**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ФКТИ**

Курсовая РАБОТА

**по дисциплине «Программирование»**

Тема: Симуляция бизнес-процесса сортировки входных данных

|  |  |  |
| --- | --- | --- |
| Студент гр. 4335 |  | Иванов Г.Д. |
| Преподаватель |  | Калмычков В.А. |

Санкт-Петербург

2025

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Иванов Г.Д. | | |
| Группа 4335 | | |
| Тема работы: Разработка программы для обработки (сортировки) входных данных небольшой фирмы по производству бриллиантов про поставленные с рудников алмазы | | |
| Исходные данные:  Все типы данных собственные. Реализация на основе односвязного списка и строки с маркером. Программа должна давать возможность исполнять процесс сортировки несколько раз. Входные и выходные данные в файлах, должны вестись логи на всех этапах обработки | | |
| Содержание пояснительной записки: Содержание, Введение, Внешний формат хранения данных; Внутренний формат хранения данных; Описание использованных типов, функций и файлов; Интерфейс пользователя; Описание алгоритма обработки; Текст программы; Пример работы программы; Вывод; | | |
| Предполагаемый объем пояснительной записки: не менее 40 страниц | | |
| Дата выдачи задания: 24.01.2025 | | |
| Дата сдачи реферата: 27.06.2025 | | |
| Дата защиты реферата: 27.06.2025 | | |
| Студент |  | Иванов Г.Д. |
| Преподаватель |  | Калмычков В.А. |

**Аннотация**

В курсовой работе разработана программа на языке C++ для моделирования процесса сортировки алмазов по весу с учётом их происхождения. Входные данные поступают из текстового файла, содержащего сведения о компаниях-рудниках и массе добытых алмазов. Алмазы группируются на заданное число весовых категорий, каждая из которых сортируется по возрастанию массы. Реализация выполнена с использованием собственных односвязных списков и строковых типов, без применения стандартных контейнеров STL. В проекте предусмотрена генерация входных данных, логирование, проверка корректности сортировки. Результаты записываются в выходной файл. Работа демонстрирует навыки проектирования алгоритмов, работы с динамической памятью и пользовательскими структурами данных.

**Summary**

This course project presents a C++ program designed to simulate the sorting process of diamonds by weight, while preserving information about their origin. Input data is read from a text file containing mining company names and diamond weights. The diamonds are grouped into a specified number of weight categories, each sorted in ascending order. The implementation uses custom singly linked lists and string types, avoiding the use of STL containers. The project includes input data generation, logging, and validation of sorting correctness. The results are written to an output file. The work demonstrates skills in algorithm design, dynamic memory management, and user-defined data structures.

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 5 |
| 1. | Внешний формат хранения данных | 6 |
| 1.1. | Входные данные | 6 |
| 1.2. | Выходные данные | 6 |
| 1.2.1 | Протокольный файл diamonds.log | 6 |
| 1.2.2 | Файл с результатом обработки | 7 |
| 2. | Внутренний формат хранения данных | 8 |
| 2.1. | Система хранения на основе списков | 8 |
| 2.2. | Текстовый формат данных | 9 |
| 2.3 | Структура diamond\_source | 10 |
| 2.4 | Структура diamond\_mine | 10 |
| 2.5 | Структура diamond\_source\_group | 10 |
| 3. | Описание использованных типов, функций, файлов | 11 |
| 4. | Описание пользовательского интерфейса | 14 |
| 5. | Описание алгоритма обработки | 16 |
| 6. | Текст программы | 20 |
| 7. | Пример работы программы | 41 |
|  | Заключение | 46 |

**введение**

Современное программирование требует умения эффективно работать с динамическими структурами данных, особенно в задачах, связанных с моделированием и обработкой информации. Одной из таких задач является сортировка и классификация объектов по заданным признакам.

Цель данной курсовой работы — разработать программу на языке C++, моделирующую процесс сортировки алмазов по их весу с сохранением информации о месте их добычи. В работе требуется реализовать все действия с применением собственных типов данных на базе односвязных списков без использования стандартных контейнеров STL.

Задачи, решаемые в ходе выполнения курсовой:

* реализация структуры хранения алмазов с учётом их характеристик;
* организация ввода данных из файла;
* распределение алмазов по весовым группам;
* сортировка внутри каждой группы;
* вывод результатов в файл;
* проверка корректности выполнения.

Выбранный подход позволяет продемонстрировать навыки работы с динамической памятью, построения пользовательских структур и алгоритмов обработки данных.

**1. внешний формат хранения данных**

* 1. **Входные данные**

Входные данные представлены в виде одного текстового файла, название указывается по ходу исполнения программы. Данные представлены в виде списка названий компаний по добыче алмазов и чисел, указывающих размер алмазов. Каждый новый элемент указывается на новой строке.

<Название компании-поставщика 1>:

<размер алмаза 1>

<размер алмаза 2>

…

<размер алмаза N>

<Название компании-поставщика 2>:

<размер алмаза N + 1>

<размер алмаза N + 2>

…

<размер алмаза N + M>

* 1. **Выходные данные**

В процессе работы программы формируются два ключевых выходных файла, каждый из которых служит для разных целей:

**1.2.1 Протокольный файл diamonds.log**

Этот файл содержит детальный журнал всех операций и событий в системе с указанием категории записи и имени функции, откуда эта запись произвелась.

[категория (info/warning/error)]: [имя функции] – [сообщение]

info: [void diamond::diamond\_engine::read\_source(const nx::forward\_string &)] - starting...  
info: [void diamond::diamond\_engine::read\_source(const nx::forward\_string &)] - attempting to read file *diamonds.in*info: [void diamond::diamond\_engine::read\_source(const nx::forward\_string &)] - file 'diamonds.in' is open for reading  
info: [void diamond::diamond\_engine::read\_source(const nx::forward\_string &)] - clearing existing sources (*0* elements)  
info: [void diamond::diamond\_engine::read\_source(const nx::forward\_string &)] - reading file 'diamonds.in'...

**1.2.2** **Файл с результатом обработки**

Этот файл содержит результат работы программы в виде списка из N групп сортировки (N указывается по ходу исполнения программы, по умолчанию 10), в каждой из которых перечисляются размеры отсортированных алмазов с указанием фирмы, откуда они поступили.

Group <номер группы> (<кол-во элементов> elements):

<размер алмаза 1> --- <компания поставщик 1>

<размер алмаза 2> --- <компания поставщик 2>

…

**2. внутренний формат хранения данных**

**2.1. Система хранения на основе списков**

Во всей программе используется единственный шаблонный тип данных nx::forward\_list реализующий односвязный список. Каждый экземпляр этого типа данных представляет собой односвязную цепочку узлов, где:

* **Узел списка (\_forward\_list\_node) содержит:**
  + Данные (\_data) - объект произвольного типа
  + Указатель (\_next) на следующий узел
* **Контейнер списка (forward\_list) содержит:**
  + Головной указатель (\_root) на первый узел
  + Счетчик количества элементов в списке (\_size)
* **Итератор списка ([\_const]\_forward\_list\_iterator)**
  + Указатель на текущий узел списка (\_node)

Ключевые особенности хранения данных:

* Каждый узел создается в куче через new \_forward\_list\_node<T>. Память освобождается в clear() с помощью delete
* Узлы связаны через указатель \_next. Последний узел имеет \_next = nullptr. Новые элементы добавляются в произвольное место в списке с помощью итератора

Изображение выглядит как текст, чек, Шрифт, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

**2.2. Текстовый формат данных**

Текстовые данные хранятся в структуре marker\_string, представляющей собой реализацию строки с маркером. Структура содержит два поля:

* char \* \_data – указатель на начало строки
* char \_marker – символ, означающий конец строки

Ключевые особенности хранения данных:

* Память под хранение строки выделяется динамически при помощи оператора new []
* Строка всегда заканчивается символом, содержащимся в поле \_marker
* При удалении объекта структуры, память очищается при помощи оператора delete []

**2.3. Структура diamond\_source**

Представляет собой единицу информации о компании-поставщике алмаза и о размере алмаза. Состоит из двух полей:

* marker\_string company\_name – Название фирмы
* double size – размер алмаза

Полная информация о всех поставщиках и алмазах формируется в процессе чтения входного файла данных и содержится в структуре nx::forward\_list<diamond\_source>

**2.4 Структура diamond\_mine**

Представляет собой структуру для хранения информации о поставщике алмазов. Состоит из двух полей:

* marker\_string mine\_name – название поставщика
* nx::forward\_list<double> diamons – размеры поставленных алмазов

**2.5 Структура diamond\_source\_group**

Представляет собой структуру для хранения группы поставленных фирме алмазов, разбитых относительно веса (размера) по группам. Состоит из трех полей:

* double lower\_bound – минимальный вес азмаза в группе
* double upper\_bound – максимальный вес алмаза в группе
* nx::forward\_list<diamond\_source> - список поставленных алмазов

