

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 дисциплине «Объектно-ориентированное программирование»

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

Выполнил студент группы 1ПИб-02-2оп-22

группа

направления подготовки (специальности)

09.03.04 Программная инженерия

шифр, наименование

Овчинников

Максим

Владимирович

фамилия, имя, отчество

Руководитель

Матевосян Р.А.

фамилия, имя, отчество

Аспирант

должность

Дата представления работы

«_____» _____ 20 ____ г.

Заключение о допуске к защите

Оценка _____, _____

количество баллов

Подпись преподавателя _____

Оглавление

Введение	3
1. Объектно-ориентированный анализ предметной области	4
2. Проектирование классов	6
3. Логическая структура программы	19
4. Модульная структура программы	20
5. Тестирование программы	21
Заключение	23
Список литературы	24
Приложение 1. Техническое задание	25
Приложение 2. Руководство пользователя	32
Приложение 3. Текст программы	37

Введение

Объектно-ориентированное программирование (ООП) – это парадигма программирования, которая представляет собой способ организации кода в виде взаимосвязанных объектов, каждый из которых обладает определенными свойствами и методами. Оно основывается на моделировании объектов и процессов реального мира, позволяя разработчикам создавать программы, которые легко масштабируются, поддерживаются и модифицируются [1].

Центральным понятием в ООП является объект. Объекты представляют конкретные экземпляры классов и обладают своим состоянием (данными) и поведением (методами). Классы, которые используются для определения структуры объектов, описывают состояние и поведение этих объектов. Состояние определяется набором переменных, называемых полями или атрибутами, а поведение - набором функций, называемых методами [2].

Принципы ООП включают в себя инкапсуляцию, наследование, полиморфизм и абстракцию.

Инкапсуляция позволяет скрыть детали реализации объекта от внешнего мира и предоставить только необходимый интерфейс взаимодействия. Наследование позволяет создавать новые классы на основе уже существующих, что способствует повторному использованию кода и упрощает его поддержку.

Полиморфизм позволяет использовать один и тот же метод или функцию для объектов разных классов, что упрощает обработку различных типов данных.

Абстракция позволяет сократить сложность кода путем выделения основных характеристик и игнорирования деталей, не относящихся к текущей задаче.

1. Объектно-ориентированный анализ предметной области

В рамках курсовой работы требуется разработать программный продукт, предназначенный для работы с объектами в сфере водного транспорта. Основной задачей является создание иерархии классов, связанных с этой областью, при помощи применения наследования.

Водный транспорт представляет собой систему, используемую для перемещения пассажиров, грузов и почты по водным маршрутам [4].

Классы, которые могут быть созданы в рамках данной иерархии, включают парусные суда (Sailboat), катамараны (Catamaran), круизные лайнеры (CruiseLiner) и атомные подводные лодки (NuclearSubmarine). Эти виды транспорта могут быть как с двигателем (EnginePoweredWaterTransport), так и без него.

Для хранения объектов классов используется шаблонный контейнер, исполненный в виде динамического вектора (MyVector<T>). Для хранения указателей на водный транспорт используется очередь (TmpQueue), наследуемая от очереди, хранящего указатели на void(VoidQueue).

Все классы взаимосвязаны между собой (рис. 1).

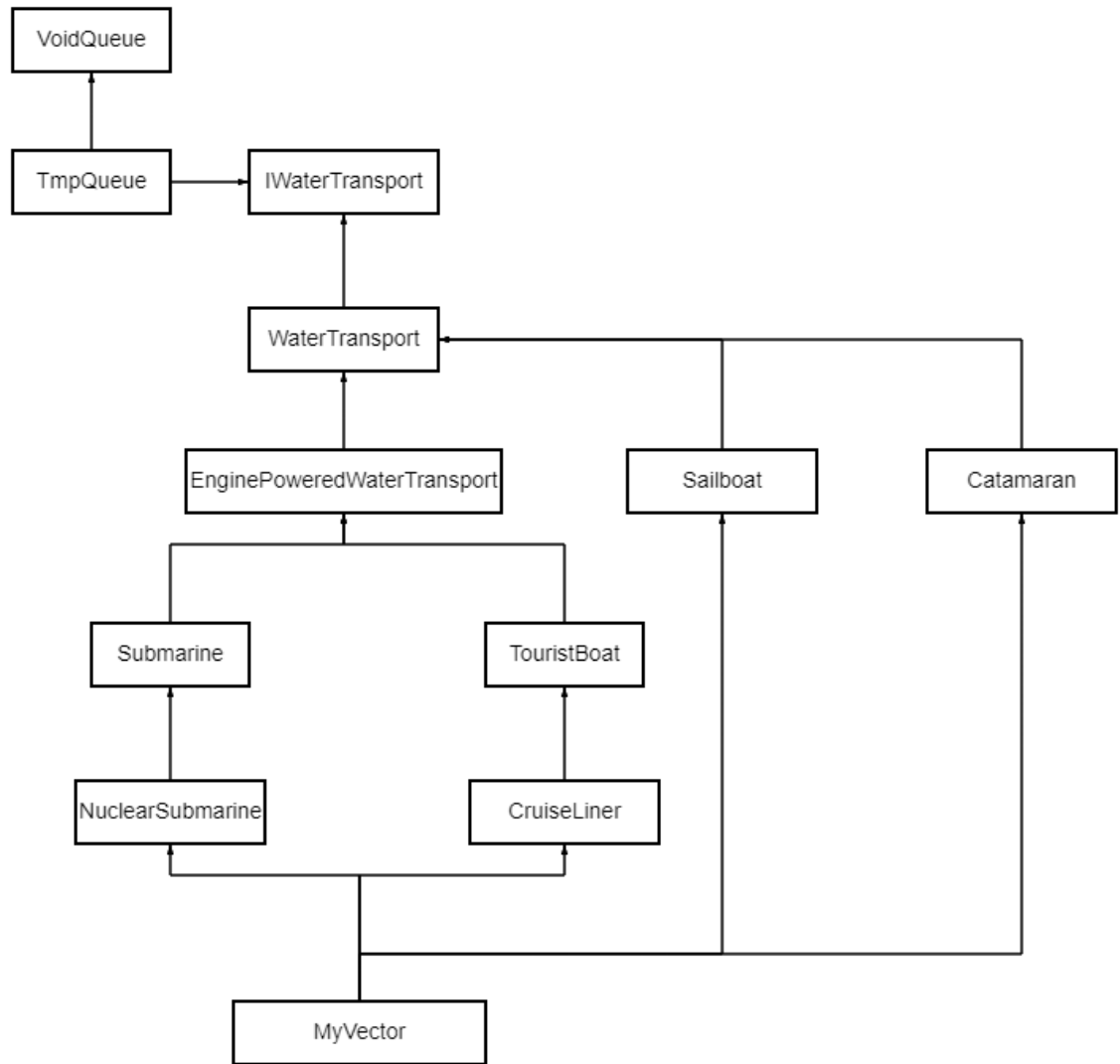


Рис. 1. Диаграмма классов

2. Проектирование классов

Классы водного транспорта поддерживают функции вывода содержимого полей в консоль и в файл, также с содержимым полей в файл выводятся 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 1-10.

Таблица 1

Функции интерфейса IWaterTransport

Функция	Назначение функции
virtual std::string GetName()	Возвращает имя объекта
virtual double GetMaxSpeed()	Возвращает значения поля максимальная скорость объекта
virtual int GetPassengerCapacity()	Возвращает значения поля вместимость пассажиров объекта
virtual void Print()	Выводит информацию о объекте на консоль
virtual void PrintToFile(std::ofstream& outputFile)	Записывает информацию о объекте в файл
virtual void ReadFromFile(std::ifstream& inFile)	Считывает информацию об объекте из файла

Функции класса WaterTransport

Функция	Назначение функции
WaterTransport()	Конструктор по умолчанию, инициализирует объект
WaterTransport(std::string name, double maxSpeed, int passengerCapacity)	Конструктор с параметрами, инициализирует объект с указанными значениями
void SetMaxSpeed(double maxSpeed)	Устанавливает значение поля максимальная скорость
double GetMaxSpeed()	Возвращает значение поля максимальную скорость
void SetPassengerCapacity(int passengerCapacity)	Устанавливает значение поля вместимость пассажиров
int GetPassengerCapacity()	Возвращает значение поля вместимость пассажиров
void SetName(std::string name)	Устанавливает имя объекта
std::string GetName()	Возвращает имя объекта
void Print()	Выводит информацию о объекте в консоль
void PrintToFile(std::ofstream& outputFile)	Записывает информацию о объекте в файл
void ReadFromFile(std::ifstream& inFile)	Считывает информацию об объекте из файла
~WaterTransport()	Деструктор, освобождает выделенные ресурсы

Функции класса EnginePoweredWaterTransport

Функция	Назначение функции
EnginePoweredWaterTransport()	Конструктор по умолчанию, инициализирует объект
EnginePoweredWaterTransport(std::string name, double maxSpeed, int passengerCapacity, int powerEngine)	Конструктор с параметрами, инициализирует объект с указанными значениями
void SetPowerEngine(int powerEngine)	Устанавливает значение поля мощность двигателя
int GetPowerEngine()	Возвращает значение поля мощность двигателя
void Print()	Выводит информацию о объекте в консоль
void PrintToFile(std::ofstream& outputFile)	Записывает информацию о объекте в файл
void ReadFromFile(std::ifstream& inFile)	Считывает информацию об объекте из файла
~EnginePoweredWaterTransport()	Деструктор, освобождает выделенные ресурсы

Функции класса Catamaran

Функция	Назначение функции
Catamaran()	Конструктор по умолчанию, инициализирует объект
Catamaran(std::string name, double maxSpeed, int passengerCapacity, int numberOfPedals)	Конструктор с параметрами, инициализирует объект с указанными значениями
void SetNumberOfPedals(int numberOfPedals)	Устанавливает значение поля количество педалей
int GetNumberOfPedals()	Возвращает значение поля количество педалей
void Print()	Выводит информацию о объекте в консоль
void PrintToFile(std::ofstream& outputFile)	Записывает информацию о объекте в файл
void ReadFromFile(std::ifstream& inFile)	Считывает информацию об объекте из файла
~Catamaran()	Деструктор, освобождает выделенные ресурсы

Функции класса Sailboat

Функция	Назначение функции
Sailboat();	Конструктор по умолчанию, инициализирует объект
Sailboat(std::string name, double maxSpeed, int passengerCapacity, int sailArea);	Конструктор с параметрами, инициализирует объект с указанными значениями
void SetSailArea(int sailArea);	Устанавливает площадь паруса
int GetSailArea();	Возвращает площадь паруса
void Print();	Выводит информацию о объекте в консоль
void PrintToFile(std::ofstream& outputFile)	Записывает информацию о объекте в файл
void ReadFromFile(std::ifstream& inFile)	Считывает информацию об объекте из файла
~EnginePoweredWaterTransport()	Деструктор, освобождает выделенные ресурсы

Функции класса Submarine

Функция	Назначение функции
Submarine()	Конструктор по умолчанию, инициализирует объект
Submarine(std::string name, double maxSpeed, int passengerCapacity, int powerEngine, double divingDepth)	Конструктор с параметрами, инициализирует объект с указанными значениями
void SetDivingDepth(double divingDepth)	Устанавливает значение поля глубина погружения
double GetDivingDepth()	Возвращает значение поля глубина погружения
void Print()	Выводит информацию о объекте в консоль
void PrintToFile(std::ofstream& outputFile)	Записывает информацию о объекте в файл
void ReadFromFile(std::ifstream& inFile)	Считывает информацию об объекте из файла
~Submarine()	Деструктор, освобождает выделенные ресурсы

