Министерство образования Республики Беларусь

Учреждение образования Белорусский государственный университет информатики и радиоэлектроники

Факультет инновационного и непрерывного обучения

Кафедра программного обеспечения информационных технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовой работе**

Выполнил: Нацевич В.Г.

 Группа № 894351, 2 курс

 №зачетки 6943009

Проверил:

Минск 2020

**Содержание**

ВВЕДЕНИЕ……………………………………………………………..3

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ………………………………4
	1. Описание предметной области………………………………….4
	2. Анализ требований к проектируемому приложению………….5
	3. Постановка решаемой задачи…………………………………...6
2. ПРОЕКТИРОВАНИЕ СТРУКТУРЫ ПРИЛОЖЕНИЯ И ИНТЕРФЕЙСОВ РЕАЛИЗУЕМЫХ КЛАССОВ……………………...7
	1. Выбор используемых шаблонов……………………………....7
		1. Шаблоны классов…………………………………………8
		2. Шаблоны функций………………………………………..9
	2. Архитектура приложения……………………………………….10
		1. Динамический двусвязный список……………………....12
		2. Абстрагирование объектов……………………………….14
		3. Реализация поиска пункта назначения…………………..16
		4. Интерфейс пользователя……………………….….19
3. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ…………………………………...20
4. ЗАКЛЮЧЕНИЕ……………………………………………………….....25
5. Список использованных источников…………………………………..26

**ВВЕДЕНИЕ**

Целью работы является практическая работа с парадигмой и технологией объектно-ориентированного программирования (ООП) и сопутствующих технологий программирования, а также - работа с основами ООП на языке программирования высокого уровня С++.

Задачей работы является применить теоретические и практические знания по принципам объектно-ориентированного проектирования и разработки объектно-ориентированного программного обеспечения ЭВМ. Отработать основные концепции объектно-ориентированного программирования, лексические и синтаксические основы объектно- ориентированного языка программирования С++, принципы построения классов и объектов, конструкторы и деструкторы, виртуальные методы и классы, абстрактные классы, принципы и виды наследования классов, шаблоны классов, обработку исключительных ситуаций, методику объектно- ориентированного анализа и проектирования. Отработать создание объектно-ориентированных программ с применением классов на языке С++, создавать иерархию классов на С++, использовать полиморфизм, проектировать с учетом множественного наследования, создавать шаблоны функции и классов, использовать механизм обработки исключений, использовать библиотеку потоковых классов, производить отладку программного кода C++.

Овладеть навыками по объектно-ориентированному проектированию и разработке объектно-ориентированного программного кода в современных операционных системах.

1. **АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**
	1. **Описание предметной области**

Роль и место задач разработки программного обеспечения в интегрированных производственных комплексах, автоматизированных системах управления техническими объектами. Факторы, обусловившие появление и содержание концепции ООП.

Основные идеи ООП: использование объекта в качестве основной компоненты программы и децентрализация управления, реализуемое представлением программы как описания взаимодействия объектов. Инкапсуляция, наследование, полиморфизм. Объектно- ориентированный подход к разработке программ. Место и роль ООП в теории и практике разработки программных систем. Объект как совокупность данных и набора операций. Семантика объекта. Представление данных. Классификация методов: конструкторы, деструкторы, селекторы и модификаторы. Классы объектов: назначение и семантика. Класс как абстракция совокупности объектов. Классы и абстрактные типы данных. Объекты как экземпляры классов. Основные действия с объектами: создание, инициализация, использование, уничтожение. Отношение наследования для классов.

Простое и множественное наследование. Иерархия классов. Объектно-ориентированный стиль программирования. Инструментальные средства автоматизации проектирования программных систем. Графический подход к решению проблемы автоматизации разработки программного обеспечения. Требования качеству современных программных средств.

Реализация концепции объектно-ориентированного программирования в языке программирования С++.

