ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Лабораторная работа №8

Программа, управляемая событиями

Вариант №15

Выполнил студент группы РИС-23-3Б

Блинов А. Е.

Проверила доцент кафедры ИТАС

О. А. Полякова

2024 г.

**Постановка задачи**

1. Определить иерархию пользовательских классов (см. лабораторную работу №5). Во главе иерархии должен стоять абстрактный класс с чисто виртуальными методами для ввода и вывода информации об атрибутах объектов.

2. Реализовать конструкторы, деструктор, операцию присваивания, селекторы и модификаторы.

3. Определить класс-группу на основе структуры, указанной в варианте.

4. Для группы реализовать конструкторы, деструктор, методы для добавления и удаления элементов в группу, метод для просмотра группы.

5. Определить класс Диалог – наследника группы, в котором реализовать методы для обработки событий.

6. Добавить методы для обработки событий группой и объектами пользовательских классов.

Базовый класс: ПЕЧАТНОЕ\_ИЗДАНИЕ(PRINT) Название– string Автор – string Производный класс ЖУРНАЛ (MAGAZIN) Количество страниц - int Группа – Дерево (Tree).

Команды:

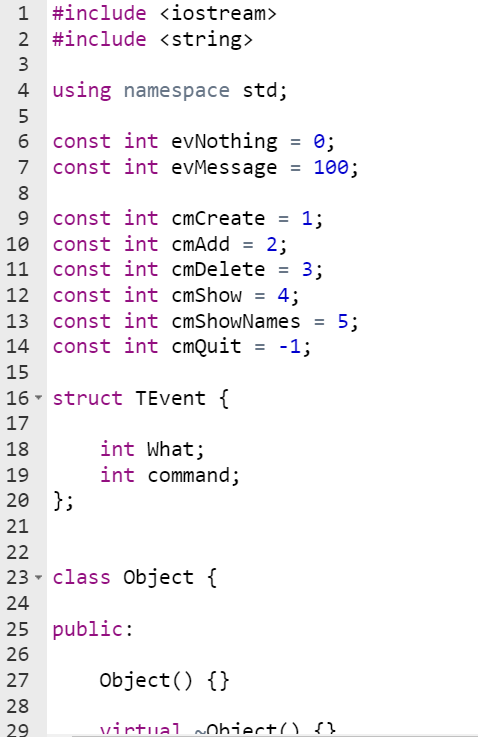
* Создать группу (формат команды: m количество элементов группы).
* Добавить элемент в группу (формат команды: +)
* Удалить элемент из группы (формат команды -)
* Вывести информацию об элементах группы (формат команды: s)
* Вывести информацию о названиях всех элементов группы (формат команды : z) Конец работы (формат команды: q)

