекурсивные функции над узлами.

Основные алгоритмы:

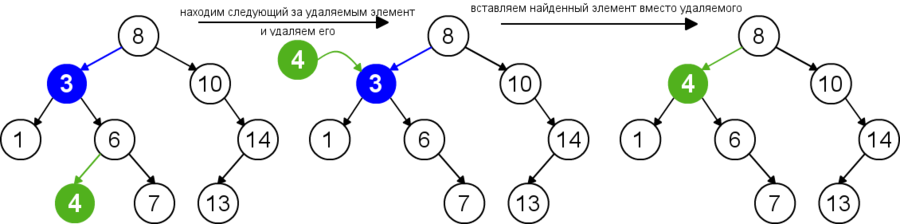
1. Вставка в двоичное дерево
2. Удаление из двоичного дерева
3. Поиск в дереве
4. Вывод структуры в консоль
5. Запись структуры в последовательный двоичный файл
6. Чтение структуры из последовательного двоичного файла
   1. Вставка

В работе реализована классическая вставка в бинарное дерево. Элементы сравниваются по ключам. Для класса Integer ключом является само значение value, а для класса Date было решено принять поле year (год) за ключ. Вставка осуществляется рекурсивно.



* 1. Удаление

Метод удаления рекурсивно обходит дерево, ищет подходящий по значению элемент и удаляет его без смещения, если этот элемент – лепесток, и со смещением, если элемент – вершина.

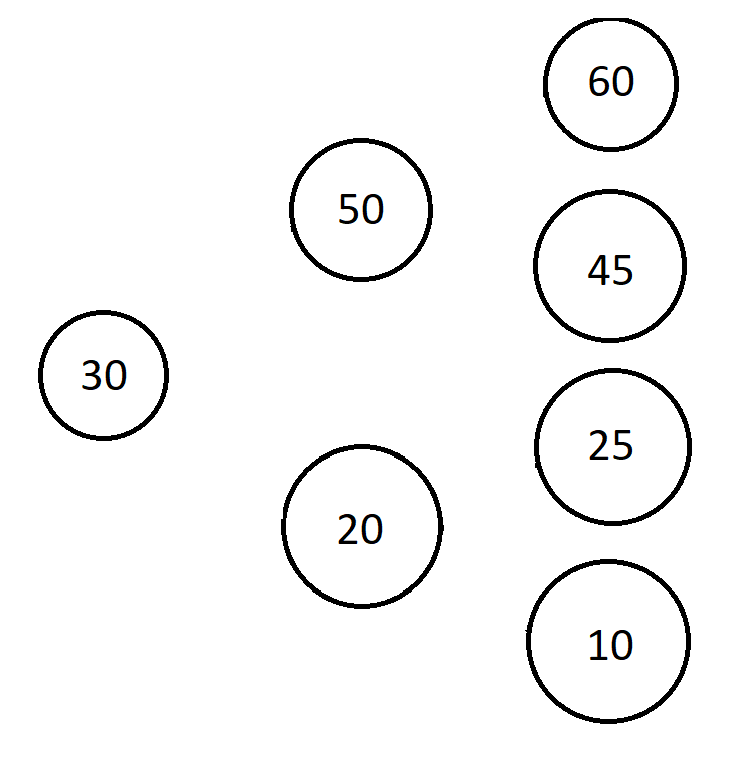


* 1. Поиск

Поиск элемента осуществляется по его целочисленному ключу. Классический рекурсивный обход. Для поиска, вставки и удаления была реализована функция equal(object\*, object\*), которая позволяет сравнивать не только однотипные объекты, но и элементы разного типа.

* 1. Консольный вывод

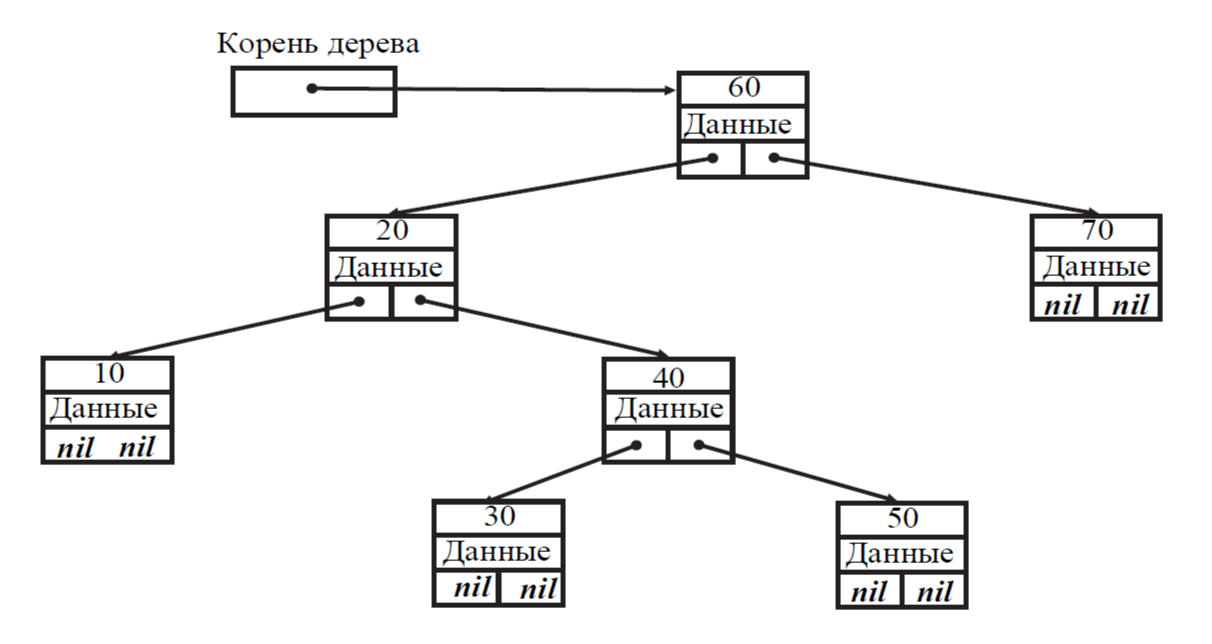
Данный метод распечатывает дерево по шаблону right-root-left. Структура печатается с поворотом на 90 против часовой стрелки. Сначала выводится ключ объекта, а затем в скобках полное строковое представление. Промежуток от левого края экрана составляет «уровень элемента» табуляций.

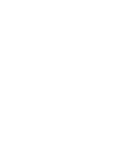


* 1. Запись в бинарный файл

Для записи в бинарный файл был написан следующий алгоритм: записать id объекта (1 – Integer, 2 – Date), чтобы удобнее считывать в дальнейшем. Затем записать сам объект в бинарный файл. Запись осуществляется рекурсивно по шаблону root-left-right. В случае отсутствия вершины, функция

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| id(root) | obj(root) | id(left) | obj(left) | id(right) | obj(right) | … |





* 1. Чтение из бинарного файла

Чтение из бинарного файла предполагает перезапись структуры. Вначале выделяется новая динамическая память в куче для дерева. Затем производится чтение уникального идентификатора объекта, затем по нему определяется тип данных. После производится чтение объекта из файла с помощью абстрактного метода и его запись в структуру при помощи метода 2.1. вставки в дерево.

**Абстрактный класс** в объектно-ориентированном программировании — базовый класс, который не предполагает создания экземпляров. Абстрактные классы реализуют на практике один из принципов ООП — полиморфизм. Абстрактный класс может содержать абстрактные методы и свойства. Абстрактный метод не реализуется для класса, в котором описан, однако должен быть реализован для его неабстрактных потомков. Абстрактные классы представляют собой наиболее общие абстракции, то есть имеющие наибольший объём и наименьшее содержание.

Был разработан абстрактный базовый класс объектов object с виртуальными методами по заданию. Также были созданы производные от него – Integer (класс целых чисел) и Date (класс даты и времени). Их поля и методы представлены ниже.