Правильно разрабатываемой программы должны не только удовлетворять своим функциональным требованиям, но и обладать такими свойствами, как:

* повторная используемость;
* расширяемость;
* устойчивость к неправильным данным;
* системность.
	1. **Анализ требований к проектируемому приложению**

В реализации справочно-информационной системы для автовокзала. Организовать поиск рейса в необходимый пункт назначения по времени, дате, классу рейса. Основные классы: населенный пункт, рейс, транспорт, остановочная станция, билет.

Для поиска рейса используется двусвязный список неопределенной длины с динамически выделяемой памятью.

Для реализации объектов используются основные физические свойства программируемых объектов. Каждый населенный пункт имеет некоторое количество станций(предположительно как минимум одну) значит объект населенный пункт будет иметь свойство ”name” - название населенного пункта. Исходя из этого и похожих свойств будет построена структура объекта. Которые в свою очередь могут быть вынесены в отдельный класс для описания основных объектов программы.

Потребуется создание структуры иерархии объектов на основе физической структуры:

Населенный пункт -> Остановочная станция -> Рейс -> Транспорт -> Билет.

Алгоритм поиска может быть основан на названии маршрута, так как номера маршрута имеют уникальное значение при сравнении станции отправления будет осуществляться поиск по станции прибытия.

В разработке архитектуры приложения должны учитываться основные принципы ООП:

* Инкапсуляция
* Наследование
* Полиморфизм

 Что в дальнейшей разработке поможет улучшить защищенность приложения от непродуманного алгоритма, поможет повысить гибкость к расширению функционала. Описанный и продуманный интерфейс поможет в дальнейшем изменять улучшать реализацию алгоритмов без изменения интерфейса.

* 1. **Постановка решаемой задачи**

Требуется реализовать справочно-информационную систему для автовокзала. Организовать поиск рейса в необходимый пункт назначения по времени, дате, классу рейса.

Программа будет обладать следующей функциональностью:

* Приложение будет реализовано в виде консольной утилиты
* Возможность динамического добавления: населенных пунктов, остановочных станций, рейсов и транспорта
* Поиск рейса по названию рейса(поиск базируется на уникальной нумерации каждого маршрута, что в свою очередь соответствует естественному состоянию объекта в жизни)
* Поиск рейса в заданный диапазон времени
1. **ПРОЕКТИРОВАНИЕ СТРУКТУРЫ ПРИЛОЖЕНИЯ И ИНТЕРФЕЙСОВ РЕАЛИЗУЕМЫХ КЛАССОВ**
	1. **Выбор используемых шаблонов**

Шаблоны — средство языка C++, предназначенное для кодирования обобщенных алгоритмов, без привязки к некоторым параметрам (например, типам данных, размерам буферов, значениям по умолчанию).

В C++ возможно создание шаблонов функций и классов.

Шаблоны позволяют создавать параметризованные классы и функции. Параметром может быть любой тип или значение одного из допустимых типов (целое число, enum, указатель на любой объект с глобально доступным именем, ссылка).

* + 1. **Шаблоны классов**

В классе, реализующем связный список объектов, алгоритмы добавления нового элемента списка, поиска нужного элемента не зависят от того, что элементы списка — объекты. Те же алгоритмы применялись бы и для списка населенных пунктов, станций, рейсов, транспорта, и так далее.

 На примере реализации двусвязного списка неопределенной длины с динамически выделяемой памятью “List”:

template <class T>

class List {

 Node<T> \*root = NULL;

 int count = 0;

public:

 List();

 List(const List& orig);

 virtual ~List();

 void add(T \*obj);

 T\* get(int i);

 int size();

 void deleteAll(Node<T> \*\*root);

private:

 void add(T \*obj, Node<T> \*\*nodes);

 T\* get(int i, Node<T> \*\*nodes);

};

 Может использоваться для построения списка объектов большой длины с хранением объектов разных типов. Разрабатывался для хранения объектов “Ride”, “Transport” и т.д. но может быть использован для хранения объектов других типов от чисел до строк.