**3. ОПИСАНИЕ ИСПОЛЬЗОВАННЫХ ТИПОВ, ФУНКЦИЙ И ФАЙЛОВ**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя модуля (файла)** | **Имя структуры, класса или функции/метода** | **Назначение** | **Входные параметры** | **Возвращаемое значение** |
| marker\_string.h | marker\_string() | Конструктор по умолчанию. Инициализирует указатель \_data как nullptr и маркер \_marker символом '$' | - | - |
| marker\_string.h | marker\_string(const char\* cstr) | Конструктор из C-строки. Выделяет память, копирует строку, добавляет маркер в конец. | const char \* - указатель на  с-style строку | - |
| marker\_string.h | marker\_string(const std::string& str) | Конструктор из std::string. Аналогичен предыдущему, но принимает объект std::string. | const std::stirng & - строка по ссылке | - |
| marker\_string.h | marker\_string(const marker\_string& other) | Конструктор копирования. Создаёт глубокую копию другого объекта marker\_string. | Const marker\_string & - объект для копирования | - |
| marker\_string.h | ~marker\_string() | Деструктор. Освобождает динамически выделенную память. | - | - |
| marker\_string.h | marker\_string::size() const | Возвращает длину строки (до маркера). | - | int |
| marker\_string.h | marker\_string::clear() | Освобождает память, выделенную под \_data. | - | void |
| marker\_string.h | marker\_string::data() | Возвращает указатель на данные. | - | char\* |
| marker\_string.h | marker\_string::marker() | Возвращает символ-маркер окончания строки. | - | char |
| marker\_string.h | operator << | Заносит строку в структуре marker\_string в поток вывода | std::basic\_ostream<char>&,  const marker\_string & | std::basic\_ostream<char> & |
| main.cpp | main | Точка входа в программу | - | int |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| marker\_string.h | marker\_string::split | Разделить строку на список строк по символу разделителю | char sep, bool keep\_empty\_parts | forward\_list<marker\_string> |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| diamond\_engine.h | diamond\_engine::exec | Запуск программы и основной цикл взаимодействия с пользователем | — | int |
| diamond\_engine.h | diamond\_engine::create\_source | Генерация входного файла с заданным количеством записей | const marker\_string&, uint32\_t | bool |
| diamond\_engine.h | diamond\_engine::read\_source | Чтение алмазов из файла и определение диапазона их размеров | const marker \_string& | void |
| diamond\_engine.h | diamond\_engine::split\_source\_into\_groups | Разделение алмазов на N весовых групп | unsigned | void |
| diamond\_engine.h | diamond\_engine::sort\_sources\_groups | Сортировка алмазов внутри каждой группы | — | void |
| diamond\_engine.h | diamond\_engine::print\_sources\_groups | Вывод отсортированных групп на экран | — | void |
| diamond\_engine.h | diamond\_engine::write\_groups | Запись групп алмазов в выходной файл | const marker \_string& | void |
| diamond\_engine.h | diamond\_engine::check\_groups | Проверка корректности сортировки внутри групп | — | void |
| nx\_forward\_list.h | forward\_list::begin | Получить итератор на начало списка | — | iterator |
| nx\_forward\_list.h | forward\_list::end | Получить итератор на конец списка | — | iterator |
| nx\_forward\_list.h | forward\_list::size | Получить количество элементов в списке | — | size\_type |
| nx\_forward\_list.h | forward\_list::clear | Удалить все элементы списка | — | void |
| nx\_forward\_list.h | forward\_list::push\_front | Добавить элемент в начало списка | const value\_type& | void |
| nx\_forward\_list.h | forward\_list::push\_back | Добавить элемент в конец списка | const value\_type& | void |
| nx\_forward\_list.h | forward\_list::pop\_front | Удалить элемент из начала списка | — | void |
| nx\_forward\_list.h | forward\_list::pop\_back | Удалить элемент из конца списка | — | void |
| nx\_forward\_list.h | forward\_list::insert | Вставить элемент в заданную позицию | const iterator&, const value\_type& | iterator |
| nx\_forward\_list.h | forward\_list::remove | Удалить элемент по позиции | const iterator& | iterator |
| nx\_forward\_list.h | forward\_list::at | Доступ к элементу по индексу | size\_type | reference |
| nx\_forward\_list.h | forward\_list::splice\_back | Добавить в конец все элементы другого списка | const forward\_list& | void |
| nx\_forward\_list.h | forward\_list::sort | Сортировка элементов списка | compare\_func (по умолчанию std::less) | void |

**4. ОПИСАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА ПРОГРАММЫ**

Программа представляет собой консольное приложение с пошаговым взаимодействием с пользователем. Интерфейс разработан таким образом, чтобы пользователь мог последовательно выполнить все необходимые этапы обработки данных: от генерации или загрузки алмазов до их сортировки и сохранения результатов.

**Основные элементы интерфейса:  
  
1. Главное меню (отображается при запуске):**  1. Запустить процесс сортировки  
 2. Указать файл входных данных  
 3. Указать файл выходных данных  
 4. Указать количество групп сортировки N  
 5. Сгенерировать файл со случайными входными данными  
 0. Завершить программу

**2. Выбор входного файла (опция 2):**

- Программа предлагает ввести имя нового входного файла  
 - Введённое имя сохраняется и используется в дальнейшем **3. Выбор выходного файла (опция 3):**

- Пользователь вводит имя файла, в который будет записан результат работы программы

- Введённое имя сохраняется и используется в дальнейшем

**4. Указание количества групп сортировки (опция 4):** - Программа запрашивает число N (от 1 до 1000)  
 - Значение проверяется на корректность, иначе запрашивается повторно **5. Генерация случайных данных (опция 5):** - Запрашивается имя нового файла и количество записей  
 - Создаётся файл с группированными по рудникам алмазами случайной массы **6. Запуск сортировки (опция 1):** - Проверяется наличие файла с исходными данными  
 - Алмазы загружаются из файла, распределяются по группам в соответствии с массой  
 - Каждая группа сортируется по возрастанию массы  
 - Выполняется проверка корректности сортировки  
 - Результаты записываются в указанный выходной файл **7. Обратная связь и логирование:** - Все ключевые действия сопровождаются текстовыми сообщениями в консоли  
 - Для каждой операции (чтение, группировка, сортировка, запись) выводится подтверждение выполнения  
 - В случае ошибок (например, отсутствие файла) программа сообщает об этом и предлагает решениеИнтерфейс не требует специальной подготовки пользователя, понятен на интуитивном уровне и позволяет контролировать весь процесс работы программы

**5. ОПИСАНИЕ АЛГОРИТМА ОБРАБОТКИ**

* Инициализация структур данных:  
  - Создание списка алмазов (forward\_list<diamond\_source>)  
  - Подготовка списка групп (forward\_list<forward\_list<diamond\_source>>)  
  - Установка значений по умолчанию для количества групп, входного и выходного файлов
* Генерация входных данных:  
  - Ввод количества алмазов  
  - Случайная генерация названий рудников и размеров алмазов  
  - Запись сгенерированных данных в файл
* Загрузка данных из файла:  
  - Построчное считывание: заголовок с названием рудника и список размеров  
  - Формирование записей diamond\_mine  
  - Формирование записей diamond\_source из списка diamond\_mine

- Определение минимального и максимального веса для последующей нормализации

* Разделение на группы:  
  - Расчёт границ групп по формуле (min + i \* step)  
  - Назначение каждой записи в соответствующую группу
* Сортировка внутри групп:  
  - Применение встроенного метода sort() с пользовательским компаратором  
  - Сортировка по возрастанию веса внутри каждой группы
* Проверка корректности сортировки:  
  - Проход по каждой группе  
  - Сравнение весов соседних элементов на неубывание
* Запись результата в файл:  
  - Последовательный проход по группам  
  - Запись всех элементов с указанием группы и происхождения
* Обработка ошибок:  
  - Проверка корректности ввода файлов и числа групп  
  - Проверка на успешность открытия файлов  
  - Защита от выхода за пределы при обращении к группам
* Завершение работы:  
  - Освобождение всех выделенных структур  
  - Вывод заключительного сообщения  
  - Возврат в главное меню либо завершение программы

Изображение выглядит как диаграмма, линия, белый, зарисовка

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 1. Блок-схема алгоритма работы метода main\_loop

Изображение выглядит как диаграмма, зарисовка, План, Технический чертеж

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 2. Блок-схема алгоритма работы метода read\_source

**Изображение выглядит как диаграмма, линия, План, зарисовка

Содержимое, созданное искусственным интеллектом, может быть неверным.**

Рисунок 3. Блок-схема алгоритма работы метода write\_groups

**6. текст программы**

**marker\_string.h**

#ifndef MARKER\_STRING\_H

#define MARKER\_STRING\_H

#include <string>

#include <iostream>

#include "nx\_forward\_list.h"

class marker\_string {

char \* \_data;

char \_marker;

void m\_init() {

\_data = nullptr;

\_marker = '$';

}

public:

marker\_string() : \_data{ nullptr }, \_marker{ '$' } {}

marker\_string(const char \* cstr, int l = -1) : marker\_string() {

auto len = (l == -1) ? strlen(cstr) : l;

\_data = new char [len + 1];

memcpy(\_data, cstr, len);

\_data[len] = \_marker;

}

marker\_string(const std::string & str) : marker\_string() {

auto len = str.size();

\_data = new char [len + 1];

memcpy(\_data, str.c\_str(), len);

\_data[len] = \_marker;

}

marker\_string(const marker\_string & other) :

\_marker{other.\_marker}

{

int len = other.size();

\_data = new char [len + 1];

\_data[len] = \_marker;

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

\_data[i] = other.\_data[i];

}

~marker\_string() {

clear();

}

marker\_string & operator=(const marker\_string & other) {

clear();

\_marker = other.\_marker;

auto len = other.size();

\_data = new char [len + 1];

memcpy(\_data, other.data(), len);

\_data[len] = \_marker;

return \*this;

}

int size () const {

if (\_data == nullptr) return 0;

int out = 0;

auto ch = data();

while (\*ch++ != \_marker) out += 1;

return out;

}

void clear () { if (\_data) delete [] \_data; }

char \* data () const { return \_data; }

char marker () const { return \_marker; }

bool empty () const { return (\_data == nullptr) || (\_data[0] == \_marker); }

const char \* c\_str() const {

static char \* buf = (char \*) malloc(2048);

auto s = size();

memcpy((void \*)buf, data(), s);

\*(buf + s) = 0;

return const\_cast<const char \*>(buf);

