МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное
образовательное учреждение высшего образования
«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт (факультет) Информационных технологий

Кафедра Математическое и программное обеспечение ЭВМ

КУРСОВАЯ РАБОТА

по модулю «Объектно-ориентированное программирование»

на тему «Объектно-ориентированное программирование на языке С++»

Выполнил студент группы 1ПИб-02-2оп-22
группа
направления подготовки (специальности)
09.03.04 Программная инженерия
шифр, наименование
Зернов Владислав Александрович
фамилия, имя, отчество

Руководитель
Матевосян Р.А.
фамилия, имя, отчество
Ассистент
должность

Дата представления работы
«_____»_____20 ___ г.

Заключение о допуске к защите

Оценка _____, _____
количество баллов

Подпись преподавателя_____

Череповец, 2024

Анотация

Данная расчётно-пояснительная записка содержит 42 страницы машинописного текста, включая 22 рисунков, у приложения и 21 таблицу.

Оглавление

Введение

Объектно-ориентированное программирование (ООП) – это стиль программирования, который модулирует действия из реального мира, решаемой задачи. Это программирование, сфокусированное на данных. Основополагающей идеей объектно-ориентированного подхода является объединение данных и действий, производимых над этими данными, в единое целое, которое называется объектом.

Соответственно, объектно-ориентированная программа представляет совокупность объектов, взаимодействующих по средствам передачи сообщений для выполнения требуемых функций. Это называется объектная декомпозиция.

Инкапсуляция – это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе и скрыть детали реализации от пользователя.

Наследование – это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым или родительским. Новый класс – потомком, наследником или производным классом. Наследование позволяет выстраивать иерархию классов.

Полиморфизм — это свойство, которое позволяет одно и то же имя использовать для решения двух или более схожих, но технически разных задач. Целью полиморфизма, является использование одного имени для задания общих для класса действий. Выполнение каждого конкретного действия будет определяться типом данных. Полиморфизм позволяет манипулировать объектами различной степени сложности путём создания общего стандартного интерфейса с реализации похожих действий.

Абстрактный класс – это класс, который может использоваться лишь в качестве базового класса для производных классов. Класс является абстрактным, если он содержит хотя бы одну абстрактную функцию.

Основная часть

1. Объектно-ориентированный анализ предметной области

В ходе выполнения курсовой работы, необходимо разработать программный продукт, для работы с объектами предметной области воздушный транспорт. Необходимо разработать иерархию родственных типов, относящихся к данной области. Для достижения цели используется наследование.

Воздушный транспорт - это система перевозки пассажиров, грузов и почты по воздуху с использованием летательных аппаратов.

К воздушному транспорту относятся крылатые транспорты (WingyTransport), винтокрыльный транспорт (RototcraftTransport), шароподобный транспорт (BalloonTransport), парирующий транспорт (GlideTransport). Такие виды транспортов могут быть на двигателе (EngineAirTransport) и ручным (ManualAirTransport), что определяет возможность их передвижения.

Воздушный транспорт на двигателе(ВТД) – это транспорт, который имеет в себе двигатель и не может лететь без него. К такому транспорту относятся все те транспорты, которые используются сейчас чаще всего.

Воздушный ручной транспорт(ВРТ) – это транспорт, который не имеет в себе двигателя и летит на погодных явлениях(например, огонь или поток ветра). Такой вид транспорта в нашем веке используется не так активно, как ВТД, но благодаря ему воздушый транспорт развился до двигательного уровня.

Для хранения объектов классов используется шаблонный контейнер, исполненный в виде стека (Stack<T>). Для хранения указателей на воздушный транспорт используется шаблонный односвязный линейный список (List), наследуемый от односвязного линейного списка, хранящего указатели на void(ListVoid).

Все классы взаимосвязаны между собой(рис.1).

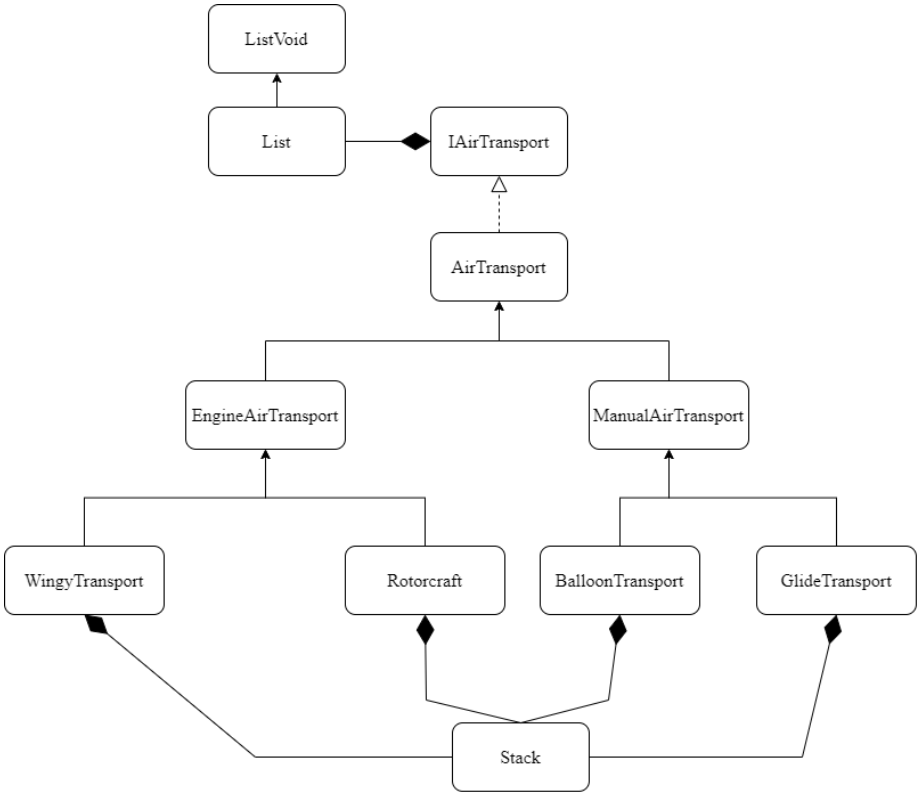

Рис.1. Контекстная диаграмма классов

2. Проектирование классов

Классы воздушного транспорта поддерживают функции вывода содержимого полей в консоль и в файл, также с содержимым полей в консоль и файл выводятся еще имена классов, к которым относятся выводимые объекты. Помимо этого, классы поддерживают функции получения и изменения значений полей объекта. О всех функциях, которые поддерживаются классами транспортов, информация находится в таблицах 1-8.

Таблица 1

Функции интерфейса IAirTransport

Функция	Назначение функции
virtual void Show()	Функция вывода объекта в консоль
virtual int GetCapacity()	Функция получение параметра вместимости по людям объекта
virtual double GetWeight()	Функция получения параметра веса в тонах объекта
virtual bool GetIsLanding()	Функция получение параметра нахождения на земле объекта
virtual bool GetIsFlying()	Функция получение параметра нахождения в воздухе объекта
virtual void Fly()	Функция изменения параметра нахождения в воздухе объекта
virtual void Land()	Функция изменения параметра нахождения на земле объекта
virtual void SaveFile(std::ofstream& f)	Функция сохранения объекта в файл

Таблица 2

Функции класса AirTransport

Функция	Назначение функции
AirTransport()	Конструктор по умолчанию
AirTransport(int capacity, double weight)	Конструктор с параметрами
void Show()	Функция вывода объекта в консоль
int GetCapacity()	Функция получение параметра вместимости по людям объекта
double GetWeight()	Функция получения параметра веса в тонах объекта
void SetCapacity(int capacity)	Функция изменения параметра вместимости по людям объекта
void SetWeight(double weight)	Функция изменения параметра веса в тонах объекта
bool GetIsLanding()	Функция получение параметра нахождения на земле объекта
bool GetIsFlying()	Функция получение параметра нахождения в воздухе объекта
void Fly()	Функция изменения параметра нахождения в воздухе объекта
void Land()	Функция изменения параметра нахождения на земле объекта
void SaveFile(std::ofstream& f)	Функция сохранения объекта в файл

Таблица 3

Функции класса EngineAirTransport

Функция	Назначение функции
EngineAirTransport ()	Конструктор по умолчанию
EngineAirTransport(double enginePower, double cargoCapacity, double capacity, int weight)	Конструктор с параметрами
void Show()	Функция вывода объекта в консоль
double GetCargoCapacity()	Функция получение параметра вместимости по грузу объекта
double GetEnginePower(double enginePower)	Функция получения параметра мощности двигателя объекта
void SetCargoCapacity(double cargoCapacity)	Функция изменения параметра вместимости по грузу объекта
void SetEnginePower(double enginePower)	Функция изменения параметра мощности двигателя объекта
void SaveFile(std::ofstream& f)	Функция сохранения объекта в файл

Таблица 4

Функции класса ManualAirTransport

Функция	Назначение функции
1	2
ManualAirTransport()	Конструктор по умолчанию
ManualAirTransport(int timeInAirSec, int capacity, double weight)	Конструктор с параметрами
void Show()	Функция вывода объекта в консоль
int GetTimeInAir()	Функция получение параметра нахождения в воздухе в секундах объекта

Продолжение табл. 4

1	2
void SetTimeInAir(int timeInAirSec)	Функция изменения параметра нахождения в воздухе в секундах объекта
void SaveFile(std::ofstream& f)	Функция сохранения объекта в файл

Таблица 5

Функции класса WingyTransport

Функция	Назначение функции
WingyTransport()	Конструктор по умолчанию
WingyTransport(int countEngines, double cargoCapacity, double enginePower, int capacity, double weight)	Конструктор с параметрами
void Show()	Функция вывода объекта в консоль
int GetCountEngines()	Функция получение параметра количества двигателей объекта
void SetCountEngines()	Функция изменения параметра количества двигателей объекта
void SaveFile(std::ofstream& f)	Функция сохранения объекта в файл