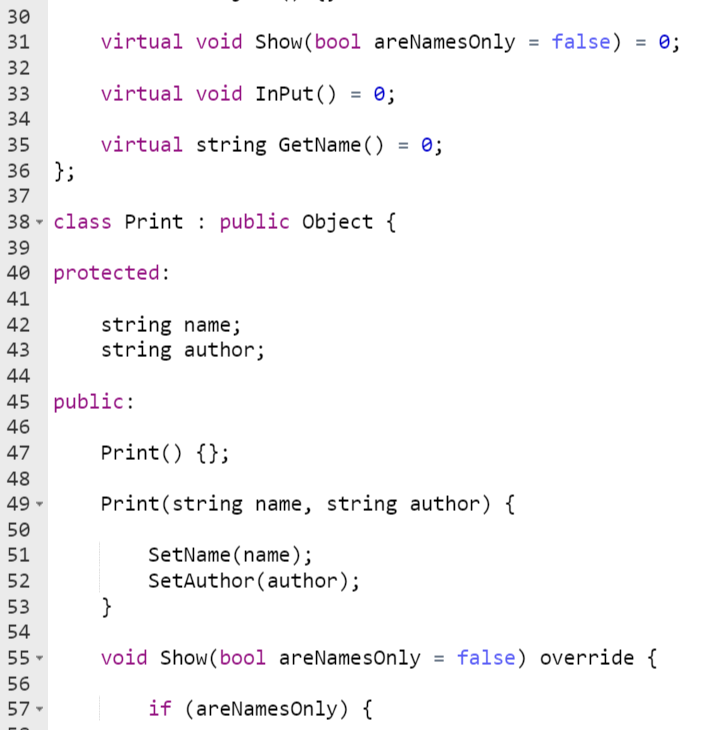
**UML диаграмма**

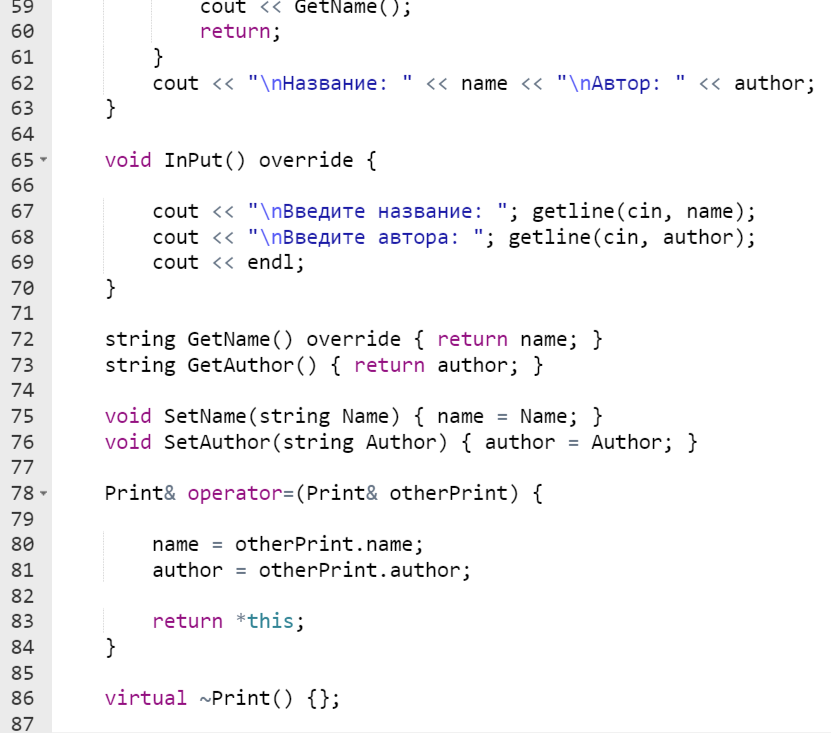
Изображение выглядит как снимок экрана, дизайн