}

nx::forward\_list<marker\_string> split (char sep, bool keep\_empty\_parts = false) const {

nx::forward\_list<marker\_string> out;

char \* p = \_data;

char \* q = \_data;

while (\*p != \_marker) {

if (\*p == sep) {

marker\_string ms {q, (int)(p - q)};

q = p + 1;

if (!ms.empty() || keep\_empty\_parts) out.push\_front(ms);

}

p++;

}

marker\_string ms {q, (int)(p - q)};

if (!ms.empty() || keep\_empty\_parts) out.push\_front(ms);

out.reverse();

return out;

}

bool operator == (const marker\_string & other) const {

int this\_size = size();

int other\_size = other.size();

if (this\_size != other\_size) return false;

char \* this\_p = data();

char \* other\_p = other.data();

while (\*this\_p != \_marker) {

if (\*this\_p != \*other\_p) return false;

this\_p++;

other\_p++;

}

return true;

}

};

inline std::basic\_ostream<char> & operator << (std::basic\_ostream<char> & stream, const marker\_string & str) {

auto ch = str.data();

while (\*ch != str.marker())

stream << \*ch++;

return stream;

}

marker\_string remove\_special\_symbols\_from\_word\_n(const marker\_string & in, int n);

namespace nx {

template <typename ...Args>

marker\_string format(const char \* c\_str, Args... args) {

char buf [2048];

int size = ::snprintf(buf, 2048, c\_str, args...);

return buf;

}

}

#endif //MARKER\_STRING\_H

**marker\_string.cpp**

#include "marker\_string.h"

marker\_string remove\_special\_symbols\_from\_word\_n(const marker\_string & in, int n) {

marker\_string out {in};

int i = 0;

bool current\_is\_word = (\*in.data() != ' ');

int word\_count = current\_is\_word ? 1 : 0;

for (char \* ch = in.data(); \*ch != in.marker(); ch++) {

if (\*ch != ' ' && (current\_is\_word == false)) {

word\_count++;

current\_is\_word = true;

}

else if (\*ch == ' ') current\_is\_word = false;

else current\_is\_word = true;

if (\*ch == ' ') {

out.data()[i++] = \*ch;

continue;

}

if (word\_count == n) {

if ((\*ch >= 'a' && \*ch < 'z') || (\*ch >= 'A' && \*ch <= 'Z'))

out.data()[i++] = \*ch;

}

else

out.data()[i++] = \*ch;

}

out.data()[i] = out.marker();

return out;

}

logger.h

#ifndef LOGGER\_H

#define LOGGER\_H

// #include "nx\_string\_extension.h"

#include "marker\_string.h"

#include <fstream>

namespace nx {

enum class log\_level {

ll\_debug = 1,

ll\_info,

ll\_warn,

ll\_error,

ll\_fatal,

};

struct log\_context {

const char \* file;

const int line;

const char \* function;

};

struct log\_message {

log\_level level;

marker\_string msg;

log\_context context;

};

struct log\_category {

char \* name;

FILE \* file;

log\_level upper\_level {log\_level::ll\_fatal};

log\_level lower\_level {log\_level::ll\_info};

};

class logger {

static logger \* \_instance;

// nx::forward\_list<log\_category> \_categories;

std::ofstream \_log\_stream;

~logger();

public:

static logger & instance ();

static void free\_instance ();

marker\_string create\_message(log\_level level, const marker\_string & msg, log\_context context);

void write\_message(const marker\_string & message);

public:

void add\_log(const log\_message & msg);

void set\_log\_file(const marker\_string & file);

};

#define log\_info(message) nx::logger::instance().add\_log({nx::log\_level::ll\_info, message, {\_\_FILE\_\_, \_\_LINE\_\_, \_\_PRETTY\_FUNCTION\_\_}})

#define log(message) nx::logger::instance().add\_log({nx::log\_level::ll\_debug, message,{\_\_FILE\_\_, \_\_LINE\_\_, \_\_PRETTY\_FUNCTION\_\_}})

#define log\_warn(message) nx::logger::instance().add\_log({nx::log\_level::ll\_warn, message, {\_\_FILE\_\_, \_\_LINE\_\_, \_\_PRETTY\_FUNCTION\_\_}})

#define log\_error(message) nx::logger::instance().add\_log({nx::log\_level::ll\_error, message, {\_\_FILE\_\_, \_\_LINE\_\_, \_\_PRETTY\_FUNCTION\_\_}})

#define log\_fatal(message) nx::logger::instance().add\_log({nx::log\_level::ll\_fatal, message, {\_\_FILE\_\_, \_\_LINE\_\_, \_\_PRETTY\_FUNCTION\_\_}})

} // nx

#endif //LOGGER\_H

**logger.cpp**

#include "logger.h"

#include <mutex>

namespace nx {

std::mutex logger\_mutex;

logger\* logger::\_instance { nullptr };

logger::~logger() {

\_log\_stream.close();

}

logger & logger::instance() {

if (!\_instance)

\_instance = new logger();

return \*\_instance;

}

void logger::free\_instance() {

if (\_instance)

delete \_instance;

}

marker\_string logger::create\_message(log\_level level, const marker\_string &msg, log\_context context) {

std::string out;

switch (level) {

case log\_level::ll\_warn:

out += "warn: ";

break;

case log\_level::ll\_error:

out += "error: ";

break;

case log\_level::ll\_info:

out += "info: ";

break;

case log\_level::ll\_fatal:

out += "fatal error: ";

break;

default:

break;

}

out += "[";

out += context.function;

out += "] - ";

out += msg.c\_str();

out += "\n";

return marker\_string(out);

}

void logger::write\_message(const marker\_string &message) {

std::lock\_guard<std::mutex> lock(logger\_mutex);

\_log\_stream << message << std::flush;

}

void logger::add\_log(const log\_message &msg) {

auto str = create\_message(msg.level, msg.msg, msg.context);

write\_message(str);

}

void logger::set\_log\_file(const marker\_string & file) {

\_log\_stream.close();

\_log\_stream.open(file.c\_str(), std::ios::app);

}

} // nx

**nx\_forward\_list.h**

#ifndef NX\_FORWARD\_LIST\_H

#define NX\_FORWARD\_LIST\_H

#include <ranges>

namespace nx {

namespace \_detail

{

struct \_forward\_list\_node\_base {

\_forward\_list\_node\_base \* \_next;

explicit \_forward\_list\_node\_base (\_forward\_list\_node\_base \* p\_next = nullptr) : \_next{p\_next} {};

};

template <typename \_Tp>

struct \_forward\_list\_node : public \_forward\_list\_node\_base

{

using base = \_forward\_list\_node\_base;

using value\_type = \_Tp;

value\_type \_data;

explicit \_forward\_list\_node (const value\_type & p\_data) :

\_data {p\_data},

\_forward\_list\_node\_base{nullptr}

{};

explicit \_forward\_list\_node (const \_Tp & p\_data, base \* p\_next) :

\_data {p\_data},

\_forward\_list\_node\_base{p\_next}

{};

base \* m\_base () const { return static\_cast<base \*> (this); }

value\_type & m\_data () { return \_data; }

value\_type const & m\_data () const { return \_data; }

value\_type \* m\_data\_ptr () { return &\_data; }

value\_type \* m\_data\_ptr () const { return &\_data; }

};

template <typename \_Tp>

struct \_forward\_list\_iterator {

using value\_type = \_Tp;

using base\_type = \_forward\_list\_node\_base;

using node\_type = \_forward\_list\_node<value\_type>;

using self\_type = \_forward\_list\_iterator<value\_type>;

base\_type \* \_node;

explicit \_forward\_list\_iterator (base\_type \* p\_node) : \_node(p\_node) {}

const self\_type & m\_const\_cast () const { return \*this; }

value\_type \* operator -> () { return static\_cast<node\_type \*>(\_node->\_next)->m\_data\_ptr(); }

value\_type & operator \* () { return static\_cast<node\_type \*>(\_node->\_next)->m\_data(); }

self\_type & operator ++ () { \_node = \_node->\_next; return \*this; }

self\_type operator ++ (int) { auto tmp = \*this; \_node = \_node->\_next; return tmp; }

bool operator == (const self\_type & p\_other) { return \_node == p\_other.\_node || (at\_end() && p\_other.at\_end()); }

bool operator != (const self\_type & other) { return !operator == (other); }

bool at\_end () const { return !valid() || (\_node->\_next == nullptr); } // || (\_node->\_next->\_next == nullptr); }

bool valid () const { return \_node != nullptr; }

};

template <typename \_Tp>

struct \_const\_forward\_list\_iterator {

using value\_type = \_Tp;

using base\_type = \_forward\_list\_node\_base;

using node\_type = \_forward\_list\_node<value\_type>;

using self\_type = \_const\_forward\_list\_iterator<value\_type>;

using iterator\_type = \_forward\_list\_iterator<value\_type>;

const base\_type \* \_node;

explicit \_const\_forward\_list\_iterator (const base\_type \* p\_node) : \_node(p\_node) {}

\_const\_forward\_list\_iterator (const iterator\_type & p\_other) : \_node(p\_other.\_node) {}

value\_type \* operator -> () const { return static\_cast<node\_type \*>(\_node->\_next)->m\_data\_ptr(); }

value\_type & operator \* () const { return static\_cast<node\_type \*>(\_node->\_next)->m\_data(); }

self\_type & operator ++ () { \_node = \_node->\_next; return \*this; }

self\_type operator ++ (int) { auto tmp = \*this; \_node = \_node->\_next; return tmp; }

bool operator == (const self\_type & p\_other) { return \_node == p\_other.\_node || (at\_end() && p\_other.at\_end());; }

bool operator != (const self\_type & other) { return !operator == (other); }

bool at\_end () const { return !valid() || (\_node->\_next == nullptr); }// || (\_node->\_next->\_next == nullptr); }

bool valid () const { return \_node != nullptr; }

};