Таблица 6

Функции класса Rotorcraft

Функция	Назначение функции
1	2
Rotorcraft()	Конструктор по умолчанию
Rotorcraft(int countRotor, double cargoCapacity, double enginePower, int capacity, double weight)	Конструктор с параметрами

Продолжение табл. 6

1	2
void Show()	Функция вывода объекта в консоль
int GetCountRotor()	Функция получение параметра количества винтов объекта
void SetCountRotor()	Функция изменения параметра количества винтов объекта
void SaveFile(std::ofstream& f)	Функция сохранения объекта в файл

Таблица 7

Функции класса BallonTransport

Функция	Назначение функции
BallonTransport()	Конструктор по умолчанию
BallonTransport(int powerFire, int timeInAirSec, int capacity, double weight)	Конструктор с параметрами
void Show()	Функция вывода объекта в консоль
int GetPowerFire()	Функция получение параметра мощности огня объекта
void SetPowerFire()	Функция изменения параметра мощности огня объекта
void SaveFile(std::ofstream& f)	Функция сохранения объекта в файл

Таблица 8

Функции класса GlideTransport

Функция	Назначение функции
1	2
GlideTransport()	Конструктор по умолчанию
GlideTransport(int length, int timeInAirSec, int capacity, double weight)	Конструктор с параметрами

Продолжение табл.8

1	2
void Show()	Функция вывода объекта в консоль
int GetLengthWing()	Функция получение параметра длины крыла в метрах объекта
void SetLengthWing()	Функция изменения параметра длины крыла в метрах объекта
void SaveFile(std::ofstream& f)	Функция сохранения объекта в файл

Класс шаблонного стека «Stack» поддерживает функции добавления элемента в стек, удаление элемента, сортировка элементов стека по значению параметра вместимости по людям, по значению параметра веса и по обоим параметрам сразу. Также по сетку пользователь может осуществлять поиск элементов по значениям параметров вместимости по людям, весу. Перечень функций данного класса находится в таблице 9.

Таблица 9

Функции класса Stack

Функция	Назначение функции
1	2
~Stack()	Деструктор
void Show()	Функция вывода всех объектов стека в консоль
void Push(T el)	Функция добавления элемента в стек
void Pop()	Функция изменения параметра мощности огня объекта
void ToNull()	Функция обнуление стека
bool IsEmpty()	Функция показывающая пуст ли стек

Продолжение табл. 9

1	2
void SaveInFile(std::ofstream& f)	Функция сохранения объектов стека в файл
void SortByCapacity()	Функция сортирует объекты стека по параметру вместимость по людям
void SortByWeight()	Функция сортирует объекты стека по параметру вес по тонам
void SearchByCapacity(int capacity)	Функция поиска объекта по параметру вместимость по людям
void SearchByWeight(double weight)	Функция поиска объекта по параметру по тонам

Класс шаблонной односвязный список «List» поддерживает функции добавления элемента в начало списка, удаление элемента из начала списка, получение элемента из начала списка, не удаляя его, изменить размер списка и т.д. Многие вышеперечисленные функция этот класс наследует от класса «ListVoid». Перечень функций этих классов находится в таблицах 10-11.

Таблица 10

Функции класса VoidList

Функция	Назначение функции
1	2
void _push_front(void* a)	Функция добавления ссылки на объект в начало списка
int _size()	Функция вычисления размера списка
bool _empty()	Функция проверки списка на пустоту

Продолжение табл. 10

1	2
void* _pop_front()	Функция удаления первого элемента списка
void* _front()	Функция возвращения первого элемента списка
void _clear()	Функция очистки списка
void _resize(int n)	Функция изменения размера списка
void _resize(int n, void* el)	Функция изменения размера списка, с уже заполненными элементами

Таблица 11

Функции класса List

Функция	Назначение функции
1	2
void PushFront(T a)	Функция добавления ссылки на объект в начало списка
int _Size()	Функция вычисления размера списка
bool Empty()	Функция проверки списка на пустоту
T PopFront()	Функция удаления первого элемента списка
T Front()	Функция возвращения первого элемента списка
void Clear()	Функция очистки списка
void Resize(int n)	Функция изменения размера списка
void Resize(int n, T el)	Функция изменения размера списка, с уже заполненными элементами

Продолжение табл. 11

1	2
void SaveInFile(std::ofstream& f)	Функция сохранения объектов списка в файл

Информация о полях и функция каждого класса сформирована в подробной диаграмме классов(рис.2).

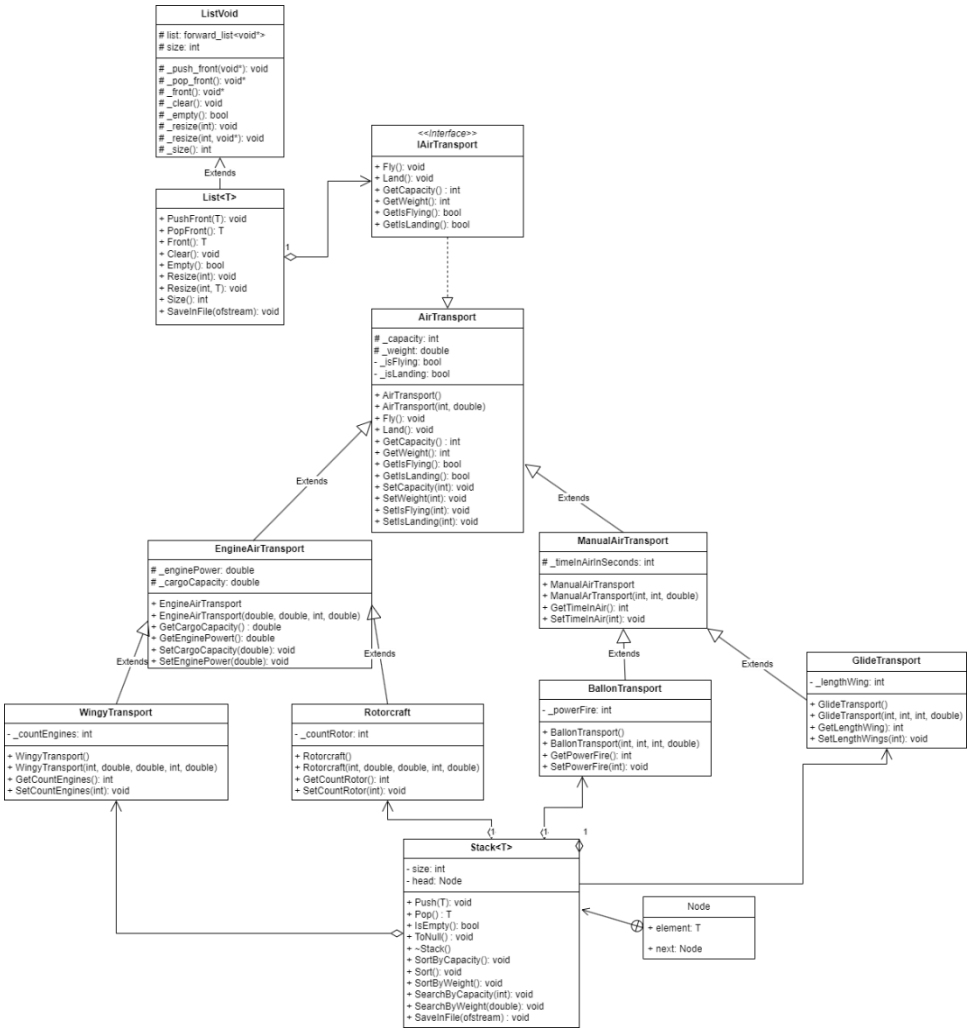

Рис.2. Подробная диаграмма классов

3. Логическая структура программы

При запуске программы вызывается модуль main. Дальнейшая работа программы определяется последовательностью действий, которую производит пользователь. Пользователь может выгрузить информацию из файла, например из файла input.txt, далее информация, считанная с файла, будет распределена по контейнерам. Существует 4 стека, хранящие объекты классов воздушного транспорта, также существует односвязный список, хранящий указатель на объекты класса интерфейса IAirTransport. Эти контейнеры заполняются объектами/указателями на объекты. Тоже самое произойдет при создании нового транспорта.

При сохранении информации в файлы, пользователь может определить хочет он создать 4 файла, содержащих объекты разных классов соответственно из стека, или сохранить все в один файл из списка.

Далее работа программы производится взаимодействием классов между собой(рис.3).