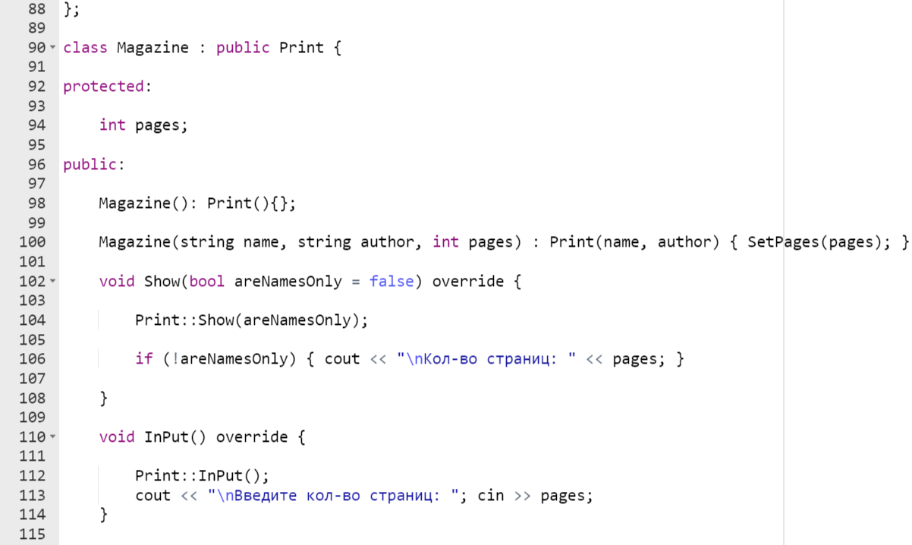
Автоматически созданное описание

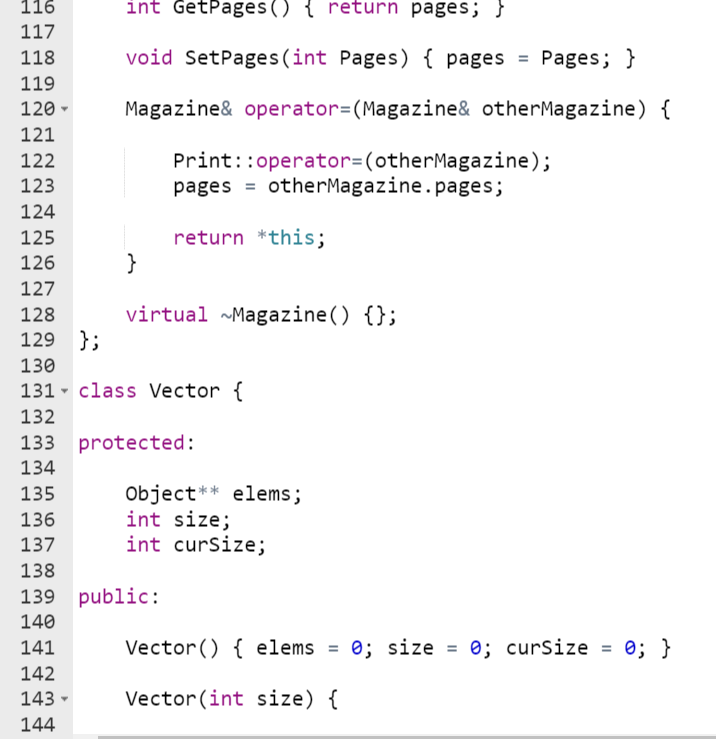
**Код на языке C++**

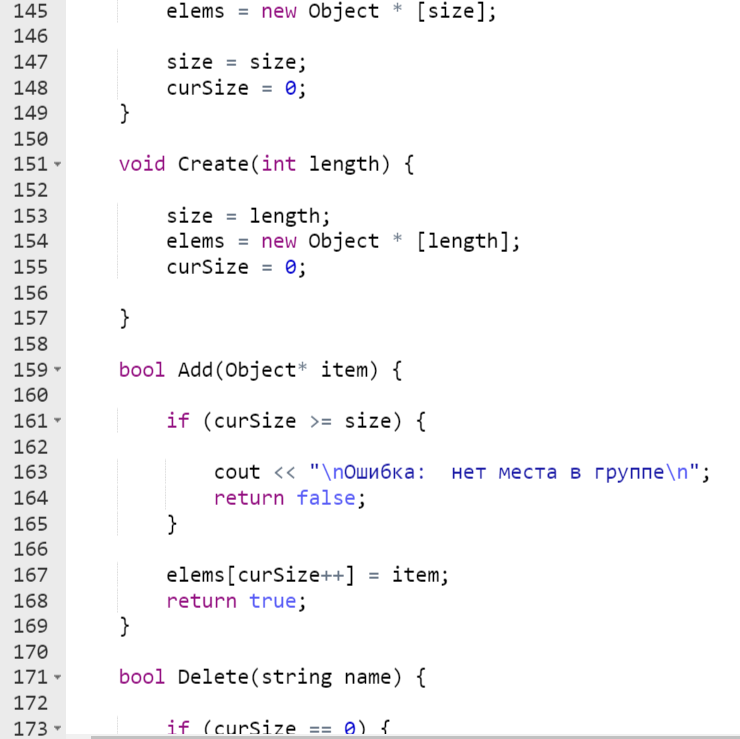
****

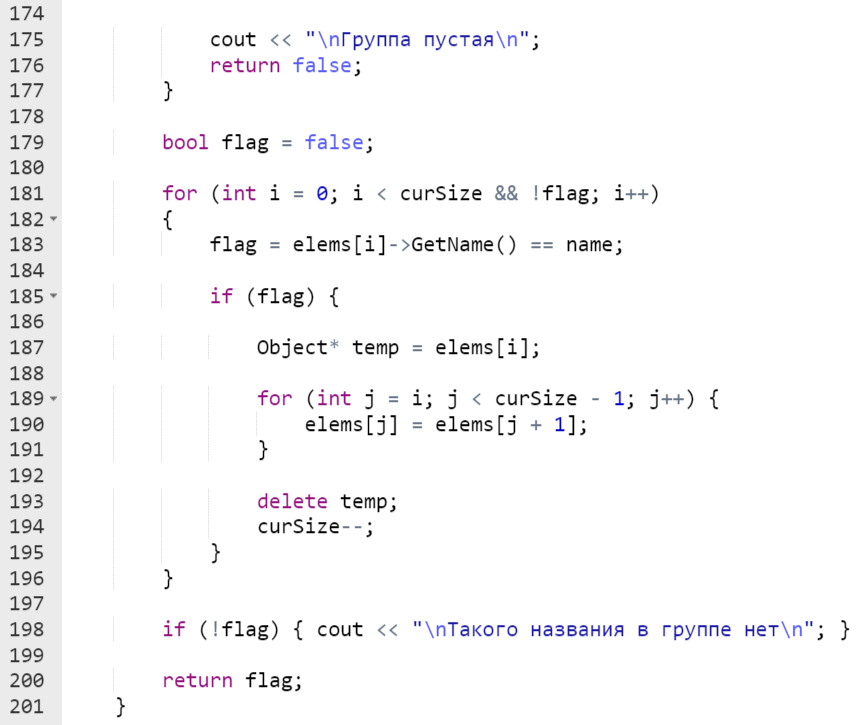
****

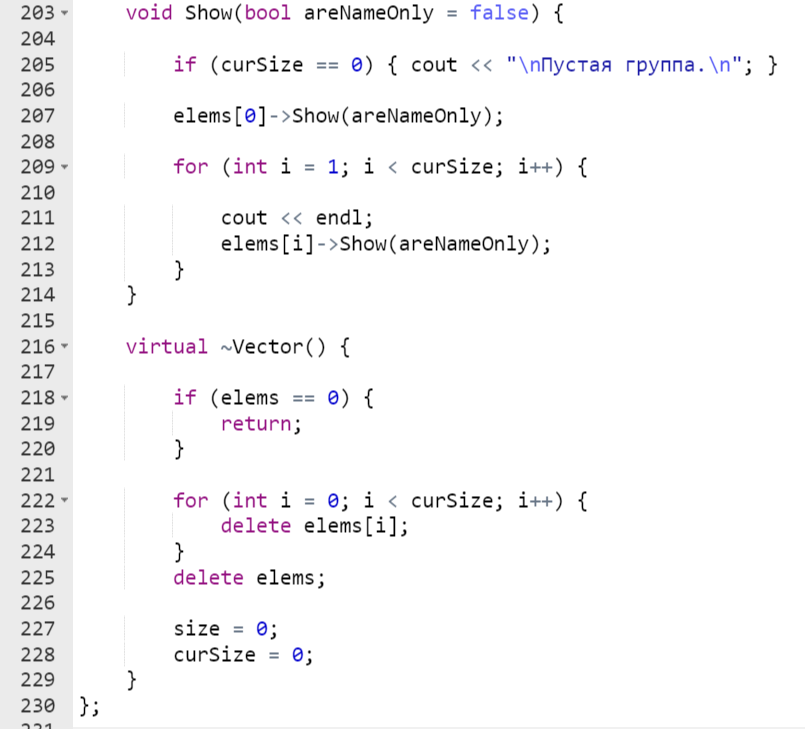
