СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире кондитерская индустрия играет значительную роль в сфере пищевого производства и розничной торговли. Кондитерские изделия, такие как торты, пирожные и печенье, являются неотъемлемой частью праздников, торжеств и повседневной жизни многих людей. С ростом спроса на разнообразные и качественные кондитерские изделия возникает необходимость в эффективном управлении ассортиментом, производством и продажами этой продукции.

В условиях развивающегося рынка и растущей конкуренции, кондитерские предприятия сталкиваются с необходимостью оптимизации своих процессов, улучшения качества обслуживания клиентов и повышения эффективности управления ресурсами. Традиционные методы ведения учета и управления ассортиментом становятся недостаточными для удовлетворения современных требований бизнеса и потребителей.

Целью данного проекта является разработка специализированной программы для управления ассортиментом кондитерских изделий. Эта программа призвана автоматизировать процессы учета, каталогизации и управления заказами различных видов кондитерской продукции, включая торты, свадебные торты и печенье.

Основными задачами разрабатываемой программы являются создание удобной системы каталогизации кондитерских изделий с учетом их специфических характеристик. Реализация функционала для эффективного управления заказами и их обработки. Обеспечение возможности быстрого поиска и редактирования информации о продукции. Внедрение системы учета ингредиентов и калорийности изделий. Разработка механизма сохранения и загрузки данных для обеспечения непрерывности работы.

Программа позволит кондитерским предприятиям оптимизировать процессы управления ассортиментом, повысить качество обслуживания клиентов и эффективность работы персонала. Она предоставит инструменты для быстрого реагирования на изменения спроса, управления запасами и планирования производства.

Таким образом, разработка программы управления ассортиментом кондитерских изделий является актуальной задачей, решение которой поможет повысить конкурентоспособность предприятий кондитерской отрасли и улучшить качество обслуживания потребителей.

1 ПОСТАНОВКА ЗАДАЧИ

Цель работы – разработать программу управления ассортиментом кондитерских изделий с использованием языка программирования C++. Программа должна предоставлять удобный и интерактивный интерфейс для добавления, удаления, редактирования и обработки информации о различных видах кондитерских изделий, а также управления заказами.

Задачи работы:

1. Анализ требований. Определить основные функциональные требования к программе управления ассортиментом кондитерских изделий, учитывая специфику отрасли и потребности пользователей.

2. Разработать структуру классов для представления различных типов кондитерских изделий. Базовый класс для всех кондитерских изделий.

Производные классы для конкретных типов изделий (торты, свадебные торты, печенье).

3. Реализовать функциональность программы. Добавление новых кондитерских изделий в ассортимент. Удаление изделий из ассортимента. Редактирование информации о существующих изделиях. Поиск и отображение информации об изделиях. Управление заказами (добавление товаров в заказ, расчет стоимости).

4. Разработать систему хранения данных. Реализовать механизм сохранения данных об ассортименте в файл. Обеспечить возможность загрузки данных из файла при запуске программы.

5. Создать пользовательский интерфейс.

Разработать консольный интерфейс с меню для взаимодействия с пользователем. Обеспечить удобный вывод информации об ассортименте и заказах.

6. Реализовать дополнительные функции. Учет ингредиентов для каждого изделия. Расчет калорийности изделий. Возможность сохранения заказов в отдельные файлы.

7. Провести тестирование программы. Проверить корректность работы всех функций программы. Протестировать обработку различных сценариев использования. Исправить выявленные ошибки и оптимизировать работу программы.

Ожидаемые результаты:

Результатом работы должна стать функциональная программа управления ассортиментом кондитерских изделий с консольным интерфейсом, обеспечивающая стабильную работу и удобное управление данными. Программа должна позволять эффективно управлять каталогом изделий, обрабатывать заказы и предоставлять необходимую информацию о продукции. Ожидается, что разработанное решение повысит эффективность управления ассортиментом и обработки заказов в кондитерских предприятиях.

2 ОБЗОР ЛИТЕРАТУРЫ

Программы управления ассортиментом кондитерских изделий представляют собой специализированные программные решения, разработанные для автоматизации и оптимизации процессов, связанных с управлением кондитерским производством и предоставлением услуг клиентам. Они помогают кондитерским предприятиям эффективно управлять своими ресурсами, включая каталоги изделий, информацию о клиентах, операции по приему и выполнению заказов, а также учет и статистику производственной деятельности.

Основные функции программ управления ассортиментом кондитерских изделий включают:

1. Каталогизация продукции: Создание и управление электронным каталогом кондитерских изделий, включая информацию о составе, калорийности, цене и других характеристиках.
2. Управление заказами: Автоматизация процессов приема, обработки и выполнения заказов клиентов.
3. Учет ингредиентов: Отслеживание наличия и расхода ингредиентов, необходимых для производства кондитерских изделий.
4. Управление клиентской базой: Хранение и обработка информации о клиентах, их предпочтениях и истории заказов.
5. Генерация отчетов: Создание различных отчетов по продажам, популярности изделий, расходу ингредиентов и другим параметрам.
6. Интеграция с другими системами: Возможность взаимодействия с системами бухгалтерского учета, управления складом и другими бизнес-приложениями.