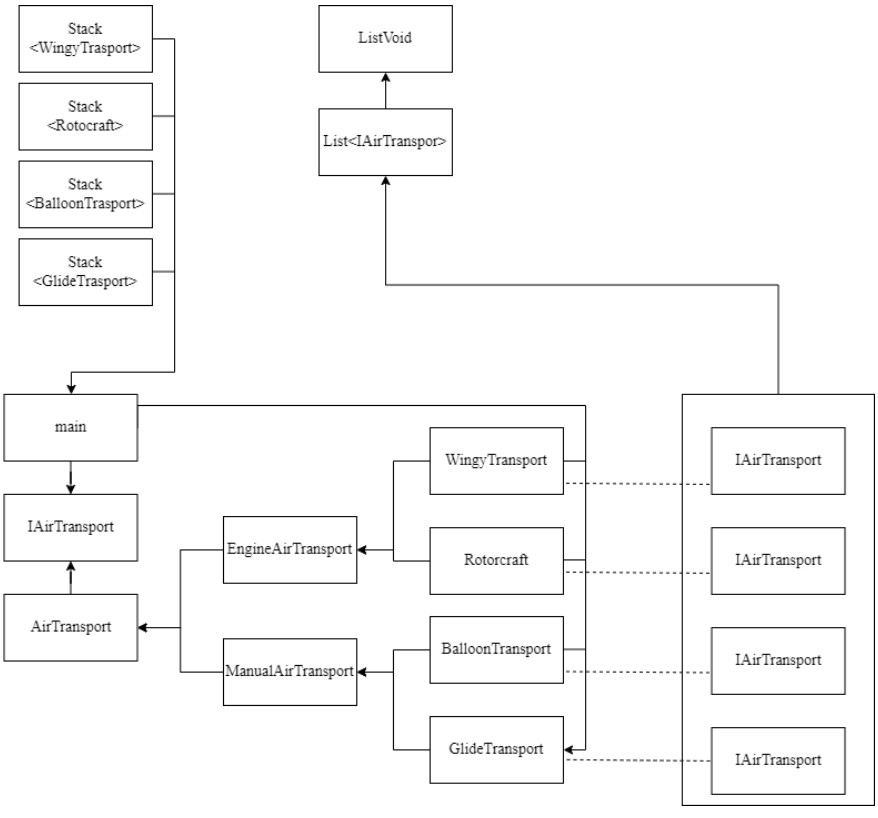

Рис. 3. Логическая схема

4. Модульная структура программы

Под модульной структурой понимается составление программы из функциональных модулей (кусков, фрагментов, сегментов, подпрограмм). Модули могут выполнять самые разнообразные функции и использовать в самых разнообразных проектах. При написании программы всегда необходимо стремиться к оптимизации исходного кода – однотипные куски кода оформлять в виде модулей, которые можно в любой момент использовать.

Взаимодействие модулей, файлов программы, отображено на рис.4.

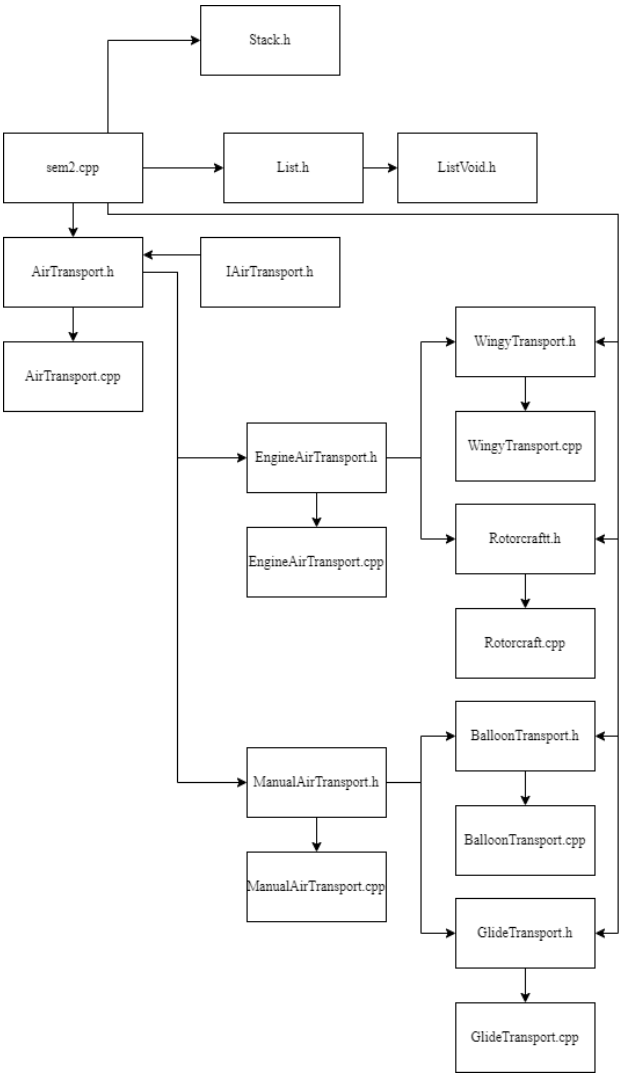

Рис.4. Модульная диаграмма

В модулях «EngineAirTransport.h», «ManualAirTransport.h», «BalloonAirTransport.h», «GlideAirTransport.h», «WingyTransport.h», «Rotorcraft.h» (и соответствующих им .cpp модулях) описываются производные классы рассматриваемых объектов. Модуль AirTransport.h (AirTransport.cpp) содержит описание базового класса для данных объектов.

Модуль «IAirTransport.h» содержит класс-интерфейс, в котором присутствуют декларации виртуальных методов для полиморфной обработки данных предметной области.

Все объекты классов и указатели на эти классы хранятся в контейнерах, описанных в модулях «Stack.h» и «List.h».

Модуль «sem2.cpp» служит для создания объектов классов, описанных в вышеописанных модулях и также для взамодействия с этими объектами.

5. Тестирование программы

В приведенных ниже таблицах представлены результаты тестирования программы.

Таблица 12

Протокол тестирования классов

Дата тестирования	Класс	Кто проводил тестирование	Описание теста	Результаты тестирования
28.05.2024	WingyTransport	Симаньков А.Е.	GetCapacity	Успех
28.05.2024	Rotorcraft	Гончаров Е. Д.	Show	Успех
28.05.2024	BalloonTransport	Овчинников М.В	SaveInFile	Успех
28.05.2024	GlideTransport	Зернов В.А.	SetLengthWing	Успех
01.06.2024	Stack	Зернов В.А	Push	Успех
01.06.2024	Stack	Зернов В.А.	SortByCapacity	Успех
03.06.2024	List	Овчинников М.В	SaveAllElements	Ошибка при сохранении в файл
03.06.2024	List	Гончаров Е. Д.	PushFront	Успех

Таблица 13

Протокол тестирования внешних функций

Дата тестирования	Функция	Кто проводил тестирование	Описание теста	Результаты тестирования
03.06.2024	main	Симаньков А.Е.	Ввод данных	Успех
			Добавление данных	Успех
			Отображение данных	Успех
			Сохранение данных	Успех
			Создание и вызов функций классов	Успех

Таблица 14

Протокол тестирования по техническому заданию

Дата тестирования	Кто проводил тестирование	Описание теста	Результаты тестирования
30.05.2024	Гончаров Е.Д.	Проверка на правильность считывания информации с файла и создание объектов по этой информации	Успех
03.06.2024	Симаньков А.Е.	Проверка на заполнение листа	Успех
02.06.2024	Овчинников М.В.	Проверка на правильность заполнения листа введенными объектами и создание объектов	Успех
04.06.2024	Удальцов А.П.	Проверка программы на заполнение стека	Успех

Заключение

В результате курсовой работы было разработано программное обеспечение для работы с объектами предметной области – «Воздушный транспорт». В рамках данной работы была разработана программа, способная хранить в стеке и односвязном линейном списке объекты, введенные пользователем, а также обрабатывать их. Созданы механизмы работы с исключительными ситуациями.

В качестве приобретенных компетенций были получены дополнительные навыки программирования на языке C++, разобрана работа с родственными типами и принцип работы void-указателя, закреплены полученные знания дисциплины объектно-ориентированное программирование.

Инструкция по работе с программой написана в приложении 3.

Список литературы

1. Страуструп, Б. Язык программирования C++ [Текст] / Страуструп, Б – Бином, 2010.

2. Ершов, Е.В. Методика и организация самостоятельной работы: учебное пособие [Текст] / Ершов Е.В., Виноградова Л.Н., Селивановских В.В. – Череповец: ЧГУ, 2015.

3. Лафоре, Р. Объектно-ориентированное программирование в C++ / Р. Лафоре. - СПб.: Питер, 2019. - 928 с.

4. Лаптев В.В. «C++. Объектно – ориентированное программирование: Учебное пособие. – СПб.: Питер, 2008. – 464 с.: ил. – (Серия «Учебное пособие»)»

5. https://aviatc.ru/articles/klassifikatsija-vozdushnogo-transporta/ - web-сайт – информация, посвященная воздушному транспорту

Приложение 1.

Техническое задание

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий
(наименование структурного подразделения)
Кафедра математического и программного обеспечения ЭВМ
(наименование кафедры)
Модуль: Объектно-ориентированное программирование
(наименование дисциплины в соответствии с учебным планом)

УТВЕРЖДАЮ
Зав. кафедрой МПО ЭВМ,
д.т.н., профессор Ершов Е.В.
«___» апреля 2024 г.

Объектно-ориентированное программирование на языке C++
Техническое задание на курсовую работу
Листов __

Руководитель: Матевосян Р.А.
Исполнитель: студент гр. 1ПИб-02-2оп-22
Зернов Владислав Александрович

2024 год

Введение

Курсовая работа посвящена разработке программы на языке C++, и разработать иерархию родственных типов, корневой класс которой абстрактный базовый класс (класс-интерфейс), для моделирования и обработки данных предметной области набором отложенных методов.

1. Основания для разработки

Основанием для разработки является задание на курсовую работу по дисциплине «Объектно-ориентированное программирование», выданное на кафедре МПО ЭВМ ИИТ ЧГУ.

Дата утверждения: 14 марта 2024 года.

Наименование темы разработки: «Объектно-ориентированное программирование на языке C++».

2. Назначение разработки

Основная задача курсовой работы: освоить на практике материал, полученный в ходе изучения дисциплины «Объектно-ориентированное программирование».

3. Требования к программе

3.1. Требования к функциональным характеристикам

Разработать иерархию родственных типов, корневой класс которой абстрактный базовый класс (класс-интерфейс), для моделирования и обработки данных предметной области набором отложенных методов - полиморфная обработка родственных объектов (согласно варианту A29 «Воздушный транспорт»).

