#include <iostream> // Библиотека для ввода/вывода

#include <vector> // Библиотека для работы с векторами

#include <string> // Библиотека для работы со строками

#include <fstream> // Библиотека для работы с файлами

#include <stdexcept> // Библиотека для обработки исключений

#include <iomanip> // Библиотека для форматированного вывода

#include <algorithm> // Библиотека для использования стандартных алгоритмов

#include <windows.h> // Библиотека для работы с Windows API

using namespace std; // Использование стандартного пространства имен

// Функция для проверки ввода

template<typename T>

T getInput(const string& prompt) {

T value;

while (true) {

cout << prompt; // Вывод подсказки

if (cin >> value) {

cin.ignore(100, '\n'); // Игнорирование лишних символов

return value; // Возврат введенного значения

}

cin.clear(); // Очистка состояния потока

cin.ignore(100, '\n'); // Пропуск некорректного ввода

cout << "Invalid input. Please try again.\n"; // Сообщение об ошибке

}

}

// Базовый класс

class ConfectioneryItem {

protected:

string name; // Название изделия

double price; // Цена изделия

string ingredients; // Список ингредиентов

int calories; // Калорийность изделия

public:

// Конструктор с параметрами

ConfectioneryItem(const string& name, double price, int calories)

: name(name), price(price), calories(calories) {}

virtual ~ConfectioneryItem() = default; // Виртуальный деструктор

virtual void displayInfo() const = 0; // Чисто виртуальная функция для отображения информации

// Геттеры для получения данных

string getName() const { return name; }

double getPrice() const { return price; }

int getCalories() const { return calories; }

// Установка ингредиентов

void setIngredients(const string& newIngredients) {

ingredients = newIngredients;

}

// Геттер для ингредиентов

const string& getIngredients() const {

return ingredients;

}

// Сеттеры для изменения данных

virtual void setName(const string& newName) { name = newName; }

virtual void setPrice(double newPrice) { price = newPrice; }

virtual void setCalories(int newCalories) { calories = newCalories; }

};

// Производный класс первого уровня - Выпечка

class BakedGood : public ConfectioneryItem {

protected:

double weight; // Вес изделия

public:

// Конструктор с параметрами

BakedGood(const string& name, double price, double weight, int calories)

: ConfectioneryItem(name, price, calories), weight(weight) {}

// Геттер для веса

double getWeight() const { return weight; }

// Сеттер для веса

void setWeight(double newWeight) { weight = newWeight; }

};

// Производный класс второго уровня - Торт

class Cake : public BakedGood {

public:

// Конструктор с параметрами

Cake(const string& name, double price, double weight, int calories)

: BakedGood(name, price, weight, calories) {}

// Переопределение метода отображения информации

void displayInfo() const override {

cout << "Cake: " << name << "\nPrice: " << price << " USD\nWeight: " << weight << " kg\n";

}

};

// Производный класс третьего уровня - Свадебный торт

class WeddingCake : public Cake {

private:

int tiers; // Количество ярусов

public:

// Конструктор с параметрами

WeddingCake(const string& name, double price, double weight, int tiers, int calories)

: Cake(name, price, weight, calories), tiers(tiers) {}

// Геттер для количества ярусов

int getTiers() const { return tiers; }

// Сеттер для количества ярусов

void setTiers(int newTiers) { tiers = newTiers; }

// Переопределение метода отображения информации

void displayInfo() const override {

cout << "Wedding Cake: " << name << "\nPrice: " << price << " USD\nWeight: " << weight << " kg\nTiers: " << tiers << "\n";

}

};

// Производный класс - Печенье

class Cookie : public ConfectioneryItem {

private:

string shape; // Форма печенья

public:

// Конструктор с параметрами

Cookie(const string& name, double price, const string& shape, int calories)

: ConfectioneryItem(name, price, calories), shape(shape) {}

// Геттер для формы

string getShape() const { return shape; }

// Сеттер для формы

void setShape(const string& newShape) { shape = newShape; }

// Переопределение метода отображения информации

void displayInfo() const override {

cout << "Cookie: " << name << "\nPrice: " << price << " USD\nShape: " << shape << "\n";

}

};

// Класс для управления ассортиментом

class ConfectioneryManager {

private:

vector<ConfectioneryItem\*> items; // Список изделий

public:

// Деструктор для очистки памяти

~ConfectioneryManager() {

for (auto item : items) {

delete item;