Разрабатываемая программа управления ассортиментом кондитерских изделий на языке C++ призвана решить ряд важных задач:

– обеспечить удобный интерфейс для управления каталогом кондитерских изделий;

– облегчить процесс приема и обработки заказов;

– предоставить инструменты для анализа популярности различных видов продукции;

– облегчить процесс учета ингредиентов и расчета калорийности изделий;

– обеспечить надежное хранение данных с возможностью их быстрого поиска и обработки;

Использование такой программы позволит кондитерским предприятиям повысить эффективность работы, улучшить качество обслуживания клиентов и оптимизировать производственные процессы. Это, в свою очередь, может привести к увеличению прибыли и укреплению позиций на рынке кондитерских изделий.

2.1 Обзор методов и алгоритмов решения поставленной задачи

В данном разделе приведен обзор существующих методов и подходов, применяемых при разработке программ для управления ассортиментом кондитерских изделий. Этот обзор поможет определить наиболее эффективные и перспективные подходы для создания программного средства "Программа управления ассортиментом кондитерских изделий".

1. Архитектура системы управления ассортиментом.

Структура данных: Обзор структуры системы, включая организацию данных о кондитерских изделиях, их ингредиентах, ценах и калорийности. Рассмотрение способов хранения и связей между данными.

Иерархическая модель: Анализ применяемых иерархических моделей для представления данных о кондитерских изделиях, например, иерархия типов изделий (торты, печенье, свадебные торты), связи между изделиями и их ингредиентами.

2. Основные функциональные возможности.

Управление ассортиментом и заказами: Обзор методов управления ассортиментом кондитерских изделий и заказами, включая добавление новых изделий в каталог, редактирование информации о существующих изделиях, формирование и обработку заказов.

Работа с атрибутами изделий: Исследование возможностей работы с атрибутами кондитерских изделий, включая название, цену, вес, калорийность, состав ингредиентов и другие характеристики.

3. Интерфейс и удобство использования.

Консольный интерфейс: Рассмотрение использования меню, форматированного вывода данных и интерактивных запросов для удобного представления и манипуляции данными.

Пользовательский опыт: Исследование концепций удобства использования, включая интуитивно понятную навигацию по меню, четкое отображение информации об изделиях и заказах, а также эффективные методы ввода данных.

4. Алгоритмы и методы обработки данных.

Поиск и сортировка: Обзор алгоритмов поиска и сортировки, применимых для работы с ассортиментом кондитерских изделий, таких как бинарный поиск, быстрая сортировка или сортировка слиянием.

Управление памятью: Анализ методов эффективного управления памятью при работе с большим количеством объектов, включая использование умных указателей и правильное освобождение ресурсов.

5. Хранение и обработка данных.

Файловый ввод/вывод: Исследование методов работы с файлами для сохранения и загрузки данных об ассортименте и заказах, включая форматы хранения данных и обработку ошибок при работе с файлами.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

3.1 Структура входных данных

Загрузка данных из файла: при запуске программы она загружает данные о кондитерских изделиях из файла "confectionery.txt". Эти данные включают в себя информацию о названии, цене, весе, калорийности, типе изделия (торт, печенье и т.д.), а также дополнительные характеристики, специфичные для каждого типа изделия.

Действия пользователя: пользователь может выбирать различные действия через консольное меню, такие как просмотр всех изделий, добавление нового изделия, удаление изделия, просмотр информации о конкретном изделии, редактирование изделия, добавление изделия в заказ, просмотр текущего заказа и сохранение заказа.

Дополнительные данные: в зависимости от выбранного действия, пользователь вводит дополнительные данные, такие как название изделия, цена, вес, калорийность, тип изделия (торт, печенье, свадебный торт), специфические характеристики (например, количество ярусов для свадебного торта или форма для печенья) и ингредиенты.

3.2 Структура выходных данных

Результаты действий пользователя: программа отображает результаты выполненных действий, такие как успешное добавление или удаление изделия, информация о конкретном изделии, подтверждение изменений при редактировании.

Данные о кондитерских изделиях: при работе с ассортиментом программа отображает список всех доступных изделий, детальную информацию о каждом изделии, текущий заказ с выбранными изделиями и их количеством.

Сохранение данных в файл: при завершении работы программы или по запросу пользователя, данные об ассортименте кондитерских изделий сохраняются в файл "confectionery.txt" для последующего использования. Кроме того, информация о заказах может быть сохранена в отдельные файлы с уникальными именами.

3.3 Разработка диаграммы классов

Диаграмма классов представлена в приложении А.

3.4 Описание классов

Класс `ConfectioneryItem` - базовый класс, представляющий кондитерское изделие. Содержит основную информацию о продукте.

Публичные методы:

`ConfectioneryItem(const string& name, double price, int calories)` - конструктор класса.

`virtual ~ConfectioneryItem() = default` - виртуальный деструктор.

`virtual void displayInfo() const = 0` - чисто виртуальный метод для отображения информации о продукте.

`string getName() const` - геттер для получения имени продукта.

`double getPrice() const` - геттер для получения цены продукта.

`int getCalories() const` - геттер для получения калорийности продукта.

`void setIngredients(const string& newIngredients)` - устанавливает ингредиенты продукта.

`const string& getIngredients()` const - возвращает ингредиенты продукта.

`virtual void setName(const string& newName)` - сеттер для изменения имени продукта.

`virtual void setPrice(double newPrice)` - сеттер для изменения цены продукта.

`virtual void setCalories(int newCalories)` - сеттер для изменения калорийности продукта.

Защищенные поля:

`string name` - название продукта.

`double price` - цена продукта.