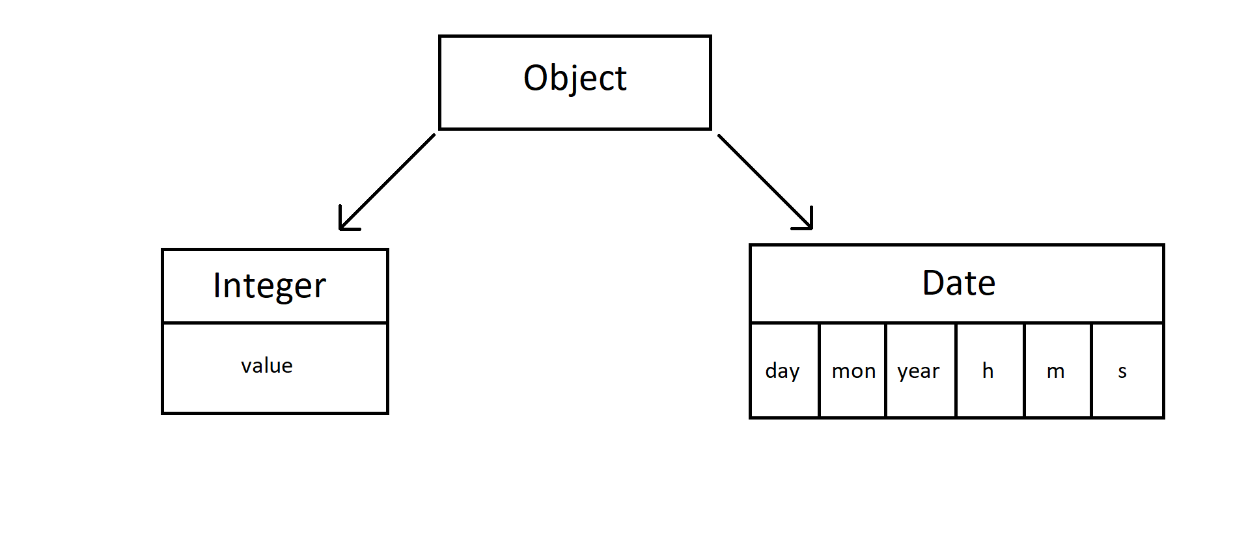


Рисунок 2. Наследование тестируемых классов

|  |
| --- |
| class Integer : public object |
| int value; |
| Integer();  Integer(int);  int getValue();  void setValue(int);  object\* loadFromString(char\*);  string uploadInString();  string getName();  int getId();  int equals(object\*);  object\* unionObj(object\*, object\*);  object\* makeCopy(object\*);  void writeInBinary(ofstream&);  void readFromBinary(ifstream&);  void getObject();  int getValueObj();  Integer& operator=(const Integer&);  Integer& operator+(const Integer&)  friend Integer& operator-(const Integer&, const Integer&);  Integer& operator++();  Integer& operator—();  friend ostream& operator<<(ostream&, Integer&);  friend istream& operator<<(istream&, Integer&); |

|  |
| --- |
| class object |
| object();  virtual object\* loadFromString(char\*);  virtual string uploadInString();  virtual string getName();  virtual int getId();  virtual int equals(object\*);  virtual object\* unionObj(object\*, object\*);  virtual object\* makeCopy(object\*);  virtual void writeInBinary(ofstream&);  virtual void readFromBinary(ifstream&);  virtual void getObject();  virtual int getValueObj();  ~object(); |

|  |
| --- |
| class Date |
| int day;  int mon;  int year;  int h;  int m;  int s; |
| Date();  Date(int, int, int);  Date(int, int, int, int, int, int);  int getDay();  int getonth();  int getYear();  int getHour();  int getMinute();  int getSecond();  int getValue();  void setYear(int);  void setMonth(int);  void setDay(int);  void setHour(int);  void setMinute(int);  void setSecond(int);  void incYear();  void incMonth();  void incDay();  void incHour();  void incMinutes();  void incSecond();  void decYear();  void decMonth();  void decDay();  void decHour();  void decMinute();  void decSecond();  object\* loadFromString(char\*);  string uploadInString();  string getName();  int getId();  int equals(object\*);  object\* unionObj(object\*, object\*);  object\* makeCopy(object\*);  void writeInBinary(ofstream&);  void readFromBinary(ifstream&);  void getObject();  int getValueObj();  Date& operator=(const Date& d);  friend ostream& operator<<(ostream&, Date&);  friend istream& operator>>(istream&, Date&);  ~Date(); |

# Функциональное описание

* 1. Класс узла Node:

Поля: int key; - ключ

object \*o; - объект

Node \*left; - указатель на левый узел

Node \*right; - указатель на правый узел

int count; - счетчик узлов

Конструкторы и деструктор:

Конструктор по-умолчанию.

Node() – инициализация object\* o, Node\* left, right в nullptr; int key, count в 0.

Конструктор с параметрами.

Node(object\* obj) – инициализация object\* o в \*obj, Node\* left, right в nullptr; int key в o->getValueObj(), count в 0.

Деструктор.

~Node()

* 1. Класс двоичного дерева Btree:

Поля: Node\* root – корень дерева.

int c – количество пройденных узлов для теста поиска

Методы:

Конструктор по-умолчанию инициализирует Node\* root в nullptr, c в 0.

Btree() {

root = nullptr;

c = 0;

}

Деструктор обнуляет поле c и отчищает указатель на корень дерева.

~Btree() {

delete root;

c = 0;

}

Вставка в двоичное дерево представляет собой булеву функцию, вызывающую рекурсивный метод добавления в дерево, начиная с его корня. При попытке вставки даты производится проверка на корректность ввода.

bool add(object \*obj) {

        Node \*n = new Node(obj);

        bool ins;

        if (!n) {

            return false;

        }

        if (((Date\*)obj)->getDay() == 1 && ((Date\*)obj)->getMonth() == 1 && ((Date\*)obj)->getYear() == 1 &&

        ((Date\*)obj)->getHour() == 0 && ((Date\*)obj)->getMinute() == 0 && ((Date\*)obj)->getSecond() == 0) return false;

        root = insertNode(root, n, ins);

        return ins;

    }

Рекурсивный метод вставки узла в дерево. Входные параметры – узел, куда вставлять, вставляемы узел и булевская переменная – признак успешной/неуспешной вставки.

Node\* insertNode(Node \*node, Node \*elem, bool &inserted) {

bool ins;

if (node == nullptr) {

inserted = true;

return elem;

}

if (equal(elem->o, node->o) < 0) {

node->left = insertNode(node->left, elem, ins);

}

else if (equal(elem->o, node->o) > 0) {

node->right = insertNode(node->right, elem, ins);

}

else {

inserted = false;

return node;

}

if (ins) node->count++;

inserted = ins;

return node;

}

Поиск элемента в дереве осуществляется по целочисленному ключу. На выход возвращается найденный объект или nullptr. Функция вызывает рекурсивный метод поиска узла.

object\* search(int k) {

object \*res = searchNode(root, k);

if (res != nullptr) return res;

else return nullptr;

}

Рекурсивный алгоритм поиска, с инкрементированием поля c. Входные поля – узел откуда начать поиск и ключ желаемого элемента.

object \*searchNode(Node \*node, int k) {

if (node == nullptr) return nullptr;

if (k == node->key) return node->o;

if (k < node->key) {

c++;

return searchNode(node->left, k);

} else {

c++;

return searchNode(node->right, k);

}

}

Вывод структуры в консоль. Вызывает рекурсивный метод вертикальной распечатки структуры, начиная с корня (нулевой уровень).

void show() {

showNode(root, 0);

}

Рекурсивный метод распечатки структуры, принимает в параметры начальный узел и его уровень. Число отступов в консоли составляет число табуляций, умноженное на уровень глубины объекта.

void showNode(Node \*t, int level) {

if (t == nullptr) {

return;

}

showNode(t->right, level+1);

for (int i = 0; i < level; i++) cout << "\t";

cout << t->o->getValueObj() << "(" << t->o->uploadInString() << ")[" << t->count << "]" << endl;

showNode(t->left, level+1);

}

Удаление элемента состоит из трех функций.

Пользовательская функция, получает во входных параметрах объект и вызывает рекурсивное удаление.

void del(object\* obj) {

root = deleteNode(root, obj);

}

Вспомогательная функция поиска минимального узла для замены удаляемого.

Node\* FindMin(Node\* r) {

while (r->left != nullptr) r = r->left;

return r;

}

Рекурсивный метод удаления объекта. Входные параметры – начальный узел, объект для удаления.

Node\* deleteNode(Node\* r, object\* obj) {

if (r == nullptr) return r;

else if (equal(obj, r->o) < 0) {

r->count--;

r->left = deleteNode(r->left, obj);

}

else if (equal(obj, r->o) > 0) {

r->count--;

r->right = deleteNode(r->right, obj);

}

else {

// нет листьев

if (r->left == nullptr && r->right == nullptr) {

delete r;

r = nullptr;

}

// один лепесток

else if(r->left == nullptr) {

r->count--;

Node\* tmp = r;

r = r->right;

delete tmp;

}

else if(r->right == nullptr) {

r->count--;

Node\* tmp = r;

r = r->left;

delete tmp;

}

// оба лепестка

else {

Node\* tmp = FindMin(r->right);

r->count--;

r->o = tmp->o;

r->key = tmp->key;

r->right = deleteNode(r->right, tmp->o);

}

}

return r;

}

Метод записи в бинарный файл для пользователя. Вызывает рекурсивную функцию записи.

void wrBin() {

ofstream fout("BinaryTree.dat", ios::binary);

if (!fout.is\_open()) cout << "Cannot open file.";

else {

cout << "Binary Tree was written in the binary file.\n";

writeInBinary(fout, root);

}

}

Рекурсивная запись двоичного дерева в бинарный файл. Если узел не пустой, записывает сначала уникальный идентификатор, затем сам объект в двоичный файл. После вызывается запись для левого и правого сына.

void writeInBinary(ofstream &fout, Node \*t) {

if (t == nullptr) {

return;

}

int key = t->o->getId();

fout.write((char\*)&key, sizeof(int));

t->o->writeInBinary(fout);

writeInBinary(fout, t->left);

writeInBinary(fout, t->right);

}

Пользовательская функция чтения структуры из бинарного файла. Открывает поток для чтения fin и вызывает метод последовательного чтения.

void rBin() {

ifstream fin("BinaryTree.dat", ios::binary);

if (!fin.is\_open()) cout << "Cannot open file.";

else {

cout << "Binary Tree was read from the binary file.\n";

readFromB(fin);

