



МИНОБРНАУКИ РОССИИ

**федеральное государственное бюджетное образовательное учреждение
высшего образования
«Новосибирский государственный университет экономики и управления «НИНХ»
(ФГБОУ ВО «НГУЭУ», НГУЭУ)**

Кафедра информационных технологий

КУРСОВАЯ РАБОТА

Консольное приложение «Заказы на производстве»

Дисциплина: Языки программирования

Ф.И.О студента: Губарева Мария Алексеевна

Направление: 02.03.02 Фундаментальная информатика и информационные технологии

Направленность (профиль): Программная инженерия

Номер группы: ФИ202

Номер зачетной книжки: 220112

Проверил: Ковригин Алексей Викторович, кандидат педагогических наук,
доцент

Новосибирск 2024



МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Новосибирский государственный университет экономики и управления «НИНХ»
(ФГБОУ ВО «НГУЭУ», НГУЭУ)

Кафедра информационных технологий

ЗАДАНИЕ
на курсовую работу

Тема: Консольное приложение «Заказы на производстве»

ФИО студента: Губарева Мария Алексеевна

Группа: ФИ202

Перечень подлежащих разработке вопросов и календарный график

№ п/п	Наименование вопросов, подлежащих разработке (этапы работы)	Срок выполнения
1.	Ознакомление с заданием, уточнение требований	06.05.2024
2.	Разработка и тестирование программного приложения согласно варианту	08.05.2024
3.	Оформление текста курсовой работы и технического задания согласно стандарту	22.05.2024
4.	Защита текста курсовой работы преподавателю	27.05.2024

Дата выдачи задания «__» _____ 20__ года

Срок сдачи работы «__» _____ 20__ года

Преподаватель _____
(фамилия и инициалы преподавателя)

(подпись)

Задание получил студент _____
(фамилия и инициалы студента)

(подпись)



МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Новосибирский государственный университет экономики и управления «НИНХ»
(ФГБОУ ВО «НГУЭУ», НГУЭУ)

Кафедра информационных технологий

ЗАЯВЛЕНИЕ
о самостоятельном характере выполненной работы

Я, Губарева Мария Алексеевна, студентка группы ФИ202, направления подготовки 02.03.02
Фундаментальная информатика и информационные технологии,

направленности (профиля) Программная инженерия,
заявляю, что в моей курсовой работе, выполненной на тему: Консольное приложение
«Заказы на производстве», не содержится элементов плагиата.

Все заимствования из печатных и электронных источников, а также из защищенных ранее
письменных работ, кандидатских и докторских диссертаций имеют соответствующие
ссылки.

«_____» _____ 20__ г.

(подпись)

И.О. Фамилия

Результаты проверки в системе «Антиплагиат»

Доля авторского текста (оригинальности) в результате автоматизированной проверки
составила 89 %.

Руководитель курсовой работы _____
(уч. степень, должность, Фамилия И.О.)

«_____» _____ 20__ г.

(подпись)

СОДЕРЖАНИЕ

1. РЕАЛИЗАЦИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ	3
1.1. Запуск и завершение работы приложения	3
1.2. Обработка и выполнение команд	6
1.1.1. Вывести список заказов	6
1.1.2. Создать новый заказ	6
1.1.3. Удалить заказ	8
1.1.4. Редактирование заказа	8
1.1.5. Добавить заказы в файл	10
1.1.6. Добавить заказы из файла	11
1.1.7. Структура заказа	12
2. СПИСОК ТЕСТОВЫХ СЛУЧАЕВ ДЛЯ РУЧНОГО ТЕСТИРОВАНИЯ ПРОГРАММЫ	16
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ	18

1. РЕАЛИЗАЦИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ

Поставленной задачей для выполнения расчетно-графической работы является разработка консольного приложения по управлению объектами данных по теме «Заказы на производстве» с использованием объектно-ориентированного программирования и реализацией изученных паттернов на языке программирования C++: необходимо создать минимум 3 объекта сущности разных видов и указать взаимодействие между объектами.

