Пермский национальный исследовательский политехнический университет

**Лабораторная работа №8**

**Дисциплина: Основы алгоритмизации и**

**программирования**

**Вариант № 9**

**"Программа, управляемая событиями"**

Выполнил: Зайченко Никита Геннадьевич

Проверила: Доцент кафедры ИТАС Полякова О. А.

Пермь 2022

**Содержание отчета**

1) Постановка задачи (общая и конкретного варианта).

2) Описание класса-контейнера.

3) Определение компонентных функций.

4) Функция main().

5) Объяснение результатов работы программы.

6) Ответы на контрольные вопросы.

**Постановка задачи (общая и конкретного варианта)**

1) Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.

2) Разработка программы, управляемой событиями.

3) Базовый класс:

ПЕЧАТНОЕ\_ИЗДАНИЕ(PRINT)

Название– string

Автор – string

Производный класс

КНИГА (BOOK)

Количество страниц - int

Издательство - string

Группа – Список (List).

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 названии элемента группы с номером k (формат команды:

z k, где k – целое число)  Конец работы (формат команды: q)

**Описание класса-контейнера.**

class List

{

public:

List();

List(int);

~List();

void Add();

void HandleEvent(const TEvent& e);

void Remove(int index);

void PrintName(int k);

void Del();

void Show();

int operator()();

protected:

struct Node

{

Object\* data; // данные элемента

Node\* next; // указатель на следующий элемент

Node(Object\* dat, Node\* n = nullptr) : data(dat), next(n) {}

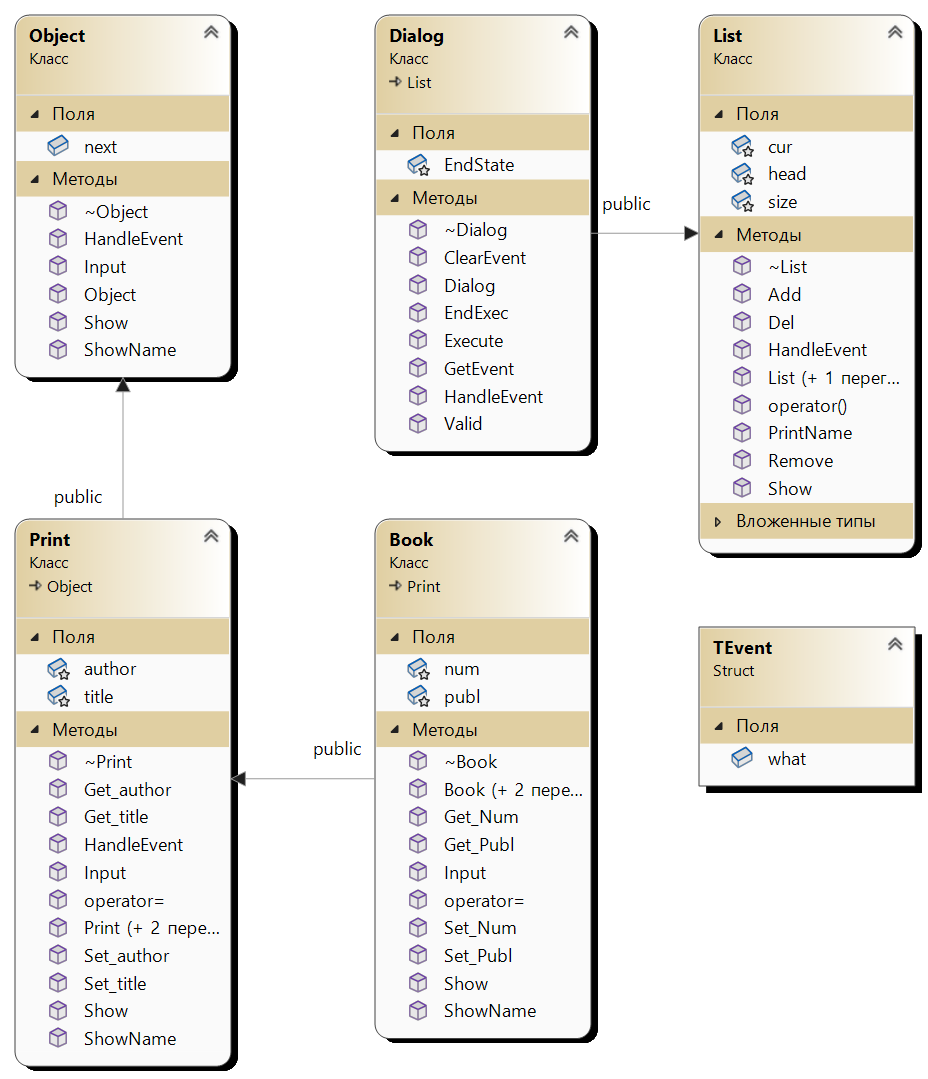
};