****

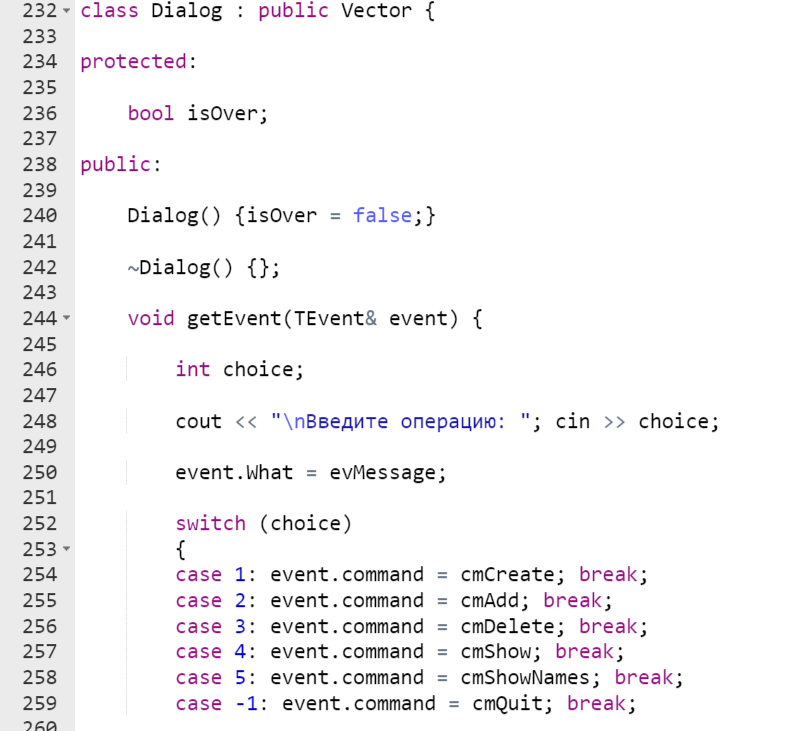
****

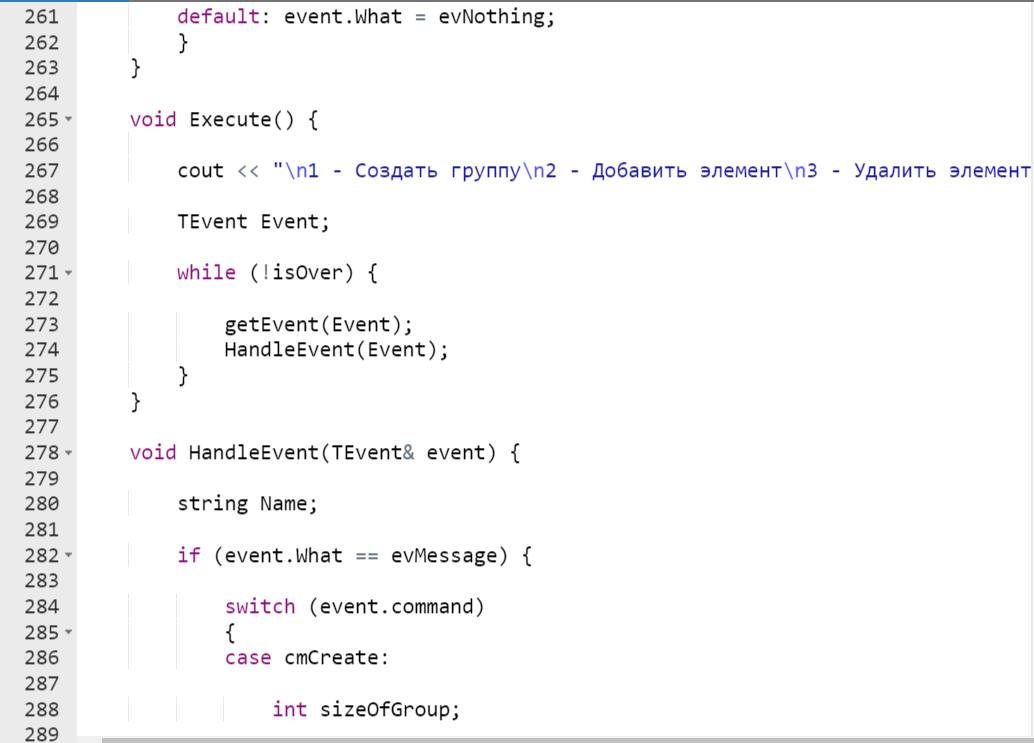
****

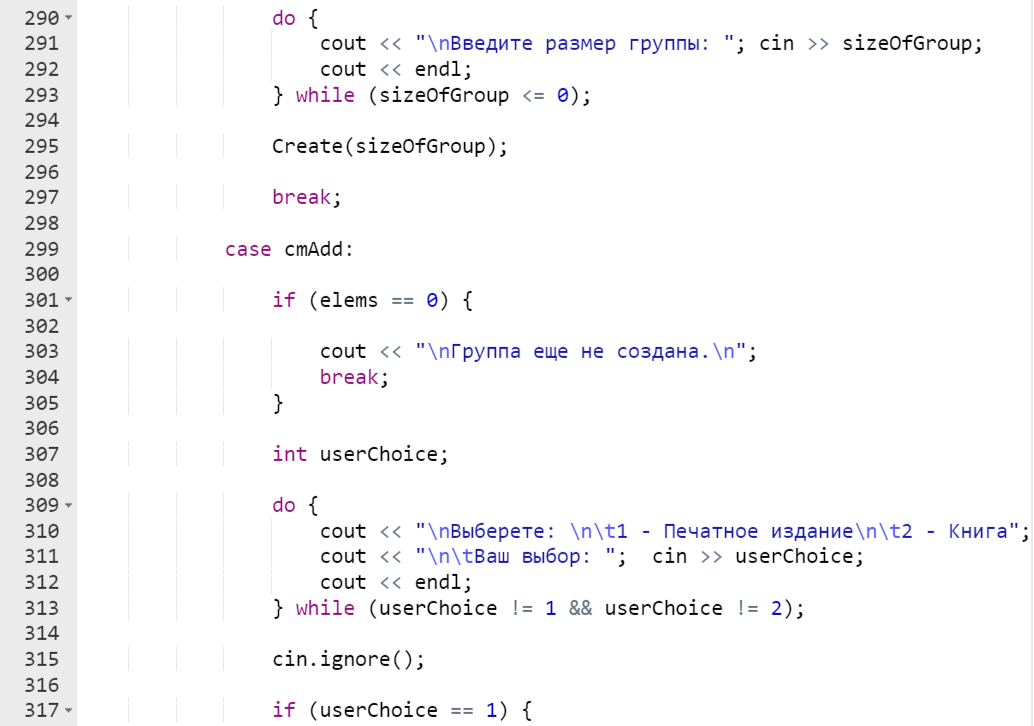
****

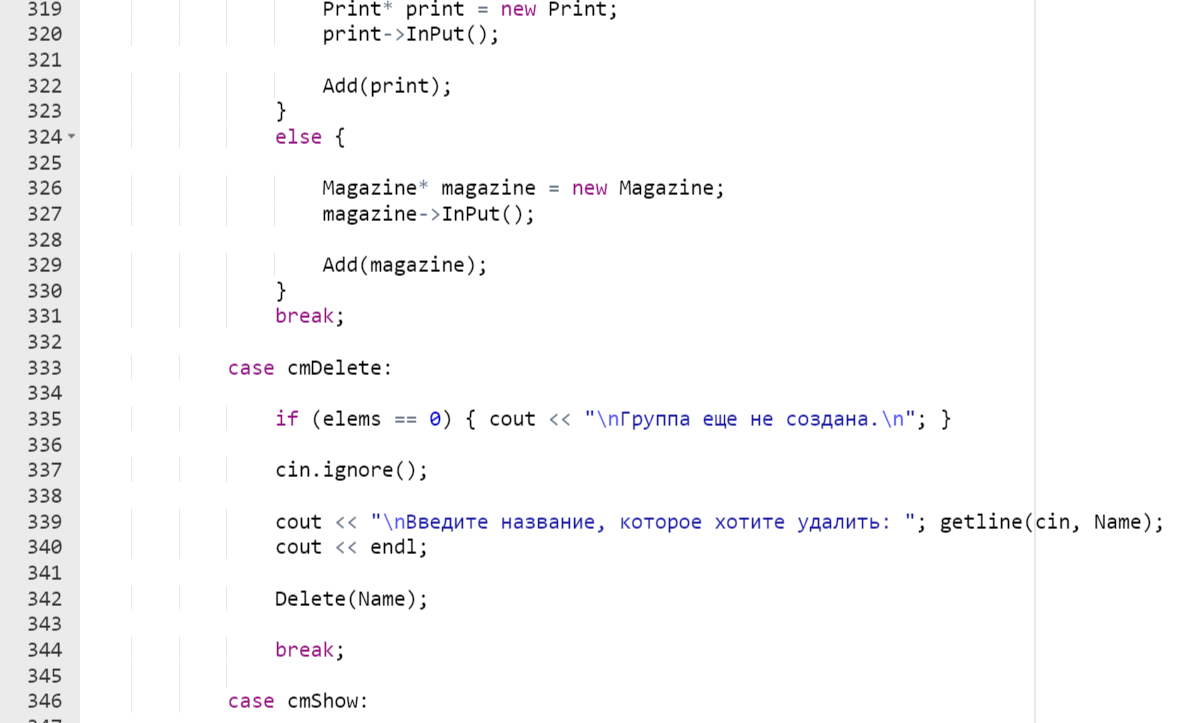
****

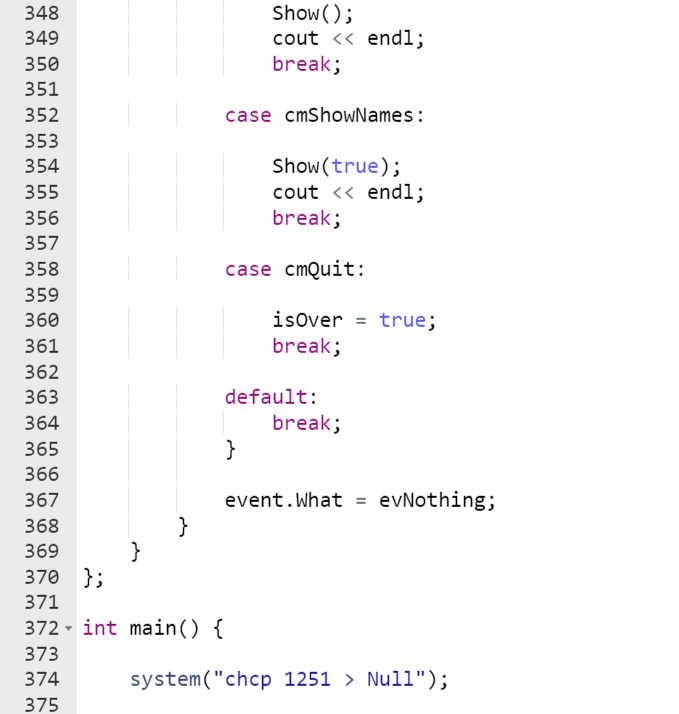
****

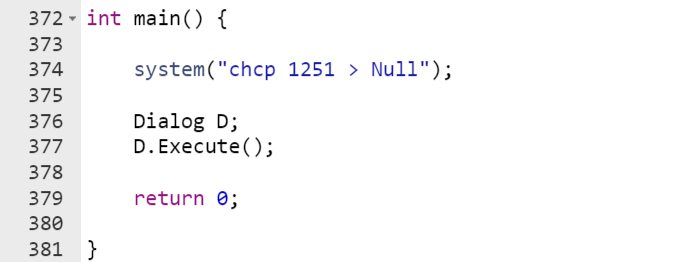