`string ingredients` - ингредиенты продукта.

`int calories` - калорийность продукта.

Класс `BakedGood` - производный класс от `ConfectioneryItem`, представляющий выпечку.

Публичные методы:

`BakedGood(const string& name, double price, double weight, int calories)` - конструктор класса.

`double getWeight() const` - геттер для получения веса выпечки.

`void setWeight(double newWeight)` - сеттер для изменения веса выпечки.

Защищенные поля:

`double weight` - вес выпечки.

Класс `Cake` - производный класс от `BakedGood`, представляющий торт.

Публичные методы:

`Cake(const string& name, double price, double weight, int calories)` - конструктор класса.

`void displayInfo() const override` - переопределенный метод для отображения информации о торте.

Класс `WeddingCake` - Производный класс от `Cake`, представляющий свадебный торт.

Публичные методы:

`WeddingCake(const string& name, double price, double weight, int tiers, int calories)` - конструктор класса.

`int getTiers() const` - геттер для получения количества ярусов торта.

`void setTiers(int newTiers)` - сеттер для изменения количества ярусов торта.

`void displayInfo() const override` - переопределенный метод для отображения информации о свадебном торте.

Приватные поля:

`int tiers` - количество ярусов свадебного торта.

Класс `Cookie` - производный класс от `ConfectioneryItem`, представляющий печенье.

Публичные методы:

`Cookie(const string& name, double price, const string& shape, int calories)` - конструктор класса.

`string getShape() const` - геттер для получения формы печенья.

`void setShape(const string& newShape)` - сеттер для изменения формы печенья.

`void displayInfo() const override` - переопределенный метод для отображения информации о печенье.

Приватные поля:

`string shape` - форма печенья.

Класс `ConfectioneryManager` - класс для управления ассортиментом кондитерских изделий.

Публичные методы:

`~ConfectioneryManager()` - деструктор класса.

`void addItem(ConfectioneryItem* item)` - добавляет новое изделие в ассортимент.

`void removeItem(const string& name)` - удаляет изделие из ассортимента по имени.

`void displayAllItems() const` - отображает информацию обо всех изделиях в ассортименте.

`ConfectioneryItem* findItem(const string& name)` - находит изделие по имени.

`void saveToFile(const string& filename) const` - сохраняет ассортимент в файл.

`void loadFromFile(const string& filename)` - загружает ассортимент из файла.

Приватные поля:

`vector<ConfectioneryItem*> items` - вектор указателей на кондитерские изделия.

Класс `Order` – класс для управления заказом.

Публичные методы:

`void addItem(ConfectioneryItem* item, int quantity)` - добавляет изделие в заказ.

`void removeItem(const string& name)` - удаляет изделие из заказа по имени.

`double calculateTotal() const` - вычисляет общую стоимость заказа.

`void displayOrder() const` - отображает информацию о текущем заказе.

`void saveToFile(const string& filename) const` - сохраняет заказ в файл.

Приватные поля:

`vector<ConfectioneryItem*> items` - вектор указателей на заказанные изделия.

`vector <int> quantities` - вектор количества каждого заказанного изделия.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 Разработка схем алгоритмов

В приложениях Б и В представлены схемы `ConfectioneryManager::addItem` и `ConfectioneryManager::displayAllItems()` соответственно.

4.2 Разработка алгоритма ConfectioneryManager::addItem()

1. Начало.
2. Входной параметр: указатель на объект ConfectioneryItem.
3. Поиск элемента с таким же именем в векторе items:
   3.1. Если элемент найден:
      3.1.2. Вывод сообщения об ошибке, что элемент с таким именем уже существует.
      3.1.3. Запрос нового имени у пользователя.
      3.1.4. Установка нового имени для элемента.
   3.2.1. Если элемент не найден:
      3.2.1. Добавление элемента в вектор items.
      3.2.2. Вывод сообщения об успешном добавлении элемента.
      3.2.3. Прерывание цикла.
4. Конец.

4.3 Разработка алгоритма ConfectioneryManager::displayAllItems()

1. Начало.
2. Вывод заголовка таблицы с информацией о кондитерских изделиях.
3. Цикл по всем элементам вектора `items`:
3.1. Вывод имени изделия.
3.2. Вывод цены изделия.
3.3. Если изделие является тортом (`Cake`):
   3.3.1. Вывод веса торта.
3.4. Иначе:
   3.4.1. Вывод прочерка вместо веса.
3.5. Определение типа изделия (`Wedding Cake`, `Cake` или `Cookie`).
3.6. Вывод типа изделия.
3.7. Вывод калорийности изделия.
3.8. Вывод дополнительной информации (количество ярусов для свадебного торта или форма для печенья).
3.9. Вывод ингредиентов изделия.
4. Конец.