}

}

// Метод для добавления нового изделия

void addItem(ConfectioneryItem\* item) {

items.push\_back(item);

}

// Метод для удаления изделия по имени

void removeItem(const string& name) {

auto it = find\_if(items.begin(), items.end(),

[&name](const ConfectioneryItem\* item) { return item->getName() == name; });

if (it != items.end()) {

delete \*it;

items.erase(it);

}

}

void displayAllItems() const {

// Вывод шапки таблицы с разделителями

cout << "\n" << string(120, '=') << "\n";

cout << left

<< setw(30) << "Name" // Название изделия

<< setw(15) << "Price (USD)" // Цена изделия

<< setw(15) << "Weight (kg)" // Вес изделия

<< setw(15) << "Type" // Тип изделия

<< setw(15) << "Calories" // Калорийность

<< setw(15) << "Additional" // Дополнительная информация

<< "Ingredients\n"; // Ингредиенты

cout << string(120, '-') << "\n"; // Разделитель

// Итерация по всем изделиям

for (const auto item : items) {

cout << left

<< setw(30) << item->getName() // Вывод названия изделия

<< setw(15) << fixed << setprecision(2) << item->getPrice() // Цена изделия

<< setw(15);

// Если изделие - торт, выводим его вес

if (auto cake = dynamic\_cast<Cake\*>(item)) {

cout << fixed << setprecision(3) << cake->getWeight();

} else {

cout << "-"; // Если вес не применим, выводим "-"

}

string itemType; // Тип изделия

string additional; // Дополнительная информация

// Проверяем, является ли изделие свадебным тортом

if (auto weddingCake = dynamic\_cast<WeddingCake\*>(item)) {

itemType = "Wedding Cake"; // Указываем тип изделия

additional = to\_string(weddingCake->getTiers()) + " tiers"; // Количество ярусов

} else if (dynamic\_cast<Cake\*>(item)) {

itemType = "Cake"; // Если обычный торт

} else if (auto cookie = dynamic\_cast<Cookie\*>(item)) {

itemType = "Cookie"; // Если изделие - печенье

additional = cookie->getShape(); // Указываем форму печенья

}

cout << left << setw(15) << itemType // Вывод типа изделия

<< setw(15) << item->getCalories() // Калорийность

<< setw(15) << additional; // Дополнительная информация

// Выводим список ингредиентов

cout << item->getIngredients() << "\n";

}

cout << string(120, '=') << "\n"; // Разделительная линия

}

ConfectioneryItem\* findItem(const string& name) {

// Поиск изделия по имени

auto it = find\_if(items.begin(), items.end(),

[&name](const ConfectioneryItem\* item) { return item->getName() == name; });

return (it != items.end()) ? \*it : nullptr; // Возврат найденного изделия или nullptr

}

void saveToFile(const string& filename) const {

// Открытие файла для записи

ofstream file(filename);

if (!file) {

throw runtime\_error("Failed to open file for writing"); // Обработка ошибки открытия файла

}

// Итерация по всем изделиям

for (const auto item : items) {

file << item->getName() << "|" << item->getPrice() << "|" << item->getCalories(); // Сохранение общих данных

// Если изделие - торт, сохраняем его вес

if (auto cake = dynamic\_cast<Cake\*>(item)) {

file << "|Cake|" << cake->getWeight();

if (auto weddingCake = dynamic\_cast<WeddingCake\*>(cake)) {

file << "|" << weddingCake->getTiers(); // Сохраняем количество ярусов

} else {

file << "|0"; // Если не свадебный торт

}

} else if (auto cookie = dynamic\_cast<Cookie\*>(item)) {

file << "|Cookie|" << cookie->getShape(); // Сохраняем форму печенья

}

file << "|" << item->getIngredients() << "\n"; // Сохраняем ингредиенты

}

file.close(); // Закрываем файл

}

void loadFromFile(const string& filename) {

// Открытие файла для чтения

ifstream file(filename);

if (!file) {

cout << "File " << filename << " not found. Creating a new file.\n"; // Если файл не найден

return;

