FROMA2

Библиотека программных компонентов для финансово-экономического моделирования частичных затрат предприятия

ОПИСАНИЕ И РУКОВОДСТВО ПО ПРИМЕНЕНИЮ

Версия 1.0

Александр Пидкасистый

2025

ОГЛАВЛЕНИЕ

РАСЧЕТ ЧАСТИЧНЫХ ЗАТРАТ И ВОЗМОЖНОСТИ FROMA2

РАСЧЕТ ЧАСТИЧНЫХ ЗАТРАТ И ЕГО МЕСТО В УПРАВЛЕНИИ ПРЕДПРИЯТИЕМ

Предпринимательские решения, принятые на основе полученной из расчета совокупных затрат информации, могут быть ошибочными. Основная причина этого – распределение постоянных затрат с помощью коэффициентов и наценок по местам возникновения и по объектам затрат, что не всегда оправдано. В связи с этим достоверный анализ и контроль затрат возможен лишь условно.

Исходя из расчета совокупных затрат большая часть постоянных затрат через решения, действующие в течении длительного времени, ложится на производственные мощности предприятия. В краткосрочном аспекте эти постоянные затраты неизменяемы, поэтому для краткосрочных решений значимы только переменные затраты. Таким образом, расчет совокупных затрат дает искажённую информацию.

Чтобы избежать этого недостатка расчета совокупных затрат, используются системы расчета частичных затрат (далее по тексту - РЧЗ). В этих системах, в противоположность расчету совокупных затрат, за отчетный период только определенная часть совокупных затрат распределяется по местам возникновения и по объектам затрат.

Библиотека FROMA2 предназначена для РЧЗ на основе переменных затрат. В этой системе все затраты за расчетный период разделяются на *постоянные* и переменные. При этом только переменные затраты распределяются по местам возникновения и объектам затрат.

Разработанный в США американским экономистом Д. Харрисом в 1936 году РЧЗ на основе переменных затрат называется Директ-костинг (Direct Costing). Центральным понятием для РЧЗ на основе переменных затрат является понятие «покрытие издержек».

Покрытие издержек определяется как разница между ценой на рынке и переменными затратами. Рассчитываться могут следующие виды покрытия издержек, связанные друг с другом иерархически:

- Покрытие затрат на единицу продукции (штучное покрытие издержек);
- Покрытие затрат за период для одного вида продукции;
- Покрытие затрат за период для одной категории/группы продуктов;
- Покрытие затрат за период для всего предприятия.

Исходя из штучного покрытия издержек, рассчитываются все последующие виды покрытия издержек, как сумма предыдущих ступеней.

Результат хозяйственной деятельности предприятия за период определяется как разница между покрытием издержек всего предприятия и его постоянными затратами. Каждый продукт с положительным покрытием издержек участвует тем самым, как минимум частично, в покрытии постоянных затрат предприятия.

Продукт, который по совокупному расчету затрат на рынке не покрыл себестоимости, тем не менее дает положительный экономический результат, пока он участвует в покрытии издержек. Поэтому оперирование категориями покрытия затрат и является базой РЧЗ на основе переменных затрат.

РЧЗ на основе переменных затрат способен принести существенную пользу при принятии решений руководством предприятия, которую невозможно получить от расчета совокупных затрат. Разложение затрат на постоянные и переменные соответствует разделению решений предприятия на краткосрочные и долгосрочные.

Краткосрочными могут считаться те области решений, которые не оказывают влияния на изменение производственных мощностей предприятия, т.е. на оснащенность машинами, персоналом и т.д. Средне- и долгосрочные решения, напротив, могут влиять на изменение мощностей предприятия и тем самым на

постоянные затраты. РЧЗ на основе переменных затрат наиболее подходит поэтому для области краткосрочных решений.

ИСПОЛЬЗОВАНИЕ В ЦЕНОВОЙ ПОЛИТИКЕ ПРЕДПРИЯТИЯ

РЧЗ на основе переменных затрат определяет в качестве краткосрочной нижней границы цены переменные затраты на единицу продукции;

ИСПОЛЬЗОВАНИЕ В АССОРТИМЕНТНОЙ ПОЛИТИКЕ ПРЕДПРИЯТИЯ

РЧЗ на основе переменных затрат позволяет провести ранжирование продуктов по величине покрытия затрат:

- для удаления из ассортимента продуктов с отрицательной величиной покрытия;
- для определения очередности и объемов загрузки дефицитных мощностей при ограничениях производственных мощностей (конкуренция между продуктами за машину, на которой они производятся, «узкое место») путем ранжирования продуктов по величине покрытия затрат с учетом востребованных рынком объемов этих продуктов и времени их обработки на этих дефицитных мощностях;

ИСПОЛЬЗОВАНИЕ В ПОЛИТИКЕ СНАБЖЕНИЯ ПРЕДПРИЯТИЯ

При необходимости принятия ращения о сокращении глубины собственного производства (концепция бережливого производства, подразумевающая отказ от производства отдельных компонентов в пользу закупки их на стороне), с помощью РЧЗ на основе переменных затрат могут быть определены верхние границы закупочных цен;

ИСПОЛЬЗОВАНИЕ РЧЗ НА ОСНОВЕ ПЕРЕМЕННЫХ ЗАТРАТ ДЛЯ АНАЛИЗА РЕНТАБЕЛЬНОСТИ.

РЧЗ на основе переменных затрат позволяет оценить уровень сбыта продукции, при котором достигается компенсация всех затрат и образуется прибыль (точка безубыточности или break-even point).

ВОЗМОЖНОСТИ FROMA2

Библиотека FROMA2[1] – это библиотека программного обеспечения для экономического моделирования, финансового анализа и планирования операционной деятельности предприятия. Библиотека FROMA2 написана на языке C++ и представляет собой набор программных модулей, предназначенных для использования программистами при создании дружественных потребителю программ для РЧЗ на основе переменных затрат. Библиотека FROMA2 является статической библиотекой.

Библиотека FROMA2 позволяет моделировать формирование частичных затрат и производить их расчет для:

- Склада ресурсов (склада сырья и материалов, склада готовой продукции, склада промежуточных изделий);
- Производства (места обработки ресурсов и превращения их в окончательные или промежуточные изделия в соответствии с рецептурами готовой продукции и/или технологическими картами).

В качестве ресурсов, затраты которых могут считаться поштучно, могут выступать:

- Сырье, материалы, комплектующие;
- Продукты, используемые в производстве других продуктов;
- Удельные (штучные) затраты энергоносителей (электричество, газ, вода, тепло);

[1] FROMA2 – аббревиатура, сокращение от "Free Operation Manager 2".

- Удельные человеческие ресурсы (сдельный труд);
- Удельные амортизационные отчисления.

Библиотека FROMA2 может быть использована как для планирования, так и для учета фактических переменных затрат.

ПРИМЕР 1

Есть предприятие, состоящее из трех подразделений: *Склада сырья и материалов*, *Производства* и *Склада готовой продукции*. Для этого предприятия типична картина:

> В разные периоды времени предприятием закупаются разные по объему и закупочным ценам партии сырья и материалов. Часть сырья и материалов формирует переходящий запас на Складе сырья и материалов, другая часть - направляется на Производство продукции. Готовая продукция направляется на другой склад – Склад готовой продукции. Часть продукции формирует переходящий запас на Складе готовой продукции, другая часть - отгружается покупателям.
>
> Руководству предприятия для принятия управленческих решений требуются данные об учетной себестоимости сырья и материалов, поступающих в производство, о материальной себестоимости произведенной продукции и о материальной себестоимости продаваемой продукции.

Библиотека FROMA2 позволяет сотрудникам IT-подразделения этого предприятия или его подрядчикам собрать финансово-экономическую модель из трех указанных выше подразделений, установить принцип учета запасов: ФИФО (англ. аббревиатура FIFO, - "первым вошел - первым вышел"), ЛИФО (англ. аббревиатура LIFO, - "последним вошел - первым вышел"[2]), или ПО СРЕДНЕЙ СЕБЕСТОИМОСТИ (англ. аббревиатура AVERAGE, - "по-среднему") и рассчитать плановые или фактические показатели:

- объем производства продукции, необходимый для отгрузки потребителям и поддержания требуемого запаса на Складе готовой продукции;
- объем закупок сырья и материалов, необходимый для производства продукции и поддержания требуемого запаса на Складе ресурсов;
- учетную себестоимость сырья и материалов, направляемых в производство;
- учетную себестоимость остатков сырья и материалов на Складе ресурсов;
- материальную себестоимость продукции, направляемой на склад готовой продукции;
- материальную себестоимость готовой продукции, отгружаемой покупателям;
- материальную себестоимость остатков продукции на Складе готовой продукции.

Сборка модели предполагает написание программы на языке C++, в которой каждое указанное выше подразделение соответствует определенному специализированному классу из библиотеки, а структура данных, передаваемых между пользователем и/или между экземплярами этих классов, соответствует стандартам информации программного интерфейса библиотеки.

О том, как сконструировать модель для данного примера можно прочитать в разделе «СОЗДАНИЕ МОДЕЛИ С ОДНИМ ЦЕНТРОМ ЗАТРАТ».

ПРИМЕР 2

Руководитель предприятия из Примера 1 поставил своему IT-подразделению новую задачу: на еженедельной основе предоставлять данные о производственной себестоимости производимой и продаваемой продукции, включающей в себя:

[2] В России метод LIFO не используется с 1 января 2008 года согласно Приказу Минфина России от 26.03.2007 N 26н «О внесении изменений в нормативные правовые акты по бухгалтерскому учету».

- материальную себестоимость продукции (см. предыдущий пример);
- прямые затраты на оплату труда рабочих, осуществляющих производство продукции, с начислениями на неё;
- затраты на электричество, учитываемое при изготовлении каждой единицы продукции.

Решение поставленной задачи с помощью библиотеки FROMA2 может быть достигнута несколькими путями.

Первый путь. Если руководству предприятия необходимы данные о производственной себестоимости продукции без разбивки на материальную себестоимость, прямые затраты труда и прямые затраты на электроэнергию, то модель из трех объектов, сформированная в прошлом примере, вполне подойдет для этой задачи. Изменяется структура данных: в нее добавляются данные о трудозатратах на единицу продукции и стоимости этого труда, о потреблении электроэнергии на каждый вид продукта и ее стоимости. В результате рассчитываем плановые или фактические показатели:

- объем производства продукции, необходимый для отгрузки потребителям и поддержания требуемого запаса на Складе готовой продукции;
- объем закупок сырья и материалов, необходимый для производства продукции и поддержания требуемого запаса на Складе ресурсов;
- трудозатраты, необходимые для производства продукции;
- электропотребление, необходимое для производства продукции;
- учетную себестоимость сырья и материалов, направляемых в производство;
- учетную себестоимость остатков сырья и материалов на Складе ресурсов;
- производственную себестоимость продукции, направляемой на склад готовой продукции;
- производственную себестоимость готовой продукции, отгружаемой покупателям;
- производственную себестоимость остатков продукции на Складе готовой продукции.

Второй путь. Руководству предприятия необходимы данные о производственной себестоимости продукции наряду с данными о материальной себестоимости продукции, т.к. для управленческих решений ему требуются оба типа данных. В этом случае к трем объектам из предыдущего примера возможно добавить четвертый так, чтобы производственное подразделение предприятия описывалось не одним, а двумя экземплярами специализированного класса из библиотеки: один объект отвечает за расчет материальной себестоимости продукции, другой – за затраты, включающие в себя трудозатраты и затраты на электричество. К существующей структуре данных добавляется новая структура данных, включающая в себя данные о трудозатратах на единицу продукции и стоимости этого труда и о потреблении электроэнергии на каждый вид продукта и ее стоимости. В результате рассчитываем плановые или фактические показатели:

- объем производства продукции, необходимый для отгрузки потребителям и поддержания требуемого запаса на Складе готовой продукции;
- объем закупок сырья и материалов, необходимый для производства продукции и поддержания требуемого запаса на Складе ресурсов;
- учетную себестоимость сырья и материалов, направляемых в производство;
- учетную себестоимость остатков сырья и материалов на Складе ресурсов;
- материальную себестоимость продукции, направляемой на склад готовой продукции;
- материальную себестоимость готовой продукции, отгружаемой покупателям;
- материальную себестоимость остатков продукции на Складе готовой продукции;
- трудозатраты, необходимые для производства продукции;
- электропотребление, необходимое для производства продукции;
- полные и удельные затраты на труд и электроэнергию;
- производственную себестоимость продукции, направляемой на склад готовой продукции;
- производственную себестоимость готовой продукции, отгружаемой покупателям;
- производственную себестоимость остатков продукции на Складе готовой продукции.

Методические рекомендации из раздела «СОЗДАНИЕ МОДЕЛИ С ОДНИМ ЦЕНТРОМ ЗАТРАТ» также подходят для решения этой задачи.

ПРИМЕР 3

На предприятии из *Примеров 1* и *2* принято решение *обособить* Склад сырья и материалов, Производство и Склад готовой продукции, выделив их в отдельные центры затрат. Это связано, например с тем, что часть сырья и материалов предприятие продает другим компаниям, и руководитель нашего предприятия хочет, чтобы затраты на труд сотрудников этого склада, электроэнергию и другие затраты покрывались отпускной ценой продаваемых ресурсов.

На складе готовой продукции, например, тоже новшества, - ввели дополнительную операцию: групповую упаковку и маркировку заказываемых одним заказчиком товаров (комплектация продукции по заказчикам). Это потребовало дополнительный расход упаковочного материала и трудозатраты на упаковывание, и руководитель нашего предприятия также хочет, чтобы затраты, возникающие на СГП, можно было бы контролировать и удерживать на плановом уровне.

Таким образом возникла новая задача: на еженедельной основе получать информацию об

- удельных и полных затратах склада сырья и материалов,
- удельных и полных затратах производственного подразделения и
- удельных и полных затратах склада готовой продукции.

Как создать модель для решения подобной задачи описано в разделе «СОЗДАНИЕ МОДЕЛИ С НЕСКОЛЬКИМИ ЦЕНТРАМИ ЗАТРАТ».

ЛИЦЕНЗИЯ НА БИБЛИОТЕКУ FROMA2

Библиотека FROMA2 распространяется под лицензией, использующей в качестве шаблона стандартную лицензию MIT (*X11 License*) в соответствии со Статьей 1286.1 Гражданского кодекса Российской Федерации (часть четвертая) от 18.12.2006 N 230-ФЗ (в ред. от 22.07.2024 и любой более поздней редакции).

СОСТАВ И СТРУКТУРА БИБЛИОТЕКИ FROMA2

Библиотека FROMA2 написана на языке программирования C++ и включает в себя несколько модулей. Ключевыми модулями библиотеки являются модуль *warehouse* и модуль *manufact*, реализующих наборы инструментов для моделирования центра учета запасов и центра учета затрат соответственно.

МОДУЛЬ WAREHOUSE

Модуль *warehouse* представлен заголовочным файлом *warehouse_module.h* и файлом реализации *warehouse_lib.cpp*. В модуле содержаться классы и методы, реализующие алгоритмы учета запасов на *складе ресурсов*.

Ядром модуля является **класс clsSKU**, описывающий склад ресурсов для одной номенклатурной позиции (одного SKU, Stock Keeping Unit - единицы складского учета) и включающий в себя основные алгоритмы учета. Экземпляр этого класса содержит информацию о количестве периодов проекта, наименовании номенклатурной позиции, единице измерения объемных показателей; информацию о закупках, отгрузках, остатков по периодам в натуральном, ценовом и денежном выражении. Класс содержит методы учета запасов FIFO, LIFO или «по-среднему», метод автоматического расчета закупок с учетом плана отгрузок и плана остатков в каждом периоде и метод расчета балансовых остатков товаров в каждом периоде проекта. В классе также присутствуют сервисные методы, позволяющие корректно изменять длительность проекта и другие параметры, а также имеются методы сериализации/ десериализации данных (запись состояния экземпляра класса в файл и восстановления этого состояния из файла) и методы визуального контроля.

Основным классом модуля *warehouse*, с которым чаще всего придется иметь дело программистам, использующим библиотеку FROMA2, является **класс clsStorage**, Этот класс описывает склад ресурсов для множества номенклатурных позиций (множества *SKU*). Экземпляр класса представляет собой контейнер и интерфейс для нескольких объектов типа *clsSKU* и позволяет моделировать движение по складу множества

номенклатурных единиц. Экземпляр этого класса содержит информацию о количестве периодов проекта, домашней валюте проекта (валюте основных расчетов), применяемом принципе учета запасов (FIFO, LIFO или «по-среднему»), количестве номенклатурных позиций, учитываемых на складе, контейнер для множества экземпляров класса clsSKU (однотоварных складов) и другие служебные параметры. Класс содержит методы сериализации и десериализации, расчетные методы и методы визуального контроля.

МОДУЛЬ MANUFACT

Модуль *manufact* представлен заголовочным файлом *manufact_module.h* и файлом реализации *manufact_lib.cpp*. В модуле содержаться классы и методы, реализующие алгоритмы для производства[3].

Ядром модуля является **класс clsRecipeItem**, описывающий рецептуру и/или технологическую карту отдельного продукта/услуги и содержащий информацию о длительности производственного цикла и расходе ресурсов (сырья, материалов, компонентов, энергоносителей, сдельного труда, штучной амортизации и проч.) на производство единицы данного продукта/услуги в натуральном выражении в каждый период производственного цикла. Класс также содержит расчетные методы, позволяющие скалькулировать объем потребления ресурсов в натуральном выражении для всего плана выпуска данного продукта на протяжении всего проекта, рассчитать объем, удельную и полную себестоимость[4] готового продукта и незавершенного производства в каждый период проекта. Класс содержит методы сериализации и десериализации и методы визуального контроля.

Класс clsManufactItem описывает производство отдельного продукта/услуги на протяжении всего проекта и содержит информацию о потребности в ресурсах для выпуска продукта/услуги в заданных объемах по заданному графику, учетные цены на ресурсы, объемы, полную и удельную себестоимость изготовленного продукта/услуги, балансовую полную и удельную стоимость незаконченного производства. Класс *clsManufactItem* является контейнером и интерфейсом для класса *clsRecipeItem*. Класс также содержит методы сериализации и десериализации и методы визуального контроля.

Основным классом модуля *manufact*, с которым чаще всего придется иметь дело программистам, использующим библиотеку FROMA2, является **класс clsManufactory**. Этот класс описывает полноценное производство *ассортимента* продуктов/услуг. Экземпляр класса представляет собой контейнер и интерфейс для нескольких объектов типа *clsManufactItem* и позволяет моделировать производство *множества* номенклатурных единиц продукции/услуг. Экземпляр этого класса содержит информацию о количестве периодов проекта, полной номенклатуре и потребности в ресурсах, используемых в производстве. Методы класса позволяют добавлять учетные цены на все ресурсы, получать объединенную информацию об объемах, удельной и полной себестоимости готовой продукции и незавершенного производства каждого продукта. Класс также содержит методы сериализации и десериализации и методы визуального контроля.

МОДУЛЬ COMMON_VALUES

Модуль *common_values* представлен единственным файлом *common_values.hpp*, в котором объединены заголовки классов и методы и их реализация.

[3] «Производство» следует понимать в широком смысле слова: это и производство товарной продукции, и производство услуг. В частности, услуг хранения на складе ресурсов и других центрах учета затрат (см. раздел «МОДЕЛИРОВАНИЕ С ПОМОЩЬЮ КЛАССОВ clsStorage и clsManufactory»).

[4] Под себестоимостью в этом разделе будем понимать себестоимость используемых ресурсов: удельная себестоимость – себестоимость ресурсов в единице продукции, полная себестоимость – себестоимость ресурсов в партии продукта. Это может быть материальная себестоимость, если ресурсами являются сырье и материалы, производственная себестоимость, если в дополнении к сырью и материалам используются сдельный труд, затраты на энергоресурсы и штучная амортизация или только часть производственной себестоимости, если в качестве ресурсов использовать только труд, энергоресурсы и проч., без сырья и материалов.

В модуле определен стандарт данных для информационных потоков внутри экземпляров классов библиотеки, между экземплярами классов и пользователем (ввод, хранение, обработка и вывод информации).

Данные о натуральных, ценовых и стоимостных величинах, а также долях и процентах этих величин представлены в виде вещественных чисел. Для вещественных чисел используется либо встроенный в C++ тип *double*, либо собственный вещественный тип библиотеки *LongReal*. В модуле можно выбрать тип представления вещественных чисел. Для типа LongReal можно также настроить количество десятичных знаков для вещественного числа и диапазон значений его экспоненты. Скорость вычислений при использовании типа *LongReal* ниже, а объем потребления вычислительных ресурсов больше, чем при использовании типа *double*, но в некоторых случаях использование типа *LongReal* позволяет избежать *ошибок потери точности* при расчетах.

В модуле *common_values* представлены:

- наиболее употребимые константы и типы, используемые во всех классах;
- классы и функции визуального контроля работоспособности методов из модулей *warehouse* и *manufact* (для удобства «отлова ошибок» при модернизации последних);
- классы и методы для вывода исходных данных и результатов расчетов на терминал и/или в текстовый файл.

МОДУЛЬ LONGREAL

Модуль *LongReal* представлен заголовочным файлом *LongReal_module.h* и файлом реализации *LongReal_lib.cpp*. В модуле содержатся классы и методы, реализующие вещественные числа и основные арифметические и логические операции над ними. Тип LongReal является альтернативой встроенному в C++ типу double и предназначен для проведения расчетов с высокой точностью.

Если точность вычислений с типом *double* программисту покажется неудовлетворительной[5] и ему будет мало 16 десятичных разрядов (например, в ситуации, когда себестоимость партий продуктов измеряется миллиардами рублей, а расход сырья на единицу продукции измеряется величинами с пятью знаками после запятой и более, могут возникать ошибки потери точности, особенно заметные на больших объемах данных и длительных сроках проекта), он может использовать тип *LongReal*. В типе *LongReal* по умолчанию установлено 64 десятичных разряда для мантиссы числа, а его экспонента лежит в диапазоне значений от минус 32768 до плюс 32768. При желании программиста и наличии больших вычислительных ресурсов можно изменить требуемое число десятичных разрядов и диапазон значений экспоненты: число десятичных разрядов ограничивается значением (numeric_limits<size_t>::max()/2-1), а значения экспоненты могут лежать в пределах от (numeric_limits<int>::min()/2+1) до (numeric_limits<int>::max()/2-1), где max() и min() – функции из стандартного для C++ модуля limits.

Вещественные числа типа *LongReal* могут быть инициализированы и им могут быть присвоены вещественные числа, представленные в виде:

- чисел типа *LongReal*,
- чисел типа *double* (в фиксированном и научном форматах),
- чисел, представленных строкой.

[5] Тип *double* представляет вещественное число двойной точности с плавающей точкой в диапазоне +/- 1.7E-308 до 1.7E+308. В памяти занимает 8 байт (64 бита). В числах *double*: 1 знаковый бит, 11 бит для экспоненты и 52 бит для мантиссы, то есть в сумме 64 бита. 52-разрядная мантисса позволяет определить точность до 16 десятичных знаков (https://metanit.com/cpp/tutorial/2.3.php).

Вещественные числа типа *LongReal* могут быть возвращены для присваивания и для вывода в виде:

- числа, представленного строкой цифр, целая и дробная часть которого разделена запятой,
- числа типа *long double*,
- числа типа *double*,
- числа типа *float*.

Вещественное число типа *LongReal* может быть выведено также в виде строки вида «*.ddddEn*» (мантисса *dddd* со степенью *n*).

Основные арифметические и логические операции над вещественными числами типа *LongReal*:

- Проверка числа на ноль,
- Проверка на положительную бесконечность,
- Проверка на отрицательную бесконечность,
- Проверка на неопределенность (nan),
- Оператор сравнения равно,
- Оператор сравнения не равно,
- Оператор умножения,
- Унарный минус,
- Оператор больше,
- Оператор меньше,
- Оператор больше или равно,
- Оператор меньше или равно,
- Оператор сложения,
- Оператор вычитания,
- Оператор декремента на заданную величину,
- Оператор инкремента на заданную величину,
- Вычисление обратного числа,
- Оператор деления,
- Оператор взятия модуля числа.

Методы сериализации и десериализации модуля позволяют сохранять в файл и читать из файла:

- Единичные вещественные числа типа *LongReal*;
- Массивы вещественных чисел типа *LongReal*.

Помимо арифметических, логических операций и операций сериализации и десериализации, в классе предусмотрены методы ввода и вывода вещественного числа и методы визуального контроля внутреннего представления вещественного числа.

МОДУЛЬ IMPEX

Модуль *Impex* представлен заголовочным файлом *Impex_module.h* и файлом реализации *Impex_lib.cpp*. В модуле содержатся классы и методы, реализующие обмен данными между объектами, относящимися к библиотеке FROMA2 и программами сторонних разработчиков с помощью CSV-файлов.

Основой модуля *Impex* является класс **clsImpex**, представляющий собой буфер обмена информацией и его методы, позволяющие:

- Прочитать информацию из *CSV-файла* в буфер;

- Вернуть всё или часть содержимого буфера в стандартном формате представления данных библиотеки FROMA2, пригодном для использования в финансово-экономической модели, сконструированной из объектов библиотеки FROMA2;
- Прочитать информацию из финансово-экономической модели, сконструированной из объектов библиотеки FROMA2 в буфер;
- Транспонировать буфер (по правилам транспонирования матрицы);
- Выгрузить всю или часть информации из буфера в *CSV-файл*.

Модуль содержит и другие вспомогательные методы.

> Данный модуль, в частности, использовался для оценки корректности расчетов финансово-экономических моделей, построенных с помощью библиотеки *FROMA2*. Для сравнения результатов расчётов строились одинаковые модели: одна с помощью библиотеки *FROMA2*, другая с помощью программы Project Expert 7, хорошо зарекомендовавшей себя на российском рынке. Исходные данные из модели, построенной в *Project Expert 7*, экспортировались в *CSV-файлы*, а затем импортировались в модель, построенную с помощью *FROMA2*. Результаты расчётов в обоих моделях были экспортированы в другие *CSV-файлы*, содержимое которых сравнивалось между собой. Сравнения показали идентичность результатов расчётов и корректность моделей, построенных с помощью библиотеки *FROMA2*.

МОДУЛЬ SERIALIZATION

Модуль *serialization* представлен единственным файлом *serialization_module.hpp*, в котором объединены заголовки функций и их реализация. В нем представлены шаблоны и нешаблонные перегруженные функции для сериализации и десериализации данных. Сериализация и десериализация предусмотрена для данных ограниченного набора типов, достаточных для функционирования библиотеки FROMA2:

- Строки символов типа *string* стандартного модуля string;
- Единичного объекта тривиально копируемого типа;
- *Массива* объектов тривиально копируемого типа.

В совокупности с методами сериализации/ десериализации вещественных чисел типа LongReal, представленных в одноименном модуле, эти функции полностью покрывают потребности любой финансово-экономической модели, сконструированной из объектов библиотеки FROMA2 в сохранении своего состояния в файл с заданным именем и последующего восстановления своего состояния из этого файла.

МОДУЛЬ BASEPROJECT

Модуль *baseproject* представлен заголовочным файлом *baseproject_module.h* и файлом реализации *baseproject_lib.cpp*. В модуле описывается класс **clsBaseProject**, единственное назначение которого – представлять удобство родительского класса (интерфейса) для пользовательского класса финансово-экономической модели, сконструированной из объектов библиотеки FROMA2. Предполагается, что любая финансово-экономическая модель, конструируемая из объектов библиотеки FROMA2, может быть представлена единственным классом, являющимся потомком *clsBaseProject*, а в качестве полей этого нового класса будут выступать экземпляры классов *clsStorage*, *clsManufactory* и других классов библиотеки *FROMA2*. Это не является обязательным условием моделирования, но может быть полезным для программистов.

Основными удобствами использования класса clsBaseProject в качестве «родителя» для классов-потомков является:

- Средства управления названием и описанием проекта;
- Средства управления выбора устройства и вывода отчета (пустое устройство, терминал или файл);

- Типовые методы класса, ответственные за инкапсуляцию, наследование и полиморфизм (конструктор по умолчанию и виртуальный деструктор, конструктор копирования и перемещения, функция обмена значениями между объектами, операторы присваивания копированием и перемещением, метод сброса состояния объекта и др.);
- Средства сериализации и десериализации.

Модуль *baseproject* находится в *стадии разработки*. По замыслу автора библиотеки FROMA2, в класс *clsBaseProject* в будущем должны войти общие расчётные методы, объединяющие передачу данных между полями класса типа *clsStorage*, *clsManufactory* и корректный последовательный вызов расчётных методов этих объектов.

МОДУЛЬ PROGRESS

Отображение прогресса или хода выполнения расчетов призвано поддержать комфортное состояние пользователя при длительной загрузке процессора вычислительными действиями. В модулях WAREHOUSE и MANUFACT присутствуют разные вычислительные методы, осуществляющие свою работу как в однопоточном, так и в многопоточном режимах. Объекты библиотеки FROMA2 могут использоваться как в консольных, так и оконных приложениях. Инструменты данного модуля *progress* предоставляют возможность обеспечивать визуализацию проводимых вычислений во всех таких случаях.

Модуль *progress* представлен единственным файлом *progress_module.hpp*, в котором объединены заголовки классов и методы и их реализация. Классы и методы модуля предназначены для выведения на экран индикатора прогресса (хода вычислений) при выполнении длительных расчётов.

ТИПЫ ДАННЫХ ДЛЯ ИНФОРМАЦИОННЫХ ПОТОКОВ

Количественные данные, такие как объемные (килограммы, штуки, партии и т.п.), стоимостные (рубли и т.п.) и ценовые величины (рублей за штуку и т.п.), представляются во всех модулях библиотеки FROMA2 вещественными числами.

Описательные данные, такие как название ресурса (например, «изделие номер 5»), единица измерения его натурального объема (например, «шт.»), представляются во всех модулях объектами типа std::string стандартного модуля C++ <string>.

СТРУКТУРА STRITEM

Количественные данные, относящиеся к какому-либо ресурсу, обладающие сразу тремя количественными характеристиками (объемной, стоимостной и ценовой) представлены структурой **strItem**. Эта структура предназначена для хранения информации об объеме, цене и стоимости единственной партии ресурсов (например, сырья или готовой продукции), поступающей, находящейся или убывающей со склада ресурсов и/или производства.

Таблица 1 Поля структуры strItem

Тип поля структуры	Описание
volume	Тип decimal - алиас, псевдоним для вещественного типа. В зависимости от выбора программиста принимает значение double или LongReal. Поле задаёт размер партии сырья и материалов в натуральном измерении (например, в кг)
price	Тип decimal. Поле задаёт цену ресурса за единицу (например, руб./кг)
value	Тип decimal. Поле задаёт стоимость партии для конкретного ресурса (например, руб)

Таблица 2 Функции структуры strItem

Функции структуры	Описание
StF	Метод сериализации состояния объекта в поток типа *ofstream*
RfF	Метод десериализации состояния объекта из потока типа *ifstream*

Все поля и функции структуры являются публичными.

bool strItem::StF(ofstream &_outF);

Метод записывает состояние объекта типа *strItem* в поток. Параметры: поток с именем _outF типа *ofstream*. Возвращает значение *true*, если операция прошла успешно, в противном случае возвращает *false*. Использует метод SEF из модуля SERIALIZATION для каждого из полей (volume, price и value).

bool strItem::RfF(ifstream &_inF);

Метод восстанавливает состояние объекта типа *strItem* из потока. Параметры: поток с именем _inF типа *ifstream*. Возвращает значение *true*, если операция прошла успешно, в противном случае возвращает *false*. Использует метод DSF из модуля SERIALIZATION для каждого из полей (volume, price и value).

Таблица 3 Перегрузки функций, не являющихся членами

Перегрузки функций, не являющихся членами	Описание
operator<<	Перегрузка оператора вывода информации в поток ostream.

friend ostream& operator<<(ostream &stream, strItem st);

Перегруженный оператор вывода информации из объекта типа *strItem* в поток.

Параметры:

Поток с именем stream типа *ostream*, экземпляр структуры типа *strItem* с именем st. Возвращает поток stream с записанными туда значениями volume, price и value, разделенных строкой символов ", ". После значения value идет комбинация символов перевода строки "\n". Использует стандартный оператор вывода в поток “<<”. Перегруженный оператор используется в методах визуального контроля модулей WAREHOUSE и MANUFACT.

СТРУКТУРА STRNAMEMEAS

Описательные данные, относящиеся к какому-либо ресурсу, имеющему свой объект типа *strItem*, представлены структурой **strNameMeas**. Эта структура предназначена для хранения информации о названии ресурса (например, «Изделие номер 5») и названия единицы его натурального измерения в партии (например, «шт.»).

Таблица 4 Поля структуры strNameMeas

Тип поля структуры	Описание
name	Тип std::string. Наименование номенклатурной позиции
measure	Тип std::string. Наименование единицы измерения номенклатурной позиции

Структура strNameMeas не имеет собственных методов.

ОБЛАСТЬ ИМЕН NMBPTYPES

Оба стандартных для библиотеки FROMA2 типа *strItem* и *strNameMeas* включены в область имен nmBPTypes модуля *common_values*. Помимо них в эту область имен включены ряд наиболее употребимых констант, типов и функций.

Таблица 5 Другие наиболее важные константы, типы и методы области имен nmBPTypes

Субъекты области имен nmBPTypes	Описание
const decimal epsln	«Бесконечно малая величина», являющаяся «условным нулем». Применяется для корректного сравнения вещественных чисел друг с другом и нулем.
ReportData	Тип данных перечисления *enum*; состоит из конечного набора именованных констант типа size_t. Назначение: выбор нужного поля при получении данных из объекта типа *strItem*: «volume» или 0 – поле volume, «price» или 1 – поле price и «value» или 2 – поле value объекта типа *strItem*
Currency	Тип данных перечисления *enum*; состоит из конечного набора именованных констант типа size_t. Каждая константа представляет собой индекс валюты, в которой представлены исходные данные и производятся расчеты денежных величин: «RUR» или 0, «USD» или 1, «EUR» или 2, «CNY» или 3.
CurrencyTXT	Массив строковых констант с наименованиями валюты в виде текста.

Для решения проблемы сравнения вещественных чисел друг с другом и/или с нулем, применяется константа **epsln**. Эта константа представляет собой «бесконечно малую величину», являющуюся «условным нулем» для корректного сравнения вещественных чисел друг с другом и нулем. По умолчанию она равна 10^{-7}. Это значение можно изменить.

Значение константы в типе *Currency* должно соответствовать индексу элемента с наименованием валюты в массиве *CurrencyTXT*. (Например, значение константы EUR типа *Currency* равно 2, и значение элемента массива CurrencyTXT [2] с индексом 2 должно быть равным «EUR».). При добавлении в тип *Currency* новых констант и в массив *CurrencyTXT* новых названий валют, необходимо следить, чтобы наименование добавляемой константы и название валюты корреспондировали по смыслу друг с другом, а также следить за тем, чтобы значение добавляемой константы равнялось индексу добавляемого элемента массива.

ТИП ВЕЩЕСТВЕННЫХ ЧИСЕЛ

Вещественные числа в библиотеке FROMA2 по желанию программиста могут быть представлены либо стандартным для C++ типом *double*, либо собственным типом библиотеки *LongReal*, реализация которого представлена в модуле LONGREAL. За установку типа вещественного числа отвечает строка кода в модуле COMMON_VALUES:

Таблица 6 Выбор типа вещественного числа

Альтернативнная строка кода	Результат
typedef double decimal;	Вещественные числа представлены типом *double*; decimal - псевдоним для вещественного типа
typedef LongReal decimal;	Вещественные числа представлены типом *LongReal*; decimal - псевдоним для вещественного типа

Вместо *double* может быть установлен иной тип вещественного числа, например, *long double* или *float*. Однако такая подстановка не рекомендуется: она не гарантирует корректность расчетов и может привести к непредсказуемому результату.

ТИП ИНДИКАТОРА ПРОГРЕССА

Для визуализации процесса вычислений может быть использован индикатор прогресса практически любого стороннего класса. Основное требование к такому классу – наличие в своем составе метода *Update(int)*. Этому требованию соответствует как входящий в библиотеку *FROMA2* класс индикатора прогресса для консольных приложений clsprogress_bar, так и класс wxProgressDialog библиотеки wxWidgets для приложений с графическим интерфейсом. При этом возможные кандидаты на роль индикатора прогресса не ограничиваются перечисленными выше классами. За установку типа индикатора отвечает строка кода в модуле COMMON_VALUES:

Альтернативнная строка кода	Результат
typedef clsprogress_bar type_progress	Выбран индикатор класса *clsprogress_bar*; *type_progress* – псевдоним для типа класса индикатора
typedef wxProgressDialog type_progress	Выбран индикатор класса *wxProgressDialog* из библиотеки *wxWidgets*; *type_progress* – псевдоним для типа класса индикатора

Вместо clsprogress_bar может быть выбран иной класс индикатора прогресса, имеющий в своем составе метода *Update(int)*.

СКЛАД РЕСУРСОВ

Основные типы, константы, классы и методы, реализующие алгоритмы для склада ресурсов, находятся в модуле WAREHOUSE. Состав модуля приведен в таблице ниже.

Таблица 7 Состав модуля WAREHOUSE

Состав модуля	Описание	Заметка
enum AccountingMethod	Тип перечисление *enum*, состоящий из значений FIFO, LIFO, AVG	Предназначен для выбора принципа учета запасов: FIFO, LIFO или По-среднему
const string AccountTXT	Массив строковых констант тип *string* с наименованиями принципов учета запасов в виде строки символов	Предназначен для вывода информации о выбранном типе учета в виде строки символов
enum ChoiseData	Тип перечисление *enum*, состоящий из значений *purchase*, *balance*, *shipment*	Предназначен для выбора данных: поступления ресурсов на склад, баланс остатков на складе и отгрузки сос клада
enum PurchaseCalc	Тип перечисление *enum*, состоящий из значений *calc* и *nocalc*	Флаг разрешающий/ запрещающий автоматический расчет закупок под требуемые объемы отгрузок. В случае запрещающего флага *nocalc*, объем закупок вводится вручную
struct TLack	Тип структуры для передачи информации о величине дефицита и наименования ресурса, где возник дефицит. Поля для данных: • decimal *lack* (величина дефицита), • string *Name* (наименование ресурса, где выявлен дефицит)	В случае ручного ввода объема поступления ресурсов на склад возможна ситуация, когда для требуемых отгрузок и остатков поступающих на склад ресурсов недостаточно; в этом случае переменная TLack возвращает величину дефицита и название резуса, где образовался этот дефицит
class clsSKU	Класс Склада для одной номенклатурной позиции (моносклад)	Экземпляр класса позволяет моделировать учетные процессы при движении через склад одной единственной номенклатурной единицы; содержит методы учета запасов

class clsStorage	Класс Склада для множества номенклатурных позиций	Экземпляр класса представляет собой контейнер для нескольких объектов типа *clsSKU* и позволяет моделировать учетные процессы при движении через склад множества номенклатурных единиц

Помимо указанных в таблице членов, в заголовочном файле warehouse_module.h содержится ряд строковых констант, используемых при выводе информации на экран/файл.

КЛАСС clsSKU

Класс clsSKU предназначен для *моделирования учётных процессов* на складе комплектующих, сырья и материалов, а также учётных процессов на складе готовой продукции[6]. Экземпляр класса clsSKU описывает склад ресурсов для *одной* номенклатурной позиции (*одного* SKU, Stock Keeping Unit - единицы складского учета).

Основные *задачи*, которые можно решить применением класса clsSKU:

1. Рассчитать объем и график закупок/ поступлений ресурсов на склад, обеспечивающих заданный объем и график отгрузки ресурсов со склада и выполнение норматива неснижаемых остатков на складе.
2. Рассчитать учётную себестоимость ресурсов, отгружаемых со склада в каждый период отгрузки в соответствии с известными ценами ресурсов на входе склада в каждый период закупки/ поступления и выбранным принципом учета запасов.

ПОЛЯ КЛАССА clsSKU

Таблица 8 Поля класса clsSKU. Секция private

Поле класса	Описание	Заметка
size_t PrCount	Тип size_t. Количество периодов проекта. Нулевой период – это период перед началом проекта	Значение устанавливается при создании объекта класса. Может корректироваться.
string name	Тип string. Название номенклатурной единицы складского учета	Значение устанавливается при создании объекта класса. Может корректироваться.
string measure	Тип *string*. Наименование единицы измерения объемов ресурсов (кг, штук и т.п.)	Значение устанавливается при создании объекта класса. Может корректироваться
strItem *Pur	(Pur – от слова Purchase.) Тип указатель на массив из партий ресурсов, поступивших на склад. Элемент массива имеет тип *strItem*	Динамический массив. При создании экземпляра класса создается и заполняется нулями
strItem *Rem	(Rem – от слова Remaining.) Тип указатель на вспомогательный массив остатков от партий ресурсов на складе на конец проекта (партии отличаются друг от друга датами поступления на склад). Элемент массива имеет тип *strItem*	Динамический массив. При создании экземпляра класса создается и заполняется нулями. Носит вспомогательный характер
strItem *Bal	(Bal – от слова Balance) Тип указатель на массив остатков ресурсов на складе в каждый период проекта (независимо от того, в какие периоды были получены ресурсы). Элемент массива имеет тип *strItem*	Динамический массив. При создании экземпляра класса создается и заполняется нулями
strItem *Ship	(Ship – от слова Shipment) Тип указатель на массив партий ресурсов, отгружаемых со склада по плану (план отгрузок)	Динамический массив. При создании экземпляра класса создается и заполняется нулями

[6] И сырье, и материалы, и комплектующие, и полуфабрикаты, и готовую продукцию будем называть одним словом «ресурсы».