5 РЕЗУЛЬТАТЫ РАБОТЫ

На рисунке 5.1 представлен вывод консоли после запуска программы

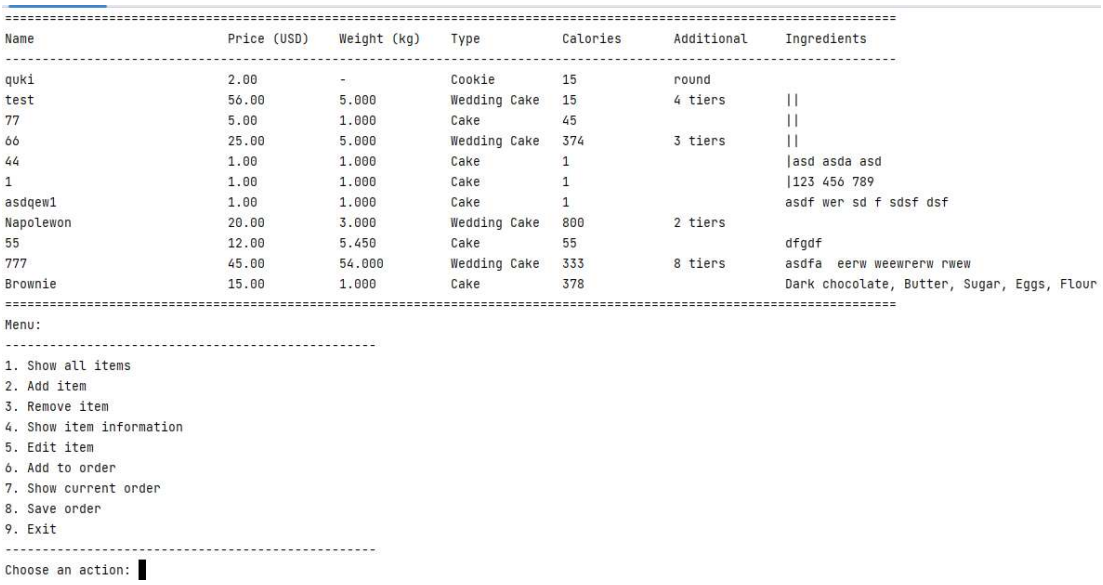

Рисунок 5.1 – Вывод консоли после запуска программы

На рисунке 5.2 представлен результат выбора показа информации об изделии Brownie

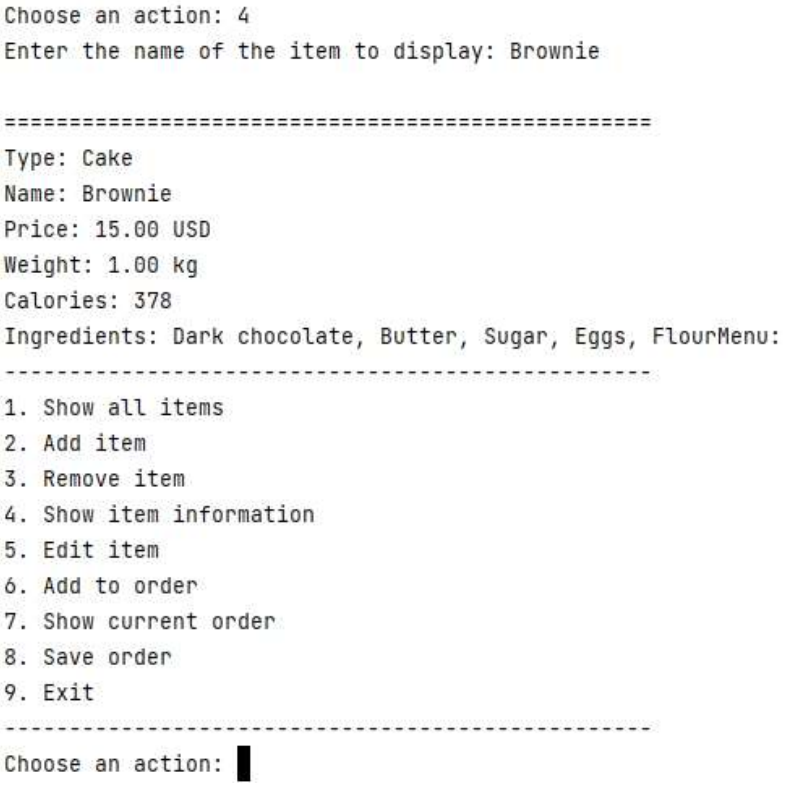

Рисунок 5.2 – Показ информации об изделии Brownie

На рисунке 5.3 представлен выведена информация о собранном заказе

```
Choose an action: 7

=================================================================================================
Item Name           Quantity  Price     Total     Calories  Ingredients
-------------------------------------------------------------------------------------------------
Brownie             4         15.00     60.00     378       Dark chocolate, Butter, Sugar, Eggs, Flour
Napolewon           2         20.00     40.00     800
-------------------------------------------------------------------------------------------------
Total:                                            100.00 USD
=================================================================================================
Menu:
--------------------------------------------------
1. Show all items
2. Add item
3. Remove item
4. Show item information
5. Edit item
6. Add to order
7. Show current order
8. Save order
9. Exit
--------------------------------------------------
Choose an action:
```

Рисунок 5.3 – Информация о собранном заказе

На рисунке 5.4 показан процесс изменения информации об изделии

```
--------------------------------------------------
Enter new price (or 0 to keep current): 46
Enter new calories per 100g (or 0 to keep current): 637
Current ingredients: asdfa  eerw weewrerw rwew
Enter new ingredients, or '0' to keep current): Vanilla, Raspberry, Buttercream, White chocolate
Enter new weight (or 0 to keep current): 6
Enter new number of tiers (or 0 to keep current): 3
Item successfully edited.
Menu:
--------------------------------------------------
1. Show all items
2. Add item
3. Remove item
4. Show item information
5. Edit item
6. Add to order
7. Show current order
8. Save order
9. Exit
--------------------------------------------------
Choose an action:
```

Рисунок 5.4 – Изменение информации об изделии

ЗАКЛЮЧЕНИЕ

В рамках курсовой работы была разработана функциональная система управления ассортиментом кондитерских изделий, предназначенная для эффективного управления и учета различных видов кондитерской продукции. Разработка данного проекта включала в себя изучение основных принципов объектно-ориентированного программирования, анализ потребностей кондитерского бизнеса и проектирование гибкой структуры классов для представления различных типов кондитерских изделий.