Функции класса TouristBoat

Функция	Назначение функции
TouristBoat()	Конструктор по умолчанию, инициализирует объект
TouristBoat(std::string name, double maxSpeed, int passengerCapacity, int powerEngine, int ticketPrice)	Конструктор с параметрами, инициализирует объект с указанными значениями
void SetTicketPrice(int ticketPrice)	Устанавливает значение поля стоимость билета
int GetTicketPrice ()	Возвращает значение поля стоимость билета
void Print()	Выводит информацию о объекте в консоль
void PrintToFile(std::ofstream& outputFile)	Записывает информацию о объекте в файл
void ReadFromFile(std::ifstream& inFile)	Считывает информацию об объекте из файла
~ TouristBoat()	Деструктор, освобождает выделенные ресурсы

Функции класса CruiseLiner

Функция	Назначение функции
1	2
CruiseLiner ()	Конструктор по умолчанию
CruiseLiner(std::string name, double maxSpeed, int passengerCapacity, int powerEngine, int ticketPrice, int heightWaterSlides)	Конструктор с параметрами
void SetHeightWaterSlides(int heightWaterSlides)	Устанавливает значение поля высота водных горок
int GetHeightWaterSlides ()	Возвращает значение поля высота водных горок
void Print()	Выводит информацию о объекте в консоль
void PrintToFile(std::ofstream& outputFile)	Записывает информацию о объекте в файл
void ReadFromFile(std::ifstream& inFile)	Считывает информацию об объекте из файла
~CruiseLiner ()	Деструктор, освобождает выделенные ресурсы

Функции класса NuclearSubmarine

Функция	Назначение функции
NuclearSubmarine()	Конструктор по умолчанию, инициализирует объект
NuclearSubmarine(std::string name, double maxSpeed, int passengerCapacity, int powerEngine, double divingDepth, double powerAtomicBomb)	Конструктор с параметрами, инициализирует объект с указанными значениями
void SetPowerAtomicBomb(double powerAtomicBomb);	Устанавливает значение поля мощность ядерного заряда
double GetPowerAtomicBomb();	Возвращает значение поля мощность ядерного заряда
void Print()	Выводит информацию о объекте в консоль
void PrintToFile(std::ofstream& outputFile)	Записывает информацию о объекте в файл
void ReadFromFile(std::ifstream& inFile)	Считывает информацию об объекте из файла
~NuclearSubmarine ()	Деструктор, освобождает выделенные ресурсы

Класс шаблонного вектора предоставляет возможности для работы с коллекцией объектов любого типа данных. Он позволяет добавлять и удалять элементы, получать доступ к элементам по индексу, определять размер и проверять на пустоту, сортировать элементы и выполнять поиск элементов в соответствии с заданными критериями.

Перечень функций данного класса находится в таблице 10.

Функции класса MyVector

Функция	Назначение функции
MyVector()	Конструктор по умолчанию, создаёт пустой вектор
T& At(size_t index)	Возвращает ссылку на элемент вектора по заданному индексу
void Push_back(const T& value)	Добавляет элемент в конец вектора
void Pop_back()	Удаляет последний элемент из вектора
T& Front()	Возвращает ссылку на первый элемент вектора
T& Back()	Возвращает ссылку на последний элемент вектора
size_t Size() const	Возвращает количество элементов вектора
bool Empty() const	Проверяет, пуст ли вектор
typename std::vector<T>::iterator Begin()	Возвращает итератор, указывающий на начало вектора
typename std::vector<T>::iterator End()	Возвращает итератор, указывающий на конец вектора
template <typename Comparator> void Sort(Comparator comp)	Сортирует элементы вектора с использованием заданного компаратора
template <typename Predicate> std::vector<T> FindInRange(Predicate pred) const	Возвращает вектор элементов, удовлетворяющих заданному предикату
~MyVector()	Деструктор, освобождает память, занимаемую вектором

Контейнерный обобщённый класс очередь (TmpQueue) поддерживает добавление и удаление элементов, проверку на пустоту, получение размера очереди, доступ к элементам по индексу, вывод содержимого в файл и чтение из файла, а также печать объектов по имени. Большинство вышеперечисленных функций этот класс наследует от класса VoidQueue.

Перечень функций этих классов находится в таблицах 11-12.

Таблица 11

Функции класса VoidQueue

Функция	Назначение функции
void Enqueue(void* item)	Добавляет элемент в конец очереди
void* Dequeue()	Удаляет и возвращает элемент из начала очереди. Возвращает nullptr, если очередь пуста
bool IsEmpty() const	Проверяет, пуста ли очередь
size_t Size() const	Возвращает количество элементов в очереди
void* At(size_t index) const	Возвращает элемент по заданному индексу. Возвращает nullptr, если индекс выходит за границы очереди
void OutputToFile(const string& filename)	Записывает содержимое очереди в файл
void InputFromFile(const string& filename)	Читает данные из файла и добавляет элементы в очередь
void PrintByName(const string& name) const	Выводит информацию об элементах с указанным именем
~VoidQueue()	Деструктор, очищает очередь

Функции класса TmpQueue

Функция	Назначение функции
void Enqueue(T* item)	Добавляет элемент в конец очереди
T* Dequeue()	Удаляет и возвращает элемент из начала очереди. Возвращает nullptr, если очередь пуста
bool IsEmpty() const	Проверяет, пуста ли очередь
size_t Size() const	Возвращает количество элементов в очереди
T* At(size_t index) const	Возвращает элемент по заданному индексу. Возвращает nullptr, если индекс выходит за границы очереди
void OutputToFile(const string& filename)	Записывает содержимое очереди в файл
void InputFromFile(const string& filename)	Читает данные из файла и добавляет элементы в очередь
void PrintByName(const string& name) const	Выводит информацию об элементах с указанным именем
~TmpQueue()	Деструктор, очищает очередь

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

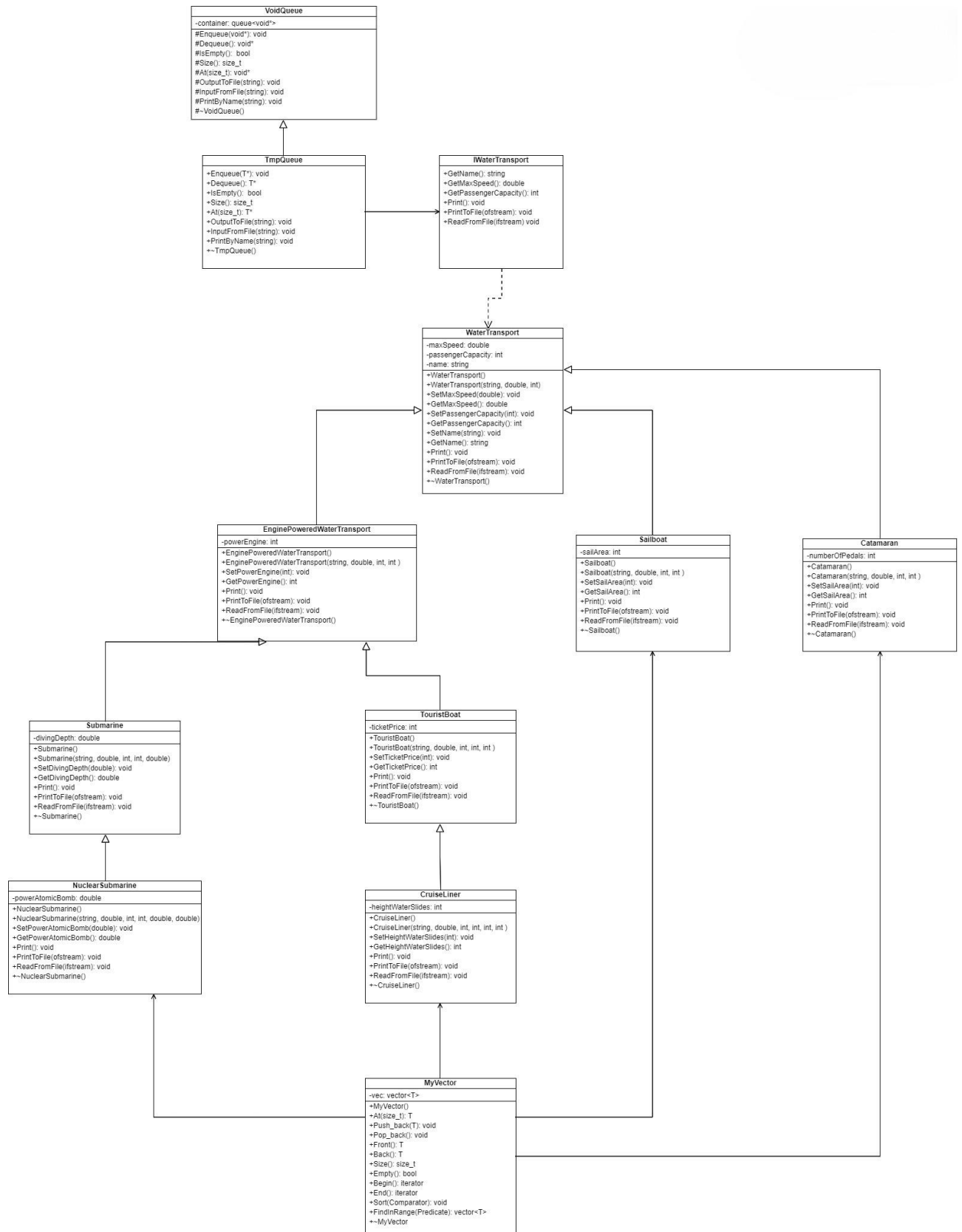


Рис. 2. Полная диаграмма классов

3. Логическая структура программы

При запуске программы выполняется модуль main. Дальнейшее взаимодействие с программой зависит от действий пользователя. Пользователь может загрузить информацию из файла или ввести данные вручную через консоль, после чего информация будет распределена по контейнерам. В программе используется четыре динамических вектора, которые содержат объекты классов водного транспорта, а также очередь, которая хранит указатели на объекты, реализующие интерфейс IWaterTransport.

Работа программы производится взаимодействием классов между собой (рис. 3).