Node\* head; // указатель на первый элемент списка

int size; // размер списка

int cur; // текущая позиция

};



**Определение компонентных функций.**

List::List() { head = nullptr; size = 0; cur = 0; }

List::List(int s)

{

head = nullptr;

size = 0;

cur = 0;

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

{

Object\* p = nullptr;

Node\* temp = new Node(p, head);

head = temp;

size++;

}

}

List::~List()

{

while (head)

{

Node\* temp = head->next;

delete head;

head = temp;

}

}

void List::Add()

{

Object\* p;

cout << "1. Print " << endl;

cout << "2. Book " << endl;

int y;

cin >> y;

if (y == 1)//добавление объекта класса Print

{

Print\* a = new Print();

a->Input();

p = a;

Node\* temp = new Node(p, head);

head = temp;

size++;

}

if (y == 2) //добавление объекта класса Book

{

Book\* b = new Book();

b->Input();

p = b;

Node\* temp = new Node(p, head);

head = temp;

size++;

}

}

void List::HandleEvent(const TEvent& e)

{

if (e.what == evMessage)

{

Node\* current\_node = head;

while (current\_node)

{

Object\* p = static\_cast<Object\*>(current\_node->data);

p->HandleEvent(e);

current\_node = current\_node->next;

}

}

}

void List::Remove(int index)

{

if (index < 0 || index >= size) return;

Node\* temp = head;

Node\* prev = nullptr;

// ищем элемент по индексу

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

{

prev = temp;

temp = temp->next;

}

if (prev == nullptr) // удаление первого элемента

{

head = temp->next;

}

else

{

prev->next = temp->next;

}

delete temp;

size--;

}

void List::PrintName(int k) {

if (k < 0 || k > size) {

cout << "Такого элемента нет\n";

return;

}

int count = 0;

Node\* temp = head;

while (temp && count != k) {

temp = temp->next;

count++;

}

if (temp) {

temp->data->ShowName();

}

}

void List::Del()

{

if (size == 0) return;

Node\* temp = head;

head = head->next;

delete temp;

size--;

}

void List::Show()

{

if (size == 0) return;

Node\* temp = head;

while (temp)

{

temp->data->Show(); // вызываем метод Show() объекта

temp = temp->next;

}

}

int List::operator()() { return size; }

**Функция main().**

void main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

cout << "Введите команду\nДоступные команды: " << endl;

cout << "m создать группу\n+ добавить объект в группу\n- удалить объект из группы\n";

cout << "s просмотр группы\nz просмотр имени группы по индексу\nq конец работы\n";