Создать обобщенный (void*) контейнерный класс (базовый) и от него, используя закрытое наследование, производный класс – шаблон для хранения указателей на абстрактный базовый класс-интерфейс (согласно варианту B3 «Линейный односвязный список»).

Для хранения объектов каждого производного класса используйте структуру данных (согласно варианту C7 «Стек»)

Реализовать файловый ввод/вывод и ввод данных с клавиатуры, вывод данных на дисплей. Предусмотреть обработку различных исключительных ситуаций. Такие как: неправильно введенный пользователем диапазон значений у полей класса.

3.2. Требования к надёжности

Реализовать функции обработки данных (сортировка и поиск по выбранным полям и задаваемым диапазонам значений, другие функции, в том числе перегруженные).

Работа всех функций должна быть проверена, и результаты проверки оформлены протоколом тестирования

3.3. Условия эксплуатации

Отсутствуют

3.4. Требования к составу и параметрам технических средств

Для нормального функционирования программного средства минимальный состав и параметры технических средств должны соответствовать нижеследующему:

- Процессор с тактовой частотой не менее 2000 MHz, частота 2,3 Ghz;
- Оперативная память 2Gb и выше;
- Архитектура с разрядностью 32 бит или 64 бит;
- Наличие компьютерной мыши, клавиатуры, монитора (для персонального компьютера).

3.5. Требования к информационной и программной совместимости

Минимальные требования для информационной и программной совместимости:

- Операционная система (Windows 7 и выше);
- Наличие компьютерного приложения Visual Studio 2019, если установлена более ранняя версия Visual Studio, необходимо установить дополнительно набор инструментов платформы v142 и пакет SDK для Windows версией 10.0.

3.6. Требования к маркировке и упаковке

Отсутствуют.

3.7. Требования к транспортированию и хранению

Файлы, необходимые для корректной работы, необходимо записать на CD-диск.

3.8. Специальные требования

Отсутствуют.

4. Требования к программной документации

4.1. Содержание расчётно-пояснительной записки:

- Оглавление;
- введение;
- программирование классов;
- логическое программирование;
- физическое проектирование;
- тестирование;
- заключение;
- список литературы;
- техническое задание;
- руководство пользователя;
- текст программы.

4.2. Требования к оформлению

Требования к оформлению, установленные ГОСТ, должны быть выполнены на протяжении всей работы без каких-либо изменений (в табл. П1.1).

Таблица П1.1

Требования к оформлению

Документ	Печать на отдельных листах формата А4 (20х297 мм); оборотная сторона не заполняется; листы нумеруются. Печать возможна ч/б.
Страницы	Ориентация — книжная; отдельные страницы, при необходимости, альбомная. Поля: верхнее, нижнее — по 2 см, левое — 3 см, правое — 2 см.
Абзацы	Межстрочный интервал — 1,5, перед и после абзаца — 0.
Шрифты	Кегль — 14. В таблицах шрифт 14. Шрифт листинга — 8 (возможно в 2 колонки).
Рисунки	Подписывается под ним по центру: «Рис.Х. Название В» приложениях: «Рис.П.3. Название»
Таблицы	Подписывается: над таблицей, выравнивание по правому: «Таблица Х». В следующей строке по центру Название Надписи в «шапке» (имена столбцов, полей) — по центру. В теле таблицы (записи) текстовые значения — выравнены по левому краю, числа, даты — по правому.

5. Стадии и этапы разработки

Стадии и этапы разработки представлены в таблице П1.2.

Таблица П1.2

Этапы разработки

Наименование этапа разработки	Сроки разработки	Результат выполнения	Отметка о выполнении
1	2	3	4
Определение темы для курсовой работы	20.03.2024	Утверждена тема разработки	Выполнено
Оформление техническое задания	10.05.2024	Выполненное тех. задания	Выполнено

Продолжение табл. П1.2

1	2	3	4
Логическое проектирование	30.04.2024	Создан алгоритм решения задания	Выполнено
Физическое проектирование	05.05.2024	Создана программа	Выполнено
Тестирование	05.06.2024	Проверка правильности	Выполнено
Тестирование	05.06.2024	Проверка правильности расчетов построения	Выполнено
Написание РПЗ	05.06.2024	Создано РПЗ по курсовой работе	Выполнено

6. Порядок контроля и приёмки

Порядок контроля и приёма представлены в таблице П1.3.

Таблица П1.3

Порядок контроля и приемки

Наименование контрольного этапа выполнения курсовой работы	Сроки контроля	Результат выполнения	Отметка о приемке результата контрольного этапа
Утверждение технического задания	19.05.2024	Документ «Техническое задание» проверен	
Демонстрация работы первой версии программы	30.05.2024	Работа программы проверена	
Поиск ошибок в программе	30.05.2024	Все найденные ошибки исправлены	
Демонстрация окончательной версии программы	30.05.2024	Работа программы проверена	
Защита курсовой работы	06.06.2024	Курсовая работа защищена	

Приложение 2.

Текст программы

```
#include <iostream>
#include <Windows.h>
#include "AirTransport.h"
#include "EngineAirTransport.h"
#include "WingyTransport.h"
#include "Rotorcraft.h"
#include "BallonTransport.h"
#include "Stack.h"
#include "List.h"
#include "GlideTransport.h"
#include <string>
using namespace std;
void showCommands() {
	cout << "----------------" << endl;
	cout << "Список действий." << endl;
	cout << "----------------" << endl;
	cout << "1.Загрузить файл." << endl;
	cout << "2.Список транспорта." << endl;
	cout << "3.Создать транспорт." << endl;
	cout << "4.Осортировать транспорта." << endl;
	cout << "5.Найти транспорт." << endl;
	cout << "6.Сохранить в файл" << endl;
	cout << "7.Закончить работу." << endl << endl;
}
void showEntity() {
	cout << "-------------------------" << endl;
	cout << "Виды транспорта." << endl;
	cout << "-------------------------" << endl;
	cout << "1. Крылатый транспорт." << endl;
	cout << "2. Винтокрылый транспорт." << endl;
	cout << "3. Шариковый транспорт." << endl;
	cout << "4. Парирующий транспорт." << endl << endl;
}
int main()
{
	setlocale(LC_ALL, "rus");
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);
	string nameFile;
	ifstream f;
	ofstream F;
	int action = 0;
	Stack<WingyTransport> wingyTransportStack;
	Stack<Rotorcraft> rotorcraftStack;
	Stack<BallonTransport> balloonTransportStack;
	Stack<GlideTransport> glideTransportStack;
	List<IAirTransport> list;
	while (true) {
		showCommands();
		cout << "Выберите действие: ";
		cin >> action;
		if (action == 7) {
			cout << "----------------------------
Завершение работы----------------------------" << endl;
			break;
		}
		switch (action)
		{
			case 1: // load
			#pragma region LoadFromFile
				cout << "Введите путь
до файла и его имя: ";
				cin >> nameFile;
				f.open(nameFile);
				while (!f.eof()) {
					string
nameClass;
					int capacity;
					double
weight;
					bool
isLanding;
					bool
isFlying;
					f >>
nameClass;
					f >> weight;
					f >>
capacity;
					f >>
isLanding;
					f >>
isFlying;
					if (isFlying
&& isLanding) {

						cout << "----------------------------Транспорт не может
одновременно быть в воздухе и на земле----------------------------";

						break;
					}
					if
(nameClass == "WingyTransport") {

						int countEngines;

						double cargoCapcity;

						double enginePower;
						f
>> countEngines;
						f
>> cargoCapcity;
						f
>> enginePower;

						try {

						WingyTransport transport(countEngines, cargoCapcity,
enginePower, capacity, weight);

						if (isLanding) transport.Land();

						else transport.Fly();

						wingyTransportStack.Push(transport);

						list.PushFront(transport);

						}

						catch (exception ex) {

						cout << "----------------------------" << ex.what() << "------
----------------------" << endl;

						}
					}
					else if
(nameClass == "BallonTransport") {
```

Рис.П2.1. Текст файла sem2.cpp

Рис.П2.1. Продолжение

```
                int powerFire;
                int timeInAirSec;
                f >> powerFire;
                f >> timeInAirSec;
                try {
                BallonTransport
transport(powerFire, timeInAirSec, capacity, weight);
                if (isLanding)
                transport.Land();
                else transport.Fly();

balloonTransportStack.Push(transport);

list.PushFront(transport);
                }
                catch (exception ex) {
                cout << "----------
----------" << ex.what() << "----------" << endl;
                }
                }
else if (nameClass == "Rotorcraft") {
                int countRotor;
                double cargoCapcity;
                double enginePower;
                f >> countRotor;
                f >> cargoCapcity;
                f >> enginePower;
                try {
                Rotorcraft
transport(countRotor, cargoCapcity, enginePower, capacity,
                weight);
                if (isLanding)
                transport.Land();
                else transport.Fly();

rotorcraftStack.Push(transport);

list.PushFront(transport);
                }
                catch (exception ex) {
                cout << "----------
----------" << ex.what() << "----------" << endl;
                }
                }
else if (nameClass == "GlideTransport") {
                int lengthWing;
                int timeInAirSec;
                f >> lengthWing;
                f >> timeInAirSec;
                try {
                GlideTransport
transport(lengthWing, timeInAirSec, capacity, weight);
                if (isLanding)
                transport.Land();
                else transport.Fly();

glideTransportStack.Push(transport);

list.PushFront(transport);
                }
                catch (exception ex) {
                cout << "----------
----------" << ex.what() << "----------" << endl;
                }
                }
                }
                f.close();
                break;
                #pragma endregion
                case 2: // to list
                #pragma region ToList
cout << "----------Весь крылатый
транспорт----------" << endl;
wingyTransportStack.Show();
cout << "----------Весь винтокрылый
транспорт----------" << endl;
rotorcraftStack.Show();
cout << "----------Весь шариковый
транспорт----------" << endl;
balloonTransportStack.Show();
cout << "----------Весь парирующий
транспорт----------" << endl;
glideTransportStack.Show();
                break;
                #pragma endregion
                case 3: // create
                #pragma region CreateEntity
                showEntity();
cout << "Выберите вид транспорта: ";
                cin >> action;
                int capacity;
                double weight;
                if (action <= 2) {
cout << "Введите вместимость по людям:
                ";
                cin >> capacity;
cout << "Введите вес транспорта по
                тоннам: ";
                cin >> weight;
                }
                if (action == 1) {
                int countEngines;
                double cargoCapacity;
                double enginePower;
                bool isFlying;
cout << "Введите кол-во двигателей: ";
                cin >> countEngines;
cout << "Введите вместимость по грузу в
                тонах: ";
                cin >> cargoCapacity;
cout << "Введите мощность двигателя: ";
                cin >> enginePower;
                try {
                WingyTransport
transport(countEngines, cargoCapacity, enginePower, capacity,
                weight);
cout << "Транспорт в полете?"
                << endl;
cout << "Введите 1, если да.
Введите 0, если нет" << endl;
                cin >> isFlying;
                if (isFlying) transport.Fly();
                else transport.Land();

wingyTransportStack.Push(transport);
                list.PushFront(transport);
                }
                catch (exception ex) {
                cout << "----------"
<< ex.what() << "----------" << endl;
                }
                }
                else if (action == 2) {
                int countRotor;
                double cargoCapacity;
                double enginePower;
                bool isFlying;
cout << "Введите кол-во винтов: ";
                cin >> countRotor;
cout << "Введите вместимость по грузу в
                тонах: ";
                cin >> cargoCapacity;
cout << "Введите мощность двигателя: ";
                cin >> enginePower;
```