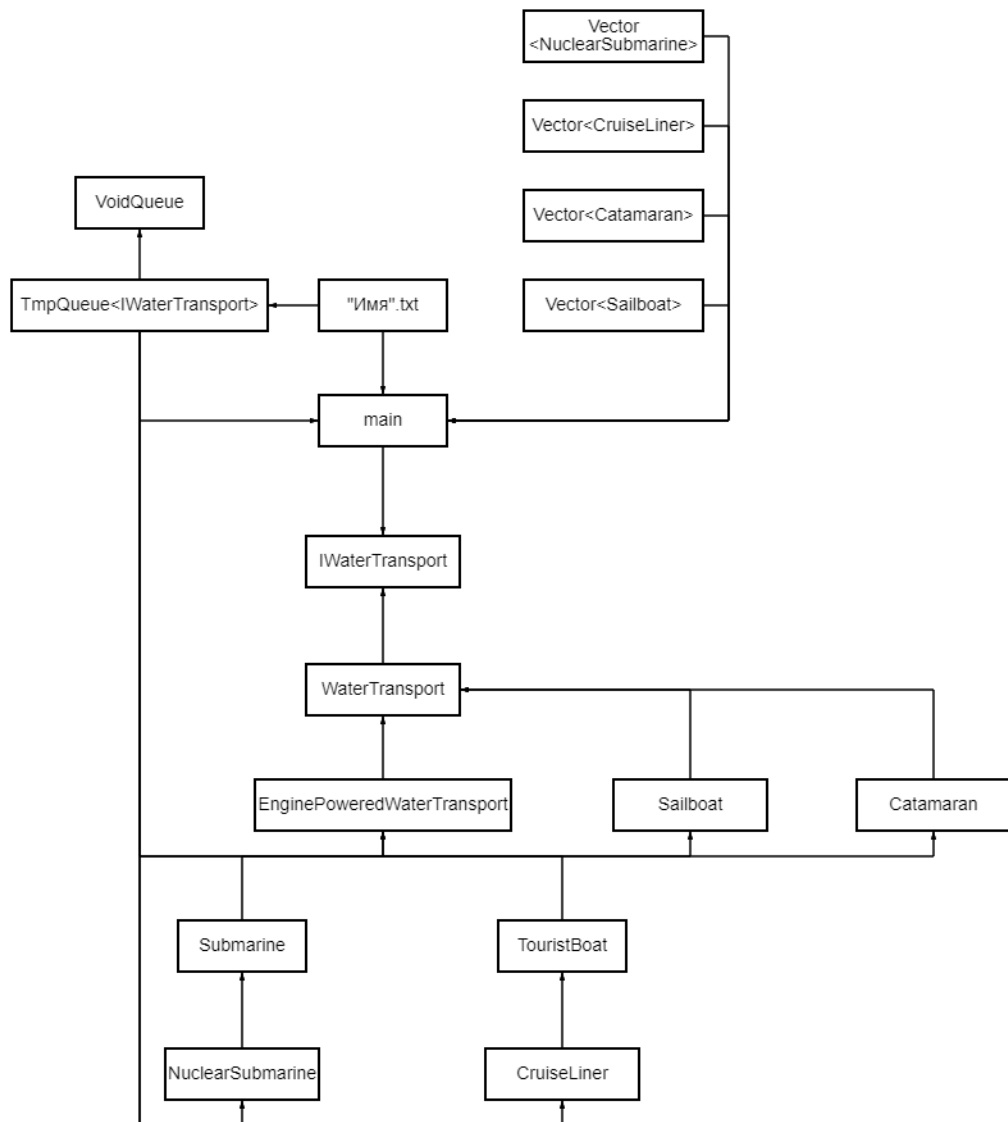


Рис. 3. Логическая схема

4. Модульная структура программы

Под модульной структурой подразумевается построение программы из отдельных функциональных модулей (фрагментов, сегментов, подпрограмм). Эти модули могут выполнять различные функции и применяться в самых разнообразных проектах. При разработке программы важно стремиться к оптимизации исходного кода: однотипные фрагменты кода следует оформлять в виде модулей, которые можно будет использовать при необходимости.

Взаимодействие между модулями и файлами программы представлено на рис. 4.

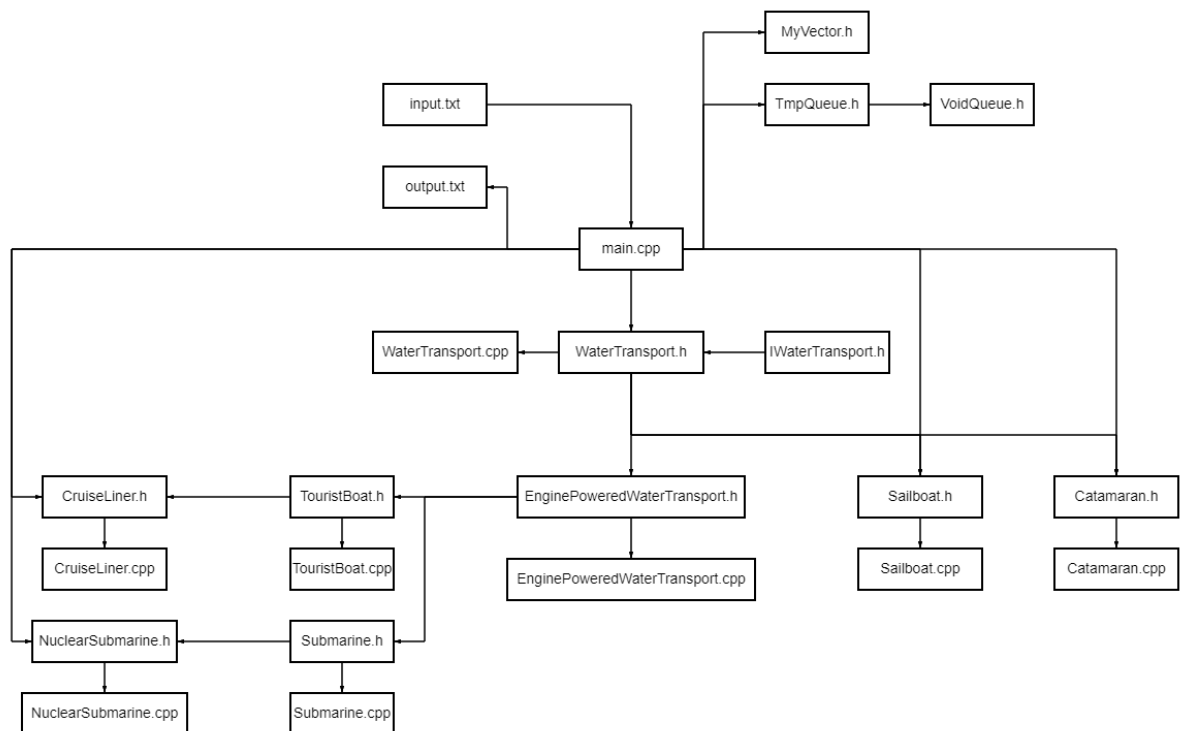


Рис.4. Модульная диаграмма

5. Тестирование программы

В приведенных ниже таблицах 12-14 представлены результаты тестирования программы.

Таблица 12

Протокол тестирования классов

Дата тестирования	Класс	Кто проводил тестирование	Описание теста	Результаты тестирования
30.05.24	Catamaran	Овчинников М.В.	Создание объектов и действия с ними	Успех
30.05.24	Sailboat	Симаньков А.Е.	Создание объектов и действие с ними	Успех
30.05.24	NuclearSubmarine	Зернов В.А.	Создание объектов и действия с ними	Успех
30.05.24	CruiseLiner	Гончаров Е. Д.	Создание объектов и действия с ними	Успех
01.06.2024	MyVector	Симаньков А.Е.	Создание вектора, заполнение его объектами	Успех
01.06.2024	MyVector	Овчинников М.В.	Сортировка вектора и поиск элементов в диапазоне	Успех
02.06.2024	TmpQueue	Зернов В.А.	Заполнение очереди и действия с ним	Ошибка при загрузке из файла
02.06.2024	TmpQueue	Овчинников М.В.	Заполнение очереди	Успех

Таблица 13

Протокол тестирования внешних функций

Дата тестирования	Функция	Кто проводил тестирование	Описание теста	Результаты тестирования
03.06.2024	main	Овчинников М.В.	Ввод данных из файла	Успех
			Ввод данных с клавиатуры в консоли	Успех
			Отображение данных	Успех
			Сохранение данных	Успех
			Сортировка и поиск данных	Успех

Таблица 14

Протокол тестирования по техническому заданию

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

Заключение

В результате выполнения курсовой работы было создано программное обеспечение для управления объектами в области "Водный транспорт". Разработанная программа способна хранить объекты, введенные пользователем, в динамическом векторе и очереди, а также обрабатывать их. Были внедрены механизмы для обработки исключений.

В процессе работы были приобретены дополнительные навыки программирования на языке C++, изучены методы работы с родственными типами и принципы использования void-указателей, а также укреплены знания в области объектно-ориентированного программирования.

Список литературы

1. Объектно-ориентированное программирование (C++) [Электронный ресурс]. Дата доступа 05.06.2024. Ссылка: <https://mariohuq.github.io/polubenceva-oop/>
2. Лафоре, Р. Объектно-ориентированное программирование в C++ / Р. Лафоре. - СПб.: Питер, 2019. - 928 с.
3. Наследование (C++) [Электронный ресурс]. Ссылка: <https://learn.microsoft.com/ru-ru/cpp/cpp/inheritance-cpp?view=msvc-170>
4. Водный транспорт – Wikipedia [Электронный ресурс]. Дата доступа 05.06.2024. Ссылка: https://ru.wikipedia.org/wiki/Водный_транспорт
5. Ершов, Е.В. Методика и организация самостоятельной работы: учебное пособие / Ершов Е.В., Виноградова Л.Н., Селивановских В.В. – Череповец: ЧГУ, 2015.

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)

Объектно-ориентированное программирование

(наименование дисциплины в соответствии с учебным планом)

УТВЕРЖДАЮ

Зав. кафедрой МПО ЭВМ,
д.т.н., профессор Ершов Е.В.
«___» апреля 2023 г.

Объектно-ориентированное программирование на языке C++
Техническое задание на курсовую работу
Листов 8

Руководитель: Матевосян Р.А.
Исполнитель: студент гр. 1ПИБ-02-2оп-22
Овчинников Максим Владимирович

2024 год

Введение

Курсовая работа посвящена разработке программы на языке C++ и созданию иерархии родственных типов, где корневым классом является абстрактный базовый класс (класс-интерфейс). Целью является моделирование и обработка данных в выбранной предметной области с использованием набора 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, корневым классом которой будет абстрактный базовый класс (интерфейс), для моделирования и обработки данных в предметной области с использованием набора отложенных методов. В данном случае рассматривается вариант А30 «Водный транспорт».

Создать обобщенный (void*) контейнерный класс (базовый) и производный класс-шаблон для хранения указателей на абстрактный базовый класс-интерфейс с использованием закрытого наследования, согласно варианту В6 «Очередь».

Для хранения объектов каждого производного класса использовать структуру данных согласно варианту С2 «Динамический вектор».

Реализовать файловый ввод/вывод, а также ввод данных с клавиатуры и вывод данных на дисплей. Обеспечить обработку различных исключительных ситуаций.

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

Программу следует запускать на компьютере в закрытом помещении с обогревом и (или) охлаждением при следующих условиях окружающей среды:

- Уровень атмосферного давления: от 70 кПа до 106 кПа;
- Предел абсолютной влажности воздуха должен быть равен 25 гр/м;
- Температура окружающего воздуха от +16°C до +28°C;
- Запыленность воздуха не более 0,75 мг/м³;
- Знание основ работы в операционной системе Windows.

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

Обычно требования к маркировке и упаковке не применимы к программе, поскольку она является цифровым продуктом, распространяемым в электронном формате.

3.7. Требования к транспортированию и хранению

Для правильной работы программы необходимо расположить соответствующие файлы на флеш-накопителе или в памяти компьютера.

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. X. Название В» приложениях: «Рис. П.3. Название»
Таблицы	Подписывается: над таблицей, выравнивание по правому: «Таблица X». 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
Определение темы для курсовой работы	20.03.2024	Утверждена тема разработки	Выполнено
Оформление технического задания	15.05.2024	Выполнено е тех. задания	Выполнено
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 и 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	06.06.2023	Курсовая работа защищена	

Приложение 2. Руководство пользователя

Руководство пользователя

1. Общие сведения о программе

Функциональным назначением программы является представление предметной области «Водный транспорт».

2. Описание установки и запуска

Для запуска приложения следует запустить исполняемый файл «CourseWork.exe». Оно не требует предварительной распаковки или установки.

3. Инструкции по работе

После запуска программы откроется консольное окно пронумерованного списка возможных действий (рис. П2.1).