Приложение предполагает работу меню из следующих пунктов:

1. Вывод всех объектов;
2. Добавление нового объекта (ввод с клавиатуры);
3. Удаление выбранного пользователем объекта;
4. Загрузка объектов из файла;
5. Сохранение объектов в файл;
6. Редактирование объектов;
7. Выход из программы.

1.1. Запуск и завершение работы приложения

Обратимся к листингу 1 и листингу 2. В программном коде используется паттерн программирования MVC (Model-View-Controller, «Модель-Представление-Контроллер») — схема разделения данных приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер). Функция `main()` импортирует `class controller`, создает объект контроллера и запускает программу. Контроллер, в свою очередь, запускает бесконечный цикл, который выводит номера команд и запрашивает номер команды, необходимый пользователю.

Листинг 1. Контроллер (Controller)

```
#include <iostream>
#include "production_orders.cpp"

using namespace std;

// вывод всех приколов
class controller {
private:
    production_orders current_order;

public:
    controller() {}
    void execute() {
        int run = 1;
        while (run) {
            menu();
            int var = 0;
            while (var < 1 || var > 7) {
                cout << "Введите номер команды: ";
                cin >> var;
            }
            if (var == 7) {
                run = 0;
            }
            if (var == 1) {
                current_order.output();
            }
            if (var == 2) {
                current_order.add ();
                cout << "Заказ создан ^_^ \n";
            }
            if (var == 3) {
                current_order.output();
                cout << "Введите номер заказа, который
необходимо удалить:\n";
                int n;
                cin >> n;
                current_order.delete_order(n);
            }
            if (var == 4) {
                cout << "Выберите заказ, который необходимо
отредактировать: \n";
                int vibor = 0;
                while (vibor < 1 || vibor >
current_order.get_n() + 1) {
                    current_order.output();
                    cout << current_order.get_n() + 1 << ". "
<< "Отмена редактирования!!! \n";
```

```

        cin >> vibor;
    }
    current_order.redact_order(vibor);
    cout << "Заказ отредактирован ^_^ \n";
}
if (var == 5) {
    current_order.add_to_file();
    cout << "Заказы сохранены в файл ^_^ \n";
}
if (var == 6) {
    current_order.add_from_file();
    cout << "Заказы из файла загружены ^_^ \n";
}
}

}

void menu() {
    cout << "Команды: " << endl;
    cout << "1. Вывести список заказов" << endl;
    cout << "2. Создать новый заказ" << endl;
    cout << "3. Удалить заказ" << endl;
    cout << "4. Редактирование заказа" << endl;
    cout << "5. Добавить заказы в файл" << endl;
    cout << "6. Загрузить заказы из файла" << endl;
    cout << "7. Выход из программы" << endl;
}

};

```

Листинг 2. Функция main()

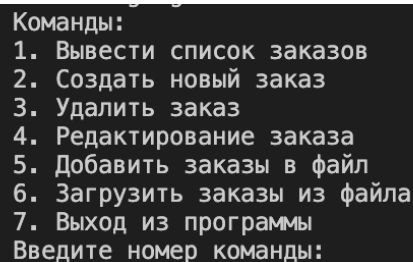
```

#include <iostream>
#include "controller.cpp"

using namespace std;

int main() {
    controller pognal;
    pognal.execute();
}

```



```

Команды:
1. Вывести список заказов
2. Создать новый заказ
3. Удалить заказ
4. Редактирование заказа
5. Добавить заказы в файл
6. Загрузить заказы из файла
7. Выход из программы
Введите номер команды:

```

Рис. 1 – меню приложения

1.2. Обработка и выполнение команд

Хранение заказов и работа с ними реализуются в классе `productions_orders`. В частных полях: переменная `n` – количество заказов; вектор `orders` содержит список заказов; массивы строк `students` и `uslugi` содержат названия групп студентов и услуг соответственно.