Оглавление >>>

decimal lack	Тип определяется псевдонимом decimal. Дефицит ресурсов для отгрузки со склада; если lack>0, то отгрузка по плану невозможна	Расчётная величина. При создании экземпляра класса получает значение ноль.
bool indr	Тип bool. Переменная, хранящая установленное пользователем разрешение использовать закупаемые в каком-либо i-м периоде ресурсы для отгрузки в этом же периоде: *true* - можно, *false* - нельзя	Значение устанавливается при создании объекта класса. Может корректироваться. Для случаев, когда приход на склад осуществляется в конце периода, а отгрузка – в начале следующего, - флаг устанавливается в значение *false*. По умолчанию устанавливается *false*.
AccountingMethod acct	Переменная типа *AccountingMethod*, хранящая установленный пользователем принцип учета запасов: *FIFO*, *LIFO* или *AVG*	Значение устанавливается при создании объекта класса. Корректировке без уничтожения экземпляра класса не подлежит
PurchaseCalc pcalc	Переменная типа *PurchaseCalc*, хранящая установленный пользователем флаг разрешения/ запрещения (*calc*/ *nocalc*) автоматического расчета закупок под требуемые объемы отгрузок.	Значение устанавливается при создании объекта класса. Может корректироваться. По умолчанию устанавливается автоматический расчет pcalc= *calc*.
decimal share	Запас ресурсов на складе в каждый период, выраженный в доле от объема отгрузок за этот период	Значение устанавливается при создании объекта класса. Может корректироваться

Методы класса *clsSKU* можно условно разделить на следующие группы:

- Конструкторы, деструктор, перегруженные операторы;
- Алгоритмы учета (методы в секции Private и методы в секции Public);
- Get-методы;
- Set-методы;
- Методы сохранения состояния объекта в файл и восстановления из файла;
- Методы визуального контроля

Ниже подробно описываются все методы.

КОНСТРУКТОРЫ, ДЕСТРУКТОР, ПЕРЕГРУЖЕННЫЕ ОПЕРАТОРЫ

clsSKU()

Конструктор по умолчанию (без параметров). Создает и инициализирует переменные: PrCount = sZero; name = measure = EmpStr; lack = dZero; indr = false; acct = FIFO; pcalc = calc; share = dZero. Динамические массивы не создаются, указатели на них устанавливаются в *nullptr*.

clsSKU(const size_t _PrCount, const string &_name, const string &_measure, AccountingMethod _ac, bool _ind, PurchaseCalc _flag, const decimal& _share, const strItem _Ship[])

Конструктор с вводом объемов отгрузок. Конструктор инициализирует переменные, создает и заполняет нулями динамические массивы Pur, Rem, Bal и Ship.

Параметры:

_PrCount - количество периодов проекта; устанавливает значение поля класса PrCount;

&_name – ссылка на название номенклатурной единицы складского учета; устанавливает значение поля класса name;

&_measure – ссылка на название единицы измерения номенклатурной единицы складского учета; устанавливает значение поля класса measure;

_ac – принцип учета запасов: FIFO, LIFO или AVG; устанавливает значение поля класса acct;

_ind – флаг разрешения/ запрещения использовать закупаемые в каком-либо i-м периоде ресурсы для отгрузки в этом же периоде; устанавливает значение поля класса indr;

_flag - флаг разрешения/ запрещения автоматического расчета закупок под требуемые объемы отгрузок; устанавливает значение поля класса pcalc;

&_share – ссылка на запас ресурсов на складе в каждый период, выраженный в доле от объема отгрузок за этот период; устанавливает значение поля класса share;

_Ship[] – указатель на массив с объемом отгрузок (в каждом элементе массива заполнены только поля volume, поля price и value заполнены нулями); устанавливает значение поля класса Ship.

clsSKU(const size_t _PrCount, const string &_name, const string &_measure, AccountingMethod _ac, bool _ind, PurchaseCalc _flag, const decimal& _share)

Конструктор с параметрами. Аналогичен предыдущему конструктору, но без ввода объемов отгрузок в виде массива strItem _Ship[].

clsSKU(const clsSKU &obj)

Конструктор копирования.

Параметры:

&obj – ссылка на копируемый константный экземпляр класса clsSKU.

clsSKU(clsSKU &&obj)

Конструктор перемещения.

Параметры:

&&obj - перемещаемый экземпляр класса clsSKU.

clsSKU& operator=(const clsSKU &obj)

Перегрузка оператора присваивания копированием.

Параметры:

&obj – ссылка на копируемый константный экземпляр класса clsSKU.

clsSKU& operator=(clsSKU &&obj)

Перегрузка оператора присваивания перемещением

Параметры:

&&obj - перемещаемый экземпляр класса clsSKU.

~clsSKU()

Деструктор.

void swap(clsSKU &obj) noexcept

Функция обмена значениями между экземплярами класса.

Параметры:

&obj – ссылка на обмениваемый экземпляр класса clsSKU.

bool operator == (const string &Rightname) const

Перегрузка оператора сравнения «равно» для поиска экземпляра класса по имени.

Параметры:

&Rightname – ссылка на строку символов, с которой сравнивается поле name экземпляра класса.

АЛГОРИТМЫ УЧЕТА. СЕКЦИЯ PRIVATE

decimal AVGcalc(const strItem P[], strItem R[], strItem S[], size_t N, bool ip)

Функция рассчитывает цены и стоимость ресурсов, отгружаемых со склада в каждом периоде проекта по принципу AVG (*по среднему*) формирует вспомогательный массив остатков от партий ресурсов на складе на конец проекта (Rem) с учетом плановых отгрузок и возвращает дефицит сырья для отгрузок, если таковой имеет место быть. Тип возвращаемого значения определяется псевдонимом decimal.

Функция не оптимизирует план отгрузок, если возникает дефицит сырья и материалов; в периоды, где возникает дефицит, расчет цен и стоимости отгружаемых партий НЕ КОРРЕКТЕН!

Блок схема алгоритма приведена на рисунке ниже (Рисунок 1).

Параметры:

P[] - указатель на массив закупок/поступлений на склад (массив изменять запрещено, const); тип элемента массива strItem.

R[] - указатель на вспомогательный массив, - после всех проведенных расчетов показывает остатки от каждой партии на конец проекта; тип элемента массива strItem.

S[] – указатель на массив отгрузок, у которого заполнены только поля *volume* (массив изменяется в процессе работы функции: рассчитываются поля *price* и *value*); тип элемента массива strItem.

N - размерность массивов закупок, остатков и отгрузок, совпадает с количеством периодов проекта; стандартный тип *size_t*.

ip - индикатор, признак того, можно ли использовать закупаемые в каком-либо периоде ресурсы для отгрузки в этом же периоде: "*true*" означает, что закупка осуществляется в начале каждого периода и закупаемые ресурсы могут быть использованы в отгрузках того же периода, "*false*" означает, что закупки производятся в конце каждого периода, поэтому отгрузка этих ресурсов возможна только в следующих периодах; стандартный тип *bool*.

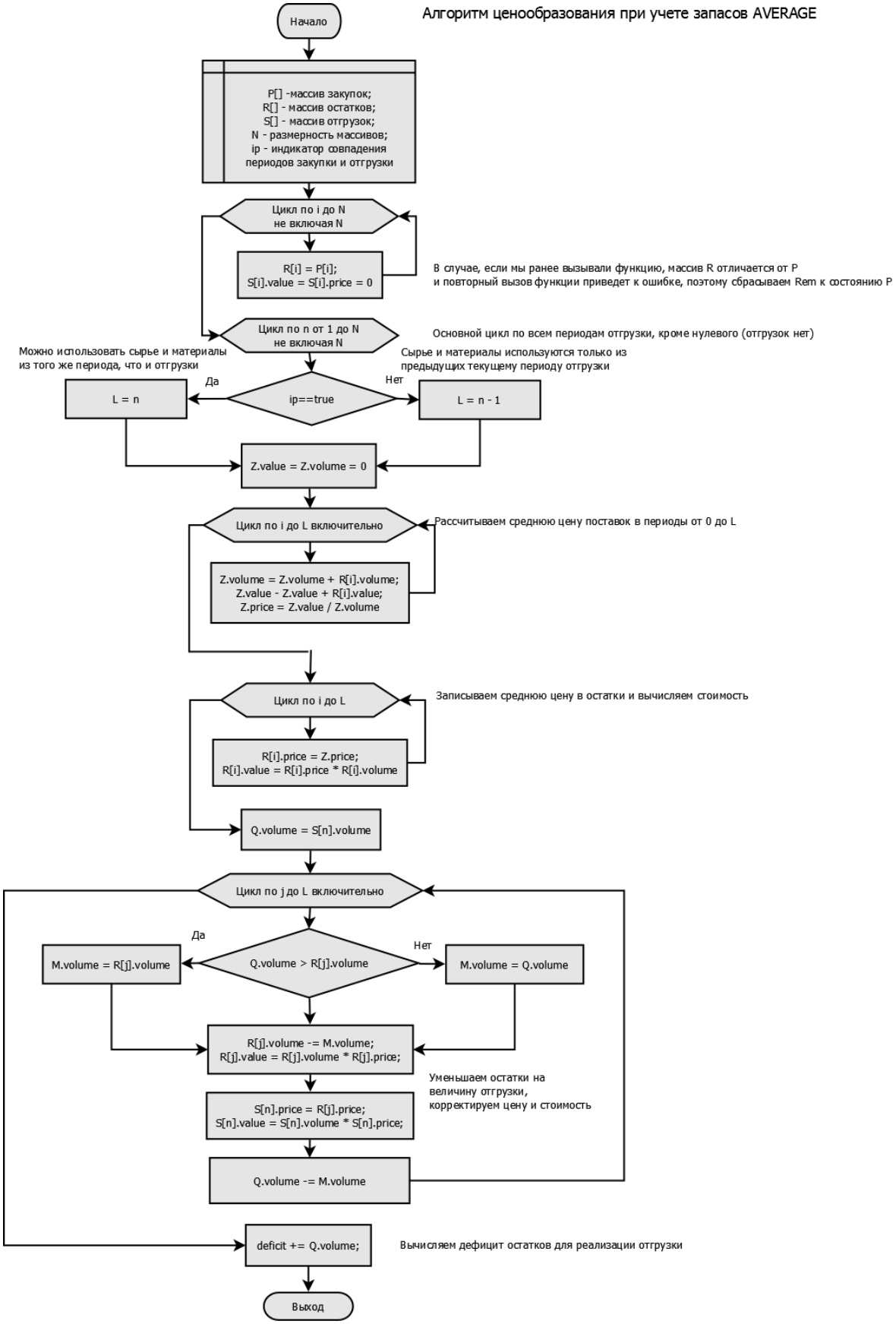

Рисунок 1 Блок-схема алгоритма учета запасов по принципу среднего (AVG)

decimal FLFcalc(const strItem P[], strItem R[], strItem S[], size_t N, bool ip, AccountingMethod ind)

Функция рассчитывает цены и стоимость сырья и материалов, отгружаемых со склада в каждом периоде проекта. по принципу FIFO или LIFO в зависимости от индикатора метода учета ind, формирует вспомогательный массив остатков от партий ресурсов на складе на конец проекта (Rem) с учетом плановых отгрузок и возвращает дефицит сырья для отгрузок, если таковой имеет место быть. Тип возвращаемого значения определяется псевдонимом decimal.

Функция не оптимизирует план отгрузок, если возникает дефицит сырья и материалов; в периоды, где возникает дефицит, расчет цен и стоимости отгружаемых партий НЕ КОРРЕКТЕН!

Блок схема алгоритма приведена на рисунке ниже (Рисунок 2).

Параметры:

P[] - указатель на массив закупок/поступлений на склад (массив изменять запрещено, const); тип элемента массива strItem.

R[] - указатель на вспомогательный массив, - после всех проведенных расчетов показывает остатки от каждой партии на конец проекта; тип элемента массива strItem.

S[] – указатель на массив отгрузок, у которого заполнены только поля *volume* (массив изменяется в процессе работы функции: рассчитываются поля *price* и *value*); тип элемента массива strItem.

N - размерность массивов закупок, остатков и отгрузок, совпадает с количеством периодов проекта; стандартный тип *size_t*.

ip - индикатор, признак того, можно ли использовать закупаемые в каком-либо периоде ресурсы для отгрузки в этом же периоде: "*true*" означает, что закупка осуществляется в начале каждого периода и закупаемые ресурсы могут быть использованы в отгрузках того же периода, "*false*" означает, что закупки производятся в конце каждого периода, поэтому отгрузка этих ресурсов возможна только в следующих периодах; стандартный тип *bool*.

Ind – флаг, устанавливающий принцип учета запасов: FIFO или LIFO.

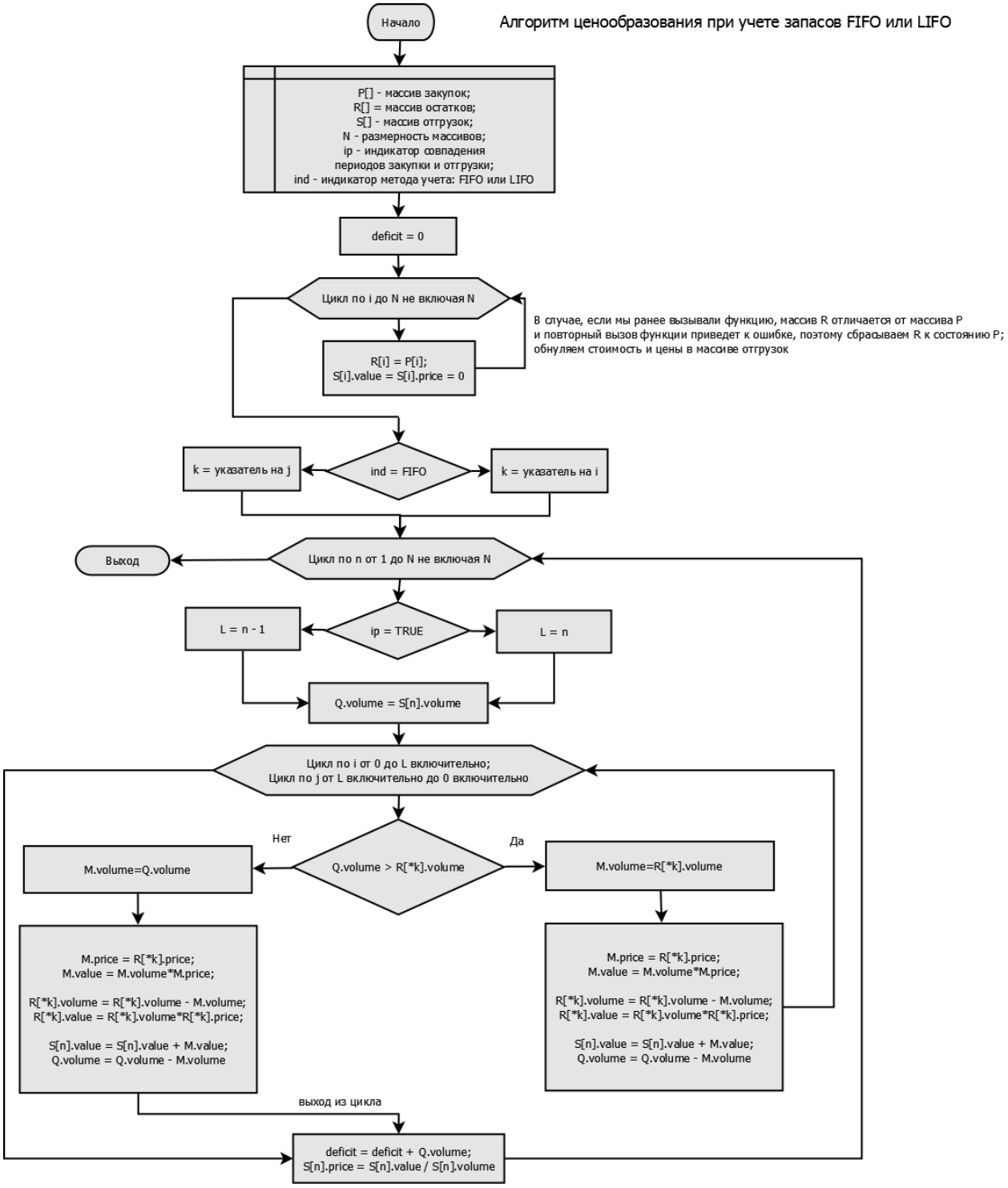

Рисунок 2 Блок-схема алгоритма учета запасов по принципам FIFO и LIFO

decimal PURcalc(strItem P[], const strItem S[], size_t N, bool ip, decimal Shr)

Функция рассчитывает объемы закупок, необходимые для выполнения плана отгрузок и обеспечения заданного остатка ресурсов (остаток задается как доля от объема отгрузок за период) и возвращает дефицит ресурсов в предплановом (нулевом) периоде. Рассчитываются закупки в плановых периодах - с первого до последнего (N-1). Закупки в нулевой период задаются вручную и характеризуют остатки ресурсов к началу проекта (элемент стартового баланса). Тип возвращаемого значения определяется псевдонимом decimal.

Блок-схема алгоритма приведена на рисунке ниже (Рисунок 3).

Параметры:

P[] - указатель на массив закупок/ поступлений на склад (изменяется в процессе работы функции); тип элемента массива strItem.

S[] - указатель на массив отгрузок, у которого заполнены только поля *volume* (массив изменять запрещено, const); тип элемента массива strItem.

N - размерность массивов закупок/ поступлений на склад, остатков и отгрузок, совпадает с количеством периодов проекта; стандартный тип *size_t*.

ip - индикатор, признак того, можно ли использовать закупаемые в каком-либо периоде ресурсы для отгрузки в этом же периоде: "*true*" означает, что закупка осуществляется в начале каждого периода и закупаемые ресурсы могут быть использованы в отгрузках того же периода, "*false*" означает, что закупки производятся в конце каждого периода, поэтому отгрузка этих ресурсов возможна только в следующих периодах; стандартный тип *bool*.

Shr - плановый остаток сырья, заданный, как доля отгрузок за период (минимальное число ноль – на складе поддерживаются нулевые остатки).

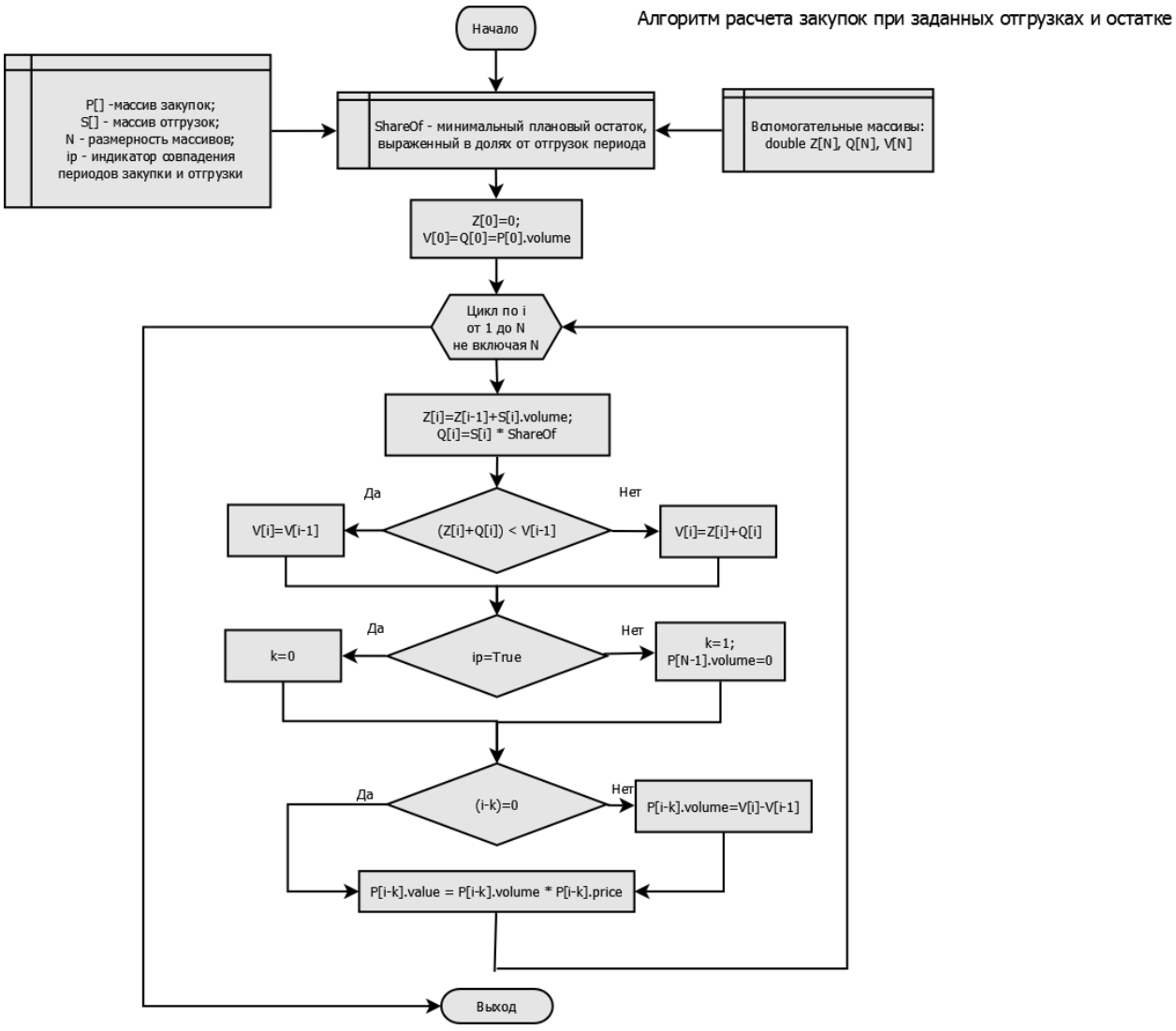

Рисунок 3 Блок-схема алгоритма расчета закупок по периодам

void BALcalc(const strItem P[], strItem B[], const strItem S[], size_t N)

Функция формирует массив балансовых остатков ресурсов на складе по периодам. Функцию следует вызывать только после корректного расчета объемов закупок (функцией PURcalc) и цен отгрузок (функцией FLFcalc или AVGcalc).

Параметры:

P[] - указатель на массив корректно рассчитанных закупок/ поступлений на склад (неизменен); тип элемента массива strItem.

B[] - указатель на массив балансовых остатков (изменяется в процессе работы функции); тип элемента массива strItem.

S[] - указатель на массив корректно рассчитанных отгрузок (неизменен); тип элемента массива strItem.

N - размерность массивов закупок/ поступлений на склад, остатков и отгрузок, совпадает с количеством периодов проекта; стандартный тип *size_t*.

АЛГОРИТМЫ УЧЕТА. СЕКЦИЯ PUBLIC

const decimal& CalcPrice()

Функция рассчитывает цены и стоимость отгружаемых ресурсов в каждом периоде на основе установленного пользователем принципа учета запасов: FIFO, LIFO или AVG, записывает результаты расчетов в массив Ship, возвращает величину дефицита сырья и материалов для обеспечения плановых отгрузок. Если дефицит lack не равен нулю, рассчитанные цены и стоимость могут нести некорректные значения. Функция вызывает методы AVGcalc, FLFcalc, в зависимости от установленного принципа учета запасов; если дефицит меньше заранее заданной «бесконечно малой величины» epsln, то дефицит обнуляется и вызывается функция BALcalc. Тип возвращаемого значения определяется псевдонимом decimal.

const decimal& CalcPurchase()

Функция проверяет флаг pcalc и, в случае установленного значения *calc*, вызывает метод PURcalc и рассчитывает объемы закупок, необходимые для выполнения плана отгрузок и обеспечения заданного остатка ресурсов, записывает результаты расчетов в массив Pur, возвращает величину дефицита ресурсов для обеспечения плановых отгрузок в нулевом периоде. Тип возвращаемого значения определяется псевдонимом decimal. Если же флаг pcalc установлен в значение *nocalc*, функция прекращает работу и возвращает ноль.

bool Resize(size_t _n)

Функция изменяет размеры массивов Pur, Rem, Bal, Ship типа strItem, устанавливая новое число периодов проекта, равное _n. При _n < PrCount, данные обрезаются, при _n > PrCount, - добавляются новые элементы массивов с нулевыми значениями. Функция возвращает *true* при удачном изменении размеров массивов, *false* - в противном случае.

Параметры:

_n – новое значение продолжительности проекта (новое количество периодов); стандартный тип *size_t*.

TLack Calculate()

Функция вызывает метод CalcPurchase и рассчитывает требуемый объем закупок/ поступлений на склад, если выставлен флаг pcalc, разрешающий автоматический расчет. Проверяет дефицит ресурсов для выполнения заданного плана отгрузок и остатков. Если дефицит lack больше заранее заданной «бесконечно малой величины» epsln, то функция возвращает величину этого дефицита и на этом работа функции заканчивается.

Если же величина дефицита меньше epsln, то функция вызывает метод CalcPrice и рассчитывает учетную удельную и полную себестоимости отгружаемой партии в соответствии с выбранным принципом учета запасов FIFO, LIFO или AVG. Функция повторно проверяет дефицит ресурсов для отгрузки со склада lack. Если дефицит lack больше заранее заданной «бесконечно малой величины» epsln, то функция возвращает величину этого дефицита и на этом работа функции заканчивается. Иначе величина дефицита обнуляется, и функция возвращает ноль. Информация возвращается в виде структуры типа TLack.

Надо помнить, что если lack > epsln, сделанные ранее расчеты объемных и стоимостных показателей могут быть некорректными.

GET-МЕТОДЫ

const size_t& Count() const

Возвращает const-ссылку на длительность проекта, выраженную в количестве периодов.

const string& Name() const

Возвращает const-ссылку на название номенклатурной единицы складского учета.

const string& Measure() const

Возвращает const-ссылку на название единицы натурального измерения номенклатурной единицы складского учета.

string AccMethod() const

Возвращает принцип учета запасов в виде текстовой строки.

string Permission() const

Возвращает флаг разрешения/запрещения отгрузок в одном периоде в виде текстовой строки.

string AutoPurchase() const

Возвращает флаг авторасчета/ ручного расчета закупок в виде текстовой строки.

const PurchaseCalc& GetAutoPurchase() const

Возвращает const-ссылку на флаг авторасчета/ ручного расчета закупок; тип возвращаемой величины PurchaseCalc.

const decimal& Share() const

Возвращает const-ссылку на норматив запаса ресурсов в долях от объема отгрузки. Тип возвращаемого значения определяется псевдонимом decimal.

const strItem* GetDataItem(const ChoiseData _cd, size_t _N) const

Функция возвращает информацию, содержащуюся в элементе массива с индексом _N в виде const-указателя на структуру типа strItem.

Параметры:

_cd – именованное целое число из перечисленных в типе ChoiseData, указывающее на массив, из которого нужно выбрать данные: “*purchase*” – из массива закупок/ поступлений (Pur), “*balance* - из массива остатков (Bal), *shipment* - из массива отгрузок (Ship);

_N – номер периода, совпадающий с индексом элемента выбранного массива.

const strItem* GetShipment() const

Метод возвращает константный указатель на массив отгрузок со склада Ship.

const strItem* GetPurchase() const

Метод возвращает константный указатель на массив закупок/поступлений на склад Pur.

const strItem* GetBalance() const

Метод возвращает константный указатель на массив остатков на складе Bal.

const decimal& GetLack() const

Метод возвращает величину дефицита ресурсов, хранящуюся в поле lack для отгрузки со склада. Тип возвращаемого значения определяется псевдонимом decimal.

SET-МЕТОДЫ

void SetName(const string& _name)

Меняет название номенклатурной единицы складского учета поле name.

Параметры:

_name – новое название номенклатурной единицы складского учета.

void SetMeasure(const string& _measure)

Меняет название единицы измерения номенклатурной единицы складского учета поле measure.

Параметры:

_measure – новое название единицы измерения номенклатурной единицы складского учета.

void SetPermission(bool _indr)

Устанавливает флаг разрешения/ запрещения отгрузок и закупок в одном периоде поле indr.

Параметры:

_indr – новый флаг разрешения/ запрещения отгрузок в одном периоде: *true* - можно, *false* – нельзя.

void SetAutoPurchase(PurchaseCalc _pcalc)

Устанавливает флаг автоматического/ ручного расчета закупок/ поступлений на склад поле pcalc.

Параметры:

_pcalc – новый флаг разрешения/ запрещения (*calc*/ *nocalc*) автоматического расчета закупок/поступлений на склад под требуемые объемы отгрузок.

void SetShare(decimal _share)

Устанавливает новый норматив запаса ресурсов на складе в долях от объема отгрузок поле share.

Параметры:

_share – новое значение величины запаса, выраженное в долях от объема закупок. Тип вещественного числа определяется псевдонимом decimal.

void SetAccount(AccountingMethod _acct)

Устанавливает новый принцип учета запасов поле acct.

Параметры:

_acct – новое значение принципа учета запасов: *FIFO*, *LIFO* или *AVG*.

bool SetDataItem(const ChoiseData _cd, const strItem _U, size_t _N)

Функция записывает в элемент массива с индексом _N (период проекта с номером _N) новые значения *volume*, *price* и *value* из переменной _U; выбор массива производится значением переменной _cd. Возвращает *true*, если запись удалась.

Параметры:

_cd – признак редактируемого массива: “purchase” - массив Pur, “balance” _ массив Bal, “shipment” – массив Ship;

_U – переменная с новыми данными, которые записываются в элемент массива;

_N – индекс редактируемого элемента массива (номер периода проекта).

bool SetPurchase(const strItem _unit[], size_t _count)

Функция загружает данные о закупке ресурсов на склад (объемы, цены, стоимость).

Параметры:

_unit[] – указатель на массив загружаемых данных типа strItem;

_count - размерность массива (может не совпадать с размерностью внутреннего массива закупок Pur, равной PrCount. Если размер переданного в функцию массива больше массива закупок, то данные обрезаются, иначе используется размер полученного массива).

bool SetPurPrice(const strItem _unit[], size_t _count)

Функция загружает данные о ценах закупаемых/поставляемых на склад ресурсов (только поля “price”). Поля с данными об объемах (“volume”) и стоимости (“value”) игнорируются и могут содержать любые значения.

Параметры:

_unit[] – указатель на массив загружаемых данных с ценами ресурсов типа strItem;

_count - размерность массива (может не совпадать с размерностью внутреннего массива закупок Pur, равной PrCount. Если размер переданного в функцию массива больше массива закупок, то данные обрезаются, иначе используется размер полученного массива).

void SetShipment(const decimal _unit[], size_t _count)

Функция загружает данные об объемах плановых отгрузок ресурсов со склада. Тип вещественного числа загружаемого массива определяется псевдонимом decimal.

Параметры:

_unit[] – указатель на массив загружаемых данных с ценами ресурсов типа decimal;

_count - размерность массива (может не совпадать с размерностью внутреннего массива закупок Ship, равной PrCount. Если размер переданного в функцию массива больше массива закупок, то данные обрезаются, иначе используется размер полученного массива).

МЕТОДЫ СОХРАНЕНИЯ СОСТОЯНИЯ ОБЪЕКТА В ФАЙЛ И ВОССТАНОВЛЕНИЯ ИЗ ФАЙЛА

bool StF(ofstream &_outF) const

Метод имплементации записи в файловую переменную текущего экземпляра класса (запись в файл, метод сериализации). Название метода является аббревиатурой, расшифровывается “StF” как “Save to File”.

Параметры:

&_outF - экземпляр класса ofstream для записи данных. При инициализации переменной *_outF* необходимо установить флаг бинарного режима открытия потока (не текстового!) ios::binary.

bool RfF(ifstream &_inF)

Метод имплементации чтения из файловой переменной текущего экземпляра класса (чтение из файла, метод десериализации). Название метода является аббревиатурой, расшифровывается “RfF” как “Read from File”.

Параметры:

&_inF - экземпляр класса ifstream для чтения данных. При инициализации переменной *_inF* необходимо установить флаг бинарного режима открытия потока (не текстового!) ios::binary.

МЕТОДЫ ВИЗУАЛЬНОГО КОНТРОЛЯ

void View() const

Метод выводит на экран состояние закупок Pur, остатков от партий на конец проекта Rem, балансовых остатков ресурсов в каждом периоде проекта Bal, отгрузок Ship. Метод использовался, как вспомогательный при разработке и отладки кода.

void ViewData(const ChoiseData _cd) const

Метод выводит на экран состояние конкретного выбранного массива Pur, Bal или Ship. выбор массива производится значением переменной _cd типа ChoiseData.

Параметры:

_cd – флаг выбора массива для вывода значений его элементов на экран: “purchase” – массив закупок Pur, “balance” – массив остатков Bal, “shipment” – массив отгрузок Ship.

КАК ПОЛЬЗОВАТЬСЯ КЛАССОМ clsSKU

Класс clsSKU может быть пригоден для самостоятельного применения в редких случаях *моноресурсного* склада (склада для *одной* номенклатурной позиции, *одного* SKU). Обычно класс clsSKU используется в составе класса clsStorage – класса для нескольких номенклатурных позиций. Тем не менее при желании использовать объект класса clsSKU в качестве самодостаточного, можно рекомендовать следующую последовательность действий.

1. Создаем новый экземпляр класса clsSKU в динамической памяти с помощью конструктора с параметрами.
2. Если использовался конструктор без ввода объемов отгрузок, вводим отгрузки с помощью метода SetShipment.
3. Вводим цены на ресурс в каждом периоде проекта с помощью метода SetPurPrice. Если подразумевается остаток ресурса на начало проекта (в нулевом периоде) или есть желание ввести объемы закупок/ поступлений на склад вручную, не используя автоматический расчет (nocalc), то ввод объемов, цен и стоимости можно объединить в методе SetPurchase.
4. При вводе некорректных данных, их изменение в любом из внутренних массивов класса в любом периоде проекта можно осуществить с помощью метода SetDataItem.
5. Рассчитываем *потребность в ресурсах* (при наличии флага calc), рассчитываем *учетную удельную и полную себестоимости* отгружаемой партии в соответствии с выбранным принципом учета запасов и проверяем дефицит с помощью метода Calculate.
6. При наличии дефицита, удаляем созданный объект, изменяем исходные данные и снова возвращаемся к п. 1, либо редактируем данные, как указано в п.4. Добиваемся отсутствия дефицита и, соответственно, корректности расчетных величин.

7. «Достаем» из экземпляра класса массивы с данными, содержащими в т.ч. и искомые расчетные величины: массив с *отгрузками* в натуральном, удельном и полном стоимостном выражении с помощью метода GetShipment, массив с *остатками* в натуральном, удельном и полном стоимостном выражении с помощью метода GetBalance, массив с *закупками/ поступлениями* на склад в натуральном, удельном и полном стоимостном выражении с помощью метода GetPurchase. Если нет необходимости в получении массивов и достаточно получить данные из конкретного массива в каком-либо заданном периоде проекта, можно воспользоваться функцией GetDataItem.
8. При желании вывести содержание массивов в стандартный поток std::cout, можно воспользоваться одним из двух методов визуального контроля: *View* или *ViewData*.
9. После того, как созданный в п.1 экземпляр класса clsSKU не нужен, его удаляем.
10. На любом этапе между п.п. 1 и 9 состояние объекта класса clsSKU можно сохранить в файл с последующим восстановлением из этого файла с помощью методов сериализации и десериализации.

ПРИМЕР ПРОГРАММЫ С КЛАССОМ clsSKU

Ниже представлен листинг программы, демонстрирующей работу класса clsSKU.

```
#include <iostream>
#include <warehouse_module.h>           // Подключаем модуль склада

using namespace std;

int main() {
    // Исходные данные
    size_t PrCount = 13;                // Количество периодов проекта
    AccountingMethod Amethod = AVG;     // Принцип учета запасов
    string Name_01 = "Pork Meat";       // Наименование позиции сырья
    string Meas_01 = "kg";              // Единица измерения
    bool PurchPerm = false;             // Разрешение закупок и отгрузок в одном периоде
    PurchaseCalc flag = calc;           // Разрешение на автоматический расчет закупок
    decimal Share = 0.5;                // Запас сырья в каждый период, выраженный в доле
                                        // от объема отгрузок за этот период
    strItem ship[PrCount] = {    0,  0,  0,
                                40,  5,  0,     // Отгрузки по периодам, начиная с 1-го
                                40,  0,  0,
                                45,  0,  0,
                                50,  0,  0,
                                55,  0,  0,
                                55,  0,  0,
                                55,  0,  0,
                                60,  0,  0,
                                60,  0,  0,
                                60,  0,  0,
                                60,  0,  0,
                                60,  0,  0 };

    strItem purc[PrCount] = {  100, 5, 500,     // Запас на начало проекта
                                 0,  6,  0,     // Закупочные цены по периодам
                                 0,  7,  0,
                                 0,  8,  0,
                                 0,  9,  0,
                                 0, 10,  0,
                                 0, 11,  0,
                                 0, 12,  0,
                                 0, 13,  0,
                                 0, 14,  0,
                                 0, 15,  0,
                                 0, 16,  0,
                                 0, 17,  0 };

                                        // Создаем склад для ресурса
    clsSKU* SKU_A = new clsSKU(PrCount, Name_01, Meas_01, Amethod, PurchPerm, flag, Share,
ship);
    SKU_A->SetPurchase(purc, PrCount);      // Загружаем остаток на начало проекта
                                            // и закупочные цены ресурса
    TLack lack = SKU_A->Calculate();        // Рассчитываем потребность в ресурсах
                                            // и проверяем дефицит
    cout << "Checking deficit. Deficit is " << lack.lack << endl;
    cout << "*** Purchase viewing ***" << endl;
    SKU_A->ViewData(purchase);              // Вывод данных закупок
    cout << "*** Shipment viewing ***" << endl;
    SKU_A->ViewData(shipment);              // Вывод данных отгрузок

    delete SKU_A;
    return 0;
}
```

Программа выводит на экран информацию о закупках и об отгрузках.

```
Checking deficit. Deficit is 0
*** Purchase viewing ***

By volume
Name            Measure              0          1          2          3          4          5          6          7          8
Pork Meat       kg              100.00       0.00      47.50      52.50      57.50      55.00      55.00      62.50      60.00

By volume
Name            Measure              9         10         11         12
Pork Meat       kg               60.00      60.00      60.00       0.00


By price
Name            Measure              0          1          2          3          4          5          6          7          8
Pork Meat       RUR/kg            5.00       6.00       7.00       8.00       9.00      10.00      11.00      12.00      13.00

By price
Name            Measure              9         10         11         12
Pork Meat       RUR/kg           14.00      15.00      16.00      17.00


By value
Name            Measure              0          1          2          3          4          5          6          7          8
Pork Meat       RUR             500.00       0.00     332.50     420.00     517.50     550.00     605.00     750.00     780.00

By value
Name            Measure              9         10         11         12
Pork Meat       RUR              840.00     900.00     960.00       0.00
```

Красной линией выделены: на первом рисунке расчетные величины закупок ресурсов, но втором – расчетные значения учетной себестоимости ресурсов при отгрузке.

```
*** Shipment viewing ***

By volume
Name            Measure              0          1          2          3          4          5          6          7          8
Pork Meat       kg                0.00      40.00      40.00      45.00      50.00      55.00      55.00      55.00      60.00

By volume
Name            Measure              9         10         11         12
Pork Meat       kg               60.00      60.00      60.00      60.00


By price
Name            Measure              0          1          2          3          4          5          6          7          8
Pork Meat       RUR/kg            0.00       5.00       5.00       6.41       7.52       8.55       9.52      10.51      11.54

By price
Name            Measure              9         10         11         12
Pork Meat       RUR/kg           12.51      13.50      14.50      15.50


By value
Name            Measure              0          1          2          3          4          5          6          7          8
Pork Meat       RUR               0.00     200.00     200.00     288.33     376.11     470.37     523.46     577.82     692.61

By value
Name            Measure              9         10         11         12
Pork Meat       RUR              750.87     810.29     870.10     930.03
```

КЛАСС clsStorage

Класс *clsStorage* предназначен для *моделирования учётных процессов* на складе ресурсов: комплектующих, сырья и материалов, готовой продукции и т.п. Экземпляр класса *clsStorage* описывает склад ресурсов для *множества* номенклатурных позиций (множества SKU). Класс представляет собой контейнер для нескольких экземпляров класса clsSKU (однотоварных складов).

Также как для класса clsSKU, применением класса *clsStorage* решаются задачи:

1. Расчёта объема и графика закупок/ поступлений ресурсов на склад, обеспечивающих заданный объем и график отгрузки ресурсов со склада и выполнение норматива неснижаемых остатков на складе многономенклатурных позиций
2. Расчёта учётной себестоимости ресурсов, отгружаемых со склада в каждый период отгрузки в соответствии с известными ценами ресурсов на входе склада в каждый период закупки/ поступления и выбранным принципом учета запасов.