}

}

Метод чтения реализован с помощью цикла while, пока не достигнем конца файла. Производится считывание уникального идентификатора, чтобы выделить нужное число байт памяти под чтение объекта. Далее считывается объект, и вызывается функция add вставки в двоичное дерево. Так как двоичный файл используется только для чтения и записи всей структуры без внутрифайловых операций, такая реализация имеет место быть. При таком считывании не нарушается композиция первоначальной структуры, потому что чтение идет последовательно.

void readFromB(ifstream& fin) {

while (true) {

int key;

object \*obj;

fin.read((char\*)&key, sizeof(int));

if (key == 1) {

obj = new Integer;

} else if (key == 2) {

obj = new Date;

}

obj->readFromBinary(fin);

add(obj);

if (fin.peek() == EOF) break;

}

}

* 1. Абстрактный базовый класс объектов object

Абстрактные методы:

object() {} - конструктор

virtual object\* loadFromString(char\*) = 0; - перевод строки в объект

virtual string uploadInString() = 0; - перевод объекта в строку

virtual string getName() = 0; - получение имени класса

virtual int getId() = 0; - получение идентификатора класса

virtual int equals(object\*, object\*) = 0; - сравнение двух объектов

virtual object\* unionObj(object\*, object\*) = 0; - объединение объектов

virtual object\* makeCopy(object\*) = 0; - создание копии объекта

virtual void writeInBinary(ofstream&) {} - запись объекта в бинарный файл

virtual void readFromBinary(ifstream&) {} - чтение объекта из бинарного файла

virtual void getObject() = 0; - ввод объекта с клавиатуры

virtual int getValueObj() = 0; - ключ для дерева

~object() {} – деструктор

* 1. Класс целых чисел Integer, производный от абстрактного

Поле: int value – целое число

Основные методы:

Запись числа из строки осуществляется при помощи функции atoi -функция для приведения строки в числовой вид.

object\* Integer::loadFromString(char \*str) {

this->value = atoi(str);

return this;

}

Загрузка числа в строку осуществляется при помощи метода to\_string().

string Integer::uploadInString() {

return to\_string(this->value);

}

Функция получения имени класса

string Integer::getName() {

return "Integer";

}

Возврат уникального идентификатора

int Integer::getId() {

return 1;

}

Сравнение двух целочисленных объектов

int Integer::equals(object\* o1, object\* o2) {

int i1 = ((Integer\*)o1)->value, i2 = ((Integer\*)o2)->value;

if (i1 > i2) return 1;

else if (i1 < i2) return -1;

else return 0;

}

Сложение двух целочисленных объектов

object\* Integer::unionObj(object\* o1, object\* o2) {

this->value += ((Integer\*)o1)->value;

this->value += ((Integer\*)o2)->value;

return this;

}

Создание копии числа

object\* Integer::makeCopy(object\* o) {

this->setValue(((Integer\*) o)->getValue());

return ((Integer\*)this);

}

Запись числа в бинарный файл

void Integer::writeInBinary(ofstream& fout) {

fout.write((char\*)&value, sizeof(int));

}

Чтение из бинарного файла

void Integer::readFromBinary(ifstream& fin) {

fin.read((char\*)&value, sizeof(int));

}

Перегрузка оператора консольного вывода

ostream& operator<<(ostream& os, Integer& i) {

os << ((Integer\*)&i)->uploadInString();

return os;

}

Перегрузка оператора консольного ввода

istream& operator>>(istream& is, Integer& i) {

is >> i.value;

return is;

}

* 1. Класс дат Date, производный от абстрактного

Поля: int day; - день

int mon; - месяц

int year; - год

int h; - часы

int m; - минуты

int s; - секунды

Основные методы:

Все необходимые методы обрабатывают исключения на корректность даты.

Функция чтения объекта из динамической строки. Поэлементная проверка входного массива символов по шаблону «dd/MM/yyyy hh:mm:ss» последовательная запись в объект при помощи сеттеров.

object\* Date::loadFromString(char \*date) {

int day, mon, year, h, m, s;

if (date[0] == 0) {

day = date[1] - '0';

} else {

day = 10 \* (date[0]-'0') + date[1] - '0';

}

if (date[3] == 0) {

mon = date[4] - '0';

} else {

mon = 10 \* (date[3] - '0') + date[4] - '0';

}

if (date[6] == 0) {

if (date[7] == 0) {

if (date[8] == 0) {

year = date[9] - '0';

} else {

year = 10 \* (date[8] - '0') + date[9] - '0';

}

} else {

year = 100 \* (date[7] - '0') + 10 \* (date[8] - '0') + date[9] - '0';

}

} else {

year = 1000 \* (date[6] - '0') + 100 \* (date[7] - '0') + 10 \* (date[8] - '0') + date[9] - '0';

}

this->setDay(day); this->setMonth(mon); this->setYear(year);

if (date[11] == 0) {

h = date[12] - '0';

} else {

h = 10 \* (date[11] - '0') + date[12] - '0';

}

if (date[14] == 0) {

m = date[15] - '0';

} else {

m = 10 \* (date[14] - '0') + date[15] - '0';

}

if (date[17] == 0) {

s = date[18] - '0';

} else {

s = 10 \* (date[17] - '0') + date[18] - '0';

}

this->setHour(h); this->setMinute(m); this->setSecond(s);

return this;

}

Запись даты в строку при помощи функции sprint. Шаблон такой же, как и в чтении.

string Date::uploadInString() {

char\* str = new char[19];

sprintf(str, "%.2d/%.2d/%.4d %.2d:%.2d:%.2d", this->getDay(), this->getMonth(), this->getYear(),

this->getHour(), this->getMinute(), this->getSecond());

return str;

}

Функция получения имени класса

string Date::getName() {

return "Date";

}

Возврат уникального идентификатора

int Date::getId() {

return 2;

}

Функция сравнения двух дат. Идея состоит в том, что создается по 2 показателя для каждого объекта – суммарное число дней и суммарное число секунд. В циклах происходит их накопление. После сравниваются сначала суммарные числа дней, затем, если они оказались равны, сравниваются суммарные числа секунд.

int Date::equals(object\* o1, object\* o2) {

int first\_d = 0, first\_s = 0, second\_d = 0, second\_s = 0;

for (int i = 1; i < ((Date\*)o1)->getYear(); i++) {

if (i % 4 == 0 || ((i % 100 != 0) && (i % 400 == 0))) {

first\_d += 366;

} else first\_d += 365;

}

for (int i = 1; i < ((Date\*)o1)->getMonth(); i++) {

if (i == 4 || i == 6 || i == 9 || i == 11) {

first\_d += 30;

} else if (i == 2) {

if (((Date\*)o1)->getYear() % 4 == 0 || ((((Date\*)o1)->getYear() % 100 != 0) && (((Date\*)o1)->getYear() % 400 == 0))) {

first\_d += 29;

} else second\_d += 28;

} else first\_d += 31;

}

first\_d += ((Date\*)o1)->getDay();

first\_s += 3600 \* ((Date\*)o1)->getHour() + 60 \* ((Date\*)o1)->getMinute() + ((Date\*)o1)->getSecond();

for (int i = 1; i < ((Date\*)o2)->getYear(); i++) {

if (i % 4 == 0 || ((i % 100 != 0) && (i % 400 == 0))) {

second\_d += 366;

} else second\_d += 365;

}

for (int i = 1; i < ((Date\*)o2)->getMonth(); i++) {

if (i == 4 || i == 6 || i == 9 || i == 11) {

second\_d += 30;

} else if (i == 2) {

if (((Date\*)o2)->getYear() % 4 == 0 || ((((Date\*)o2)->getYear() % 100 != 0) && (((Date\*)o2)->getYear() % 400 == 0))) {

second\_d += 29;

} else second\_d += 28;

} else second\_d += 31;

}

second\_d += ((Date\*)o2)->getDay();

second\_s += 3600 \* ((Date\*)o2)->getHour() + 60 \* ((Date\*)o2)->getMinute() + ((Date\*)o2)->getSecond();

if (first\_d > second\_d) return 1;

else if (first\_d < second\_d) return -1;

else {

if (first\_s > second\_s) return 1;

else if (first\_s < second\_s) return -1;

else return 0;

}

}

Метод объединения двух дат реализован инкрементированием полей в цикле. Границами циклов являются суммы соответствующих полей объектов. Функции инкрементирования реализованы с учетом високосного года и прочих тонкостях даты (к примеру 30 дней в месяце).

object\* Date::unionObj(object\* o1, object\* o2) {

int sec = ((Date\*)o1)->getSecond() + ((Date\*)o2)->getSecond();

for (int i = 0; i < sec; i++) {

this->incSecond();

}

int min = ((Date\*)o1)->getMinute() + ((Date\*)o2)->getMinute();

for (int i = 0; i < min; i++) {

this->incMinutes();

}

int hours = ((Date\*)o1)->getHour() + ((Date\*)o2)->getHour();

for (int i = 0; i < hours; i++) {

this->incHour();

}

int years = ((Date\*)o1)->getYear() + ((Date\*)o2)->getYear() - 1;

for (int i = 0; i < years; i++) {

this->incYear();

}

int months = ((Date\*)o1)->getMonth() + ((Date\*)o2)->getMonth() - 1;

for (int i = 0; i < months; i++) {

this->incMonth();