В конструкторе `productionorders()` устанавливается значение переменной `n` равным 1, и создается значение по умолчанию для заказа `order_structure` с данными "Алексей", "Ковригин", "8-999-999-99-99", "Студенты ФИ", 5, "знать C++". Этот заказ добавляется в список заказов с помощью метода `emplaceback()`.

1.1.1. Вывести список заказов

Рассмотрим листинг 3. Метод класса `void output()` выводит информацию о заказах. Используется цикл для перебора всех заказов в массиве `orders` и вызывает метод `output` для каждого заказа, который выводит информацию о текущем заказе. Для каждого заказа выводится его порядковый номер.

Листинг 3. Вывод заказов

```
void output() {
    for (int i = 0; i < n; i++) {
        order_structure current_order = orders[i];
        cout << i + 1 << ". ";
        current_order.output();
    }
}
```

1.1.2. Создать новый заказ

Листинг 4. Создание заказа

```
void add () {
    string gruppa, usluga, name, surname, number;
    int grade;
    cout << "Выберите группу студентов:" << endl;
    int vibor = 0;
    for (int i = 0; i < 5; i++) {
        cout << i + 1 << ". " << students[i] << endl;
    }
    while (vibor < 1 || vibor > 5) {
        cin >> vibor;
    }
    gruppa = students[vibor - 1];

    cout << "Укажите оценку, на которую должна быть
выполнена услуга (1-5): ";
    cin >> grade;
    vibor = 0;

    cout << "Выберите услугу:" << endl;
    for (int i = 0; i < 5; i++) {
        cout << i + 1 << ". " << uslugi[i] << endl;
    }
    while (vibor < 1 || vibor > 5) {
        cin >> vibor;
    }
    usluga = uslugi[vibor - 1];

    cout << "Имя: ";
    cin >> name;

    cout << "Фамилия: ";
    cin >> surname;

    cout << "Номер телефона для связи: ";
    cin >> number;

    order_structure new_order(name, surname, number,
    gruppa, grade, usluga); // создание 1 заказа
    orders.emplace_back(new_order); // добавление в конец
    списка заказов
    n++;
}
```

Обратимся к листингу 4. Метод класса add добавляет новый заказ. Сначала запрашивается информация о группе студентов, услуге, оценке, имени, фамилии, и номере телефона. После ввода всех данных создается новый объект order_structure, затем новый заказ добавляется в конец списка заказов

`orders` с помощью метода `emplaceback`, количество заказов `n` увеличивается на 1.

1.1.3. Удалить заказ

Листинг 5. Удаление заказа

```
void delete_order(int k) {  
    orders.erase(orders.begin() + k - 1);  
    n--;  
}
```

Метод класса `delete_order` удаляет заказ из списка заказов: принимает параметр `k` – индекс заказа, который нужно удалить. Сначала с помощью метода `erase` удаляется заказ из списка `orders` по индексу `k - 1`, затем количество заказов `n` уменьшается на 1.

1.1.4. Редактирование заказа

Листинг 6. Редактирование заказа

```
void redact_order(int n) {  
    orders[n - 1].redact();  
}
```

Метод класса `redact_order (int n)` обращается к заказу с индексом `n-1` (так как индексация начинается с нуля) и вызывает метод `redact` (см. листинг 7).