ПОЛЯ КЛАССА clsStorage

Таблица 9 Поля класса clsStorage. Секция Private

Поле класса	Описание	Заметка

size_t PrCount	Тип size_t. Количество периодов проекта. Нулевой период – это период перед началом проекта	Значение устанавливается при создании объекта класса. Может корректироваться.
Currency hmcur	Тип Currency. Домашняя (основная) валюта проекта	Значение устанавливается при создании объекта класса. Может корректироваться. Домашняя валюта распространяется на все номенклатурные позиции (элементы типа clsSKU).
AccountingMethod acct	Переменная типа AccountingMethod, хранящая установленный пользователем принцип учета запасов: FIFO, LIFO или AVG.	Значение устанавливается при создании объекта класса. Корректировке без уничтожения экземпляра класса не подлежит. Принцип учета действует на все номенклатурные позиции (элементы типа clsSKU).
vector <clsSKU> stock	Тип vector<clsSKU>. Контейнер для множества экземпляров класса clsSKU (однотоварных складов)	Статическая переменная
atomic<bool> Calculation_Exit	Тип atomic<bool>. Флаг завершения работы метода Calculate(size_t, size_t)	Значение *false* устанавливается при создании объекта класса. При копировании и перемещении объектов класса не копируется, не перемещается, а инициализируется значением *false*. Не участвует в обмене состояниями между объектами с помощью метода swap. Не сериализуется и не десериализуется. Используется в методе Calculate(size_t, size_t) как флаг остановки дальнейших расчетов при обнаружении дефицита; позволяет остановить расчеты в других потоках при многопоточных вычислениях
clsProgress_shell<type_progress>* pshell{nullptr}	Тип указатель на класс clsProgress_shell. Адрес внешнего объекта-оболочки для индикатора прогресса. Тип самого индикатора определяется псевдонимом type_progress.	При создании объекта класса устанавливается значение *nullptr*: по умолчанию индикация прогресса не осуществляется

Методы класса clsStorage можно условно разделить на следующие группы:

- Конструкторы, деструктор, перегруженные операторы;
- Get-методы (методы в секции Private и методы в секции Public);
- Set-методы (методы в секции Private и методы в секции Public);
- Расчетные методы;
- Методы сохранения состояния объекта в файл и восстановления из файла;
- Методы визуального контроля

Ниже подробно описываются все методы.

КОНСТРУКТОРЫ, ДЕСТРУКТОР, ПЕРЕГРУЖЕННЫЕ ОПЕРАТОРЫ

clsStorage()

Конструктор по умолчанию (без параметров). Создает и инициализирует переменные: PrCount = 0; hmcur = RUR; acct = FIFO; создает на стеке пустой контейнер stock. Инициализирует флаг Calculation_Exit значением *false*.

clsStorage(size_t _pcnt, Currency _cur, AccountingMethod _ac)

Конструктор с вводом длительности проекта, валюты и принципа учета запасов. Инициализирует флаг Calculation_Exit значением *false*.

Параметры:

_pcnt – количество периодов проекта; устанавливает значение поля PrCount; тип *size_t*;

_cur – домашняя валюта проекта; устанавливает значение поля класса hmcur;

_ac – принцип учета запасов: FIFO, LIFO или AVG; устанавливает значение поля класса acct.

clsStorage(size_t _pcnt, Currency _cur, AccountingMethod _ac, size_t stocksize)

Конструктор с параметрами. Аналогичен предыдущему конструктору, дополнительно резервирует память контейнера stock для нескольких номенклатурных позиций[7]. Инициализирует флаг Calculation_Exit значением *false*.

Параметры:

_pcnt – количество периодов проекта; устанавливает значение поля класса PrCount;

_cur – домашняя валюта проекта; устанавливает значение поля класса hmcur;

_ac – принцип учета запасов: FIFO, LIFO или AVG; устанавливает значение поля класса acct;

Stocksize – количество номенклатурных позиций (экземпляров класса clsSKU), планируемых к добавлению в контейнер. Тип *size_t*.

clsStorage(const clsStorage &obj)

Конструктор копирования. Инициализирует флаг Calculation_Exit значением *false*.

Параметры:

&obj – ссылка на копируемый константный экземпляр класса clsStorage.

clsStorage(clsStorage &&obj)

Конструктор перемещения. Инициализирует флаг Calculation_Exit значением *false*.

Параметры:

&&obj - перемещаемый экземпляр класса clsStorage.

clsStorage::swap(clsStorage &obj) noexcept

Функция обмена значениями между экземплярами класса.

Параметры:

&obj – ссылка на обмениваемый экземпляр класса clsStorage.

[7] Резервирование памяти (https://cplusplus.com/reference/vector/vector/reserve/) существенно снижает нагрузку на компьютер при последующем добавлении новых элементов в контейнер.

clsStorage::operator=(const clsStorage &obj)

Перегрузка оператора присваивания копированием.

Параметры:

&obj – ссылка на копируемый константный экземпляр класса clsStorage.

clsStorage::operator=(clsStorage &&obj)

Перегрузка оператора присваивания перемещением

Параметры:

&&obj - перемещаемый экземпляр класса clsStorage.

~clsStorage()

Деструктор.

GET-МЕТОДЫ. СЕКЦИЯ PRIVATE

strItem* getresult(const strItem* (clsSKU::*)() const) const

Метод создает динамический массив типа strItem и возвращает на него указатель. Массив содержит информацию об *объемах*, *учетной удельной* и *полной себестоимости*. В зависимости от функции, указатель на которую передается в качестве параметра, информация в массиве относится либо к закупкам/ поставкам, либо к остаткам, либо к отгрузкам. Используется в методах класса clsStorage: GetPure, GetBal и GetShip.

Параметры:

const strItem* (clsSKU::*f)() const – указатель на один из методов класса clsSKU: GetPurchase – функция возврата информации о закупках/ поставках, GetBalance – функция возврата информации об остатках на складе, GetShipment - функция возврата информации об отгрузках.

GET-МЕТОДЫ. СЕКЦИЯ PUBLIC

const size_t& ProjectCount() const

Возвращает const-ссылку на длительность проекта, выраженную в количестве периодов. Тип возвращаемой ссылки *size_t*.

size_t Size() const

Возвращает размер контейнера stock, равный количеству номенклатурных позиций (количеству SKU). Тип возвращаемой величины *size_t*.

const string& Name(size_t i) const

Возвращает const-ссылку на наименование номенклатурной позиции для элемента контейнера с индексом i. Тип возвращаемой величины – ссылка на тип *string*.

Параметры:

I – Индекс элемента в контейнере stock, тип параметра *size_t*.

const string& Measure(size_t i) const

Возвращает const-ссылку на наименование единицы измерения номенклатурной позиции для элемента контейнера с индексом i. Тип возвращаемой величины – ссылка на тип *string*.

Параметры:

I – Индекс элемента в контейнере stock, тип параметра *size_t*.

strNameMeas* GetNameMeas()

Метод создает динамический массив типа strNameMeas и возвращает на него указатель. Массив содержит информацию о наименованиях номенклатурных позиций и наименованиях их единиц измерения для всех ресурсов, проходящих через склад.

string Permission(size_t i) const

Метод возвращает флаг разрешения/запрещения закупок и отгрузок в одном и том же периоде в виде текстовой строки для элемента контейнера с индексом i. Тип возвращаемой величины *string*.

Параметры:

I – Индекс элемента в контейнере stock, тип параметра *size_t*.

string AutoPurchase(size_t i) const

Метод возвращает признак автоматического/ ручного расчета закупок в виде текстовой строки для элемента контейнера с индексом i. Тип возвращаемой величины *string*.

Параметры:

I – Индекс элемента в контейнере stock, тип параметра *size_t*.

const size_t GetAutoPurchase(size_t i) const

Метод возвращает признак автоматического/ ручного расчета закупок в виде целого числа для элемента контейнера с индексом i. Тип возвращаемой величины *size_t*.

string HomeCurrency() const

Метод возвращает основную валюту проекта в виде текстовой строки. Тип возвращаемой величины *string*.

string Accounting() const

Метод возвращает принцип учета в виде текстовой строки. Тип возвращаемой величины *string*.

const decimal& Share(size_t i) const

Метод возвращает норматив запаса ресурсов в долях от объема отгрузки для элемента контейнера с индексом i. Тип возвращаемой величины определяется псевдонимом decimal.

Параметры:

I – индекс элемента в контейнере stock, тип параметра *size_t*.

decimal* GetShipPrice()

Метод создает массив с учетной удельной себестоимостью отгружаемых ресурсов для всей номенклатуры и возвращает указатель на него. Массив является одномерным аналогом матрицы размером stock.size()*PrCount. Тип возвращаемого указателя определяется псевдонимом decimal.

Метод обеспечивает совместимость с форматом входных данных для экземпляра класса производство, когда под ресурсами подразумеваются сырье, материалы, компоненты для производства продукции.

strItem* GetShip() const

Метод создает массив с объемами, учетной удельной и полной себестоимостями для всей номенклатуры отгрузок и возвращает указатель на него. Массив является одномерным аналогом матрицы размером stock.size()*PrCount[8]. Тип возвращаемого указателя strItem.

В коде метода используется вызов функции getresult с параметром &clsSKU::GetShipment.

strItem* GetPure() const

Метод создает массив с объемами, учетной удельной и полной себестоимостями для всей номенклатуры закупок/ поступлений и возвращает указатель на него. Массив является одномерным аналогом матрицы размером stock.size()*PrCount. Тип возвращаемого указателя strItem.

В коде метода используется вызов функции getresult с параметром &clsSKU::GetPurchase.

strItem* GetBal() const

Метод создает массив с объемами, учетной удельной и полной себестоимостями для всей номенклатуры остатков на складе и возвращает указатель на него. Массив является одномерным аналогом матрицы размером stock.size()*PrCount. Тип возвращаемого указателя strItem.

В коде метода используется вызов функции getresult с параметром &clsSKU::GetPurchase.

const strItem* GetDataItem(const ChoiseData _cd, size_t index, size_t _N) const

Метод возвращает данные для элемента контейнера (номенклатурной позиции, SKU) с индексом index и периодом _N. Данные возвращаются в виде const-указателя на структуру типа strItem, содержащую информацию об объемах, ценах и стоимости закупки/ поступления, балансовых остатках или отгрузки; выбор массива с данными производится значением переменной _cd.

В коде метода используется вызов одноименной функции clsSKU::GetDataItem, которой передаются параметры _cd и _N.

Параметры:

_cd – именованное целое число из перечисленных в типе ChoiseData, указывающее на массив, из которого нужно выбрать данные: “*purchase*” – из массива закупок/ поступлений (Pur), “*balance* - из массива остатков (Bal), *shipment* - из массива отгрузок (Ship);

Index - индекс элемента в контейнере stock, тип параметра *size_t*;

_N – номер периода, совпадающий с индексом элемента выбранного массива

SET-МЕТОДЫ. СЕКЦИЯ PRIVATE

void _setdataitem(size_t i, const auto val, void (clsSKU::*f)(const auto val))

Для SKU с индексом i метод устанавливает новые: наименование, единицу измерения, разрешение на отгрузку и закупку в одном и том же периоде, флаг автоматического/ ручного расчета закупок, норматив запаса сырья. Используется в методах класса clsStorage: SetName, SetMeasure, SetPermission, SetAutoPurchase, SetShare.

Параметры:

I – индекс элемента в контейнере stock, тип параметра *size_t*;

[8] Обращение к (i,j)-элементу такой «матрицы» с помощью арифметики указателей производится так: *(указатель_на_массив + количество_столбцов * i + j), где i - номер строки, j - номер столбца знак “*” перед открывающей скобкой – оператор разыменовывания.

val – новый устанавливаемый параметр; для наименования и единицы измерения параметр должен иметь тип *const string&*, для разрешения на отгрузку и закупку - тип *const bool*, для флага автоматического/ ручного расчета - тип *const PurchaseCalc*, для норматива остатков – тип *const decimal* (вещественное число);

void (clsSKU::*f)(const auto val) – указатель на один из методов класса clsSKU: SetName, SetMeasure, SetPermission, SetAutoPurchase, SetShare;

SET-МЕТОДЫ. СЕКЦИЯ PUBLIC

void Set_progress_shell(clsProgress_shell<type_progress>* val)

Метод присваивает указателю pshell адрес внешнего объекта-оболочки индикатора прогресса. Тип оболочки clsProgress_shell, тип индикатора прогресса определяется псевдонимом type_progress.

Параметры:

*val – указатель на внешний объект-оболочку для класса индикатора прогресса.

void SetCount(const size_t _n)

Метод устанавливает новое количество периодов проекта – изменяет значение поля clsStorage::PrCount. Изменение одноименных полей элементов контейнера clsSKU::PrCount происходит позднее, в теле другого метода clsStorage::Calculate(size_t, size_t). До вызова метода *Calculate(…)* значения поля clsStorage::PrCount и одноименных полей элементов контейнера могут быть разными.

Параметры:

_n – новое значение количества периодов (длительности) проекта. Тип *size_t*.

void SetName(size_t i, const string &_name)

Метод меняет наименование номенклатурной позиции (SKU) для элемента контейнера stock с индексом i. Для смены наименования метод вызывает функцию clsStorage::_setdataitem.

Параметры:

I – индекс элемента в контейнере stock, тип параметра *size_t*;

&_name - сылка на новое наименование номенклатурной позиции.

void SetMeasure(size_t i, const string &_measure)

Метод меняет наименование единицы измерения номенклатурной единицы (SKU) для элемента контейнера stock с индексом i. Для смены единицы измерения метод вызывает функцию clsStorage::_setdataitem.

Параметры:

I – индекс элемента в контейнере stock, тип параметра *size_t*;

&_ measure – сылка на новое наименование единицы измерения номенклатурной позиции.

void SetCurrency(const Currency _cur)

Метод меняет основную валюту проекта – изменяет значение поля clsStorage::hmcur.

Параметры:

_cur – новая валюта проекта. Тип Currency.

void SetPermission(size_t i, const bool _indr)

Устанавливает флаг разрешения/ запрещения отгрузок и закупок в одном и том же периоде для элемента контейнера с индексом i. Для смены флага метод вызывает функцию clsStorage::_setdataitem.

Параметры:

I – индекс элемента в контейнере stock, тип параметра *size_t*;

_indr – новый флаг разрешения/ запрещения отгрузок в одном периоде: *true* - можно, *false* – нельзя.

void SetAutoPurchase(size_t i, const PurchaseCalc _pcalc)

Устанавливает флаг автоматического/ ручного расчета закупок/ поступлений на склад для элемента контейнера с индексом i. Для смены флага метод вызывает функцию clsStorage::_setdataitem.

Параметры:

I – индекс элемента в контейнере stock, тип параметра *size_t*;

_pcalc – новый флаг автоматического/ ручного расчета закупок/ поступлений на склад: *calc* – автоматический расчет, *nocalc* – ручной.

void SetShare(size_t i, const decimal _share)

Устанавливает новый норматив запаса ресурсов на складе в долях от объема отгрузок для элемента контейнера с индексом i. Для изменения норматива запаса метод вызывает функцию clsStorage::_setdataitem.

Параметры:

I – индекс элемента в контейнере stock, тип параметра *size_t*;

_share – новое значение величины запаса, выраженное в долях от объема закупок. Тип вещественного числа определяется псевдонимом decimal.

void SetAccounting(const AccountingMethod _acct)

Устанавливает новый принцип учета запасов для всех элементов контейнера – изменяет значение поля clsStorage::acct.

Параметры:

_acct – новое значение принципа учета запасов: *FIFO*, *LIFO* или *AVG*.

bool SetDataItem(const ChoiseData _cd, const strItem _U, size_t index, size_t _N)

Функция записывает в элемент контейнера с индексом index и период проекта с номером _N новые значения *volume*, *price* и *value* из переменной _U; выбор массива элемента контейнера производится значением переменной _cd. Метод работает только с непустым контейнером. В случае успешной записи, метод возвращает *true*.

Параметры:

_cd – признак редактируемого массива: “purchase” - массив clsSKU::Pur, “balance” _ массив clsSKU::Bal, “shipment” – массив clsSKU::Ship;

_U – переменная с новыми данными, которые записываются в элемент массива; тип strItem;

Index – индекс элемента в контейнере stock, тип параметра *size_t*;

_N – номер периода проекта (индекс редактируемого элемента в выбранном массиве clsSKU::Pur, clsSKU::Bal или clsSKU::Ship в индивидуальном складе с индексом Index); тип *size_t*.

bool SetSKU(const string &Name, const string &Measure, PurchaseCalc _flag, decimal _share, bool _perm,

strItem _ship[], strItem _pur[])

Метод создает моноресурсный склад для новой номенклатурной позиции и добавляет его в контейнер (создает новый элемент непосредственно в контейнере – экземпляр класса clssSKU). Метод работает как с пустым, так и не пустым контейнером stock. Метод проверяет наличие в контейнере номенклатурной позиции с тем же наименованием. Если таковой нет, то вызывается конструктор с параметрами clsSKU и с вводом объемов отгрузок. В созданном элементе вызывается метод clsSKU::SetPurchase для ввода объемов закупок/поступлений на склад. Предварительно предполагается, что поля PrCount и acct заполнены корректно.

Параметры:

&Name – ссылка на наименование новой номенклатурной позиции; тип ссылка на *string*;

&Measure – ссылка на название единицы измерения новой номенклатурной позиции; тип ссылка на *string*;

_flag – флаг разрешения/ запрещения автоматического расчета закупок под требуемые объемы отгрузок; тип PurchaseCalc;

_share - запас ресурсов на складе в каждый период, выраженный в доле от объема отгрузок за этот период; тип вещественного числа определяется псевдонимом decimal;

_perm – флаг разрешения/ запрещения использовать закупаемые в каком-либо i-м периоде ресурсы для отгрузки в этом же периоде; тип *bool*;

_ship[] – указатель на массив с объемом отгрузок (в каждом элементе массива заполнены только поля volume, поля price и value заполнены нулями); тип указатель на массив strItem;

_pur[] – указатель на массив с объемами закупок/поступлений на склад; тип указатель на массив strItem.

bool SetPurchase(size_t isku, const strItem _unit[], size_t _count)

Метод загружает массив закупок _unit размером _count в индивидуальный склад с номером isku (для элемента контейнера с индексом isku вызывается метод clsSKU::SetPurchase, которому передаются параметры _unit[] и _count). Возвращает значение *true*, если операция прошла успешно, в противном случае возвращает *false*.

Параметры:

Isku – индекс элемента в контейнере stock, тип параметра *size_t*;

_unit[] указатель на массив с объемами закупок/поступлений на склад; тип указатель на массив strItem;

_count – размер массива _unit[]; может не совпадать со значением из поля PrCount; тип параметра *size_t*.

bool SetPurPrice(const strItem _unit[])

Функция загружает данные о ценах закупаемых/поставляемых на склад ресурсов (только поля “price”). Поля с данными об объемах (“volume”) и стоимости (“value”) игнорируются и могут содержать любые значения. Одномерный массив _unit является аналогом двумерной матрицы размером stock.size()*PrCount[9]. Для каждого элемента контейнера вызывается метод clsSKU::SetPurPrice, которому передается указатель на часть массива, имеющей отношение к этому элементу: (_unit+PrCount*i), где I – номер элемента контейнера и количество периодов PrCount. Возвращает значение *true*, если операция прошла успешно, в противном случае возвращает *false*.

Параметры:

[9] Обращение к (i,j)-элементу такой «матрицы» с помощью арифметики указателей производится так: *(указатель_на_массив + количество_столбцов * i + j), где i - номер строки, j - номер столбца, знак “*” перед открывающей скобкой – оператор разыменовывания.

_unit – указатель на массив загружаемых данных с ценами ресурсов типа strItem.

bool CheckPurchase(const strItem _pur[]) const

При флаге, запрещающем автоматический расчет закупок/ поступлений на склад clsSKU::pcalc, массив закупок должен содержать корректные цифры, т.е произведение *volume* * *price* должно быть равным *value*. Если это условие не соблюдается, расчетная себестоимость ресурса может считаться некорректно. Данный метод проверяет каждый элемент массива, переданного функции в качестве параметра. Возвращает значение *true*, если для каждого элемента массива соотношение между *volume*, *price* и *value* корректное, в противном случае возвращает *false*.

Параметры:

_pur[] – указатель на проверяемый массив типа strItem.

bool SetStorage(size_t RMCount, const strNameMeas RMNames[], decimal ShipPlan[], decimal PricePur[])

Метод создает склад сразу для нескольких номенклатурных позиций ресурсов. После создания "пустого" экземпляра класса clsStorage с помощью одного из двух конструкторов с параметрами, данный метод создает в контейнере моноресурсные склады для отдельных ресурсов в количестве, равном величине RMCount, копирует названия этим позициям и единицы измерения из массива RMNames, копирует объемы отгрузок в полях “volume” массива clsSKU::Ship и цены закупаемых/получаемых ресурсов в полях “price” массива clsSKU::Pur.

Этот метод устанавливает для всех частных моноресурсных складов флаг *разрешающий* автоматический расчет закупок (pcalc = calc), *нулевой* запас ресурсов на складе в каждый период (share = 0) и флаг *разрешающий* закупки и отгрузки в одном и том же периоде (indr = true); в дальнейшем эти параметры можно скорректировать для каждого склада отдельно другими методами класса.

Возвращает значение *true*, если операция прошла успешно, в противном случае возвращает *false*.

Параметры:

RMCount – число номенклатурных позиций ресурсов; тип *size_t*; при RMCount=0, метод ничего не делает и возвращает *false*;

RMNames – указатель на массив с названиями ресурсов и названиями единиц их натурального измерения, размерностью RMCount; тип strNameMeas; при указатели равном nullptr, метод ничего не делает и возвращает *false*;

ShipPlan – указатель на массив отгрузок всех ресурсов в каждом периоде проекта; массив представляет собой одномерный аналог двумерной матрицы размером RMCount*PrCount; тип вещественного числа, определяемый псевдонимом decimal; при указатели равном nullptr, метод ничего не делает и возвращает *false*;

PricePur – указатель на массив цен всех закупаемых ресурсов в каждом периоде проекта; массив представляет собой одномерный аналог двумерной матрицы размером RMCount*PrCount; тип вещественного числа, определяемый псевдонимом decimal; при указатели равном nullptr, метод ничего не делает и возвращает *false*.

bool SetStorage(size_t RMCount, const strNameMeas RMNames[], strItem ShipPlan[], strItem PricePur[])

Метод, аналогичный предыдущему, но массивы с отгрузками и ценами имеют тип strItem. Возвращает значение *true*, если операция прошла успешно, в противном случае возвращает *false*.

Метод разрешает принятие в качестве указателя на массив закупок/ поставок *PricePur* значение *nullptr*. В этом случае ввод закупок может быть осуществлен позже другими методами (SetPurchase – для ввода объемов, удельной и полной учетной себестоимости закупок в выбранный моноресурсный склад; SetPurPrice – для ввода цен/удельной учетной себестоимости сразу во все моноресурсные склады).

Параметры:

RMCount – число номенклатурных позиций ресурсов; тип *size_t*; при RMCount=0, метод ничего не делает и возвращает *false*;

RMNames – указатель на массив с названиями ресурсов и названиями единиц их натурального измерения, размерностью RMCount; тип strNameMeas; при указатели равном nullptr, метод ничего не делает и возвращает *false*;

ShipPlan – указатель на массив отгрузок всех ресурсов в каждом периоде проекта; массив представляет собой одномерный аналог двумерной матрицы размером RMCount*PrCount; тип strItem, используются только поля “volume”, другие поля не используются и должны быть заполнены нулями; при указатели равном nullptr, метод ничего не делает и возвращает *false*;

PricePur – указатель на массив цен всех закупаемых ресурсов в каждом периоде проекта; массив представляет собой одномерный аналог двумерной матрицы размером RMCount*PrCount; тип strItem, используются только поля “price”, другие поля не используются и должны быть заполнены нулями. Метод допускает использование указателя PricePur = *nullptr*.

bool SetData(const clsSKU &obj)

Метод добавляет в контейнер новый моноресурсный склад путем *копирования* отдельно созданного экземпляра класса clsSKU в контейнер. Перед копированием в контейнер нового элемента проверяется наличие в контейнере ресурса с таким же именем. Если совпадение не найдено, и операция копирования прошла успешно, возвращает значение *true*, в противном случае возвращает *false*.

Параметры:

&obj – ссылка на копируемый моноресурсный склад; тип clsSKU.

bool SetData(clsSKU &&obj)

Метод добавляет в контейнер новый моноресурсный склад путем *перемещения* отдельно созданного экземпляра класса clsSKU в контейнер. Перед перемещением в контейнер нового элемента проверяется наличие в контейнере ресурса с таким же именем. Если совпадение не найдено, и операция перемещения прошла успешно, возвращает значение *true*, в противном случае возвращает *false*.

Параметры:

&&obj – ссылка на перемещаемый моноресурсный склад; тип clsSKU.

void EraseSKU(size_t _i)

Метод удаляет номенклатурную позицию под номером _i и все связанные с ней данные.

Параметры:

_i – индекс элемента в контейнере; тип size_t.

РАСЧЕТНЫЕ МЕТОДЫ. СЕКЦИЯ PRIVATE

TLack Calculate(size_t bg, size_t en)

Многоцелевая функция, выполняющая следующие действия для каждого моноресурсного склада типа clsSKU, являющегося элементом контейнера с индексом от bg до (en-1) включительно:

- Проверяет флаг остановки расчетов Calculation_Exit. Если флаг установлен в *true*, завершает работу метода и возвращает нулевой дефицит.

- Вызывает у каждого элемента контейнера из заданного диапазона индексов метод clsSKU::SetAccount и устанавливает принцип учета запасов из поля clsStorage::acct контейнера в поле clsSKU::acct элемента.

- Проверяет количество периодов проекта clsSKU::PrCount у каждого элемента контейнера из заданного диапазона индексов: если оно отличается от количества, установленного в переменной clsStorage::PrCount, то для этого элемента вызывается функция clsSKU::Resize с параметром clsStorage::PrCount, которая приводит количество периодов у элемента контейнера к заданному.

- Поочередно вызывает у каждого элемента контейнера из заданного диапазона индексов метод clsSKU::Calculate, рассчитывая для этого элемента необходимый объем закупок (поля “volume” у массива clsSKU::Pur, если у элемента выставлен флаг calc), удельные и полные себестоимости отгружаемых партий (поля “price” и “value” массива clsSKU::Ship), а также дефицит ресурсов.

Функция завершает работу при нахождении первого дефицита и сообщает о размере этого дефицита и наименовании SKU, где найден дефицит. При обнаружении дефицита, флаг *Calculation_Exit* устанавливается в *true*. При многопоточных вычислениях установленный в *true* флаг обеспечивает остановку вычислений в других потоках. Дальнейшие расчеты не производятся. Информация возвращается в виде структуры типа TLack.

В случае успешного завершения всех расчетов, функция возвращает переменную типа TLack, у которой поле lack равно нулю, а поле Name равно пустой строке.

Все расчеты над элементами контейнера производятся последовательно.

Параметры:

bg – индекс первого элемента из диапазона обрабатываемых элементов контейнера; тип *size_t*;

en – индекс элемента, следующего за последней позицией из диапазона обрабатываемых элементов контейнера; тип *size_t.*

РАСЧЕТНЫЕ МЕТОДЫ. СЕКЦИЯ PUBLIC

TLack Calculate()

Многоцелевая функция, аналогичная предыдущей, обрабатывает *все* элементы контейнера. Вызывает метод Calculate(size_t bg, size_t en) с параметрами bg = 0, en = (stock.size()-1) и возвращает структуру типа TLack. Перед тем, как вернуть результаты расчетов, сбрасывает флаг Calculation_Exit в состояние *false*. Это позволяет использовать метод повторно. Все расчеты над всеми элементами контейнера производятся последовательно.

TLack Calculate_future()

Метод является альтернативой методу Calculate. В методе *Calculate_future* расчеты производятся в параллельных потоках, число которых на единицу меньше числа ядер используемого компьютера, и число обрабатываемых элементов контейнера в каждом потоке примерно одинаково. В каждом потоке вызывается метод Calculate(size_t bg, size_t en) с границами диапазона, предназначенного для обработки в этом потоке. В каждом отдельном потоке расчеты производятся последовательно, но сами потоки выполняются параллельно. Это позволяет существенно ускорить вычисления. Для организации потоков используются инструменты из стандартного заголовочного файла <future>.

При выявлении дефицита у какого-либо моноресурсного склада в процессе расчетов, поток, где найден дефицит, устанавливает флаг Calculation_Exit в состояние *true* и завершается. Это дает сигнал другим потокам также завершить работу. Все результаты расчетов записываются во временное хранилище. По завершению работы всех потоков, функция проверяет временное хранилище и ищет дефицит. Функция завершает работу при нахождении первого дефицита и сообщает о размере этого дефицита и наименовании SKU, где найден дефицит.

TLack Calculate_thread()

Этот метод идентичен по назначению и своей структуре предыдущему методу. Отличие в том, что в нем используются инструменты из стандартного заголовочного файла <thread>.

Выбор метода для проведения расчетов (*Calculate*, *Calculate_future* или *Calculate_thread*) остается за пользователем. Для совершения выбора необходимо принять во внимание широту номенклатуры ресурсов (количество элементов контейнера), длительность (количество периодов) проекта и тип используемого вещественного числа. Создание потоков – ресурсоемкий процесс, и на коротких проектах с малым числом номенклатурных позиций использование последовательных вычислений без создания потоков (метод *Calculate*) может оказаться быстрее. Использование вещественных чисел типа LongReal вместо встроенного типа *double* для получения результатов повышенной точности, потребует больше компьютерных ресурсов, и выигрыш от использования параллельных вычислений может стать заметнее.

В таблице ниже (Таблица 10) приведено среднее время вычислений для разных расчетных методов. Для каждого из трех расчетных методов было сделано 150 вычислений: по 50 вычислений с каждым принципом учета запасов. В качестве вещественного числа использовался собственный тип LongReal с количеством десятичных знаков мантиссы 64. Было установлено 480 периодов проекта. Расчет велся по 100 номенклатурным позициям. Вычисления проводились на ноутбуке ASUS UX303U со следующими характеристиками: Процессор Intel Core i3-6100U CPU @ 2.30 GHz; Оперативная память 4,00 Gb; 64-bit Operation System Windows 10 Home Single Language, x64-based processor. Расчеты проводились в трех параллельных потоках.

Таблица 10 Сравнение времени вычислений для разных методов класса clsStorage

Названия столбцов	AVG		FIFO		LIFO	
Названия строк	Среднее время выполнения, сек.	Стандартное отклонение времени выполнения, сек.	Среднее время выполнения, сек.	Стандартное отклонение времени выполнения, сек.	Среднее время выполнения, сек.	Стандартное отклонение времени выполнения, сек.
Calculate	102,1145	0,8944	15,0407	0,1689	3,5799	0,1221
Calculate_future	48,4498	0,7373	7,1692	0,0858	1,6881	0,0214
Calculate_thread	48,5894	0,9135	7,1472	0,0701	1,6901	0,0293

Каждая финансово-экономическая модель строится, как правило, для многократного использования: заказчику модели часто нужны ответы на вопрос «что, если…». Это означает подстановку новых данных и расчет новых результатов. При этом обычно широта номенклатуры ресурсов во всех интересующих заказчика сценариях более-менее одинакова и длительность проекта более-менее укладывается в похожие рамки. Поэтому при создании модели можно попробовать применение всех трех расчетных методов и выбрать наиболее подходящий.

В Приложении XXX приведены тестовые данные, программный код теста скорости вычислений и результаты тестирования каждого метода.

МЕТОДЫ СОХРАНЕНИЯ СОСТОЯНИЯ ОБЪЕКТА В ФАЙЛ И ВОССТАНОВЛЕНИЯ ИЗ ФАЙЛА

bool StF(ofstream &_outF)

Метод имплементации записи в файловую переменную текущего экземпляра класса (запись в файл, метод сериализации).

Параметры:

&_outF - экземпляр класса ofstream для записи данных. При инициализации переменной *_outF* необходимо установить флаг бинарного режима открытия потока (не текстового!) ios::binary.

bool RfF(ifstream &_inF)

Метод имплементации чтения из файловой переменной текущего экземпляра класса (метод десериализации).

Параметры:

&_inF - экземпляр класса ifstream для чтения данных. При инициализации переменной *_inF* необходимо установить флаг бинарного режима открытия потока (не текстового!) ios::binary.

МЕТОДЫ ВИЗУАЛЬНОГО КОНТРОЛЯ

void ViewSettings() const

Функция выводит на экран информацию о настройках для расчета. К настройкам относятся

- общие параметры проекта: длительность проекта, домашняя (основная) валюта, метод учета запасов
- и параметры, относящиеся к конкретному моноресурсному складу: разрешение/запрет на закупки и отгрузки в одном периоде, признак автоматического/ ручного расчета закупок, норматив запаса ресурсов.

void View() const

Функция выводит на экран информацию по всему складу, путем обращения к методу clsSKU::View() каждого элемента контейнера в цикле.

void ViewChoise(ChoiseData _arr, ReportData flg) const

В зависимости от выбранного признака *_arr*, метод выводит на экран закупки/поступления, балансовые остатки или отгрузки ресурсов; в зависимости от выбранного флага *flg*, вывод этих данных осуществляется в натуральном, удельном стоимостном или полном стоимостном выражении.

Параметры:

_arr – параметр для выбора данных: “purchase” – закупки/ поступления, “balance” – остатки ресурсов на складе, “shipment” – отгрузки ресурсов со склада; тип ChoiseData;

flg – флаг для выбора типа представляемых данных: “volume” - в натуральном, “value” - в полном стоимостном, “price” - в удельном стоимостном измерении; тип ReportData.

КАК ПОЛЬЗОВАТЬСЯ КЛАССОМ clsStorage

Класс clsStorage – основной класс, с которым придется иметь дело пользователю, моделирующему учетные процессы на складе. Можно рекомендовать следующую последовательность действий:

1. Создаем новый экземпляр класса clsStorage в динамической памяти с помощью конструктора с параметрами.
2. Создание элементов контейнера (наполнение контейнера экземплярами класса clsSKU) можно осуществлять несколькими путями:
   a. Можно создать *сразу все элементы* контейнера с помощью одного из двух методов SetStorage с вводом отгрузок и цен в виде массивов вещественных чисел, тип которых определяется псевдонимом decimal или с вводом отгрузок и цен в виде массивов типа strItem;
   b. Можно *последовательно* создавать отдельные элементы сразу *внутри* контейнера с помощью метода SetSKU;
   c. Можно последовательно создавать отдельные элементы контейнера путем копирования или перемещения *ранее созданных* объектов типа *clsSKU* в контейнер с помощью методов копирования или перемещения SetData.
3. При вводе некорректных данных, их изменение в любом из внутренних массивов класса в любом периоде проекта можно осуществить с помощью метода SetDataItem.

4. Вызываем метод Calculate (Calculate_future или Calculate_thread). Этот метод вызывает у каждого элемента свой метод clsSKU::Calculate, который рассчитывает для данного моноресурсного склада *потребность в ресурсах* (при наличии у него флага clsSKU::calc), *учетную удельную и полную себестоимости* отгружаемой партии в соответствии с выбранным принципом учета запасов и сообщает о дефиците, если таковой имеется и имени номенклатурной позиции, где он обнаружен.
5. «Достаем» из экземпляра класса массивы с данными, содержащими в т.ч. и искомые расчетные величины: массив с *учетной удельной себестоимостью* ресурсов для всей номенклатуры с помощью метода GetShipPrice (тип массива определяется псевдонимом decimal), массив с *отгрузками* в натуральном, удельном и полном стоимостном выражении с помощью метода GetShip, массив с *остатками* в натуральном, удельном и полном стоимостном выражении с помощью метода GetBal, массив с *закупками/ поступлениями* на склад в натуральном, удельном и полном стоимостном выражении с помощью метода GetPure (типы массивов, указатели на которые возвращают три последние функции, strItem). Все массивы представляют собой одномерные аналоги двумерной матрицы размером stock.size()*PrCount.
6. Если нет необходимости в получении массивов и достаточно получить данные из конкретного массива в каком-либо заданном периоде проекта, можно воспользоваться функцией GetDataItem.
7. Для отображения данных в стандартный поток std::cout, можно воспользоваться следующими методами:
    a. методом ViewSettings – для вывода информации о длительности проекта, домашней валюте, методе учета запасов, о разрешении/запрете на закупки и отгрузки в одном периоде, о признаке автоматического/ ручного расчета закупок, о нормативе запаса ресурсов;
    b. одним из двух методов визуального контроля: View или ViewChoise – для вывода информации о закупках/ поступлениях, остатках на складе и об отгрузках.
8. После того, как созданный в п.1 экземпляр класса clsStorage не нужен, его удаляем.
9. На любом этапе между п.п. 1 и 8 состояние объекта класса clsSKU можно сохранить в файл с последующим восстановлением из этого файла с помощью методов сериализации и десериализации.

В финансово-экономических моделях класс *clsStorage* может поприсутствовать в виде двух экземпляров разного назначения: один, как *Склад готовой продукции*, другой – как *Склад сырья и материалов*. Между ними может находится объект другого класса clsManufactory, описывающий *Производство*. В такой конструкции, например, для *Склада готовой продукции*, может потребоваться *двухэтапная* работа:

1. На первом этапе в созданный экземпляр класса clsStorage вводятся данные о *плановых отгрузках* клиентам в *натуральном выражении* и *норматив запаса* готовых продуктов на складе. Ввод может осуществляться, например, методом SetStorage(size_t, const strNameMeas*, strItem*, strItem*). Далее производится расчет *плановых поступлений* в *натуральном выражении* на склад одним из трех вычислительных методов (Calculate, Calculate_future или Calculate_thread). Рассчитанные таким образом данные вводятся в качестве уже исходных данных в другие объекты с помощью их собственных методов ввода (например, в объект «Производство») для вычисления в этих объектах удельной учетной себестоимости (например, производственной себестоимости).
2. На втором этапе данные с *удельной учетной себестоимостью* (например, производственной себестоимостью) *плановых поступлений*, полученные от соседнего объекта, загружаются в склад готовой продукции методами SetPurPrice или SetPurchase. Далее производится расчет *удельной и полной учетной себестоимости отгружаемой* клиентам продукции одним из трех вычислительных методов (Calculate, Calculate_future или Calculate_thread).

Таким образом, класс *clsStorage* позволяет осуществлять *двухэтапный ввод данных* и *двухэтапные расчеты* для совместимости с другими классами библиотеки FROMA2.