Рис.П2.1. Продолжение

```
                try {
                    Rotorcraft transport(countRotor,
cargoCapacity, enginePower, capacity, weight);
                    cout << "Транспорт в полете?"
                        << endl;
                    cout << "Введите 1, если да.
Введите 0, если нет" << endl;
                    cin >> isFlying;
                    if (isFlying) transport.Fly();
                    else transport.Land();
                    rotorcraftStack.Push(transport);
                    list.PushFront(transport);
                }
                catch (exception ex) {
                    cout << "----------------------------"
<< ex.what() << "----------------------------" << endl;
                }
            }
            else if (action == 3) {
                int powerFire;
                int timeInAirSec;
                bool isFlying;
                cout << "Введите мощность огня: ";
                cin >> powerFire;
                cout << "Введите время в воздухе в
                    секундах: ";
                cin >> timeInAirSec;
                try {
                    BallonTransport
transport(powerFire, timeInAirSec, capacity, weight);
                    cout << "Транспорт в полете?"
                        << endl;
                    cout << "Введите 1, если да.
Введите 0, если нет" << endl;
                    cin >> isFlying;
                    if (isFlying) transport.Fly();
                    else transport.Land();

                    balloonTransportStack.Push(transport);
                    list.PushFront(transport);
                }
                catch (exception ex) {
                    cout << "----------------------------"
<< ex.what() << "----------------------------" << endl;
                }
            }
            else if (action == 4) {
                int lengthWing;
                int timeInAirSec;
                bool isFlying;
                cout << "Введите длину крыла: ";
                cin >> lengthWing;
                cout << "Введите время в воздухе в
                    секундах: ";
                cin >> timeInAirSec;
                try {
                    GlideTransport
transport(lengthWing, timeInAirSec, capacity, weight);
                    cout << "Транспорт в полете?"
                        << endl;
                    cout << "Введите 1, если да.
Введите 0, если нет" << endl;
                    cin >> isFlying;
                    if (isFlying) transport.Fly();
                    else transport.Land();

                    glideTransportStack.Push(transport);
                    list.PushFront(transport);
                }
                catch (exception ex) {
                    cout << "----------------------------"
<< ex.what() << "----------------------------" << endl;
                }
            }
            else {
                cout << "----------------------------
Неизвестный вид транспорта----------------------------" << endl;
            }
            break;
        #pragma endregion
        case 4: // sort
        #pragma region Sort
            cout << "По какому признаку вы бы хотели
                отсортировать транспорт?" << endl;
            cout << "1.Вместимость по людям" << endl;
            cout << "2.Вес" << endl;
            cout << "3.Вес и вместимость." << endl;
            cout << "Введите номер признака для сортировки: ";
            cin >> action;
            if (action == 1) {
                wingyTransportStack.SortByCapacity();
                rotorcraftStack.SortByCapacity();
                balloonTransportStack.SortByCapacity();
                glideTransportStack.SortByCapacity();
            }
            else if (action == 2) {
                wingyTransportStack.SortByWeight();
                rotorcraftStack.SortByWeight();
                balloonTransportStack.SortByWeight();
                glideTransportStack.SortByWeight();
            }
            else if (action == 3) {
                wingyTransportStack.Sort();
                rotorcraftStack.Sort();
                balloonTransportStack.Sort();
                glideTransportStack.Sort();
            }
            else {
                cout << "----------------------------
Неизвестный признак----------------------------" << endl;
                break;
            }
            cout << "----------------------------Сортировка
                завершена----------------------------" << endl;
            break;
        #pragma endregion
        case 5: // search
        #pragma region Search
            cout << "Выберите по какому признаку будете
                искать объект:" << endl;
            cout << "1. Вес" << endl;
            cout << "2. Вместимость" << endl;
            cout << "Введите номер признака для поиска: ";
            cin >> action;
            if (action == 1) {
                double weight;
                cout << "Введите вес: ";
                cin >> weight;

                wingyTransportStack.SearchByWeight(weight);
                rotorcraftStack.SearchByWeight(weight);

                balloonTransportStack.SearchByWeight(weight);

                glideTransportStack.SearchByWeight(weight);
                cout << endl << endl;
            }
            else if (action == 2) {
                int capacity;
                cout << "Введите вместимость: ";
                cin >> capacity;

                wingyTransportStack.SearchByCapicity(capacity);
                rotorcraftStack.SearchByCapicity(capacity);

                balloonTransportStack.SearchByCapicity(capacity);
                glideTransportStack.Search
```

Рис.П2.1. Продолжение

```
case 6: // save
#pragma region Save
cout << "Выберите
способ сохранения:" << endl;
cout << "1.
Сохранить транспорты в 1 файл" << endl;
cout << "2.
Распределить транспорт по видам по файлам" << endl;
cin >> action;
if (action == 1) {
if
(list.Size() == 0) {

cout << "----------Нет транспорта,
который можно сохранить----------" << endl;

break;
}
cout <<
"Введите путь до файла и его имя: ";
cin >>
nameFile;

F.open(nameFile);

list.SaveAllElements(F);
F.close();
}
else if (action == 2) {
if
(list.Size() == 0) {

cout << "----------Нет транспорта,
который можно сохранить----------" << endl;

break;
}
if
(!wingyTransportStack.IsEmpty()) {

cout << "Введите путь до файла и его имя, куда
будет сохранятся трапнспорт крылатого вида: ";

cin >> nameFile;

F.open(nameFile);

wingyTransportStack.SaveInFile(F);

F.close();
}
if
(!rotorcraftStack.IsEmpty()) {

cout << "Введите путь до файла и его имя, куда
будет сохранятся трапнспорт винтокрылого вида: ";

cin >> nameFile;

F.open(nameFile);

rotorcraftStack.SaveInFile(F);

F.close();
}
if
(!balloonTransportStack.IsEmpty()) {

cout << "Введите путь до файла и его имя, куда
будет сохранятся трапнспорт шарикового вида: ";

cin >> nameFile;

F.open(nameFile);

balloonTransportStack.SaveInFile(F);

F.close();
}
if
(!glideTransportStack.IsEmpty()) {

cout << "Введите путь до файла и его имя, куда
будет сохранятся трапнспорт парирующего вида: ";

cin >> nameFile;

F.open(nameFile);

glideTransportStack.SaveInFile(F);

F.close();
}
}
else {
cout << "---
----------Неизвестный способ----------"
<< endl;
}
break;
#pragma endregion
default:
cout << "----------
----------Неизвестная команда, введите еще раз----------
-------" << endl;
break;
}
}
}
```

```
#pragma once
#include <fstream>
class IAirTransport
{
    public:
        virtual int GetCapacity() = 0;
        virtual double GetWeight() = 0;
        virtual bool GetIsFlying() = 0;
        virtual bool GetIsLanding() = 0;
        virtual void Fly() = 0;
        virtual void Land() = 0;
        virtual void Show() = 0;
        virtual void SaveInFile(std::ofstream& f) = 0;
        virtual bool operator==(IAirTransport& transport) = 0;
        virtual bool operator>(IAirTransport& transport) = 0;
        virtual bool operator>=(IAirTransport& transport) = 0;
        virtual bool operator<(IAirTransport& transport) = 0;
        virtual bool operator<=(IAirTransport& transport) = 0;
        virtual bool operator!=(IAirTransport& transport) = 0;
};
```