* + 1. **Шаблон функции**

В отсутствие шаблонов приходится писать отдельные функции для каждого используемого типа данных. Хотя многие языки программирования определяют встроенную функцию минимума для элементарных типов (таких как целые и вещественные числа), такая функция может понадобиться и для сложных (например «число» или «строка») и очень сложных («транспортное средство» в программе) объектов.

 На примере реализации метода “add” класса “List”:

template <class T>

void List<T>::add(T \*obj, Node<T> \*\*nodes) {

 static Node<T> \*prev = NULL;

 if ((\*nodes) == NULL) {

 (\*nodes) = new Node<T>();

 (\*nodes)->data = obj;

 (\*nodes)->prev = prev;

 this->count++;

 return;

 }

 if ((\*nodes)->next == NULL) {

 prev = (\*nodes);

 }

 this->add(obj, &(\*nodes)->next);

}

Описанный выше метод наполняет динамически создаваемый список объектами различных типов. В разрабатываемой программе это “Locality”, “Station”, “Ride”, “Transport”. Что по окончанию ввода данных создает иерархическую структуру в начале которой находится населенный пункт и в конце билет на транспортное средство.

* 1. **Архитектура приложения**

Архитектура приложения базируется на методологии программирования ООП. Основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

 Основная идеология ООП - это подход к программированию как к моделированию информационных объектов, решающий на новом уровне основную задачу структурного программирования, структурирование информации с точки зрения управляемости, что существенно улучшает управляемость самим процессом моделирования, что особенно важно при реализации крупных проектов.

Управляемость для иерархических систем предполагает минимизацию данных и их целостность, поэтому созданное удобно управляемым будет и удобно в понимании написанного. Таким образом через задачу управляемости решается стратегическая задача транслировать понимание задачи программистом в наиболее удобную для дальнейшего использования форму.

Основные принципы структурирования в случае ООП связаны с различными аспектами базового понимания предметной задачи, которое требуется для оптимального управления соответствующей моделью:

* абстрагирование для выделения в моделируемом предмете важного для решения конкретной задачи по предмету, в конечном счете контекстное понимание предмета, формализуемое в виде класса;
* инкапсуляция для быстрой и безопасной организации собственно иерархической управляемости: чтобы было достаточно простой команды «что делать», без одновременного уточнения как именно делать;
* наследование для быстрой и безопасной организации родственных понятий: чтобы было достаточно на каждом иерархическом шаге учитывать только изменения, не дублируя все остальное, учтенное на предыдущих шагах;
* полиморфизм для определения точки, в которой единое управление лучше распараллелить или наоборот собрать воедино.

То есть фактически речь идет о прогрессирующей организации информации согласно семантическим критериям. Прогрессирование, в частности, на последнем этапе дает возможность перехода на следующий уровень детализации, что замыкает общий процесс.

Обычный человеческий язык в целом отражает идеологию ООП, начиная с инкапсуляции представления о предмете в виде его имени и заканчивая полиморфизмом использования слова в переносном смысле, что в итоге развивает выражение представления через имя предмета до полноценного понятия класса.

 В архитектуре приложения используются такие элементы ООП как абстрактный класс, шаблоны классов и шаблоны функций немного описанные выше. Значительную роль в приложении занимает динамически пополняемый двусвязный список. Речь о которых пойдет ниже.

* + 1. **Динамический двусвязный список**

Динамический двусвязный(двунаправленный) список является расширенной версией односвязного списка с отличием от последнего дополнительным адресным пространством для хранения адреса предыдущего объекта.