Листинг 7. Редактирование заказа

```
void redact() {  
    cout << "Выберите пункт заказа, который необходимо  
отредактировать:\n";  
    cout << "1. Группа студентов \n";  
    cout << "2. Оценка \n";  
    cout << "3. Услуга \n";  
    cout << "4. Информация о клиенте \n";  
    int vibor = 0;  
    while (vibor < 1 || vibor > 4){  
        cin >> vibor;  
    }  
    switch (vibor) {
```

```

        case 1: {
            cout << "Выберите группу студентов:" << endl;
            int vibor = 0;
            for (int i = 0; i < 5; i++) {
                cout << i + 1 << ". " << students[i] << endl;
            }
            while (vibor < 1 || vibor > 6) {
                cin >> vibor;
            }
            gruppa = students[vibor - 1];
            break;
        }

        case 2: {
            cout << "Введите оценку, на которую должна быть
оказана услуга (1-5): ";
            cin >> grade;
            break;
        }

        case 3: {
            int vibor = 0;
            cout << "Выберите услугу: " << endl;
            for (int i = 0; i < 5; i++) {
                cout << i + 1 << ". " << uslugi[i] << endl;
            }
            while (vibor < 1 || vibor > 5) {
                cin >> vibor;
            }
            usluga = uslugi[vibor - 1];
        }

        case 4: {
            string name, surname, number;
            cout << "Имя: ";
            cin >> name;
            cout << "Фамилия: ";
            cin >> surname;
            cout << "Номер телефона для связи: ";
            cin >> number;
            client = clients(name, surname, number);
            break;
        }
    }
}

```

Обратимся к листингу 7. Выводится меню с вариантами для редактирования: группы студентов, оценки, услуги и информации о клиенте. После выбора пользователем одного из пунктов вводится соответствующая информация:

1. Если выбран пункт «Группа студентов», выводится список групп студентов.
2. Если выбран пункт «Оценка», запрашивается ввод новой оценки.
3. Если выбран пункт «Услуга», отображается список доступных услуг, из которого пользователь может выбрать нужную для заказа.
4. Если выбран пункт «Информация о клиенте», запрашивается информация о клиенте, включая имя, фамилию и номер телефона.

1.1.5. Добавить заказы в файл

Листинг 8. Добавление текущих заказов в файл

```
void add_to_file() {
    ofstream output_file;
    output_file.open("orders_not_stupid_students.txt",
ios::app);
    if (output_file.is_open()) {
        string order_information;
        for (int i = 0; i < n; i++) {
            order_structure current_order = orders[i];
            order_information
current_order.get_client().get_name()          +      "^ ^"      +
current_order.get_client().get_surname()        +      " ^ ^"      +
current_order.get_client().get_number()          +      " ^ ^"      +
current_order.get_gruppa()                      +      " ^ ^"      +
to_string(current_order.get_grade())              +      " ^ ^"      +
current_order.get_usluga();
            output_file << order_information << endl;
        }
        output_file.close();
    }
}
```

Рассмотрим листинг 8. Метод класса `add_to_file` отвечает за добавление информации о заказах в текстовый файл. Открывается файл `"orders_not_stupid_students.txt"`, перебираются все заказы и сохраняется информация о каждом из них в формате `"имя^_фамилия^_номер^_группа^_оценка^_услуга"`. После этого файл закрывается.

1.1.6. Добавить заказы из файла

Листинг 9. Добавление заказов из файла

```
void add_from_file() {
    ifstream input_file("orders_not_stupid_students.txt");
    if (input_file.is_open()){
        string new_string;
        while (getline(input_file, new_string)) {
            string name = new_string.substr(0,
new_string.find("^_"));
            new_string =
new_string.substr(new_string.find("^_") + 3, new_string.length()
- new_string.find("^_"));

            string surname = new_string.substr(0,
new_string.find("^_"));
            new_string =
new_string.substr(new_string.find("^_") + 3, new_string.length()
- new_string.find("^_"));

            string number = new_string.substr(0,
new_string.find("^_"));
            new_string =
new_string.substr(new_string.find("^_") + 3, new_string.length()
- new_string.find("^_"));

            string gruppa = new_string.substr(0,
new_string.find("^_"));
            new_string =
new_string.substr(new_string.find("^_") + 3, new_string.length()
- new_string.find("^_"));

            int grade = stof(new_string.substr(0,
new_string.find("^_")));
            new_string =
new_string.substr(new_string.find("^_") + 3, new_string.length()
- new_string.find("^_"));

            string usluga = new_string.substr(0,
new_string.length());
            order_structure new_order(name, surname,
number, gruppa, grade, usluga);
            orders.emplace_back(new_order);
            n++;
        }
    }
    input_file.close();
}
```