}

// Очищаем текущий список изделий

for (auto item : items) {

delete item; // Удаляем изделие

}

items.clear(); // Очищаем контейнер

string line; // Строка для чтения данных

while (getline(file, line)) {

istringstream iss(line); // Создаем поток для разбора строки

string name, type, shape, ingredientsStr;

double price, weight;

int calories, tiers;

getline(iss, name, '|'); // Читаем название

iss >> price; // Читаем цену

iss.ignore(); // Пропускаем разделитель

iss >> calories; // Читаем калории

iss.ignore(); // Пропускаем разделитель

getline(iss, type, '|'); // Читаем тип изделия

ConfectioneryItem\* item = nullptr;

if (type == "Cake") {

iss >> weight; // Читаем вес торта

iss.ignore(); // Пропускаем разделитель

iss >> tiers; // Читаем количество ярусов

if (tiers > 0) {

item = new WeddingCake(name, price, weight, tiers, calories); // Свадебный торт

} else {

item = new Cake(name, price, weight, calories); // Обычный торт

}

} else if (type == "Cookie") {

getline(iss, shape, '|'); // Читаем форму печенья

item = new Cookie(name, price, shape, calories); // Создаем печенье

}

if (item) {

// Читаем ингредиенты как одну строку

getline(iss, ingredientsStr);

if (!ingredientsStr.empty() && ingredientsStr[0] == '|') {

ingredientsStr.erase(0, 1); // Удаляем начальный разделитель

}

item->setIngredients(ingredientsStr); // Устанавливаем ингредиенты

addItem(item); // Добавляем изделие в список

}

}

file.close(); // Закрываем файл

cout << "Data successfully loaded from file " << filename << "\n"; // Вывод сообщения об успешной загрузке

}

// Класс для управления заказами

class Order { // Класс, отвечающий за управление заказами

private:

vector<ConfectioneryItem\*> items; // Вектор для хранения кондитерских изделий

vector<int> quantities; // Вектор для хранения количества каждого изделия

public:

void addItem(ConfectioneryItem\* item, int quantity) { // Метод для добавления изделия в заказ

items.push\_back(item); // Добавляем изделие в список

quantities.push\_back(quantity); // Добавляем количество изделия

}

double calculateTotal() const { // Метод для подсчета общей стоимости заказа

double total = 0; // Инициализируем общую стоимость

for (size\_t i = 0; i < items.size(); ++i) { // Проходим по всем изделиям

total += items[i]->getPrice() \* quantities[i]; // Умножаем цену на количество и добавляем к общей стоимости

}

return total; // Возвращаем общую стоимость

}

void displayOrder() const { // Метод для отображения текущего заказа

cout << "Your order:\n"; // Выводим заголовок заказа

cout << setw(20) << "Name" << setw(10) << "Qty" << setw(10) << "Price" << setw(10) << "Total" << setw(20) << "Ingredients\n"; // Заголовок таблицы

cout << string(70, '-') << "\n"; // Разделитель

for (size\_t i = 0; i < items.size(); ++i) { // Проходим по всем изделиям

cout << setw(20) << items[i]->getName() // Имя изделия

<< setw(10) << quantities[i] // Количество

<< setw(10) << items[i]->getPrice() // Цена за единицу

<< setw(10) << items[i]->getPrice() \* quantities[i] // Общая стоимость

<< setw(20) << items[i]->getIngredients() << "\n"; // Ингредиенты

}

cout << string(70, '-') << "\n"; // Разделитель

cout << "Total: " << calculateTotal() << " USD\n"; // Общая стоимость заказа

}

void saveToFile(const string& filename) const { // Метод для сохранения заказа в файл

string fullFilename = filename + ".txt"; // Добавляем расширение к имени файла

ofstream file(fullFilename); // Открываем файл для записи

if (!file) { // Проверяем успешность открытия

throw runtime\_error("Failed to open file for writing: " + fullFilename); // Генерируем исключение в случае ошибки

}

file << "Order:\n"; // Заголовок заказа в файле

file << string(120, '-') << "\n"; // Разделитель

file << left << setw(30) << "Item Name" // Заголовок таблицы

<< right << setw(10) << "Quantity"

<< setw(15) << "Price"

<< setw(15) << "Total"

<< setw(15) << "Calories"