Рассмотрим на примере класса “Node”:

template <class T>

class Node {

public:

 Node();

 Node(const Node& orig);

 virtual ~Node();

 T \*data = NULL;

 Node<T> \*prev = NULL;

 Node<T> \*next = NULL;

private:

};

 В приложении для хранения информационных объектов разработан на основе класса “Node” двусвязный список который позволяет перемещаться как вперед по списку так и назад за счет двух адресных пространств - это свойства объекта “prev” и “next”. Реализованный двусвязный линейный список в свойстве объекта “Node” “data” который позволяет хранить любый данные от числа до объекта типа “Locality”. Так как свойство “data” это указатель на тип данных “T” при добавлении объекта в список придется динамически выделять память с помощью оператора “new”. Отсюда и расшифровывается название - динамический список.

 Для освобождения памяти от занимаемых данных по окончанию работы приложения используется деструктор класса “List” который является интерфейсом для работы с линейным списком.

Деструктор класса “List”:

template <class T>

List<T>::~List() {

 this->deleteAll(&this->root);

}

 template <class T>

void List<T>::deleteAll(Node<T> \*\*root) {

 //delete data of T type

 if ((\*root)->data != NULL) {

 delete (\*root)->data;

 }

 //delete next object

 if ((\*root)->next != NULL) {

 this->deleteAll(&(\*root)->next);

 }

 //after deleting data of objects it is deleting a self node

 delete (\*root);

}

 Для удаления данных используется рекурсивная функция “deleteAll()” в которую аргументом передается первый объект класса “Node” при движении по списку в первую очередь удаляются данные в объекте а по возвращении обратно при схлопывании рекурсии удаляется сам объект типа “Node”.

 Динамический список позволяет организовать иерархическую структуру моделируемых информационных объектов: Locality, Station, Ride и т.д.

**2.2.2. Абстрагирование объектов**

В программе созданы основные классы которые информационно моделируют моделируют основные физические объекты: “Locality”, “Station”, “Ride”, “Transport”.

 Пример класса “Station”:

 class Station: public Object {

public:

 //rides of this station

 List<Ride> \*rides;

 Station(std::string name, List<Ride> \*rides);

 Station(const Station& orig);

 virtual ~Station();

 friend std::ostream& operator<<(std::ostream &stream, Station &s);

private:

};

Все эти классы имеют похожую структуру, они все имеют название которое информативно представлено в виде свойства “name” и ссылку на дочерний список объектов. Свойство “name” вынесено в абстрактный класс “Object”.

Абстрактный класс “Object”:

class Object {

public:

 //name of object

 std::string name;

 Object();

 Object(std::string name);

 Object(const Object& orig);

 virtual ~Object() = 0;

private:

};

Который в дальнейшем при расширении функционала программы может быть расширен и все классы наследуемые от него получат доступ к этому функционалу.

* + 1. **Реализация поиска пункта назначения**

 Поиск пункта назначения основан на принципе именования рейсов. Каждый рейс имеет уникальное числовое обозначение, например

**210 "ДС Запад-3 - Аксаковщина"**. Для начала поиска пользователю потребуется ввести населенный пункт отправления и пункт назначения и примерное время отправления. Алгоритм поиска поиска реализован в методе “search” класса “Locality”.

 Ride\* Locality::search(

 List<Locality> \*localities,

 std::string from,

 std::string to,

 std::string from\_time,

 std::string to\_time

) {

 Locality \*loc;

 Locality \*loc\_from = NULL, \*loc\_to = NULL;

 for (int i = 1; i <= localities->size(); i++) {

 loc = localities->get(i);

 if (loc->name == from) {

 loc\_from = loc;

 } else if(loc->name == to) {

 loc\_to = loc;

 }

 }

 std::cout << "from: " << loc\_from->name << std::endl;

 std::cout << "to: " << loc\_to->name << std::endl;

 if (loc\_from->stations->size() > 0) {

 std::cout << "Have found " << loc\_from->stations->size() << " stations in source locality =)" << std::endl;

 for (int isf = 1; isf <= loc\_from->stations->size(); isf++) {

 Station \*sf = loc\_from->stations->get(isf);

 if (sf->rides->size() > 0) {

 std::cout << "Have found " << sf->rides->size() << " rides in source locality =)" << std::endl;

 for (int irf = 1; irf <= sf->rides->size(); irf++) {

 Ride \*rf = sf->rides->get(irf);