Обратимся к листингу 9. Сначала открывается файл и проверяется, открыт ли он для чтения, затем файл читается построчно с помощью функции `getline` и извлекается информация о заказах из каждой строки. Каждая строка представляет собой данные о заказе, разделенные символами "`^_`". На каждой итерации цикла `while` происходит извлечение последовательности строк с помощью функций `substr` и `find`. Извлеченная информация используется для создания объекта `order_structure` с помощью конструктора `order_structure(name, surname, number, gruppa, grade, usluga)`. Каждый новый заказ добавляется в конец списка заказов с помощью метода `emplace_back`, а затем увеличивается счетчик заказов `n`. После завершения цикла чтения файла, файл закрывается.

1.1.7. Структура заказа

Информация о клиенте хранится в классе `client` (см. листинг 10): он имеет три приватных поля: `name`, `surname` и `number`, которые представляют имя, фамилию и номер телефона клиента. Помимо этого, класс имеет несколько публичных методов: `get_name()`, `get_surname()` и `get_number()`, они используются для получения значений полей `name`, `surname` и `number` соответственно. Конструктор `clients()` устанавливает значения по умолчанию для имени, фамилии и номера телефона. Конструктор `clients(string name, string surname, string number)` инициализирует поля класса переданными значениями. Метод `output()` выводит информацию о клиенте в формате "фамилия имя, телефон для связи: номер".

Информация о заказе хранится в классе `order_structure` (см. листинг 11): он имеет приватные поля `gruppa`, `usluga`, `grade` и `client`, которые представляют группу студентов, услугу, оценку и информацию о клиенте. Массивы строк `students` и `uslugi` содержат названия групп студентов и услуг. Класс содержит два конструктора: один устанавливает значения по

умолчанию для группы студентов, оценки и услуги, а второй инициализирует эти значения с помощью переданных аргументов, а также создает новый объект клиента. Методы `getgruppa()`, `getgrade()`, `getusluga()` и `getclient()`, которые используются для получения соответствующих значений полей класса. Метод `output()` выводит информацию о заказе, а метод `redact()` позволяет редактировать информацию о заказе: группу студентов, оценку, услугу и информацию о клиенте.

Листинг 10. Информация о клиенте

```
class clients {
private:
    string name, surname, number;
public:
    string get_name() {
        return name;
    }
    string get_surname() {
        return surname;
    }
    string get_number() {
        return number;
    }

    clients() {
        name = "Алексей";
        surname = "Ковригин";
        number = "81234567890";
    }

    clients(string name, string surname, string number) :
name(name), surname(surname), number(number) {}

    void output() {
        cout << surname << " " << name << ", телефон для связи:
" << number << endl;
    }
};
```

Листинг 11. Структура заказа

```
class order_structure {
private:
    string gruppa, usluga;
    int grade;
    clients client;
```

```

string students[5] = {
    "Студенты ФИ", "Студенты БИ", "Студенты ИН",
    "Студенты ИСИТ", "Студенты СТ"
};
string uslugi[5] = {
    "Знать C++", "Знать Python",
    "Знать CSS", "Знать HTML", "Знать JavaScript"
};

public:
    order_structure() {
        gruppa = "Студенты ФИ";
        grade = 5;
        usluga = "Знать C++";
    }
    order_structure(string name, string surname, string number,
string gruppa, int grade, string usluga) :
        client(*(new clients(name, surname, number))),
        grade(grade), gruppa(gruppa), usluga(usluga) {}

    string get_gruppa() {
        return gruppa;
    }

    int get_grade() {
        return grade;
    }

    string get_usluga() {
        return usluga;
    }

    clients get_client() {
        return client;
    }

    void output() {
        client.output();
        cout << gruppa << " должны на оценку " << grade << " "
<< usluga << endl;
    }

    void redact() {
        cout << "Выберите пункт заказа, который необходимо
отредактировать:\n";
        cout << "1. Группа студентов \n";
        cout << "2. Оценка \n";
        cout << "3. Услуга \n";
        cout << "4. Информация о клиенте \n";
        int vibor = 0;
        while (vibor < 1 || vibor > 4){
            cin >> vibor;
        }
    }