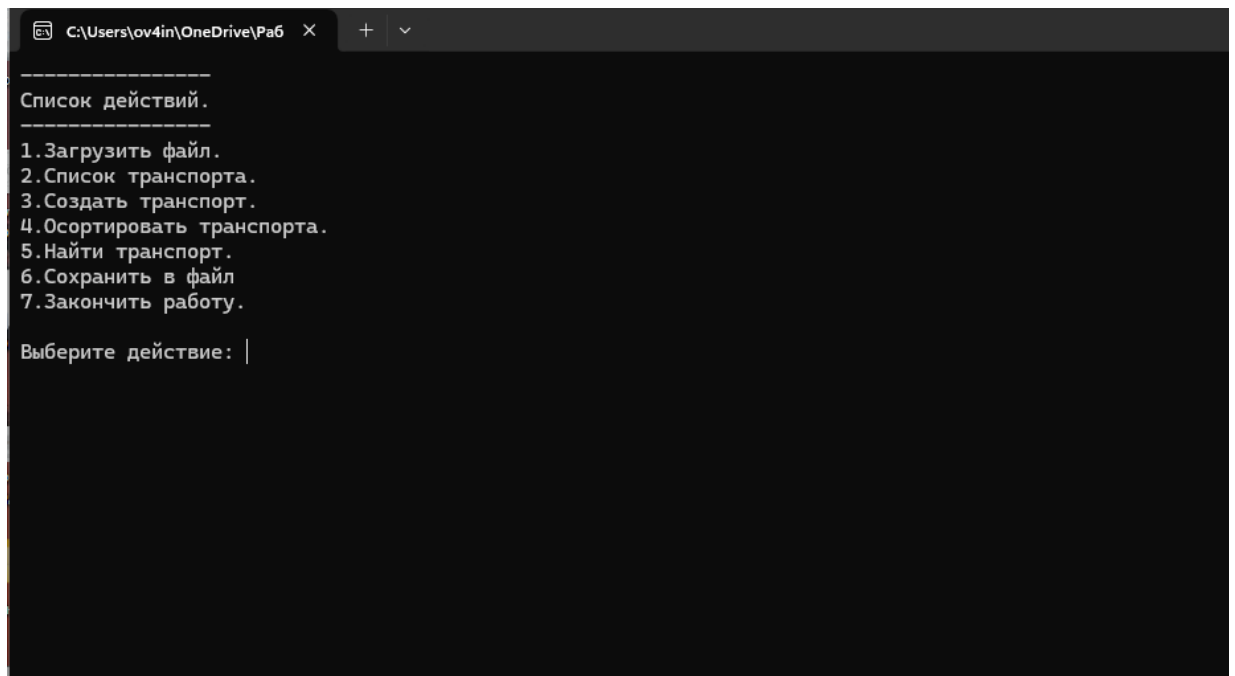
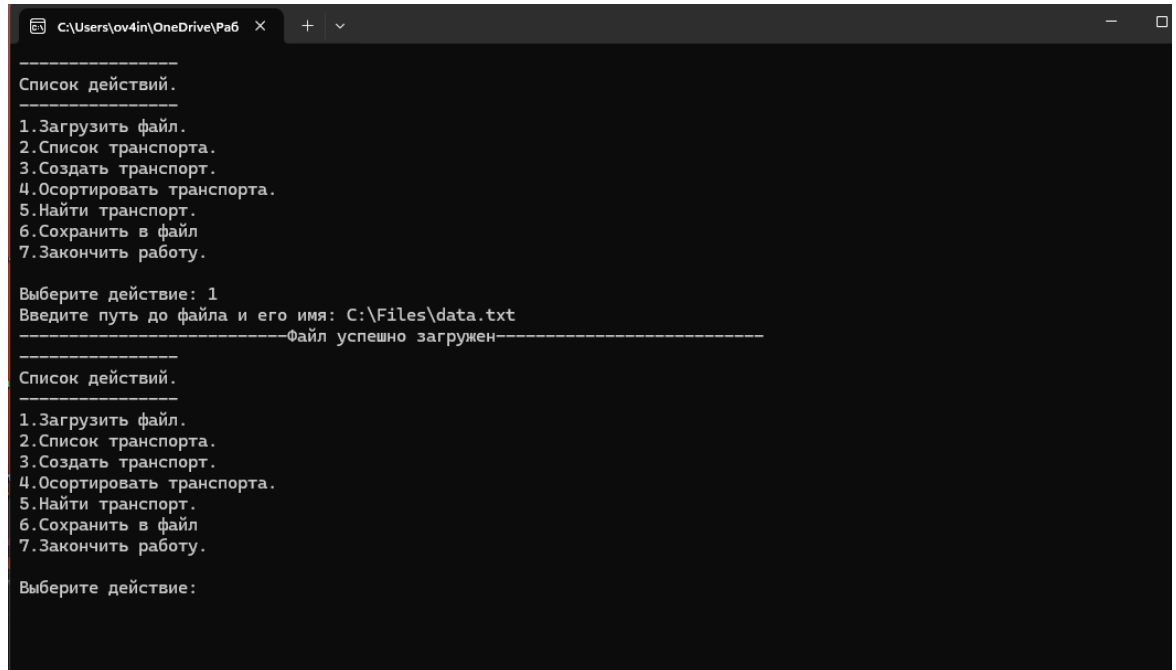


Рис. П2.1. Консольное окно при запуске приложения

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 будет выведено сообщение об успешной загрузке или об ошибке ненайденного файла (рис. П2.2).



```
C:\Users\ov4in\OneDrive\Раб X + -
-----
Список действий.
-----
1.Загрузить файл.
2.Список транспорта.
3.Создать транспорт.
4.Осорттировать транспорта.
5.Найти транспорт.
6.Сохранить в файл
7.Закончить работу.

Выберите действие: 1
Введите путь до файла и его имя: C:\Files\data.txt
-----Файл успешно загружен-----
-----
Список действий.
-----
1.Загрузить файл.
2.Список транспорта.
3.Создать транспорт.
4.Осорттировать транспорта.
5.Найти транспорт.
6.Сохранить в файл
7.Закончить работу.

Выберите действие:
```

Рис. П2.2. Консольное окно при выгрузке из файла

Для того, чтобы увидеть весь список транспортов, который хранится в контейнерах, пользователю необходимо ввести в консоль 2 (рис. П2.3).

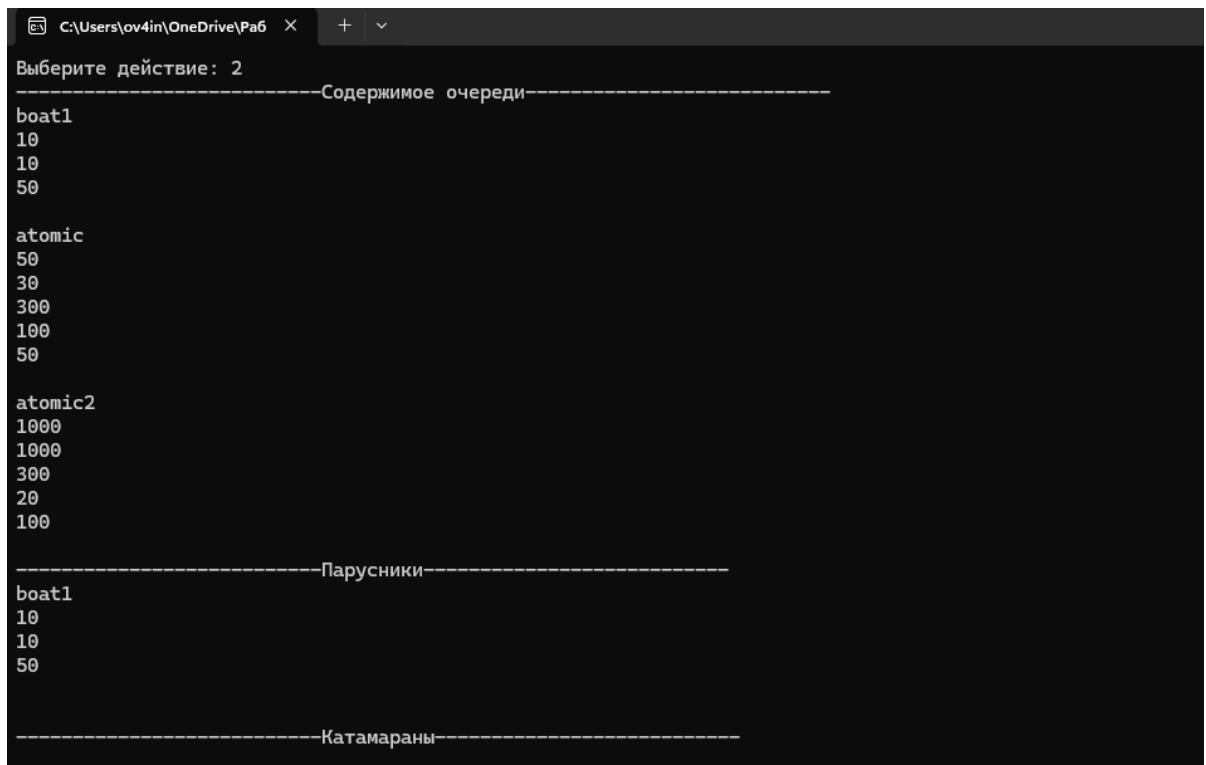


Рис. П2.3. Список транспорта

Можно создать новый объект, который будет помещён к остальным объектам в контейнеры, введя в консоль 3. Для создания объекта, пользователю необходимо выбрать класс этого объекта и заполнить поля с помощью ввода с клавиатуры (рис. П3.4).

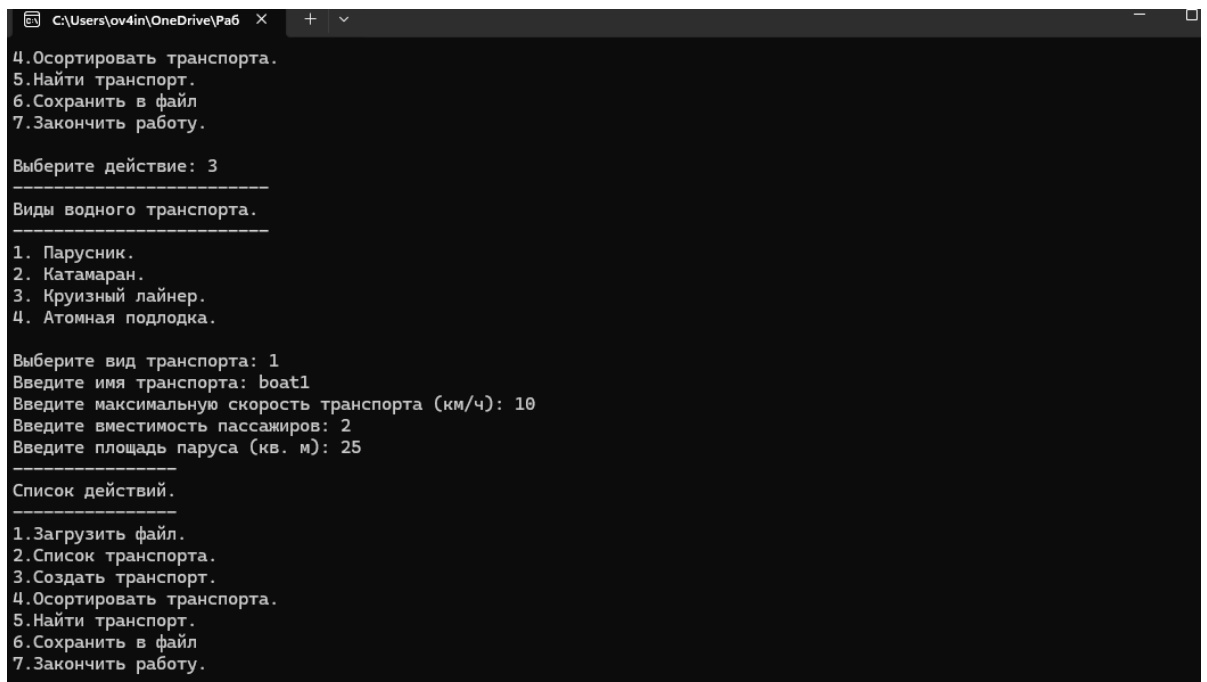


Рис. П2.4. Создание нового объекта

Пользователь может отсортировать транспорта по параметрам максимальной скорости или вместимости пассажиров, введя команду 4 (рис. П2.5).

```

Выберите действие: 4
По какому признаку вы бы хотели отсортировать транспорт?
1.Скорость
2.Вместимость пассажиров
Введите номер признака для сортировки: 1
1. Сортировать по возрастанию
2. Сортировать по убыванию
1
-----Сортировка завершена-----
-----
Список действий.
-----
1.Загрузить файл.
2.Список транспорта.
3.Создать транспорт.
4.Осортировать транспорта.
5.Найти транспорт.
6.Сохранить в файл

```

Рис. П2.5 Сортировка элементов

Для того чтобы осуществить поиск элементов, необходимо ввести команду 5. Пользователю будет представлена возможность осуществить поиск по имени объекта или в диапазоне полей максимальная скорость или вместимость пассажиров (рис. П2.6).

```

Выберите действие: 5
По какому признаку вы бы хотели найти объект?
1.Имя
2.В диапазоне скорости
3.В диапазоне вместимости пассажир:
2
Введите диапазон от min до max.
Введите min.
20
Введите max.
100
atomic
50
30
300
100
50

```

Рис. П2.6 Поиск элементов

Введя команду 6, пользователь сохранит транспорта в файл, путь и название, для которого будет предоставлено указать пользователю (рис. П2.7).

```
-----  
Список действий.  
-----  
1.Загрузить файл.  
2.Список транспорта.  
3.Создать транспорт.  
4.Осортировать транспорта.  
5.Найти транспорт.  
6.Сохранить в файл  
7.Закончить работу.  
  
Выберите действие: 6  
Введите путь до файла и его имя: C:/Files/data.txt  
-----Файл успешно сохранён-----  
-----
```

Рис. П2.7 Сохранение в файл

Пользователь может завершить работу программы введя команду 7.