 //search by a destination locality

 if (loc\_to->stations->size() > 0) {

 std::cout << "Have found " << loc\_to->stations->size() << " stations in destination locality =)" << std::endl;

 for (int ist = 1; ist <= loc\_to->stations->size(); ist++) {

 Station \*st = loc\_to->stations->get(ist);

 if (st->rides->size() > 0) {

 for (int irt = 1; irt <= st->rides->size(); irt++) {

 Ride \*rt = st->rides->get(irt);

 if (rf->name == rt->name) {

 if (rf->start\_time >= from\_time && rf->finish\_time <= to\_time) {

 std::cout << "Have found " << rt->name << " ride in destination locality =)" << std::endl;

 std::cout << rt->name << ": start time: " << rf->start\_time << " finish time: " << rf->finish\_time << std::endl;

 }

 }

 }

 }

 }

 } else {

 std::cout << "Have not found any stations in " << loc\_to->name << " =(" << std::endl;

 }

 }

 }

 }

 } else {

 std::cout << "Have not found any stations in " << loc\_from->name << " =(" << std::endl;

 }

}

Принцип поиска достаточно прост, в населенном пункте отправления и назначения сравниваются все маршруты на всех станциях, Так как маршруты имеют одинаковое обозначение в результате поиска выводится найденное совпадение. Оно может быть не одно в указанный промежуток времени.

* + 1. **Интерфейс пользователя**

 Интерфейс пользователя — интерфейс, обеспечивающий передачу информации между пользователем-человеком и программно-аппаратными компонентами компьютерной системы.

 Для программы был разработан интерфейс командной строки.

Интерфейс командной строки — разновидность текстового интерфейса (CUI) между человеком и компьютером, в котором инструкции компьютеру даются в основном путём ввода с клавиатуры текстовых строк (команд), в UNIX-системах возможно применение мыши. Также известен под названием консоль. Изображен на рис. 1.

Интерфейс командной строки противопоставляется системам управления программой на основе меню, а также различным реализациям графического интерфейса.

Формат вывода информации в интерфейсе командной строки не регламентируется; обычно это также простой текстовый вывод, но может быть и графическим, звуковым и т. д.

 

рис. 1 - Интерфейс командной строки

Интерфейс пользователя прост в использовании. Не требует специальных навыков.

1. **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Программа разрабатывалась под ОС Ubuntu 17.04. Для начала работы с программой потребуется выполнить действия в терминале:

cd /path/to/download/dir;
unzip kr\_oop-master.zip;
cd kr\_oop-master;
make;
cd dist/Debug/GNU-Linux;
sudo chmod 777 -R .;
./kr\_oop;

 Команда “make” обработает “Makefile” и компилятор с++ скомпилирует приложение. После выполнения последней команды программа предложит создать транспорт(рис. 2):
 
 рис. 2 - Создание транспорта

После транспорта программ предложит создать маршрут(рис. 3), транспорт созданный ранее будет закреплен за этим маршрутом.

 Создаем маршруты:


 рис. 3 - Создание маршрут

Далее потребуется создать станции(рис. 4). Станций может быть несколько и для каждой станции есть возможность добавить свой список маршрутов:



 рис. 4 - Создание станций

Далее последует создание населенного пункта:


 рис. 5 - Создание населенного пункта

За созданным населенным пунктом будут закреплены ранее созданные станции и маршруты на этих станциях. После создания населенного пункта программа предложит создать еще один населенный пункт и все действия от создания транспорта потребуется выполнить еще раз для этого населенного пункта.

После создания транспорта, рейсов, станций и населенных пунктов будет доступен поиск (рис. 6) по названию маршрута, дате и времени отправления и времени прибытия.