<< "Ingredients\n";

file << string(120, '-') << "\n"; // Разделитель

int totalCalories = 0; // Инициализация общей калорийности

for (size\_t i = 0; i < items.size(); ++i) { // Проходим по всем изделиям

ConfectioneryItem\* item = items[i]; // Получаем текущее изделие

int quantity = quantities[i]; // Получаем количество

double itemPrice = item->getPrice(); // Получаем цену

double itemTotal = itemPrice \* quantity; // Общая стоимость текущего изделия

int itemCalories = item->getCalories() \* quantity; // Общая калорийность текущего изделия

totalCalories += itemCalories; // Добавляем калории к общей сумме

file << left

<< setw(30) << item->getName() // Имя изделия

<< right << setw(10) << quantity // Количество

<< setw(15) << fixed << setprecision(2) << itemPrice // Цена за единицу

<< setw(15) << fixed << setprecision(2) << itemTotal // Общая стоимость

<< setw(15) << itemCalories; // Калории

const auto& ingredients = item->getIngredients(); // Получаем ингредиенты

if (!ingredients.empty()) { // Если ингредиенты есть

file << ingredients[0]; // Выводим первый ингредиент

for (size\_t i = 1; i < ingredients.size(); ++i) { // Для остальных ингредиентов

file << ", " << ingredients[i]; // Выводим через запятую

}

}

file << "\n"; // Переход на новую строку

}

file << string(120, '-') << "\n"; // Разделитель

file << left << setw(70) << "Total:" // Общая стоимость

<< right << setw(15) << fixed << setprecision(2) << calculateTotal() << " USD\n";

file << left << setw(70) << "Total Calories:" // Общая калорийность

<< right << setw(15) << totalCalories << "\n";

file.close(); // Закрываем файл

cout << "Order successfully saved to file: " << fullFilename << "\n"; // Сообщаем о сохранении

}

};

int main() { // Основной метод программы

// Устанавливаем кодировку консоли в UTF-8

SetConsoleOutputCP(CP\_UTF8); // Устанавливаем кодировку выхода консоли в UTF-8

SetConsoleCP(CP\_UTF8); // Устанавливаем кодировку ввода консоли в UTF-8

char buffer[MAX\_PATH]; // Создаём массив для хранения текущей директории

GetCurrentDirectory(MAX\_PATH, buffer); // Получаем текущую рабочую директорию

cout << "Current working directory: " << buffer << endl;

ConfectioneryManager manager; // Создаём объект управления кондитерскими изделиями

Order currentOrder; // Создаём текущий заказ

try {

manager.loadFromFile("confectionery.txt"); // Загружаем данные из файла

} catch (const exception& e) {

cout << "Error loading data: " << e.what() << "\n"; // Обработка ошибки при загрузке данных

}

cout << "Current list of items:\n";

manager.displayAllItems(); // Показать все изделия

while (true) {

cout << "\n" << string(50, '=') << "\n"; // Показать меню

cout << "Menu:\n" << string(50, '-') << "\n"

<< "1. Show all items\n"

<< "2. Add item\n"

<< "3. Remove item\n"

<< "4. Show item information\n"

<< "5. Edit item\n" // Новый пункт меню

<< "6. Add to order\n"

<< "7. Show current order\n"

<< "8. Save order\n"

<< "9. Exit\n"

<< string(50, '-') << "\n"

<< "Choose an action: ";

int choice = getInput<int>(""); // Ввод выбора пользователем

cout << string(50, '=') << "\n"; // Разделитель

switch (choice) {

case 1:

manager.displayAllItems(); // Показать все изделия

break;

case 2: {

string name = getInput<string>("Enter name: "); // Ввод имени

double price = getInput<double>("Enter price: "); // Ввод цены

int calories = getInput<int>("Enter calories per 100g: "); // Ввод калорий на 100г

cout << "Enter ingredients (separated by commas): "; // Ввод ингредиентов

string ingredients;

getline(cin, ingredients);

cout << "Choose item type:\n"

<< "1. Cake\n"

<< "2. Wedding Cake\n"

<< "3. Cookie\n";

int typeChoice = getInput<int>("Enter type number: "); // Выбор типа изделия

ConfectioneryItem\* newItem = nullptr;

switch (typeChoice) {

case 1: {

double weight = getInput<double>("Enter cake weight (kg): "); // Ввод веса торта

newItem = new Cake(name, price, weight, calories); // Создание нового торта

break;

}

case 2: {

double weight = getInput<double>("Enter wedding cake weight (kg): "); // Ввод веса свадебного торта

int tiers = getInput<int>("Enter number of tiers: "); // Ввод количества ярусов

newItem = new WeddingCake(name, price, weight, tiers, calories); // Создание нового свадебного торта

break;

}

case 3: {

string shape = getInput<string>("Enter cookie shape: "); // Ввод формы печенья

newItem = new Cookie(name, price, shape, calories); // Создание нового печенья

break;

}

default:

cout << "Invalid item type choice.\n"; // Обработка неверного выбора типа изделия

break;