}

int days = ((Date\*)o1)->getDay() + ((Date\*)o2)->getDay() - 1;

for (int i = 0; i < days; i++) {

this->incDay();

}

return this;

}

Создание динамической копии даты при помощи сеттеров и геттеров.

object\* Date::makeCopy(object\* o) {

this->setDay(((Date\*) o)->getDay());

this->setMonth(((Date\*) o)->getMonth());

this->setYear(((Date\*) o)->getYear());

this->setHour(((Date\*) o)->getHour());

this->setMinute(((Date\*) o)->getMinute());

this->setSecond(((Date\*) o)->getSecond());

return (Date\*)this;

}

Запись даты в бинарный файл происходит поэлементно.

void Date::writeInBinary(ofstream& fout) {

fout.write((char\*)&day, sizeof(int));

fout.write((char\*)&mon, sizeof(int));

fout.write((char\*)&year, sizeof(int));

fout.write((char\*)&h, sizeof(int));

fout.write((char\*)&m, sizeof(int));

fout.write((char\*)&s, sizeof(int));

}

Чтение из бинарного файла также поэлементное.

void Date::readFromBinary(ifstream& fin) {

fin.read((char\*)&day, sizeof(int));

fin.read((char\*)&mon, sizeof(int));

fin.read((char\*)&year, sizeof(int));

fin.read((char\*)&h, sizeof(int));

fin.read((char\*)&m, sizeof(int));

fin.read((char\*)&s, sizeof(int));

}

Перегрузка оператора вывода включает в себя метод строкового представления даты.

ostream& operator<<(ostream& os, Date& d) {

os << ((Date\*)&d)->uploadInString();

return os;

}

Перегрузка оператора консольного ввода обрабатывается исключениями.

istream& operator>>(istream& is, Date& d) {

int day, mon, year, h, m, s;

is >> day >> mon >> year >> h >> m >> s;

try

{

if (day > 31 || day < 0 || mon > 12 || mon < 0 || year < 0 ||

h > 23 || h < 0 || m > 59 || m < 0 || s > 59 || s < 0) throw runtime\_error("Invalid Date Format.");

d.setDay(day);

d.setMonth(mon);

d.setYear(year);

d.setHour(h);

d.setMinute(m);

d.setSecond(s);

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

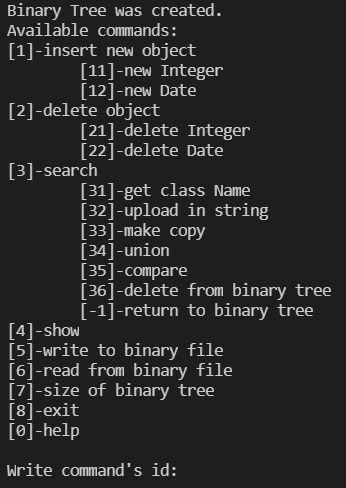
}

return is;

}

# Описание пользовательского интерфейса

Взаимодействие с коллекцией осуществляется через консольное меню:



При запуске программы уже выделена динамическая память для двоичного дерева. Доступные пользователю команды:

[1] – вставка нового объекта.

[11] – вставить целое число

[12] – вставить дату

[2] – удаление объекта.

[21] – удалить целое число

[22] – удалить дату

[3] – поиск

[31] – получить название класса найденного объекта

[32] – получить строковое представление найденного объекта

[33] – сделать динамическую копию найденного элемента

[34] – объединение 2х однотипных объектов (ввод второго с клавиатуры)

[35] – сравнение 2х объектов (ввод второго с клавиатуры)

[36] – удалить найденный элемент из дерева

[-1] – выход к функциям двоичного дерева

[4] – распечатка структуры в консоль

[5] – запись структуры в последовательный двоичный файл

[6] – чтение структуры из последовательного двоичного файла

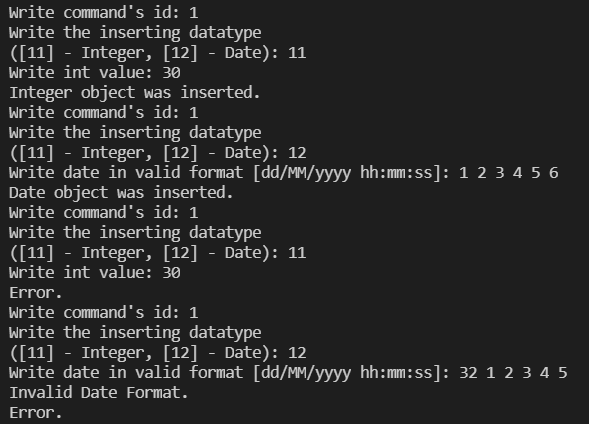
[7] – возврат количества узлов двоичного дерева

[8] – выход из меню

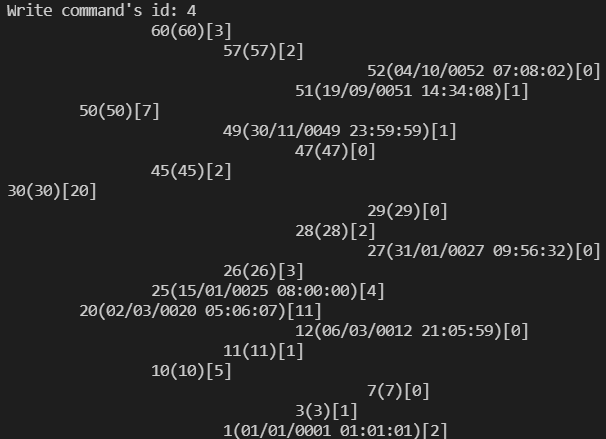
[0] – вывод в консоль список команд меню

Сразу после компиляции программы, пользователю предлагается ввести id номер интересующей команды. Данное взаимодействие реализовано в бесконечном цикле, пока не введен индекс выхода из меню.

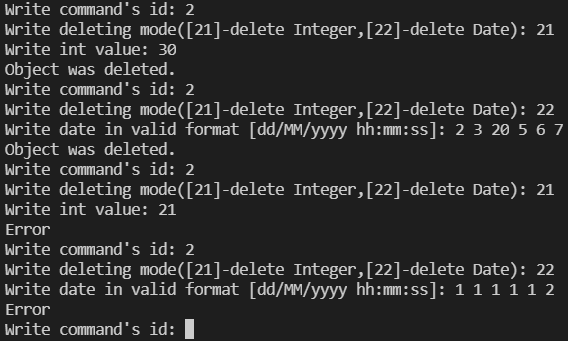
Ниже на скриншоте продемонстрировано поведение программы при попытке вставки объекта в структуру. Можно увидеть, что выводится сообщение об ошибке при попытке ввода уже существующего объекта, или, в случае с датой, при некорректном формате введенной даты.



Распечатка коллекции в консоли выглядит следующим образом, сначала выводится ключ, затем в круглых скобках указано строковое представление элемента, в квадратных скобках – число узлов, исходящих из текущего.



При удалении элемента интерфейс ведет себя следующим образом. В случае успешного удаления выводится соответствующее сообщение. При некорректном вводе удаляемого элемента или при отсутствии данного элемента в коллекции выводится сообщение об ошибке.



Успешный поиск элемента предоставляет несколько операций над найденным объектом.

На примере поиска даты ниже представлена хронология работы с интерфейсом.

31 – возвращает название класса найденного объекта в виде строки;

32 – возвращает строковое представление найденного объекта;

33 – создает копию найденного объекта;

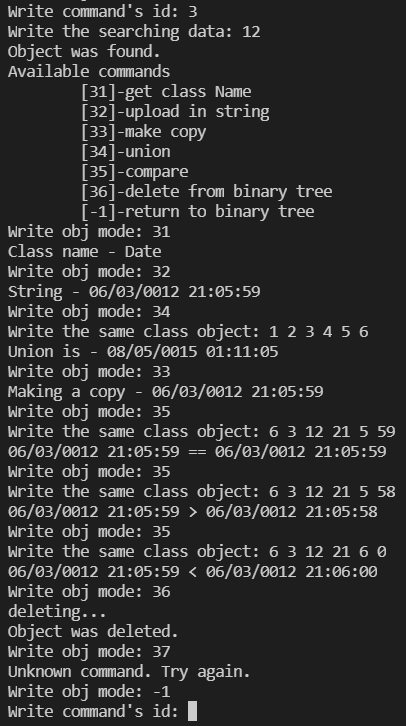
34 – сложение двух однотипных объектов, для этого пользователю нужно корректно ввести второе слагаемое. Возвращается строковое представление суммы объектов;

35 – сравнение двух однотипных объектов. Пользователю нужно ввести сравниваемый объект с клавиатуры в корректной форме;

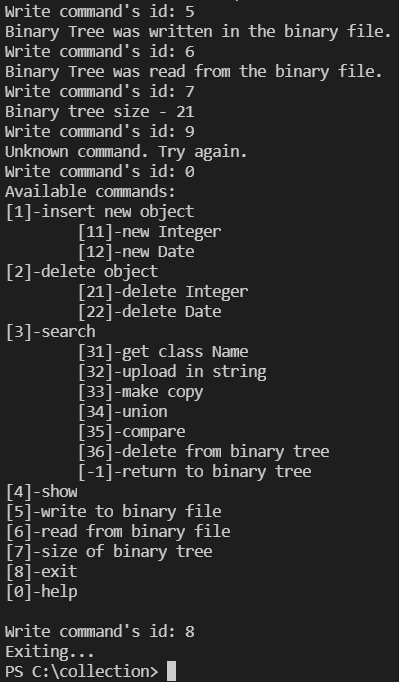
36 – удаление найденного объекта из коллекции;

-1 – возвращает пользователя к командам относительно коллекции.