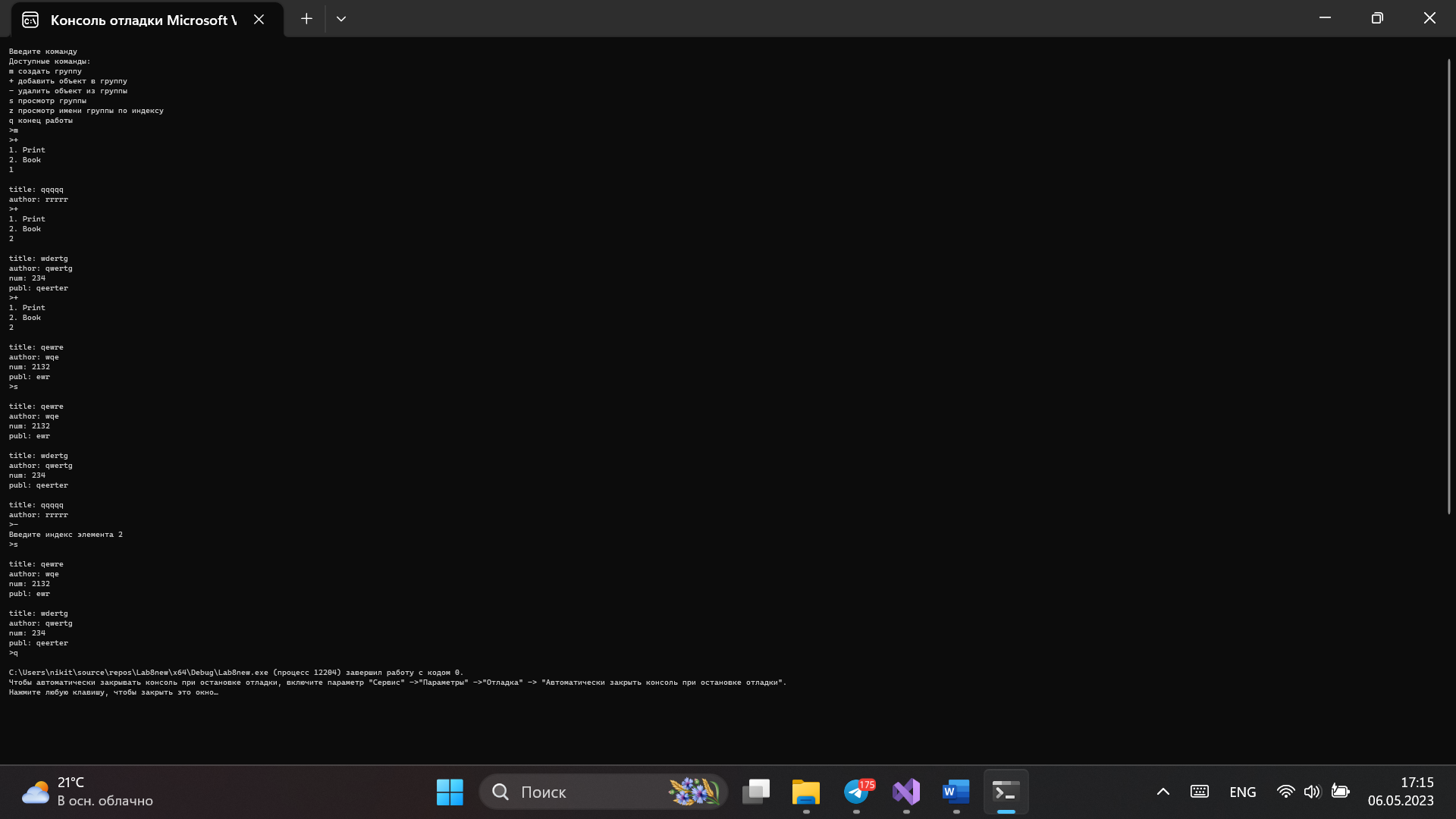
Dialog D;

D.Execute();

}

**Объяснение результатов работы программы.**

Данная программа демонстрирует, управления событиями с помощью команд, при вводе команды m создается группа – список, далее с помощью команды +, добавляются элементы базового класса Book в список, так же можно удалить элемент, просмотреть, а с помощью команды z, можем найти группу по индексу.



**Ответы на контрольные вопросы**

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

Класс-группа - это группа классов, объединенных по общим свойствам или функциональности.

class Figure {

protected:

int x;

int y;

…

};

class Circle : public Figure {

private:

int radius;

…

};

class Triangle : public Figure {

private:

int base;

int height;

…

};

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

class List {

private:

int \*data;

int size;

int count;

public:

List(); // конструктор без параметров

List(int size); // конструктор с параметром

List(const List& other); // конструктор копирования

~List(); // деструктор

void push\_back(int value); // добавление элемента в конец списка

void pop\_back(); // удаление элемента из конца списка

int get\_size(); // получение размера списка

int operator[](int index); // перегрузка оператора [] для доступа к элементу по индексу

};

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

List::List() {

size = 10;

data = new int[size];

count = 0;

}

List::List(int size) {

this->size = size;

data = new int[size];

count = 0;

}

List::List(const List& other) {

this->size = other.size;

this->count = other.count;

data = new int[size];

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

data[i] = other.data[i];

}

}

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

List::~List() {

delete[] data;

}

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

int List::operator[](int index) {

if (index < 0 || index >= count) {

// обработка ошибки

}

return data[index];

}

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

Группа дает иерархию по входным или выходным данным. Например, группа классов графиков, имеющих общую функцию вывода данных на экран, может давать иерархию типов графиков по их структуре.