}

if (newItem) {

newItem->setIngredients(ingredients); // Установка ингредиентов

manager.addItem(newItem); // Добавление нового изделия

std::cout << "Item successfully added.\n"; // Успешное добавление

manager.saveToFile("confectionery.txt"); // Сохранение в файл

std::cout << "Current list of items:\n";

manager.displayAllItems(); // Показать обновлённый список изделий

}

break;

}

case 3: {

string name = // Ввод имени изделия для удаления

}

}

}

return 0;

}

int main() {

// Устанавливаем кодировку консоли в UTF-8

SetConsoleOutputCP(CP\_UTF8); // Устанавливаем кодировку выхода консоли в UTF-8

SetConsoleCP(CP\_UTF8); // Устанавливаем кодировку ввода консоли в UTF-8

char buffer[MAX\_PATH]; // Создаём массив для хранения текущей директории

GetCurrentDirectory(MAX\_PATH, buffer); // Получаем текущую рабочую директорию

cout << "Current working directory: " << buffer << endl; // Выводим текущую директорию

ConfectioneryManager manager; // Создаём объект управления кондитерскими изделиями

Order currentOrder; // Создаём текущий заказ

try {

manager.loadFromFile("confectionery.txt"); // Загружаем данные из файла

} catch (const exception& e) {

cout << "Error loading data: " << e.what() << "\n"; // Обработка ошибки при загрузке данных

}

cout << "Current list of items:\n";

manager.displayAllItems(); // Показать все изделия

while (true) {

cout << "\n" << string(50, '=') << "\n"; // Показать меню

cout << "Menu:\n" << string(50, '-') << "\n"

<< "1. Show all items\n"

<< "2. Add item\n"

<< "3. Remove item\n"

<< "4. Show item information\n"

<< "5. Edit item\n" // Новый пункт меню

<< "6. Add to order\n"

<< "7. Show current order\n"

<< "8. Save order\n"

<< "9. Exit\n"

<< string(50, '-') << "\n"

<< "Choose an action: ";

int choice = getInput<int>(""); // Ввод выбора пользователем

cout << string(50, '=') << "\n"; // Разделитель

switch (choice) {

case 1:

manager.displayAllItems(); // Показать все изделия

break;

case 2: {

string name = getInput<string>("Enter name: "); // Ввод имени

double price = getInput<double>("Enter price: "); // Ввод цены

int calories = getInput<int>("Enter calories per 100g: "); // Ввод калорий на 100г

cout << "Enter ingredients (separated by commas): "; // Ввод ингредиентов

string ingredients;

getline(cin, ingredients);

cout << "Choose item type:\n" // Выбор типа изделия

<< "1. Cake\n"

<< "2. Wedding Cake\n"

<< "3. Cookie\n";

int typeChoice = getInput<int>("Enter type number: ");

ConfectioneryItem\* newItem = nullptr;

switch (typeChoice) {

case 1: {

double weight = getInput<double>("Enter cake weight (kg): "); // Ввод веса торта

newItem = new Cake(name, price, weight, calories); // Создание нового торта

break;

}

case 2: {

double weight = getInput<double>("Enter wedding cake weight (kg): "); // Ввод веса свадебного торта

int tiers = getInput<int>("Enter number of tiers: "); // Ввод количества ярусов

newItem = new WeddingCake(name, price, weight, tiers, calories); // Создание нового свадебного торта

break;

}

case 3: {

string shape = getInput<string>("Enter cookie shape: "); // Ввод формы печенья

newItem = new Cookie(name, price, shape, calories); // Создание нового печенья

break;

}

default:

cout << "Invalid item type choice.\n"; // Обработка неверного выбора типа изделия

break;

}

if (newItem) {

newItem->setIngredients(ingredients); // Установка ингредиентов

manager.addItem(newItem); // Добавление нового изделия

std::cout << "Item successfully added.\n"; // Успешное добавление

manager.saveToFile("confectionery.txt"); // Сохранение в файл

std::cout << "Current list of items:\n";

manager.displayAllItems(); // Показать обновлённый список изделий

}

break;

}

case 3: {

string name = getInput<string>("Enter the name of the item to remove: "); // Ввод имени изделия для удаления

manager.removeItem(name); // Удаление изделия

break;

}

case 4: {

string name = getInput<string>("Enter the name of the item: "); // Ввод имени изделия