****

****

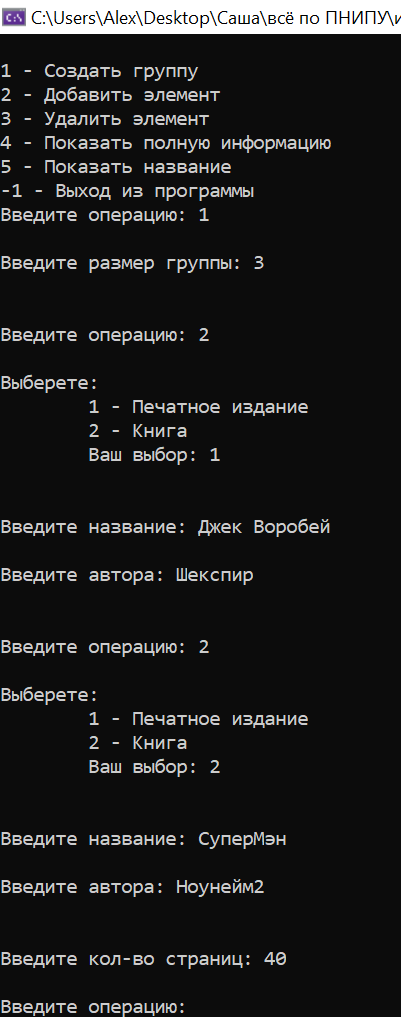
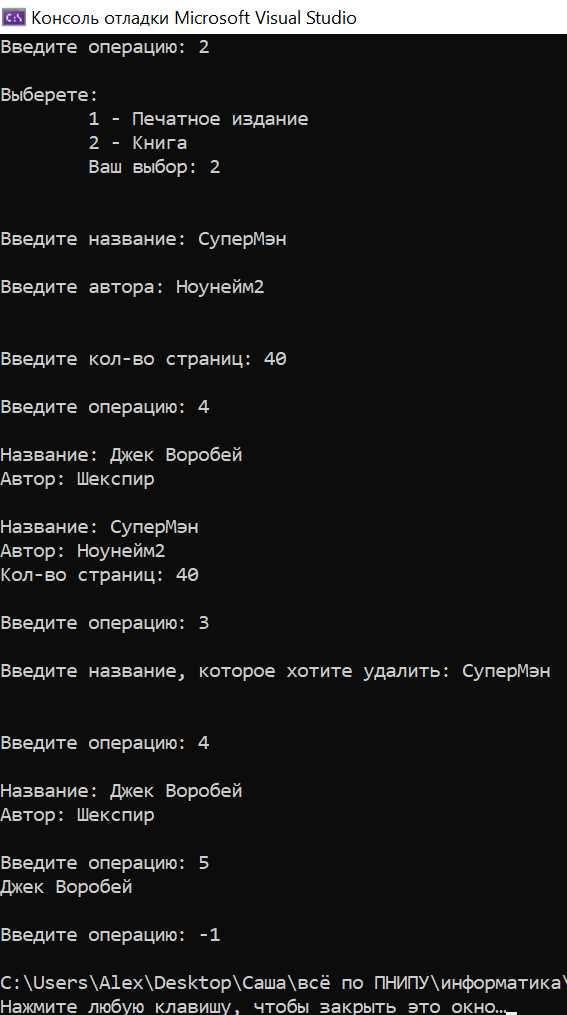
****

****

****

****

**Результаты работы программы**

****  ****

**Выводы**

Программа работает успешно

GitHub: <https://github.com/alexblin123/Laba-8-OOP.git>

**Контрольные вопросы**

**1. Что такое класс-группа? Привести примеры таких классов.**

Класс-группа - это класс, который содержит в себе коллекцию объектов других классов. Он предназначен для организации и управления группой объектов одного типа или различных типов.