Прочие введенные значения ведут к сообщению об ошибке и повторному запрашиванию команды.



Отклик программы при оставшихся командах можно увидеть на скриншоте ниже. Учтено также «непопадание» в диапазон id команд.



# Описание работы программы на контрольных примерах

Программа была протестирована на типе данных Integer, который является производным от абстрактного базового класса object. Тестировался метод поиска целочисленного элемента в двоичном дереве, заполненном случайным образом. Для данного теста был написан метод search2(int k), который возвращает число узлов, пройденных до нахождения желаемого элемента; если элемент не был найден, возвращается -1. В данном тесте было реализовано 10% «непопадание» путем генерации 1.1\*n случайных элементов. Текст программы представлен в приложении. Результаты были сравнены с теоретическими и представлены ниже в виде таблиц и графиков.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | 1000 | 2000 | 3000 | 4000 | 5000 | 6000 | 7000 | 8000 | 9000 | 10000 |
| Т | 13,852 | 15,242 | 16,056 | 16,632 | 17,079 | 17,446 | 17,755 | 18,022 | 18,259 | 18,469 |
| П | 9,398 | 10,434 | 11,4747 | 11,88 | 12,8428 | 13,4507 | 13,7997 | 13,9723 | 14,9998 | 13,6482 |
| Time | 0,023 | 0,115 | 0,248 | 0,397 | 0,66 | 0,663 | 0,964 | 1,637 | 1,795 | 2,122 |

В таблице выше:

n – число элементов в коллекции;

Т – Теоретическое среднее число пройденных узлов;

П – практическое среднее число пройденных узлов;

Time – время выполнения теста в секундах.

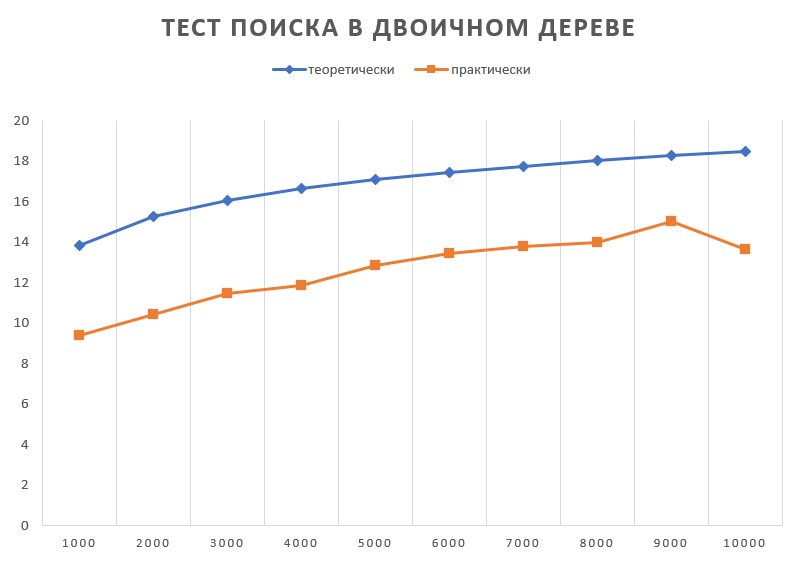


Рисунок 3. График теста поиска

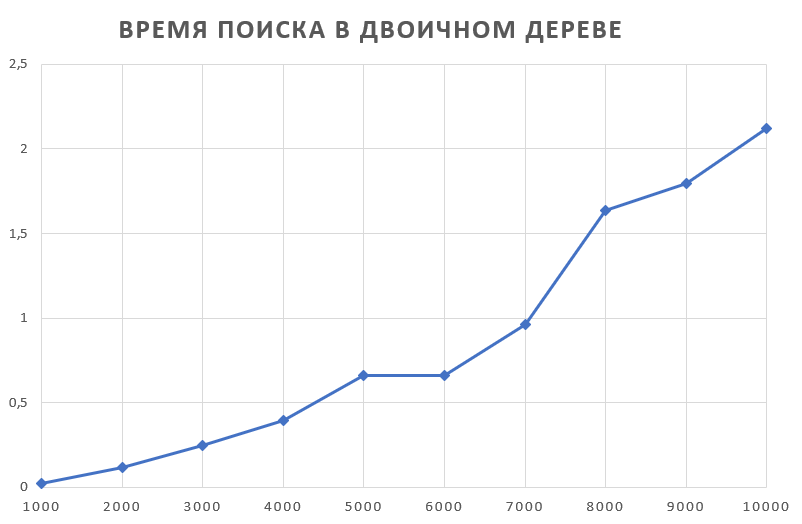


Рисунок 4. Время выполнения теста

# Выводы

В ходе выполнения курсового проекта мною был разработан интерфейс для объединения в двоичное дерево множества объектов различных классов. Элементы дерева хранят указатели на объекты абстрактного класса, это позволяет осуществить полиморфизм – один из главных принципов объектно-ориентированного программирования.

Программа была протестирована на поиск при большом количестве элементов. Анализируя графики, можно сделать вывод, что практический поиск находит объект примерно на 5 узла раньше теоретического результата. Время поиска при интерполяции имеет логарифмический рост, что является эффективным показателем. Полученный разброс данных является вычислительной погрешностью программы.

Главным преимуществом данной коллекции является функция поиска, которая имеет логарифмическую вычислительную сложность, в данном проекте я лично в этом убедился.

К недостаткам можно отнести сложность поддержания сбалансированности дерева. Также стоит отметить, что для хранения малого количества элементов лучше подойдут другие структуры данных.

# Список литературы

1. Романенко Т.А. Учебное пособие «Разработка программных коллекций данных», НГТУ 2020г.
2. Ravesli.com - Урок №168. Чистые виртуальные функции, Интерфейсы и Абстрактные классы.
3. Prog-cpp.ru – Структуры данных. Дерево.

# Приложение

Файлы программы.

* btree.hpp
* object.hpp
* Integer.hpp
* Date.hpp
* main.cpp
* test.cpp
* BinaryTree.dat
* Btree.hpp

#pragma once

#include <iostream>

#include <fstream>

#include "object.hpp"

#include "Integer.hpp"

#include "Date.hpp"

using namespace std;

int equal(object\* o1, object\* o2) {

if ((o1->getId() == 1 && o2->getId() == 2) || (o1->getId() == 2 && o2->getId() == 1)) {

if (o1->getValueObj() < o2->getValueObj()) return -1;

else if (o1->getValueObj() > o2->getValueObj()) return 1;

else return 0;

}

else return o1->equals(o1,o2);

}

class Btree;

class Node {

public:

Node() {

o = nullptr;

key = 0;

left = nullptr;

right = nullptr;

count = 0;

}

Node(object\* obj) {

o = obj;

key = obj->getValueObj();

left = nullptr;

right = nullptr;

count = 0;

}

friend class Btree;

private:

int key;

object \*o;

Node \*left;

Node \*right;

int count;

};

class Btree {

public:

Btree() {

root = nullptr;

c = 0;

}

// кол-во узлов в дереве

int size() {

if (root != nullptr) return root->count + 1;

return 0;

}

// добавление

bool add(object \*obj) {

Node \*n = new Node(obj);

bool ins; //флаг об успешной/неуспешной вставке

if (!n) {

return false;

}

// проверка на невалидность даты

if (((Date\*)obj)->getDay() == 1 && ((Date\*)obj)->getMonth() == 1 && ((Date\*)obj)->getYear() == 1 &&

((Date\*)obj)->getHour() == 0 && ((Date\*)obj)->getMinute() == 0 && ((Date\*)obj)->getSecond() == 0) return false;

root = insertNode(root, n, ins);

return ins;

}

// вставка узла

Node\* insertNode(Node \*node, Node \*elem, bool &inserted) {

bool ins;

if (node == nullptr) {

inserted = true;

return elem;

}

if (equal(elem->o, node->o) < 0) {

node->left = insertNode(node->left, elem, ins);

}

else if (equal(elem->o, node->o) > 0) {

node->right = insertNode(node->right, elem, ins);

}

else {

inserted = false;

return node;

}

if (ins) node->count++;

inserted = ins;

return node;

}

// кол-во пройденных узлов до найденного

int search2(int k) {

object\* res = searchNode(root, k);

if (res != nullptr) {

int cc = c;

c = 0;

return cc;

}

return -1; // в случае отсутствия элемента в дереве

}

// поиск элемента в дереве

object\* search(int k) {

object \*res = searchNode(root, k);

if (res != nullptr) return res;

else return nullptr;

}

// поиск узла

object \*searchNode(Node \*node, int k) {

if (node == nullptr) return nullptr;

if (k == node->key) return node->o;

if (k < node->key) {

c++;

return searchNode(node->left, k);

} else {

c++;

return searchNode(node->right, k);

}

}

// вывод структуры

void show() {

showNode(root, 0);