};

template <typename \_Tp = void>

class forward\_list {

public:

using value\_type = \_Tp;

using reference = value\_type&;

using pointer = value\_type\*;

using const\_reference = const value\_type&;

using const\_pointer = const value\_type\*;

using iterator = \_detail::\_forward\_list\_iterator<value\_type>;

using const\_iterator = \_detail::\_const\_forward\_list\_iterator<value\_type>;

using list\_node = \_detail::\_forward\_list\_node<value\_type>;

using self\_type = forward\_list<value\_type>;

#ifdef \_\_aarch64\_\_

using size\_type = uint64\_t;

#else

using size\_type = size\_t;

#endif

private:

\_detail::\_forward\_list\_node\_base \_root;

size\_type \_size;

void m\_init() {

\_root.\_next = nullptr;

\_size = 0;

}

const\_iterator m\_last\_node () {

auto it = begin();

for (; !it.at\_end(); ++it) {}

return it;

}

public:

forward\_list () { m\_init(); }

~forward\_list () { clear(); }

forward\_list (::std::initializer\_list<value\_type> p\_list) {

m\_init();

auto back\_it = begin();

for (auto it = p\_list.begin(); it != p\_list.end(); ++it, ++back\_it)

insert(back\_it, \*it);

}

forward\_list (const self\_type & p\_other) {

m\_init();

splice\_back(p\_other);

}

forward\_list (self\_type && p\_other) {

\_root = p\_other.\_root;

p\_other.m\_init();

}

forward\_list & operator = (const self\_type & p\_other) {

clear();

m\_init();

splice\_back(p\_other);

return \*this;

}

forward\_list & operator = (self\_type && p\_other) {

clear();

m\_init();

\_root = p\_other.\_root;

\_size = p\_other.\_size;

p\_other.m\_init();

return \*this;

}

iterator begin () { return iterator(&\_root); }

const\_iterator begin () const { return const\_iterator(&\_root); }

iterator end () { return iterator(nullptr); }

const\_iterator end () const { return iterator(nullptr); }

iterator true\_end () {auto it = begin(); while (!it.at\_end()) {++it;} return it; }

const\_iterator true\_end () const {auto it = begin(); while (!it.at\_end()) {++it;} return it; }

void clear() {

auto cur\_ptr = &\_root;

auto next\_ptr = \_root.\_next;

while (next\_ptr && (cur\_ptr != next\_ptr)) {

cur\_ptr = next\_ptr;

next\_ptr = next\_ptr->\_next;

delete cur\_ptr;

}

m\_init();

}

size\_type size () const {

return \_size;

}

bool empty () const { return \_size == 0; }

iterator insert (const iterator & pos, const value\_type & p\_data) {

if (!pos.valid()) return iterator(nullptr);

auto new\_node = new list\_node(p\_data, pos.\_node->\_next);

pos.\_node->\_next = new\_node;

\_size += 1;

return iterator(new\_node);

}

iterator remove (const iterator & pos) {

if (!pos.valid()) { return iterator(nullptr); }

if (pos.at\_end()) { pop\_back(); return end(); }

auto node = pos.\_node->\_next;

pos.\_node->\_next = node->\_next;

delete node;

return iterator(pos.\_node);

}

void push\_front(const value\_type & p\_data) {

auto new\_node = new list\_node(p\_data, \_root.\_next);

\_root.\_next = new\_node;

\_size += 1;

}

void pop\_front () {

if (\_root.\_next == nullptr) return;

auto tmp = \_root.\_next;

\_root.\_next = \_root.\_next->\_next;

delete tmp;

\_size -= 1;

}

void push\_back(const value\_type & data) {

auto it = begin();

for (; !it.at\_end(); ++it) {}

insert(it, data);

}

void pop\_back() {

if (empty()) return;

auto tmp = &\_root;

while (tmp->\_next->\_next != nullptr) { tmp = tmp->\_next; }

delete tmp->\_next;

tmp->\_next = nullptr;

\_size -= 1;

}

void splice\_back (const self\_type & p\_other) {

auto back\_it = iterator(&\_root);

for (; !back\_it.at\_end(); ++back\_it) {}

auto other\_it = p\_other.begin();

while (other\_it != p\_other.end()) {

insert(back\_it, \*other\_it);

++back\_it;

++other\_it;

}

}

reference at (size\_type p\_pos) {

// static\_assert(p\_pos < \_size && p\_pos > 0);

size\_type pos { 0 };

for (auto it = begin(); it != end(); ++it, pos++)

if (pos == p\_pos)

return \*it;

}

const\_reference at (size\_type p\_pos) const {

// static\_assert(p\_pos < \_size && p\_pos > 0);

size\_type pos { 0 };

for (auto it = begin(); it != end(); ++it, pos++)

if (pos == p\_pos)

return \*it;

}

template <typename compare\_func = std::less<\_Tp>>

void sort (compare\_func comp = std::less<\_Tp>())

{

if (size() <= 1) return;

value\_type pivot = \*begin();

pop\_front();

self\_type less;

self\_type greater;

for (const auto& item : \*this) {

if (comp(item, pivot)) {

less.push\_back(item);

} else {

greater.push\_back(item);

}

}

clear();

less.sort(comp);

greater.sort(comp);

splice\_back(less);

push\_back(pivot);

splice\_back(greater);

}

self\_type reversed () const {

forward\_list out;

for (auto it = begin(); it != end(); ++it)

out.push\_front(\*it);

return out;

}

self\_type & reverse () {

return \*this = reversed();

}

};

} // nx

#endif //NX\_FORWARD\_LIST\_H

**diamond\_engine.h**

#ifndef DIAMOND\_ENGINE\_H  
#define DIAMOND\_ENGINE\_H  
  
#include "nx\_forward\_list.h"  
#include "marker\_string.h"  
  
///  
/// Добавить: 1) чтение списка названий компаний из файла  
/// 2) проверку на корректность наличия данных (?)  
  
namespace diamond {  
 struct diamond\_mine  
 {  
 marker\_string mine\_name { "unknown" };  
 nx::forward\_list<double> diamonds;  
 };  
 struct diamond\_source  
 {  
 marker\_string company\_name { "unknown" };  
 double size { 0 };  
 };  
 struct diamond\_source\_group  
 {  
 double lower\_bound { 0 };  
 double upper\_bound { 0 };  
 nx::forward\_list<diamond\_source> sources;  
 };  
  
 class diamond\_engine {  
 nx::forward\_list <diamond\_mine> \_mines {};  
 nx::forward\_list <diamond\_source> \_sources {};  
 double \_max\_size {0}, \_min\_size {0};  
 nx::forward\_list <diamond\_source\_group> \_groups {};  
  
 void init ();  
 void main\_loop ();  
 public:  
 int exec ();  
  
 bool create\_source (const marker\_string & p\_filename, uint32\_t p\_entries);  
 void read\_source (const marker\_string & p\_filename);  
 void merge\_source ();  
 void split\_source\_into\_groups(unsigned p\_group\_count);  
 void sort\_sources\_groups();  
 void print\_sources\_groups();  
 void write\_groups(const marker\_string & p\_filename);  
 void check\_groups ();  
 };  
  
}  
  
#endif //DIAMOND\_ENGINE\_H

**diamond\_engine.cpp**

#include "diamond\_engine.h"  
#include <fstream>  
#include <iomanip>  
#include "logger.h"  
  
namespace diamond {  
 const nx::forward\_list<marker\_string> diamond\_companies {  
 "Crystal Edge",  
 "Diamond Stream",  
 "Shining Point",  
 "Bright Core",  
 "Pure Gems",  
 "RoughSpark Inc",  
 "Glint Mines",  
 "Stone Nest",  
 "Glitter Depth",  
 "Facet Group",  
 "Radiant Drill",  
 "Northern Vein",  
 "Hidden Jewel",  
 "White Glare",  
 "Raw Cutters",  
 "Gleam Mining",  
 "Gem Ridge",  
 "Hard Light",  
 "Diamond Pulse",  
 "Clarity Diggers",  
 "Stoneburst Ltd",  
 "Lustre Lab",  
 "Gemwork Co",  
 "Facet Brothers",  
 "TrueCarbon Ltd",  
 "Crystalroot Inc",  
 "Bright Hollow",  
 "Drillstream",  
 "Roughstone Co",  
 "Deep Spark",  
 "Blunt Crystal",  
 "ShineValley",  
 "Minebeam",  
 "Prism Pickers",  
 "Rocklight Ltd",  
 "Diamond Creek",  
 "Brightstone Group",  
 "Glint & Rock",  
 "Raw Glory",  
 "Gem Splitters",  
 "Veinflow Corp",  
 "Lumen Core",  
 "Cold Spark",  
 "Crystal Mill",  
 "White Edge Co",  
 "True Gemline",  
 "Stonecore Inc",  
 "Brilliance Basin",  
 "Deep Carbon",  
 "ShinyWorks",  
 "FacetField",  
 "Carbon Spark",  
 "Cutter Ridge",  
 "Diamond Drop",  
 "Bright Quarry",  
 "Gleam Zone",  
 "RoughLight Inc",  
 "Core Radiance",  
 "Gemfront Co",  
 "Stone Flash",  
 "Sharp Diamond",  
 "Clarity Forge",  
 "Miners Bright",  
 "Crystal Works",  
 "Veinlight Inc",  
 "True Stone",  
 "Hardstone Gems",  
 "Depthshine Ltd",  
 "Gem Vault",  
 "Luster Flow",  
 "Hardcore Gems",  
 "White Carbon",  
 "Deepcore Ltd",  
 "Quartzlight",  
 "Diamond Drift",  
 "Gem Vein",  
 "Rock Prism",  
 "Facet Blaze",  
 "Roughshine Co",  
 "Brightforge",  
 "Miner's Prism",  
 "Shineplate",  
 "Stonecut Co",  
 "Hard Edge Ltd",  
 "Luminous Depths",  
 "Drillforge Inc",  
 "Gemtrace Ltd",  
 "Brightseed Mining",  
 "Lustercut",  
 "Polish Rock",  
 "Crystal Prospect",  
 "Hardshine Works",  
 "Clarity Mine",  
 "Deep Hollow Gems",  
 "Rockbeam Ltd",  
 "Rawstone Group",  
 "Stoneveil Co",  
 "Sparklight Inc",  
 "Facet Horizon",  
 "Bright Path Gems",  
 "Deep Sparkle",  
 "Shine Hill Mining",  
 "TrueGlint"  
};  
  
