**1. dynamic\_cast**

Позволяет безопасно преобразовывать указатели или ссылки между типами в иерархии наследования. Работает только с классами, где есть виртуальные функции. Если преобразование невозможно, результат будет либо nullptr, либо ошибка.

**2. Указатели**

Это переменные, которые хранят адреса других переменных. Используются для управления памятью, передачи данных по ссылке и динамического создания объектов. Основные операции: разыменование (доступ к значению по адресу) и работа с адресами.

**3. Виртуальные функции**

Методы в базовом классе, которые можно переопределить в наследниках. Позволяют вызывать версии метода, соответствующие реальному типу объекта, даже если работа идёт через указатель или ссылку на базовый класс.

**4. Итераторы**

Объекты, которые позволяют перемещаться по элементам контейнера (например, массивов, списков, векторов). Упрощают работу с коллекциями, делая её похожей на работу с указателями.

### ****5.**** auto

Ключевое слово, которое позволяет компилятору автоматически определить тип переменной на основе значения, которое ей присваивается. Удобно при работе с длинными или сложными типами.

**6. Чтение\запись**

Запись:  
1. Открывает файл для записи с помощью ofstream.

2. Проходит по всем элементам в items.

3. Записывает информацию о каждом элементе в файл, разделяя поля символом |.

4. Записывает ингредиенты, разделяя их символом ;.

5. Закрывает файл после записи всех данных.

Чтение:

Открывает файл для чтения с помощью ifstream.

2. Очищает текущий список изделий.

3. Читает файл построчно.

4. Разбирает каждую строку, извлекая информацию о кондитерском изделии.

5. Создает соответствующий объект (Cake, WeddingCake или Cookie) на основе прочитанных данных.

6. Добавляет ингредиенты к созданному объекту.

7. Добавляет созданный объект в список изделий.

8. Закрывает файл после чтения всех данных.

### ****3.**** size\_t

Беззнаковый целочисленный тип, используемый для представления размеров объектов и индексов массивов.

* Размер зависит от архитектуры (обычно 32 или 64 бита).
* Гарантирует, что может хранить максимальный размер любого объекта.

### ****4.**** setw

Манипулятор для потоков ввода-вывода, задающий минимальную ширину поля для вывода следующего элемента.

1. Устанавливает минимальную ширину поля для следующего элемента.

2. Если выводимый элемент короче заданной ширины, он дополняется пробелами.

3. Если элемент длиннее, он выводится полностью без усечения.

4. По умолчанию выравнивание происходит вправо, но можно изменить с помощью std::left.

В коде setw(75) устанавливает ширину поля в 75 символов для вывода слова "Total:".

### ****5.**** istringstream

Класс, позволяющий работать со строкой как с потоком ввода, упрощая её разбор.

1. Инициализируется строкой, которая обычно читается из файла.
2. Используется для разбора данных, разделённых символами (|, ;, пробелами и т. д.).
3. Методы (getline, >>, ignore) помогают извлекать данные по частям.

Пример:  
Создание istringstream:

istringstream iss(line);

iss создается из строки line, которая представляет одну строку, прочитанную из файла.

Использование iss:

getline(iss, name, '|');

iss >> price;

iss.ignore();

iss >> calories;

iss.ignore();

getline(iss, type, '|');

getline(iss, name, '|') читает название изделия до разделителя '|'.

iss >> price извлекает цену.

iss.ignore() пропускает разделитель.

iss >> calories читает количество калорий.

Следующий getline(iss, type, '|') читает тип изделия.

| разделяет в файле значения, но не выводится.

### ****6.**** Использовать как свой контейнер, так и STL контейнеры.

Где и зачем используется собственный контейнер MyVector?  
Сделали файл myvector.h там прописали контейнер свой и используется он в классе Order для хранения количества товаров в заказе.

То есть хранится оно там:  
Пример: заказ из 2 тортов, 5 печений и 1 торта

vector<ConfectioneryItem\*> items; // Хранит указатели на товары

MyVector<int> quantities; // Хранит количество каждого товара

items = [указатель\_на\_Наполеон, указатель\_на\_Овсяное, указатель\_на\_Королевский]

quantities = [2, 5, 1]

И вот так рассчитывается Total в заказе  
double calculateTotal() const {

double total = 0;

for (size\_t i = 0; i < items.size(); ++i) {

total += items[i]->getPrice() \* quantities[i];

}

return total;

}

И для вывода:  
void displayOrder() const {

for (size\_t i = 0; i < items.size(); ++i) {

cout << items[i]->getName() << " x " << quantities[i] << "\n";

}

}

**7. Слабые места**

При формировании заказа, заказ не удаляется, нужно перезаходить в программу.