рис. 6 - Поиск по рейсам

Во время поиска программа сообщает о количестве найденных станциях, маршрутов и выводит найденные маршруты в виде:

“210-minsk-grodno: start time: 2018-06-13 9:45:00 finish time: 2018-06-13 12:00:00”

1. **ЗАКЛЮЧЕНИЕ**

В курсовой работе было разработано приложение справочный помощник на языке С++. В приложении был реализован функционал добавления транспорта, рейсов, станций и населенных пунктов. Также был создан поиск маршрута от места отправления до места назначения по всем станциям и маршрутам.

В результате объектно-ориентированного подхода к проектированию программ процесс разработки программы превращается в процесс передового программирования, который для внесения каких-либо изменений и дополнений в программу не требует значительно пересмотра ранее разработанного функционала. Передовой способ программирования опирается на сохранение целостности объектов программы, то есть внесение изменений в программу не будет затрагивать внутреннюю организацию существующих в ней классов

При разработке приложения были соблюдены все основные концепции ООП: полиморфизм, наследование, инкапсуляция. Также для решения некоторых задач были применены шаблоны языка С++ для работы с различными типами данных.

1. **Список использованных источников**

1.1. И.Г.Гоулд, Дж.С.Тутилл. Терминологическая работа IFIP (Международная федерация по обработке информации) и ICC (Международный вычислительный центр) // Журн. вычисл. матем. и матем. физ., 1965, #2. - С. 377-386.

1.2. Г.Майерс. Надежность программного обеспечения. - М.: Мир, 1980.

1.3. Ian Sommerville. Software engineering. - Addison-Wesley Publishing Company, 1992.

1.4. Э. Дейкстра. Заметки по структурному программированию / У. Дал, Э. Дейкстра, К. Хоор. Структурное программирование. - М.: Мир, 1975. - С. 7-97.

1.5. Criteria for evaluation of software. - ISO TC97/SC7 #367 (Supersedes Document #327).

1.6. С.И. Ожегов. Словарь русского языка. - М.: Советская энциклопедия, 1975.

1.7. Ф.Я. Дзержинский, И.М. Калиниченко. Дисциплина программирования Д: концепция и опыт реализации методических средств программной инженерии. - М.: ЦНИИ информации и технико-экономических исследований по атомной науке и технике, 1988.

1.8. В. Турский. Методология программирования. - М.: Мир, 1981.

1.9. Г. Буч. Объектно-ориентированное проектирование с примерами применения. - М.: Конкорд, 1992.

1.10. Е.А. Жоголев. Система программирования с использованием библиотеки подпрограмм / Система автоматизация программирования. - М.: Физматгиз, 1961. - С. 15-52.

1.11. Ф.П. Брукс, мл. Как проектируются и создаются программные комплексы. - М.: Наука, 1979.

1.12. R.C. Holt. Structure of computer programs: A Survey // Proceedings of the IEEE, 1975, 63(6). - P. 879-893.

1.13. Дж. Хьюз, Дж. Мичтом. Структурный подход к программированию. - М.: Мир, 1980.

1.14. Е.А. Жоголев. Технологические основы модульного программирования // Программирование, 1980, #2. - С. 44-49.

1.15. Б. Боэм, Дж. Браун, Х. Каспар и др. Характеристики качества программного обеспечения. - М.: Мир, 1981.

1.16. В.В. Липаев. Качество программного обеспечения. - М.: Финансы и статистика, 1983.

1.17. Б. Шнейдерман. Психология программирования. - М.: Радио и связь, 1984.

1.18. Revised version of DP9126 - Criteria of the evaluation of software quality characteristics. ISO TC97/SC7 #610. Part 6.

1.19. В.Ш. Кауфман. Языки программирования. Концепции и принципы. - М.: Радио и связь, 1993.

1.20. Требования и спецификации в разработке программ. - М.: Мир, 1984.

1.21. В.Н. Агафонов. Спецификация программ: понятийные средства и их организация. - Новосибирск: Наука (Сибирское отделение), 1987.