В ходе работы было проведено исследование специфики кондитерского производства для определения ключевых характеристик различных типов изделий и их взаимосвязей. На основе этого анализа была разработана система классов, отражающая структуру кондитерских изделий, включая базовый класс ConfectioneryItem и производные классы для конкретных типов изделий (Cake, WeddingCake, Cookie).

Для обеспечения основных операций по управлению ассортиментом был создан класс ConfectioneryManager, позволяющий добавлять, удалять, отображать и редактировать изделия. Также была реализована система заказов с использованием класса Order, который обеспечивает формирование и управление заказами клиентов, включая расчет общей стоимости и калорийности заказа.

Особое внимание было уделено разработке функций сохранения и загрузки данных, что позволило обеспечить долговременное хранение и возможность восстановления информации об ассортименте и заказах. Для удобства пользователей было разработано консольное меню, предоставляющее доступ ко всем функциям системы.

В процессе разработки было проведено комплексное тестирование системы для выявления и устранения возможных ошибок, что обеспечило стабильную работу приложения.

В результате курсовой работы удалось создать полноценную систему управления ассортиментом кондитерских изделий, способную эффективно организовывать информацию о различных типах продукции, управлять заказами и обеспечивать удобную работу с данными. Разработанная система может найти практическое применение в кондитерских предприятиях различного масштаба, способствуя оптимизации процессов управления ассортиментом и обработки заказов. системы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Документация по C++ [Электронный ресурс]. – Режим доступа: https://en.cppreference.com/w/

[2] Руководство по использованию STL (Standard Template Library) [Электронный ресурс]. – Режим доступа: https://www.geeksforgeeks.org/the-c-standard-template-library-stl/

[3] Документация по работе с файлами в C++ [Электронный ресурс]. – Режим доступа: https://www.cplusplus.com/doc/tutorial/files/

[4] Руководство по объектно-ориентированному программированию в C++ [Электронный ресурс]. – Режим доступа: https://www.learncpp.com/cpp-tutorial/welcome-to-object-oriented-programming/

[5] Документация по использованию векторов в C++ [Электронный ресурс]. – Режим доступа: https://www.cplusplus.com/reference/vector/vector/

[6] Руководство по обработке исключений в C++ [Электронный ресурс]. – Режим доступа: https://www.tutorialspoint.com/cplusplus/cpp_exceptions_handling.htm

[7] Документация по форматированному вводу-выводу в C++ [Электронный ресурс]. – Режим доступа: https://en.cppreference.com/w/cpp/io/

ПРИЛОЖЕНИЕ А
(Обязательное)
Диаграмма классов

ПРИЛОЖЕНИЕ Б
(Обязательное)
Схема алгоритма
ConfectioneryManager::addItem()

ПРИЛОЖЕНИЕ В
(Обязательное)
Схема алгоритма
ConfectioneryManager::displayAllItems()