Русские символы не получилось нормально записывать в файл, поэтому весь интерфейс на английском, и русские символы в прогу не нужно вводить) Можно ответить, чтобы проблемы с кодировкой винды, что пробовал и 1251, и utf-8 – не помогло

В файле заказа таблица нормально не выводится, кривовато, даже если брать то как она ровная в консоль выводится, 2 часа убил.

При вводе веса торта, нужно вводить через точку, если через запятую, то дробная часть отбросится

**8. В записке описание двух функций, их желательно лучше остальных понимать**  
  
**Разработка алгоритма ConfectioneryManager::addItem()**

**1. Начало.**

**2. Входной параметр: указатель на объект ConfectioneryItem.**

Код: void addItem(ConfectioneryItem\* item) {

3. Поиск элемента с таким же именем в векторе items:

Код: while (true) { auto it = find\_if(items.begin(), items.end(), [&item](const ConfectioneryItem\* existingItem) { return existingItem->getName() == item->getName(); });

3.1. Если элемент найден:

Код: if (it != items.end()) {

3.1.1. Вывод сообщения об ошибке, что элемент с таким именем уже существует.

Код: cout << "Error: An item with the name '" << item->getName() << "' already exists.\n";

3.1.2. Запрос нового имени у пользователя.

Код: string newName = getInput<string>("Please enter a different name: ");

3.1.3. Установка нового имени для элемента.

Код: item->setName(newName);

3.2. Если элемент не найден:

Код: } else {

3.2.1. Добавление элемента в вектор items.

Код: items.push\_back(item);

3.2.2. Вывод сообщения об успешном добавлении элемента.

Код: cout << "Item '" << item->getName() << "' has been added successfully.\n";

3.2.3. Прерывание цикла.

Код: break;

**4.Конец.**

**Разработка алгоритма void ConfectioneryManager::displayAllItems() const**

**1. Начало.**

**2. Вывод заголовка таблицы с информацией о кондитерских изделиях.**

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

cout << left

<< setw(30) << "Name"

<< setw(15) << "Price (USD)"

<< setw(15) << "Weight (kg)"

<< setw(15) << "Type"

<< setw(15) << "Calories"

<< setw(15) << "Additional"

<< "Ingredients\n";

**3. Вывод разделительной линии.**

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

**4. Цикл по всем элементам вектора items:**

for (const auto item : items) {

4.1. Вывод имени изделия.

4.2. Вывод цены изделия.

cout << left

<< setw(30) << item->getName()

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

<< setw(15);

4.3. Если изделие является тортом (Cake):

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

4.3.1. Вывод веса торта.

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

}

4.4. Иначе:

else {

4.4.1. Вывод прочерка вместо веса.

cout << "-";

}

4.5. Определение типа изделия (Wedding Cake, Cake или Cookie).

string itemType;

string additional;

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

itemType = "Wedding Cake";

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

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

itemType = "Cake";

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

itemType = "Cookie";

additional = cookie->getShape();

}

4.6. Вывод типа изделия.

4.7. Вывод калорийности изделия.

4.8. Вывод дополнительной информации (количество ярусов для свадебного торта или форма для печенья).

cout << left << setw(15) << itemType

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

<< setw(15) << additional;

4.9. Вывод ингредиентов изделия.

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

}

**5. Вывод разделительной линии.**

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

**6. Конец.**

}

**9. Стрелка с ромбом на диаграмме классов**

Агрегация — это тип отношения между классами, при котором один класс содержит ссылки на экземпляры другого, но они существуют независимо друг от друга. Например, ConfectioneryManager управляет объектами ConfectioneryItem, но их жизненные циклы не связаны: удаление менеджера не удаляет сами элементы. В отличие от агрегации, наследование обозначает "является" (is-a) отношение, при котором один класс наследует свойства и поведение другого, формируя иерархию классов, где подкласс зависит от базового.