}

// распечатка rihgt -> root -> left

void showNode(Node \*t, int level) {

if (t == nullptr) {

return;

}

showNode(t->right, level+1);

for (int i = 0; i < level; i++) cout << "\t";

cout << t->o->getValueObj() << "(" << t->o->uploadInString() << ")[" << t->count << "]" << endl;

showNode(t->left, level+1);

}

// удаление элемента для пользователя

void del(object\* obj) {

root = deleteNode(root, obj);

}

// поиск минимального для замены

Node\* FindMin(Node\* r) {

while (r->left != nullptr) r = r->left;

return r;

}

// рекурсивное удаление узла

Node\* deleteNode(Node\* r, object\* obj) {

if (r == nullptr) return r;

else if (equal(obj, r->o) < 0) {

r->count--;

r->left = deleteNode(r->left, obj);

}

else if (equal(obj, r->o) > 0) {

r->count--;

r->right = deleteNode(r->right, obj);

}

// нашли, что удалять

else {

// лепесток

if (r->left == nullptr && r->right == nullptr) {

delete r;

r = nullptr;

}

// один сын

else if(r->left == nullptr) {

r->count--;

Node\* tmp = r;

r = r->right;

delete tmp;

}

else if(r->right == nullptr) {

r->count--;

Node\* tmp = r;

r = r->left;

delete tmp;

}

// оба сына

else {

Node\* tmp = FindMin(r->right);

r->count--;

r->o = tmp->o;

r->key = tmp->key;

r->right = deleteNode(r->right, tmp->o);

}

}

return r;

}

// запись структуры в бинарный файл для пользователя

void wrBin() {

ofstream fout("BinaryTree.dat", ios::binary);

if (!fout.is\_open()) cout << "Cannot open file.";

else {

cout << "Binary Tree was written in the binary file.\n";

writeInBinary(fout, root);

}

}

// запись структуры в бинарный файл

void writeInBinary(ofstream &fout, Node \*t) {

if (t == nullptr) {

return;

}

// получаем ID объекта

int key = t->o->getId();

fout.write((char\*)&key, sizeof(int));

t->o->writeInBinary(fout);

writeInBinary(fout, t->left);

writeInBinary(fout, t->right);

}

// чтение структуры из бинарного файла для пользоователя

void rBin() {

ifstream fin("BinaryTree.dat", ios::binary);

if (!fin.is\_open()) cout << "Cannot open file.";

else {

cout << "Binary Tree was read from the binary file.\n";

readFromB(fin);

}

}

// чтение структуры из бинарного файла

void readFromB(ifstream& fin) {

// считываем, пока не конец файла

while (true) {

int key;

object \*obj;

fin.read((char\*)&key, sizeof(int));

if (key == 1) {

obj = new Integer;

} else if (key == 2) {

obj = new Date;

}

obj->readFromBinary(fin);

add(obj);

if (fin.peek() == EOF) break;

}

}

// деструктор

~Btree() {

delete root;

c = 0;

}

private:

Node \*root;

int c;

};

* Object.hpp

#pragma once

#include <iostream>

#include <string>

using namespace std;

class object {

public:

object() {}

// перевод строки в объект

virtual object\* loadFromString(char\*) = 0;

// перевод объекта в строку

virtual string uploadInString() = 0;

// получение имени класса

virtual string getName() = 0;

// получение идентификатора класса

virtual int getId() = 0;

// сравнение двух объектов

virtual int equals(object\*, object\*) = 0;

// сумма объектов

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

// создание копии объекта

virtual object\* makeCopy(object\*) = 0;

// запись объекта в бинарный файл

virtual void writeInBinary(ofstream&) {}

// чтение объекта из бинарного файла

virtual void readFromBinary(ifstream&) {}

// ввод объекта с клавиатуры

virtual void getObject() = 0;

// ключ для дерева

virtual int getValueObj() = 0;

~object() {}

};

* Integer.hpp

#pragma once

#include <iostream>

#include <string.h>

#include <fstream>

#include "math.h"

#include "object.hpp"

using namespace std;

class Integer : public object {

public:

Integer();

Integer(int);

int getValue();

void setValue(int);

// перевод строки в число

object\* loadFromString(char\*);

// перевод числа в строку

string uploadInString();

// получение имени класса

string getName();

// получение идентификатора класса

int getId();

// сравнение двух объектов

int equals(object\*, object\*);

// сумма объектов

object\* unionObj(object\*, object\*);

// создание копии объекта

object\* makeCopy(object\*);

// запись объекта в бинарный файл

void writeInBinary(ofstream&);

// чтение объекта из бинарного файла

void readFromBinary(ifstream&);

// ввод числа с клавиатуры

void getObject();

// получение ключа для дерева

int getValueObj();

Integer& operator=(const Integer&);

Integer operator+(const Integer&);

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

friend ostream& operator<<(ostream&, Integer&);

friend istream& operator>>(istream&, Integer&);

~Integer();

private:

int value;

};

Integer::Integer() {

this->value = 0;

}

Integer::Integer(int value) {

this->value = value;

}

int Integer::getValue() {

return this->value;

}

// получение ключа объекта

int Integer::getValueObj() {

return this->value;

}

void Integer::setValue(int value) {

this->value = value;

}

// чтение числа из строки

object\* Integer::loadFromString(char \*str) {

this->value = atoi(str);

return this;

}

// загрузка числа в строку

string Integer::uploadInString() {

return to\_string(this->value);

}

// имя класса

string Integer::getName() {

return "Integer";

}

// уникальный идентификатор

int Integer::getId() {

return 1;

}

// сравнение двух целочисленных объектов

int Integer::equals(object\* o1, object\* o2) {

int i1 = ((Integer\*)o1)->value, i2 = ((Integer\*)o2)->value;

if (i1 > i2) return 1;

else if (i1 < i2) return -1;

else return 0;

}

// сложение двух целых чисел

object\* Integer::unionObj(object\* o1, object\* o2) {

this->value += ((Integer\*)o1)->value;

this->value += ((Integer\*)o2)->value;

return this;

}

// создание копии числа

object\* Integer::makeCopy(object\* o) {

this->setValue(((Integer\*) o)->getValue());

return ((Integer\*)this);

}

// запись в бинарный файл

void Integer::writeInBinary(ofstream& fout) {

fout.write((char\*)&value, sizeof(int));

}

// чтение из бинарного файла

void Integer::readFromBinary(ifstream& fin) {

fin.read((char\*)&value, sizeof(int));

}

// консольный ввод

void Integer::getObject() {

cin >> this->value;

}

// оператор присваивания

Integer& Integer::operator=(const Integer& n) {

this->value = n.value;

return \*this;

}

// оператор сложения

Integer Integer::operator+(const Integer& n) {

Integer res;

res.value = this->value + n.value;

return res;

}

// оператор вычитания

Integer operator-(const Integer& first, const Integer& second) {

Integer res;

res.value = first.value - second.value;

return res;

}

// оператор консольного вывода

ostream& operator<<(ostream& os, Integer& i) {

os << ((Integer\*)&i)->uploadInString();

return os;

}

// оператор консольного ввода

istream& operator>>(istream& is, Integer& i) {

is >> i.value;

return is;

}

Integer::~Integer() {

delete this;

}

* Date.hpp

#pragma once

#include <iostream>

#include <string.h>

#include <fstream>

#include "math.h"

#include "object.hpp"

using namespace std;

class Date : public object {

public:

Date();

Date(int, int, int);

Date(int, int, int, int, int, int);

// геттеры полей

int getDay();

int getMonth();

int getYear();

int getHour();

int getMinute();

int getSecond();

int getValue();

// сеттеры полей

void setYear(int);

void setMonth(int);

void setDay(int);

void setHour(int);

void setMinute(int);

void setSecond(int);

// инкрементирование полей

void incYear();

void incMonth();

void incDay();

void incHour();

void incMinutes();

void incSecond();

// перевод строки в дату

object\* loadFromString(char\*);

// перевод даты в строку

string uploadInString();

// получение имени класса

string getName();

// получение идентификатора класса

int getId();

// сравнение двух объектов

int equals(object\*, object\*);

// сумма объектов

object\* unionObj(object\*, object\*);

// создание копии объекта

object\* makeCopy(object\*);

// запись объекта в бинарный файл

void writeInBinary(ofstream&);

// чтение объекта из бинарного файла

void readFromBinary(ifstream&);

// ввод числа с клавиатуры

void getObject();

int getValueObj();

Date& operator=(const Date& d);

friend ostream& operator<<(ostream&, Date&);

friend istream& operator>>(istream&, Date&);

~Date();

private:

int day;

int mon;

int year;

int h;

int m;

int s;

};

Date::Date() {

this->setDay(1);

this->setMonth(1);

this->setYear(1);

this->setHour(0);

this->setMinute(0);

this->setSecond(0);

}

Date::Date(int day, int mon, int year) {

try {

if (day > 31 || day < 0 || mon > 12 || mon < 0 || year < 0) throw runtime\_error("Invalid Date Format.");

this->setDay(day);

this->setMonth(mon);

this->setYear(year);

this->setHour(0);

this->setMinute(0);

this->setSecond(0);

} catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

}

}

Date::Date(int day, int mon, int year, int h, int m, int s) {

try {

if (day > 31 || day < 0 || mon > 12 || mon < 0 || year < 0 ||

h > 23 || h < 0 || m > 59 || m < 0 || s > 59 || s < 0) throw runtime\_error("Invalid Date Format.");

this->setDay(day);

this->setMonth(mon);

this->setYear(year);

this->setHour(h);

this->setMinute(m);

this->setSecond(s);

} catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

}

}

int Date::getDay() {

return this->day;

}

int Date::getMonth() {

return this->mon;

}

int Date::getYear() {

return this->year;

}

int Date::getHour() {

return this->h;

}

int Date::getMinute() {

return this->m;

}

int Date::getSecond() {

return this->s;

}

int Date::getValueObj() {

return this->year;

}

void Date::setYear(int year) {

try

{

if (year < 0) throw runtime\_error("Invalid Date Format, check year");

this->year = year;

}

catch(exception& e)

{

cerr << e.what() << '\n';

}

}

void Date::setMonth(int mon) {

try

{

if (mon > 12 || mon < 0) throw runtime\_error("Invalid Date Format, check month");

this->mon = mon;

}

catch(exception& e)

{

cerr << e.what() << '\n';

}

}

void Date::setDay(int day) {

try

{

if (day > 31 || day < 0) throw runtime\_error("Invalid Date Format, check day");

this->day = day;

}

catch(exception& e)

{

cout<< e.what() << '\n';

}

}

void Date::setHour(int h) {

try

{

if (h > 23 || h < 0) throw runtime\_error("Invalid Date Format, check hours");

this->h = h;

}

catch(exception& e)

{

cerr << e.what() << '\n';

}

}

void Date::setMinute(int m) {

try

{

if (m > 59 || m < 0) throw runtime\_error("Invalid Date Format, check minutes");

this->m = m;

}

catch(exception& e)

{

cerr << e.what() << '\n';

}

}

void Date::setSecond(int s) {

try

{

if (s > 59 || s < 0) throw runtime\_error("Invalid Date Format, check seconds");

this->s = s;

}

catch(exception& e)

{

cerr << e.what() << '\n';

}

}

void Date::incYear() {

this->setYear(this->getYear() + 1);

}

void Date::incMonth() {

if (this->getMonth() == 12) {

this->incYear();

this->setMonth(1);

} else {

this->setMonth(this->getMonth() + 1);

}

}

void Date::incDay() {

if ((this->getMonth() == 4 || this->getMonth() == 6 ||

this->getMonth() == 9 || this->getMonth() == 11) && this->getDay() == 30) {

this->incMonth();

this->setDay(1);

} else if (this->getMonth() == 2) {

if (this->getYear() % 4 == 0) {

if (this->getYear() % 100 == 0 && this->getYear() % 400 != 0) {

if (this->getDay() == 28) {

this->incMonth();

this->setDay(1);

} else {

this->setDay(this->getDay() + 1);

}

} else {

if (this->getDay() == 29) {

this->incMonth();

this->setDay(1);

} else {

this->setDay(this->getDay() + 1);

}

}

} else if (this->getDay() == 28) {

this->incMonth();

this->setDay(1);

} else {

this->setDay(this->getDay() + 1);

}

} else if (this->getDay() == 31) {

this->incMonth();

this->setDay(1);

} else {

this->setDay(this->getDay() + 1);

}

}

void Date::incHour() {

if (this->getHour() == 23) {

this->incDay();

this->setHour(0);

} else {

this->setHour(this->getHour() + 1);

}

}

void Date::incMinutes() {

if (this->getMinute() == 59) {

this->incHour();

this->setMinute(0);

} else {

this->setMinute(this->getMinute() + 1);

}

}

void Date::incSecond() {

if (this->getSecond() == 59) {

this->incMinutes();

this->setSecond(0);

} else {

this->setSecond(this->getSecond() + 1);

}

}

object\* Date::loadFromString(char \*date) {

int day, mon, year, h, m, s;

if (date[0] == 0) {

day = date[1] - '0';

} else {

day = 10 \* (date[0]-'0') + date[1] - '0';

}

if (date[3] == 0) {

mon = date[4] - '0';

} else {

mon = 10 \* (date[3] - '0') + date[4] - '0';

}

if (date[6] == 0) {

if (date[7] == 0) {

if (date[8] == 0) {

year = date[9] - '0';

} else {

year = 10 \* (date[8] - '0') + date[9] - '0';

}

} else {

year = 100 \* (date[7] - '0') + 10 \* (date[8] - '0') + date[9] - '0';

}

} else {

year = 1000 \* (date[6] - '0') + 100 \* (date[7] - '0') + 10 \* (date[8] - '0') + date[9] - '0';

}

this->setDay(day); this->setMonth(mon); this->setYear(year);

if (date[11] == 0) {

h = date[12] - '0';

} else {

h = 10 \* (date[11] - '0') + date[12] - '0';

}

if (date[14] == 0) {

m = date[15] - '0';

} else {

m = 10 \* (date[14] - '0') + date[15] - '0';

}

if (date[17] == 0) {

s = date[18] - '0';

} else {

s = 10 \* (date[17] - '0') + date[18] - '0';

}

this->setHour(h); this->setMinute(m); this->setSecond(s);

return this;

}

string Date::uploadInString() {

char\* str = new char[19];

sprintf(str, "%.2d/%.2d/%.4d %.2d:%.2d:%.2d", this->getDay(), this->getMonth(), this->getYear(),

this->getHour(), this->getMinute(), this->getSecond());

return str;

}

string Date::getName() {

return "Date";

}

int Date::getId() {

return 2;

}

int Date::equals(object\* o1, object\* o2) {

int first\_d = 0, first\_s = 0, second\_d = 0, second\_s = 0;

for (int i = 1; i < ((Date\*)o1)->getYear(); i++) {

if (i % 4 == 0 || ((i % 100 != 0) && (i % 400 == 0))) {

first\_d += 366;

} else first\_d += 365;

}

for (int i = 1; i < ((Date\*)o1)->getMonth(); i++) {

if (i == 4 || i == 6 || i == 9 || i == 11) {

first\_d += 30;

} else if (i == 2) {

if (((Date\*)o1)->getYear() % 4 == 0 || ((((Date\*)o1)->getYear() % 100 != 0) && (((Date\*)o1)->getYear() % 400 == 0))) {

first\_d += 29;

} else first\_d += 28;

} else first\_d += 31;

}

first\_d += ((Date\*)o1)->getDay();

first\_s += 3600 \* ((Date\*)o1)->getHour() + 60 \* ((Date\*)o1)->getMinute() + ((Date\*)o1)->getSecond();

for (int i = 1; i < ((Date\*)o2)->getYear(); i++) {

if (i % 4 == 0 || ((i % 100 != 0) && (i % 400 == 0))) {

second\_d += 366;

} else second\_d += 365;

}

for (int i = 1; i < ((Date\*)o2)->getMonth(); i++) {

if (i == 4 || i == 6 || i == 9 || i == 11) {

second\_d += 30;

} else if (i == 2) {

if (((Date\*)o2)->getYear() % 4 == 0 || ((((Date\*)o2)->getYear() % 100 != 0) && (((Date\*)o2)->getYear() % 400 == 0))) {

second\_d += 29;

} else second\_d += 28;

} else second\_d += 31;

}

second\_d += ((Date\*)o2)->getDay();

second\_s += 3600 \* ((Date\*)o2)->getHour() + 60 \* ((Date\*)o2)->getMinute() + ((Date\*)o2)->getSecond();

if (first\_d > second\_d) return 1;

else if (first\_d < second\_d) return -1;

else {

if (first\_s > second\_s) return 1;

else if (first\_s < second\_s) return -1;

else return 0;

}

}

object\* Date::unionObj(object\* o1, object\* o2) {

int sec = ((Date\*)o1)->getSecond() + ((Date\*)o2)->getSecond();

int min = ((Date\*)o1)->getMinute() + ((Date\*)o2)->getMinute();

int hours = ((Date\*)o1)->getHour() + ((Date\*)o2)->getHour();

int years = ((Date\*)o1)->getYear() + ((Date\*)o2)->getYear();

int months = ((Date\*)o1)->getMonth() + ((Date\*)o2)->getMonth();

int days = ((Date\*)o1)->getDay() + ((Date\*)o2)->getDay();

for (int i = 1; i < days; i++) {

this->incDay();

}

for (int i = 1; i < months; i++) {

this->incMonth();

}

for (int i = 1; i < years; i++) {

this->incYear();

}

for (int i = 0; i < sec; i++) {

this->incSecond();

}

for (int i = 0; i < min; i++) {

this->incMinutes();

}

for (int i = 0; i < hours; i++) {

this->incHour();

}

return this;

}

object\* Date::makeCopy(object\* o) {

this->setDay(((Date\*) o)->getDay());

this->setMonth(((Date\*) o)->getMonth());

this->setYear(((Date\*) o)->getYear());

this->setHour(((Date\*) o)->getHour());

this->setMinute(((Date\*) o)->getMinute());

this->setSecond(((Date\*) o)->getSecond());

return (Date\*)this;

}

void Date::writeInBinary(ofstream& fout) {

fout.write((char\*)&day, sizeof(int));

fout.write((char\*)&mon, sizeof(int));

fout.write((char\*)&year, sizeof(int));

fout.write((char\*)&h, sizeof(int));

fout.write((char\*)&m, sizeof(int));

fout.write((char\*)&s, sizeof(int));