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

Абстрактный класс во главе иерархии позволяет определить общие свойства и функциональность для всех классов в группе, не создавая экземпляры этого класса. Также он является базовым классом для всех классов в группе.

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

Событие - это сообщение от объекта программы о том, что произошло какое-то событие, на которое следует реагировать. События могут использоваться для уведомления других объектов о происходящих действиях, например, для синхронизации действий объектов или для обновления интерфейса.

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

Событие-сообщение должно иметь тип, содержание и источник. Тип сообщения определяет, что произошло, содержание сообщения содержит дополнительную информацию о происшествии, а источник сообщения указывает на объект, который послал сообщение.

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

Пример структуры, описывающей событие:

struct MouseEvent {

int x;

int y;

};

struct KeyDownEvent {

char key;

};

struct MessageEvent {

std::string message;

};

struct Event {

int type;

union {

MouseEvent mouse\_event;

KeyDownEvent keydown\_event;

MessageEvent message\_event;

};

};

**11. Задана структура события**

**struct TEvent**

**{**

**int what;**

**union**

**{**

**MouseEventType mouse;**

**KeyDownEvent keyDown;**

**MessageEvent message;**

**}**

**};**

**Какие значения, и в каких случаях присваиваются полю what?**

В поле what записывается код события, который может иметь различное значение в зависимости от типа события. Например, если это MouseEventType, то поле what может принимать значения, указывающие на тип события мыши, например, код 1 может означать нажатие левой кнопки мыши, а код 2 - нажатие правой кнопки мыши. Если это KeyDownEvent, то поле what может содержать код нажатой клавиши на клавиатуре. Если это MessageEvent, то поле what может содержать код сообщения, например, код ошибки или информационного сообщения.

**12. Задана структура события**

**struct TEvent**

**{**

**int what;//тип события**

**union**

**{**

**int command;//код комманды**

**struct//параметры команды**

**{**

**int message;**

**int a;**

**};**

**};**

**};**

**Какие значения, и в каких случаях присваиваются полю command?**

Полю command могут присваиваться коды команд, которые должны быть обработаны. Например, если это приложение, связанное с управлением файлами, то код команды может означать действие, которое должно быть выполнено с файлом или каталогом (например, создание, удаление, копирование, перемещение и т.д.).

**13. Задана структура события**

**struct TEvent**

**{**

**int what;//тип события**

**union**

**{**

**int command;//код комманды**

**struct//параметры команды**

**{**

**int message;**

**int a;**

**};**

**};**

**};**

**Для чего используются поля a и message?**

Поля message и а могут использоваться для передачи параметров команды. Например, если это команда создания нового файла, то поле message может содержать имя файла, а поле а - дополнительные параметры создания (например, права доступа, размер файла и т.д.).

**14. Какие методы необходимы для организации обработки сообщений?**

Для организации обработки сообщений необходимы методы-обработчики, которые занимаются обработкой событий, полученных от других объектов программы. Также может потребоваться метод для подписки на события, метод для отписки от событий и т.д.

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

Главный цикл обработки событий-сообщений представляет собой бесконечный цикл, в котором происходит ожидание событий от объектов программы, их обработка и вызов соответствующих методов-обработчиков. При получении нового события оно помещается в очередь событий, откуда извлекается в ходе выполнения цикла.

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

Метод ClearEvent() служит для очистки данных, связанных со событием (например, очистки буфера, из которого было прочитано событие).

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

Метод HandleEvent() служит для обработки полученного события и вызывает соответствующий метод-обработчик для данного события.

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

Метод GetEvent() служит для получения нового события из очереди событий.

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

Поле EndState может использоваться для определения состояния объекта после окончания обработки события. Класс, содержащий это поле, может быть любым, в зависимости от задачи программы.

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

Функция Valid() может использоваться для проверки корректности данных, переданных в качестве параметров, например, при обработке событий. Она может вернуть логическое значение true, если данные корректны, и false, если нет.