Рис.П2.2. Текст интерфейса IAirTransport.h

```
#pragma once
#include "IAirTransport.h"
class AirTransport : public IAirTransport
{
    private:
        bool _isFlying;
        bool _isLanding;
    protected:
        int _capacity;
        double _weight;
    public:
        AirTransport();
        AirTransport(int capacity, double weight);
        int GetCapacity();
        void SetCapacity(int capacity);
        double GetWeight();
        void SetWeight(double weight);
        bool GetIsFlying();
        bool GetIsLanding();
        void Fly();
        void Land();
        void Show();
        void SaveInFile(std::ofstream& f);
        bool operator==(IAirTransport& transport);
        bool operator>(IAirTransport& transport);
        bool operator>=(IAirTransport& transport);
        bool operator<(IAirTransport& transport);
        bool operator<=(IAirTransport& transport);
        bool operator!=(IAirTransport& transport);
};#include "airTransport.h"
#include <exception>
#include <stdexcept>
#include "CheckValidation.h"
#include <iostream>
using namespace std;
AirTransport::AirTransport()
{
    _weight = 50;
    _capacity = 50;
    _isFlying = false;
    _isLanding = true;
}

AirTransport::AirTransport(int capacity, double weight)
{
    _capacity = CheckValidation::CheckNumber(capacity);
    _weight = CheckValidation::CheckNumber(weight);
    _isFlying = false;
    _isLanding = true;
}

int AirTransport::GetCapacity()
{
    return _capacity;
}

void AirTransport::SetCapacity(int capacity)
{
    _capacity = CheckValidation::CheckNumber(capacity);
}

double AirTransport::GetWeight()
{
    return _weight;
}

void AirTransport::SetWeight(double weight)
{
    _weight = CheckValidation::CheckNumber(weight);
}

bool AirTransport::GetIsFlying()
{
    return _isFlying;
}

bool AirTransport::GetIsLanding()
{
    return _isLanding;
}

void AirTransport::Fly()
{
    if (_isFlying && _isLanding) {
        throw runtime_error("Такого не может быть!!!!");
    }
    _isFlying = true;
    _isLanding = false;
}

void AirTransport::Land()
{
    if (_isFlying && _isLanding) {
        throw logic_error("Такого не может быть!!!!");
    }
    _isFlying = false;
    _isLanding = true;
}

void AirTransport::Show()
{
    cout << "AirTransport" << endl;
    cout << "weight: " << _weight << endl;
    cout << "capacity: " << _capacity << endl;
}
```

Рис.П2.3. Текст класса AirTransport

Продолжение рис. П2.3

```
	void AirTransport::SaveInFile(std::ofstream& f)
	{
		f << "AirTransport" << endl;
		f << _weight << endl;
		f << _capacity << endl;
		f << _isLanding << endl;
		f << _isFlying << endl;
	}

	bool AirTransport::operator==(IAirTransport& transport)
	{
		return this->GetWeight() == transport.GetWeight() &&
this->GetCapacity() == transport.GetCapacity();
	}

	bool AirTransport::operator>(IAirTransport& transport)
	{
		return this->GetWeight() > transport.GetWeight() &&
this->GetCapacity() > transport.GetCapacity();
	}

	bool AirTransport::operator>=(IAirTransport& transport)


	#pragma once
	#include "AirTransport.h"
	class EngineAirTransport : public AirTransport
	{
		protected:
			double _enginePower;
			double _cargoCapacity;
		public:
			EngineAirTransport(double enginePower,
double cargoCapacity, double capacity, int weight);
			EngineAirTransport();
			double GetEnginePower();
			double GetCargoCapacity();
			void SetEnginePower(double enginePower);
			void SetCargoCapacity(double
cargoCapacity);
			void Show();
			void SaveInFile(std::ofstream& f);
	};
#include "EngineAirTransport.h"
#include "CheckValidation.h"
#include <iostream>
using namespace std;
EngineAirTransport::EngineAirTransport(double enginePower,
double cargoCapacity, double capacity, int weight) :
AirTransport(capacity, weight)
{
		_cargoCapacity =
CheckValidation::CheckNumber(cargoCapacity);
		_enginePower =
CheckValidation::CheckNumber(enginePower);
}
EngineAirTransport::EngineAirTransport() : AirTransport()
{
		_cargoCapacity = 20;
		_enginePower = 50;
	{
		return this->GetWeight() >= transport.GetWeight() &&
this->GetCapacity() >= transport.GetCapacity();
	}

	bool AirTransport::operator<(IAirTransport& transport)
	{
		return this->GetWeight() < transport.GetWeight() &&
this->GetCapacity() < transport.GetCapacity();
	}

	bool AirTransport::operator<=(IAirTransport& transport)
	{
		return this->GetWeight() <= transport.GetWeight() &&
this->GetCapacity() <= transport.GetCapacity();
	}

	bool AirTransport::operator!=(IAirTransport& transport)
	{
		return this->GetWeight() != transport.GetWeight() &&
this->GetCapacity() != transport.GetCapacity();
	}

}
double EngineAirTransport::GetEnginePower()
{
		return _enginePower;
}
double EngineAirTransport::GetCargoCapacity()
{
		return _cargoCapacity;
}
void EngineAirTransport::SetEnginePower(double enginePower)
{
		_enginePower =
CheckValidation::CheckNumber(enginePower);
}
void EngineAirTransport::SetCargoCapacity(double
cargoCapacity)
{
		_cargoCapacity =
CheckValidation::CheckNumber(cargoCapacity);
}
void EngineAirTransport::Show()
{
		cout << "EngineAirTransport" << endl;
		cout << "cargo capaciry: " << _cargoCapacity << endl;
		cout << "engine power: " << _enginePower << endl;
		cout << "weight: " << _weight << endl;
		cout << "capacity: " << _capacity << endl;
}
void EngineAirTransport::SaveInFile(std::ofstream& f)
{
		f << "EngineAirTransport" << endl;
		f << _cargoCapacity << endl;
		f << _enginePower << endl;
		f << _weight << endl;
		f << _capacity << endl;
		f << GetIsLanding() << endl;
		f << GetIsFlying() << endl;
}
```

Рис.П2.5. Текст класса EngineAirTransport

```
#pragma once
#include "AirTransport.h"
class ManualAirTransport : public AirTransport
{
    protected:
        int _timeInAirInSeconds;
    public:
        ManualAirTransport();
        ManualAirTransport(int timeInAirSec, int capacity, double weight);
        int GetTimeInAir();
        void SetTimeInAir(int timeInAirInSeconds);
        void Show();
        void SaveInFile(std::ofstream& f);
};
#include "ManualAirTransport.h"
#include "CheckValidation.h"
#include <iostream>
using namespace std;
ManualAirTransport::ManualAirTransport() : AirTransport(4, 25)
{
    _timeInAirInSeconds = 25;
}
ManualAirTransport::ManualAirTransport(int timeInAirSec, int capacity, double weight) : AirTransport(capacity, weight)
{
    _timeInAirInSeconds = CheckValidation::CheckNumber(timeInAirSec);
}
int ManualAirTransport::GetTimeInAir()
{
    return _timeInAirInSeconds;
}
void ManualAirTransport::SetTimeInAir(int timeInAirInSeconds)
{
    _timeInAirInSeconds = CheckValidation::CheckNumber(timeInAirInSeconds);
}
void ManualAirTransport::Show()
{
    cout << "ManualAirTransport" << endl;
    cout << "time in air in seconds: " << _timeInAirInSeconds << endl;
    cout << "weight: " << _weight << endl;
    cout << "capacity: " << _capacity << endl;

}
void ManualAirTransport::SaveInFile(std::ofstream& f)
{
    f << "ManualAirTransport" << endl;
    f << _timeInAirInSeconds << endl;
    f << _weight << endl;
    f << _capacity << endl;
    f << GetIsLanding() << endl;
    f << GetIsFlying() << endl;
}
```

Рис.П2.6. Текст класса ManualAirTransport

```
#pragma once
#include "EngineAirTransport.h"
class WingyTransport : public EngineAirTransport
{
    private:
        int _countEngines;
    public:
        WingyTransport();
        WingyTransport(int countEngines, double cargoCapacity, double enginePower, int capacity, double weight);
        int GetCountEngines();
        void SetCountEngines(int countEngines);
        void Show();
        void SaveInFile(std::ofstream& f);
};

#include "WingyTransport.h"
#include "CheckValidation.h"
#include <iostream>
using namespace std;
WingyTransport::WingyTransport() : EngineAirTransport(60, 70, 100, 46)
{
    _countEngines = 4;
}
WingyTransport::WingyTransport(int countEngines, double cargoCapacity, double enginePower, int capacity, double weight) :
    EngineAirTransport(enginePower, cargoCapacity, capacity, weight)
{
    _countEngines = CheckValidation::CheckNumber(countEngines);
}
int WingyTransport::GetCountEngines()
{
    return _countEngines;
}
void WingyTransport::SetCountEngines(int countEngines)
{
    _countEngines = CheckValidation::CheckNumber(countEngines);
}
void WingyTransport::Show()
{
    cout << "WingyTransport" << endl;
    cout << "count of engines: " << _countEngines << endl;
    cout << "cargo capaciry: " << _cargoCapacity << endl;
    cout << "engine power: " << _enginePower << endl;
    cout << "weight: " << _weight << endl;
    cout << "capacity: " << _capacity << endl;
}
void WingyTransport::SaveInFile(std::ofstream& f)
{
    f << "WingyTransport" << endl;
    f << _weight << endl;
    f << _capacity << endl;
    f << GetIsLanding() << endl;
    f << GetIsFlying() << endl;
    f << _countEngines << endl;
    f << _cargoCapacity << endl;
    f << _enginePower << endl;
}
```