Примеры: список, стек, очередь и т. д.

**2. Привести пример описания класса-группы Список (List)**

template <typename T>

class List {

private:

int size;

Node<T>\* head;

Node<T>\* tail;

public:

// Конструкторы

List(); // Конструктор без параметров

List(int size, T data); // Конструктор с параметрами

List(const List& list); // Конструктор копирования

// Деструктор

~List();

// Методы

void push\_back(T data); // Добавление элемента в конец списка

T& operator[](int index); // Доступ к элементу по индексу

int getSize() const; // Получение размера списка

};

**3. Привести пример конструктора (с параметром, без параметров, копирования) для класса-группы Список.**

**Конструктор без параметров**

template <typename T>

List<T>::List() {

size = 0;

head = nullptr;

tail = nullptr;

}

**Конструктор с параметрами**

template <typename T>

List<T>::List(int size, T data) {

this->size = size;

if (size > 0) {

Node<T>\* node = new Node<T>;

node->data = data;

head = node;

tail = node;

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

Node<T>\* newNode = new Node<T>;

newNode->data = data;

tail->next\_node = newNode;

newNode->prev\_node = tail;

tail = newNode;

}

tail->next\_node = nullptr;

} else {

head = nullptr;

tail = nullptr;

}

}

**Конструктор копирования**

template <typename T>

List<T>::List(const List& list) {

head = nullptr;

tail = nullptr;

size = 0;

Node<T>\* current\_node = list.head;

while (current\_node != nullptr) {

push\_back(current\_node->data);

current\_node = current\_node->next\_node;

}

}

**4. Привести пример деструктора для класса-группы Список.**

template <typename T>

List<T>::~List() {

Node<T>\* current\_node = head;

while (current\_node != nullptr) {

Node<T>\* next\_node = current\_node->next\_node;

delete current\_node;

current\_node = next\_node;

}

head = nullptr;

}

**5. Привести пример метода для просмотра элементов для класса-группы Список.**

template <typename T>

T& List<T>::operator[](int index) {

Node<T>\* current\_node = head;

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

current\_node = current\_node->next\_node;

}

return current\_node->data;

}

**6. Какой вид иерархии дает группа?**

Группа дает линейную иерархию, где каждый элемент является равным членом этой группы.

**7. Почему во главе иерархии классов, содержащихся в группе объектов должен находиться абстрактный класс?**

Абстрактный класс ставится во главе иерархии, чтобы определить общий интерфейс для всех классов, содержащихся в группе объектов. Он содержит чисто виртуальные функции, которые должны быть реализованы в производных классах. Это позволяет использовать указатели на абстрактный класс для работы с различными объектами в группе, не зная их конкретных типов, что обеспечивает гибкость и упрощает работу с группой объектов.

**8. Что такое событие? Для чего используются события?**

Событие в ООП – это то, что может случится с каким-либо объектом при определенных условиях (например, с кнопкой при клике на неё мышкой). При взаимодействии других объектов с этим событием (клик мыши), данный объект будет знать, что клик мыши произошел.

Событие с точки зрения языка С++ – это объект, отдельные поля которого характеризуют те или иные свойства передаваемой информации.

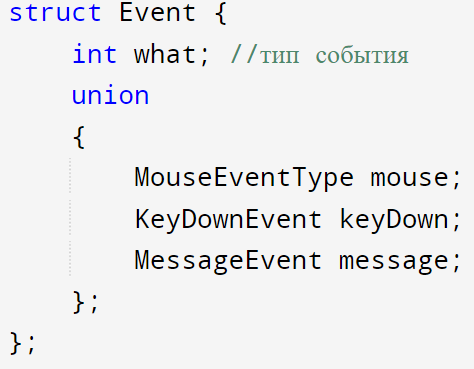
События сами по себе не производят никаких действий в программе, но в ответ на событие могут создаваться новые объекты, модифицироваться или уничтожаться существующие, что и приводит к изменению состояния программы.

**9. Какие характеристики должно иметь событие-сообщение?**

Сообщение, передаваемое от одних объектов другим, имеет, как правило, следующие характеристики:

* код класса сообщения, отличающий сообщения объектов одного класса от объектов другого класса;
* адрес объекта, которому предназначено сообщение (м. б. не задан, тогда сообщение могут прочитать все объекты);
* информационное поле

**10. Привести пример структуры, описывающей событие.**

****

**11.**

Полю **what** в структуре **TEvent** присваиваются значения, которые указывают на тип события или тип действия, которое произошло. Например:

* Если событие является событием мыши (**MouseEventType**), то полю **what** может быть присвоено значение, которое указывает на тип события мыши, такое как нажатие кнопки мыши, перемещение курсора и т.д.
* Если событие является событием нажатия клавиши (**KeyDownEvent**), то полю **what** может быть присвоено значение, которое указывает на тип нажатой клавиши.
* Если событие является сообщением (**MessageEvent**), то полю **what** может быть присвоено значение, которое указывает на тип сообщения, например, тип ошибки, предупреждения или информационное сообщение.