Приложение 3. Текст программы

Текст файла `main.cpp` представлен на рис. ПЗ.1:

```

#include <iostream>
#include "TmpQueue.h"
#include "MyVector.h"
#include "IWaterTransport.h"
#include "WaterTransport.h"
#include "EnginePoweredWaterTransport.h"
#include "TouristBoat.h"
#include "Sailboat.h"
#include "Catamaran.h"
#include "CruiseLiner.h"
#include "NuclearSubmarine.h"
#include "Windows.h"
#include <fstream>
#include <string>

using namespace std;

template <typename T>
ostream& operator<<(ostream& os, MyVector<T*>& vec)
{
    for (size_t i = 0; i < vec.Size(); ++i) {
        vec.At(i)->Print();
        os << endl;
    }
    return os;
}

void ShowCommands() {
    std::cout << "-----" << endl;
    std::cout << "Список действий." << endl;
    std::cout << "-----" << endl;
    std::cout << "1. Загрузить файл." << endl;
    std::cout << "2. Список транспорта." << endl;
    std::cout << "3. Создать транспорт." << endl;
    std::cout << "4. Осортировать транспорта." << endl;
    std::cout << "5. Найти транспорт." << endl;
    std::cout << "6. Сохранить в файл" << endl;
    std::cout << "7. Закончить работу." << endl << endl;
}

void ShowEntity() {
    std::cout << "-----" << endl;
    std::cout << "Виды водного транспорта." << endl;
    std::cout << "-----" << endl;
    std::cout << "1. Парусник." << endl;
    std::cout << "2. Катамаран." << endl;
    std::cout << "3. Круизный лайнер." << endl;
    std::cout << "4. Атомная подлодка." << endl << endl;
}

}

int main() {
    std::setlocale(LC_ALL, "Russian");
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    string nameFile;
    int action = 0;
    ifstream inputFile;
    string type;
    string name;
    TmpQueue<IWaterTransport> queue;
    MyVector<Sailboat*> sailboatVec;
    MyVector<Catamaran*> catamaranVec;
    MyVector<NuclearSubmarine*> nuclearSubmarineVec;
    MyVector<CruiseLiner*> cruiseLinerVec;

    while (true) {
        ShowCommands();
        std::cout << "Выберите действие: ";

        std::cin >> action;

        if (action == 7) {
            std::cout << "-----"
            -----Завершение работы-----" << endl;
            break;
        }

        switch (action)
        {
            case 1:
                std::cout << "Введите
                путь до файла и его имя: ";
                std::cin >> nameFile;
                inputFile.open(nameFile);
                if (!inputFile.is_open()) {
                    std::cout << "Ошибка: Не удалось открыть
                    файл для чтения.\n";
                    break;
                }

                try
                {
                    queue.InputFromFile(nameFile);
                }
            }
        }
    }
}

```

Рис. ПЗ.1. Продолжение

```

catch
(invalid_argument ex) {

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

}

inputFile.close();

inputFile.open(nameFile);
if (!inputFile.is_open()) {
    std::cout << "Ошибка: Не удалось открыть
файл для чтения.\n";
    break;
}
while (getline(inputFile, type)) {
    if (type == "CATAMARAN") {

        Catamaran* boat = new Catamaran();

        try {
            boat->ReadFromFile(inputFile);
            catamaranVec.Push_back(boat);
        }
        catch (invalid_argument ex) {
            std::cout << "-----"
<< ex.what() << "-----" << endl;
        }
    }
    else if (type == "SAIL_BOAT") {

        Sailboat* boat = new Sailboat();

        try {
            boat->ReadFromFile(inputFile);
            sailboatVec.Push_back(boat);
        }
        catch (invalid_argument ex) {
            std::cout << "-----"
<< ex.what() << "-----" << endl;
        }
    }
    else if (type == "CRUISE_LINER") {

        CruiseLiner* boat = new CruiseLiner();

        try {
            boat->ReadFromFile(inputFile);
            cruiseLinerVec.Push_back(boat);
        }
        catch (invalid_argument ex) {
            std::cout << "-----"
<< ex.what() << "-----" << endl;
        }
    }
    else if (type ==
"NUCLEAR_SUBMARINE") {

        NuclearSubmarine* boat = new
NuclearSubmarine();

        try {
            boat->ReadFromFile(inputFile);
            nuclearSubmarineVec.Push_back(boat);
        }
        catch (invalid_argument ex) {
            std::cout << "-----"
<< ex.what() << "-----" << endl;
        }
    }
    inputFile.close();

    cout << "-----Файл успешно
загружен-----" << endl;
    break;

    case 2:

        std::cout << "-----Содержимое
очереди-----" << endl;
        for (size_t i = 0; i < queue.Size(); ++i) {
            queue.At(i)->Print();

            cout << endl;

            std::cout << "-----
-----Парусники-----" << endl;
            std::cout << sailboatVec
<< endl;
            std::cout << "-----
-----Катамараны-----" << endl;
            std::cout <<
catamaranVec << endl;
            std::cout << "-----
-----Атомные подлодки-----" << endl;
            std::cout <<
nuclearSubmarineVec << endl;
            std::cout << "-----
-----Круизные лайнеры-----" << endl;

```

Рис. ПЗ.1. Продолжение

```

std::cout << cruiseLinerVec << endl;
break;

case 3:
    ShowEntity();
    std::cout << "Выберите
вид транспорта: ";

    std::cin >> action;
    double maxSpeed;
    int passengerCapacity;
    std::cout << "Введите
имя транспорта: ";

    std::cin >> name;
    std::cout << "Введите
максимальную скорость транспорта (км/ч): ";

    std::cin >> maxSpeed;
    std::cout << "Введите
вместимость пассажиров: ";

    std::cin >>
passengerCapacity;

    if (action == 1) {
        int sailArea;
        std::cout <<
"Введите площадь паруса (кв. м): ";
        std::cin >> sailArea;

        try {
            Sailboat* boat = new Sailboat(name, maxSpeed,
passengerCapacity, sailArea);

            sailboatVec.Push_back(boat);

            queue.Enqueue(boat);
        }
        catch
(invalid_argument ex) {

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

        }
        break;
    }

    if (action == 2) {
        int
numberOfPedals;

        std::cout <<
"Введите количество педалей: ";

        std::cin >>
numberOfPedals;

        try {
            Catamaran* boat = new Catamaran(name, maxSpeed,
passengerCapacity, numberOfPedals);

            catamaranVec.Push_back(boat);

            queue.Enqueue(boat);
        }
        catch
(invalid_argument ex) {

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

        }
        break;
    }

    else if (action == 3) {
        int
powerEngine;

        int
ticketPrice;

        int
heightWaterSlides;

        std::cout <<
"Введите мощность двигателя (Л/С): ";

        std::cin >>
powerEngine;

        std::cout <<
"Введите стоимость билета (руб.): ";

        std::cin >>
ticketPrice;

        std::cout <<
"Введите высоту водных горок (м): ";

        std::cin >>
heightWaterSlides;

        try {
            CruiseLiner* boat = new CruiseLiner(name, maxSpeed,
passengerCapacity, powerEngine, ticketPrice, heightWaterSlides);

            cruiseLinerVec.Push_back(boat);

            queue.Enqueue(boat);
        }
        catch
(invalid_argument ex) {

```

Рис. ПЗ.1. Продолжение

```

    }
    break;

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

    }
    break;
}
else if (action == 4) {
    int
powerEngine;
double
divingDepth;
double
powerAtomicBomb;
std::cout <<
"Введите мощность двигателя (Л/С): ";
std::cin >>
powerEngine;
std::cout <<
"Введите глубину погружения (м): ";
std::cin >>
divingDepth;
std::cout <<
"Введите мощность ядерного заряда (Мт): ";
std::cin >>
powerAtomicBomb;

    try {

        NuclearSubmarine* boat = new NuclearSubmarine(name,
maxSpeed, passengerCapacity, powerEngine, divingDepth,
powerAtomicBomb);

        nuclearSubmarineVec.Push_back(boat);

        queue.Enqueue(boat);

    }
    catch

(invalid_argument ex) {

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

    }
    break;
}
else {

    std::cout <<
"-----Неизвестный вид транспорта-----
-----" << endl;

}
}

case 4:

    std::cout << "По какому
признаку вы бы хотели отсортировать транспорт?" << endl;
    std::cout <<
"1.Скорость" << endl;

    std::cout <<
"2.Вместимость пассажиров" << endl;

    std::cout << "Введите
номер признака для сортировки: ";

    std::cin >> action;
    int sort;
    if (action == 1) {

        std::cout <<
"1. Сортировать по возрастанию" << endl;

        std::cout <<
"2. Сортировать по убыванию" << endl;

        std::cin >>

        sort;

        if (sort == 1)

        {

            sailboatVec.Sort([](Sailboat* a, Sailboat* b) {

                return a->GetMaxSpeed() < b->GetMaxSpeed();

            });

            catamaranVec.Sort([](Catamaran* a, Catamaran* b) {

                return a->GetMaxSpeed() < b->GetMaxSpeed();

            });

            nuclearSubmarineVec.Sort([](NuclearSubmarine* a,
NuclearSubmarine* b) {

                return a->GetMaxSpeed() < b->GetMaxSpeed();

            });

            cruiseLinerVec.Sort([](CruiseLiner* a, CruiseLiner* b) {

                return a->GetMaxSpeed() < b->GetMaxSpeed();

            });

        }

    }

}

```


Рис. ПЗ.1. Продолжение

```

    });

    }
    else if (sort

== 2) {

        sailboatVec.Sort([](Sailboat* a, Sailboat* b) {

            return a->GetMaxSpeed() > b->GetMaxSpeed();

        });

        catamaranVec.Sort([](Catamaran* a, Catamaran* b) {

            return a->GetMaxSpeed() > b->GetMaxSpeed();

        });

        nuclearSubmarineVec.Sort([](NuclearSubmarine* a,
NuclearSubmarine* b) {

            return a->GetMaxSpeed() > b->GetMaxSpeed();

        });

        cruiseLinerVec.Sort([](CruiseLiner* a, CruiseLiner* b) {

            return a->GetMaxSpeed() > b->GetMaxSpeed();

        });

    }
    else std::cout

<< "Неверный выбор." << endl;

    }

    else if (action == 2) {

        std::cout <<

"1. Сортировать по возрастанию" << endl;

        std::cout <<

"2. Сортировать по убыванию" << endl;

        std::cin >>

sort;

        if (sort == 1)

        {

            sailboatVec.Sort([](Sailboat* a, Sailboat* b) {

                return a->GetPassengerCapacity() < b-
>GetPassengerCapacity();

            });

            catamaranVec.Sort([](Catamaran* a, Catamaran* b) {

                return a->GetPassengerCapacity() < b-
>GetPassengerCapacity();

            });

            nuclearSubmarineVec.Sort([](NuclearSubmarine* a,
NuclearSubmarine* b) {

                return a->GetPassengerCapacity() < b-
>GetPassengerCapacity();

            });

            cruiseLinerVec.Sort([](CruiseLiner* a, CruiseLiner* b) {

                return a->GetPassengerCapacity() < b-
>GetPassengerCapacity();

            });

        }
        else if (sort

== 2) {

            sailboatVec.Sort([](Sailboat* a, Sailboat* b) {

                return a->GetPassengerCapacity() > b-
>GetPassengerCapacity();

            });

            catamaranVec.Sort([](Catamaran* a, Catamaran* b) {

                return a->GetPassengerCapacity() > b-
>GetPassengerCapacity();

            });

            nuclearSubmarineVec.Sort([](NuclearSubmarine* a,
NuclearSubmarine* b) {

                return a->GetPassengerCapacity() > b-
>GetPassengerCapacity();

            });

        }
    }
}