Рис.П2.7. Текст класса WingyTransport

```
#pragma once
#include "EngineAirTransport.h"
class Rotorcraft : public EngineAirTransport
{
    private:
        int _countRotor;
    public:
        Rotorcraft();
        Rotorcraft(int countRotor, double cargoCapacity, double enginePower, int capacity, double weight);
        int GetCountRotor();
        void SetCountRotor(int countRotor);
        void Show();
        void SaveInFile(std::ofstream& f);
};
```

```
#include "Rotorcraft.h"
#include "CheckValidation.h"
#include <iostream>
using namespace std;
Rotorcraft::Rotorcraft() : EngineAirTransport()
{
    _countRotor = 4;
}
Rotorcraft::Rotorcraft(int countRotor, double cargoCapacity, double enginePower, int capacity, double weight) :
    EngineAirTransport(enginePower, cargoCapacity, capacity, weight)
{
    _countRotor = CheckValidation::CheckNumber(countRotor);
}
int Rotorcraft::GetCountRotor()
{
    return _countRotor;
}
void Rotorcraft::SetCountRotor(int countRotor)
{
    _countRotor = CheckValidation::CheckNumber(countRotor);
}
void Rotorcraft::Show()
{
    cout << "Rotorcraft" << endl;
    cout << "count of rotor: " << _countRotor << endl;
    cout << "cargo capaciry: " << _cargoCapacity << endl;
    cout << "engine power: " << _enginePower << endl;
    cout << "weight: " << _weight << endl;
    cout << "capacity: " << _capacity << endl;
}
void Rotorcraft::SaveInFile(std::ofstream& f)
{
    f << "Rotorcraft" << endl;
    f << _weight << endl;
    f << _capacity << endl;
    f << GetIsLanding() << endl;
    f << GetIsFlying() << endl;
    f << _countRotor << endl;
    f << _cargoCapacity << endl;
    f << _enginePower << endl;
}
```

Рис.П2.8. Текст класса Rotorcraft

```
#pragma once
#include "ManualAirTransport.h"
class BallonTransport : public ManualAirTransport
{
    private:
        int _powerFire;
    public:
        BallonTransport();
        BallonTransport(int powerFire, int timeInAirSec, int capacity, double weight);
        int GetPowerFire();
        void SetPowerFire(int powerFire);
        void Show();
        void SaveInFile(std::ofstream& f);
};
```

```
#include "BallonTransport.h"
#include <iostream>
#include "CheckValidation.h"
using namespace std;
BallonTransport::BallonTransport() : ManualAirTransport()
{
    _powerFire = 45;
}
BallonTransport::BallonTransport(int powerFire, int timeInAirSec, int capacity, double weight) : ManualAirTransport(timeInAirSec, capacity, weight)
{
    _powerFire = CheckValidation::CheckNumber(powerFire);
}
int BallonTransport::GetPowerFire()
{
    return _powerFire;
}
void BallonTransport::SetPowerFire(int powerFire)
{
    _powerFire = CheckValidation::CheckNumber(powerFire);
}
void BallonTransport::Show()
{
    cout << "BallonTransport" << endl;
    cout << "power fire: " << _powerFire << endl;
    cout << "time in air in seconds: " << _timeInAirInSeconds << endl;
    cout << "weight: " << _weight << endl;
    cout << "capacity: " << _capacity << endl;
}
void BallonTransport::SaveInFile(std::ofstream& f)
{
    f << "BallonTransport" << endl;
    f << _weight << endl;
    f << _capacity << endl;
    f << GetIsLanding() << endl;
    f << GetIsFlying() << endl;
    f << _powerFire << endl;
    f << _timeInAirInSeconds << endl;
}
```

Рис.П2.9. Текст класса BalloonAirTransport

```
#pragma once
#include "ManualAirTransport.h"
class GlideTransport : public ManualAirTransport
{
private:
    int _lengthWing;

public:
    GlideTransport();
    GlideTransport(int length);
    GlideTransport(int length, int timeInAirSec, int capacity,
double weight);
    int GetLengthWing();
    void SetLengthWing(int length);
    void Show();
    void SaveInFile(std::ofstream& f);
};
#include "GlideTransport.h"
#include "CheckValidation.h"
#include <iostream>
using namespace std;
GlideTransport::GlideTransport() : ManualAirTransport()
{
    _lengthWing = 20;
}
GlideTransport::GlideTransport(int length) :
ManualAirTransport()
{
    _lengthWing = length;
}
GlideTransport::GlideTransport(int length, int timeInAirSec,
int capacity, double weight) : ManualAirTransport(timeInAirSec,
capacity, weight)
{
    _lengthWing                                          =
CheckValidation::CheckNumber(length);
}
int GlideTransport::GetLengthWing()
{
    return _lengthWing;
}
void GlideTransport::SetLengthWing(int length)
{
    _lengthWing                                          =
CheckValidation::CheckNumber(length);
}
void GlideTransport::Show()
{
    cout << "GlideTransport" << endl;
    cout << "length wing: " << _lengthWing << endl;
    cout << "time in air in seconds: " <<
_timeInAirInSeconds << endl;
    cout << "weight: " << _weight << endl;
    cout << "capacity: " << _capacity << endl;
}
void GlideTransport::SaveInFile(std::ofstream& f)
{
    f << "GlideTransport" << endl;
    f << _weight << endl;
    f << _capacity << endl;
    f << GetIsLanding() << endl;
    f << GetIsFlying() << endl;
    f << _lengthWing << endl;
    f << _timeInAirInSeconds << endl;
}
```

Рис.П2.10. Текст класса GlideTransport

```
#pragma once
#include<forward_list>
using namespace std;
class ListVoid
{
protected:
    forward_list<void*> list;
    int size;
    ListVoid() {
        list.clear();
    }
    void _push_front(void* const& pointer) {
        list.push_front(pointer);
        size++;
    }
    void _pop_front() {
        list.pop_front();
        size--;
    }
    void* _front() {
        return list.front();
    }
    void _clear() {
        list.clear();
        size = 0;
    }
    bool _empty() {
        return list.empty();
    }
    void _resize(int n) {
        list.resize(n);
        size = n;
    }
    void _resize(int n, void* const& value) {
        list.resize(n, value);
        size = n;
    }
    int _size() {
        return size;
    }
};
```

Рис.П2.11. Текст класса ListVoid

```
#pragma once
#include "AirTransport.h"
#include<fstream>
#include <stdexcept>
#include <iostream>
using namespace std;
template<class T>
class Stack
{
	private:
		class Node {
			public:
				T element;
				Node *next;
		};
		int size = 0;
		Node *head;
	public:
		~Stack();
		void Push(T& airTransport);
		T& Pop();
		bool IsEmpty();
		void ToNull();
		void Show();
		void SaveInFile(std::ofstream& f);
		void SortByCapacity();
		void Sort();
		void SortByWeight();
		void SearchByCapicity(int capcity);
		void SearchByWeight(double weight);
		T& operator[](int index);
};
template<class T>
Stack<T>::~Stack()
{
	ToNull();
}
template<class T>
void Stack<T>::Push(T& airTransport)
{
	Node* tmp = new Node;
	tmp->next = head;
	head = tmp;
	head->element = airTransport;
	size++;
}
template<class T>
T& Stack<T>::Pop()
{
	if (IsEmpty()) throw out_of_range("Стек пуст!");
	Node* tmp = head;
	T air = tmp->element;
	head = head->next;
	delete(tmp);
	size--;
	return air;
}
template<class T>
bool Stack<T>::IsEmpty()
{
	return size == 0 && head == nullptr;
}
template<class T>
void Stack<T>::ToNull()
{
	while (!IsEmpty()) {
		Pop();
	}
}
template<class T>
void Stack<T>::Show()
{
	Node* tmp = head;
	if (tmp == nullptr) cout << "Empty" << endl;
	while (tmp != nullptr) {
		tmp->element.Show();
		cout << endl;
		tmp = tmp->next;
	}
}
template<class T>
void Stack<T>::SaveInFile(ofstream& f)
{
	for (int i = 0; i < size; i++) {
		this->operator[](i).SaveInFile(f);
	}
}
template<class T>
void Stack<T>::SortByCapacity()
{
	for (int i = 0; i < size - 1; i++) {
		for (int j = i + 1; j < size; j++) {
			if (this->operator[](i).GetCapacity() > this->operator[](j).GetCapacity()) {
				swap(this->operator[](i), this->operator[](j));
			}
		}
	}
}
template<class T>
void Stack<T>::Sort()
{
	for (int i = 0; i < size - 1; i++) {
		for (int j = i + 1; j < size; j++) {
			if (this->operator[](i) > this->operator[](j)) {
				swap(this->operator[](i), this->operator[](j));
			}
		}
	}
}
template<class T>
void Stack<T>::SortByWeight()
{
	for (int i = 0; i < size - 1; i++) {
		for (int j = i + 1; j < size; j++) {
			if (this->operator[](i).GetWeight() > this->operator[](j).GetWeight()) {
				swap(this->operator[](i), this->operator[](j));
			}
		}
	}
}
template<class T>
void Stack<T>::SearchByCapicity(int capacity)
{
	for (int i = 0; i < size; i++) {
		if (capacity == this->operator[](i).GetCapacity()) {
			this->operator[](i).Show();
		}
	}
}
template<class T>
void Stack<T>::SearchByWeight(double weight) {
	for (int i = 0; i < size; i++) {
		if (weight == this->operator[](i).GetWeight()) {
			this->operator[](i).Show();
		}
	}
}
template<class T>
T& Stack<T>::operator[](int index)
{
	Node* tmp = head;
	for (int i = 0; i < index; i++) {
		tmp = tmp->next;
	}
```

Рис.П2.12. Текст класса Stack

Приложение 3.

Руководство пользователя

1. Общие сведения о программе

Исполнительный файл: sem2.exe. Данная программа предназначена для работы с запоминающими устройствами.

2. Описание установки

Установка программы не требуется.

3. Описание запуска

Двойным нажатием левой кнопки мыши запустить файл sem2.exe

4. Инструкция по работе

4.1 Описание возможностей программы

В открывшемся консольном окне ввести номер действия, которое вы желаете выполнить (рис. П3.1). Номера действий пронумерованы по левому краю.