ПРИМЕР ПРОГРАММЫ С КЛАССОМ clsStorage

Оглавление >>>

Ниже представлен листинг программы, демонстрирующей работу класса clsStorage. Данная программа может быть использована для тестирования быстродействия методов Calculate, Calculate_future или Calculate_thread при разных количестве периодов проекта PrCount и количестве номенклатурных позиций ресурсов RMCount (см. код ниже). С целью сокращения объема выводимой на экран информации, в данном примере эти величины малы (17 и 10 соответственно).

```
#include <iostream>
#include <warehouse_module.h>           // Подключаем модуль склада

using namespace std;

int main()
{
    setlocale(LC_ALL, "Russian");       // Установка русского языка для вывода
    cout << endl;
    cout << "************************************************************" << endl;
    cout << "***   Тестирование расчетных методов класса clsStorag   ***" << endl;
    cout << "************************************************************" << endl << endl;

    /** Общие данные проекта **/

    size_t PrCount = 17;            // Количество периодов проекта
    Currency Cur = RUR;             // Домашняя валюта проекта
    AccountingMethod Amethod = AVG; // Принцип учета запасов
    size_t RMCount = 10;            // Количество номенклатурных позиций
    bool PurchPerm = true;          // Разрешение закупок и отгрузок в одном периоде
    PurchaseCalc flag = calc;       // Разрешение на автоматический расчет закупок
```

```
    double Share = 0.5;             // Запас сырья в каждый период, выраженный
                                    // в доле от объема отгрузок за этот период
    TLack tlack;                    // Дефицит сырья и материалов


/** Формирование массива имен сырья и материалов с единицами измерения **/

    strNameMeas* RMNames = new strNameMeas[RMCount];
    for(size_t i{}; i<RMCount; i++) {
        (RMNames+i)->name = "Rawmat_" + to_string(i);
        (RMNames+i)->measure = "kg";
    };

/** Формирование тестового массива отгрузок **/

    strItem* ship = new strItem[PrCount];
    for(size_t i=sZero; i<PrCount; i++) {
        (ship+i)->volume = 40.0;
        (ship+i)->value = (ship+i)->price  =  0.0;
    };

    /** Формирование тестового массива закупок (только закупочных цен) **/

    strItem* purc = new strItem[PrCount];
    for(size_t i=sZero; i<PrCount; i++) {
        (purc+i)->volume = (purc+i)->value = 0.0;
        (purc+i)->price  =  (5.0+i);
    };

/** Создаем склад для нескольких продуктов **/

    // Создаем контейнер для склада
    clsStorage* Warehouse = new clsStorage(PrCount, Cur, Amethod, RMCount);
    for(size_t i=sZero; i<RMCount; i++) {   // Добавляем склад для i-го продукта
        if(!Warehouse->SetSKU((RMNames+i)->name, (RMNames+i)->measure, flag,\
            Share, PurchPerm, ship, purc)) {
            cout << "Ошибка добавления склада для " << (RMNames+i)->name << endl;
            delete[] RMNames;       // Освобождение динамической памяти
            delete[] ship;
            delete[] purc;
            delete Warehouse;
            return EXIT_FAILURE;    // выходим из программы с кодом неудачного завершения
        };
    };

    /** Рассчитываем проект **/

    int stop;
    cout << "Приостановка перед работой алгоритма. Для продолжения нажмите ENTER";
    stop = getchar();        // Приостановка до ввода любого символа
    clock_t start = clock(); // Начальное количество тиков процессора, которое сделала
программа
    // Расчет проекта и расчет дефицита.
    // tlack = Warehouse->Calculate();           // Последовательные вычисления
    // tlack = Warehouse->Calculate_future();    // Параллельные вычисления
    tlack = Warehouse->Calculate_thread();       // Параллельные вычисления
    if(tlack.lack>dZero)
        cout << "Дефицит для " << tlack.Name << " равен " << tlack.lack << endl;
    clock_t end = clock();  // Конечное количество тиков процессора, которое сделала
программа
    double seconds = (double)(end - start) / CLOCKS_PER_SEC;    // Подсчет секунд
выполнения алгоритма
    printf("Время вычислений: %f секунд\n", seconds);           // Вывод времени работы
процессора на экран
    cout << "Приостановка после работы алгоритма. Для продолжения нажмите ENTER";
    stop = getchar();

    /** Методы визуального контроля **/
    // Отображение удельной себестоимости отгружаемых ресурсов
    Warehouse->ViewChoise(shipment, price);

    /** Освобождение динамической памяти **/

    delete[] RMNames;
    delete[] ship;
    delete[] purc;
    delete Warehouse;

    return EXIT_SUCCESS;
```

В методе *Warehouse->ViewChoise(shipment, price)* мы выбрали вывод на экран информации об удельной учетной себестоимости (*price*) отгружаемых (*shipment*) ресурсов.

```
By price
Name            Measure              0          1          2          3          4          5          6          7          8
Rawmat_0        RUR/kg            0.00       6.00       6.67       7.56       8.52       9.51      10.50      11.50      12.50
Rawmat_1        RUR/kg            0.00       6.00       6.67       7.56       8.52       9.51      10.50      11.50      12.50
Rawmat_2        RUR/kg            0.00       6.00       6.67       7.56       8.52       9.51      10.50      11.50      12.50
Rawmat_3        RUR/kg            0.00       6.00       6.67       7.56       8.52       9.51      10.50      11.50      12.50
Rawmat_4        RUR/kg            0.00       6.00       6.67       7.56       8.52       9.51      10.50      11.50      12.50
Rawmat_5        RUR/kg            0.00       6.00       6.67       7.56       8.52       9.51      10.50      11.50      12.50
Rawmat_6        RUR/kg            0.00       6.00       6.67       7.56       8.52       9.51      10.50      11.50      12.50
Rawmat_7        RUR/kg            0.00       6.00       6.67       7.56       8.52       9.51      10.50      11.50      12.50
Rawmat_8        RUR/kg            0.00       6.00       6.67       7.56       8.52       9.51      10.50      11.50      12.50
Rawmat_9        RUR/kg            0.00       6.00       6.67       7.56       8.52       9.51      10.50      11.50      12.50

By price
Name            Measure              9         10         11         12         13         14         15         16
Rawmat_0        RUR/kg           13.50      14.50      15.50      16.50      17.50      18.50      19.50      20.50
Rawmat_1        RUR/kg           13.50      14.50      15.50      16.50      17.50      18.50      19.50      20.50
Rawmat_2        RUR/kg           13.50      14.50      15.50      16.50      17.50      18.50      19.50      20.50
Rawmat_3        RUR/kg           13.50      14.50      15.50      16.50      17.50      18.50      19.50      20.50
Rawmat_4        RUR/kg           13.50      14.50      15.50      16.50      17.50      18.50      19.50      20.50
Rawmat_5        RUR/kg           13.50      14.50      15.50      16.50      17.50      18.50      19.50      20.50
Rawmat_6        RUR/kg           13.50      14.50      15.50      16.50      17.50      18.50      19.50      20.50
Rawmat_7        RUR/kg           13.50      14.50      15.50      16.50      17.50      18.50      19.50      20.50
Rawmat_8        RUR/kg           13.50      14.50      15.50      16.50      17.50      18.50      19.50      20.50
Rawmat_9        RUR/kg           13.50      14.50      15.50      16.50      17.50      18.50      19.50      20.50
```

ПРОИЗВОДСТВО

Основные типы, константы, классы и методы, реализующие алгоритмы для производства, находятся в модуле MANUFACT. Состав модуля приведен в таблице ниже.

Таблица 11 Состав модуля Manufact

Состав модуля	Описание	Заметка
void Sum	Вспомогательная функция	Метод суммирует поэлементно два массива, результат сохраняет в первом массиве
void Dif	Вспомогательная функция	Метод вычитает поэлементно два массива, результат сохраняет в первом массиве
decimal *Mult	Вспомогательная функция	Метод возвращает результат умножения элементов двух массивов с одинаковыми индексами друг на друга. Результат возвращается в виде указателя на новый массив.
class clsRecipeItem	Класс технологической карты	Экземпляр класса представляет собой технологическую карту производства одной номенклатурной позиции готовой продукции; содержит методы, позволяющие моделировать учетные процессы в производстве этого единственного продукта
class clsManufactItem	Класс Производство для одной номенклатурной позиции (монопроизводство)	Экземпляр класса представляет собой контейнер для класса технологической карты (объекта типа *clsRecipeItem*). Содержит исходные, расчетные данные и методы управления объектом типа *clsRecipeItem*
class clsManufactory	Класс Производство для множества номенклатурных позиций	Экземпляр класса представляет собой контейнер для нескольких объектов типа *clsManufactItem* и позволяет моделировать учетные процессы в производстве множества номенклатурных единиц

Помимо указанных в таблице членов, в заголовочном файле manufact_module.h содержится ряд строковых констант, используемых при выводе информации на экран/файл.

ВСПОМОГАТЕЛЬНЫЕ МЕТОДЫ

inline void Sum(decimal arr1[], const decimal arr2[], size_t N);

Метод суммирует поэлементно два массива (складываются элементы с одинаковыми индексами), результат сохраняет в первом массиве, замещая его первоначальные элементы. Оба массива должны иметь одинаковый размер N. Метод *inline* для встраивания его кода в точку вызова. Метод не проверяет совпадение размерности массивов, не проверят существование массивов (не nullptr), это ответственность пользователя.

Параметры:

arr1 – указатель на первый массив; тип вещественного числа определяется псевдонимом decimal;

arr2 – константный указатель на второй массив; тип вещественного числа определяется псевдонимом *decimal*;

N - размерность обоих массивов; стандартный тип *size_t*.

inline void Dif(decimal arr1[], const decimal arr2[], size_t N)

Метод вычитает поэлементно два массива (разность между элементом первого массива и элементом второго массива с одинаковыми индексами), результат сохраняет в первом массиве, замещая его первоначальные элементы. Оба массива должны иметь одинаковый размер N. Метод *inline* для встраивания его кода в точку вызова. Метод не проверяет совпадение размерности массивов, не проверят существование массивов (не nullptr), это ответственность пользователя.

Параметры:

arr1 – указатель на первый массив; тип вещественного числа определяется псевдонимом decimal;

arr2 – константный указатель на второй массив; тип вещественного числа определяется псевдонимом *decimal*;

N - размерность обоих массивов; стандартный тип *size_t*.

inline decimal *Mult(const decimal arr1[], const decimal arr2[], const size_t N)

Метод возвращает результат умножения элементов двух массивов с одинаковыми индексами друг на друга. Результат возвращается в виде указателя на новый массив. Тип вещественного числа возвращаемого массива определяется псевдонимом decimal. Оба массива должны иметь одинаковый размер N. Метод *inline* для встраивания его кода в точку вызова. Метод не проверяет совпадение размерности массивов, не проверят существование массивов (не nullptr), это ответственность пользователя. Если происходит ошибка выделения памяти новому массиву, метод возвращает nullptr.

Параметры:

arr1 – константный указатель на первый массив; тип вещественного числа определяется псевдонимом decimal;

arr2 – константный указатель на второй массив; тип вещественного числа определяется псевдонимом *decimal*;

N - размерность обоих массивов; стандартный тип *size_t*.

КЛАСС clsRecipeItem

Экземпляр класса clsRecipeItem представляет собой *рецептуру/ технологическую карту* производства одной номенклатурной позиции готовой продукции или услуги; содержит методы, позволяющие *моделировать учетные процессы* в производстве этого единственного продукта.

Основные *задачи*, которые можно решить применением класса clsRecipeItem:

1. Сохранить и, при необходимости, вернуть рецептуру / технологическую карту продукта;
2. Рассчитать потребность в разных ресурсах и график обеспечения ими для производства одного какого-то конкретного продукта;
3. Рассчитать объем, полную и удельную ресурсную себестоимость незавершенного производства и готового продукта одного конкретного наименования.

ПОЛЯ КЛАССА clsRecipeItem

Таблица 12 Поля класса clsRecipeItem. Секция private

Поле класса	Описание	Заметка
string name	Тип *string*. Название продукта	Значение устанавливается при создании объекта класса. Может корректироваться.
string measure	Тип *string*. Название единицы измерения натурального объема продукта	Значение устанавливается при создании объекта класса. Может корректироваться.
size_t duration	Тип *size_t*. Длительность производственного цикла	Значение устанавливается при создании объекта класса. Может корректироваться.
size_t rcount	Тип *size_t*. Количество позиций в номенклатуре ресурсов, требуемых для производства продукта	Значение устанавливается при создании объекта класса. Может корректироваться.
strNameMeas *rnames	Тип strNameMeas. Указатель на одномерный динамический массив с названиями ресурсов и единицами их измерения. Размерность *rcount*	Значение устанавливается при создании объекта класса. Может корректироваться
decimal *recipeitem	Тип decimal. Указатель на одномерный динамический массив с рецептурой/ технологической картой продукта.	Массив представляет собой одномерный аналог двумерной матрицы размером rcount*duration. Обращение к i,j-элементу матрицы производится так: *(recipeitem + duration * i + j), где i - номер строки, j - номер столбца. Условная (i,j)-ячейка матрицы содержит расход i-го ресурса на производство единицы продукта в j-м периоде от начала его производственного цикла

Методы класса *clsRecipeItem* можно условно разделить на следующие группы:

- Конструкторы, деструктор, перегруженные операторы;
- Алгоритмы учета (методы в секции Private и методы в секции Public);
- Get-методы;
- Методы сохранения состояния объекта в файл и восстановления из файла;
- Методы визуального контроля

Ниже подробно описываются все методы.

КОНСТРУКТОРЫ, ДЕСТРУКТОР, ПЕРЕГРУЖЕННЫЕ ОПЕРАТОРЫ

clsRecipeItem()

Конструктор по умолчанию (без параметров). Создает и инициализирует переменные name = “”, measure = “”, duration = 0, rcount = 0. Динамические массивы rnames и recipeitem не создаются, указатели на них устанавливаются в *nullptr*.

clsRecipeItem(const string &_name, const string &_measure, const size_t _duration, const size_t _rcount, const strNameMeas _rnames[], const decimal _recipeitem[]);

Конструктор с вводом названия продукта, единицы его натурального измерения, длительности производственного цикла, массивов с названиями ресурсов, участвующих в технологическом цикле, рецептуры/ технологической карты продукта.

Параметры:

&_name – ссылка на название продукта; устанавливает значение поля класса name; тип *string*;

&_measure – ссылка на название единицы измерения натурального объема продукта; устанавливает значение поля класса measure; тип *string*;

_duration – длительность производственного цикла; устанавливает значение поля класса duration; тип *size_t*.

_rcount – количество позиций ресурсов, требуемых для производства продукта; устанавливает значение поля класса rcount; тип *size_t*.

_rnames[] - указатель на массив с наименованиями ресурсов и единицами их измерения размером *_rcount*; устанавливает значение поля класса rnames; тип элемента массива strNameMeas;

_recipeitem[] – указатель на массив с рецептурами (одномерный аналог матрицы размерностью *_rcount** *_duration*); устанавливает значение поля класса recipeitem; вещественный тип элемента массива определяется псевдонимом decimal.

clsRecipeItem(const clsRecipeItem &obj)

Конструктор копирования.

Параметры:

&obj – ссылка на копируемый константный экземпляр класса *clsRecipeItem*.

clsRecipeItem(clsRecipeItem &&obj)

Конструктор перемещения.

Параметры:

&&obj - перемещаемый экземпляр класса *clsRecipeItem*.

clsRecipeItem &operator= (const clsRecipeItem &obj)

Перегрузка оператора присваивания копированием.

Параметры:

&obj – ссылка на копируемый константный экземпляр класса *clsRecipeItem*.

clsRecipeItem &operator= (clsRecipeItem &&obj)

Перегрузка оператора присваивания перемещением

Параметры:

&&obj - перемещаемый экземпляр класса *clsRecipeItem*.

~clsRecipeItem()

Деструктор.

void swap(clsRecipeItem& obj) noexcept

Функция обмена значениями между экземплярами класса.

Параметры:

&obj – ссылка на обмениваемый экземпляр класса *clsRecipeItem*.

АЛГОРИТМЫ УЧЕТА. СЕКЦИЯ PRIVATE

decimal *CalcRawMatVolumeItem(const size_t PrCount, const size_t period, const decimal volume) const

Метод рассчитывает объем потребления ресурсов в каждом периоде *технологического цикла* в зависимости от требуемого объема готового продукта на выходе этого цикла. Объем рассчитывается в натуральном выражении. Метод рассчитывает потребность в ресурсах только для одного *единичного производственного цикла* (ЕПЦ). ЕПЦ — это процесс производства одного наименования продукта в каком-либо отдельном периоде проекта.

Метод возвращает указатель на вновь создаваемый одномерный динамический массив, аналог двумерной матрицы, размером rcount**PrCount* с рассчитанными объемами потребных в производстве ресурсов: число строк матрицы совпадает с числом позиций ресурсов, число столбцов – с длительностью проекта. (нулевой период, соответствующий стартовому балансу, не задействуется, остается равным нулю). Тип возвращаемых значений массива определяется псевдонимом decimal.

Параметры:

PrCount - число периодов проекта; тип *size_t*;

period - номер периода проекта, в котором требуется готовый произведенный продукт в заданном количестве; тип *size_t*;

volume - требуемый объем этого продукта в заданном периоде; тип вещественного числа определяется псевдонимом decimal.

decimal *CalcWorkingVolumeItem(const size_t PrCount, const size_t period, const decimal volume) const

Метод рассчитывает объемы незавершенного производства в каждом периоде *технологического цикла* в зависимости от требуемого объема готового продукта на выходе этого цикла. Объемы рассчитываются в натуральном выражении. Также, как и предыдущий метод, данная функция рассчитывает объемы незавершенного производства только для одного *единичного производственного цикла*.

Метод возвращает указатель на вновь создаваемый одномерный динамический массив рассчитанных объемов незавершенного производства в натуральном выражении размером PrCount. Тип возвращаемых значений массива определяется псевдонимом decimal.

Параметры:

PrCount - число периодов проекта; тип *size_t*;

period - номер периода проекта, в котором требуется готовый произведенный продукт в заданном количестве; тип *size_t*;

volume - требуемый объем этого продукта в заданном периоде; тип вещественного числа определяется псевдонимом decimal.

decimal *CalcWorkingVolume(const size_t PrCount, const strItem volume[]) const

Метод рассчитывает объемы незавершенного производства в каждом периоде *проекта* в зависимости от требуемого объема готового продукта в каждом периоде проекта. Объемы рассчитываются в натуральном выражении.

Метод суммирует элементы массивов с одинаковыми индексами, полученных вызовом функции CalcWorkingVolumeItem для каждого периода проекта: начиная с периода, равного значению в поле duration и до периода, равного предыдущему значению из параметра PrCount.

Метод возвращает указатель на вновь создаваемый одномерный динамический массив рассчитанных объемов незавершенного производства в натуральном выражении размером *PrCount*. Тип возвращаемых значений массива определяется псевдонимом decimal.

Параметры:

PrCount - число периодов проекта; тип *size_t*;

volume[] – указатель на массив размерности *PrCount* с объемами производства продукта по периодам (план выпуска продукта); Тип значений массива определяется псевдонимом decimal.

decimal *CalcWorkingValueItem(const size_t _PrCount, const size_t period, const decimal rmprice[], const decimal volume) const

Метод рассчитывает стоимость потребляемых ресурсов в каждом периоде *технологического цикла* в зависимости от требуемого объема готового продукта на выходе этого цикла и цен на ресурсы. Стоимость рассчитывается в денежном эквиваленте. Метод рассчитывает стоимость потребляемых ресурсов только для одного *единичного производственного цикла*.

Метод перемножает элементы массива с объемами потребляемых ресурсов в натуральном выражении, полученного с помощью вызова функции CalcRawMatVolumeItem, на элементы массива с ценами этих ресурсов, передаваемого в функцию в качестве параметра.

Метод возвращает указатель на вновь создаваемый одномерный динамический массив со стоимостью потребляемых ресурсов размером *PrCount*. Тип возвращаемых значений массива определяется псевдонимом decimal.

Параметры:

_PrCount - число периодов проекта; тип *size_t*;

period - номер периода проекта, в котором требуется готовый произведенный продукт в заданном количестве; тип *size_t*;

rmprice[] – указатель на одномерный массив, аналог двумерной матрицы, размером rcount*_*PrCount*, с ценами ресурсов; Тип значений массива определяется псевдонимом decimal;

volume - требуемый объем этого продукта в заданном периоде; тип вещественного числа определяется псевдонимом decimal.

decimal *CalcWorkingValue(const size_t _PrCount, const size_t period, const decimal rmprice[], const decimal volume) const

Метод рассчитывает *аккумулированную* стоимость потребляемых ресурсов в каждом периоде *технологического цикла* в зависимости от требуемого объема готового продукта на выходе этого цикла и цен на ресурсы. Аккумулированная стоимость рассчитывается в денежном эквиваленте. Метод рассчитывает аккумулированную стоимость потребляемых ресурсов только для одного *единичного производственного цикла*.

Метод создает и возвращает указатель на массив, элементы которого представляют собой сумму текущего и всех предыдущих элементов массива со стоимостью потребляемых ресурсов, полученных методом CalcWorkingValueItem.

Метод возвращает указатель на вновь создаваемый одномерный динамический массив с аккумулированной стоимостью потребляемых ресурсов размером *PrCount*. Тип возвращаемых значений массива определяется псевдонимом decimal.

Полученный массив де-факто представляет собой сумму стоимостей незавершенного производства и готовой продукции. Последующие методы (CalcWorkingBalanceItem и CalcProductBalanceItem) призваны разделить их между собой и вернуть нужный результат: стоимость незавершенного производства и стоимость готового продукта по отдельности.

Параметры:

_PrCount - число периодов проекта; тип *size_t*;

period - номер периода проекта, в котором требуется готовый произведенный продукт в заданном количестве; тип *size_t*;

rmprice[] – указатель на одномерный массив, аналог двумерной матрицы, размером rcount*_*PrCount*, с ценами ресурсов; Тип значений массива определяется псевдонимом decimal;

volume - требуемый объем этого продукта в заданном периоде; тип вещественного числа определяется псевдонимом decimal.

decimal *CalcWorkingBalanceItem(const size_t _PrCount, const size_t period, const decimal rmprice[], const decimal volume) const

Метод рассчитывает себестоимость незавершенного производства в каждом периоде *технологического цикла* в зависимости от требуемого объема готового продукта на выходе этого цикла и цен на ресурсы. Себестоимость рассчитываются в денежном эквиваленте. Метод рассчитывает себестоимость незавершенного производства только для одного *единичного производственного цикла*.

Метод получает массив с аккумулированной стоимостью потребляемых ресурсов (вызывая функцию CalcWorkingValue) и массив с объемами незавершенного производства в натуральном выражении (вызывая функцию CalcWorkingVolumeItem). Далее в методе обнуляются те элементы массива с аккумулированной себестоимостью ресурсов, которые равны нулю в массиве с объемами незавершенного производства и имеют тот же индекс, что и элементы первого массива. Полученный «прореженный» массив содержит себестоимость незавершенного производства и возвращается методом. Метод возвращает указатель на вновь создаваемый одномерный динамический массив с элементами, тип которых определяется псевдонимом decimal.

Параметры:

_PrCount - число периодов проекта; тип *size_t*;

period - номер периода проекта, в котором требуется готовый произведенный продукт в заданном количестве; тип *size_t*;

rmprice[] – указатель на одномерный массив, аналог двумерной матрицы, размером rcount*_*PrCount*, с ценами ресурсов; Тип значений массива определяется псевдонимом decimal;

volume - требуемый объем этого продукта в заданном периоде; тип вещественного числа определяется псевдонимом decimal.

decimal *CalcProductBalanceItem(const size_t _PrCount, const size_t period, const decimal rmprice[], const decimal volume) const

Метод рассчитывает себестоимость готового продукта на конец *технологического цикла* в зависимости от требуемого объема этого продукта на выходе этого цикла и цен на ресурсы. Себестоимость рассчитываются в денежном эквиваленте. Метод рассчитывает себестоимость готового продукта только для одного *единичного производственного цикла*.

Метод получает массив с аккумулированной стоимостью потребляемых ресурсов (вызывая функцию CalcWorkingValue) и обнуляет в нем те элементы массива, индекс которых не совпадает с заданным периодом. Полученный «прореженный» массив содержит себестоимость готового продукта и возвращается методом. Метод возвращает указатель на вновь создаваемый одномерный динамический массив с элементами, тип которых определяется псевдонимом decimal.

Параметры:

_PrCount - число периодов проекта; тип *size_t*;

period - номер периода проекта, в котором требуется готовый произведенный продукт в заданном количестве; тип *size_t*;

rmprice[] – указатель на одномерный массив, аналог двумерной матрицы, размером rcount*_*PrCount*, с ценами ресурсов; Тип значений массива определяется псевдонимом decimal;

volume - требуемый объем этого продукта в заданном периоде; тип вещественного числа определяется псевдонимом decimal.

void clsEraser()

Метод "обнуляет" все поля экземпляра класса: name = “”, measure = “”, duration = 0, rcount = 0. Указатели на динамические массивы rnames и recipeitem устанавливаются в *nullptr*. Метод используется в конструкторе перемещения.

void View_f_Item(const size_t PrCount, const strItem ProPlan[], const decimal rmprice[], const string& _hmcur, decimal* (clsRecipeItem::*f)(const size_t, const size_t, const decimal*, const decimal) const) const

Служебная функция для реализации визуального контроля. Позволяет реализовать контроль работоспособности функций: CalcWorkingValueItem, CalcWorkingValue, CalcWorkingBalanceItem и CalcProductBalanceItem. Используется функциями: ViewWorkingValueItem, ViewWorkingValue, ViewWorkingBalanceItem и ViewProductBalanceItem при отладке.

Параметры:

PrCount - число периодов проекта; тип *size_t*;

ProPlan[] - указатель на массив с планом выхода готовой продукции размерностью *PrCount*; тип элемента массива strItem;

rmprice[] – указатель на одномерный массив, аналог двумерной матрицы, размером rcount*_*PrCount*, с ценами ресурсов; тип значений массива определяется псевдонимом decimal;

&_hmcur – ссылка на наименование денежной единицы (валюты); тип *string*;

decimal* (clsRecipeItem::*f)(const size_t, const size_t, const decimal*, const decimal) const – указатель на одну из функций: CalcWorkingValueItem, CalcWorkingValue, CalcWorkingBalanceItem или CalcProductBalanceItem.

void View_f_Balance(const size_t _PrCount, const strItem ProPlan[], const decimal rmprice[], const string& _hmcur, strItem* (clsRecipeItem::*f)(const size_t, const decimal*, const strItem*) const) const

Служебная функция для реализации визуального контроля. Позволяет реализовать контроль работоспособности функций: CalcWorkingBalance и CalcProductBalance. Используется функциями: ViewWorkingBalance и ViewProductBalance при отладке.

Параметры:

_PrCount - число периодов проекта; тип *size_t*;

ProPlan[] - указатель на массив с планом выхода готовой продукции размерностью *PrCount*; тип элемента массива strItem;

rmprice[] – указатель на одномерный массив, аналог двумерной матрицы, размером rcount*_*PrCount*, с ценами ресурсов; тип значений массива определяется псевдонимом decimal;

&_hmcur – ссылка на наименование денежной единицы (валюты); тип *string*;

strItem* (clsRecipeItem::*f)(const size_t, const decimal*, const strItem*) const – указатель на одну из функций CalcWorkingBalance или CalcProductBalance.

АЛГОРИТМЫ УЧЕТА. СЕКЦИЯ PUBLIC

strItem *CalcWorkingBalance(const size_t _PrCount, const decimal rmprice[], const strItem ProPlan[]) const

Метод рассчитывает объем, удельную и полную себестоимость незавершенного производства для конкретного продукта, выпускаемого на протяжении всего проекта. Метод возвращает указатель на вновь создаваемый динамический массив типа strItem размером, равным числу периодов проекта *_PrCount*.

Метод использует функции CalcWorkingVolume, CalcWorkingBalanceItem.

Параметры:

_PrCount - число периодов проекта; тип *size_t*;

rmprice[] – указатель на одномерный массив, аналог двумерной матрицы, размером rcount*_*PrCount*, с ценами ресурсов; тип значений массива определяется псевдонимом decimal;

ProPlan[] - указатель на массив с планом выхода готовой продукции размерностью *PrCount*; тип элемента массива strItem.

strItem *CalcProductBalance(const size_t _PrCount, const decimal rmprice[], const strItem ProPlan[]) const

Метод рассчитывает объем, удельную и полную себестоимость готового продукта, выпускаемого на протяжении всего проекта. Метод возвращает указатель на вновь создаваемый динамический массив типа strItem размером, равным числу периодов проекта *_PrCount*.

Метод использует функцию CalcProductBalanceItem.

Параметры:

_PrCount - число периодов проекта; тип *size_t*;

rmprice[] – указатель на одномерный массив, аналог двумерной матрицы, размером rcount*_*PrCount*, с ценами ресурсов; тип значений массива определяется псевдонимом decimal;

ProPlan[] - указатель на массив с планом выхода готовой продукции размерностью *PrCount*; тип элемента массива strItem.

decimal* CalcRawMatVolume(const size_t PrCount, const strItem volume[]) const

Метод рассчитывает и возвращает объем потребления ресурсов в натуральном выражении для всего плана выпуска продукта в динамике, - по периодам. Метод возвращает указатель на вновь создаваемый одномерный динамический массив, аналог двумерной матрицы размером rcount * *PrCount*, строки которой представляют собой название ресурсов, а столбцы - периоды проекта. Тип элементов массива определяется псевдонимом decimal.

Параметры:

PrCount - число периодов проекта; тип *size_t*;

volume[] – указатель на массив размерности *PrCount* с объемами производства продукта по периодам (план выпуска продукта); Тип значений массива определяется псевдонимом decimal.

GET- МЕТОДЫ

strNameMeas *GetRawNamesItem() const

Метод возвращает указатель на вновь создаваемый одномерный динамический массив с наименованиями ресурсов и названиями их единиц измерения. Массив создается как *копия* внутреннего массива rnames. Размер создаваемого массива rcount. Тип элементов массива strNameMeas.

const strNameMeas* GetRefRawNamesItem() const

Метод возвращает *константный указатель* на внутренний массив с наименованиями ресурсов и названиями их единиц измерения rnames. Тип элементов массива strNameMeas.

decimal* GetRecipeitem() const

Метод возвращает указатель на вновь создаваемый одномерный динамический массив с рецептурой/ технологической картой продукта (аналог двумерной матрицы размером rcount*duration). Массив создается как *копия* внутреннего массива recipeitem. Тип элементов массива определяется псевдонимом decimal.

const decimal* GetRefRecipeitem() const

Метод возвращает *константный* указатель на внутренний массив с рецептурой/ технологической картой продукта recipeitem. Тип элементов массива определяется псевдонимом decimal.

const size_t GetDuration() const

Метод возвращает длительность производственного цикла (значение поля duration). Тип возвращаемого значения *size_t*.

const size_t GetRCount() const

Метод возвращает количество позиций ресурсов, используемых в технологическом цикле (значение поля rcount). Тип возвращаемого значения *size_t*.

const string& GetName() const

Метод возвращает ссылку на наименование продукта (ссылка на поле name). Тип возвращаемого значения ссылка на *string*.

const string& GetMeasure() const

Метод возвращает ссылку на наименование единицы измерения натурального объема продукта (ссылка на поле measure). Тип возвращаемого значения ссылка на *string*.

МЕТОДЫ СОХРАНЕНИЯ СОСТОЯНИЯ ОБЪЕКТА В ФАЙЛ И ВОССТАНОВЛЕНИЯ ИЗ ФАЙЛА

bool StF(ofstream &_outF)

Метод имплементации записи в файловый поток текущего состояния экземпляра класса (запись в файловый поток, метод сериализации). Название метода является аббревиатурой, расшифровывается “StF” как “Save to File”. При удачной сериализации возвращает *true*, при ошибке записи возвращает *false*.

Параметры:

&_outF - экземпляр класса ofstream для записи данных. При инициализации переменной _outF необходимо установить флаг бинарного режима открытия потока (не текстового!) ios::binary.

bool RfF(ifstream &_inF)

Метод имплементации чтения из файлового потока текущего состояния экземпляра класса (чтение из файлового потока, метод десериализации). Название метода является аббревиатурой, расшифровывается “RfF” как “Read from File”. При удачной десериализации возвращает *true*, при ошибке чтения возвращает *false*.

Параметры:

&_inF - экземпляр класса ifstream для чтения данных. При инициализации переменной *_inF* необходимо установить флаг бинарного режима открытия потока (не текстового!) ios::binary.

bool SaveToFile(const string _filename)

Метод записи текущего состояния экземпляра класса в файл. При удачной записи в файл, возвращает *true*, при ошибке записи возвращает *false*.

Параметры:

_filename – имя файла; тип *string*.

bool ReadFromFile(const string _filename)

Метод восстановления состояния экземпляра класса из файла. При удачном чтении из файла, возвращает *true*, при ошибке чтения возвращает *false*.

_filename – имя файла; тип *string*.

МЕТОДЫ ВИЗУАЛЬНОГО КОНТРОЛЯ

void ViewMainParametrs() const

Метод выводит в стандартный поток std::cout основные параметры класса: *название* продукта (контроль работоспособности метода GetName), *единицу измерения* продукта (контроль работоспособности метода GetMeasure), *длительность технологического цикла* (контроль работоспособности метода GetDuration), *количество позиций ресурсов* в рецептуре/ технологической карте (контроль работоспособности метода GetRCount).

void ViewRawNames() const

Метод выводит в стандартный поток std::cout основные параметры класса: *наименование ресурсов* (контроль работоспособности метода GetRawNamesItem).

void ViewRefRawNames() const

Метод выводит в стандартный поток std::cout основные параметры класса: *наименование ресурсов* (контроль работоспособности метода GetRefRawNamesItem).

void ViewRecipeitem() const

Метод выводит в стандартный поток std::cout основные параметры класса: рецептуру/ технологическую карту продукта (контроль работоспособности метода GetRecipeitem).

void ViewRefRecipeitem() const

Метод выводит в стандартный поток std::cout основные параметры класса: рецептуру/ технологическую карту продукта (контроль работоспособности метода GetRefRecipeitem).

void ViewRawMatVolume(const size_t PrCount, const strItem ProPlan[]) const

Метод выводит в стандартный поток std::cout объем потребления ресурсов в натуральном выражении для всего плана выпуска продукта в динамике, - по периодам (контроль работоспособности метода CalcRawMatVolume).

Параметры:

PrCount - число периодов проекта; тип *size_t*;

ProPlan[] – указатель на массив размерности *PrCount* с объемами производства продукта по периодам (план выпуска продукта); Тип значений массива определяется псевдонимом decimal.

void ViewWorkingValueItem(const size_t PrCount, const strItem ProPlan[], const decimal rmprice[], const string& hmcur) const

Метод выводит в стандартный поток std::cout стоимость потребляемых ресурсов в каждом периоде *технологического цикла* (контроль работоспособности метода CalcWorkingValueItem). Информация выводится по *всем циклам проекта* в виде таблицы, где строки - единичные технологические циклы, столбцы – расход в денежном эквиваленте по периоду.

Параметры:

PrCount - число периодов проекта; тип *size_t*;

ProPlan[] – указатель на массив размерности *PrCount* с объемами производства продукта по периодам (план выпуска продукта); Тип значений массива определяется псевдонимом decimal.

rmprice[] – указатель на одномерный массив, аналог двумерной матрицы, размером rcount*_*PrCount*, с ценами ресурсов; Тип значений массива определяется псевдонимом decimal;

&hmcur – ссылка на наименование денежной единицы (валюты); тип *string*.

void ViewWorkingValue(const size_t PrCount, const strItem ProPlan[], const decimal rmprice[], const string& hmcur) const

Метод выводит в стандартный поток std::cout аккумулированную стоимость потребляемых ресурсов в каждом периоде *технологического цикла* (контроль работоспособности метода CalcWorkingValue). Информация выводится по *всем циклам проекта* в виде таблицы, где строки - единичные технологические циклы, столбцы – аккумулированный расход в денежном эквиваленте по периоду.

Параметры:

PrCount - число периодов проекта; тип *size_t*;

ProPlan[] – указатель на массив размерности *PrCount* с объемами производства продукта по периодам (план выпуска продукта); Тип значений массива определяется псевдонимом decimal.

rmprice[] – указатель на одномерный массив, аналог двумерной матрицы, размером rcount*_*PrCount*, с ценами ресурсов; Тип значений массива определяется псевдонимом decimal;

&hmcur – ссылка на наименование денежной единицы (валюты); тип *string*.

void ViewWorkingBalanceItem(const size_t PrCount, const strItem ProPlan[], const decimal rmprice[], const string& hmcur) const

Метод выводит в стандартный поток std::cout себестоимость незавершенного производства в каждом периоде *технологического цикла* (контроль работоспособности метода CalcWorkingBalanceItem). Информация выводится по *всем циклам проекта* в виде таблицы, где строки - единичные технологические циклы, столбцы - незавершенное производство по периоду.

Параметры:

PrCount - число периодов проекта; тип *size_t*;

ProPlan[] – указатель на массив размерности *PrCount* с объемами производства продукта по периодам (план выпуска продукта); Тип значений массива определяется псевдонимом decimal.

rmprice[] – указатель на одномерный массив, аналог двумерной матрицы, размером rcount*_*PrCount*, с ценами ресурсов; Тип значений массива определяется псевдонимом decimal;

&hmcur – ссылка на наименование денежной единицы (валюты); тип *string*.

void ViewWorkingBalance(const size_t _PrCount, const strItem ProPlan[], const decimal rmprice[], const string& hmcur) const

Метод выводит в стандартный поток std::cout объем, удельную и полную себестоимость незавершенного производства для конкретного продукта, выпускаемого на протяжении всего проекта (контроль работоспособности метода CalcWorkingBalance). Информация выводится в виде трех таблиц: объем, удельная себестоимость, полная себестоимость. Каждая таблица имеет одну строку с названием продукта и столбцы по количеству периодов проекта.

Параметры:

_PrCount - число периодов проекта; тип *size_t*;

ProPlan[] – указатель на массив размерности *PrCount* с объемами производства продукта по периодам (план выпуска продукта); Тип значений массива определяется псевдонимом decimal.

rmprice[] – указатель на одномерный массив, аналог двумерной матрицы, размером rcount*_*PrCount*, с ценами ресурсов; Тип значений массива определяется псевдонимом decimal;

&hmcur – ссылка на наименование денежной единицы (валюты); тип *string*.

void ViewProductBalanceItem(const size_t PrCount, const strItem ProPlan[], const decimal rmprice[], const string& hmcur) const

Метод выводит в стандартный поток std::cout себестоимость готового продукта на конец *технологического цикла* (контроль работоспособности метода CalcProductBalanceItem). Информация выводится по *всем циклам проекта* в виде таблицы, где строки - единичные технологические циклы, столбцы - себестоимость готового продукта по периоду.

Параметры:

PrCount - число периодов проекта; тип *size_t*;

ProPlan[] – указатель на массив размерности *PrCount* с объемами производства продукта по периодам (план выпуска продукта); Тип значений массива определяется псевдонимом decimal.

rmprice[] – указатель на одномерный массив, аналог двумерной матрицы, размером rcount*_*PrCount*, с ценами ресурсов; Тип значений массива определяется псевдонимом decimal;

&hmcur – ссылка на наименование денежной единицы (валюты); тип *string*.

void ViewProductBalance(const size_t _PrCount, const strItem ProPlan[], const decimal rmprice[], const string& hmcur) const

Метод выводит в стандартный поток std::cout объем, удельную и полную себестоимость готового продукта, выпускаемого на протяжении всего проекта (контроль работоспособности метода CalcProductBalance). Информация выводится в виде трех таблиц: объем, удельная себестоимость, полная себестоимость. Каждая таблица имеет одну строку с названием продукта и столбцы по количеству периодов проекта.

Параметры:

_PrCount - число периодов проекта; тип *size_t*;

ProPlan[] – указатель на массив размерности *PrCount* с объемами производства продукта по периодам (план выпуска продукта); Тип значений массива определяется псевдонимом decimal.

rmprice[] – указатель на одномерный массив, аналог двумерной матрицы, размером rcount*_*PrCount*, с ценами ресурсов; Тип значений массива определяется псевдонимом decimal;

&hmcur – ссылка на наименование денежной единицы (валюты); тип *string*.

КЛАСС clsManufactItem

Если класс clsRecipeItem предоставляет основные расчетные *методы* позволяющие моделировать учетные процессы в производстве единственного продукта, то класс *clsManufactItem* инкапсулирует эти *методы*, *исходные данные* и *результаты расчетов* и представляет собой *инструмент* моделирования частичных затрат на основе переменных затрат полноценного производства для единственного наименования продукта (производства монопродукта, однотоварного производства).

К задачам, которые можно решить применением класса *clsRecipeItem*, добавляются возможности по расчету и сохранению данных:

1. Количества периодов проекта;
2. Плана производства продукта в натуральном, удельном и полном стоимостном выражении;
3. Плана поставок ресурсов в производство в натуральном выражении;
4. Цен ресурсов и график их изменения;
5. Плана незавершенного производства в натуральном, удельном и полном стоимостном выражении.

На рисунке ниже (Рисунок 4) представлена схема информационных потоков однотоварного производства.

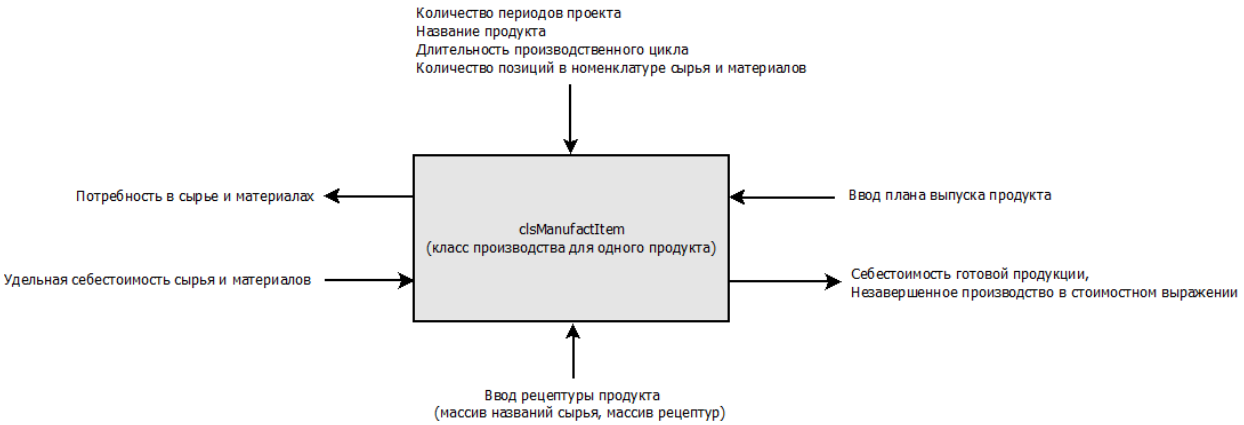

Рисунок 4 Принципиальная схема информационных потоков класса clsManufactItem

ПОЛЯ КЛАССА clsManufactItem

Таблица 13 Поля класса clsManufactItem. Секция private

Поле класса	Описание	Заметка
size_t PrCount	Тип *size_t*. Количество периодов проекта	Значение устанавливается при создании объекта класса. Может корректироваться.
const string* name	Указатель на название продукта. Тип данных *string*.	Устанавливается при создании объекта класса. В конструкторе без параметров равен nullptr. В конструкторах с параметрами указывает на одноименное поле объекта *Recipe* (рецептура/ технологическая карта продукта)
const string* measure	Указатель на название единицы измерения натурального объема продукта. Тип данных *string*.	Устанавливается при создании объекта класса. В конструкторе без параметров равен nullptr. В конструкторах с параметрами указывает на одноименное поле объекта *Recipe*
size_t duration	Тип *size_t*. Длительность производственного цикла	Устанавливается при создании объекта класса. В конструкторе без параметров равен нулю. В конструкторах с параметрами значение устанавливается при создании объекта класса как копия одноименного поля объекта *Recipe*
size_t rcount	Тип *size_t*. Количество позиций в номенклатуре ресурсов, требуемых для производства продукта	Устанавливается при создании объекта класса. В конструкторе без параметров равен нулю. В конструкторах с параметрами значение устанавливается при создании объекта класса как копия одноименного поля объекта *Recipe*
clsRecipeItem *Recipe	Указатель на рецептуру/ технологическую карту продукта. Тип объекта clsRecipeItem	Устанавливается при создании объекта класса. В конструкторе без параметров равен *nullptr*. В конструкторах с параметрами вызывает один из конструкторов класса *clsRecipeItem*
strItem *ProductPlan	Указатель на массив с планом производства продукта. Массив одномерный, размером *PrCount*, Тип элемента массива strItem	Устанавливается при создании объекта класса равным *nullptr*. В дальнейшем поля *volume* вводятся, как исходные данные, поля *price* и *value* рассчитываются. В конструкторах копирования/ перемещения принимает состояние одноименного копируемого/ перемещаемого объекта
decimal *RawMatPurchPlan	Указатель на массив с планом поставок ресурсов в натуральном выражении. Массив одномерный, аналог двумерной матрицы:	Устанавливается при создании объекта класса равным *nullptr*. В дальнейшем значения элементов массива рассчитываются. В

Оглавление >>>

	строки – ресурсы, столбцы – объемы поставок в разные периоды проекта; размер массива *rcount*PrCount*. Тип вещественного числа элемента массива определяется псевдонимом decimal	конструкторах копирования/ перемещения принимает состояние одноименного копируемого/ перемещаемого объекта
decimal *RawMatPrice	Указатель на массив с ценами ресурсов. Массив одномерный, аналог двумерной матрицы: строки – ресурсы, столбцы – цены в разные периоды проекта; размер массива *rcount*PrCount*. Тип вещественного числа элемента массива определяется псевдонимом decimal	Устанавливается при создании объекта класса равным *nullptr*. В дальнейшем значения элементов массива вводятся, как исходные данные. В конструкторах копирования/ перемещения принимает состояние одноименного копируемого/ перемещаемого объекта
strItem *Balance	Указатель на массив с планом незавершенного производства. Массив одномерный, размером *PrCount*, Тип элемента массива strItem	Устанавливается при создании объекта класса равным *nullptr*. В дальнейшем значения элементов массива рассчитываются. В конструкторах копирования/ перемещения принимает состояние одноименного копируемого/ перемещаемого объекта

Методы класса *clsManufactItem* можно условно разделить на следующие группы:

- Конструкторы, деструктор, перегруженные операторы;
- Вычислительные методы;
- Get-методы;
- Set – методы
- Методы сохранения состояния объекта в файл и восстановления из файла;
- Методы визуального контроля

Ниже подробно описываются все методы.