```

Рис. ПЗ.1. Продолжение

```

    return a->GetPassengerCapacity() > b-
>GetPassengerCapacity();

    });

    cruiseLinerVec.Sort([](CruiseLiner* a, CruiseLiner* b) {

        return a->GetPassengerCapacity() > b-
>GetPassengerCapacity();

    });

    }
    else std::cout
<< "Неверный выбор." << endl;

    }
    else {

        std::cout <<
"-----Неизвестный признак-----"
" << endl;

        break;

    }
    std::cout << "-----"
-----Сортировка завершена-----" << endl;

    break;

    case 5:

        std::cout << "По какому
признаку вы бы хотели найти объект?" << endl;
        std::cout << "1.Имя" <<
endl;

        std::cout << "2.В
диапазоне скорости" << endl;

        std::cout << "3.В
диапазоне вместимости пассажир:" << endl;

        std::cin >> action;
        int min;
        int max;

        if (action == 1) {

            std::cout <<
"Введите имя объекта которое хотите найти:" << endl;

            std::cin >>
name;

            queue.PrintByName(name);

    }
    else if (action == 2) {

        std::cout <<
"Введите диапазон от min до max." << endl;

        std::cout <<

        "Введите min." << endl;

        std::cin >>

        min;

        std::cout <<

        "Введите max." << endl;

        std::cin >>

        max;

        auto result1 =
sailboatVec.FindInRange([min, max](Sailboat* boat) {

            return boat->GetMaxSpeed() >= min && boat-
>GetMaxSpeed() <= max;

        });

        for (const
auto& boat : result1) {

            boat->Print();

            cout << endl;

        }

        auto result2 =
catamaranVec.FindInRange([min, max](Catamaran* boat) {

            return boat->GetMaxSpeed() >= min && boat-
>GetMaxSpeed() <= max;

        });

        for (const
auto& boat : result2) {

            boat->Print();

            cout << endl;

        }

        auto result3 =
nuclearSubmarineVec.FindInRange([min, max](NuclearSubmarine*
boat) {

            return boat->GetMaxSpeed() >= min && boat-
>GetMaxSpeed() <= max;

        });

        for (const
auto& boat : result3) {

```

Рис. ПЗ.1. Продолжение

```

        boat->Print();

        cout << endl;

    }
    auto result4 =
cruiseLinerVec.FindInRange([min, max](CruiseLiner* boat) {

        return boat->GetMaxSpeed() >= min && boat-
>GetMaxSpeed() <= max;

    });
    for (const
auto& boat : result4) {

        boat->Print();

        cout << endl;

    }
    }
    else if (action == 3) {

        std::cout << "Введите диапазон от min до max." <<
endl;

        std::cout << "Введите min." << endl;

        std::cin >> min;

        std::cout << "Введите max." << endl;

        std::cin >> max;

        std::cout << endl;

        auto result1 = sailboatVec.FindInRange([min,
max](Sailboat* boat) {

            return boat->GetPassengerCapacity() >= min
&& boat->GetPassengerCapacity() <= max;

        });

        for (const auto& boat : result1) {

            boat->Print();

            cout << endl;

        }

    }

    auto result2 = catamaranVec.FindInRange([min,
max](Catamaran* boat) {

        return boat->GetPassengerCapacity() >= min
&& boat->GetPassengerCapacity() <= max;

    });

    for (const auto& boat : result2) {

        boat->Print();

        cout << endl;

    }

    auto result3 = nuclearSubmarineVec.FindInRange([min,
max](NuclearSubmarine* boat) {

        return boat->GetPassengerCapacity() >= min
&& boat->GetPassengerCapacity() <= max;

    });

    for (const auto& boat : result3) {

        boat->Print();

        cout << endl;

    }

    auto result4 = cruiseLinerVec.FindInRange([min,
max](CruiseLiner* boat) {

        return boat->GetPassengerCapacity() >= min
&& boat->GetPassengerCapacity() <= max;

    });

    for (const auto& boat : result4) {

        boat->Print();

        cout << endl;

    }

}

```

Рис. ПЗ.1. Продолжение

Рис. ПЗ.1. Продолжение

```

                                cout << "-----"
-----Файл успешно сохранён-----" << endl;
                                break;
    }
                                else {
                                default:
                                std::cout << "-----"
"-----Неизвестный признак-----"
                                std::cout << "-----"
" << endl;
                                -----Неизвестная команда, введите еще раз-----
                                -----" << endl;
                                break;
                                }
                                break;
                                }
                                }
                                case 6:
                                std::cout << "Введите
путь до файла и его имя: ";
                                return 0;
                                cin >> nameFile;
                                }

                                queue.OutputToFile(nameFile);

```

Рис. ПЗ.1. Текст файла main.cpp

Текст файла IWaterTransport.h представлен на рис. ПЗ.2:

```

#pragma once
#include <fstream>
#include <string>
// класс интерфейс
class IWaterTransport
{
public:
    virtual std::string GetName() { return ""; };
    virtual double GetMaxSpeed() { return 0; };
    virtual int GetPassengerCapacity() { return 0; };
    virtual void Print() {};
    virtual void PrintToFile(std::ofstream& outputFile) {}
    virtual void ReadFromFile(std::ifstream& inFile) {}
};

```

Рис. ПЗ.2. Текст файла IWaterTransport.h

Текст файла WaterTransport.h представлен на рис. ПЗ.3:

```

#include "IWaterTransport.h"
#include <iostream>
#pragma once

// базовый класс
class WaterTransport :
    public IWaterTransport
{
private:
    double maxSpeed;
    int passengerCapacity;
    std::string name;
public:
    WaterTransport();

    WaterTransport(std::string name, double maxSpeed, int
passengerCapacity) : name(name), maxSpeed(maxSpeed),
passengerCapacity(passengerCapacity) {};
    void SetMaxSpeed(double maxSpeed);
    double GetMaxSpeed() override;
    void SetPassengerCapacity(int passengerCapacity);
    int GetPassengerCapacity() override;
    void SetName(std::string type);
    std::string GetName() override;
    void Print() override;
    void PrintToFile(std::ofstream& outputFile) override;
    void ReadFromFile(std::ifstream& inFile) override;
    ~WaterTransport() {};
};

```

Рис. ПЗ.3. Текст файла WaterTransport.h

Текст файла WaterTransport.cpp представлен на рис. ПЗ.4:

```
#include "WaterTransport.h"
using namespace std;

WaterTransport::WaterTransport() {
    name = "WaterTransport";

    maxSpeed = 70;
    passengerCapacity = 6;
}

void WaterTransport::SetMaxSpeed(double maxSpeed) {
    if (maxSpeed < 0) {
        throw std::invalid_argument("Максимальная скорость не
        может быть отрицательной.");
    }
    this->maxSpeed = maxSpeed;
}

double WaterTransport::GetMaxSpeed() {
    return maxSpeed;
}

void WaterTransport::SetPassengerCapacity(int passengerCapacity)
{
    if (passengerCapacity < 0) {
        throw std::invalid_argument("Вместимость пассажиров не
        может быть отрицательной.");
    }
    this->passengerCapacity = passengerCapacity;
}

int WaterTransport::GetPassengerCapacity() {
    return passengerCapacity;
}

void WaterTransport::SetName(std::string type) {
    this->name = type;
}

std::string WaterTransport::GetName() {
    return name;
}

void WaterTransport::Print() {
    cout << this->GetName() << endl;
    cout << this->GetMaxSpeed() << endl;
    cout << this->GetPassengerCapacity() << endl;
}

void WaterTransport::PrintToFile(std::ofstream& outputFile) {
    outputFile << "WATER_TRANSPORT" << endl;
    outputFile << this->GetName() << std::endl;
    outputFile << this->GetMaxSpeed() << std::endl;
    outputFile << this->GetPassengerCapacity() << std::endl;
}

void WaterTransport::ReadFromFile(std::ifstream& inFile) {
    string name;
    double maxSpeed;
    int passengerCapacity;

    getline(inFile, name);
    inFile >> maxSpeed;
    inFile >> passengerCapacity;

    this->SetName(name);
    this->SetMaxSpeed(maxSpeed);
    this->SetPassengerCapacity(passengerCapacity);
}
```

Рис. ПЗ.4. Текст файла WaterTransport.cpp

Текст файла EnginePoweredWaterTransport.h представлен на рис. ПЗ.5:

```
#pragma once
#include <iostream>
#include "WaterTransport.h"

//класс для водного транспорта с двигателем
class EnginePoweredWaterTransport :
    public WaterTransport
{
private:
    int powerEngine;
public:
    EnginePoweredWaterTransport();
    EnginePoweredWaterTransport(std::string name, double
    maxSpeed, int passengerCapacity, int powerEngine);

    void SetPowerEngine(int powerEngine);
    int GetPowerEngine();
    void Print();
    void PrintToFile(std::ofstream& outputFile);
    void ReadFromFile(std::ifstream& inFile);
    ~EnginePoweredWaterTransport() {};
};
```

Рис. ПЗ.5. Текст файла EnginePoweredWaterTransport.h

Текст файла EnginePoweredWaterTransport.cpp представлен на рис. ПЗ.6:

```
#include "EnginePoweredWaterTransport.h"
using namespace std;

EnginePoweredWaterTransport::EnginePoweredWaterTransport() {
    powerEngine = 300;
}

EnginePoweredWaterTransport::EnginePoweredWaterTransport(std::string name, double maxSpeed, int passengerCapacity, int powerEngine) : powerEngine(powerEngine) {
    this->SetName(name);
    this->SetMaxSpeed(maxSpeed);
    this->SetPassengerCapacity(passengerCapacity);
};

void EnginePoweredWaterTransport::SetPowerEngine(int powerEngine) {
    if (powerEngine < 0) {
        throw std::invalid_argument("Мощность двигателя не может быть отрицательной.");
    }
    this->powerEngine = powerEngine;
}

int EnginePoweredWaterTransport::GetPowerEngine() {
    return powerEngine;
}

void EnginePoweredWaterTransport::Print() {
    cout << this->GetName() << endl;
    cout << this->GetMaxSpeed() << endl;
    cout << this->GetPassengerCapacity() << endl;
    cout << this->GetPowerEngine() << endl;
}

void EnginePoweredWaterTransport::PrintToFile(ofstream& outputFile) {
    outputFile << "ENGINE_POWERED_WATER_TRANSPORT" << endl;
    outputFile << this->GetName() << endl;
    outputFile << this->GetMaxSpeed() << endl;
    outputFile << this->GetPassengerCapacity() << endl;
    outputFile << this->GetPowerEngine() << endl;
}

void EnginePoweredWaterTransport::ReadFromFile(std::ifstream& inFile) {
    string name;
    double maxSpeed;
    int passengerCapacity;
    int powerEngine;

    getline(inFile, name);
    inFile >> maxSpeed;
    inFile >> passengerCapacity;
    inFile >> powerEngine;

    this->SetName(name);
    this->SetMaxSpeed(maxSpeed);
    this->SetPassengerCapacity(passengerCapacity);
    this->SetPowerEngine(powerEngine);
}
```

