**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ**

**БРЯНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ**

**Кафедра**

**" Информатика и программное обеспечение "**

**Отчет по преддипломной практике**

**Реализация мультиметодов на языке с++**

Студент гр. 15-Ивт1(мг)

\_\_\_\_\_\_\_\_\_\_\_\_Куликова Ю. С.

Руководитель к.т.н., доц.:

\_\_\_\_\_\_\_\_\_\_\_\_Израилев В. Я.

Проверил к.т.н., доц.:

\_\_\_\_\_\_\_\_\_Подвесовский А. Г.

Брянск 2017

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc482132176)

[1. Анализ Требований 6](#_Toc482132177)

[1.1. Описание процесса множественной диспетчеризации 6](#_Toc482132178)

[1.2. Типы данных. 8](#_Toc482132179)

[1.3. Обзор и сравнение реализации библиотек мультиметодов. 9](#_Toc482132180)

[1.4. Обзор и сравнение реализации мультиметодов на различных языках программирования. 11](#_Toc482132181)

[1.5. Модель предметной области (AS IS). 13](#_Toc482132182)

[2. Техническое задание 16](#_Toc482132183)

[2.1. Основания для разработки 16](#_Toc482132184)

[2.1.1. Назначение разработки 16](#_Toc482132185)

[1.5.1. Требования к программе 16](#_Toc482132186)

[1.5.2. Требования к программной документации 19](#_Toc482132187)

[1.5.3. Стадии и этапы разработки 19](#_Toc482132188)

[1.5.4. Порядок контроля и приемки 20](#_Toc482132189)

[1.6. Выводы 21](#_Toc482132190)

[3. ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ 22](#_Toc482132191)

[3.1. Цель и задачи исследования 22](#_Toc482132192)

[3.2. Описание исследования 22](#_Toc482132193)

[3.2.1. Литература 22](#_Toc482132194)

[3.3. Паттерн «Визитор» 23](#_Toc482132195)

[3.4. Статический метод 25](#_Toc482132196)

[3.5. Статический метод c симметрией 26](#_Toc482132197)

[3.6. Динамическая диспетчеризация 27](#_Toc482132198)

[3.7. Динамическая диспетчеризация с ассоциативным массивом 28](#_Toc482132199)

[3.8. Выводы и результаты исследования 28](#_Toc482132200)

[4. РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 30](#_Toc482132201)

[4.1. Описание модели ЖЦПО 30](#_Toc482132202)

[4.2. Разработка архитектуры ПО. 31](#_Toc482132203)

[4.3. Функциональные требования 31](#_Toc482132204)

[4.4. Выбор средств разработки 32](#_Toc482132205)

[4.4.1. Язык программирования 32](#_Toc482132206)

[4.5. Обоснование выбора среды программирования 32](#_Toc482132207)

[4.6. Описание нововведений языка С++ 33](#_Toc482132208)

[4.7. Посетитель 35](#_Toc482132209)

[ЗАКЛЮЧЕНИЕ 38](#_Toc482132210)

[СПИСОК ЛИТЕРАТУРЫ 39](#_Toc482132211)

# ВВЕДЕНИЕ

Язык программирования C++ не имеет встроенной реализации множественной диспетчеризации (хотя поддержка множественной диспетчеризации рассматривается) поэтому для расширения виртуальной функции используются различные обходы. Методы обхода аналогичны другим языкам: либо использование паттерна посетителя, либо динамического приведения типов (Runtime Type Information).

Мультиметод - это общий случай виртуального метода. Вызов виртуального метода будет зависеть от конкретного типа или значения объекта [6]. Вызов же мультиметода зависит от нескольких участвующих в вызове объектов, так же, Страуструп рекомендует по возможности отдавать предпочтение свободным функциям, вместо внутренних методов-членов, если такой функции достаточно публичного интерфейса класса.

**Актуальность** работы заключается в том, что алгоритм мультифункций, не был включен в стандартную библиотеку. На 2017 год, C++ поддерживает только одиночную диспетчеризацию, хотя поддержка множественной диспетчеризации рассматривается.

**Целью** работы реализация нескольких алгоритмов мультиметода, используя инструменты С++ (11–17 версии).

Поставленная в работе цель достигается путем решения следующих **задач**:

* анализ предметной области;
* составление технического задания для разрабатываемой системы;
* исследование алгоритмов мультиметодов и выделение закономерностей;
* проектирование и реализация системы по описанным требованиям;
* тестирование работоспособности разработанного приложения.

**Объектом** исследования является множественная диспетчеризация.

**Предметом** исследования является реализация алгоритмов мультифункций на языке с++.

В качестве **методов исследования** были рассмотрены объектно-ориентированный язык программирования, объектно-ориентированный инструмент проектирования.

**Теоретическая значимость** диссертационной работы заключается   
в нахождении оптимального алгоритма множественной диспетчеризации.

С точки зрения **практической значимости** диссертационной работы разработана библиотека, с помощью которой можно использовать мультифункции, реализации, которых нет в самом языке.

1. Анализ Требований

## **Описание процесса множественной диспетчеризации**

Мультиметод имеет большое прикладное значение. Определить, когда следует использовать мультиметод, очень просто. Допустим, в программе есть операция, манипулирующая несколькими полиморфными объектами с помощью указателей или ссылок на их базовые классы. Необходимо модифицировать операцию в соответствии с динамическими типами этих объектов.