**12.**

Поле **command** в структуре **TEvent** присваивается значения в случаях, когда событие представляет собой команду.

Например, если событие представляет собой выполнение какой-то команды, то в поле **command** будет содержаться код этой команды.

**13.**

Поля **a** и **message** используются для передачи параметров команды, которая может быть представлена в **TEvent**.

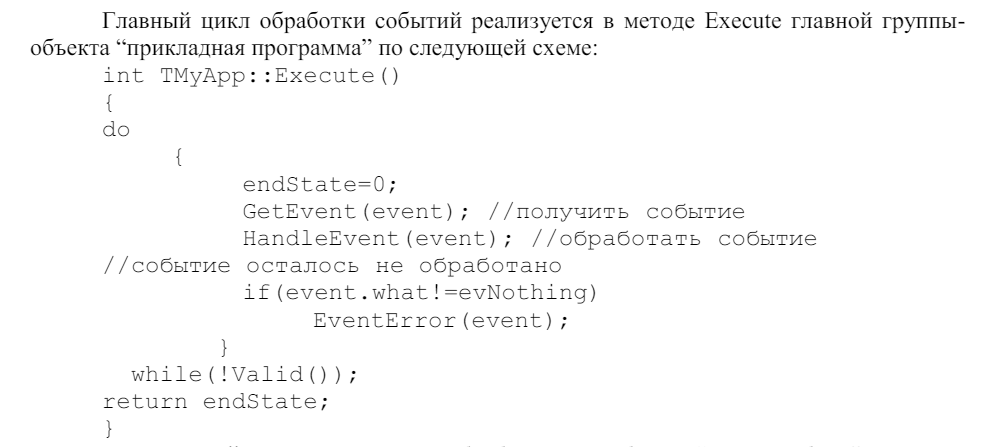
Например, если команда требует передачи дополнительных данных или аргументов, таких как числовые значения или сообщения, то эти данные могут быть храниться в полях **a** и **message**.

**14.**

Для организации обработки сообщений необходимо реализовать следующие методы:

* Метод для обработки событий: Этот метод должен принимать событие в качестве параметра и выполнять соответствующие действия в зависимости от типа события и его параметров.
* Метод для отправки сообщений: Этот метод используется для отправки сообщений или событий другим частям программы или объектам, которые могут их обработать.
* Методы доступа к параметрам события: Если необходимо получить доступ к параметрам события (например, **a** и **message**), то нужно предоставить методы доступа к этим полям, чтобы другие части программы могли использовать эти данные при обработке событий.

**15. Какой вид имеет главный цикл обработки событий-сообщений?**



Execute реализует главный цикл обработки событий. Он постоянно получает событие путем вызова GetEvent и обрабатывает их с помощью HandleEvent. Этот цикл завершается, когда поступит событие «конец».

**16. Какую функцию выполняет метод ClearEvent()? Каким образом?**

ClearEvent очищает событие, присваивая полю event.What значение evNothing.

**17. Какую функцию выполняет метод HandleEvent ()? Каким образом?**

Метод HandleEvent программы обрабатывает событие “конец работы”, вызывая метод EndExec. EndExec изменяет значение private – переменной EndState. Значение этой переменной проверяет метод–функция Valid, возвращающая значение true, если “конец работы”.

Такой несколько сложный способ завершения работы программы связан с тем, что в активном состоянии могут находиться несколько элементов группы. Тогда метод Valid группы, вызывая методы Valid своих подэлементов, возвратит true, если все они возвратят true. Это гарантирует, что программа завершит свою работу, когда завершат работу все ее элементы. Если событие осталось не обработанным, то вызывается метод EventError, которая в простейшем случае может просто выдать сообщение.

**18. Какую функцию выполняет метод GetEvent ()?**

GetEvent – формирование события

**19. Для чего используется поле EndState? Какой класс (объект) содержит это поле?**

Поле **EndState** обычно используется для хранения информации о конечном состоянии события или действия. Это поле указывает на то, что произошло завершение какого-либо процесса или события.

В контексте ООП поле **EndState** может содержаться в классе, представляющем событие или действие. Например, если у нас есть класс **Event**, который представляет событие, то поле **EndState** может храниться в этом классе. Оно будет указывать на состояние, в котором событие завершается, например, успешно выполнено ли событие или возникла ошибка во время его выполнения.

**20. Для чего используется функция Valid()?**

Функция **Valid()** обычно используется для проверки корректности или валидности объекта или данных. Она может выполнять различные проверки, чтобы убедиться, что объект находится в правильном состоянии или что данные соответствуют определенным критериям.

Например, если у нас есть класс **Event**, функция **Valid()** может проверять, содержит ли событие необходимую информацию, такую как тип события, данные о событии и т. д., чтобы быть полностью определенным и готовым к обработке. Если какая-то информация отсутствует или не корректна, функция **Valid()** может вернуть **false**, указывая на невалидное состояние объекта.