 marker\_string get\_random\_distributor ()  
 {  
 auto index = rand() % diamond\_companies.size();  
 return diamond\_companies.at(index);  
 }  
  
 double get\_random\_diamond\_size (double p\_lower, double p\_upper)  
 {  
 unsigned int unsign\_rand = (unsigned int)(rand());  
 double zero\_to\_one = (unsign\_rand \* 1.0) / RAND\_MAX;  
 return zero\_to\_one \* (p\_upper - p\_lower) + p\_lower;  
 }  
  
 bool get\_yes\_or\_with\_deault (std::basic\_istream<char> & stream, bool default\_yes = true)  
 {  
 std::string answer;  
 stream >> answer;  
 if (answer.empty())  
 return default\_yes;  
 return answer == "y" || answer == "Y" || answer == "yes" || answer == "Yes";  
 }  
  
 void diamond\_engine::init()  
 {  
 srand((unsigned)(time(nullptr)));  
 auto & logger = nx::logger::instance();  
 logger.set\_log\_file("diamond.log");  
  
 log\_info("\n\nDiamonds - startup initialisation");  
  
 marker\_string \_credits =  
 "======================================================\n"  
 "Курсовая работа по дисциплине \"Программирование\"\n"  
 "Автор: Иванов Григорий Денисович\n"  
 "Группа: 4335 (подгруппа 1)\n"  
 "======================================================\n";  
  
 std::cout << \_credits << std::endl;  
 std::cout << "--- Сортировщик бриллиантов ---\n" << std::endl;  
  
 std::cout << "Суть задания: " << std::endl;  
 std::cout << "\tНебогатая фирма по производству бриллиантов получает алмазы с нескольких рудников,\n"  
 "\tс каждого рудника свой контейнер с разным количеством алмазов. \n"  
 "\tТребуется разделить поступившие алмазы по их весу (размеру) на N групп, \n"  
 "\tсохранив информацию о месте их добычи, при этом рассортировав их в пределах каждой группы\n"  
 << std::endl;  
  
 std::cout << "Входные данные: " << std::endl;  
 std::cout << "\tN - количество групп отсортированных бриллиантов (по умолчанию: 10)" << std::endl;  
 std::cout << "\tinput\_file - файл с входными данными (по умолчанию: 'diamonds.in')" << std::endl;  
 std::cout << "\toutput\_file - файл с выходными данными (по умолчанию: 'diamonds.out')" << std::endl;  
 }  
  
 void diamond\_engine::main\_loop()  
 {  
 log\_info("Entering main loop");  
 marker\_string input\_file = "diamonds.in";  
 marker\_string output\_file = "diamonds.out";  
 int N = 10;  
 bool running = true;  
  
 auto print\_menu = [&] () {  
 log\_info("printing menu for the user");  
 std::cout << "\nВыберите действие: \n"  
 "\t1. Запустить процесс сортировки\n"  
 "\t2. Указать файл входных данных (текущее значение: '" << input\_file << "')\n"  
 "\t3. Указать файл выходных данных (текущее значение: '" << output\_file << "')\n"  
 "\t4. Указать количество групп сортировки N (текущее значение: " << N << ")\n"  
 "\t5. Сгенерировать файл со случайными входными данными\n"  
 "\t0. Звершить программу"  
 << std::endl;  
 };  
 auto generate\_new\_input\_file = [this] (const marker\_string & p\_filename) -> bool {  
 log\_info(nx::format("generating new input file '%s'", p\_filename.c\_str()));  
 if (std::filesystem::exists(p\_filename.c\_str())) {  
 std::cout << "Файл с указанным именем уже существует. Вы хотите переписать его? [y/N]:";  
 bool yes = get\_yes\_or\_with\_deault(std::cin, false);  
 if (!yes)  
 return false;  
 log\_warn("an existing file will be overwritten with new data");  
 }  
 uint64\_t entries;  
 std::cout << "Введите количество записей в файле: ";  
 std::cin >> entries;  
 this->create\_source(p\_filename, entries);  
 // this->create\_random\_source(entries);  
 // this->write\_source(p\_filename);  
 this->\_sources.clear();  
 std::cout << "Данные записаны в файл " << p\_filename << std::endl;  
 return true;  
 };  
 auto check\_input\_file\_exists = [generate\_new\_input\_file] (const marker\_string & p\_filename) -> bool {  
 if (std::filesystem::exists(p\_filename.c\_str()))  
 return true;  
 log\_info (nx::format("file '%s' does not exist. will ask the user to create one", p\_filename.c\_str()));  
 std::cout << "Файл '" << p\_filename << "' не найден. Вы хотите сгенерировать новый? [Y/n]:";  
 bool yes = get\_yes\_or\_with\_deault(std::cin, true);  
 if (!yes)  
 return false;  
 generate\_new\_input\_file(p\_filename);  
 return yes;  
 };  
  
 while (running)  
 {  
 log\_info ("waiting for user input");  
 int choise = 0;  
 print\_menu();  
 std::cout << "> ";  
 std::cin >> choise;  
 switch (choise) {  
 case 0: {  
 log\_info("exising main loop");  
 running = false;  
 break;  
 }  
 case 2: {  
 log\_info("set input file name option chosen");  
 std::cout << "Введите новое значение: ";  
 std::string std\_input\_file;  
 std::cin >> std\_input\_file;  
 input\_file = marker\_string(std\_input\_file);  
 log\_info (nx::format("input file name is set to '%s'", input\_file.c\_str()));  
 break;  
 }  
 case 3: {  
 log\_info("set output file name option chosen");  
 std::cout << "Введите новое значение: ";  
 std::string std\_output\_file;  
 std::cin >> std\_output\_file;  
 output\_file = marker\_string(std\_output\_file);  
 break;  
 log\_info(nx::format("output file name is set to '%s'", output\_file.c\_str()));  
 }  
 case 4: {  
 log\_info(marker\_string("set group count option chosen"));  
 do {  
 std::cout << "Введите новое значение (от 1 до 1000): ";  
 std::cin >> N;  
 if (N <= 0 || N > 1000)  
 {  
 log\_warn(nx::format("invalid group count set: %i", N));  
 std::cout << "Ошибка: значение должно находиться в интервале от 1 до 1000" << std::endl;  
 continue;  
 }  
 break;  
 }while (true);  
 log\_info(nx::format("group count is set to %i", N));  
 break;  
 }  
 case 5: {  
 log\_info("generate new data file option is chosen");  
 std::string filename;  
 std::cout << "Введите название: ";  
 std::cin >> filename;  
 generate\_new\_input\_file(filename);  
 break;  
 }  
 case 1: {  
 log\_info("run sorting algorithm option chosen");  
 check\_input\_file\_exists(input\_file);  
 read\_source(input\_file);  
 merge\_source();  
 split\_source\_into\_groups(N);  
 sort\_sources\_groups();  
 check\_groups();  
 write\_groups(output\_file);  
 break;  
 }  
 default: {  
 std::cout << "Некорректный номер команды" << std::endl;  
 log\_info ("invalid command entered");  
 break;  
 }  
 }  
 }  
 }  
  
 void diamond\_engine::read\_source(const marker\_string&p\_filename)  
 {  
 log\_info ("starting...");  
 log\_info (nx::format("attempting to read file %s", p\_filename.c\_str()));  
 std::ifstream fin(p\_filename.c\_str());  
 if (!fin)  
 {  
 std::cerr << "File " << p\_filename << " cannot be opened for read" << std::endl;  
 log\_error(nx::format("failure opening file '%s'. aborting", p\_filename.c\_str()));  
 return;  
 }  
 log\_info (nx::format("file '%s' is open for reading", p\_filename.c\_str()));  
 // log\_info (nx::format("clearing existing sources (%i elements)", \_sources.size()));  
 // \_sources.clear();  
 // \_max\_size = 0;  
 // \_min\_size = 100000;  
  
 log\_info (nx::format("clearing existing mines (%i elements)", \_mines.size()));  
 \_mines.clear();  
  
  
 std::string line;  
 std::string cur\_company { "unknown" };  
 int counter = 0;  
 double cur\_size = 0;  
 size\_t pos = 0;  
  
 while (std::getline(fin, line) && ++counter) {  
 if (line.empty()) {  
 log\_warn(nx::format("parsing warning: line %i is empty", counter));  
 continue;  
 }  
 if (line.ends\_with(':')) {  
 cur\_company = line.substr(0, line.size() - 1);  
 log\_info(nx::format("reading data for %s company...", cur\_company.c\_str()));  
 \_mines.push\_front({cur\_company});  
 continue;  
 }  
 cur\_size = std::stod(line.c\_str(), &pos);  
 if (pos == 0) {  
 log\_warn(nx::format("cannot process line %i: %s", counter, line.c\_str()));  
 continue;  
 }  
 // \_max\_size = cur\_size > \_max\_size ? cur\_size : \_max\_size;  
 // \_min\_size = cur\_size < \_min\_size ? cur\_size : \_min\_size;  
 // \_sources.push\_front({cur\_company.c\_str(), cur\_size});  
 \_mines.at(0).diamonds.push\_front(cur\_size);  
 }  
  
 std::cout << "[diamond\_engine][read\_source] : read sources for "  
 << \_mines.size() << " companies from '" << p\_filename << "'" << std::endl;  
 log\_info(nx::format("read %i sources from file '%s'", \_mines.size(), p\_filename.c\_str()));  
 log\_info("finished successfully");  
 }  
  
 void diamond\_engine::merge\_source() {  
 log\_info ("starting...");  
 log\_info (nx::format("clearing existing sources (%i elements)", \_sources.size()));  
 \_sources.clear();  
  