КОНСТРУКТОРЫ, ДЕСТРУКТОР, ПЕРЕГРУЖЕННЫЕ ОПЕРАТОРЫ

clsManufactItem()

Конструктор по умолчанию (без параметров). Создает и инициализирует переменные PrCount = duration = rcount = 0; указатели на name и measure устанавливаются в nullptr; объект Recipe и динамические массивы ProductPlan, RawMatPurchPlan, RawMatPrice и Balance не создаются, указатели на них устанавливаются в *nullptr*.

clsManufactItem(const size_t _PrCount, const clsRecipeItem &obj)

Конструктор с вводом длительности проекта и рецептуры/ технологической карты.

Параметры:

_PrCount – число периодов проекта; устанавливает значение поля PrCount; тип *size_t*;

&obj – рецептура/ технологическая карта продукта; ссылка на экземпляр класса clsRecipeItem. Объект *obj* копируется в поле Recipe.

Поле PrCount инициализируется значением из параметра *_PrCount*; поля *Recipe*, *name*, *measure*, *duration*, *rcount* инициализируются значениями из *obj*; динамические массивы ProductPlan, RawMatPurchPlan, RawMatPrice и Balance не создаются, указатели на них устанавливаются в *nullptr*.

clsManufactItem(const size_t _PrCount, clsRecipeItem &&obj)

Конструктор с вводом длительности проекта и рецептуры/ технологической карты.

Параметры:

_PrCount – число периодов проекта; устанавливает значение поля PrCount; тип *size_t*;

&obj – рецептура/ технологическая карта продукта; rvalue-ссылка на экземпляр класса clsRecipeItem. Объект obj перемещается в поле Recipe.

Поле PrCount инициализируется значением из параметра *_PrCount*; поля *Recipe*, *name*, *measure*, *duration*, *rcount* инициализируются значениями из *obj*; динамические массивы ProductPlan, RawMatPurchPlan, RawMatPrice и Balance не создаются, указатели на них устанавливаются в *nullptr*.

clsManufactItem(const size_t _PrCount, const string &_name, const string &_measure, const size_t _duration, const size_t _rcount, const strNameMeas _rnames[], const decimal _recipeitem[])

Конструктор с параметрами, создает внутри себя экземпляр класса рецептур.

Параметры:

_PrCount – число периодов проекта; устанавливает значение поля PrCount; тип *size_t*;

&_name – ссылка на название продукта; используется как параметр в конструкторе класса clsRecipeItem для создания объекта в поле Recipe и устанавливает указатель name на одноименное поле объекта *Recipe*; тип *string*;

&_measure – ссылка на наименование единицы измерения натурального объема продукта; используется как параметр в конструкторе класса clsRecipeItem для создания объекта в поле Recipe и устанавливает указатель measure на одноименное поле объекта *Recipe*; тип *string*;

_duration – длительность производственного цикла; используется как параметр в конструкторе класса clsRecipeItem для создания объекта в поле Recipe и устанавливает значение в поле duration; тип *size_t*;

_rcount – количество позиций ресурсов, требуемых для производства продукта; используется как параметр в конструкторе класса clsRecipeItem для создания объекта в поле Recipe и устанавливает значение поля rcount; тип *size_t*;

_rnames[] – указатель на массив с наименованиями ресурсов и единицами их измерения размером *_rcount*; используется как параметр в конструкторе класса clsRecipeItem для создания объекта в поле Recipe; тип элемента массива strNameMeas;

_recipeitem[] – указатель на массив с рецептурами (одномерный аналог матрицы размерностью *_rcount* _duration*); используется как параметр в конструкторе класса clsRecipeItem для создания объекта в поле Recipe; вещественный тип элемента массива определяется псевдонимом decimal.

clsManufactItem(const clsManufactItem &obj)

Конструктор копирования.

Параметры:

&obj – ссылка на копируемый константный экземпляр класса *clsManufactItem*.

clsManufactItem(clsManufactItem &&obj)

Конструктор перемещения.

Параметры:

&&obj - перемещаемый экземпляр класса *clsManufactItem*.

clsManufactItem &operator= (const clsManufactItem &obj)

Перегрузка оператора присваивания копированием

Параметры:

&obj – ссылка на копируемый константный экземпляр класса *clsManufactItem*.

clsManufactItem &operator= (clsManufactItem &&obj)

Перегрузка оператора присваивания перемещением

Параметры:

&&obj - перемещаемый экземпляр класса *clsManufactItem*.

bool operator == (const string &Rightname) const

Переопределение оператора сравнения для поиска экземпляра объекта по наименованию продукта

Параметры:

&Rightname – ссылка на строку символов, с которой сравнивается значение, на которое указывает поле name экземпляра класса.

~clsManufactItem()

Деструктор.

void swap(clsManufactItem& obj) noexcept

Функция обмена значениями между объектами типа *clsManufactItem*.

Параметры:

&obj – ссылка на обмениваемый экземпляр класса *clsManufactItem*.

АЛГОРИТМЫ УЧЕТА. СЕКЦИЯ PRIVATE

void clsEraser()

Метод "обнуляет" все поля экземпляра класса: PrCount = duration = rcount = 0, устанавливает указатели name и measure = *nullptr*. Устанавливает указатели Recipe, ProductPlan, RawMatPurchPlan, RawMatPrice и Balance, в *nullptr*. Метод используется в конструкторе перемещения.

АЛГОРИТМЫ УЧЕТА. СЕКЦИЯ PUBLIC

void CalcRawMatPurchPlan()

Метод вызывает функцию clsRecipeItem::CalcRawMatVolume объекта из поля Recipe и рассчитывает объем потребления ресурсов в натуральном выражении для всего плана выпуска продукта и создает и заполняет массив RawMatPurchPlan.

bool CalculateItem()

Метод вызывает функции clsRecipeItem::CalcWorkingBalance и clsRecipeItem::CalcProductBalance объекта из поля Recipe и рассчитывает объем, удельную и полную себестоимость незавершенного производства и готовой продукции для конкретного продукта, выпускаемого на протяжении всего проекта. Если вычисления проведены без ошибок, метод возвращает *true*, в противном случае – *false*.

GET-МЕТОДЫ

const size_t& GetPrCount() const

Возвращает const-ссылку на количество периодов проекта. Возвращается ссылка на тип *size_t*.

const size_t& GetRCount() const

Возвращает const-ссылку на количество позиций сырья, участвующего в производстве. Возвращается ссылка на тип *size_t*.

const size_t& GetDuration() const

Возвращает const-ссылку на длительность производственного цикла. Возвращается ссылка на тип *size_t*.

const string* GetName() const

Возвращает const-указатель на наименование продукта. Возвращает указатель на тип *string*.

const string* GetMeasure() const

Возвращает const-указатель на наименование единицы измерения натурального объема продукта. Возвращает указатель на тип *string*.

const decimal* GetRefRecipe() const

Метод вызывает функцию clsRecipeItem::GetRefRecipeitem объекта из поля Recipe и возвращает const-указатель на внутренний массив с рецептурами объекта из этого поля. Вещественный тип элементов массива определяется псевдонимом decimal.

const strNameMeas* GetRefRawNames() const

Метод вызывает функцию clsRecipeItem::GetRefRawNamesItem объекта из поля Recipe и возвращает const-указатель на внутренний массив с наименованиями ресурсов и единицами их натурального измерения объекта из этого поля. Тип элементов массива strNameMeas.

const decimal* GetRawMatPurchPlan() const

Метод возвращает константный указатель на массив RawMatPurchPlan с объемом потребления ресурсов в натуральном выражении для всего плана выпуска продукта. Вещественный тип элементов массива определяется псевдонимом decimal.

const decimal* GetRawMatPrice() const

Метод возвращает константный указатель на массив RawMatPrice с ценами на ресурсы. Вещественный тип элементов массива определяется псевдонимом decimal.

const strItem* GetBalance() const

Метод возвращает константный указатель на массив Balance с объемом, удельной и полной себестоимостью незавершенного производства для конкретного продукта, выпускаемого на протяжении всего проекта. Если массив не инициирован расчетными данными, то возвращает *nullptr*. Тип элемента массива strItem.

const strItem* GetProductPlan() const

Метод возвращает константный указатель на массив ProductPlan с объемом, удельной и полной себестоимостью готовой продукции для конкретного продукта, выпускаемого на протяжении всего проекта. Если массив не инициирован исходными данными, то возвращает *nullptr*. Если в массиве отсутствуют расчетные данные, то возвращает только натуральные объемы (значения полей strItem.volume). Тип элемента массива strItem.

SET – МЕТОДЫ

bool SetProductPlan(const strItem _ProductPlan[])

Метод ввода плана выпуска продукта (объем выпуска в натуральном выражении и график выпуска).

Параметры:

_ProductPlan[] – указатель на массив с планом выпуска продукта в натуральном выражении. Тип элемента массива strItem. В каждом элементе массива полезной информацией заполнены поля *volume*; поля *price* и *value* содержат нули. В случае удачного завершения работы метода, возвращается *true*, иначе – возвращается *false*.

bool SetRawMatPrice(const decimal _Price[])

Метод ввода цен на ресурсы. Предполагается, что после получения складом ресурсов информации о потребности в ресурсах (с помощью метода GetRawMatPurchPlan), склад возвращает информацию об учетных ценах на эти ресурсы. Эта информация с помощью данного метода *копируется* в массив RawMatPrice размером rcount*PrCount. В случае удачного завершения работы метода, возвращается *true*, иначе – возвращается *false*.

Параметры:

_Price[] – указатель на *копируемый* массив с ценами ресурсов; массив является одномерным аналогом двумерной матрицы, размером *rcount***PrCount*. Вещественный тип элемента массива определяется псевдонимом decimal.

bool MoveRawMatPrice(decimal _Price[])

Метод ввода цен на ресурсы. Предполагается, что после получения складом ресурсов информации о потребности в ресурсах (с помощью метода GetRawMatPurchPlan), склад возвращает информацию об учетных ценах на эти ресурсы. Эта информация с помощью данного метода *перемещается* в массив RawMatPrice размером rcount*PrCount. В случае удачного завершения работы метода, возвращается *true*, иначе – возвращается *false*.

Параметры:

_Price[] – указатель на *перемещаемый* массив с ценами ресурсов; массив является одномерным аналогом двумерной матрицы, размером *rcount***PrCount*. Вещественный тип элемента массива определяется псевдонимом decimal.

МЕТОДЫ СОХРАНЕНИЯ СОСТОЯНИЯ ОБЪЕКТА В ФАЙЛ И ВОССТАНОВЛЕНИЯ ИЗ ФАЙЛА

bool StF(ofstream &_outF)

Метод имплементации записи в файловый поток текущего состояния экземпляра класса (запись в файловый поток, метод сериализации). Название метода является аббревиатурой, расшифровывается “StF” как “Save to File”. При удачной сериализации возвращает *true*, при ошибке записи возвращает *false*.

Параметры:

&_outF - экземпляр класса ofstream для записи данных. При инициализации переменной *_outF* необходимо установить флаг бинарного режима открытия потока (не текстового!) ios::binary.

bool RfF(ifstream &_inF)

Метод имплементации чтения из файлового потока текущего состояния экземпляра класса (чтение из файлового потока, метод десериализации). Название метода является аббревиатурой, расшифровывается “RfF” как “Read from File”. При удачной десериализации возвращает *true*, при ошибке чтения возвращает *false*.

Параметры:

&_inF - экземпляр класса ifstream для чтения данных. При инициализации переменной *_inF* необходимо установить флаг бинарного режима открытия потока (не текстового!) ios::binary.

bool SaveToFile(const string _filename)

Метод записи текущего состояния экземпляра класса в файл. При удачной записи в файл, возвращает *true*, при ошибке записи возвращает *false*.

Параметры:

_filename – имя файла; тип *string*.

bool ReadFromFile(const string _filename)

Метод восстановления состояния экземпляра класса из файла. При удачном чтении из файла, возвращает *true*, при ошибке чтения возвращает *false*.

_filename – имя файла; тип *string*.

МЕТОДЫ ВИЗУАЛЬНОГО КОНТРОЛЯ

void ViewMainParametrs() const

Метод выводит в стандартный поток std::cout основные параметры класса: *длительность* проекта (контроль работоспособности метода GetPrCount), *название* продукта (контроль работоспособности метода GetName), *единицу измерения* натурального объема продукта (контроль работоспособности метода GetMeasure), *длительность технологического цикла* (контроль работоспособности метода GetDuration), *количество позиций ресурсов* в рецептуре/ технологической карте (контроль работоспособности метода GetRCount).

void ViewRefRecipe() const

Метод выводит в стандартный поток std::cout рецептуру/ технологическую карту продукта (контроль работоспособности методов GetRefRecipe и GetRefRawNames). Метод использует функцию вывода информации в формате таблицы ArrPrint.

void ViewRawMatPurchPlan() const

Метод выводит в стандартный поток std::cout *потребность в ресурсах* для продукта (контроль работоспособности методов GetRCount, GetRefRawNames, GetRawMatPurchPlan, GetPrCount). Перед вызовом данного метода необходимо провести вычисления методом CalcRawMatPurchPlan, иначе метод *GetRawMatPurchPlan* вернет *nullptr*. Метод использует функцию вывода информации в формате таблицы ArrPrint.

void ViewRawMatPrice(const string& hmcur) const

Метод выводит в стандартный поток std::cout *цены на ресурсы* для продукта (контроль работоспособности методов GetRCount, GetRefRawNames, GetRawMatPrice, GetPrCount). Перед вызовом данного метода необходимо ввести цены на ресурсы методом SetRawMatPrice, иначе метод *GetRawMatPrice* вернет *nullptr*. Метод использует функцию вывода информации в формате таблицы ArrPrint.

Параметры:

&hmcur – ссылка на название денежной единицы. Тип *string*.

void ViewCalculate(const string& hmcur) const

Метод выводит в стандартный поток std::cout *незавершенное производство* в *натуральном*, *удельном* и *полном стоимостном* выражении (контроль работоспособности метода GetBalance), а также *готовую продукцию* в *натуральном*, *удельном* и *полном стоимостном* выражении (контроль работоспособности метода GetProductPlan). Перед вызовом данного метода необходимо:

- Создать объект класса и ввести длительность проекта, рецептуру/ технологическую карту, план выпуска продукта (например, используя соответствующий конструктор и метод SetProductPlan);
- Рассчитать потребность в ресурсах для производства продукта (используя метод CalcRawMatPurchPlan);

- Ввести цены на ресурсы (используя метод SetRawMatPrice);
- Провести расчет незавершенного производства и готовой продукции (используя метод CalculateItem).

Если пропустить какой-либо из перечисленных выше пунктов, массивы Balance и ProductPlan будут не заполнены расчетными данными, и методы GetBalance и GetProductPlan вернут некорректные данные. Таким образом, данный метод является финализирующим в цепочке визуального контроля работоспособности методов класса.

Метод использует функцию вывода информации в формате таблицы ArrPrint.

Параметры:

&hmcur – ссылка на название денежной единицы. Тип *string*.

ПРИМЕР ПРОГРАММЫ С КЛАССОМ clsManufactItem

Ниже представлен листинг программы, демонстрирующей работу класса clsManufactItem.

```
#include <iostream>
#include <Manufact_module.h>

using namespace std;

int main() {

    setlocale(LC_ALL, "Russian");       // Установка русского языка для вывода

    /** Основные параметры проекта **/
        const size_t PrCount = 9;       // Количество периодов проекта
        const string Cur = "RUR";       // Домашняя валюта проекта

    /** Продукт **/
        const string Name_00 = "Фрикасе из курицы с зеленым горошком";
        const string Meas_00 = "шт.";

    /** План отгрузок готовой продукции **/

    const strItem PPlan[PrCount] {  {0, 0, 0},  {5003, 0, 0},  {8482, 0, 0},
                                {11825, 0, 0}, {15137, 0, 0}, {18428, 0, 0 },
                                {21706, 0, 0}, {24975, 0, 0}, {28239, 0, 0}};

    /** Исходные данные для рецептуры продукта **/
        const size_t rcount_00  = 9;    // Число наименований ресурсов
        const size_t duratio_00 = 1;    // Длительность производственного цикла

        const strNameMeas rnames_00[rcount_00]  // Номенклатура ресурсов
            {{"Горошек зеленый с/м", "кг."}, {"Итальянские травы", "кг."},
             {"Курица бедро", "кг."}, {"Молоко", "кг."},\
             {"Морковь", "кг."}, {"Мука пшеничная", "кг."}, {"Соль", "кг."},
             {"Упаковка", "шт."}, {"Чеснок", "кг."}};

        const decimal recipe_00[rcount_00*duratio_00]  {0.1200,
                                                        0.0012,
                                                        0.2700,
                                                        0.1200,
                                                        0.1320,
                                                        0.0120,
                                                        0.0042,
                                                        1.0000,
                                                        0.0024 };
    /** Создаем рецептуру продукта **/
    clsRecipeItem* rcp_00 =
        new clsRecipeItem(Name_00, Meas_00, duratio_00, rcount_00,
        rnames_00, recipe_00);

    /** Создаем единицчное производство **/
    clsManufactItem* man_00 = new clsManufactItem(PrCount, move(*rcp_00));
    man_00->SetProductPlan(PPlan);  // Вводим план отгрузок готовой продукции
    man_00->CalcRawMatPurchPlan();  // Рассчитываем план потребления ресурсов

   /** Отдаем на склад ресурсов план потребления сырья и материалов
(например,указатель на массив RMPlan, полученный так: const decimal* RMPlan =
man_00->GetRawMatPurchPlan()), и получаем от склада данные о себестоимости
необходимых нам ресурсов. Например, склад вернул нам следующий массив: **/

const decimal RMRpice[rcount_00*PrCount] {
 100.00,  101.00,  102.01,  103.03,  104.06,  105.10,  106.15,  107.21,  108.28,
4062.50, 4103.13, 4144.16, 4185.60, 4227.45, 4269.73, 4312.43, 4355.55, 4399.11,
 225.00,  227.25,  229.52,  231.82,  234.14,  236.48,  238.84,  241.23,  243.64,
  39.00,   39.39,   39.78,   40.18,   40.58,   40.99,   41.40,   41.81,   42.23,
  15.00,   15.15,   15.30,   15.45,   15.61,   15.77,   15.92,   16.08,   16.24,
  41.00,   41.41,   41.82,   42.24,   42.66,   43.09,   43.52,   43.96,   44.40,
  19.00,   19.19,   19.38,   19.58,   19.77,   19.97,   20.17,   20.37,   20.57,
  20.00,   20.20,   20.40,   20.61,   20.81,   21.02,   21.23,   21.44,   21.66,
 105.00,  106.05,  107.11,  108.18,  109.26,  110.36,  111.46,  112.57, 113.70};

if(man_00->SetRawMatPrice(RMRpice));    // Вводим цены ресурсов в производство
```

```
/** Рассчитываем учетную себестоимость готовой продукции **/
    if(!man_00->CalculateItem()) cout << "Ошибка расчета" << endl;
    /** Выводим на экран информаицю о готовом продукте и незавершенном производстве **/
    man_00->ViewCalculate(Cur);

delete rcp_00;
    delete man_00;
    return 0;
}
```

Программа выводит на экран информацию о незавершенном производстве и готовом продукте.

```
Контроль работы методов GetBalance и GetProductPlan

Незавершенное производство в натуральном, удельном и полном стоимостном выражении

By volume
Name            Measure              0          1          2          3          4          5          6          7          8
Фрикасе из кури шт.               0.00       0.00       0.00       0.00       0.00       0.00       0.00       0.00       0.00


By price
Name            Measure              0          1          2          3          4          5          6          7          8
Фрикасе из кури RUR/шт.           0.00       0.00       0.00       0.00       0.00       0.00       0.00       0.00       0.00


By value
Name            Measure              0          1          2          3          4          5          6          7          8
Фрикасе из кури RUR               0.00       0.00       0.00       0.00       0.00       0.00       0.00       0.00       0.00

Готовое производство в натуральном, удельном и полном стоимостном выражении

By volume
Name            Measure              0          1          2          3          4          5          6          7          8
Фрикасе из кури шт.               0.00    5003.00    8482.00   11825.00   15137.00   18428.00   21706.00   24975.00   28239.00


By price
Name            Measure              0          1          2          3          4          5          6          7          8
Фрикасе из кури RUR/шт.           0.00     106.16     107.22     108.30     109.38     110.47     111.57     112.69     113.82


By value
Name            Measure              0          1          2          3          4          5          6          7          8
Фрикасе из кури RUR               0.00  531117.95  909423.86 1280617.56 1655612.98 2035769.43 2421816.87 2814356.88 3214125.87


Process returned 0 (0x0)   execution time : 0.144 s
Press any key to continue.
```

Поскольку длительность технологического цикла равна одному периоду, незавершенное производство отсутствует.

КЛАСС clsManufactory

Класс *clsManufactory* предназначен для моделирования учетных процессов на производстве. Экземпляр класса *clsManufactory* описывает полностью работоспособное производство, выпускающее множество номенклатурных позиций готовой продукции. Класс представляет собой контейнер для нескольких экземпляров класса clsManufactItem (однотоварных производств).

Также как для класса *clsManufactItem*, применением класса *clsManufactory* решаются задачи:

1. Рассчитать потребность в разных ресурсах и график обеспечения ими для производства множества номенклатурных позиций готовой продукции;
2. Принять учетные цены всех необходимых производству ресурсов и график их изменения;
3. Рассчитать объем, полную и удельную ресурсную себестоимость незавершенного производства и готовой продукции множества номенклатурных позиций.

ПОЛЯ КЛАССА clsManufactory

Таблица 14 Поля класса clsManufactory. Секция Private

Поле класса	Описание	Заметка
size_t PrCount	Тип *size_t*. Количество периодов проекта	Значение устанавливается при создании объекта класса. Может корректироваться. В случае

		отличия одноименных полей элементов контейнера (объектов clsManufactItem), их значения устанавливаются равными данной величие.
size_t RMCount	Тип *size_t*. Полное количество позиций в номенклатуре ресурсов	Значение устанавливается при создании объекта класса. Может корректироваться.
Currency hmcur	Тип Currency. Домашняя (основная) валюта.	Значение устанавливается при создании объекта класса. По умолчанию принимает значение RUR (индекс 0) и соответствует валюте «Российский рубль».
strNameMeas *RMNames	Тип strNameMeas. Полный массив с названиями ресурсов и наименованиями их единиц измерения. Размерность *RMCount*	Этот массив в отличие от массивов clsRecipeItem::*rnames содержит полный список всех позиций сырья и материалов. Соответственно, *RMCount* >= clsRecipeItem::rcount.
vector <clsManufactItem> Manuf	Тип vector<clsManufactItem>. Контейнер для множества экземпляров класса clsManufactItem (однотоварных производств)	Статическая переменная
clsProgress_shell<type_progress>* pshell{nullptr}	Тип указатель на класс clsProgress_shell. Адрес внешнего объекта-оболочки для индикатора прогресса. Тип самого индикатора определяется псевдонимом type_progress.	При создании объекта класса устанавливается значение *nullptr*: по умолчанию индикация прогресса не осуществляется

Методы класса clsManufactory можно условно разделить на следующие группы:

- Конструкторы, деструктор, перегруженные операторы;
- Set-методы;
- Сервисные методы;
- Расчетные методы;
- Get-методы (методы в секции Private и методы в секции Public);
- Методы сохранения состояния объекта в файл и восстановления из файла;
- Методы визуального контроля

Ниже подробно описываются все методы

КОНСТРУКТОРЫ, ДЕСТРУКТОР, ПЕРЕГРУЖЕННЫЕ ОПЕРАТОРЫ

clsManufactory()

Конструктор по умолчанию (без параметров). Создает и инициализирует переменные: PrCount = RMCount = 0; hmcur = RUR; создает на стеке пустой контейнер Manuf; Устанавливает указатель RMNames = *nullptr*.

clsManufactory(const size_t _PrCount, const size_t _RMCount, const strNameMeas _RMNames[], const size_t msize)

Конструктор с вводом длительности проекта, массива с наименованиями и единицами измерения ресурсов и резервированием памяти для заданного числа однотоварных производств.

Параметры:

_PrCount – количество периодов проекта; устанавливает значение поля PrCount; тип *size_t*;

_RMCount – количество номенклатурных позиций ресурсов, используемых в производстве; устанавливает значение поля RMCount; тип *size_t*;

_RMNames[] – указатель на массив с наименованиями и единицами измерения ресурсов, используемых в производстве; устанавливает значение поля RMNames; тип strNameMeas;

msize – предварительное количество однотоварных производств (объектов типа clsManufactItem) в контейнере.

clsManufactory(const clsManufactory &obj)

Конструктор копирования.

Параметры:

&obj – ссылка на копируемый константный экземпляр класса *clsManufactory*.

clsManufactory(clsManufactory &&obj)

Конструктор перемещения.

Параметры:

&&obj – ссылка на перемещаемый константный экземпляр класса *clsManufactory*.

void swap(clsManufactory& obj) noexcept

Функция обмена значениями между экземплярами класса.

Параметры:

&obj – ссылка на обмениваемый экземпляр класса *clsManufactory*.

clsManufactory& operator=(const clsManufactory &obj)

Перегрузка оператора присваивания копированием.

Параметры:

&obj – ссылка на копируемый константный экземпляр класса *clsManufactory*.

clsManufactory& operator=(clsManufactory&&)

Перегрузка оператора присваивания перемещением

Параметры:

&&obj - перемещаемый экземпляр класса *clsManufactory*.

~clsManufactory()

Деструктор.

SET-МЕТОДЫ.

Все Set-методы класса находятся в секции Public.

void Set_progress_shell(clsProgress_shell<type_progress>* val)

Метод присваивает указателю pshell адрес внешнего объекта-оболочки индикатора прогресса. Тип оболочки clsProgress_shell, тип индикатора прогресса определяется псевдонимом type_progress.

Параметры:

*val – указатель на внешний объект-оболочку для класса индикатора прогресса.

void SetCurrency(const Currency & _cur)

Метод устанавливает основную валюту проекта.

Параметры:

& _cur – ссылка на основную валюту проекта. Тип Currency.

bool SetManufItem(const string &_name, const string &_measure, const size_t _duration, const size_t _rcount, const strNameMeas _rnames[], const decimal _recipe[], const strItem _pplan[])

Метод создания производства для конкретного продукта. Метод проверяет совпадение имен нового и имеющихся в контейнере продуктов; если наименование продукта уникально (нет совпадений), то создает новый экземпляр класса clsManufactItem непосредственно в контейнере и возвращает *true*; иначе производство не создается, и метод возвращает *false*. Метод также возвращает *false*, если длительность технологического цикла или количество позиций ресурсов равно нулю или же отсутствуют массивы с наименованиями ресурсов, рецептура/технологическая карта или план отгрузок готовой продукции.

Параметры:

&_name – наименование продукта; тип *string*;

&_measure – единица измерения натурального объема продукта; тип тип *string*;

_duration – длительность технологического цикла; тип *size_t*;

_rcount – количество позиций ресурсов, требуемых для производства продукта; тип *size_t*;

_rnames[] – указатель на массив с именами и единицами измерения готового продукта, размерностью

rcount; тип элемента массива strNameMeas;

_recipe[] – указатель на одномерный массив с рецептурой, являющийся аналогом двумерной матрицы, размером *rcount***duration*;

_pplan[] – указатель на массив с планом отгрузки готового продукта, размером PrCount. Тип элемента массива strItem.

Метод проверяет корректность параметров и вызывает непосредственно в контейнере Manuf конструктор с параметрами класса *clsManufactItem*. Следом за ним вызывается метод clsManufactItem::SetProductPlan для ввода плана отгрузки готового продукта.

Метод позволяет создавать производство при отсутствии плана отгрузки *_pplan[]* = *nullptr*.

bool SetManufItem(const clsRecipeItem &obj, const strItem _pplan[])

Метод имеет назначение, что и предыдущий. В методе также проверяется совпадение имени продукта из передаваемого в качестве параметра объекта типа clsRecipeItem и имен имеющихся в контейнере продуктов; если наименование продукта уникально (нет совпадений), то метод создает новый экземпляр класса clsManufactItem непосредственно в контейнере и возвращает *true*; иначе производство не создается, и метод возвращает *false*.. Метод вызывает соответствующий конструктор класса *clsManufactItem* непосредственно в контейнере. Следом за ним вызывается метод clsManufactItem::SetProductPlan для ввода плана отгрузки готового продукта.

Параметры:

&obj – копируемый объект рецептура/технологическая карта типа *clsRecipeItem*;

_pplan[] – указатель на массив с планом отгрузки готового продукта, размером PrCount. Тип элемента массива strItem.

Метод позволяет создавать производство при отсутствии плана отгрузки *_pplan[]* = *nullptr*.

bool SetManufItem(clsRecipeItem &&obj, const strItem _pplan[])

Метод, аналогичный предыдущему, но объект рецептура/технологическая карта перемещается в конструкторе класса *clsManufactItem*.

&&obj – перемещаемый объект рецептура/технологическая карта типа *clsRecipeItem*;

_pplan[] – указатель на массив с планом отгрузки готового продукта, размером PrCount. Тип элемента массива strItem.

Метод позволяет создавать производство при отсутствии плана отгрузки *_pplan[]* = *nullptr*.

bool SetManufItem(const clsManufactItem &obj)

Метод имеет назначение, что и предыдущие (создание производства для конкретного продукта). В методе также проверяется совпадение имени продукта из передаваемого в качестве параметра объекта типа clsManufactItem и имен имеющихся в контейнере продуктов. Если наименование продукта уникально (нет совпадений), то метод создает новый экземпляр класса clsManufactItem непосредственно в контейнере и возвращает *true*; иначе производство не создается, и метод возвращает *false*.

Параметры:

&obj – копируемый экземпляр класса *clsManufactItem*.

bool SetManufItem(clsManufactItem &&obj)

Аналогичен предыдущему методу, но объект-параметр типа clsManufactItem перемещается в контейнер.

Параметры:

&&obj – перемещаемый экземпляр класса *clsManufactItem*.

bool SetProdPlan(const strItem _ProdPlan[])

Метод вводит план отгрузки для всех продуктов.

Параметры:

_ProdPlan[] указатель на одномерный массив с планом отгрузок *всех* продуктов, являющийся аналогом двумерной матрицы размером Manuf.size()*PrCount. Тип элемента массива strItem. При благополучном завершении операции возвращает *true*.

bool SetRawMatPrice(const decimal _Price[])

Несмотря на формальное отнесение данного метода к Set-методам, он по сути является *вычислительным*. Метод вводит цены на ресурсы в экземпляр класса «производство».

Параметры:

_Price[] – указатель на массив с ценами ресурсов; массив одномерный, аналог двумерной матрицы размером RMCount * PrCount. Вещественный тип элемента массива определяется псевдонимом decimal. При благополучном завершении операции возвращает *true*.

СЕРВИСНЫЕ МЕТОДЫ

bool IncreaseCapacity(size_t msize)

Новые производства добавляются в контейнер без ограничений по количеству. Однако, чем больше элементов уже содержит контейнер, тем больше ресурсов и времени затрачивается на добавление новых. Это связано с особенностью контейнеров типа vector. Для повышения эффективности работы класса *clsManufactory* в конструкторе с параметрами предусмотрена возможность резервирования числа однотоварных производств.

Оглавление >>>

Это наилучшее с точки зрения эффективности решение. Другой возможностью является возможность увеличить «количество вакантных мест» для новых производств, если экземпляр класса *clsManufactory* создавался без резервирования.

Данный метод увеличивает память вектора для эффективного добавления новых элементов. Однако сам метод вызывает копирование и удаление всех элементов вектора, что весьма затратно. Лучше использовать этот метод в самом начале создания мульти-товарного производства и потом не менять размер вектора.

Параметры:

msize – новый размер резервируемой памяти контейнера (новое число элементов типа clsManufactItem в контейнере: занятых и вакантных в сумме).

ВЫЧИСЛИТЕЛЬНЫЕ МЕТОДЫ

Void CalcRawMatPurchPlan(size_t bg, size_t en)

Метод рассчитывает объем потребления ресурсов в натуральном выражении в соответствии с планом выпуска для каждого продукта (каждого однотоварного производства из контейнера) в диапазоне: от продукта с индексом bg до продукта с индексом (en-1).

Параметры:

Bg – нижняя граница диапазона продуктов (равен начальному индексу элемента контейнера из диапазона); тип *size_t*;

En – верхняя граница диапазона продуктов (равен следующему за конечным индексом диапазона индексу элемента контейнера); *size_t*;

Метод поочередно вызывает методы clsManufactItem::CalcRawMatPurchPlan у каждого элемента контейнера в диапазоне индексов от *bg* до (*en-1*) включительно и формирует массивы clsManufactItem::RawMatPurchPlan.

void CalcRawMatPurchPlan()

Метод, аналогичный предыдущему, обрабатывает *все* элементы контейнера. Вызывает метод CalcRawMatPurchPlan(size_t bg, size_t en) с параметрами *bg* = 0, *en* = Manuf.size()-1).

void CalcRawMatPurchPlan_future()

Метод является альтернативой методу CalcRawMatPurchPlan. В методе *CalcRawMatPurchPlan_future* расчеты производятся в параллельных потоках, число которых на единицу меньше числа ядер используемого компьютера, и число обрабатываемых элементов контейнера в каждом потоке примерно одинаково. В каждом потоке вызывается метод CalcRawMatPurchPlan(size_t bg, size_t en) с границами диапазона, предназначенного для обработки в этом потоке. В каждом отдельном потоке расчеты производятся последовательно, но сами потоки выполняются параллельно. Это позволяет существенно ускорить вычисления. Для организации потоков используются инструменты из стандартного заголовочного файла <future>.

void CalcRawMatPurchPlan_thread()

Этот метод идентичен по назначению и своей структуре предыдущему методу. Отличие в том, что в нем используются инструменты из стандартного заголовочного файла <thread>.

Выбор метода из приведенных выше (*CalcRawMatPurchPlan*, *CalcRawMatPurchPlan_future* или *CalcRawMatPurchPlan_thread*) остается за пользователем. Рекомендации те же, что и при выборе методов расчета в классе clsStorage.

bool Calculate(size_t bg, size_t en)

Метод используется после ввода в экземпляр класса *clsManufactory* данных о ценах на ресурсы, которые используются в производстве продуктов (метод SetRawMatPrice) и рассчитывает объем, удельную и полную себестоимость незавершенного производства и готовой продукции для продуктов с индексами в диапазоне от bg до en-1, выпускаемых на протяжении всего проекта. Метод поочередно вызывает методы clsManufactItem::CalculateItem для каждого единичного производства; формирует массивы продуктов и баланса незавершенногопроизводства для каждого продукта из заданного диапазона. Метод возвращает *true*, если в контейнере есть элементы и вычисления во всех элементах диапазона вернули *true*; иначе метод возвращает *false*.

Параметры:

Bg – нижняя граница диапазона продуктов (равен начальному индексу элемента контейнера из диапазона); тип *size_t*;

En – верхняя граница диапазона продуктов (равен следующему за конечным индексом диапазона индексу элемента контейнера); *size_t*;

bool Calculate()

Метод, аналогичный предыдущему, обрабатывает *все* элементы контейнера. Вызывает метод Calculate(size_t bg, size_t en) с параметрами *bg* = 0, *en* = Manuf.size()-1). Метод возвращает *true*, если в контейнере есть элементы и вычисления во всех элементах контейнера вернули *true*; иначе метод возвращает *false*.

bool Calculate_future()

Метод является альтернативой методу Calculate. В методе *Calculate_future* расчеты производятся в параллельных потоках, число которых на единицу меньше числа ядер используемого компьютера, и число обрабатываемых элементов контейнера в каждом потоке примерно одинаково. В каждом потоке вызывается метод Calculate(size_t bg, size_t en) с границами диапазона, предназначенного для обработки в этом потоке. В каждом отдельном потоке расчеты производятся последовательно, но сами потоки выполняются параллельно. Это позволяет существенно ускорить вычисления. Для организации потоков используются инструменты из стандартного заголовочного файла <future>. Метод возвращает *true*, если в контейнере есть элементы и вычисления во всех элементах контейнера вернули *true*; иначе метод возвращает *false*.

bool Calculate_thread()

Этот метод идентичен по назначению и своей структуре предыдущему методу. Отличие в том, что в нем используются инструменты из стандартного заголовочного файла <thread>. Метод также возвращает *true*, если в контейнере есть элементы и вычисления во всех элементах контейнера вернули *true*; иначе метод возвращает *false*.

Выбор метода из приведенных выше (*Calculate*, *Calculate_future* или *Calculate_thread*) остается за пользователем. Рекомендации те же, что и при выборе методов расчета в классе *clsStorage*.

В таблице ниже (Таблица 15) приведено среднее время вычислений для разных расчетных методов. Комбинации методов следующие:

- Calculate + CalcRawMatPurchPlan;
- Calculate_future + CalcRawMatPurchPlan_future;
- Calculate_thread + CalcRawMatPurchPlan_thread.

Для каждой из приведенных выше комбинаций было сделано не менее 50 вычислений. В качестве вещественного числа использовался собственный тип LongReal с количеством десятичных знаков мантиссы 64. Было установлено 480 периодов проекта. Расчет велся по 100 номенклатурным позициям продукции и 100 номенклатурным позициям ресурсов. Вычисления проводились на ноутбуке ASUS UX303U со следующими характеристиками: Процессор Intel Core i3-6100U CPU @ 2.30 GHz; Оперативная память 4,00 Gb; 64-bit Operation System Windows 10 Home Single Language, x64-based processor. Расчеты проводились в трех параллельных потоках.

Таблица 15 Сравнение времени вычислений для разных методов класса clsManufactory

Названия строк	Среднее время выполнения, сек.	Станд. отклонение времени выполнения, сек.
Calculate + CalcRawMatPurchPlan	143,0841	0,6503
Calculate_future + CalcRawMatPurchPlan_future	67,3102	0,7039
Calculate_thread + CalcRawMatPurchPlan_thread	67,4106	0,8951

GET-МЕТОДЫ. СЕКЦИЯ PRIVATE

strItem* gettotal(const strItem* (clsManufactItem::*f)() const) const

Метод создает динамический массив и возвращает на него указатель. Созданный одномерный массив, является аналогом двумерной матрицы с *балансами незавершенного производства* (при подстановке вместо *f* метода clsManufactItem::GetBalance) или *планами выхода всех продуктов* (при подстановке вместо *f* метода clsManufactItem::GetProductPlan) в натуральном, удельном и полном стоимостном выражении. Размер матрицы Manuf.size()*PrCount. Каждый элемент матрицы имеет тип strItem. Используется в методах класса clsManufactory: GetTotalBalance и GetTotalProduct.

Параметры:

const strItem (clsManufactItem::*f)() const* – указатель на один из методов класса *clsManufactItem*: *GetBalance* – функция возврата константного указателя на массив Balance с объемом, удельной и полной себестоимостью незавершенного производства для конкретного продукта, *GetProductPlan* – функция возврата константного указателя на массив ProductPlan с объемом, удельной и полной себестоимостью готовой продукции для конкретного продукта.