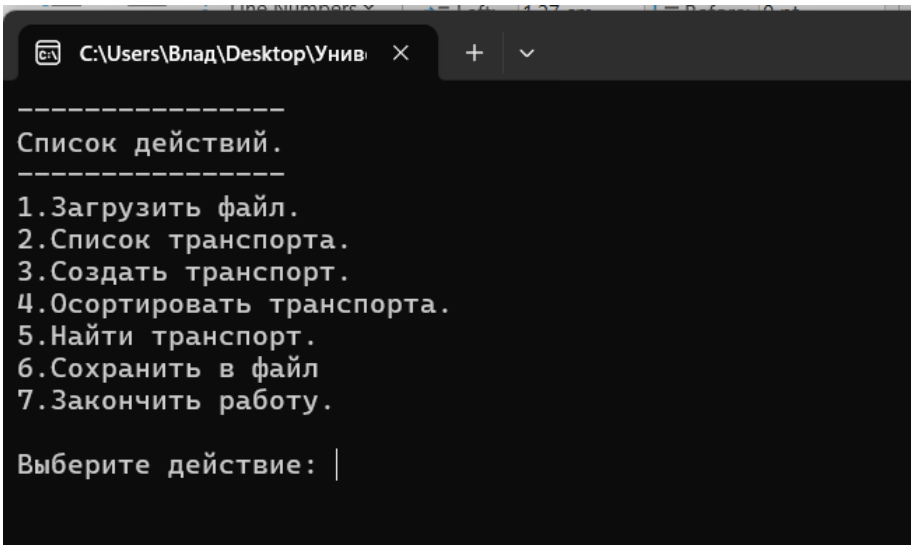

Рис. П3.1. Консольное окно при запуске приложения

Для того, чтобы выгрузить информацию из файла, необходимо ввести в консоль 1. Далее программа предложит ввести путь до файла и имя файл вместе с его расширением, из которого необходимо считать информацию. После программа вновь предложит список возможных действий (рис. П3.1).

Для того, чтобы увидеть весь перечень транспортов, пользователю необходимо ввести в консоль число 2(рис. П3.2).

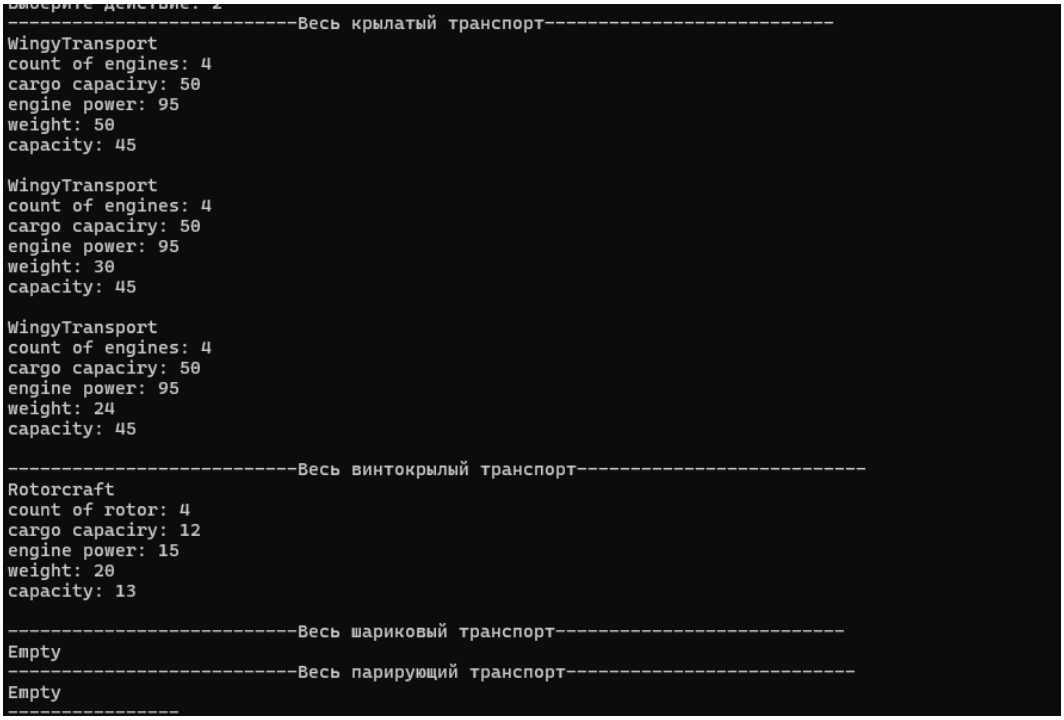

Рис.П3.2. Консольное окно при выводе элементов

После пользователю снова будет предложен перечень возможных действий. Можно создать новый объект, введя в консоль цифру 3. Для создания объекта, пользователю необходимо выбрать класс этого объекта и заполнить поля(рис.П3.3).

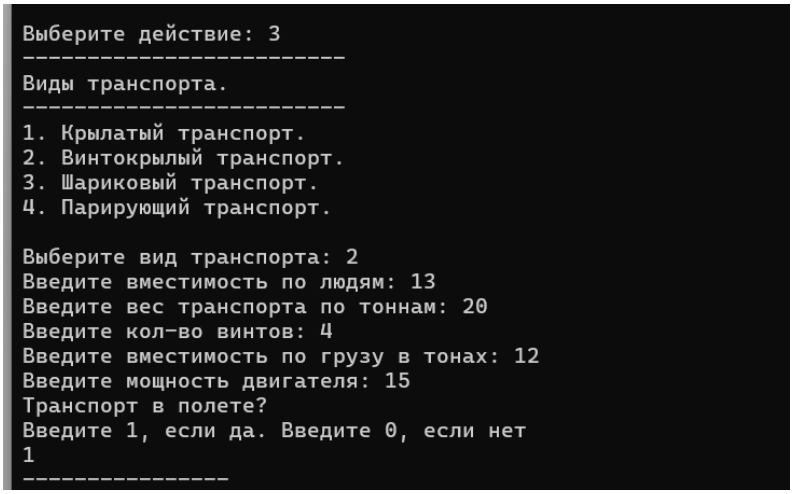

Рис.П3.3. Создание нового объекта

Этот объект будет сохранен (но не в файл) и добавлен ко всем остальным.

Также пользователь может отсортировать транспорты по параметрам весу и вместимости по людям и по всему сразу, выбрав цифру 4 (рис.П3.4).

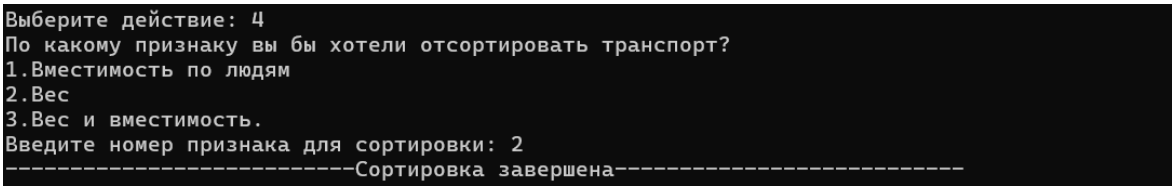

Рис.П3.4 Сортировка элементов

Для того, чтобы реализовать поиск элементов, необходимо ввести цифру 5. Пользователю будет предоставлен выбор параметров, по которым будет производиться поиск(рис.П3.5).

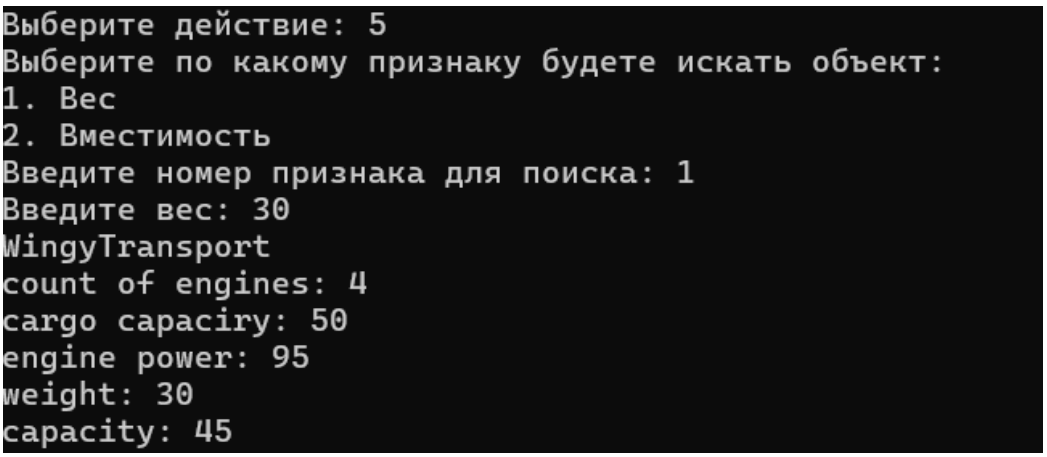

Рис.П3.5 Поиск элементов

Также пользователь может сохранить транспорты в файл. Ему на выбор будет предоставлено 2 варианта: сохранить все элементы в 1 файл или отсортировать по классам в 4 файла. И в том, и в ином случае пользователю будет необходимо ввести имена файлов, куда будут сохранены элементы. Если таких файлов прежде не было в папке с программой, они будут созданы автоматически(рис.П3.6).

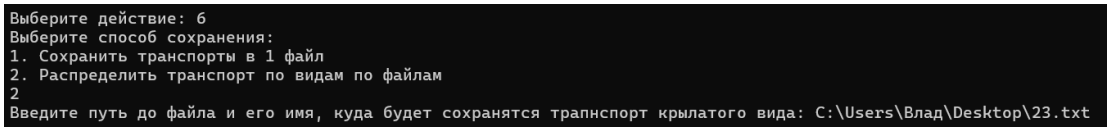

Рис.П3.6 Сохранение в файл

Также пользователь может завершить работу программы нажав на 7.