 \_max\_size = 0;  
 \_min\_size = 100000;  
  
 for (auto & mine: \_mines) {  
 for (auto diamond: mine.diamonds) {  
 \_max\_size = diamond > \_max\_size ? diamond : \_max\_size;  
 \_min\_size = diamond < \_min\_size ? diamond : \_min\_size;  
 \_sources.push\_front({mine.mine\_name, diamond});  
 }  
 }  
  
 log\_info(nx::format("determined maximum and minimum sizes: max=%f, min=%f", \_max\_size, \_min\_size));  
 log\_info(nx::format("merged %i entries in total", \_sources.size()));  
 log\_info("finished successfully");  
 }  
  
 void diamond\_engine::split\_source\_into\_groups(unsigned p\_group\_count)  
 {  
 log\_info ("starting...");  
 log\_info ("clearing existing diamond groups");  
 \_groups.clear();  
 log\_info (nx::format("allocating memory for %i diamond groups...", p\_group\_count));  
 for (int i = 0; i < p\_group\_count; i++) {  
 double lower\_bound = (\_max\_size - \_min\_size)/p\_group\_count \* i + \_min\_size;  
 double upper\_bound = (\_max\_size - \_min\_size)/p\_group\_count \* (i + 1) + \_min\_size;  
 \_groups.push\_front({lower\_bound, upper\_bound});  
 }  
 \_groups.reverse();  
 log\_info (nx::format("begin going through sources"));  
 for (auto it = \_sources.begin(); it != \_sources.end(); ++it)  
 {  
 int s = it->size;  
 int a = (\_max\_size - \_min\_size);  
 int b = (int)((s - \_min\_size)) \* p\_group\_count;  
 int group\_index = (b - 1)/a;  
 \_groups.at(group\_index).sources.push\_front(\*it);  
 }  
 std::cout << "[diamond\_engine][split\_source\_into\_groups] : split sources into "  
 << p\_group\_count << " groups" << std::endl;  
 for (int i = 0; i < \_groups.size(); ++i) {  
 std::cout << "\tgroup " << i + 1 << ": " << \_groups.at(i).sources.size() << " elements" << std::endl;  
 }  
 log\_info (nx::format("successfully divided sources into %i groups", p\_group\_count));  
 }  
  
 void diamond\_engine::sort\_sources\_groups()  
 {  
 log\_info("starting...");  
 log\_info("going through diamond groups...");  
 for (int i = 0; i < \_groups.size(); ++i)  
 {  
 std::cout << "[diamond\_engine][sort\_sources\_groups] : sorting elements in group "  
 << i + 1 << "... " << std::flush;  
 log\_info(nx::format("begin sorting group %i (%i elements)", i + 1, \_groups.at(i).sources.size()));  
 \_groups.at(i).sources.sort([] (const diamond\_source & first, const diamond\_source & second) -> bool {  
 return first.size < second.size;  
 });  
 std::cout << "finished" << std::endl;  
 log\_info(nx::format("finished sorting group %i", i + 1));  
 }  
 std::cout << "[diamond\_engine][sort\_sources\_groups] : finished sorting" << std::endl;  
 log\_info("finished successfully");  
 }  
  
 void diamond\_engine::print\_sources\_groups()  
 {  
 log\_info ("starting...");  
 log\_info ("going through diamond groups...");  
 int i = 0;  
 for (auto it = \_groups.begin(); it != \_groups.end(); ++it, ++i)  
 {  
 auto & group = \*it;;  
 std::cout << "Group " << i + 1 << " (" << group.lower\_bound << "-" << group.upper\_bound << ")"  
 << "[" << group.sources.size() << " elements]:" << std::endl;  
 for (auto src: group.sources)  
 {  
 std::cout << "\t" << std::setw(30) << std::setfill('-') << src.company\_name << " : " << src.size << std::endl;  
 }  
 log\_info(nx::format("printed group %1", i + 1));  
 }  
 log\_info ("finished successfully");  
 }  
  
 void diamond\_engine::write\_groups(const marker\_string &p\_filename) {  
 log\_info("starting...");  
 log\_info(nx::format("opening output file '%s'", p\_filename.c\_str()));  
 std::ofstream fout(p\_filename.c\_str());  
 if (!fout)  
 {  
 std::cerr << "[diamond\_engine][write\_groups] : failed to open output file '" << p\_filename << "'" << std::endl;  
 log\_error(nx::format("failure while opening output file '%s'. aborting", p\_filename.c\_str()));  
 return;  
 }  
  
 log\_info("begin going through initial data mines...");  
 fout << "Initial data: " << std::endl;  
 for (auto & mine: \_mines) {  
 fout << mine.mine\_name << ": " << std::endl;  
 for (auto diamond: mine.diamonds) {  
 fout << "\t" << diamond << std::endl;  
 }  
 }  
 log\_info("finished writing initial data into result file");  
  
 fout << std::endl << "Result: " << std::endl;  
 log\_info("begin going though diamond groups...");  
 int i = 0;  
 for (int i = 0; i < \_groups.size(); ++i)  
 {  
 auto & group = \_groups.at(i);  
 int gr\_size = group.sources.size();  
 // fout << "Group " << i + 1 << " (" << group.sources.size()  
 // << (group.size() == 1 ? " element" : " elements") << "):" << std::endl;  
 fout << "Group " << i + 1 << " (" << group.lower\_bound << "-" << group.upper\_bound << ")"  
 << "[" << group.sources.size() << " elements]:" << std::endl;  
 log\_info(nx::format("writing group %i (%i elements)...", i + 1, group.sources.size()));  
 for (auto src: group.sources)  
 {  
 fout << "\t" << src.size << " --- " << src.company\_name << std::endl;  
 }  
 std::cout << "[diamond\_engine][write\_groups] : wrote group " << i + 1 << " to output file" << std::endl << std::flush;  
 log\_info(nx::format("finished writing group %i", i + 1));  
 }  
 log\_info("closing output file");  
 fout.close();  
 std::cout << "[diamond\_engine][write\_groups] : finished writing to output file '" << p\_filename << "'" << std::endl;  
 log\_info(nx::format("finished successfully (wrote %i groups in total)", \_groups.size()));  
 }  
  
 void diamond\_engine::check\_groups()  
 {  
 log\_info("starting...");  
 std::cout << "[diamond\_engine][check\_groups] : begin checking sorted groups" << std::endl;  
 int counter = 1;  
 bool all\_ok = true;  
 log\_info("begin going through diamond groups...");  
 for (auto gr\_it = \_groups.begin(); gr\_it != \_groups.end(); ++gr\_it, counter++)  
 {  
 std::cout << "[diamond\_engine][check\_groups] : checking group " << counter << "... ";  
 log\_info(nx::format("checking group %i", counter));  
 // double min = gr\_it->begin()->size;  
 double min = 0;  
 bool group\_ok = true;  
 for (auto it = gr\_it->sources.begin(); it != gr\_it->sources.end(); ++it)  
 {  
 if (min > it->size)  
 {  
 std::cerr << "[diamond\_engine][check\_groups] : group " << counter << " failed the check" << std::endl;  
 log\_error(nx::format("group %i failed the check. Elements are not sorted in ascending order", counter));  
 group\_ok = false;  
 all\_ok = false;  
 break;  
 }  
 min = it->size;  
 }  
 if (group\_ok) { std::cout << "passed" << std::endl; }  
 else std::cout << "failed" << std::endl;  
 }  
 if (all\_ok) {  
 std::cout << "[diamond\_engine][check\_groups] : check finished successfully" << std::endl;  
 log\_info("finished successfully");  
 }  
 else {  
 std::cout << "[diamond\_engine][check\_groups] : check failed" << std::endl;  
 log\_error("check failed");  
 }  
 }  
  
  
 int diamond\_engine::exec()  
 {  
 init();  
 main\_loop();  
 log\_info("exiting with code 0");  
 return 0;  
 }  
  
 bool diamond\_engine::create\_source(const marker\_string &p\_filename, uint32\_t p\_entries) {  
 std::cout << "[diamond\_engine][create\_source] : starting..." << std::endl;  
 log\_info("starting...");  
 log\_info("allocating temporal memory for storing data...");  
 nx::forward\_list<nx::forward\_list<double>> diamonds\_in\_mines;  
 // diamonds\_in\_mines.resize(diamond\_companies.size());  
 for (int i = 0; i < diamond\_companies.size(); ++i) {diamonds\_in\_mines.push\_front({}); }  
 uint32\_t mine\_id = 0;  
 double size;  
 for (int i = 0; i < p\_entries; ++i) {  
 mine\_id = rand() % diamonds\_in\_mines.size();  
 size = get\_random\_diamond\_size(10, 1000);  
 diamonds\_in\_mines.at(mine\_id).push\_front(size);  
 }  
 uint64\_t total\_memory = diamonds\_in\_mines.size() \* sizeof(nx::forward\_list<double>); // size of lists  
 total\_memory += p\_entries \* sizeof(double); // size of data in lists  
 log\_info(nx::format("total memory allocated = %i byte", total\_memory));  
 log\_info(nx::format("attemting to write data to file '%s'...", p\_filename.c\_str()));  
 std::ofstream fout(p\_filename.c\_str());  
 if (!fout) {  
 std::cerr << "[diamond\_engine][create\_source] : failed to open file '" << p\_filename << "'" << std::endl;  
 log\_error(nx::format("failed to open file '%s' for write. aborting", p\_filename.c\_str()));  
 return false;  
 }  
 log\_info("going through data...");  
 int i = 0;  
 for (auto it = diamonds\_in\_mines.begin(); it != diamonds\_in\_mines.end(); ++it, ++i) {  
 if (it->empty())  
 continue;  
 fout << diamond\_companies.at(i) << ":" << std::endl;  
 for (auto s: \*it)  
 fout << s << std::endl;  
 log\_info(nx::format("wrote entries for '%s'", diamond\_companies.at(i).c\_str()));  
 }  
 fout.close();  
 std::cout << "[diamond\_engine][create\_source] : wrote " << p\_entries << " entries into '" << p\_filename << "'" << std::endl;  
 log\_info(nx::format("successfully generated source file (%i entries written)", p\_entries));  
 return true;  
 }  
} // diamond