GET-МЕТОДЫ. СЕКЦИЯ PUBLIC

const size_t GetRMCount() const

Метол возвращает общее число позиций ресурсов, участвующих в производстве. Тип *size_t*.

const size_t GetPrCount() const

Метол возвращает число периодов проекта. Тип *size_t*.

const size_t GetProdCount() const

Метол возвращает число производимых продуктов. Тип *size_t*.

const string GetCurrency() const

Метол возвращает основную валюту проекта в виде текстовой строки. Тип *string*.

strNameMeas *GetProductDescription() const

Метод возвращает указатель на вновь создаваемый массив с именами и единицами измерения всех продуктов. Размер массива равен Manuf.size(). Тип элемента массива strNameMeas.

const strNameMeas *GetRMNames() const

Метод возвращает константный указатель на внутренний массив RMNames с именами и единицами измерения всех ресурсов. Размер массива равен RMCount. Тип элемента массива strNameMeas.

strNameMeas *GetRawMatDescription() const

Метод возвращает указатель на вновь создаваемый массив с именами и единицами измерения всех ресурсов. Размер массива равен RMCount. Тип элемента массива strNameMeas.

decimal *GetRawMatPurchPlan() const

Оглавление >>>

Несмотря на формальное отнесение данного метода к Get-методам, он по сути является *вычислительным*. Метод вызывается после проведения расчетов потребности в ресурсах методами CalcRawMatPurchPlan, CalcRawMatPurchPlan_future или CalcRawMatPurchPlan_thread и возвращает указатель на вновь создаваемый одномерный массив, являющийся аналогом двумерной матрицы размером RMCount*PrCount, в котором содержится потребность для каждого наименования ресурса в каждом периоде проекта. Потребность в ресурсах суммарная, по всем однотоварным производствам. Тип вещественного числа элемента массива определяется псевдонимом decimal.

strItem *GetRMPurchPlan() const

Данный метод возвращает тот же результат, что и предыдущий метод, но тип элементов массива strItem. В каждом элементе массива заполнены только поля *volume*, поля *price* и *value* нулевые.

strItem *GetTotalBalance() const

Метод возвращает указатель на вновь создаваемый одномерный массив, являющийся аналогом двумерной матрицы с балансами незавершенного производства для всех продуктов. Размер матрицы Manuf.size()*PrCount. Каждый элемент матрицы имеет тип strItem, т.е. имеет в своем составе значения *volume*, *price* и *value* для незавершенного производства.

strItem *GetTotalProduct() const

Метод возвращает указатель на вновь создаваемый одномерный массив, являющийся аналогом двумерной матрицы с планами выхода всех продуктов в натуральном, удельном и полном стоимостном выражении. Размер матрицы Manuf.size()*PrCount. Каждый элемент матрицы имеет тип strItem, т.е. имеет в своем составе значения *volume*, *price* и *value* для готового продукта.

const decimal* GetRecipeItem(const string& Name) const

Метод возвращает константный указатель на внутренний массив с рецептурами clsManufactItem::clsRecipeItem::recipeitem для производства продукта с именем *Name*. Тип вещественного числа элемента массива определяется псевдонимом decimal.

Параметры:

&Name – ссылка на строку с названием продукта. Тип *string*.

const decimal* GetRecipeItem(const size_t _ind) const

Перегруженный предыдущий метод; возвращает константный указатель на внутренний массив с рецептурами clsManufactItem::clsRecipeItem::recipeitem для производства продукта с индексом *_ind*. Тип вещественного числа элемента массива определяется псевдонимом decimal.

Параметры:

_ind – индекс элемента в контейнере Manuf. Тип *size_t*.

const strNameMeas* GetRawNamesItem(const string& Name) const

Метод возвращает константный указатель на внутренний массив с наименованиями ресурсов и единицами их измерения clsManufactItem::clsRecipeItem::rnames для производства продукта с именем *Name*. Тип элементов массива strNameMeas.

Параметры:

&Name – ссылка на строку с названием продукта. Тип *string*.

const strNameMeas* GetRawNamesItem(const size_t _ind) const

Перегруженный предыдущий метод; возвращает константный указатель на внутренний массив с наименованиями ресурсов и единицами их измерения clsManufactItem::clsRecipeItem::rnames для производства продукта с индексом *_ind*. Тип элементов массива strNameMeas.

Параметры:

_ind – индекс элемента в контейнере Manuf. Тип *size_t*.

const size_t GetDuration(const string& Name) const

Метод возвращает длительность производственного цикла в рецептуре продукта с именем *Name*. Тип возвращаемого значения *size_t*.

Параметры:

&Name – ссылка на строку с названием продукта. Тип *string*.

const size_t GetDuration(const size_t _ind) const

Перегруженный предыдущий метод; возвращает длительность производственного цикла в рецептуре продукта с индексом *_ind*. Тип возвращаемого значения *size_t*.

Параметры:

_ind – индекс элемента в контейнере Manuf. Тип *size_t*.

const size_t GetRCount(const string& Name) const

Метод возвращает число позиций ресурсов, участвующих в рецептуре продукта с именем *Name* (размер массива с наименованиями сырья, возвращаемого функцией GetRawNamesItem). Тип возвращаемого значения *size_t*.

Параметры:

&Name – ссылка на строку с названием продукта. Тип *string*.

const size_t GetRCount(const size_t _ind) const

Перегруженный предыдущий метод; возвращает число позиций ресурсов, участвующих в рецептуре продукта с индексом *_ind* (размер массива с наименованиями сырья, возвращаемого функцией GetRawNamesItem). Тип возвращаемого значения *size_t*.

Параметры:

_ind – индекс элемента в контейнере Manuf. Тип *size_t*.

const strNameMeas GetNameItem(const size_t _ind) const

Метод возвращает наименование продукта и единицу его измерения для продукта с индексом *_ind*. Тип возвращаемого значения strNameMeas.

Параметры:

_ind – индекс элемента в контейнере Manuf. Тип *size_t*.

МЕТОДЫ СОХРАНЕНИЯ СОСТОЯНИЯ ОБЪЕКТА В ФАЙЛ И ВОССТАНОВЛЕНИЯ ИЗ ФАЙЛА

bool StF(ofstream &_outF)

Метод имплементации записи в файловый поток текущего состояния экземпляра класса (запись в файловый поток, метод сериализации). Название метода является аббревиатурой, расшифровывается “StF” как “Save to File”. При удачной сериализации возвращает *true*, при ошибке записи возвращает *false*.

Параметры:

&_outF - экземпляр класса ofstream для записи данных. При инициализации переменной *_outF* необходимо установить флаг бинарного режима открытия потока (не текстового!) ios::binary.

bool RfF(ifstream &_inF)

Метод имплементации чтения из файлового потока текущего состояния экземпляра класса (чтение из файлового потока, метод десериализации). Название метода является аббревиатурой, расшифровывается “RfF” как “Read from File”. При удачной десериализации возвращает *true*, при ошибке чтения возвращает *false*.

Параметры:

&_inF - экземпляр класса ifstream для чтения данных. При инициализации переменной *_inF* необходимо установить флаг бинарного режима открытия потока (не текстового!) ios::binary.

bool SaveToFile(const string _filename)

Метод записи текущего состояния экземпляра класса в файл. При удачной записи в файл, возвращает *true*, при ошибке записи возвращает *false*.

Параметры:

_filename – имя файла; тип *string*.

bool ReadFromFile(const string _filename)

Метод восстановления состояния экземпляра класса из файла. При удачном чтении из файла, возвращает *true*, при ошибке чтения возвращает *false*.

Параметры:

_filename – имя файла; тип *string*.

МЕТОДЫ ВИЗУАЛЬНОГО КОНТРОЛЯ

void ViewProjectParametrs() const

Метод выводит в стандартный поток std::cout основные параметры класса: *длительность* проекта (контроль работоспособности метода GetPrCount), *количество* номенклатурных позиций *готовой продукции* (контроль работоспособности метода GetProdCount) и *количество* номенклатурных позиций *ресурсов* (контроль работоспособности метода GetRMCount).

void ViewRawMatPurchPlan() const

Метод выводит в стандартный поток std::cout потребность в ресурсах для всех продуктов в динамике (контроль работоспособности метода GetRawMatPurchPlan).

void ViewRawMatPrice() const

Метод выводит в стандартный поток std::cout цены на ресурсы (контроль работоспособности метода SetRawMatPrice).

void ViewCalculate() const

Метод выводит в стандартный поток std::cout *незавершенное производство* в *натуральном*, *удельном* и *полном стоимостном* выражении, а также *готовую продукцию* в *натуральном*, *удельном* и *полном*

стоимостном выражении для всех продуктов. Для каждого элемента контейнера метод вызывает его собственную функцию clsManufactItem::ViewCalculate. Для каждого продукта метод выводит шесть однострочных таблиц:

- незавершенное производство в натуральном выражении;
- незавершенное производство в удельном стоимостном выражении;
- незавершенное производство в полном стоимостном выражении;
- готовая продукция в натуральном выражении;
- готовая продукция в удельном стоимостном выражении;
- готовая продукция в полном стоимостном выражении.

Количество выводимых таблиц равно числу продуктов умноженному на шесть.

void ViewBalance() const

Метод выводит в стандартный поток std::cout *незавершенное производство* в *натуральном*, *удельном* и *полном стоимостном* выражении для всех продуктов (контроль работоспособности методов GetProductDescription и GetTotalBalance). Метод выводит три таблицы:

- незавершенное производство в натуральном выражении для всех продуктов;
- незавершенное производство в удельном стоимостном выражении для всех продуктов;
- незавершенное производство в полном стоимостном выражении для всех продуктов.

void ViewProductPlan() const

Метод выводит в стандартный поток std::cout *готовую продукцию* в *натуральном*, *удельном* и *полном стоимостном* выражении для всех продуктов (контроль работоспособности методов GetProductDescription и GetTotalProduct). Метод выводит три таблицы:

- готовая продукция в натуральном выражении для всех продуктов;
- готовая продукция в удельном стоимостном выражении для всех продуктов;
- готовая продукция в полном стоимостном выражении для всех продуктов.

void ViewRecipes() const

Метод поочередно выводит в стандартный поток std::cout рецептуры всех продуктов. Для каждого элемента контейнера метод вызывает его собственную функцию clsManufactItem::ViewRefRecipe.

ПРИМЕР ПРОГРАММЫ С КЛАССОМ clsManufactory

Ниже представлен листинг программы, демонстрирующей работу класса *clsManufactory*. Данная программа может быть использована для тестирования быстродействия методов Calculate, CalcRawMatPurchPlan, Calculate_future, CalcRawMatPurchPlan_future, Calculate_thread, CalcRawMatPurchPlan_thread при разных количестве периодов проекта *PrCount*, количестве номенклатурных позиций готовой продукции *MSize* и количестве номенклатурных позиций ресурсов *RMCount* (см. код ниже). С целью сокращения объема выводимой на экран информации, в данном примере эти величины малы.

```
#include <iostream>
#include <manufact_module.h>    // Подключаем модуль "производство"

using namespace std;

int main() {

    setlocale(LC_ALL, "Russian");   // Установка русского языка для вывода

    /** Основные параметры проекта **/
    size_t PrCount = 8;     // Длительность проекта
    size_t RMCount = 10;    // Полное число наименований ресурсов
    size_t MSize = 5;       // Число позиций в ассортименте готовой продукции
    size_t rcount  = 5;     // Число наименований ресурсов в рецептуре продукта
    size_t duratio = 7;     // Длительность производственного цикла

/** Формирование полного массива имен ресурсов с единицами измерения **/
    strNameMeas* RMNames = new strNameMeas[RMCount];
    for(size_t i =sZero; i<RMCount; i++) {
        (RMNames+i)->name = "Rawmat_" + to_string(i);
        (RMNames+i)->measure = "kg";
    };

/** Формирование массива цен (удельной себестоимости) для ресурсов **/
    decimal* RMPrice = new decimal[RMCount*PrCount];
    for(size_t i=sZero; i<RMCount; i+=5 ) {
        for(size_t j=sZero; j<PrCount; j++) {
            *(RMPrice+PrCount*i+j) = 10.0;
            *(RMPrice+PrCount*(i+1)+j) =  5.0;
            *(RMPrice+PrCount*(i+2)+j) = 20.0;
            *(RMPrice+PrCount*(i+3)+j) = 2.5;
            *(RMPrice+PrCount*(i+4)+j) = 50.0;
        };
    };

/** Рецептура для единичного продукта **/
    strNameMeas* rnames = new strNameMeas[rcount] { {"Rawmat_0", "kg"}, {"Rawmat_1", "kg"},
    {"Rawmat_2", "kg"}, {"Rawmat_3", "kg"}, {"Rawmat_4", "kg"} };   // Номенклатура ресурсов
    decimal* recipe = new decimal[rcount*duratio]{
        10.0, 10.0, 10.0, 10.0, 10.0, 10.0, 10.0,  // Рецептура
        50.0,  0.0,  0.0, 50.0,  0.0,  0.0,  0.0,
        30.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,
        0.0,  0.0,  0.0,  0.0,  0.0,  0.0, 20.0,
        0.0,  0.0,  0.0,  10.0, 0.0,  0.0,  0.0 };
```

```
/** Формирование плана выпуска продуктов **/
    strItem* ProPlan = new strItem[PrCount];
    for(size_t i=sZero; i<PrCount; i++) {
        (ProPlan+i)->volume = 10.0;
        (ProPlan+i)->value = (ProPlan+i)->price  =  0.0;
    };

/** Формирование списка продуктов **/
    strNameMeas* ProdNames = new strNameMeas[MSize];
    for(size_t i =sZero; i<MSize; i++) {
        (ProdNames+i)->name = "Product_" + to_string(i);
        (ProdNames+i)->measure = "kg";
    };

/** Создание произвоства **/
    clsManufactory* MMM = new clsManufactory(PrCount, RMCount, RMNames, MSize);

/** Ввод продуктов, рецептур и плана выпуска продуктов **/
    for(size_t i=sZero; i<MSize; i++)
        if(!MMM->SetManufItem((ProdNames+i)->name,\
            (ProdNames+i)->measure, duratio, rcount,\
            rnames, recipe, ProPlan))
            cout << "Ошибка добавления производства для "\
            << (ProdNames+i)->name << " продукта" << endl;

/** Расчет потребности в сырье и материалах **/
    MMM->CalcRawMatPurchPlan();

/** Ввод цен сырья и материалов **/
    MMM->SetRawMatPrice(RMPrice);

/** Расчет себестоимости продуктов и незавершенного производства **/
    MMM->Calculate();

/** Выборочный визуальный контроль работоспособности **/
    MMM->ViewProjectParametrs();
    MMM->ViewBalance();
    MMM->ViewProductPlan();

delete[] RMNames;
    delete[] RMPrice;
    delete[] rnames;
    delete[] recipe;
    delete[] ProPlan;
    delete[] ProdNames;
    delete MMM;

    return 0;
}
```

Вывод на экран выбранной информации представлен ниже.

```
*** Основные параметры проекта ***
Количество периодов проекта            8
Общее число позиций продуктов          5
Общее число позиций сырья и материалов 10
Валюта проекта                         RUR

*** Незавершенное производство ***

By volume
Name           Measure              0         1         2         3         4         5         6         7
Product_0      kg                0.00     10.00     10.00     10.00     10.00     10.00     10.00      0.00
Product_1      kg                0.00     10.00     10.00     10.00     10.00     10.00     10.00      0.00
Product_2      kg                0.00     10.00     10.00     10.00     10.00     10.00     10.00      0.00
Product_3      kg                0.00     10.00     10.00     10.00     10.00     10.00     10.00      0.00
Product_4      kg                0.00     10.00     10.00     10.00     10.00     10.00     10.00      0.00


By price
Name           Measure              0         1         2         3         4         5         6         7
Product_0      RUR/kg            0.00    950.00   1050.00   1150.00   2000.00   2100.00   2200.00      0.00
Product_1      RUR/kg            0.00    950.00   1050.00   1150.00   2000.00   2100.00   2200.00      0.00
Product_2      RUR/kg            0.00    950.00   1050.00   1150.00   2000.00   2100.00   2200.00      0.00
Product_3      RUR/kg            0.00    950.00   1050.00   1150.00   2000.00   2100.00   2200.00      0.00
Product_4      RUR/kg            0.00    950.00   1050.00   1150.00   2000.00   2100.00   2200.00      0.00


By value
Name           Measure              0         1         2         3         4         5         6         7
Product_0      RUR               0.00   9500.00  10500.00  11500.00  20000.00  21000.00  22000.00      0.00
Product_1      RUR               0.00   9500.00  10500.00  11500.00  20000.00  21000.00  22000.00      0.00
Product_2      RUR               0.00   9500.00  10500.00  11500.00  20000.00  21000.00  22000.00      0.00
Product_3      RUR               0.00   9500.00  10500.00  11500.00  20000.00  21000.00  22000.00      0.00
Product_4      RUR               0.00   9500.00  10500.00  11500.00  20000.00  21000.00  22000.00      0.00



*** Готовая продукция ***

By volume
Name           Measure              0         1         2         3         4         5         6         7
Product_0      kg                0.00      0.00      0.00      0.00      0.00      0.00      0.00     10.00
Product_1      kg                0.00      0.00      0.00      0.00      0.00      0.00      0.00     10.00
Product_2      kg                0.00      0.00      0.00      0.00      0.00      0.00      0.00     10.00
Product_3      kg                0.00      0.00      0.00      0.00      0.00      0.00      0.00     10.00
Product_4      kg                0.00      0.00      0.00      0.00      0.00      0.00      0.00     10.00


By price
Name           Measure              0         1         2         3         4         5         6         7
Product_0      RUR/kg            0.00      0.00      0.00      0.00      0.00      0.00      0.00   2350.00
Product_1      RUR/kg            0.00      0.00      0.00      0.00      0.00      0.00      0.00   2350.00
Product_2      RUR/kg            0.00      0.00      0.00      0.00      0.00      0.00      0.00   2350.00
Product_3      RUR/kg            0.00      0.00      0.00      0.00      0.00      0.00      0.00   2350.00
Product_4      RUR/kg            0.00      0.00      0.00      0.00      0.00      0.00      0.00   2350.00


By value
Name           Measure              0         1         2         3         4         5         6         7
Product_0      RUR               0.00      0.00      0.00      0.00      0.00      0.00      0.00  23500.00
Product_1      RUR               0.00      0.00      0.00      0.00      0.00      0.00      0.00  23500.00
Product_2      RUR               0.00      0.00      0.00      0.00      0.00      0.00      0.00  23500.00
Product_3      RUR               0.00      0.00      0.00      0.00      0.00      0.00      0.00  23500.00
Product_4      RUR               0.00      0.00      0.00      0.00      0.00      0.00      0.00  23500.00
```

КАК ПОЛЬЗОВАТЬСЯ КЛАССАМИ МОДУЛЯ MANUFACT

Классы *clsRecipeItem*, *clsManufactItem* и *clsManufactory* предназначены для описания учетных процессов в производстве при моделировании учета частичных затрат на основе переменных затрат. Объединяющим классом является класс clsManufactory. Основная задача классов - расчет ресурсной себестоимости готовой продукции и незавершенного производства.

Общая логика работы классов представлена на рисунке ниже (Рисунок 5):

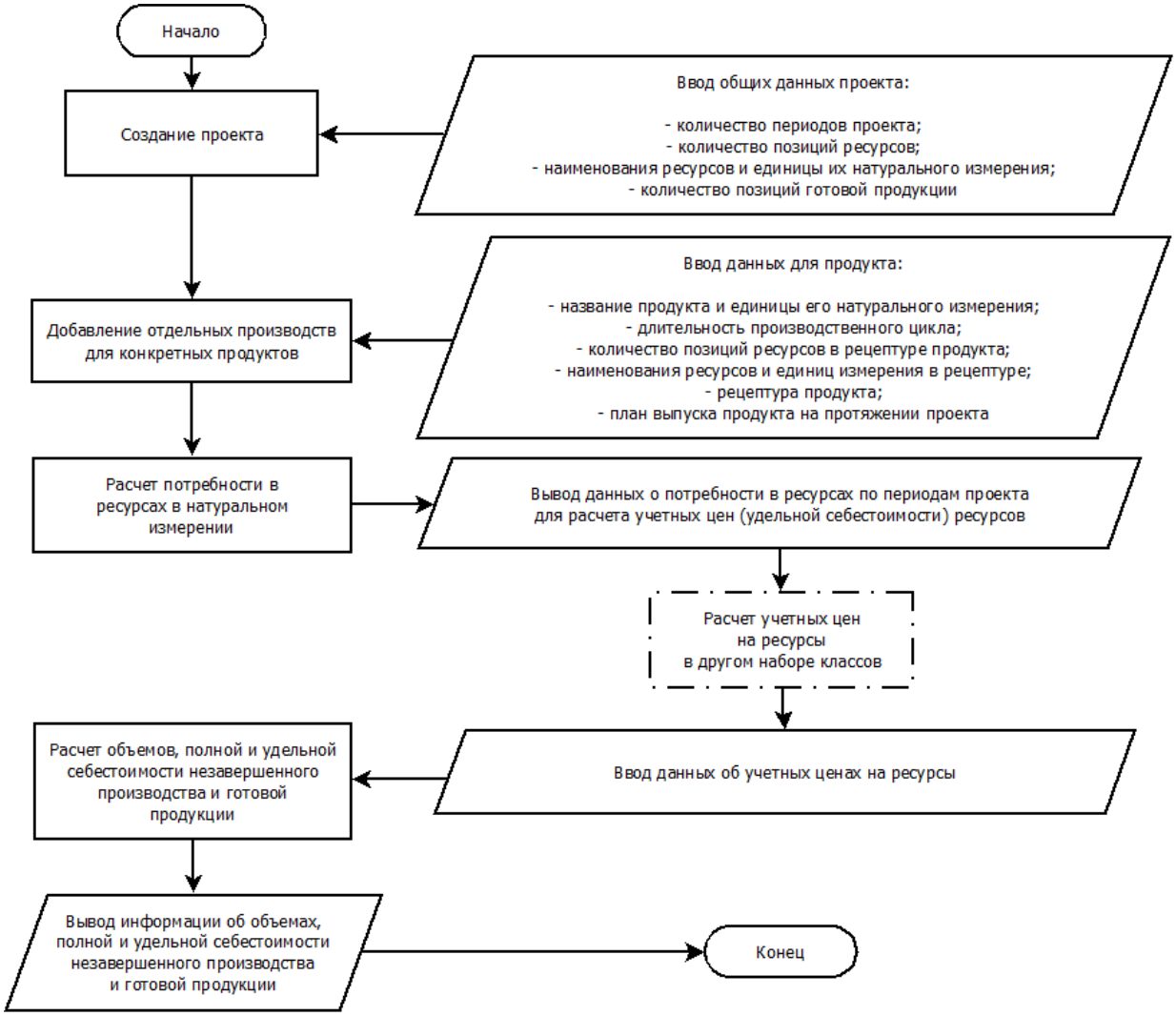

Рисунок 5 Принципиальная схема взаимодействия классов модуля manufact

Роли классов в этом процессе следующие:

- класс **clsRecipeItem** - ядро всего алгоритма, содержит описание рецептуры/ технологической карты одной номенклатурной единицы (SKU) продукции и методы расчета потребности в ресурсах для производства этого продукта, методы расчета объемов, полной и удельной себестоимости незавершенного производства и готового продукта;
- класс **clsManufactItem** - контейнер для экземпляра класса *clsRecipeItem* и хранилище данных о количестве периодов проекта, плане выпуска одного вида продукта, потребности в ресурсах для производства этого продукта, учетных цен на ресурсы для его производства, объемов, полной и удельной себестоимости незавершенного производства и готового продукта; этот класс представляет собой полноценное производство для одного вида продукта; Set-методы вводят данных во внутренние поля, Calc-методы обращаются к Calc-методам внутреннего объекта типа *clsRecipeItem* и заполняют поля расчетными данными, Get-методы возвращают данные из внутренних полей;
- класс **clsManufactory** - контейнер для нескольких экземпляров класса *clsManufactItem*, - аналог полноценного производства ассортимента продукции и хранилище данных о количестве периодов проекта, полном количестве позиций ресурсов, используемых в производстве, наименованиях и единицах измерения полного списка ресурсов; Set-методы класса позволяют добавить в контейнер производство какого-либо конкретного продукта, учетные цены на все ресурсы; Calc_методы обращаются к Calc-методам отдельных производств (объектам типа *clsManufactItem*), Get-методы возвращают объединенную информацию.

Все классы имеют в своем составе методы сериализации и десериализации и методы визуального контроля.

Схема вложенности классов и отношений между ними приведена на рисунке ниже (Рисунок 6).

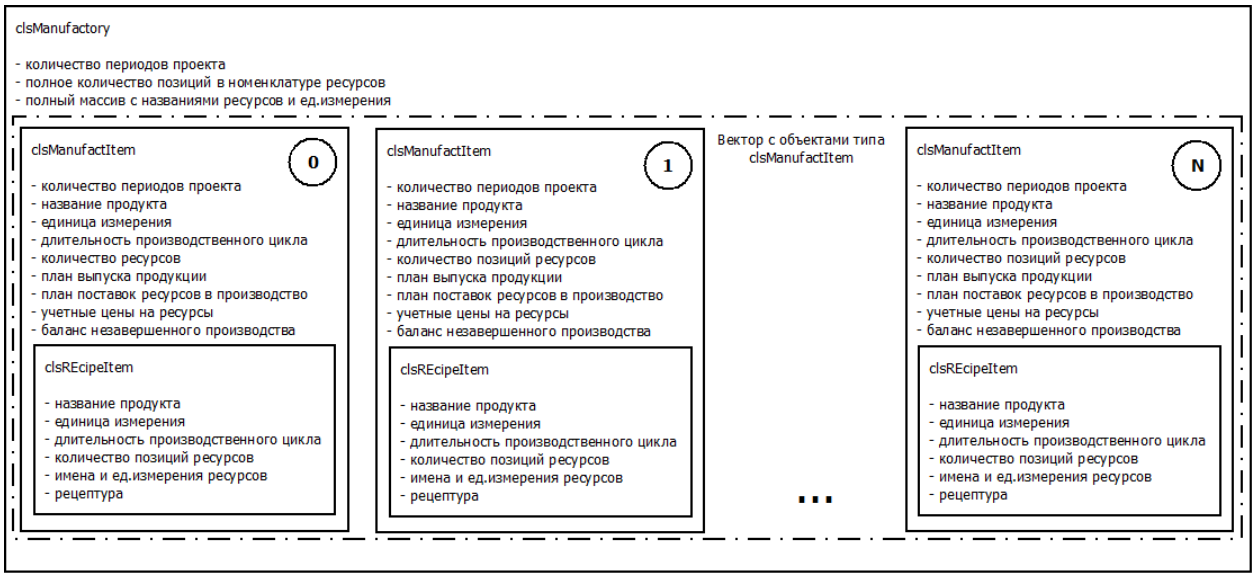

Рисунок 6 Схема отношений между экземплярами классов модуля manufact

СОЗДАНИЕ ПРОЕКТА

Создание проекта начинается с создания экземпляра класса *clsManufactory*. Для создания используется, например, конструктор с вводом длительности проекта, массива с наименованиями и единицами измерения ресурсов и резервированием памяти для заданного числа однотоварных производств.

На этом этапе вводятся исходные данные:

- количество периодов проекта;
- полное количество позиций в номенклатуре сырья и материалов;
- наименования и единицы измерения номенклатурных позиций сырья и материалов в виде массива;
- предварительное количество позиций в номенклатуре готовой продукции.

Созданный проект пока пуст и не имеет в своем составе ни одного производства. Это лишь контейнер, в который могут быть добавлены отдельные производства, - экземпляры класса clsManufactItem, - для конкретных наименований продукции.

ДОБАВЛЕНИЕ ПРОИЗВОДСТВА ДЛЯ КОНКРЕТНОГО ПРОДУКТА

При добавлении производства для конкретного продукта требуется информация:

- наименование продукта;
- единица измерения его натурального количества;
- длительность производственного цикла;
- число номенклатурных позиций сырья и материалов, участвующих в производстве данного продукта;
- наименования сырья и материалов и единицы измерения этих позиций сырья и материалов; при этом эти наименования и единицы измерения должны быть подмножеством введенного на этапе создания проекта множества имен и единиц измерения для сырья и материалов;
- рецептура продукта;
- план выпуска продукта на протяжении проекта.

В результате добавления нового производства в контейнере появляется новый элемент типа clsManufactItem. Добавление производства для конкретного продукта осуществляется с помощью метода clsManufactory::SetManufItem. Существуют несколько вариантов этого метода:

- с *копированием* ранее созданного экземпляра класса clsManufactItem в новый элемент контейнера;
- с *перемещением* ранее созданного экземпляра класса *clsManufactItem* в новый элемент контейнера;
- с *копированием* ранее созданного экземпляра класса clsRecipeItem и копированием массива с планом выпуска продукта в новый элемент контейнера;
- с *перемещением* ранее созданного экземпляра класса *clsRecipeItem* и копированием массива с планом выпуска продукта в новый элемент контейнера;
- с *вводом параметров*, указанных в предыдущем абзаце и непосредственным созданием экземпляра класса clsManufactItem в новом элементе вектора.

РАСЧЕТ ПОТРЕБНОСТИ В СЫРЬЕ И МАТЕРИАЛАХ ДЛЯ ПРОИЗВОДСТВА ПРОДУКЦИИ

Следующим этапом после ввода всех планируемых к производству продуктов, является расчет потребности в ресурсах в натуральном измерении для всех продуктов. Расчет осуществляется с помощью одного из трех методов на выбор: clsManufactory::CalcRawMatPurchPlan, clsManufactory::CalcRawMatPurchPlan_future или clsManufactory::CalcRawMatPurchPlan_thread. Каждый из этих методов вызывает поочередно одноименные методы для каждого элемента контейнера типа clsManufactItem, которые в свою очередь, вызывают методы clsRecipeItem::CalcRawMatVolume экземпляра класса рецептур каждого продукта (см. Рисунок 7).

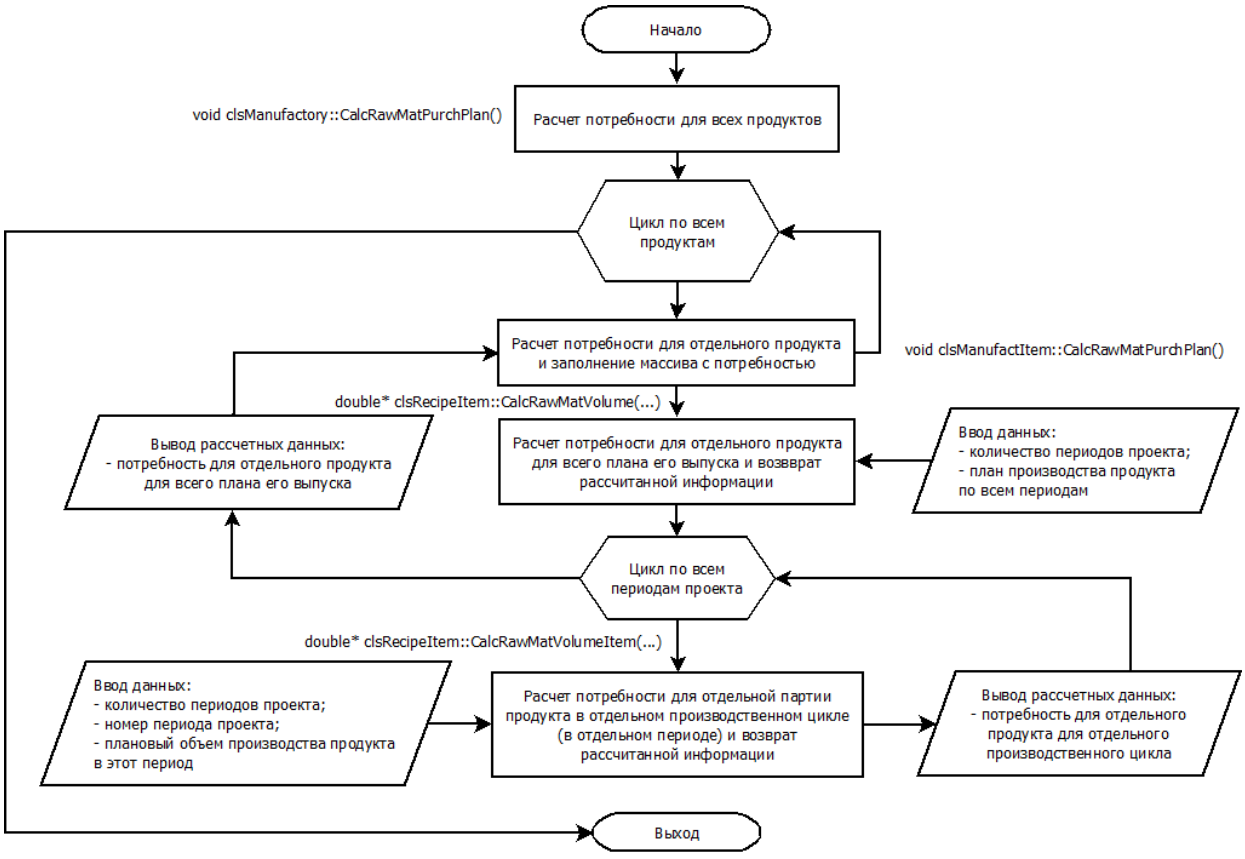

Рисунок 7 Расчет потребности в ресурсах в натуральном выражении для всех продуктов

Метод не имеет параметров, он использует информацию, введенную ранее на этапе создания производств для конкретных продуктов. Метод не возвращает рассчитанную информацию, представляя собой исключительно расчетный инструмент. Результаты расчетов для каждого наименования продукта записываются в соответствующие элементы контейнера в массивы clsManufactItem::RawMatPurchPlan.

ПОЛУЧЕНИЕ ИНФОРМАЦИИ О ПОТРЕБНОСТИ В РЕСУРСАХ ДЛЯ ПРОИЗВОДСТВА ПРОДУКЦИИ

Для получения информации о потребности в ресурсах, рассчитанной на предыдущем этапе, используется один из двух методов: decimal* clsManufactory::GetRawMatPurchPlan или strItem* clsManufactory::GetRMPurchPlan. Первый метод возвращает информацию о потребностях в ресурсах в виде указателя на массив вещественных чисел типа decimal, другой – указатель на массив типа strItem, в котором заполнены только поля *volume*, остальные поля заполнены нулями. Оба массива одномерные, являются аналогами двумерной матрицы с числом строк, равным общему числу позиций сырья и материалов RMCount и числом столбцов, равному числу периодов проекта PrCount. Обращение к *(i,j)-элементу* такой «матрицы» производится так:

```
*(arrayref + PrCount * i + j),
```

где *i* - номер строки, *j* - номер столбца, *arrayref* – имя массива, выполняющее роль указателя на первый элемент массива. Условная *(i,j)-ячейка* матрицы содержит расход *i*-го наименования сырья на производство всех продуктов в *j*-м периоде проекта.

Массив создается вновь. Ответственность за освобождение памяти после использования этого массива лежит на программисте, использующем данный набор классов. После использования полученного массива (например, с именем *arrayref*), требуется его ручное удаление командой *delete[]*.

РАСЧЕТ УЧЕТНЫХ ЦЕН НА СЫРЬЕ И МАТЕРИАЛЫ

Для расчета учетных цен на ресурсы используются сторонние программные инструменты. Например, рекомендуются к использованию инструменты «Склад ресурсов» с классами clsSKU и clsStorage.

Предполагается, что после получения *складом* информации о потребности в ресурсах, склад возвращает информацию об учетных ценах на сырье и материалы (например, с помощью методов clsStorage::GetShipPrice или clsStorage::GetShip).

ВВОД УЧЕТНЫХ ЦЕН НА СЫРЬЕ И МАТЕРИАЛЫ

Ввод учетных цен на ресурсы осуществляется методом clsManufactory::SetRawMatPrice.

Каждый элемент контейнера содержит свой собственный массив цен clsManufactItem::RawMatPrice, отличающийся от общего массива, передаваемого указанным выше методом. В общем массиве цен присутствует *полный список* ресурсов, а в массивах *clsManufactItem::RawMatPrice* находятся только те позиции ресурсов, которые участвуют в производстве данного конкретного продукта. Поэтому метод *clsManufactory::SetRawMatPrice* «разделяет» общий массив на ряд частных массивов. Принципиальная схема алгоритма приведена ниже (Рисунок 8).

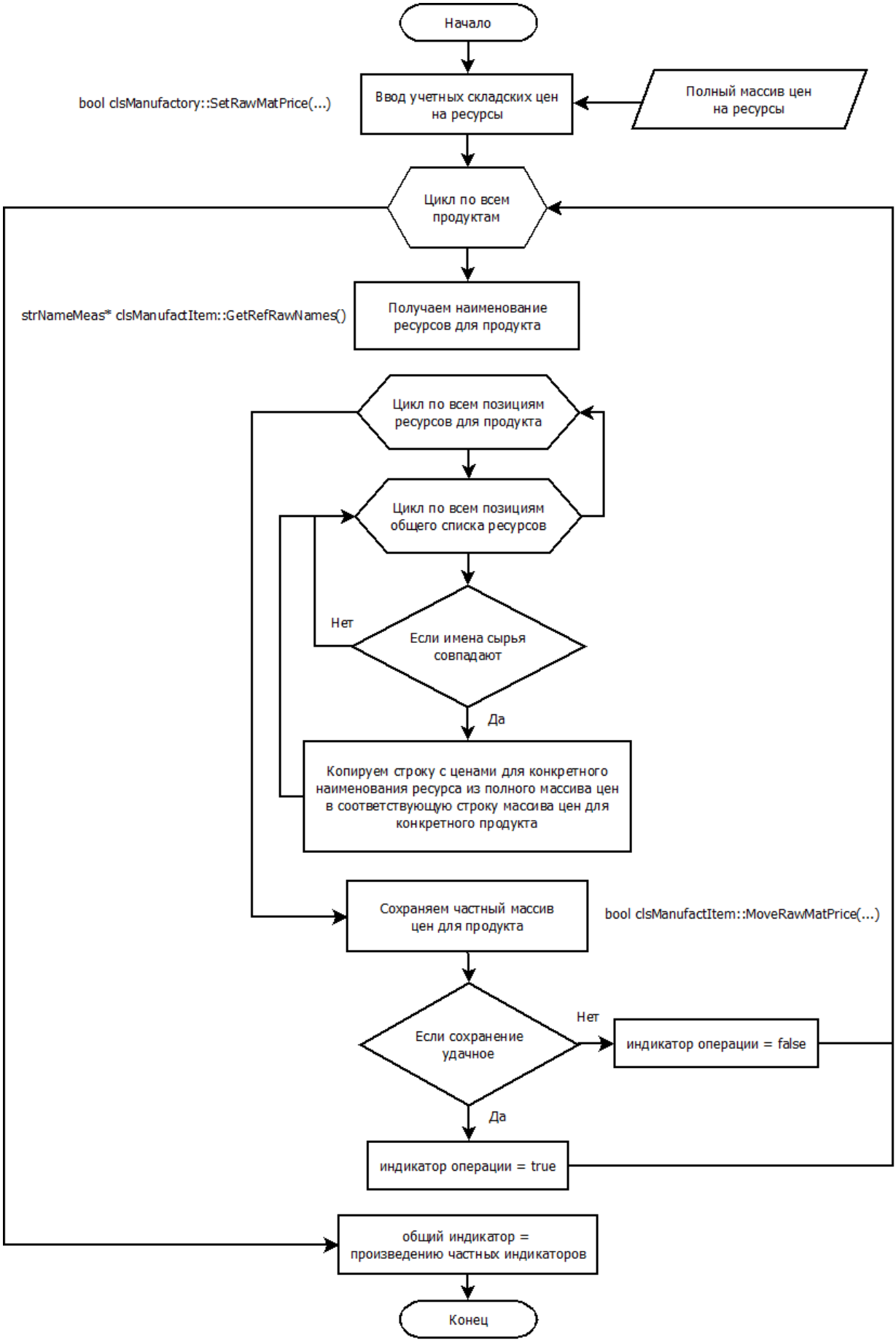

Рисунок 8 Ввод учетных цен на ресурсы

Оглавление >>>

РАСЧЕТ СЕБЕСТОИМОСТИ НЕЗАВЕРШЕННОГО ПРОИЗВОДСТВА И ГОТОВОЙ ПРОДУКЦИИ

Следующим этапом после ввода учетных цен на ресурсы является, собственно, расчет себестоимости незавершенного производства и готовой продукции. Расчет осуществляется с помощью одного из трех методов на выбор: clsManufactory::Calculate, clsManufactory::Calculate_future или clsManufactory::Calculate_thread. Каждый из этих методов вызывает поочередно для каждого элемента контейнера функцию clsManufactItem::CalculateItem которая, в свою очередь, вызывает методы clsRecipeItem::CalcWorkingBalance и clsRecipeItem::CalcProductBalance экземпляра класса рецептур каждого продукта (см. Рисунок 9).