ПРИЛОЖЕНИЕ Г
(Обязательное)
Исходный код программы

```
001 #include <iostream>
002 #include <vector>
003 #include <string>
004 #include <fstream>
005 #include <stdexcept>
006 #include <iomanip>
007 #include <algorithm>
008 #include <windows.h>
009 #include "myvector.h"
010
011 using namespace std;
012
013
014 template<typename T>
015 T getInput(const string& prompt) {
016     T value;
017     while (true) {
018         cout << prompt;
019         if (cin >> value) {
020             cin.ignore(100, '\n');
021             return value;
022         }
023         cin.clear();
024         cin.ignore(100, '\n');
025         cout << "Invalid input. Please try again.\n";
026     }
027 }
028
029 // Базовый класс
030 class ConfectioneryItem {
031 protected:
032     string name;
033     double price;
034     string ingredients;
035     int calories;
036
037 public:
038     ConfectioneryItem(const string& name, double price, int)
039             : name(name), price(price), calories(calories) {}
040
041     virtual ~ConfectioneryItem() = default;
042
043     virtual void displayInfo() const = 0;
044
045     string getName() const { return name; }
046     double getPrice() const { return price; }
047     int getCalories() const { return calories; }
048
049     void setIngredients(const string& newIngredients) {
050         ingredients = newIngredients;
051     }
052
053     const string& getIngredients() const {
054         return ingredients;
055     }
056
```

```
057     virtual void setName(const string& newName) { name = newName; }
058     virtual void setPrice(double newPrice) { price = newPrice; }
059     virtual void setCalories(int newCalories)
060 };
061
062 // Производный класс первого уровня - Выпечка
063 class BakedGood : public ConfectioneryItem {
064 protected:
065     double weight;
066
067 public:
068     BakedGood(const string& name, double price, double weight, int)
069             : ConfectioneryItem(name, price, calories) {}
070
071     double getWeight() const { return weight; }
072     void setWeight(double newWeight) { weight = newWeight; }
073 };
074
075 // Производный класс второго уровня - Торт
076 class Cake : public BakedGood {
077 public:
078     Cake(const string& name, double price, double weight, int)
079             : BakedGood(name, price, weight, calories) {}
080
081     void displayInfo() const override {
082         cout << "\n" << string(50, '=') << "\n";
083         cout << "Type: Cake\n"
084              << "Name: " << name << "\n"
085              << "Price: " << fixed << setprecision(2) << price
086              << "Weight: " << weight << " kg\n"
087              << "Calories: " << calories << "\n"
088              << "Ingredients: " << ingredients;
089     }
090
091 };
092
093 // Производный класс третьего уровня - Свадебный торт
094 class WeddingCake : public Cake {
095 private:
096     int tiers;
097
098 public:
099     WeddingCake(const string& name, double price, double weight,)
100             : Cake(name, price, weight, calories), tiers(tiers) {}
101
102     int getTiers() const { return tiers; }
103     void setTiers(int newTiers) { tiers = newTiers; }
104
105     void displayInfo() const override {
106         cout << "\n" << string(50, '=') << "\n";
107         cout << "Type: Wedding Cake\n"
108              << "Name: " << name << "\n"
109              << "Price: " << fixed << setprecision(2) << price
110              << "Weight: " << weight << " kg\n"
111              << "Tiers: " << tiers << "\n"
112              << "Calories: " << calories << "\n"
113              << "Ingredients: " << ingredients;
114     }
115 };
116
117 // Производный класс - Печенье
118 class Cookie : public ConfectioneryItem {
```

```
119 private:
120     string shape;
121
122 public:
123     Cookie(const string& name, double price, const string& shape,)
124             : ConfectioneryItem(name, price, calories), shape(shape)
125
126     string getShape() const { return shape; }
127     void setShape(const string& newShape) { shape = newShape; }
128
129     void displayInfo() const override {
130         cout << "\n" << string(50, '=') << "\n";
131         cout << "Type: Cookie\n"
132              << "Name: " << name << "\n"
133              << "Price: " << fixed << setprecision(2) << price
134              << "Shape: " << shape << "\n"
135              << "Calories: " << calories << "\n"
136              << "Ingredients: " << ingredients;
137     }
138 };
139
140 // Класс для управления ассортиментом
141 class ConfectioneryManager {
142 private:
143     vector<ConfectioneryItem*> items;
144
145 public:
146     ~ConfectioneryManager() {
147         for (auto item : items) {
148             delete item;
149         }
150     }
151
152     void addItem(ConfectioneryItem* item) {
153         while (true) {
154             // Проверяем, существует ли уже элемент с таким именем
155             auto it = find_if(items.begin(), items.end(),
156                               [&item](const ConfectioneryItem* {
157                                   return existingItem->getName()
158                               });
159
160             if (it != items.end()) {
161
162                 cout << "Error: An item with the name '";
163                 string newName = getInput<string>;
164                 item->setName(newName);
165             } else {
166
167                 items.push_back(item);
168                 cout << "Item '" << item->getName();
169                 break;
170             }
171         }
172     }
173
174     void removeItem(const string& name) {
175         auto it = find_if(items.begin(), items.end(),
176                           [&name](const ConfectioneryItem*);
177         if (it != items.end()) {
178             delete *it;
179             items.erase(it);
180         }
```

```
181     }
182
183     void displayAllItems() const {
184         cout << "\n" << string(120, '=') << "\n";
185         cout << left
186              << setw(30) << "Name"
187              << setw(15) << "Price (USD)"
188              << setw(15) << "Weight (kg)"
189              << setw(15) << "Type"
190              << setw(15) << "Calories"
191              << setw(15) << "Additional"
192              << "Ingredients\n";
193         cout << string(120, '-') << "\n";
194         for (const auto item : items) {
195             cout << left
196                  << setw(30) << item->getName()
197                  << setw(15) << fixed << setprecision(2)
198                  << setw(15);
199
200             if (auto cake = dynamic_cast<Cake*>(item)) {
201                 cout << fixed << setprecision(3);
202             } else {
203                 cout << "-";
204             }
205
206             string itemType;
207             string additional;
208
209             if (auto weddingCake = dynamic_cast<WeddingCake*>(item))
210                 itemType = "Wedding Cake";
211                 additional = to_string(weddingCake->getTiers());
212             } else if (dynamic_cast<Cake*>(item)) {
213                 itemType = "Cake";
214             } else if (auto cookie = dynamic_cast<Cookie*>(item)) {
215                 itemType = "Cookie";