**main.cpp**

#include "diamond\_engine.h"

int main(int argc, char \* argv [])

{

diamond::diamond\_engine engine;

return engine.exec();

}

**7. ПРИМЕР РАБОТЫ ПРОГРАММЫ**

**Сценарий работы:**

1. Пользователь создает новый файл с исходными данными с названием test.in
2. Пользователь меняет количество групп сортировки на 6
3. Пользователь изменяет название входного файла
4. Пользователь изменяет название выходного файла
5. Пользователь запускает сортировку
6. Пользователь завершает программу

**Консоль:**

======================================================

Лабораторная работа по дисциплине "Программирование"

Автор: Иванов Григорий Денисович

Группа: 4335 (подгруппа 1)

======================================================

--- Сортировщик бриллиантов ---

Суть задания:

Небогатая фирма по производству бриллиантов получает алмазы с нескольких рудников,

с каждого рудника свой контейнер с разным количеством алмазов.

Требуется разделить поступившие алмазы по их весу (размеру) на N групп,

сохранив информацию о месте их добычи, при этом рассортировав их в пределах каждой группы

Входные данные:

N - количество групп отсортированных бриллиантов (по умолчанию: 10)

input\_file - файл с входными данными (по умолчанию: 'diamonds.in')

output\_file - файл с выходными данными (по умолчанию: 'diamonds.out')

Выберите действие:

1. Запустить процесс сортировки

2. Указать файл входных данных (текущее значение: 'diamonds.in')

3. Указать файл выходных данных (текущее значение: 'diamonds.out')

4. Указать количество групп сортировки N (текущее значение: 10)

5. Сгенерировать файл со случайными входными данными

0. Звершить программу

> 5

Введите название: test.in

Файл с указанным именем уже существует. Вы хотите переписать его? [y/N]:y

Введите количество записей в файле: 15

[diamond\_engine][create\_source] : starting...

[diamond\_engine][create\_source] : wrote 15 entries into 'test.in'

Данные записаны в файл test.in

Выберите действие:

1. Запустить процесс сортировки

2. Указать файл входных данных (текущее значение: 'diamonds.in')

3. Указать файл выходных данных (текущее значение: 'diamonds.out')

4. Указать количество групп сортировки N (текущее значение: 10)

5. Сгенерировать файл со случайными входными данными

0. Звершить программу

> 4

Введите новое значение (от 1 до 1000): 6

Выберите действие:

1. Запустить процесс сортировки

2. Указать файл входных данных (текущее значение: 'diamonds.in')

3. Указать файл выходных данных (текущее значение: 'diamonds.out')

4. Указать количество групп сортировки N (текущее значение: 6)

5. Сгенерировать файл со случайными входными данными

0. Звершить программу

> 2

Введите новое значение: test.in

Выберите действие:

1. Запустить процесс сортировки

2. Указать файл входных данных (текущее значение: 'test.in')

3. Указать файл выходных данных (текущее значение: 'diamonds.out')

4. Указать количество групп сортировки N (текущее значение: 6)

5. Сгенерировать файл со случайными входными данными

0. Звершить программу

> 3

Введите новое значение: test.out

Выберите действие:

1. Запустить процесс сортировки

2. Указать файл входных данных (текущее значение: 'test.in')

3. Указать файл выходных данных (текущее значение: 'test.out')

4. Указать количество групп сортировки N (текущее значение: 6)

5. Сгенерировать файл со случайными входными данными

0. Звершить программу

> 1

[diamond\_engine][read\_source] : read 15 source entries from 'test.in'

[diamond\_engine][split\_source\_into\_groups] : split sources into 6 groups

group 1: 3 elements

group 2: 2 elements

group 3: 3 elements

group 4: 2 elements

group 5: 2 elements

group 6: 3 elements

[diamond\_engine][sort\_sources\_groups] : sorting elements in group 1... finished

[diamond\_engine][sort\_sources\_groups] : sorting elements in group 2... finished

[diamond\_engine][sort\_sources\_groups] : sorting elements in group 3... finished

[diamond\_engine][sort\_sources\_groups] : sorting elements in group 4... finished

[diamond\_engine][sort\_sources\_groups] : sorting elements in group 5... finished

[diamond\_engine][sort\_sources\_groups] : sorting elements in group 6... finished

[diamond\_engine][sort\_sources\_groups] : finished sorting

[diamond\_engine][check\_groups] : begin checking sorted groups

[diamond\_engine][check\_groups] : checking group 1... passed

[diamond\_engine][check\_groups] : checking group 2... passed

[diamond\_engine][check\_groups] : checking group 3... passed

[diamond\_engine][check\_groups] : checking group 4... passed

[diamond\_engine][check\_groups] : checking group 5... passed

[diamond\_engine][check\_groups] : checking group 6... passed

[diamond\_engine][check\_groups] : check finished successfully

[diamond\_engine][write\_groups] : wrote group 1 to output file

[diamond\_engine][write\_groups] : wrote group 2 to output file

[diamond\_engine][write\_groups] : wrote group 3 to output file

[diamond\_engine][write\_groups] : wrote group 4 to output file

[diamond\_engine][write\_groups] : wrote group 5 to output file

[diamond\_engine][write\_groups] : wrote group 6 to output file

[diamond\_engine][write\_groups] : finished writing to output file 'test.out'

Выберите действие:

1. Запустить процесс сортировки

2. Указать файл входных данных (текущее значение: 'test.in')

3. Указать файл выходных данных (текущее значение: 'test.out')

4. Указать количество групп сортировки N (текущее значение: 6)

5. Сгенерировать файл со случайными входными данными

0. Звершить программу

> 0

Process finished with exit code 0

**Протокольный файл diamond.log:**

Diamonds - startup initialisation

info: [void diamond::diamond\_engine::main\_loop()] - Entering main loop

info: [void diamond::diamond\_engine::main\_loop()] - waiting for user input

info: [auto diamond::diamond\_engine::main\_loop()::(anonymous class)::operator()() const] - printing menu for the user

info: [void diamond::diamond\_engine::main\_loop()] - generate new data file option is chosen

info: [auto diamond::diamond\_engine::main\_loop()::(anonymous class)::operator()(const marker\_string &) const] - generating new input file 'test.in'

warn: [auto diamond::diamond\_engine::main\_loop()::(anonymous class)::operator()(const marker\_string &) const] - an existing file will be overwritten with new data

info: [bool diamond::diamond\_engine::create\_source(const marker\_string &, uint32\_t)] - starting...

info: [bool diamond::diamond\_engine::create\_source(const marker\_string &, uint32\_t)] - allocating temporal memory for storing data...

info: [bool diamond::diamond\_engine::create\_source(const marker\_string &, uint32\_t)] - total memory allocated = 1768 byte

info: [bool diamond::diamond\_engine::create\_source(const marker\_string &, uint32\_t)] - attemting to write data to file 'test.in'...

info: [bool diamond::diamond\_engine::create\_source(const marker\_string &, uint32\_t)] - going through data...

info: [bool diamond::diamond\_engine::create\_source(const marker\_string &, uint32\_t)] - wrote entries for 'Glint Mines'

info: [bool diamond::diamond\_engine::create\_source(const marker\_string &, uint32\_t)] - wrote entries for 'Radiant Drill'

info: [bool diamond::diamond\_engine::create\_source(const marker\_string &, uint32\_t)] - wrote entries for 'Minebeam'

info: [bool diamond::diamond\_engine::create\_source(const marker\_string &, uint32\_t)] - wrote entries for 'Stonecore Inc'

info: [bool diamond::diamond\_engine::create\_source(const marker\_string &, uint32\_t)] - wrote entries for 'Carbon Spark'

info: [bool diamond::diamond\_engine::create\_source(const marker\_string &, uint32\_t)] - wrote entries for 'Core Radiance'

info: [bool diamond::diamond\_engine::create\_source(const marker\_string &, uint32\_t)] - wrote entries for 'Veinlight Inc'

info: [bool diamond::diamond\_engine::create\_source(const marker\_string &, uint32\_t)] - wrote entries for 'True Stone'

info: [bool diamond::diamond\_engine::create\_source(const marker\_string &, uint32\_t)] - wrote entries for 'Depthshine Ltd'

info: [bool diamond::diamond\_engine::create\_source(const marker\_string &, uint32\_t)] - wrote entries for 'Facet Blaze'

info: [bool diamond::diamond\_engine::create\_source(const marker\_string &, uint32\_t)] - wrote entries for 'Crystal Prospect'

info: [bool diamond::diamond\_engine::create\_source(const marker\_string &, uint32\_t)] - wrote entries for 'Rawstone Group'

info: [bool diamond::diamond\_engine::create\_source(const marker\_string &, uint32\_t)] - wrote entries for 'Stoneveil Co'