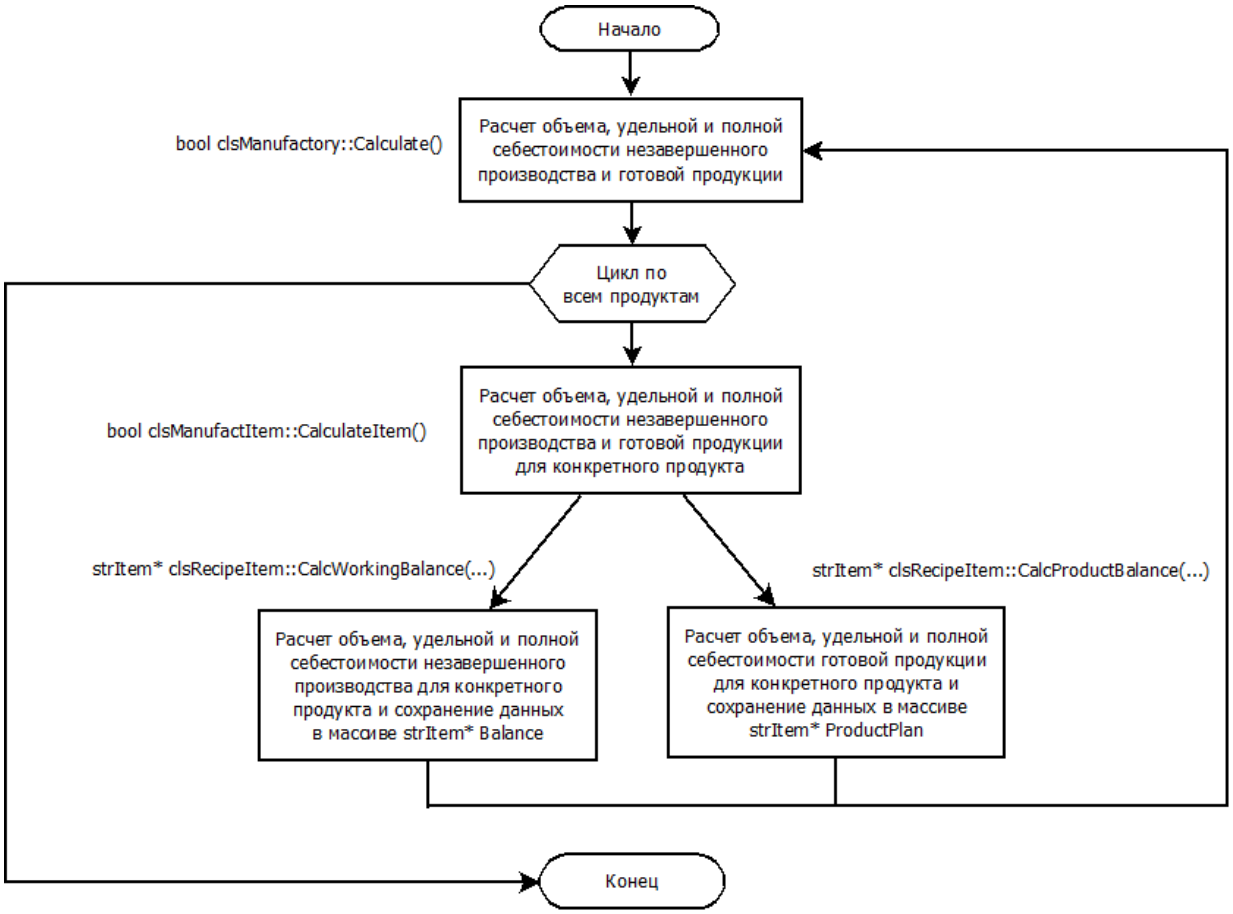

Рисунок 9 Расчет объема, удельной и полной себестоимости незавершенного производства и готовой продукции

Метод не имеет параметров, он использует информацию, полученную ранее. Метод не возвращает рассчитанную информацию, представляя собой исключительно расчетный инструмент. Результаты расчетов для каждого наименования продукта записываются в соответствующие элементы контейнера в массивы clsManufactItem::Balance и clsManufactItem::ProductPlan.

ПОЛУЧЕНИЕ ИНФОРМАЦИИ О СЕБЕСТОИМОСТИ НЕЗАВЕРШЕННОГО ПРОИЗВОДСТВА И ГОТОВОЙ ПРОДУКЦИИ

Для получения информации о себестоимости незавершенного производства и готовой продукции, рассчитанной на предыдущем этапе, используются методы:

- strItem* clsManufactory::GetTotalBalance – для получения указателя на массив с балансами незавершенного производства для всех продуктов;
- strItem* clsManufactory::GetTotalProduct – для получения указателя на массив с планами выхода всех продуктов.

Оба массива одномерные, являются аналогами двумерной матрицы с числом строк, равным общему числу продуктов Manuf.size() и числом столбцов, равному числу периодов проекта PrCount. Тип элемента массива strItem. Поля *volume* неизменны с момента добавления производства для конкретного продукта, поля *price* и *value* – результаты расчетов, произведенных на предыдущем этапе.

Массивы создаются вновь. Ответственность за освобождение памяти после использования этих массивов лежит на программисте, использующем данный набор классов.

МОДЕЛИРОВАНИЕ С ПОМОЩЬЮ КЛАССОВ clsStorage И clsManufactory

При моделировании предприятия необходимо принимать во внимание, сколько *центров затрат* (Cost Centre) планируется выделить в модели. Центр затрат может быть описан только классом *clsManufactory*. С его помощью можно аллокировать затраты на единицу ресурсов/ продукции, услуг и получить *добавленную стоимость*. В отличие от класса *clsStorage*, в котором такой функционал отсутствует.

СОЗДАНИЕ МОДЕЛИ С ОДНИМ ЦЕНТРОМ ЗАТРАТ

Пример 1, описанный в разделе «Возможности FROMA2», представляет предприятие с *тремя подразделениями* (Склад сырья и материалов, Производство и Склад готовой продукции), но одним *центром затрат*. Принципиальная схема модели приведена на рисунке ниже.

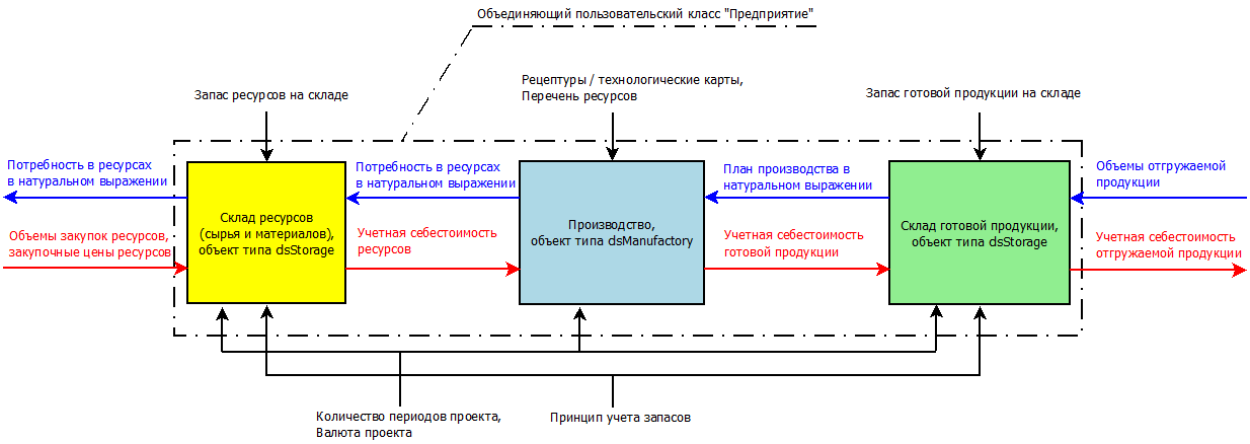

Рисунок 10 Принципиальная схема модели с одним центром затрат

Информационные потоки о готовой продукции и ресурсах в *натуральном выражении* представлены на схеме ***синими стрелками***. Информационные потоки о *стоимостных показателях* готовой продукции и ресурсов представлены на схеме ***красными стрелками***.

Рассмотрим подробнее *алгоритм расчета* учетной себестоимости отгружаемой со склада готовой продукции.

1. Для описания предприятия целесообразно создать *объединяющий* класс «Предприятие», в котором полями будут выступать Склад готовой продукции (экземпляр класса *clsStorage*), Производство (экземпляр класса *clsManufactory*) и Склад готовой продукции (экземпляр класса *clsStorage*), а также поля, обеспечивающие хранение и передачу данных между ними. Например, как во фрагменте кода ниже.

```
class clsEnterprise {
...
 size_t PrCount;                    // Количество периодов проекта
 Currency Cur;                      // Домашняя валюта проекта
 AccountingMethod Amethod;          // Принцип учета запасов
 size_t ProdCount;                  // Полное число выпускаемых продуктов
 strItem* Ship;                     // Указатель на массив отгрузок со склада
 strNameMeas* ProdNames; // Указатель на массив названий и ед. измерения продуктов
 size_t RMCount;                    // Полное число наименований ресурсов
 strNameMeas* RMNames;   // Указатель на массив названий и ед. измерения ресурсов
 decimal* Purch;                    // Указатель на массив цен на ресурсы
 clsStorage* Warehouse;             // Указатель на склад готовой продукции
 clsManufactory* Manufactory;       // Указатель на производство
 clsStorage* RawMatStock;           // Указатель на склад сырья и материалов
...
}
```

2. Далее необходимо создать Склад готовой продукции, ввести в него длительность проекта, данные об отгрузках готовой продукции в натуральном выражении, норматив остатков продукции на складе, принцип учета запасов и рассчитать потребность в продукции, получаемой из производства (Производственный план). Подробнее об использовании класса clsStorage см. раздел «Как пользоваться классом clsStorage».
3. На следующем шаге мы создаем Производство, вводим в него длительность проекта, рецептуры/ технологические карты и полученный на предыдущем этапе производственный план. Рассчитываем потребность в ресурсах для обеспечения выполнения производственного плана. Подробнее об использовании класса clsManufactory см. раздел «Как пользоваться классами модуля manufact».
4. Далее необходимо создать Склад ресурсов, ввести в него длительность проекта, данные об отгрузках ресурсов в производство в натуральном выражении, норматив остатков продукции на складе, принцип учета запасов и рассчитать потребность ресурсах, закупаемых предприятием.
5. После того, как определен план закупок ресурсов и проведены переговоры с поставщиками, сотрудники предприятия имеют на руках окончательную версию графика и объемов закупок ресурсов и цен на них. Этот график и цены ресурсов вводятся в Склад ресурсов и рассчитывается учетная себестоимость этих ресурсов на выходе со склада.
6. Эта учетная себестоимость ресурсов вводится в Производство и рассчитывается учетная себестоимость готовой продукции.
7. Учетная себестоимость готовой продукции вводится в Склад готовой продукции и рассчитывается учетная себестоимость отгружаемой с этого склада продукции.

Нетрудно заметить, что алгоритм расчета учетной себестоимости отгружаемой со Склада готовой продукции является *двухпроходным*: сначала *от клиентов к поставщикам* идет расчет *объемных* показателей в натуральном выражении, потом *от поставщиков к клиентам* идет расчет *денежных* показателей. Эту специфику следует учитывать при создании агрегированных расчетных методов объединяющего класса.

СОЗДАНИЕ МОДЕЛИ С НЕСКОЛЬКИМИ ЦЕНТРАМИ ЗАТРАТ

Рассмотрим пример модели, в которой *три подразделения* (Склад сырья и материалов, Производство и Склад готовой продукции) и, соответственно, *три одноименных центра затрат*. Принципиальная схема модели приведена на рисунке ниже.

Как и на предыдущем рисунке, информационные потоки о готовой продукции и ресурсах в *натуральном выражении* представлены на схеме ***синими стрелками***. Информационные потоки о *стоимостных показателях* готовой продукции и ресурсов представлены на схеме ***красными стрелками***.

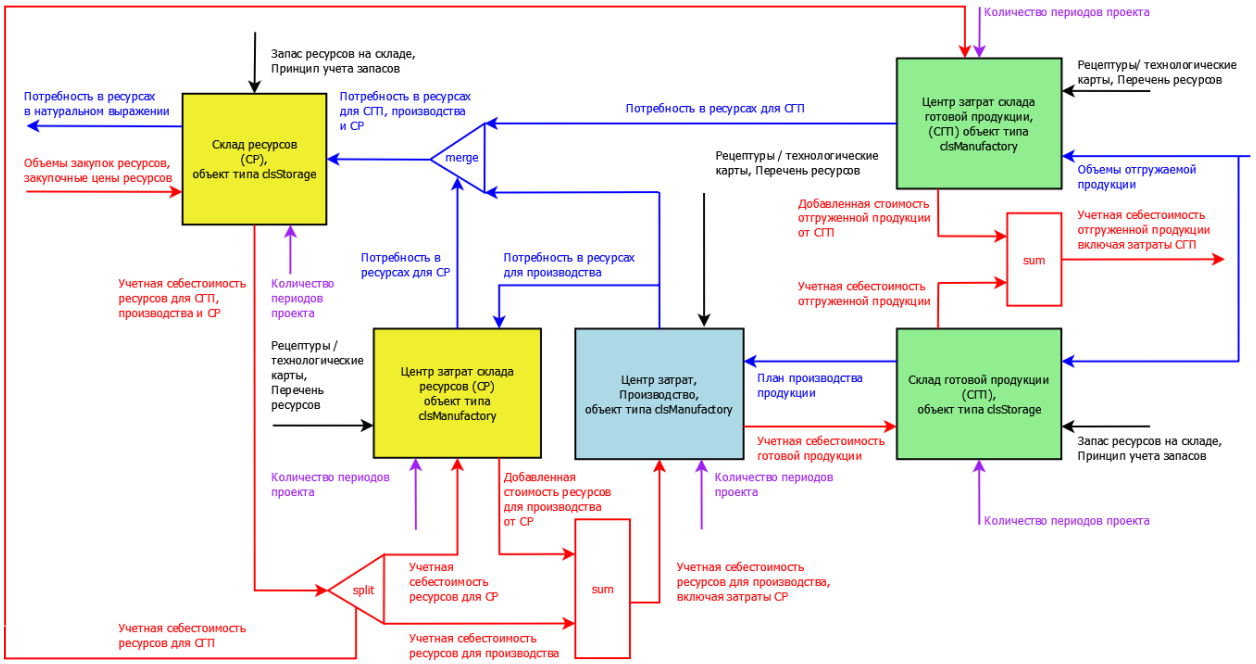

Рисунок 11 Принципиальная схема модели с тремя центрами затрат

На рисунке *Склад готовой продукции* (СГП) и *Склад ресурсов* (СР), каждый из них представлен *двумя объектами*: СГП – прямоугольниками ***зеленого цвета***, СР – прямоугольниками ***желтого цвета***. Каждый склад имеет в своем составе *Центр учета затрат*, представленный экземпляром класса *clsManufactory* и *Центр учета запасов,* представленный экземпляром класса *clsStorage*.

Рассмотрим подробнее *алгоритм расчета* учетной себестоимости отгружаемой со склада готовой продукции.

1. Как и в предыдущем примере, для описания предприятия целесообразно предусмотреть *объединяющий* класс «Предприятие». Но в данном случае, также, может быть полезным дополнительно предусмотреть объединяющие классы для складов: отдельно объединяющий класс для СГП, в котором полями будут выступать Центр учета затрат (экземпляр класса *clsManufactory*) и Центр учета запасов (экземпляр класса *clsStorage*), и объединяющий класс для СР с аналогичными полями. Объединяющие классы складов, в свою очередь, станут полями объединяющего класса для всего предприятия. (Для облегчения читаемости рисунка, мы не стали наносить на него линии объединяющих классов.)
2. После создания экземпляра класса «Предприятие», мы создаем *Склад готовой продукции* и в нем создаем *Центр учета запасов* СГП и *Центр учета затрат* СГП. В оба этих центра вводим данные: длительность проекта, данные об отгрузках готовой продукции в натуральном выражении. В Центр учета запасов ещё вводятся принцип учета запасов и норматив остатков продукции на складе. В Центр учета затрат вводятся: перечень ресурсов, используемых на СГП и технологические карты, описывающие применение этих ресурсов на СГП. *Технологические карты* для центра затрат СГП содержат информацию о затратах СГП на единицу отгружаемой продукции, например, расход групповой упаковочной тары (если упаковка происходит на складе) и трудозатраты на упаковочный процесс. В Центре учета запасов рассчитывается потребность в продукции, получаемой из производства (Производственный план). В Центре учета затрат рассчитывается потребность в ресурсах для СГП.
3. На следующем шаге мы создаем *Производство*, вводим в него длительность проекта, перечень ресурсов для производства продукции (например, список сырья и материалов, используемых на производстве), рецептуры/ технологические карты и полученный на предыдущем шаге производственный план. Рассчитываем потребность в ресурсах для обеспечения выполнения производственного плана.

4. Далее необходимо создать *Склад ресурсов* и в нем создать *Центр учета запасов* СР и *Центр учета затрат* СР. В оба этих центра вводится длительность проекта. В Центр учета затрат ещё вводятся: перечень ресурсов, используемых на СР и технологические карты, а также потребность в ресурсах на производстве. *Технологические карты* для центра затрат СР содержат информацию о затратах СР на единицу отгружаемых ресурсов, например, трудозатраты на сортировку сырья и материалов. Производится расчет потребности в ресурсах для СР.
5. В объединяющем классе для СР необходимо предусмотреть *функцию, производящую слияние* массивов с информацией о потребности в ресурсах для СР, потребности в ресурсах для производства и потребности в ресурсах для СГП в единый массив с общей потребностью в ресурсах (на схеме эта функция обозначена *синим треугольником с надписью "**merge**"*). Этот общий массив вводится в Центр учета запасов СР. В Центр учета запасов ещё вводятся принцип учета запасов и норматив остатков продукции на Складе ресурсов. Производится расчет потребности в закупках всех ресурсов в натуральном выражении (План закупок ресурсов).
6. После того, как определен план закупок ресурсов и проведены переговоры с поставщиками, сотрудники предприятия имеют на руках окончательную версию графика и объемов закупок ресурсов и цен на них. Этот график и цены ресурсов вводятся в *Центр учета запасов* Склада ресурсов и рассчитывается учетная себестоимость этих ресурсов на выходе со склада.
7. В объединяющем классе для СР необходимо предусмотреть ещё одну *функцию, производящую разделение* общего массива с учетными стоимостями ресурсов на массив с учетной себестоимостью ресурсов для СР, массив с учетной себестоимостью ресурсов для СГП и массив с учетной себестоимостью ресурсов для Производства (на схеме эта функция обозначена *красным треугольником с надписью "**split**"*).
8. Полученный в результате разделения массив с учетной себестоимостью ресурсов для СГП передается для ввода в *Центр учета затрат* СГП; массив с учетной себестоимостью ресурсов для СР вводится в *Центр учета затрат* СР, в котором производится расчет добавленной на СР стоимости ресурсов для производства. Полученный в результате расчета массив с добавленной стоимостью суммируется поэлементно с массивом с учетной себестоимостью ресурсов для Производства (например, с помощью функции Sum из модуля Manufact; на схеме обозначена *прямоугольником красного цвета с надписью "**sum**"*). Результат суммирования передается для ввода в *Производство*.
9. Учетная себестоимость ресурсов для производства, включающая в себя добавленную на СР стоимость вводится в *Производство* и рассчитывается учетная себестоимость готовой продукции.
10. Учетная себестоимость готовой продукции вводится в *Центр учета запасов* СГП; учетная себестоимость ресурсов для СГП вводится в *Центр учета затрат* СГП. В Центре учета запасов рассчитывается учетная себестоимость отгружаемой продукции, а в Центре учета затрат рассчитывается добавленная на СГП стоимость отгружаемой продукции. Полученные в результате этих расчетов массив с учетной себестоимостью отгружаемой продукции и массив с добавленной на СГП стоимостью отгружаемой продукции суммируются поэлементно (например, с помощью той же функции Sum). Результат суммирования и представляет собой *искомую учетную себестоимость* отгружаемой со склада готовой продукции с учетом вносимых складами затрат.

ИМПОРТ И ЭКСПОРТ ДАННЫХ

Обмен данными между объектами библиотеки FROMA2 и сторонними программами осуществляется с помощью CSV-файлов. Основные типы, константы, классы и методы, реализующие эти алгоритмы, находятся в модуле Impex. Состав модуля приведен в таблице ниже.

Таблица 16 Состав модуля Impex

Состав модуля	Описание	Заметка

void MeasRestore	Вспомогательная функция	Метод заполняет поля с единицами измерения в малом массиве, беря их из большого массива при совпадении имен в обоих массивах
class clsImpex	Класс для обмена данными между объектами библиотеки FROMA2 и сторонними программами	Экземпляр класса позволяет импортировать данные из CSV-файлов, а также массивов типа strNameMeas и strItem во внутреннее хранилище. Из внутреннего хранилища экземпляр класса позволяет экспортировать эти данные в CSV-файл, в массивы типа decimal, strNameMeas, strItem и *string*. В классе предусмотрена операция транспонирования

КЛАСС clsImpex

Основные задачи, которые можно решать с помощью класса *clsImpex*:

1. Импорт данных из CSV-файла во внутреннее хранилище;
2. Импорт данных из массива типа strNameMeas и strItem во внутреннее хранилище;
3. Транспонирование данных, представленных во внутреннем хранилище матрицей;
4. Экспорт данных из внутреннего хранилища в *CSV-файл*;
5. Выборка данных, заданных начальными и конечными индексами строк и столбцов из внутреннего хранилища (представляющего собой матрицу) и получение этих данных в виде динамического массива вещественного типа decimal; используется для получения численных данных;
6. Выборка данных, заданных начальными и конечными индексами строк и столбцов из внутреннего хранилища и получение этих данных в виде динамического массива типа strItem; используется для получения численных данных;
7. Выборка данных, заданных начальными и конечными индексами строк и столбцов из внутреннего хранилища и получение этих данных в виде динамического массива типа strNameMeas; используется для получения наименований продуктов/ресурсов и единиц их натурального измерения;
8. Выборка данных, заданных начальными и конечными индексами строк и столбцов из внутреннего хранилища и получение этих данных в виде динамического массива типа *string*; используется для получения только наименований продуктов/ресурсов;
9. Получение *числа строк и столбцов матрицы*, являющейся внутренним хранилищем.

ПОЛЯ КЛАССА CLSIMPEX

Таблица 17 Поля класса clsImpex. Секция Private

Поле класса	Описание	Заметка
vector<std::vector<std::string>> m_data	Внутреннее хранилище. Тип vector<std::vector<std::string>>	Хранит импортированные данные в виде прямоугольной матрицы
size_t m_rowcount	Тип size_t. Количество строк матрицы внутреннего хранилища	
size_t m_colcount	Тип size_t. Количество столбцов матрицы внутреннего хранилища	
char separator	Тип *char*. Разделитель между значениями в *CSV-файле*.	

Методы класса clsImpex можно условно разделить на следующие группы:

- Конструкторы, деструктор, перегруженные операторы, сервисные функции;
- Методы импорта;
- Методы преобразования;
- Методы экспорта;
- Get-методы;
- Методы визуального контроля.

Ниже подробно описываются все методы.

КОНСТРУКТОРЫ, ДЕСТРУКТОР, ПЕРЕГРУЖЕННЫЕ ОПЕРАТОРЫ, СЕРВИСНЫЕ ФУНКЦИИ

clsImpex()

Конструктор по умолчанию (без параметров). Создает пустое внутреннее хранилище m_data. Создает и инициализирует переменные: m_rowcount = m_colcount = 0, separator = ';'.

clsImpex(ifstream& ifs, const char& ch)

Конструктор с параметрами. Импортирует данные из файлового потока во внутреннее хранилище.

Параметры:

&ifs – файловый поток, открытый для чтения; тип ifstream;

&ch – разделитель, используемый в файловом потоке &ifs; тип *char*.

clsImpex(const size_t ncount, const nmBPTypes::strNameMeas names[], const nmBPTypes::strItem data[], const size_t dcount, nmBPTypes::ReportData flg)

Конструктор с параметрами. Импортирует данные из двух массивов: массива с наименованиями ресурсов/продуктов и единицами их натурального измерения и массива с данными. Импорт производится во внутреннее хранилище.

Параметры:

ncount – число строк, равное числу элементов массива names; тип *size_t*;

names – указатель на массив с наименованиями строк и единицами измерения; тип strNameMeas;

data – указатель на массив с данными; представляет собой одномерный массив, аналог двумерной матрицы размером ncount*dcount; тип элементов массива strItem;

dcount – число столбцов двумерной матрицы, хранимой в массиве data; тип *size_t*;

flg – флаг, определяющий тип используемых данных: volume, price или value из структуры типа *strItem*; тип параметра ReportData.

~clsImpex()

Деструктор.

clsImpex(const clsImpex& other)

Конструктор копирования.

Параметры:

& other – ссылка на копируемый константный экземпляр класса *clsImpex*.

clsImpex(clsImpex&& other)

Конструктор перемещения.

Параметры:

&& other – перемещаемый экземпляр класса *clsImpex*.

void swap(clsImpex& other) noexcept

Функция обмена значениями между экземплярами класса.

Параметры:

& other – ссылка на обмениваемый экземпляр класса *clsImpex*.

clsImpex& operator=(const clsImpex& other)

Перегрузка оператора присваивания копированием.

Параметры:

& other – ссылка на копируемый константный экземпляр класса *clsImpex*.

clsImpex& operator=(clsImpex &&obj)

Перегрузка оператора присваивания перемещением

Параметры:

&& other – перемещаемый экземпляр класса *clsImpex*.

void reset()

Сбрасывает состояние объекта до значения по умолчанию. Метод vector::clear не гарантирует успешного перераспределения памяти, в настоящем методе используется альтернатива ему. Кроме того, данный метод устанавливает счетчики строк и столбцов внутреннего хранилища в ноль, а сепаратор в значение по умолчанию (';').

bool is_Empty() const

Возвращает *true*, если внутреннее хранилище (контейнер m_data) пустое.

МЕТОДЫ ИМПОРТА

bool Import(ifstream& ifs, const char& ch)

Метод импортирует данные из файлового потока во внутреннее хранилище. В случае успешной операции возвращает *true*.

Параметры:

&ifs – файловый поток, открытый для чтения; тип ifstream;

&ch – разделитель, используемый в файловом потоке &ifs; тип *char*;

bool Import(const size_t ncount, const nmBPTypes::strNameMeas names[], const nmBPTypes::strItem data[], const size_t dcount, nmBPTypes::ReportData flg)

Метод импортирует данные из двух массивов: массива с наименованиями ресурсов/ продуктов и единицами их натурального измерения и массива с данными. Импорт производится во внутреннее хранилище. В случае успешной операции возвращает *true*.

Параметры:

ncount – число строк, равное числу элементов массива names; тип *size_t*;

names – указатель на массив с наименованиями строк и единицами измерения; тип strNameMeas;

data – указатель на массив с данными; представляет собой одномерный массив, аналог двумерной матрицы размером ncount*dcount; тип элементов массива strItem;

dcount – число столбцов двумерной матрицы, хранимой в массиве data; тип *size_t*;

flg – флаг, определяющий тип используемых данных: *volume*, *price* или *value* из структуры типа *strItem*; тип параметра ReportData.

МЕТОДЫ ПРЕОБРАЗОВАНИЯ

void Transpon()

Метод транспонирует матрицу m_data.

МЕТОДЫ ЭКСПОРТА

void csvExport(ofstream& ofs) const

Метод экспорта данных из внутреннего хранилища в *CSV-файл* с разделителем, заданным переменной separator.

Параметры:

&ofs – файловый поток, открытый для чтения; тип ifstream;

GET-МЕТОДЫ

decimal* GetDecimal(const size_t brow, const size_t erow, const size_t bcol, const size_t ecol) const

Метод возвращает выбранный из внутреннего хранилища *фрагмент* данных, конвертированных в тип decimal.

Параметры:

brow – индекс начальной строки выбранного фрагмента данных из внутреннего хранилища; тип size_t;

erow – индекс конечной строки выбранного фрагмента данных из внутреннего хранилища; тип size_t;

bcol – индекс начального столбца выбранного фрагмента данных из внутреннего хранилища; тип size_t;

ecol – индекс конечного столбца выбранного фрагмента данных из внутреннего хранилища; тип size_t.

Метод возвращает указатель на одномерный динамический массив, являющийся аналогом двумерной матрицы размером (*erow-brow*+1)*(*ecol-bcol*+1). Вещественный тип элементов массива определяется псевдонимом decimal.

Для совместимости с параметрами методов классов *clsStorage* и *clsManufactory* матрица внутреннего хранилища (в векторе m_data) должна иметь горизонтальную ориентацию (разные периоды - в разных столбцах). В противном случае перед применением настоящего метода матрицу необходимо транспонировать с помощью метода Transpon.

nmBPTypes::strNameMeas* GetNames(const size_t brow, const size_t erow, const size_t idName, const size_t idMeas) const

Метод возвращает выбранный из внутреннего хранилища *фрагмент* данных, конвертированных в тип strNameMeas. Используется для получения массива с наименованиями ресурсов/ продуктов и единиц их натурального измерения.

Параметры:

brow – индекс начальной строки выбранного фрагмента данных из внутреннего хранилища; тип size_t;

erow – индекс конечной строки выбранного фрагмента данных из внутреннего хранилища; тип size_t;

idName – номер столбца с названиями ресурсов/ продуктов; тип size_t;

idMeas – номер столбца с названиями единиц измерения натурального выражения ресурсов/ продуктов; тип size_t.

Метод возвращает одномерный динамический массив размером (*erow-brow*+1). Тип элементов массива strNameMeas.

Для совместимости с параметрами методов классов *clsStorage* и *clsManufactory* матрица внутреннего хранилища (в векторе m_data) должна иметь горизонтальную ориентацию (разные периоды - в разных столбцах). В противном случае перед применением настоящего метода матрицу необходимо транспонировать с помощью метода Transpon.

string* GetNames(const size_t brow, const size_t erow, const size_t idName) const

Метод возвращает выбранный из внутреннего хранилища *фрагмент* данных, тип которых совпадает с типом данных хранилища, - тип *string*. Используется для получения массива с наименованиями ресурсов/ продуктов или наименованиями единиц их натурального измерения.

Параметры:

brow – индекс начальной строки выбранного фрагмента данных из внутреннего хранилища; тип size_t;

erow – индекс конечной строки выбранного фрагмента данных из внутреннего хранилища; тип size_t;

idName – номер столбца с названиями ресурсов/ продуктов; тип size_t;

Метод возвращает одномерный динамический массив размером (*erow-brow*+1). Тип элементов массива *string*.

Для совместимости с параметрами методов классов *clsStorage* и *clsManufactory* матрица внутреннего хранилища (в векторе m_data) должна иметь горизонтальную ориентацию (разные периоды - в разных столбцах). В противном случае перед применением настоящего метода матрицу необходимо транспонировать с помощью метода Transpon.

strItem* GetstrItem(const size_t brow, const size_t erow, const size_t bcol, const size_t ecol, nmBPTypes::ReportData flg) const

Метод возвращает выбранный из внутреннего хранилища *фрагмент* данных, конвертированных в структурный тип strItem.

Параметры:

brow – индекс начальной строки выбранного фрагмента данных из внутреннего хранилища; тип size_t;

erow – индекс конечной строки выбранного фрагмента данных из внутреннего хранилища; тип size_t;

bcol – индекс начального столбца выбранного фрагмента данных из внутреннего хранилища; тип size_t;

ecol – индекс конечного столбца выбранного фрагмента данных из внутреннего хранилища; тип size_t;

flg – флаг, определяющий тип выбираемых данных: *volume*, *price* или *value* из структуры типа *strItem*; тип параметра ReportData.

Метод возвращает одномерный динамический массив, являющийся аналогом двумерной матрицы размером (*erow-brow*+1)*(*ecol-bcol*+1). Тип элементов массива strItem. Используемые поля заполняются данными из матрицы, неиспользуемые – нулями.

Для совместимости с параметрами методов классов *clsStorage* и *clsManufactory* матрица внутреннего хранилища (в векторе m_data) должна иметь горизонтальную ориентацию (разные периоды - в разных столбцах). В противном случае перед применением настоящего метода матрицу необходимо транспонировать с помощью метода Transpon.

size_t GetRowCount() const

Метод возвращает число строк матрицы внутреннего хранилища; тип size_t.

size_t GetColCount() const

Метод возвращает число столбцов матрицы внутреннего хранилища; тип size_t.

МЕТОДЫ ВИЗУАЛЬНОГО КОНТРОЛЯ

void View(ostream& os) const

Метод визуального контроля импортированной из файла информации. Выводит содержимое внутреннего хранилища в поток типа ostream. Используется в отладочных целях. При выводе каждый элемент матрицы, представляющий собой тип string, обрезается до 15 символов.

Параметры:

&os – выходной поток для вывода информации; тип ostream.

ПРИМЕР ПРОГРАММЫ С КЛАССОМ clsImpex

Ниже представлен листинг программы, демонстрирующей работу класса *clsImpex*.

```
#include <iostream>
#include <fstream>
#include <impex_module.h>
#include <common_values.hpp>

using namespace std;
using namespace nmBPTypes;

int main(int argc, char* argv[]) {
    setlocale(LC_ALL, "Russian");               // Установка русского языка

    /** Импортируем данные из CSV-файла во внутреннее хранилище **/
    ifstream input("ship.csv");                 // Связываем файл с потоком
    const char ch =';';                         // Выбираем разделитель
    clsImpex* Data = new clsImpex(input, ch);   // Импортируем данные из файла
    input.close();                              // Закрываем файл

    /** Визуальный контроль результата импорта **/
    ofstream View_ship("View_ship.txt");        // Связываем файл с потоком
    Data->View(View_ship);                      // Отображаем хранилище
    View_ship.close();                          // Закрываем файл

    /** Проверка работы Get - методов **/
    size_t PrCount   = Data->GetColCount()-2;   // Число периодов проекта
    size_t ProdCount = Data->GetRowCount()-1;   // Число продуктов
    cout << "Число периодов проекта " << PrCount << endl;
    cout << "Число продуктов " << ProdCount << endl;

    strItem* Ship = Data->GetstrItem(1, 6, 2, 61, volume);  // Получаем массив
    if(!Ship) cout << "Ship = nullptr" << endl;
    strNameMeas* ProdNames = Data->GetNames(1, 6, 0, 1);    // Получаем массив
    if(!ProdNames) cout << "ProdNames = nullptr" << endl;

    /** Печатаем в файл полученные массивы в виде таблицы **/
    ofstream Output("Array.txt");               // Связываем файл с потоком
    Output << setw(15) << "Name" << setw(15) << "Measure";  // Заголовки
    for(size_t j{}; j<PrCount; j++)
        Output << setw(15) << j;
    Output << endl;
    for(size_t i{}; i<ProdCount; i++) {                     // Данные
        Output << setw(15) << ((ProdNames+i)->name).substr(0,15) << setw(15) <<
(ProdNames+i)->measure;
        for(size_t j{}; j<PrCount; j++)
            Output << fixed << setprecision(7) << setw(15) <<
(Ship+PrCount*i+j)->volume;
        Output << endl;
    };
    Output.close();                                         // Закрываем файл
```

```
/** Транспонирование матрицы внутреннего хранилища **/
    Data->Transpon();

    /** Визуальный контроль результата транспонирования **/
    ofstream View_ship_T("View_ship_T.txt");    // Связываем файл с потоком
    Data->View(View_ship_T);                    // Выводим результат в файл
    View_ship_T.close();                        // Закрываем файл

    /** Экспорт транспонированной матрицы внутреннего хранилища в CSV-файл **/
    ofstream ship_T("ship_T.csv");          // Связываем файл с потоком
    Data->csvExport(ship_T);                // Экспорт транспонированной матрицы
    ship_T.close();                         // Закрываем выходной файл

    delete Data;                            // Удаляем класс импорта-экспорта
    delete[] Ship;                          // Удаляем массив
    delete[] ProdNames;                     // Удаляем массив
    return 0;
}
```

АРИФМЕТИКА ДЛИННЫХ ВЕЩЕСТВЕННЫХ ЧИСЕЛ

Как было сказано в разделе «ТИП ВЕЩЕСТВЕННЫХ ЧИСЕЛ», вещественные числа в библиотеке FROMA2 по желанию программиста могут быть представлены либо стандартным для C++ типом *double*, либо собственным типом библиотеки *LongReal*, реализация которого представлена в модуле LONGREAL. Модуль *LONGREAL* представлен заголовочным файлом LongReal_module.h и файлом реализации LongReal_lib.cpp. В модуле содержатся классы и методы, реализующие вещественные числа и основные арифметические и логические операции над ними.

Каждый экземпляр класса *LongReal* являет собой вещественное число. Внутреннее представление вещественного числа типа *LongReal* включает в себя знак числа, мантиссу и экспоненту и выражается формулой:

$$sign * 0.d_1d_2...d_n * 10^{exponent},$$

где

sign – знак числа;
d_1, d_2, d_n – цифры мантиссы числа;
exponent – экспонента числа;
знак ^ – знак возведения в степень.

Таблица 18 Состав модуля LongReal

Состав модуля	Описание	Заметка
using Stype = bool	Псевдоним типа	Тип *знака* вещественного числа типа LongReal: *true* для отрицательного и *false* для положительного
using Dtype = short int	Псевдоним типа	Тип цифры *мантиссы* числа (каждая цифра имеет этот тип). Обязательно тип со знаком (знак используется временно при промежуточных вычислениях)
using Etype = int	Псевдоним типа	Тип числа *экспоненты*
const Etype MinEtype	Минимальное значение экспоненты. Тип *Etype*	При выполнении операций с двумя операндами для исключения ситуации с выходом экспоненты результирующего числа за пределы диапазона заданного типа, необходимо ограничивать минимальное значение экспоненты числом *numeric_limits<Etype>::min()/2+1*, а максимальное значение числом *numeric_limits<Etype>::max()/2-1* (например, в 32-разрядной среде эти значения становятся равными -2147483648/2+1 и 2147483647/2-1 соответственно). Для целей использования в библиотеке FROMA2 предельные значения избыточны и ресурсоемки. Поэтому минимальное и максимальное значение экспоненты существенно ограничены. В текущей реализации эти значения взяты равными *-32768* и *32768*.
const Etype MaxEtype	Максимальное значение экспоненты. Тип *Etype*	
const size_t manDigits	Максимальное число знаков мантиссы. Тип *size_t*	При выполнении операций над числами и последующем использовании результатов расчетов в новых операциях, может происходить увеличение количества цифр мантиссы. Если таких операций много, число знаков мантиссы растет неконтролируемо, приводит к неоправданному потребления ресурсов компьютера и увеличению времени вычислений. Введение ограничения на длину мантиссы является компромиссом между точностью с одной стороны и ресурсоемкостью с другой. В текущей реализации длина мантиссы ограничена *64* десятичными знаками. Это не ограничивает количество цифр мантиссы вводимых в операцию чисел, но ограничивает количество цифр мантиссы получаемого результата операции.
enum Scale{NANO=-9, MICRO=-6, MILL=-	Именованные числа. Тип *int*.	Используются для удобства при масштабировании чисел в кило-, мега-, милли-, микро- и т.п. новые числа. Вместо указанных собственных имен можно использовать любые целые числа

3, KILO = 3, MEGA=6, GIGA=9};		
class LongReal	Класс вещественного числа для арифметики длинных вещественных чисел	Экземпляр класса хранит одно число в виде *sign * 0.$d_1d_2...d_n$ * 10^exponent* и методы, реализующие сервисные и арифметические операции с ним.

Помимо указанных в таблице членов, в файле реализации *LongReal_lib.cpp* содержится ряд используемых исключительно в этом файле констант.

КЛАСС LongReal

Возможности экземпляров класса *LongReal*, основные арифметические и логические операции над вещественными числами описаны ранее в разделе «Модуль LongReal» и покрывают все потребности библиотеки FROMA2.

ПОЛЯ КЛАССА LongReal

Таблица 19 Поля класса LongReal. Секция Private

Поле класса	Описание	Заметка
size_t NumCount	Количество цифр (десятичных разрядов) в вещественном числе. Тип *size_t*.	Максимальное значение ограничено константой manDigits
Stype sign	Знак числа (1 для отрицательных и 0 для положительных). Тип переменной определяется псевдонимом Stype.	Значение псевдонима, с которым тестировался класс, *bool*.
Dtype* digits	Указатель на динамический массив цифр длинной *NumCount* для хранения мантиссы числа. Тип элемента массива определяется псевдонимом Dtype.	Значение псевдонима, с которым тестировался класс, *short int*.
Etype exponent	Экспонента числа. Тип элемента массива определяется псевдонимом Etype	Значение псевдонима, с которым тестировался класс, *int*. Максимальное и минимальное значения ограничены константами MaxEtype и MinEtype соответственно.

Методы класса LongReal можно условно разделить на следующие группы:

- Сервисные функции;
- Конструкторы, деструктор, метод обмена;
- Перегрузка операторов присваивания;
- Get – методы;
- Методы масштабирования числа;
- Перегрузка логических и арифметических операторов;
- Перегрузка операторов ввода и вывода;
- Методы сериализации и десериализации;

СЕРВИСНЫЕ ФУНКЦИИ. СЕКЦИЯ PRIVATE

inline void setzero()

Метод вводит в экземпляр объекта число "ноль". Ноль может быть с любым знаком, но по умолчанию создается положительный ноль. Отрицательный ноль может возникнуть только в арифметических операциях, его присвоить нельзя.

Положительным нулем считается следующая комбинация значений полей класса:

sign = false,
NumCount = 0,
digits = nullptr,
exponent = MinEtype.

Отрицательным нулем считается комбинация, у которой *sign = true.*

inline void setposinf()

Метод устанавливает значение числа равное *положительной бесконечности*.

Положительной бесконечностью считается следующая комбинация значений полей класса:

sign = false,
NumCount = 0,
digits = nullptr,
exponent = MaxEtype.

inline void setneginf()

Метод устанавливает значение числа равное *отрицательной бесконечности*.

Отрицательной бесконечностью считается следующая комбинация значений полей класса:

sign = true,
NumCount = 0,
digits = nullptr,
exponent = MaxEtype.

inline void setNaN()

Метод устанавливает число в NaN (Not-A-Number). Значения NaN используются для идентификации неопределенных или непредставимых значений для элементов с плавающей запятой, например, таких как результат деления ноль на ноль.