}

void Date::readFromBinary(ifstream& fin) {

fin.read((char\*)&day, sizeof(int));

fin.read((char\*)&mon, sizeof(int));

fin.read((char\*)&year, sizeof(int));

fin.read((char\*)&h, sizeof(int));

fin.read((char\*)&m, sizeof(int));

fin.read((char\*)&s, sizeof(int));

}

void Date::getObject() {

cin >> this->day >> this->mon >> this->year >> this->h >> this->m >> this->s;

}

Date& Date::operator=(const Date& d) {

try {

if (day > 31 || day < 0 || mon > 12 || mon < 0 || year < 0 ||

h > 23 || h < 0 || m > 59 || m < 0 || s > 59 || s < 0) throw runtime\_error("Invalid Date Format.");

this->setDay(d.day);

this->setMonth(d.mon);

this->setYear(d.year);

this->setHour(d.h);

this->setMinute(d.m);

this->setSecond(d.s);

} catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

}

return \*this;

}

ostream& operator<<(ostream& os, Date& d) {

os << ((Date\*)&d)->uploadInString();

return os;

}

istream& operator>>(istream& is, Date& d) {

int day, mon, year, h, m, s;

is >> day >> mon >> year >> h >> m >> s;

try

{

if (day > 31 || day < 0 || mon > 12 || mon < 0 || year < 0 ||

h > 23 || h < 0 || m > 59 || m < 0 || s > 59 || s < 0) throw runtime\_error("Invalid Date Format.");

d.setDay(day);

d.setMonth(mon);

d.setYear(year);

d.setHour(h);

d.setMinute(m);

d.setSecond(s);

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

}

return is;

}

Date::~Date() {

delete this;

}

* Main.cpp

#include <iostream>

#include "Date.hpp"

#include "Integer.hpp"

#include "btree.hpp"

using namespace std;

int main() {

cout << "Binary Tree was created.\n";

string commands = "Available commands:\n[1]-insert new object\n\t[11]-new Integer\n\t[12]-new Date\n[2]-delete object\n\t[21]-delete Integer\n\t[22]-delete Date\n[3]-search\n\t[31]-get class Name\n\t[32]-upload in string\n\t[33]-make copy\n\t[34]-union\n\t[35]-compare\n\t[36]-delete from binary tree\n\t[-1]-return to binary tree\n[4]-show\n[5]-write to binary file\n[6]-read from binary file\n[7]-size of binary tree\n[8]-exit\n[0]-help\n\nWrite command's id: ";

cout << commands;

Btree \*bt = new Btree;

int mode, insMode, delMode;

int objMode;

cin >> mode;

while (true) {

// вставка объекта

if (mode == 1) {

cout << "Write the inserting datatype\n([11] - Integer, [12] - Date): ";

cin >> insMode;

if (insMode == 11) {

cout << "Write int value: ";

object \*i = new Integer;

cin >> \*(Integer\*)i;

if (bt->add(i)) cout << "Integer object was inserted.\n";

else cout << "Error.\n";

}

else if (insMode == 12) {

cout << "Write date in valid format [dd/MM/yyyy hh:mm:ss]: ";

object \*d = new Date;

cin >> \*(Date\*)d;

if (bt->add(d)) cout << "Date object was inserted.\n";

else cout << "Error.\n";

}

else {

cout << "Invalid datatype, try again.\n";

}

cout << "Write command's id: ";

}

// удаление объекта

if (mode == 2) {

object\* dell;

cout << "Write deleting mode([21]-delete Integer,[22]-delete Date): ";

cin >> delMode;

if (delMode == 21) {

cout << "Write int value: ";

dell = new Integer;

}

else if (delMode == 22) {

cout << "Write date in valid format [dd/MM/yyyy hh:mm:ss]: ";

dell = new Date;

}

dell->getObject();

if (bt->search(dell->getValueObj()) != nullptr) {

if (bt->search(dell->getValueObj())->getId() == dell->getId() && (equal(bt->search(dell->getValueObj()), dell) == 0)) {

bt->del(dell);

cout << "Object was deleted.\n";

}

else cout << "Error\n";

}

else cout << "Error\n";

cout << "Write command's id: ";

}

// поиск объекта

if (mode == 3) {

cout << "Write the searching data: ";

int k;

cin >> k;

object\* res = bt->search(k);

if (res != nullptr) {

cout << "Object was found.\n";

int id = res->getId();

string objCommands = "Available commands\n\t[31]-get class Name\n\t[32]-upload in string\n\t[33]-make copy\n\t[34]-union\n\t[35]-compare\n\t[36]-delete from binary tree\n\t[-1]-return to binary tree\n";

cout << objCommands;

cout << "Write obj mode: ";

cin >> objMode;

while (true) {

// выход к командам дерева

if (objMode == -1) {

break;

}

// вывод названия класса

if (objMode == 31) {

cout << "Class name - ";

if (id == 1) cout << ((Integer\*)res)->getName() << endl;

else if (id == 2) cout << ((Date\*)res)->getName() << endl;

cout << "Write obj mode: ";

cin >> objMode;

}

// строковое представление

if (objMode == 32) {

string str = res->uploadInString();

cout << "String - " << str << endl;

cout << "Write obj mode: ";

cin >> objMode;

}

if (objMode == 33) {

cout << "Making a copy - ";

object\* res2;

if (id == 1) {

res2 = new Integer;

}

else if (id == 2) {

res2 = new Date;

}

res2->makeCopy(res);

cout << "Copy - " << res2->uploadInString() << endl;

cout << "Write obj mode: ";

cin >> objMode;

}

if (objMode == 34) {

cout << "Write the same class object for union: ";

object\* sum;

if (id == 1) {

sum = new Integer;

cin >> \*(Integer\*)sum;

cout << "Union is - ";

object\* re = new Integer;

re->unionObj(sum, res);

cout << re->uploadInString() << endl;

}

else if (id == 2) {

sum = new Date;

cin >> \*(Date\*)sum;

cout << "Union is - ";

object\* re = new Date;

re->unionObj(sum, res);

cout << re->uploadInString() << endl;

}

cout << "Write obj mode: ";

cin >> objMode;

}

if (objMode == 35) {

cout << "Write the same class object for comparing: ";

object\* r;

if (id == 1) {

r = new Integer;

cin >> \*(Integer\*)r;

}

else if (id == 2) {

r = new Date;

cin >> \*(Date\*)r;

}

int cmp = res->equals(res, r);

if (cmp > 0) cout << res->uploadInString() << " > " << r->uploadInString() << endl;

else if (cmp < 0) cout << res->uploadInString() << " < " << r->uploadInString() << endl;

else cout << res->uploadInString() << " == " << r->uploadInString() << endl;

cout << "Write obj mode: ";

cin >> objMode;

}

if (objMode == 36) {

cout << "deleting...\n";

bt->del(res);

cout << "Object was deleted.\n";

cout << "Write obj mode: ";

cin >> objMode;

}

if ((objMode > 36 || objMode < 31) && objMode != -1) {

cout << "Unknown command. Try again.\nWrite obj mode: ";

cin >> objMode;

}

}

}

else cout << "Cannot find this data.\n";

cout << "Write command's id: ";

}

// вывод структуры в консоль

if (mode == 4) {

bt->show();

cout << "Write command's id: ";

}

// запись структуры в бинарный файл

if (mode == 5) {

bt->wrBin();

cout << "Write command's id: ";

}

// чтение структуры из бинарного файла

if (mode == 6) {

bt = new Btree;

bt->rBin();

cout << "Write command's id: ";

}

// размер дерева

if (mode == 7) {

cout << "Binary tree size - " << bt->size() << endl;

cout << "Write command's id: ";

}

// выход

if (mode == 8) {

cout << "Exiting...";

exit(-1);

}

// список команд

if (mode == 0) {

cout << commands;

}

if (mode > 8 || mode < 0) {

cout << "Unknown command. Try again.\nWrite command's id: ";

}

cin >> mode;

}

return 0;

}

* Test.cpp

#include "btree.hpp"

#include "Integer.hpp"

#include "Date.hpp"

#include <ctime>

void test\_rand(int n) {

Btree bt;

// массив значений

int\* m = new int[n];

srand(clock());

int random, pog10 = n + n%10;

for (int i = 0; i < n; i++) {

// для уникальности генерируемых значений

for (;;) {

bool good = true;

random = 1 + rand()%n;

for (int j = 0; j < n; j++) {

if (random == m[j]) {

good = false;

break;

}

}

if (good) break;

}

m[i] = random;

// вставка

bt.add(new Integer(m[i]));

}

cout << "items count: " << bt.size() << endl;

double S = 0;

for (int i = 0; i < n/2; i++) {

S += bt.search2(m[i]);

}

cout << "theoretical count: 1.39 \* log2(n) = " << 1.39\*(log((double) n) / log(2.0)) << endl;

cout << "practical count: " << S/(n/2) << endl;

delete[] m;

}

int main() {

time\_t start, end;

for (int i = 1000; i <= 10000; i+=1000) {

start = clock();

test\_rand(i);

end = clock();

cout << "Elapsed time - " << (double) (end-start) / CLOCKS\_PER\_SEC << " seconds." << endl;

}

return 0;

}

* BinaryTree.dat