216                 additional = cookie->getShape();
217             }
218
219             cout << left << setw(15) << itemType
220                  << setw(15) << item->getCalories()
221                  << setw(15) << additional;
222
223             // Выводим ингредиенты
224             cout << item->getIngredients() << "\n";
225         }
226         cout << string(120, '=') << "\n";
227     }
228
229     ConfectioneryItem* findItem(const string& name) {
230         auto it = find_if(items.begin(), items.end(),
231                                 [&name](const ConfectioneryItem*);
232         return (it != items.end()) ? *it : nullptr;
233     }
234
235     void saveToFile(const string& filename) const {
236         ofstream file(filename);
237         if (!file) {
238             throw runtime_error("Failed to open file for writing");
239         }
240
241         for (const auto item : items) {
242             file << item->getName() << "|" << item->getPrice();
```

```
243
244             if (auto cake = dynamic_cast<Cake*>(item)) {
245                 file << "|Cake|" << cake->getWeight();
246
247                     file << "|" << weddingCake->getTiers();
248                 } else {
249                     file << "|0";
250                 }
251             } else if (auto cookie = dynamic_cast<Cookie*>(item)) {
252                 file << "|Cookie|" << cookie->getShape();
253             }
254
255             // Сохраняем ингредиенты
256             file << "|" << item->getIngredients() << "\n";
257         }
258
259         file.close();
260     }
261
262     void loadFromFile(const string& filename) {
263         ifstream file(filename);
264         if (!file) {
265             cout << "File " << filename << " not found.";
266             return;
267         }
268
269         // Очищаем текущий список изделий
270         for (auto item : items) {
271             delete item;
272         }
273         items.clear();
274
275         string line;
276         while (getline(file, line)) {
277             istringstream iss(line);
278             string name, type, shape, ingredientsStr;
279             double price, weight;
280             int calories, tiers;
281
282             getline(iss, name, '|');
283             iss >> price;
284             iss.ignore();
285             iss >> calories;
286             iss.ignore();
287             getline(iss, type, '|');
288
289             ConfectioneryItem* item = nullptr;
290
291             if (type == "Cake") {
292                 iss >> weight;
293                 iss.ignore();
294                 iss >> tiers;
295                 if (tiers > 0) {
296                     item = new WeddingCake(name, price, weight);
297                 } else {
298                     item = new Cake(name, price, weight, calories);
299                 }
300             } else if (type == "Cookie") {
301                 getline(iss, shape, '|');
302                 item = new Cookie(name, price, shape, calories);
303             }
304
```

```
            if (item) {
                // Читаем ингредиенты как одну строку
                getline(iss, ingredientsStr);
                // Удаляем начальный разделитель, если он есть
                if (!ingredientsStr.empty() && ingredientsStr[0]) {
                    ingredientsStr.erase(0, 1);
                }
                item->setIngredients(ingredientsStr);
                addItem(item);
            }
        }

        file.close();
        cout << "Data successfully loaded from file " << filename;
    }
};

// Класс для управления заказами
class Order {
private:
    vector<ConfectioneryItem*> items;
    MyVector<int> quantities;

public:
    void addItem(ConfectioneryItem* item, int quantity) {
        items.push_back(item);
        quantities.push_back(quantity);
    }

    double calculateTotal() const {
        double total = 0;
        for (size_t i = 0; i < items.size(); ++i) {
            total += items[i]->getPrice() * quantities[i];
        }
        return total;
    }

    void displayOrder() const {
        cout << "\n" << string(100, '=') << "\n";
        cout << left
             << setw(20) << "Item Name"
             << setw(10) << "Quantity"
             << setw(10) << "Price"
             << setw(10) << "Total"
             << setw(10) << "Calories"
             << "Ingredients\n";
        cout << string(100, '-') << "\n";

        for (size_t i = 0; i < items.size(); ++i) {
            cout << left
                 << setw(20) << items[i]->getName()
                 << setw(10) << quantities[i]
                 << setw(10) << fixed << setprecision(2)
                 << setw(10) << fixed << setprecision(2)
                 << setw(10) << items[i]->getCalories()
                 << items[i]->getIngredients() << "\n";
        }

        cout << string(100, '-') << "\n";
        cout << left << setw(50) << "Total:"
             << fixed << setprecision(2) << calculateTotal();
        cout << string(100, '=') << "\n";
```

```
367     }
368
369
370     void saveToFile(const string& filename) const {
371         string fullFilename = filename + ".txt";
372         ofstream file(fullFilename);
373         if (!file) {
374             throw runtime_error("Failed to open file for writing");
375         }
376
377         file << "Order:\n";
378         file << string(120, '-') << "\n";
379         file << left << setw(30) << "Item Name"
380              << right << setw(10) << "Quantity"
381              << setw(12) << "Price"
382              << setw(12) << "Total"
383              << setw(12) << "Calories"
384              << "  Ingredients\n";
385         file << string(120, '-') << "\n";
386
387         int totalCalories = 0;
388         for (size_t i = 0; i < items.size(); ++i) {
389             ConfectioneryItem* item = items[i];
390             int quantity = quantities[i];
391             double itemPrice = item->getPrice();
392             double itemTotal = itemPrice * quantity;
393             int itemCalories = item->getCalories() * quantity;
394             totalCalories += itemCalories;
395
396             file << left << setw(30) << item->getName()
397                  << right << setw(10) << quantity
398                  << setw(12) << fixed << setprecision(2)
399                  << setw(12) << fixed << setprecision(2)
400                  << setw(12) << itemCalories
401                  << "  " << left << item->getIngredients() << "\n";
402         }
403
404         file << string(120, '-') << "\n";
405         file << left << setw(30) << "Total:"
406              << right << setw(34) << fixed << setprecision(2);
407         file << left << setw(30) << "Total Calories:"
408              << right << setw(34) << totalCalories << "\n";
409
410         file.close();
411         cout << "Order successfully saved to file: " << fullFilenam;
412     }
413 };
414
415 int main() {
416     // Устанавливаем кодировку консоли в UTF-8
417     SetConsoleOutputCP(CP_UTF8);
418     SetConsoleCP(CP_UTF8);
419
420     char buffer[MAX_PATH];
421     GetCurrentDirectory(MAX_PATH, buffer);