info: [bool diamond::diamond\_engine::create\_source(const marker\_string &, uint32\_t)] - wrote entries for 'Deep Sparkle'

info: [bool diamond::diamond\_engine::create\_source(const marker\_string &, uint32\_t)] - wrote entries for 'TrueGlint'

info: [bool diamond::diamond\_engine::create\_source(const marker\_string &, uint32\_t)] - successfully generated source file (15 entries written)

info: [void diamond::diamond\_engine::main\_loop()] - waiting for user input

info: [auto diamond::diamond\_engine::main\_loop()::(anonymous class)::operator()() const] - printing menu for the user

info: [void diamond::diamond\_engine::main\_loop()] - set group count option chosen

info: [void diamond::diamond\_engine::main\_loop()] - group count is set to 6

info: [void diamond::diamond\_engine::main\_loop()] - waiting for user input

info: [auto diamond::diamond\_engine::main\_loop()::(anonymous class)::operator()() const] - printing menu for the user

info: [void diamond::diamond\_engine::main\_loop()] - set input file name option chosen

info: [void diamond::diamond\_engine::main\_loop()] - input file name is set to 'test.in'

info: [void diamond::diamond\_engine::main\_loop()] - waiting for user input

info: [auto diamond::diamond\_engine::main\_loop()::(anonymous class)::operator()() const] - printing menu for the user

info: [void diamond::diamond\_engine::main\_loop()] - set output file name option chosen

info: [void diamond::diamond\_engine::main\_loop()] - waiting for user input

info: [auto diamond::diamond\_engine::main\_loop()::(anonymous class)::operator()() const] - printing menu for the user

info: [void diamond::diamond\_engine::main\_loop()] - run sorting algorithm option chosen

info: [void diamond::diamond\_engine::read\_source(const marker\_string &)] - starting...

info: [void diamond::diamond\_engine::read\_source(const marker\_string &)] - attempting to read file test.in

info: [void diamond::diamond\_engine::read\_source(const marker\_string &)] - file 'test.in' is open for reading

info: [void diamond::diamond\_engine::read\_source(const marker\_string &)] - clearing existing sources (0 elements)

info: [void diamond::diamond\_engine::read\_source(const marker\_string &)] - reading data for Glint Mines company...

info: [void diamond::diamond\_engine::read\_source(const marker\_string &)] - reading data for Radiant Drill company...

info: [void diamond::diamond\_engine::read\_source(const marker\_string &)] - reading data for Minebeam company...

info: [void diamond::diamond\_engine::read\_source(const marker\_string &)] - reading data for Stonecore Inc company...

info: [void diamond::diamond\_engine::read\_source(const marker\_string &)] - reading data for Carbon Spark company...

info: [void diamond::diamond\_engine::read\_source(const marker\_string &)] - reading data for Core Radiance company...

info: [void diamond::diamond\_engine::read\_source(const marker\_string &)] - reading data for Veinlight Inc company...

info: [void diamond::diamond\_engine::read\_source(const marker\_string &)] - reading data for True Stone company...

info: [void diamond::diamond\_engine::read\_source(const marker\_string &)] - reading data for Depthshine Ltd company...

info: [void diamond::diamond\_engine::read\_source(const marker\_string &)] - reading data for Facet Blaze company...

info: [void diamond::diamond\_engine::read\_source(const marker\_string &)] - reading data for Crystal Prospect company...

info: [void diamond::diamond\_engine::read\_source(const marker\_string &)] - reading data for Rawstone Group company...

info: [void diamond::diamond\_engine::read\_source(const marker\_string &)] - reading data for Stoneveil Co company...

info: [void diamond::diamond\_engine::read\_source(const marker\_string &)] - reading data for Deep Sparkle company...

info: [void diamond::diamond\_engine::read\_source(const marker\_string &)] - reading data for TrueGlint company...

info: [void diamond::diamond\_engine::read\_source(const marker\_string &)] - read 15 sources from file 'test.in'

info: [void diamond::diamond\_engine::read\_source(const marker\_string &)] - determined maximum and minimum sizes: max=858.354000, min=21.106600

info: [void diamond::diamond\_engine::read\_source(const marker\_string &)] - finished successfully

info: [void diamond::diamond\_engine::split\_source\_into\_groups(unsigned int)] - starting...

info: [void diamond::diamond\_engine::split\_source\_into\_groups(unsigned int)] - clearing existing diamond groups

info: [void diamond::diamond\_engine::split\_source\_into\_groups(unsigned int)] - allocating memory for 6 diamond groups...

info: [void diamond::diamond\_engine::split\_source\_into\_groups(unsigned int)] - begin going through sources

info: [void diamond::diamond\_engine::split\_source\_into\_groups(unsigned int)] - successfully divided sources into 6 groups

info: [void diamond::diamond\_engine::sort\_sources\_groups()] - starting...

info: [void diamond::diamond\_engine::sort\_sources\_groups()] - going through diamond groups...

info: [void diamond::diamond\_engine::sort\_sources\_groups()] - begin sorting group 1 (3 elements)

info: [void diamond::diamond\_engine::sort\_sources\_groups()] - finished sorting group 1

info: [void diamond::diamond\_engine::sort\_sources\_groups()] - begin sorting group 2 (2 elements)

info: [void diamond::diamond\_engine::sort\_sources\_groups()] - finished sorting group 2

info: [void diamond::diamond\_engine::sort\_sources\_groups()] - begin sorting group 3 (3 elements)

info: [void diamond::diamond\_engine::sort\_sources\_groups()] - finished sorting group 3

info: [void diamond::diamond\_engine::sort\_sources\_groups()] - begin sorting group 4 (2 elements)

info: [void diamond::diamond\_engine::sort\_sources\_groups()] - finished sorting group 4

info: [void diamond::diamond\_engine::sort\_sources\_groups()] - begin sorting group 5 (2 elements)

info: [void diamond::diamond\_engine::sort\_sources\_groups()] - finished sorting group 5

info: [void diamond::diamond\_engine::sort\_sources\_groups()] - begin sorting group 6 (3 elements)

info: [void diamond::diamond\_engine::sort\_sources\_groups()] - finished sorting group 6

info: [void diamond::diamond\_engine::sort\_sources\_groups()] - finished successfully

info: [void diamond::diamond\_engine::check\_groups()] - starting...

info: [void diamond::diamond\_engine::check\_groups()] - begin going through diamond groups...

info: [void diamond::diamond\_engine::check\_groups()] - checking group 1

info: [void diamond::diamond\_engine::check\_groups()] - checking group 2

info: [void diamond::diamond\_engine::check\_groups()] - checking group 3

info: [void diamond::diamond\_engine::check\_groups()] - checking group 4

info: [void diamond::diamond\_engine::check\_groups()] - checking group 5

info: [void diamond::diamond\_engine::check\_groups()] - checking group 6

info: [void diamond::diamond\_engine::check\_groups()] - finished successfully

info: [void diamond::diamond\_engine::write\_groups(const marker\_string &)] - starting...

info: [void diamond::diamond\_engine::write\_groups(const marker\_string &)] - opening output file 'test.out'

info: [void diamond::diamond\_engine::write\_groups(const marker\_string &)] - begin going though diamond groups...

info: [void diamond::diamond\_engine::write\_groups(const marker\_string &)] - writing group 1 (3 elements)...

info: [void diamond::diamond\_engine::write\_groups(const marker\_string &)] - finished writing group 1

info: [void diamond::diamond\_engine::write\_groups(const marker\_string &)] - writing group 2 (2 elements)...

info: [void diamond::diamond\_engine::write\_groups(const marker\_string &)] - finished writing group 2

info: [void diamond::diamond\_engine::write\_groups(const marker\_string &)] - writing group 3 (3 elements)...

info: [void diamond::diamond\_engine::write\_groups(const marker\_string &)] - finished writing group 3

info: [void diamond::diamond\_engine::write\_groups(const marker\_string &)] - writing group 4 (2 elements)...

info: [void diamond::diamond\_engine::write\_groups(const marker\_string &)] - finished writing group 4

info: [void diamond::diamond\_engine::write\_groups(const marker\_string &)] - writing group 5 (2 elements)...

info: [void diamond::diamond\_engine::write\_groups(const marker\_string &)] - finished writing group 5

info: [void diamond::diamond\_engine::write\_groups(const marker\_string &)] - writing group 6 (3 elements)...

info: [void diamond::diamond\_engine::write\_groups(const marker\_string &)] - finished writing group 6

info: [void diamond::diamond\_engine::write\_groups(const marker\_string &)] - closing output file

info: [void diamond::diamond\_engine::write\_groups(const marker\_string &)] - finished successfully (wrote 6 groups in total)

info: [void diamond::diamond\_engine::main\_loop()] - waiting for user input

info: [auto diamond::diamond\_engine::main\_loop()::(anonymous class)::operator()() const] - printing menu for the user

info: [void diamond::diamond\_engine::main\_loop()] - exising main loop

info: [int diamond::diamond\_engine::exec()] - exiting with code 0

**заключение**

В рамках курсовой работы была разработана программа на языке C++, моделирующая процесс сортировки алмазов по весовым группам с сохранением информации об их происхождении. Основной целью проекта было создание эффективного решения с применением пользовательских структур данных, реализованных без использования стандартных контейнеров STL.

В процессе работы были выполнены все этапы, предусмотренные заданием: генерация входных данных, считывание и обработка информации из файла, группировка алмазов по весу, сортировка внутри каждой группы, проверка корректности результатов и запись их в выходной файл. Также реализован интерактивный пользовательский интерфейс, обеспечивающий гибкое управление параметрами выполнения.

Программа полностью соответствует поставленной цели, демонстрирует навыки работы с динамической памятью, построения односвязных списков, обработки данных и построения алгоритмов. Полученные результаты подтверждают корректность и надёжность предложенного решения.