Рис. ПЗ.6. Текст файла EnginePoweredWaterTransport.cpp

Текст файла Sailboat.h представлен на рис. ПЗ.7:

```
#pragma once
#include "WaterTransport.h"
class Sailboat :
    public WaterTransport
{
private:
    int sailArea;

public:
    Sailboat();

    Sailboat(std::string name, double maxSpeed, int passengerCapacity, int sailArea);
    void SetSailArea(int sailArea);
    int GetSailArea();
    void Print();
    void PrintToFile(std::ofstream& outputFile);
    void ReadFromFile(std::ifstream& inFile);
    ~Sailboat() {};
};
```

Рис. ПЗ.7. Текст файла Sailboat.h

Текст файла Sailboat.cpp представлен на рис. ПЗ.8:

```
#include "Sailboat.h"
using namespace std;

Sailboat::Sailboat() {
    sailArea = 14;
}

Sailboat::Sailboat(std::string name, double maxSpeed, int
passengerCapacity, int sailArea) : sailArea(sailArea) {
    this->SetName(name);
    this->SetMaxSpeed(maxSpeed);
    this->SetPassengerCapacity(maxSpeed);
};

void Sailboat::SetSailArea(int sailArea) {
    if (sailArea <= 0) {
        throw std::invalid_argument("Площадь не может быть
отрицательной или не может быть равна 0.");
    }
    this->sailArea = sailArea;
}

int Sailboat::GetSailArea() {
    return sailArea;
}

void Sailboat::Print() {
    cout << this->GetName() << endl;
    cout << this->GetMaxSpeed() << endl;
    cout << this->GetPassengerCapacity() << endl;

    cout << this->GetSailArea() << endl;
}

void Sailboat::PrintToFile(std::ofstream& outputFile) {
    outputFile << "SAIL_BOAT" << endl;
    outputFile << this->GetName() << std::endl;
    outputFile << this->GetMaxSpeed() << std::endl;
    outputFile << this->GetPassengerCapacity() << std::endl;
    outputFile << this->GetSailArea() << std::endl;
}

void Sailboat::ReadFromFile(std::ifstream& inFile) {
    string name;
    double maxSpeed;
    int passengerCapacity;
    int sailArea;

    getline(inFile, name);
    inFile >> maxSpeed;
    inFile >> passengerCapacity;
    inFile >> sailArea;

    this->SetName(name);
    this->SetMaxSpeed(maxSpeed);
    this->SetPassengerCapacity(passengerCapacity);
    this->SetSailArea(sailArea);
}
```

Рис. ПЗ.8. Текст файла Sailboat.cpp

Текст файла Catamaran.h представлен на рис. ПЗ.9:

```
#pragma once
#include "WaterTransport.h"
class Catamaran :
    public WaterTransport
{
private:
    int numberOfPedals;

public:
    Catamaran();

    Catamaran(std::string name, double maxSpeed, int
passengerCapacity, int numberOfPedals);

    void SetNumberOfPedals(int NumberOfPedals);
    int GetNumberOfPedals();
    void Print();
    void PrintToFile(std::ofstream& outputFile);
    void ReadFromFile(std::ifstream& inFile);
    ~Catamaran() { };
};
```

Рис. ПЗ.9. Текст файла Catamaran.h

Текст файла Catamaran.cpp представлен на рис. ПЗ.10:

```
#include "Catamaran.h"
using namespace std;

Catamaran::Catamaran() {
    numberOfPedals = 2;
}
Catamaran::Catamaran(std::string name, double maxSpeed, int
passengerCapacity, int numberOfPedals) :
numberOfPedals(numberOfPedals) {
    this->SetName(name);
    this->SetMaxSpeed(maxSpeed);
    this->SetPassengerCapacity(maxSpeed);
};
void Catamaran::SetNumberOfPedals(int numberOfPedals) {
    if (numberOfPedals <= 0) {
        throw std::invalid_argument("Количество педалей не
может быть отрицательным или равно 0.");
    }
    this->numberOfPedals = numberOfPedals;
}
int Catamaran::GetNumberOfPedals() {
    return numberOfPedals;
}

void Catamaran::Print() {
    cout << this->GetName() << endl;
    cout << this->GetMaxSpeed() << endl;
    cout << this->GetPassengerCapacity() << endl;

    cout << this->GetNumberOfPedals() << endl;
}

void Catamaran::PrintToFile(std::ofstream& outputFile) {
    outputFile << "CATAMARAN" << endl;
    outputFile << this->GetName() << std::endl;
    outputFile << this->GetMaxSpeed() << std::endl;
    outputFile << this->GetPassengerCapacity() << std::endl;
    outputFile << this->GetNumberOfPedals() << std::endl;
}

void Catamaran::ReadFromFile(std::ifstream& inFile) {
    string name;
    double maxSpeed;
    int passengerCapacity;
    int numberOfPedals;

    getline(inFile, name);
    inFile >> maxSpeed;
    inFile >> passengerCapacity;
    inFile >> numberOfPedals;

    this->SetName(name);
    this->SetMaxSpeed(maxSpeed);
    this->SetPassengerCapacity(passengerCapacity);
    this->SetNumberOfPedals(numberOfPedals);
}
```

Рис. ПЗ.10. Текст файла Catamaran.cpp

Текст файла Submarine.h представлен на рис. ПЗ.11:

```
#pragma once

Submarine(std::string name, double maxSpeed, int
passengerCapacity, int powerEngine, double divingDepth);

void SetDivingDepth(double divingDepth);

double GetDivingDepth();

void Print();

void PrintToFile(std::ofstream& outputFile);

void ReadFromFile(std::ifstream& inFile);

~Submarine() {};

};

public:

Submarine();
```

Рис. ПЗ.11. Текст файла Submarine.h

Текст файла Submarine.cpp представлен на рис. ПЗ.12:

```
#include "Submarine.h"

using namespace std;

Submarine::Submarine() {
    divingDepth = 500;
}

Submarine::Submarine(std::string name, double maxSpeed, int
passengerCapacity, int powerEngine, double divingDepth) :
divingDepth(divingDepth) {

    this->SetName(name);

    this->SetMaxSpeed(maxSpeed);

    this->SetPassengerCapacity(passengerCapacity);

    this->SetPowerEngine(powerEngine);
};

void Submarine::SetDivingDepth(double divingDepth) {
    if (divingDepth < 0) {
        throw std::invalid_argument("Глубина погружения не может
быть отрицательной.");
    }

    this->divingDepth = divingDepth;
}

double Submarine::GetDivingDepth() {
    return divingDepth;
}

void Submarine::Print() {
    cout << this->GetName() << endl;

    cout << this->GetMaxSpeed() << endl;

    cout << this->GetPassengerCapacity() << endl;

    cout << this->GetPowerEngine() << endl;

    cout << this->GetDivingDepth() << endl;
}

void Submarine::PrintToFile(std::ofstream& outputFile) {
    outputFile << "SUBMARINE" << endl;

    outputFile << this->GetName() << std::endl;

    outputFile << this->GetMaxSpeed() << std::endl;

    outputFile << this->GetPassengerCapacity() << std::endl;

    outputFile << this->GetPowerEngine() << std::endl;

    outputFile << this->GetDivingDepth() << std::endl;
}

void Submarine::ReadFromFile(std::ifstream& inFile) {
    string name;

    double maxSpeed;

    int passengerCapacity;

    double divingDepth;

    getline(inFile, name);

    inFile >> maxSpeed;

    inFile >> passengerCapacity;

    inFile >> divingDepth;

    this->SetName(name);

    this->SetMaxSpeed(maxSpeed);

    this->SetPassengerCapacity(passengerCapacity);

    this->SetDivingDepth(divingDepth);
}
```

Рис. ПЗ.12. Текст файла Submarine.cpp

Текст файла TouristBoat.h представлен на рис. ПЗ.13:

```
#pragma once
#include "EnginePoweredWaterTransport.h"
class TouristBoat :
    public EnginePoweredWaterTransport
{
private:
    int ticketPrice;

public:
    TouristBoat();

    TouristBoat(std::string name, double maxSpeed, int
passengerCapacity, int powerEngine, int ticketPrice);
    void SetTicketPrice(int ticketPrice);
    int GetTicketPrice();
    void Print();
    void PrintToFile(std::ofstream& outputFile);
    void ReadFromFile(std::ifstream& inFile);
    ~TouristBoat() {};
};
```

Рис. ПЗ.13. Текст файла TouristBoat.h

Текст файла TouristBoat.cpp представлен на рис. ПЗ.14:

```
#include "TouristBoat.h"
using namespace std;

TouristBoat::TouristBoat() {
    ticketPrice = 1000;
}

TouristBoat::TouristBoat(std::string name, double maxSpeed, int
passengerCapacity, int powerEngine, int ticketPrice) :
ticketPrice(ticketPrice) {

    this->SetName(name);

    this->SetMaxSpeed(maxSpeed);

    this->SetPassengerCapacity(passengerCapacity);

    this->SetPowerEngine(powerEngine);
};

void TouristBoat::SetTicketPrice(int ticketPrice) {
    if (ticketPrice < 0) {
        throw std::invalid_argument("Цена билета не может быть
отрицательной.");
    }

    this->ticketPrice = ticketPrice;
}

int TouristBoat::GetTicketPrice() {
    return ticketPrice;
}

void TouristBoat::Print() {
    cout << this->GetName() << endl;
    cout << this->GetMaxSpeed() << endl;
    cout << this->GetPassengerCapacity() << endl;
    cout << this->GetPowerEngine() << endl;
    cout << this->GetTicketPrice() << endl;
}

void TouristBoat::PrintToFile(std::ofstream& outputFile) {
    outputFile << "TOURIST_BOAT" << endl;
    outputFile << this->GetName() << std::endl;
    outputFile << this->GetMaxSpeed() << std::endl;
    outputFile << this->GetPassengerCapacity() << std::endl;
    outputFile << this->GetTicketPrice() << std::endl;
}

void TouristBoat::ReadFromFile(std::ifstream& inFile) {
    string name;
    double maxSpeed;
    int passengerCapacity;
    int ticketPrice;
    getline(inFile, name);
    inFile >> maxSpeed;
    inFile >> passengerCapacity;
    inFile >> ticketPrice;
    this->SetName(name);
    this->SetMaxSpeed(maxSpeed);
    this->SetPassengerCapacity(passengerCapacity);
    this->SetTicketPrice(ticketPrice);
}
```

Рис. ПЗ.14. Текст файла TouristBoat.cpp

Текст файла NuclearSubmarine.h представлен на рис. ПЗ.15:

```
#pragma once
#include "Submarine.h"
class NuclearSubmarine :
    public Submarine
{
private:
    double powerAtomicBomb; // в мегатоннах
public:
    NuclearSubmarine();

    NuclearSubmarine(std::string name, double maxSpeed, int
passengerCapacity, int powerEngine, double divingDepth, double
powerAtomicBomb);

    void SetPowerAtomicBomb(double powerAtomicBomb);
    double GetPowerAtomicBomb();

    void Print();
    void PrintToFile(std::ofstream& outputFile);
    void ReadFromFile(std::ifstream& inFile);
    ~NuclearSubmarine() {};
```