422     cout << "Current working directory: " << buffer << endl;
423
424     ConfectioneryManager manager;
425     Order currentOrder;
426
427     try {
428         manager.loadFromFile("confectionery.txt");
```

```
429     } catch (const exception& e) {
430         cout << "Error loading data: " << e.what() << "\n";
431     }
432
433     cout << "Current list of items:\n";
434     manager.displayAllItems();
435
436     while (true) {
437         //cout << "\n" << string(50, '=') << "\n";
438         cout << "Menu:\n" << string(50, '-') << "\n"
439                   << "1. Show all items\n"
440                   << "2. Add item\n"
441                   << "3. Remove item\n"
442                   << "4. Show item information\n"
443                   << "5. Edit item\n"  // New menu item
444                   << "6. Add to order\n"
445                   << "7. Show current order\n"
446                   << "8. Save order\n"
447                   << "9. Exit\n"
448                   << string(50, '-') << "\n"
449                   << "Choose an action: ";
450
451         int choice = getInput<int>("");
452
453         //cout << string(50, '=') << "\n";
454
455         switch (choice) {
456             case 1:
457                 manager.displayAllItems();
458                 break;
459             case 2: {
460                 string name = getInput<string>("Enter name: ");
461                 double price = getInput<double>("Enter price: ");
462                 int calories = getInput<int>("Enter calories per ");
463
464                 cout << "Enter ingredients: ";
465                 string ingredients;
466                 getline(cin, ingredients);
467
468                 cout << "Choose item type:\n1. Cake\n2. Wedding ";
469                 int typeChoice = getInput<int>("");
470
471                 ConfectioneryItem* newItem = nullptr;
472
473                 switch (typeChoice) {
474                     case 1: {
475                         double weight = getInput<double>("Enter");
476                         newItem = new Cake(name, price, weight);
477                         break;
478                     }
479                     case 2: {
480                         double weight = getInput<double>;
481                         int tiers = getInput<int>;
482                         newItem = new WeddingCake;
483                         break;
484                     }
485                     case 3: {
486                         string shape = getInput<string>;
487                         newItem = new Cookie(name, price, shape);
488                         break;
489                     }
490                     default:
```

```
491                         cout << "Invalid item type choice.\n";
492                         break;
493                 }
494                 if (newItem) {
495                     newItem->setIngredients(ingredients);
496                     manager.addItem(newItem);
497                     std::cout << "Item successfully added.\n";
498                     manager.saveToFile("confectionery.txt");
499                     std::cout << "Current list of items:\n";
500                     manager.displayAllItems();
501                 }
502                 break;
503             }
504             case 3: {
505                 string name = getInput<string>;
506                 manager.removeItem(name);
507                 break;
508             }
509             case 4: {
510                 string name = getInput<string>;
511                 ConfectioneryItem *item = manager.findItem(name);
512                 if (item) {
513                     item->displayInfo();
514                 } else {
515                     cout << "Item not found\n";
516                 }
517                 break;
518             }
519             case 5: {
520                 string name = getInput<string>;
521                 ConfectioneryItem *item = manager.findItem(name);
522                 if (item) {
523                     cout << "Current item details:\n";
524                     item->displayInfo();
525 
526                     string newName = getInput<string>;
527                     if (newName != "0") item->setName(newName);
528 
529                     double newPrice = getInput<double>;
530                     if (newPrice != 0) item->setPrice(newPrice);
531 
532                     int newCalories = getInput<int>;
533                     if (newCalories != 0);
534 
535                     std::cout << "Current ingredients: ";
536                     std::cout << "Enter new ingredients ";
537                     std::string newIngredientsInput;
538                     std::getline(std::cin, newIngredientsInput);
539                     if (newIngredientsInput != "0") {
540                         item->setIngredients(newIngredientsInput);
541                     }
542 
543                     if (auto cake = dynamic_cast<Cake *>(item)) {
544                         double newWeight = getInput<double>;
545                         if (newWeight != 0);
546 
547 
548                             int newTiers = getInput<int>;
549                             if (newTiers != 0);
550                         }
551 
552                         string newShape = getInput<string>;
```

```
553                         if (newShape != "0");
554                     }
555
556                     cout << "Item successfully edited.\n";
557                     manager.saveToFile("confectionery.txt");
558                 } else {
559                     cout << "Item not found\n";
560                 }
561                 break;
562             }
563             case 6: {
564                 string name = getInput<string>;
565                 ConfectioneryItem *item = manager.findItem(name);
566                 if (item) {
567                     int quantity = getInput<int>;
568                     currentOrder.addItem(item, quantity);
569                 } else {
570                     cout << "Item not found\n";
571                 }
572                 break;
573             }
574             case 7:
575                 currentOrder.displayOrder();
576                 break;
577             case 8: {
578                 string filename = getInput<string>;
579                 try {
580                     currentOrder.saveToFile(filename);
581                     cout << "Order saved to file " << filename;
582                 } catch (const exception &e) {
583                     cout << "Error saving order: " << e.what();
584                 }
585                 break;
586             }
587             case 9:
588                 try {
589                     manager.saveToFile("confectionery.txt");
590                     cout << "Data saved.\n";
591                 } catch (const exception &e) {
592                     cout << "Error saving data: " << e.what();
593                 }
594
595                 return 0;
596             default:
597                 cout << "Invalid choice. Please try again.\n";
598                 break;
599         }
600     }
601     return 0;
602 }
```

ПРИЛОЖЕНИЕ Д
(Обязательное)
Ведомость документов