ConfectioneryItem \*item = manager.findItem(name); // Ищем изделие по имени

if (item) {

item->displayInfo(); // Отображаем информацию об изделии

cout << "Ingredients: ";

for (const auto &ingredient: item->getIngredients()) { // Отображаем ингредиенты

cout << ingredient << ", ";

}

cout << "\n";

} else {

cout << "Item not found\n"; // Обработка отсутствия изделия

}

break;

}

case 5: {

string name = getInput<string>("Enter the name of the item to edit: "); // Ввод имени изделия для редактирования

ConfectioneryItem \*item = manager.findItem(name); // Ищем изделие по имени

if (item) {

cout << "Current item details:\n";

item->displayInfo(); // Отображаем текущие данные изделия

string newName = getInput<string>("Enter new name (or 0 to keep current): "); // Ввод нового имени

if (newName != "0") item->setName(newName); // Устанавливаем новое имя, если оно не 0

double newPrice = getInput<double>("Enter new price (or 0 to keep current): "); // Ввод новой цены

if (newPrice != 0) item->setPrice(newPrice); // Устанавливаем новую цену, если она не 0

int newCalories = getInput<int>("Enter new calories per 100g (or 0 to keep current): "); // Ввод новых калорий

if (newCalories != 0) item->setCalories(newCalories); // Устанавливаем новые калории, если они не 0

std::cout << "Current ingredients: " << item->getIngredients() << "\n";

std::cout << "Enter new ingredients (separated by commas, or '0' to keep current): ";

std::string newIngredientsInput;

std::getline(std::cin, newIngredientsInput);

if (newIngredientsInput != "0") {

item->setIngredients(newIngredientsInput); // Устанавливаем новые ингредиенты, если они не равны 0

}

if (auto cake = dynamic\_cast<Cake \*>(item)) { // Проверяем, является ли изделие тортом

double newWeight = getInput<double>("Enter new weight (or 0 to keep current): "); // Ввод нового веса торта

if (newWeight != 0) cake->setWeight(newWeight); // Устанавливаем новый вес торта, если он не 0

if (auto weddingCake = dynamic\_cast<WeddingCake \*>(cake)) { // Проверяем, является ли торт свадебным

int newTiers = getInput<int>("Enter new number of tiers (or 0 to keep current): "); // Ввод нового количества ярусов

if (newTiers != 0) weddingCake->setTiers(newTiers); // Устанавливаем новое количество ярусов, если оно не 0

}

} else if (auto cookie = dynamic\_cast<Cookie \*>(item)) { // Проверяем, является ли изделие печеньем

string newShape = getInput<string>("Enter new shape (or '0' to keep current): "); // Ввод новой формы печенья

if (newShape != "0") cookie->setShape(newShape); // Устанавливаем новую форму печенья, если она не равна 0

}

cout << "Item successfully edited.\n"; // Успешное редактирование изделия

manager.saveToFile("confectionery.txt"); // Сохранение изменений в файл

} else {

cout << "Item not found\n"; // Обработка отсутствия изделия

}

break;

}

case 6: {

string name = getInput<string>("Enter the name of the item to add to the order: "); // Ввод имени изделия для добавления в заказ

ConfectioneryItem \*item = manager.findItem(name); // Ищем изделие по имени

if (item) {

int quantity = getInput<int>("Enter quantity: "); // Ввод количества

currentOrder.addItem(item, quantity); // Добавляем изделие в заказ

} else {

cout << "Item not found\n"; // Обработка отсутствия изделия

}

break;

}

case 7:

currentOrder.displayOrder(); // Отображаем текущий заказ

break;

case 8: {

string filename = getInput<string>("Enter the filename to save the order: "); // Ввод имени файла для сохранения заказа

try {

currentOrder.saveToFile(filename); // Сохраняем заказ в файл

cout << "Order saved to file " << filename << "\n"; // Успешное сохранение заказа

} catch (const exception &e) {

cout << "Error saving order: " << e.what() << "\n"; // Обработка ошибки сохранения заказа

}

break;

}

case 9:

try {

manager.saveToFile("confectionery.txt"); // Сохраняем данные в файл перед выходом

cout << "Data saved.\n"; // Успешное сохранение данных

} catch (const exception &e) {

cout << "Error saving data: " << e.what() << "\n"; // Обработка ошибки сохранения данных

}

cout << "Goodbye!\n"; // Выводим сообщение об окончании работы

return 0; // Завершаем выполнение программы

default:

cout << "Invalid choice. Please try again.\n"; // Обработка неверного выбора

break;

}

}

return 0;

}