Рис. ПЗ.15. Текст файла NuclearSubmarine.h

Текст файла NuclearSubmarine.cpp представлен на рис. ПЗ.16:

```
#include "NuclearSubmarine.h"
using namespace std;
NuclearSubmarine::NuclearSubmarine() {
    powerAtomicBomb = 50;
}

NuclearSubmarine::NuclearSubmarine(std::string name,
double maxSpeed, int passengerCapacity, int powerEngine, double
divingDepth, double powerAtomicBomb) :
powerAtomicBomb(powerAtomicBomb) {
    this->SetName(name);
    this->SetMaxSpeed(maxSpeed);
    this->SetPassengerCapacity(passengerCapacity);
    this->SetPowerEngine(powerEngine);
    this->SetDivingDepth(divingDepth);
};

void NuclearSubmarine::SetPowerAtomicBomb(double
powerAtomicBomb) {
    if (powerAtomicBomb < 0) {
        throw std::invalid_argument("Мощность атомной
бомбы не может быть отрицательной.");
    }

    this->powerAtomicBomb = powerAtomicBomb;
}

double NuclearSubmarine::GetPowerAtomicBomb() {
    return powerAtomicBomb;
}

void NuclearSubmarine::Print() {
    cout << this->GetName() << endl;
    cout << this->GetMaxSpeed() << endl;

    cout << this->GetPassengerCapacity() << endl;
    cout << this->GetPowerEngine() << endl;

    cout << this->GetDivingDepth() << endl;
    cout << this->GetPowerAtomicBomb() << endl;
}

void NuclearSubmarine::PrintToFile(std::ofstream&
outputFile) {
    outputFile << "NUCLEAR_SUBMARINE" << endl;

    outputFile << this->GetName() << std::endl;

    outputFile << this->GetMaxSpeed() << std::endl;

    outputFile << this->GetPassengerCapacity() << std::endl;

    outputFile << this->GetPowerEngine() << std::endl;

    outputFile << this->GetDivingDepth() << std::endl;

    outputFile << this->GetPowerAtomicBomb() << std::endl;
}

void NuclearSubmarine::ReadFromFile(std::ifstream& inFile)
{
    string name;
    double maxSpeed;
    int passengerCapacity;
    double divingDepth;
    double powerAtomic;

    getline(inFile, name);
    inFile >> maxSpeed;
    inFile >> passengerCapacity;
    inFile >> divingDepth;
    inFile >> powerAtomic;
```

Рис. ПЗ.16. Продолжение

```

this->SetName(name);
this->SetMaxSpeed(maxSpeed);
this->SetPassengerCapacity(passengerCapacity);

this->SetDivingDepth(divingDepth);
this->SetPowerAtomicBomb(powerAtomic);
}

```

Рис. ПЗ.16. Текст файла NuclearSubmarine.cpp

Текст файла CruiseLiner.h представлен на рис. ПЗ.17:

```

#pragma once
#include "TouristBoat.h"

class CruiseLiner :
    public TouristBoat
{
private:
    int heightWaterSlides;

public:
    CruiseLiner();

    CruiseLiner(std::string name, double maxSpeed, int
passengerCapacity, int powerEngine, int ticketPrice, int
heightWaterSlides);

    void SetHeightWaterSlides(int heightWaterSlides);

    int GetHeightWaterSlides();

    void Print();

    void PrintToFile(std::ofstream& outputFile);

    void ReadFromFile(std::ifstream& inFile);

    ~CruiseLiner() {};
};

```

Рис. ПЗ.17. Текст файла CruiseLiner.h

Текст файла CruiseLiner.cpp представлен на рис. ПЗ.18:

```

#include "CruiseLiner.h"
using namespace std;

CruiseLiner::CruiseLiner() {
    heightWaterSlides = 15;
}

CruiseLiner::CruiseLiner(std::string name, double maxSpeed,
int passengerCapacity, int powerEngine, int ticketPrice, int
heightWaterSlides) : heightWaterSlides(heightWaterSlides) {
    this->SetName(name);
    this->SetMaxSpeed(maxSpeed);
    this->SetPassengerCapacity(passengerCapacity);
    this->SetPowerEngine(powerEngine);
    this->SetTicketPrice(ticketPrice);
};

void CruiseLiner::SetHeightWaterSlides(int
heightWaterSlides) {
    if (heightWaterSlides < 0) {
        throw std::invalid_argument("Высота водных горок не
может быть отрицательной.");
    }
    this->heightWaterSlides = heightWaterSlides;
}

int CruiseLiner::GetHeightWaterSlides() {
    return heightWaterSlides;
}

void CruiseLiner::Print() {
    cout << this->GetName() << endl;
    cout << this->GetMaxSpeed() << endl;
    cout << this->GetPassengerCapacity() << endl;
    cout << this->GetPowerEngine() << endl;
    cout << this->GetTicketPrice() << endl;
    cout << this->GetHeightWaterSlides() << endl;
}

void CruiseLiner::PrintToFile(std::ofstream& outputFile) {

```

Рис. ПЗ.18. Продолжение

```

outputFile << "CRUISE_LINER" << endl;
outputFile << this->GetName() << std::endl;
outputFile << this->GetMaxSpeed() << std::endl;
outputFile << this->GetPassengerCapacity() << std::endl;
outputFile << this->GetTicketPrice() << std::endl;
outputFile << this->GetHeightWaterSlides() << std::endl;
}
void CruiseLiner::ReadFromFile(std::ifstream& inFile) {
    string name;
    double maxSpeed;
    int passengerCapacity;
    int ticketPrice;
    int HeightWaterSlides;

    getline(inFile, name);
    inFile >> maxSpeed;
    inFile >> passengerCapacity;
    inFile >> ticketPrice;
    inFile >> HeightWaterSlides;

    this->SetName(name);
    this->SetMaxSpeed(maxSpeed);
    this->SetPassengerCapacity(passengerCapacity);
    this->SetTicketPrice(ticketPrice);
    this->SetHeightWaterSlides(HeightWaterSlides);
}

```

Рис. ПЗ.18. Текст файла CruiseLiner.cpp

Текст файла MyVector.h представлен на рис. ПЗ.19:

```

#pragma once
#include <iostream>
#include <vector>
#include <fstream>
#include <string>
#include "IWaterTransport.h"
#include "WaterTransport.h"
#include "EnginePoweredWaterTransport.h"
#include "TouristBoat.h"
#include "Sailboat.h"
#include "Catamaran.h"
#include "CruiseLiner.h"
#include "NuclearSubmarine.h"
using namespace std;

template <typename T>
class MyVector {
private:
    std::vector<T> vec;

public:
    MyVector() {};

    T& At(size_t index) {
        return vec.at(index);
    }

    void Push_back(const T& value) {
        vec.push_back(value);
    }

    void Pop_back() {
        vec.pop_back();
    }

    T& Front() {
        return vec.front();
    }

    T& Back() {
        return vec.back();
    }

    size_t Size() const {
        return vec.size();
    }

    bool Empty() const {
        return vec.empty();
    }

    typename std::vector<T>::iterator Begin() {
        return vec.begin();
    }

    typename std::vector<T>::iterator End() {
        return vec.end();
    }
}

```

Рис. ПЗ.19. Продолжение

<pre> template <typename Comparator> void Sort(Comparator comp) { sort(vec.begin(), vec.end(), comp); } template <typename Predicate> std::vector<T> FindInRange(Predicate pred) const { std::vector<T> result; for (const auto& item : vec) { if (pred(item)) { </pre>	<pre> result.push_back(item); } } return result; } ~MyVector() { vec.clear(); } }; </pre>
--	--

Рис. ПЗ.19. Текст файла MyVector.h

Текст файла VoidQueue.h представлен на рис. ПЗ.20:

<pre> #pragma once #include <queue> #include <iostream> #include <fstream> #include <string> #include "IWaterTransport.h" #include "WaterTransport.h" #include "EnginePoweredWaterTransport.h" #include "TouristBoat.h" #include "Sailboat.h" #include "Catamaran.h" #include "CruiseLiner.h" #include "NuclearSubmarine.h" using namespace std; class VoidQueue { private: std::queue<void*> container; protected: void Enqueue(void* item) { container.push(item); } </pre>	<pre> void* Dequeue() { if (container.empty()) { return nullptr; } void* item = container.front(); container.pop(); return item; } bool IsEmpty() const { return container.empty(); } size_t Size() const { return container.size(); } void* At(size_t index) const { if (index >= container.size()) { return nullptr; } std::queue<void*> tempQueue = container; for (size_t i = 0; i < index; ++i) { tempQueue.pop(); } </pre>
--	---

Рис. ПЗ.20. Продолжение

```

return tempQueue.front();
}

void OutputToFile(const string& filename) {
    ofstream outputFile(filename);
    if (!outputFile.is_open()) {
        cout << "Ошибка: Не удалось открыть файл для
записи.\n";
        return;
    }
    size_t queueSize = Size();
    for (size_t i = 0; i < queueSize; ++i) {
        static_cast<IWaterTransport*>(At(i))-
>PrintToFile(outputFile);
    }
    outputFile.close();
}

void InputFromFile(const string& filename) {
    ifstream inputFile(filename);
    if (!inputFile.is_open()) {
        cout << "Ошибка: Не удалось открыть файл для
чтения.\n";
        return;
    }
    string type;
    while (getline(inputFile, type)) {
        if (type == "WATER_TRANSPORT") {
            WaterTransport* boat = new WaterTransport();
            boat->ReadFromFile(inputFile);
            Enqueue(boat);
        }
        else if (type ==
"ENGINE_POWERED_WATER_TRANSPORT") {
            EnginePoweredWaterTransport* boat = new
EnginePoweredWaterTransport();
            boat->ReadFromFile(inputFile);
            Enqueue(boat);

```

```

}
else if (type == "CATAMARAN") {
    Catamaran* boat = new Catamaran();
    boat->ReadFromFile(inputFile);
    Enqueue(boat);
}
else if (type == "SAIL_BOAT") {
    Sailboat* boat = new Sailboat();
    boat->ReadFromFile(inputFile);
    Enqueue(boat);
}
else if (type == "SUBMARINE") {
    Submarine* boat = new Submarine();
    boat->ReadFromFile(inputFile);
    Enqueue(boat);
}
else if (type == "TOURIST_BOAT") {
    TouristBoat* boat = new TouristBoat();
    boat->ReadFromFile(inputFile);
    Enqueue(boat);
}
else if (type == "CRUISE_LINER") {
    CruiseLiner* boat = new CruiseLiner();
    boat->ReadFromFile(inputFile);
    Enqueue(boat);
}
else if (type == "NUCLEAR_SUBMARINE") {
    NuclearSubmarine* boat = new NuclearSubmarine();
    boat->ReadFromFile(inputFile);
    Enqueue(boat);
}
}
inputFile.close();
}

void PrintByName(const string& name) const {
    std::queue<void*> tempQueue = container;
    bool found = false;

```

Рис. ПЗ.20. Продолжение

```

        cout << "Объекты с именем " << name << " не найдены."
        << endl;
    }
}

while (!tempQueue.empty()) {
    void* item = tempQueue.front();
    tempQueue.pop();

    if (static_cast<IWaterTransport*>(item)->GetName() ==
name) {
        static_cast<IWaterTransport*>(item)->Print();
        found = true;
    }
}
if (!found) {
    ~VoidQueue() {
        while (!container.empty()) {
            container.pop();
        }
    }
};

```

Рис. ПЗ.20. Текст файла VoidQueue.h

Текст файла TmpQueue.h представлен на рис. ПЗ.21:

```

#pragma once
#include "VoidQueue.h"

template <typename T>

class TmpQueue : private VoidQueue {
public:
    void Enqueue(T* item) {
        VoidQueue::Enqueue(static_cast<void*>(item));
    }

    T* Dequeue() {
        return static_cast<T*>(VoidQueue::Dequeue());
    }

    bool IsEmpty() const {
        return VoidQueue::IsEmpty();
    }

    size_t Size() const {
        return VoidQueue::Size();
    }

    T* At(size_t index) const {
        return static_cast<T*>(VoidQueue::At(index));
    }

    void OutputToFile(const string& filename) {
        VoidQueue::OutputToFile(filename);
    }

    void InputFromFile(const string& filename) {
        VoidQueue::InputFromFile(filename);
    }

    void PrintByName(const string& name) {
        VoidQueue::PrintByName(name);
    }
}

```


Рис. ПЗ.21. Продолжение

```
~TmpQueue() {  
    while (!IsEmpty()) {  
        delete Dequeue();  
    }  
};
```

Рис. ПЗ.21. Текст файла TmpQueue.h