NaN считается следующая комбинация значений полей класса:

sign = false,
NumCount = 1,
**digits = 256*, (соответствует единице в девятом бите),
exponent = 0.

Если *NaN* установить не удалось (неудачное выделение памяти массиву digits), метод вызывает setposinf и возвращает положительную бесконечность.

void normalize()

Метод *нормализует* число, приводя его к виду *sign * 0.$d_1d_2...d_n$ * 10^exponent*.

inline Dtype* cut(const Dtype _digits[], size_t Num)

Метод возвращает обрезанную копию произвольного массива, указатель на который передан в качестве параметра. В новый массив копируются только заданное число первых элементов исходного массива. Типы элементов исходного и нового массивов совпадают.

Параметры:

_digits[] – указатель на исходный массив; тип элементов массива определяется псевдонимом Dtype;

Num – число первых элементов исходного массива, которые будут скопированы в новый массив; тип *size_t*.

inline void LongReal::Resize(const size_t _n)

Метод обрезает внутренний массив digits, оставляя в нем только заданное число первых элементов.

Параметры:

_n – новый размер массива digits; тип *size_t*.

void init(const string& s)

Метод инициализирует экземпляр класса *LongReal* числом, представленным строкою.

Параметры:

&s – ссылка на строку, состоящую из знака «+» или «-», цифр, одной точки (являющейся разделителем целой и дробной частей числа); тип *string*. Предполагается, что других символов в строке нет.

stringstream getstream() const

Метод возвращает хранимое число в поток типа stringstream. Используется в публичных методах для вывода числа в переменные типа *string*, *long double*, *double*, *float*.

inline bool ovrflw(const Etype& E1, const Etype& E2) const

Метод осуществляет контроль переполнения: возвращает *true*, если при сложении экспонент двух чисел (например, в операции умножения) происходит выход за границы максимального или минимального значений для экспоненты, заданных константами MaxEtype или MinEtype соответственно.

Параметры:

&E1 – ссылка на экспоненту первого числа; тип целого числа определяется псевдонимом Etype;

&E2 – ссылка на экспоненту второго числа; тип целого числа определяется псевдонимом Etype.

LongReal incrE(const Etype& _E) const

Метод проверяет, что новое значение экспоненты, переданное в параметрах, больше, чем текущее. Только в этом случае метод создает копию текущего экземпляра, увеличивает в ней экспоненту до заданного большего значения, не изменяя самого числа (увеличивает экспоненту и одновременно добавляет слева нули в массив digits нового объекта) и возвращает это число. Метод используется в алгоритме выравнивания экспонент и мантисс у пары чисел для дальнейшего проведения с ними арифметических операций.

Параметры:

&_E – новое большее, чем текущее значение экспоненты; тип целого числа определяется псевдонимом Etype.

void incrD(const size_t& _N)

Метод проверяет, что новое значение длины массива digits, переданное в параметрах, больше, чем текущее. Только в этом случае метод увеличивает длину массива *digits* текущего экземпляра до нового большего значения за счет правых нулей. Число не меняется. Метод используется в алгоритме выравнивания экспонент и мантисс у пары чисел для дальнейшего проведения с ними арифметических операций.

КОНСТРУКТОРЫ, ДЕСТРУКТОР, МЕТОД ОБМЕНА

LongReal()

Конструктор по умолчанию, без параметров. Создает экземпляр класса с числом *ноль*.

LongReal(const string& value)

Конструктор с инициализацией числа из строки.

Параметры:

&value – число, представленное строкою; тип *string*. При наличии в строке символов, отличных от символов «+» или «-», цифр и точки, лишние символы удаляются.

LongReal(const double& value)

Конструктор с инициализацией числа из вещественного числа типа *double*. Количество символов мантиссы получаемого числа типа *LongReal* в этом случае ограничено значением std::numeric_limits<double>::max_digits10.

Параметры:

&value – вещественное число; тип *double*.

LongReal(const LongReal& obj)

Конструктор копирования.

Параметры:

&obj – копируемый объект типа *LongReal*.

LongReal(LongReal&& obj)

Конструктор перемещения.

Параметры:

&&obj – перемещаемый объект типа *LongReal*.

~LongReal()

Деструктор.

void swap(LongReal& obj) noexcept

Функция обмена значениями между объектами типа *LongReal*.

Параметры:

&obj – ссылка на обмениваемый экземпляр класса *LongReal*.

ПЕРЕГРУЗКА ОПЕРАТОРОВ ПРИСВАИВАНИЯ

LongReal& operator=(const LongReal &obj)

Перегрузка оператора присваивания копированием объекта типа *LongReal*.

Параметры:

&obj – присваиваемый объект типа *LongReal*.

LongReal& operator=(const double &value)

Перегрузка оператора присваивания копированием объекта типа *double*.

Параметры:

&value – присваиваемый объект типа *double*.

LongReal& operator=(const string &value)

Перегрузка оператора присваивания копированием объекта типа *string*.

Параметры:

&value – присваиваемый объект типа *string*.

LongReal& operator=(LongReal &&obj)

Перегрузка оператора присваивания перемещением объекта типа *LongReal*.

Параметры:

&obj – присваиваемый объект типа *LongReal*.

GET – МЕТОДЫ

size_t Size() const

Метод возвращает размер текущего экземпляра в байтах.

template <typename T>
T Get() const

Шаблонный метод возвращает число в форме *string*, *long double*, *double* или *float*. Экземпляры шаблона:

- **template string Get<string>() const** – определяет возвращаемую величину типа *string*;
- **template long double Get<long double>() const** – определяет возвращаемую величину типа *long double*;
- **template double Get<double>() const** – определяет возвращаемую величину типа *double*;
- **template float Get<float>() const** – определяет возвращаемую величину типа *float*.

string Get(size_t _n) const

Метод возвращает число в виде строки *string* с заданным количеством знаков после разделителя (точки).

Параметры:

_n – число знаков дробной части; тип *size_t*.

string EGet(size_t n) const

Метод возвращает число в виде строки *string* в научной форме: *.ddddEn* (мантисса со степенью).

Параметры:

_n – число знаков дробной части; тип *size_t*.

МЕТОДЫ МАСШТАБИРОВАНИЯ ЧИСЛА

LongReal& Expchange(const Etype _exch)

Метод возвращает объект, у которого экспонента изменена на заданную величину. Для удобства можно использовать вместо числа предопределенные константы из перечисления Scale: NANO=-9, MICRO=-6, MILL=-3, KILO = 3, MEGA=6, GIGA=9.

Метод позволяет использовать в качестве исходных данных числа с очень большими или очень малыми экспонентами, которые невозможно представить в виде чисел типа *long double*, *double* или *float*, а также неудобно представлять строкой из-за слишком большой ее длины. Достаточно ввести мантиссу числа любым доступным методом и затем изменить экспоненту до нужного размера.

Параметры:

_exch – величина, на которую необходимо изменить экспоненту текущего числа; тип параметра определяется псевдонимом Etype.

ПЕРЕГРУЗКА ЛОГИЧЕСКИХ И АРИФМЕТИЧЕСКИХ ОПЕРАТОРОВ

bool isZero() const

Метод возвращает *true*, если число равно *положительному* или *отрицательному* нулю.

bool isposInf() const

Метод возвращает *true*, если число равно Infinity (*положительная бесконечность*).

bool isnegInf() const

Метод возвращает *true*, если число равно *-Infinity* (*отрицательная бесконечность*).

bool isNaN() const

Метод возвращает *true*, если число *неопределено* (равно NaN).

bool operator==(const LongReal& x) const

Оператор *сравнения*. Сравнивает два числа и возвращает *true*, если числа равны. Плюс ноль и минус ноль равны друг другу. Если любой объект сравнения *NaN* то объекты не равны друг другу.

bool operator!=(const LongReal& x) const

Оператор неравенства. Обратный оператор предыдущему.

LongReal operator*(const LongReal& x) const

Оператор арифметического *умножения*. Возвращает результат умножения двух чисел типа *LongReal* в виде *нового* объекта типа *LongReal*.

Параметры:

&x – второй сомножитель для текущего числа; тип *LongReal*.

LongReal operator-() const

Оператор унарного минуса. Возвращает *текущий* объект со знаком минус.

bool operator>(const LongReal& x) const

Оператор сравнения «*больше*». Возвращает *true*, если текущее число больше сравниваемого.

Параметры:

&x – число для сравнения; тип *LongReal*.

bool operator<(const LongReal& x) const

Оператор сравнения «*меньше*». Возвращает *true*, если текущее число меньше сравниваемого.

Параметры:

&x – число для сравнения; тип *LongReal*.

bool operator>=(const LongReal& x) const

Оператор сравнения «*больше или равно*». Возвращает *true*, если текущее число больше сравниваемого или равно ему.

Параметры:

&x – число для сравнения; тип *LongReal*.

bool operator<=(const LongReal& x) const

Оператор сравнения «*меньше или равно*». Возвращает *true*, если текущее число меньше сравниваемого или равно ему

Параметры:

&x – число для сравнения; тип *LongReal*.

LongReal operator+(const LongReal& x) const

Оператор арифметического сложения. Возвращает результат сложения двух чисел типа *LongReal* в виде *нового* объекта типа *LongReal*.

Параметры:

&x – второе слагаемое для текущего числа; тип *LongReal*.

LongReal operator-(const LongReal& x) const

Оператор арифметического вычитания. Возвращает разность между двумя числами типа *LongReal* в виде *нового* объекта типа *LongReal*.

Параметры:

&x – вычитаемое число для текущего уменьшаемого числа; тип *LongReal*.

LongReal& operator-=(const LongReal& x)

Оператор декремента. Возвращает *текущее* число типа *LongReal*, уменьшенное на заданную величину.

Параметры:

&x – вычитаемое число для текущего уменьшаемого числа; тип *LongReal*.

LongReal& operator+=(const LongReal& x)

Оператор инкремента. Возвращает *текущее* число типа *LongReal*, увеличенное на заданную величину.

Параметры:

&x – слагаемое для текущего увеличиваемого числа; тип *LongReal.*

LongReal inverse() const

Вычисление обратного числа текущему. Возвращает результат деления единицы на текущее число типа *LongReal* в виде *нового* объекта типа *LongReal*.

LongReal operator/(const LongReal& x) const

Оператор арифметического деления. Возвращает результат деления двух чисел типа *LongReal* в виде *нового* объекта типа *LongReal*.

Параметры:

&x – делитель для текущего делимого числа; тип *LongReal*.

friend LongReal fabs(const LongReal& x)

Возвращает модуль заданного числа типа *LongReal* в виде *нового* объекта типа *LongReal*. Функция не является членом класса *LongReal*, но объявлена дружественной ему (friend), чтобы иметь доступ к закрытым полям класса.

Параметры:

&x – число, модуль которого требуется найти; тип *LongReal.*

ПЕРЕГРУЗКА ОПЕРАТОРОВ ВВОДА И ВЫВОДА

friend ostream& operator<<(ostream& os, const LongReal& value)

Перегруженный оператор вывода вещественного числа типа *LongReal* в поток ostream. Функция не является членом класса *LongReal*, но объявлена дружественной ему (friend), чтобы иметь доступ к закрытым полям класса.

Параметры:

&os – поток типа *ostream*;

&value – вещественное число типа *LongReal*.

friend stringstream& operator>>(stringstream& ss, LongReal& value)

Перегруженный оператор ввода вещественного числа типа *LongReal* из потока stringstream. Функция не является членом класса *LongReal*, но объявлена дружественной ему (friend), чтобы иметь доступ к закрытым полям класса.

Параметры:

&ss – поток типа *stringstream*;

&value – вещественное число типа *LongReal*.

МЕТОДЫ СОХРАНЕНИЯ СОСТОЯНИЯ ОБЪЕКТА В ФАЙЛ И ВОССТАНОВЛЕНИЯ ИЗ ФАЙЛА

bool StF(ofstream &_outF) const

Метод имплементации записи в файловый поток текущего состояния экземпляра класса (запись в файловый поток, метод сериализации). Название метода является аббревиатурой, расшифровывается “StF” как “Save to File”. При удачной сериализации возвращает *true*, при ошибке записи возвращает *false*.

Параметры:

&_outF - экземпляр класса ofstream для записи данных. При инициализации переменной *_outF* необходимо установить флаг бинарного режима открытия потока (не текстового!) ios::binary.

bool RfF(ifstream &_inF)

Метод имплементации чтения из файлового потока текущего состояния экземпляра класса (чтение из файлового потока, метод десериализации). Название метода является аббревиатурой, расшифровывается “RfF” как “Read from File”. При удачной десериализации возвращает *true*, при ошибке чтения возвращает *false*.

Параметры:

&_inF - экземпляр класса ifstream для чтения данных. При инициализации переменной *_inF* необходимо установить флаг бинарного режима открытия потока (не текстового!) ios::binary.

bool SaveToFile(const string _filename) const

Метод записи текущего состояния экземпляра класса в файл. При удачной записи в файл, возвращает *true*, при ошибке записи возвращает *false*.

Параметры:

_filename – имя файла; тип *string*.

bool ReadFromFile(const string _filename)

Метод восстановления состояния экземпляра класса из файла. При удачном чтении из файла, возвращает *true*, при ошибке чтения возвращает *false*.

Параметры:

_filename – имя файла; тип *string*.

friend bool SEF(ofstream &_outF, LongReal& x)

Перегруженная функция SEF из модуля SERIALIZATION для записи в файловый поток вещественного числа типа *LongReal*. Функция не является членом класса *LongReal*, но объявлена дружественной ему (friend), чтобы иметь доступ к закрытым полям класса.

Параметры:

&_outF - экземпляр класса ofstream для записи данных;

&x – вещественное число типа *LongReal*.

В модуле SERIALIZATION одноименная функция записывает в файловый поток вещественное число типа *double*. Перегрузка этой функции для числа типа *LongReal* позволяет использовать методы сериализации более высокого порядка без оглядки на тип вещественного числа, представленного псевдонимом decimal.

friend bool SEF(ofstream &_outF, LongReal x[], size_t Cnt)

Перегруженная функция SEF из модуля SERIALIZATION для записи в файловый поток массива вещественных чисел *LongReal*. Функция не является членом класса *LongReal*, но объявлена дружественной ему (friend), чтобы иметь доступ к закрытым полям класса.

Параметры:

&_outF - экземпляр класса ofstream для записи данных;

&x – указатель на массив вещественных чисел типа *LongReal*;

Cnt – размер массива вещественных чисел; тип *size_t*.

В модуле SERIALIZATION одноименная функция записывает в файловый поток массив вещественных чисел типа *double*. Перегрузка этой функции для числа типа *LongReal* позволяет использовать методы сериализации более высокого порядка без оглядки на тип элемента массива, представленного псевдонимом decimal.

friend bool DSF(ifstream &_inF, LongReal& x)

Перегруженная функция DSF из модуля SERIALIZATION для чтения из в файлового потока вещественного числа типа *LongReal*. Функция не является членом класса *LongReal*, но объявлена дружественной ему (friend), чтобы иметь доступ к закрытым полям класса.

Параметры:

&_inF – экземпляр класса ifstream для записи данных;

&x – вещественное число типа *LongReal*.

В модуле SERIALIZATION одноименная функция читает из файлового потока вещественное число типа *double*. Перегрузка этой функции для числа типа *LongReal* позволяет использовать методы десериализации более высокого порядка без оглядки на тип вещественного числа, представленного псевдонимом decimal.

friend bool DSF(ifstream &_inF, LongReal x[], size_t Cnt)

Перегруженная функция DSF из модуля SERIALIZATION для чтения из файлового потока массива вещественных чисел *LongReal*. Функция не является членом класса *LongReal*, но объявлена дружественной ему (friend), чтобы иметь доступ к закрытым полям класса.

Параметры:

&_inF – экземпляр класса ifstream для записи данных;

&x – указатель на массив вещественных чисел типа *LongReal*;

Cnt – размер массива вещественных чисел; тип *size_t*.

В модуле SERIALIZATION одноименная функция читает из файлового потока массив вещественных чисел типа *double*. Перегрузка этой функции для числа типа *LongReal* позволяет использовать методы сериализации более высокого порядка без оглядки на тип элемента массива, представленного псевдонимом decimal.

МЕТОДЫ ВИЗУАЛЬНОГО КОНТРОЛЯ

void View(ostream& os) const

Метод выводит в поток типа *ostream* вещественное число типа *LongReal* так, как оно хранится: отдельно знак числа, отдельно экспоненту, отдельно массив, содержащий мантиссу. Например, число 11.6508114157806 будет выведено так:

```
sign= 0
NumCount= 15
exponent= 2
digits are: 116508114157806
```

Параметры:

&os – экземпляр класса *ostream* для вывода числа.

void ViewM(ostream& os) const

Метод, аналогичный предыдущему, но массив с мантиссой числа выводится в столбик.

Параметры:

&os – экземпляр класса *ostream* для вывода числа.

СЕРИАЛИЗАЦИЯ И ДЕСЕРИАЛИЗАЦИЯ ДАННЫХ

Как было сказано в разделе «МОДУЛЬ SERIALIZATION», основная цель функций, представленных в этом модуле, - обеспечить сохранение в файл и восстановление из файла данных в полях экземпляров основных классов библиотеки FROMA2 (сериализация и десериализация). Шаблоны и нешаблонные перегруженные функции для сериализации и десериализации данных типа *LongReal* были описаны ранее в параграфе «МЕТОДЫ СОХРАНЕНИЯ СОСТОЯНИЯ ОБЪЕКТА В ФАЙЛ И ВОССТАНОВЛЕНИЯ ИЗ ФАЙЛА» раздела «АРИФМЕТИКА ДЛИННЫХ ВЕЩЕСТВЕННЫХ ЧИСЕЛ». Шаблоны и нешаблонные перегруженные функции для сериализации и десериализации тривиально копируемых данных описаны ниже. Их сигнатура полностью совпадает с сигнатурой функций для данных типа LongReal для «бесшовного» переключения пользователем с использования данных базового типа *double* на использование данных типа *LongReal*.

template<typename T>
constexpr bool is_srlzd_v = std::is_trivially_copyable<T>::value

Функция с именем *is_srlzd_v* для проверки условия, является ли *тип T* тривиально копируемым типом. Возвращает *value = true*, если *тип T* является тривиально копируемым типом, *value = false* – в противном случае. Функция выполняется на *этапе компиляции* программного кода.

Параметры:

T – тип, используемый в шаблонах функций сериализации и десериализации.

template<typename T=string>
bool SEF(ofstream &_outF, string& str)

Нешаблонная функция сериализации строки в файловый поток.

Параметры:

&_outF – файловый поток типа ofstream;

&str – ссылка на строку, записываемую в поток; тип *string*.

template<typename T, class=std::enable_if_t<is_srlzd_v<T>>>
bool SEF(ofstream &_outF, T x[], size_t Cnt)

Шаблонная функция сериализации массива элементов типа T в файловый поток.

На этапе компиляции программного кода вызывается функция is_srlzd_v, которая проверяет *тип T* элементов массива на условие *тривиальной копируемости*. Если условие не выполняется, компилятор выдает ошибку. В случае, если элементы массива относятся к тривиально копируемому типу, компиляция завершается благополучно.

Параметры:

&_outF – файловый поток типа ofstream;

x[] – указатель на массив с элементами *типа T*;

Cnt – размер массива; тип *size_t*.

template<typename T, class=std::enable_if_t<is_srlzd_v<T>>>
bool SEF(ofstream &_outF, T& x)

Шаблонная функция сериализации переменной типа T в файловый поток.

На этапе компиляции программного кода вызывается функция is_srlzd_v, которая проверяет *тип T* переменной на условие *тривиальной копируемости*. Если условие не выполняется, компилятор выдает ошибку. В случае, если тип переменной относится к тривиально копируемому типу, компиляция завершается благополучно.

Параметры:

&_outF – файловый поток типа ofstream;

&x – ссылка на переменную *типа T*.

template<typename T=string>
bool DSF(ifstream &_inF, string& str)

Нешаблонная функция десериализации строки из файлового потока. Файловый поток должен быть связан с файлом, в котором предварительно была записана строка с помощью метода bool SEF(ofstream&, string&).

Параметры:

&_inF – файловый поток типа ifstream;

&str – ссылка на строку, читаемую из потока; тип *string*.

template<typename T, class=std::enable_if_t<is_srlzd_v<T>>>
bool DSF(ifstream &_inF, T x[], size_t Cnt)

Шаблонная функция для десериализации массива элементов типа T из файлового потока. Файловый поток должен быть связан с файлом, в котором предварительно был записан массив с помощью метода bool SEF(ofstream &_outF, T x[], size_t Cnt).

На этапе компиляции программного кода вызывается функция is_srlzd_v, которая проверяет *тип T* элементов массива на условие *тривиальной копируемости*. Если условие не выполняется, компилятор выдает ошибку. В случае, если элементы массива относятся к тривиально копируемому типу, компиляция завершается благополучно.

Параметры:

&_inF – файловый поток типа ifstream;

x[] – указатель на массив с элементами *типа T*;

Cnt – размер массива; тип *size_t*.

template<typename T, class=std::enable_if_t<is_srlzd_v<T>>>
bool DSF(ifstream &_inF, T& x)

Шаблонная функция для десериализации переменной типа T из файлового потока. Файловый поток должен быть связан с файлом, в котором предварительно был записан массив с помощью метода bool SEF(ofstream &_outF, T& x).

На этапе компиляции программного кода вызывается функция is_srlzd_v, которая проверяет *тип T* переменной на условие *тривиальной копируемости*. Если условие не выполняется, компилятор выдает ошибку. В случае, если тип переменной относится к тривиально копируемому типу, компиляция завершается благополучно.

Параметры:

&_inF – файловый поток типа ifstream;

&x – ссылка на переменную *типа T*.

РОДИТЕЛЬСКИЙ КЛАСС ДЛЯ МОДЕЛИ

На принципиальной схеме модели с одним центром затрат (Рисунок 10) *Склад ресурсов*, *Производство* и *Склад готовой продукции* обведены штрихпунктирной линией с названием «Объединяющий пользовательский класс "Предприятие"». Удобство такого объединения очевидны и для модели с одним центром затрат, и для модели с несколькими центрами затрат (Рисунок 11). Эти удобства были перечислены в разделе «МОДУЛЬ BASEPROJECT». Состав модуля приведен в таблице ниже.

Таблица 20 Состав модуля BaseProject

Состав модуля	Описание	Заметка
enum Tdev{nulldev=nmBPTypes::sZero, terminal, file}	Именованные числа. Тип данных перечисления *enum*; состоит из конечного набора именованных констант типа *size_t*.	Назначение: выбор устройства для вывода отчета (пустое устройство, терминал, файл).
class clsBaseProject	Класс базового проекта	Предоставляет удобство родительского класса (интерфейс) для объединения в себе полей типа clsStorage, clsManufactory и других полей, типы которых определены в библиотеке FROMA2

Остановимся подробнее на классе *clsBaseProject*, представляющем собой родительский класс (интерфейс) для такого объединения.

КЛАСС clsBaseProject

Состав класса *clsBaseProject* представлен в таблице ниже.

ПОЛЯ КЛАССА clsBaseProject

Таблица 21 Поля класса clsBaseProject. Секция Private

Поле класса	Описание	Заметка
string Title	Название проекта. Тип *string*.	Секция Private
struct Descript { string *sComment; size_t sCount; **} About;**	Структура *About* с описанием проекта; тип *Descript* включает в себя два поля: sComment – указатель на массив строк типа *string* с текстовым описанием проекта; sCount – количество элементов этого массива типа *size_t*.	Секция Private
string FName	Переменная с именем файла для чтения/ записи состояния экземпляра класса на диск	Секция Protected
string RName	Переменная с именем файла для вывода отчета в файл	Секция Protected
Tdev Rdevice	Переменная с признаком устройства для вывода отчета	Секция Protected

Методы класса clsBaseProject можно условно разделить на следующие группы:

- Конструкторы, деструктор, метод обмена, метод сброса состояния; Перегрузка операторов присваивания;
- Set – методы;
- Функции вывода отчета;
- Методы сериализации и десериализации;
- *Расчетные методы (в стадии разработки)*[10].

КОНСТРУКТОРЫ, ДЕСТРУКТОР, МЕТОД ОБМЕНА, МЕТОД СБРОСА СОСТОЯНИЯ; ПЕРЕГРУЗКА ОПЕРАТОРОВ ПРИСВАИВАНИЯ

clsBaseProject()

Конструктор по умолчанию. Устанавливаем пустое название проекта, указатель на массив строк с описанием проекта на "пусто", нулевое число строк в описании проекта, пустую строку вместо имени файла для записи/ чтения состояния в файл и имени файла отчета; устанавливает терминал в качестве устройства вывода отчета.

virtual ~clsBaseProject()

Виртуальный деструктор.

clsBaseProject(const clsBaseProject& other)

Конструктор копирования.

[10] По замыслу автора, в класс clsBaseProject в будущем должны войти общие расчётные методы, объединяющие передачу данных между полями класса типа clsStorage, clsManufactory и корректный последовательный вызов расчётных методов этих объектов.

Параметры:

& other – ссылка на копируемый константный экземпляр класса clsBaseProject.

virtual void swap(clsBaseProject& other) noexcept

Функция обмена значениями между экземплярами класса.

Параметры:

& other – ссылка на обмениваемый экземпляр класса clsBaseProject.

clsBaseProject(clsBaseProject&& other)

Конструктор перемещения.

Параметры:

&& other – перемещаемый экземпляр класса clsBaseProject.

clsBaseProject& operator=(const clsBaseProject& other)

Перегрузка оператора присваивания копированием.

Параметры:

& other – ссылка на копируемый константный экземпляр класса clsBaseProject.

clsBaseProject& operator=(clsBaseProject&& other)

Перегрузка оператора присваивания перемещением

Параметры:

&& other – перемещаемый экземпляр класса clsBaseProject.

virtual void Reset()

Метод сброса всей информации в экземпляре класса до "пусто".

SET – МЕТОДЫ

void SetTitle(const string& _title)

Метод ввода титула проекта. Устанавливает поле Title.

Параметры:

&_title – ссылка на строку, содержащую титул проекта; тип ссылки string.

void SetComment(const string* _comment, const size_t& _count)

Метод ввода описания проекта. Устанавливает поле About.

Параметры:

*_comment – указатель на массив строк с описанием проекта; тип указателя string;

&_count – размер массива; тип size_t.

void SetFName(const string _filename)

Метод ввода имени файла для сериализации и десериализации. Устанавливает поле FName.

Параметры:

_filename – имя файла; тип string.

Метод удобно использовать для получения имени файла сериализации из полного имени исполняемого файла программы, находящегося в переменной *argv[0]* метода *main* консольного приложения C++. При подстановке переменной *argv[0]* вместо параметра *_filename*, функция ищет в имени расширение ".exe" и, если оно есть, то заменяет его расширением ".dat". Таким образом, например, из имени “C:\Users\...\ Business_06.exe ” можно получить имя “C:\Users\...\ Business_06. dat”.

Если строка символов _filename не содержит расширение ".exe", новое расширение ".dat" просто добавляется в конце строки.

void SetFName(const string _filename, const string _ext)

Метод ввода имени файла для сериализации и десериализации с явным указанием расширения для имени файла. Устанавливает поле FName.

Параметры:

_filename – имя файла; тип string;

_ext – расширение, добавляемое к имени файла _filename; тип string.

Метод удобно использовать для получения имени файла сериализации с произвольным расширением.

void SetRName(const string _filename)

Метод ввода имени для файла отчёта. Устанавливает поле RName.

Параметры:

_filename – имя файла; тип string.

Метод удобно использовать для получения имени файла отчёта из полного имени исполняемого файла программы, находящегося в переменной *argv[0]* метода *main* консольного приложения C++. При подстановке переменной *argv[0]* вместо параметра *_filename*, функция ищет в имени расширение ".exe" и, если оно есть, то заменяет его расширением ". txt". Таким образом, например, из имени “C:\Users\...\ Business_06.exe ” можно получить имя “C:\Users\...\ Business_06. txt”.

Если строка символов _filename не содержит расширение ".exe", новое расширение ". txt" просто добавляется в конце строки.

void SetDevice(const Tdev& val)

Метод устанавливает выходное устройство для отчета. Устанавливает поле Rdevice.

Параметры:

&val – индекс устройства: *nulldev* - пустое устройство, *terminal* - вывод на экран, *file* - вывод в файл; тип перечисления целых чисел Tdev.

ФУНКЦИИ ВЫВОДА ОТЧЕТА

ОТОБРАЖЕНИЕ ПРОГРЕССА ПРИ ВЫПОЛНЕНИИ РАСЧЕТОВ

Как указывалось в разделе «МОДУЛЬ PROGRESS», присутствующие в нем инструменты позволяют визуализировать ход выполнения расчетов как в однопоточном, так и в многопоточном режимах, как в консольных, так и оконных приложениях. Состав модуля представлен в таблице ниже.

Таблица 22 Состав модуля PROGRESS

Состав модуля	Описание	Заметка
template <typename T> class clsProgress_shell	Оболочка для любого класса прогресс-бара типа *T*.	Данный класс представляет собой интерфейс, который используют объекты библиотеки FROMA2 для подключения сторонних произвольных индикаторов прогресса (например, wxProgressDialog из библиотеки wxWidgets).
class clsprogress_bar	Простой консольный индикатор прогресса	Один из возможных индикаторов прогресса для консольных приложений.

КЛАСС clsProgress_shell

Данный класс является шаблонным и подразумевает явное указание типа класса индикатора, который будет в нем использоваться. Например, для использования прогресс-индикатора из этого же модуля, объявление переменной “x” будет выглядеть как *class clsProgress_shell<clsprogress_bar> x*.

ПОЛЯ КЛАССА clsProgress_shell

Таблица 23 Поля класса clsProgress_shell. Секция private

Поле класса	Описание	Заметка
T* pbar	Указатель шаблонного типа *T* на объект прогресс-индикатор	По умолчанию равен nullptr.
atomic<int> counter	Счетчик выполненных операций; тип atomic<int>.	При создании инициируется нулем. Используется для подсчета выполненных операций в *многопоточных* вычислениях.
int maxcounter	Максимальное число операций; тип *int*.	По умолчанию равен 100.
int stepcount	Шаг, с которым выводится информация о выполненных операциях; тип *int*.	По умолчанию равен 1.

МЕТОДЫ КЛАССА clsProgress_shell

clsProgress_shell(T* _pbar, const int _maxcounter, const int _stepcount)

Конструктор с параметрами.

Параметры:

_pbar – указатель на объект прогресс-индикатор типа *T*; при *_pbar = nullptr* вывод индикатора не производится;

_maxcounter – максимальное число операций; тип *int*;

_stepcount – шаг, с которым выводится информация о выполненных операциях; тип *int*.

void Set_shell(T* _pbar, const int _maxcounter, const int _stepcount)

Метод устанавливает указатель на новый объект прогресс-индикатор и инициализирует поля заново.

Параметры:

_pbar – указатель на объект прогресс-индикатор типа *T*;

_maxcounter – максимальное число операций; тип *int*;

_stepcount – шаг, с которым выводится информация о выполненных операциях; тип *int*.

void Counter_inc()

Метод увеличивает значение поля счетчика counter на заданную величину stepcount. Метод предназначен для внедрения в каждый поток при многопоточном вычислении.

Логика индикации при многопоточных вычислениях следующая. В каждом потоке, где выполняется какая-либо задача, после ее благополучного завершения, вызывается метод *Counter_inc*, который увеличивает счетчик выполненных задач. В основном потоке другой метод (см. ниже метод *Progress_indicate*) считывает изменение счетчика и выводит индикатор на экран. Если в потоке выполняется цикл с вычислениями, то метод *Counter_inc* вызывается внутри тела этого цикла при каждой удачной итерации.

void Progress_indicate()

Метод считывает изменение счетчика выполненных в других потоках операций и выводит индикатор прогресса на экран. Используется для вывода индикатора прогресса в многопоточных вычислениях. При использовании экземпляров объекта типа thread для организации многопоточных вычислений, вызов метода *Progress_indicate* рекомендуется осуществлять в основном потоке после добавления задачи в поток и перед присоединением этого потока к главному (до вызова join).

При значении поля pbar равном nullptr, метод завершает работу без индикации прогресса.

void Counter_reset()

Метод устанавливает поле счетчика counter в ноль. Вызывается перед повторным использованием экземпляра класса *clsProgress_shell*.

void Counter_max()

Метод устанавливает поле счетчика counter в максимальное значение, равное maxcounter. Вызывается при досрочном завершении работы потока (прерывании цикла вычислений). Обеспечивает корректное завершение работы метода *Progress_indicate*.

void Update(int val)

Метод вывода индикатора прогресса на экран при однопоточных вычислениях. При однопоточных вычислениях счетчик counter не используется. Вызывается отрисовка индикатора в соответствии с переданным параметром. Вызов метода располагается в цикле с вычислениями сразу за благополучно завершенной операцией; в качестве параметра передается значение *счетчика цикла*.

Параметры:

val – номер благополучно завершенной операции; совпадает со значением *счетчика цикла* с вычислениями. Тип *Int*.

КЛАСС clsprogress_bar

Данный класс представляет собой вариант прогресс-индикатора для консольных приложений[11]. Вместо использования объекта данного класса для визуализации прогресса вычислений может быть использован экземпляр любого другого класса аналогичного назначения. Основное требование к такому классу – совпадение сигнатуры метода вывода индикатора Update.

ПОЛЯ КЛАССА clsprogress_bar

[11] Класс clsprogress_bar сделан с использованием кода Toby Speight “An alternative approach”, https://codereview.stackexchange.com/questions/186535/progress-bar-in-c

Таблица 24 Поля класса clsprogress_bar. Секция private

Поле класса	Описание	Заметка
static const int overhead = sizeof"[100%]"	Размер заданной константной части строки	
ostream &os	Выходной поток для вывода индикатора	
string message	Сообщение, являющееся началом индикатора	
const string full_bar	Строка, выбранная часть которой выводится как индикатор прогресса	
int maxcount	Максимальное число итераций/ операций, соответствующее 100% выполнению задачи	

МЕТОДЫ КЛАССА clsprogress_bar

void write(double fraction)

Метод выводит в выходной поток индикатор прогресса, соответствующий *текущему состоянию* вычислительной задачи. Метод *private*.

Параметры:

Fraction – дробное число в диапазоне от 0 до 1, соответствующее текущему состоянию выполнения вычислительной задачи; тип *double*. Значение “0” соответствует состоянию перед началом выполнения задачи, значение “100” соответствует завершенной задаче.

clsprogress_bar(ostream &_os, int const line_width, string _message, const char symbol='.', const int _mx=100)

Конструктор с параметрами.

Параметры:

&_os – выходной поток для отображения индикатора; тип ostream;

line_width – ширина (длина строки) индикатора в символах; тип *int*;

_message – сообщение, являющееся началом индикатора прогресса; тип *string*;

symbol – символ заполнителя индикатора прогресса; тип *char*;

_mx – максимальное число итераций/ операций, соответствующее 100% выполнению задачи; тип *int*.

clsprogress_bar(const clsprogress_bar&) = delete

Конструктор копирования; копирование экземпляров класса запрещено.

clsprogress_bar& operator=(const clsprogress_bar&) = delete

Оператор присвоения копированием; присвоение экземпляров класса запрещено.

~clsprogress_bar()

Деструктор.

void Update(int value)

Метод выводит в выходной поток индикатор прогресса, соответствующий *текущему состоянию* вычислительной задачи. Метод *public*.

Параметры:

value – целое число в диапазоне от 0 до *maxcount*, соответствующее текущему состоянию выполнения вычислительной задачи; тип *double*. Значение “0” соответствует состоянию перед началом выполнения задачи, значение *maxcount* соответствует завершенной задаче. Тип параметра *int*.

Название и сигнатура метода совпадает с названием и сигнатурой метода аналогичного назначения класса wxProgressDialog библиотеки wxWidgets.

void Set_maxcount(const int _mx)

Метод устанавливает новое значение максимального числа операций *maxcount*.

Параметры:

_mx – новое значение поля *maxcount*.

КАК ВКЛЮЧИТЬ И ОТКЛЮЧИТЬ ИНДИКАЦИЮ ПРОГРЕССА

При желании включить индикацию прогресса при выполнении расчетов в классе clsStorage, необходимо выполнить следующие действия:

1. Создать экземпляр класса *индикатора прогресса*, например экземпляр класса clsprogress_bar:

```
type_progress* progress = new type_progress(std::clog, 70u, "Working");
```

где *std::clog* – поток для вывода индикатора в консоль;

70u – количество символов индикатора;

"Working" – строка, являющаяся началом индикатора прогресса.

2. Создать экземпляр класса *оболочки* для индикатора прогресса и передать ему указатель на экземпляр класса индикатора прогресса:

```
clsProgress_shell<type_progress>* shell = new clsProgress_shell<type_progress>(progress, RMCount, 1);
```

где *progress* – указатель на ранее созданный экземпляр класса индикатора прогресса;

RMCount – число SKU[12], по каждому из которых будет производится расчет;

1 – шаг шкалы индикатора прогресса.

3. Передать экземпляру класса *clsStorage* (например, с именем *Warehouse*) указатель на экземпляр класса оболочки для индикатора прогресса:

```
Warehouse->Set_progress_shell(shell);
```

4. По окончании работы с индикатором удалить динамические объекты (если оболочка и индикатор создавались, как динамические объекты):

```
delete shell;
shell = nullptr;
delete progress
progress = nullptr;
```

Для отключения ранее включенной индикации можно выполнить следующие *альтернативные* действия:

[12] Stock Keeping Unit - единица складского учета

Вариант 1	Вариант 2	Вариант 3	Вариант 4
Warehouse->Set_progress_shell(nullptr);	Shell-> Set_shell(nullptr);	delete shell; shell = nullptr;	delete progress; progress = nullptr;

Аналогично можно включить индикацию прогресса при выполнении расчетов в классе *clsManufactory*. Если в программе используются объекты обоих классов (clsStorage и clsManufactory), то создавать второй экземпляр класса *индикатора* прогресса и экземпляр класса *оболочки* для индикатора прогресса нет необходимости. В этом случае для включения индикации достаточно передать экземпляру класса *clsManufactory* (например, с именем *Manufactory*) указатель на *существующий* экземпляр класса оболочки для индикатора прогресса. Удаление динамических объектов оболочки и индикатора производится в самом конце программы после того, как они более не нужны.

```
Manufactory ->Set_progress_shell(shell);
```

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1. МОДЕЛЬ ЧАСТИЧНЫХ ЗАТРАТ ПРЕДПРИЯТИЯ, ПРОИЗВОДЯЩЕГО ГОТОВУЮ ЕДУ

Данный пример реализован в виде программного кода для предприятия, производящего готовую еду. Модель реализована в виде консольной программы. Исходные данные вводятся из текстового и CSV-файлов. Соответствующая входная информация и название файла с ней приведены в таблице ниже.

Таблица 25 Исходные данные для программы. Приложение 1

Входные данные	Файл
Название и текстовое описание проекта	about.txt
Названия продуктов и объемы их отгрузок со Склада готовой продукции	ship.csv
Название и цены сырья и материалов в разные периоды времени	rowmatprices.csv
Рецептуры для шести продуктов	recipe_0.csv, recipe_1.csv, recipe_2.csv, recipe_3.csv, recipe_4.csv, recipe_5.csv

Валюта проекта и принцип учета запасов, как редко изменяемые для предприятия параметры, устанавливается в главном методе программы *main* в файле main.cpp.

В основном коде программы (метод *main*, файл main.cpp) для каждого продукта и каждой позиции сырья и материалов также устанавливаются:

- разрешение на отгрузку и поступление в одном периоде,
- разрешение на автоматический расчет поступлений на склад (СГП/ ССМ),
- норматив запаса готовой продукции в долях от объема отгрузок/ закупок.

Результаты расчётов выводятся в *файл отчетов* и *CSV-файлы*.

Название файла отчёта «*clsEnterprise_LR.txt*» или «*clsEnterprise_DB.txt*» в зависимости от типа используемых в программе вещественных чисел: LongReal или *double*.

В таблице ниже приведены названия *CSV-файлов* и соответствующая выходная информация. В названии файла может присутствовать суффикс “*LR*” или “*DB*”, также в зависимости от типа используемых в программе вещественных чисел: *LongReal* или *double*.

Выходные данные	CSV-файл
Объемы отгрузок продуктов в натуральном выражении с СГП	f_ws_volume.csv
Удельная себестоимость отгружаемых с СГП продуктов	f_ws_price.csv
	f_ws_value.csv
	f_wb_volume.csv
	f_wb_value.csv
	f_wp_volume.csv
	f_wp_price.csv
	f_wp_value.csv
	f_rs_volume.csv
	f_rs_price.csv
	f_rs_value.csv
	f_rb_volume.csv
	f_rb_price.csv
	f_rb_value.csv
	f_rp_volume.csv
	f_rp_price.csv
	f_rp_value.csv
	f_mp_volume.csv
	f_mp_price.csv
	f_mp_value.csv
	f_mb_volume.csv
	f_mb_price.csv
	f_mb_value.csv
	f_ms_volume.csv
	f_ms_price.csv
	f_ms_value.csv

Структура каталогов программы.

Пример предоставляется в виде набора файлов с исходными кодами, написанными на C++. Набор файлов пригоден для сборки проекта с помощью утилиты CMake.