```



```

switch (vibor) {

    case 1: {
        cout << "Выберите группу студентов:" << endl;
        int vibor = 0;
        for (int i = 0; i < 5; i++) {
            cout << i + 1 << ". " << students[i] << endl;
        }
        while (vibor < 1 || vibor > 6) {
            cin >> vibor;
        }
        gruppa = students[vibor - 1];
        break;
    }

    case 2: {
        cout << "Введите оценку, на которую должна быть
оказана услуга (1-5): ";
        cin >> grade;
        break;
    }

    case 3: {
        int vibor = 0;
        cout << "Выберите услугу: " << endl;
        for (int i = 0; i < 5; i++) {
            cout << i + 1 << ". " << uslugi[i] << endl;
        }
        while (vibor < 1 || vibor > 5) {
            cin >> vibor;
        }
        usluga = uslugi[vibor - 1];
    }

    case 4: {
        string name, surname, number;
        cout << "Имя: ";
        cin >> name;
        cout << "Фамилия: ";
        cin >> surname;
        cout << "Номер телефона для связи: ";
        cin >> number;
        client = clients(name, surname, number);
        break;
    }

}

};

```

2. СПИСОК ТЕСТОВЫХ СЛУЧАЕВ ДЛЯ РУЧНОГО ТЕСТИРОВАНИЯ ПРОГРАММЫ

Программный код приложения следует проверить на правильность выполнения команд и общую функциональность. В таблице 1 перечислены возможные действия пользователя и ожидаемый результат их выполнения.

Таблица 1

Тест-план для ручного тестирования программы

№	Описание тестового случая	Ожидаемый результат
1	Запуск приложения	Вывод списка возможных команд и приглашение на взаимодействие: «Команды: 1. Вывести список заказов 2. Создать новый заказ 3. Удалить заказ 4. Редактирование заказа 5. Добавить заказы в файл 6. Загрузить заказы из файла 7. Выход из программы Введите номер команды: »
2	Ввод команды «Создать новый заказ»	Запрос данных об объекте: 1. Выберите группу студентов 2. Введите оценку, на которую услуга должна быть оказана 3. Выберите услугу 4. Имя 5. Фамилия 6. Номер телефона для связи После успешного ввода необходимых данных для добавления объекта сообщение «Заказ создан ^_^».
3	Ввод команды «Удалить заказ»	Вывод пронумерованного списка возможных объектов для удаления. После ввода номера объекта сообщение «Заказ удален ^_^».
4	Ввод команды «Редактирование заказа»	Вывод пронумерованного списка возможных параметров редактирования. Пользователь вводит необходимые данные, изменения сохраняются.

5	Ввод команды «Добавить заказы в файл»	Текущие заказы сохраняются в файл в формате "имя^_^фамилия^_^номер^_^группа^_^оценка^_^услуга".
6	Ввод команды «Загрузить заказы из файла»	К текущим заказам добавляются заказы из файла.
7	Ввод команды «Выход»	Завершение работы программы.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Основы C++ / [Электронный ресурс] // Яндекс Образование : [сайт].
— URL: <https://education.yandex.ru/handbook/cpp>
2. Васильев А. Н. Программирование на C++ в примерах и задачах /
Васильев А. Н.. – Москва : Эксмо, 2023. – 368 с.