Классическим примером мультиметодов является задача пересечения геометрических фигур. Например, при решении задачи на столкновения машин. Требуется реализовать функцию, проверяющую, пересечение грузовой и легковой машины (машин может быть больше). Параметрами данной функции являются объекты базового типа (например, Car). Другими словами, параметры машин неизвестны (скорость, вес) на момент компиляции программы. Поскольку в каждом пересечении участвуют два объекта, простые виртуальные функции не решают задачу.

Существует два основных подхода к решению данной проблемы. Первый состоит в разработке максимально обобщенной функции пересечения машин, что требует написания алгоритмически очень сложного и длинного кода. Другой подход состоит в написании простых функций, пересекающих заранее известные типы машин, например, легковая и грузовая машина, легковая и пикап и т.д. Очевидно, что написание таких точечных функций не сопряжено с решением сложных алгоритмических задач. Проблема состоит в механизме диспетчеризации вызовов этих точечных функций.

Мультифункции как раз и предоставляют необходимый для решения подобных задач механизм диспетчеризации.

Некоторые языки программирования, например Common Lisp [9], поддерживают мультифункции напрямую. Однако наиболее распространенные промышленные объектноориентированные языки программирования, такие как C++, Java и C#, их не поддерживают.

Существует два основных подхода к обеспечению поддержки мультифункций:

* подход, основанный на динамическом расширении типа;
* подход, основанный только на стандартных средствах используемого языка.

Ниже приводится краткое описание способа реализации мультифункций в языке C++ (11–й версии) с использованием динамического расширения типов [24].

Проще всего реализовать диспетчеризацию на основе двойного переключения по типу. Для этого нужно попытаться последовательно выполнить динамическое приведение типа первого объекта к каждому из типов объектов, которые могут стоять в левой части выражения. Для каждой ветви нужно сделать то же самое со вторым аргументом. После обнаружения типов для обоих объектов становится ясно, какую функцию следует вызвать.

Подход заставляет писать много строк кода, и эффективность уменьшается при добавлении новых классов. Так же важна последовательность кода, что добавляет сложность для понимания и поддержания функций.

Наилучшим способом автоматизации прежнего подхода, будет добавление диспетчера. Можно реализовать списки типов, которые включают в себя списки алгоритмов для манипуляции коллекцией типов. При этом общее правило таково: наиболее отдаленные потомки базового типа должны находится в начале списка типов. Затем рекурсивный шаблон генерирует дерево, в каждой ветви которого происходит динамическое приведение типов и сравнение каждого объекта в списке. Теперь можно работать с разными коллекциями.

Можно дополнить данный алгоритм хэш-таблицей, которая будет хранить в себе хэш – данные о типе и указатель на нужную функцию [18]. С помощью сдвига битов возможно объединение параметров функции и вызов нужного метода. Что поможет увеличить скорость работы программы.

Нужно выбрать функцию, с помощью которой можно будет произвести динамическое преобразование аргументов функции. Если в момент регистрации уже известны типы, для которых функция будет активирована и базовый класс не входит в иерархию классов несколько раз, то уже излишне повторять проверку типов с помощью dynamic\_cast, если простой оператор static\_cast позволяет достичь тех же результатов за меньшее время.

Достоинства и недостатки предлагаемого подхода

Выделим достоинства предлагаемого подхода по сравнению с существующими:

* используется механизм RTTI (Run-Time Type Identification);
* простота использования;
* не используются нестандартные библиотеки;

Основным недостатком подхода является высокая нагрузка на компилятор, так же нечеткие ошибки при неправильном использовании программы.

Предлагаемый подход позволяет эмулировать поддержку мульфункций в языке программирования C++. При этом, предлагаемый подход использует только механизм RTTI, что позволяет уменьшить количество кода.

## **Типы данных.**

При работе с языками, в которых типы данных различаются во время компиляции, выбор среди доступных вариантов функций может происходить во время компиляции. Создание таких вариантов альтернативных функций для выбора во время компиляции обычно называют перегрузкой функций [18].

Для того чтобы иметь возможность использовать несколько вариантов подпрограммы с одним и тем же именем, но с разным числом аргументов или другими типами аргументов (то есть с разной сигнатурой, так как список аргументов — часть сигнатуры), вводится перегрузка подпрограмм. Такая перегрузка возможна в рамках процедурной парадигмы, без применения объектно-ориентированного программирования.

При трансляции происходит контроль одноимённых процедур и функций, чтобы они различались по сигнатуре, так как в этом случае транслятор может однозначно определить вызов нужной подпрограммы.

Чтобы исключить ошибку программиста, давшего случайно имя подпрограмме, которое уже используется, вводится дополнительное требование написания ключевого слова. Так сделано, например, в языке Delphi (ключевое слово overload) [14].

В языках программирования, которые определяют типы данных во время выполнения программы (позднее связывание), выбор среди вариантов функций должен происходить во время выполнения на основе динамически определяемых типов аргументов функций. Функции, альтернативные реализации которых выбираются таким образом, обычно называются мультиметодами.

Существуют некоторые затраты во время выполнения программы, связанные с динамической диспетчеризацией вызовов функций. В некоторых языках различие между перегрузкой функций и мультиметодами могут быть размыты, при этом компилятор определяет, может ли выбор вызываемой функции быть произведён во время компиляции, или потребуется более медленная диспетчеризация во время выполнения программы.

## **Обзор и сравнение реализации библиотек мультиметодов.**

Мультифункция - это общий случай виртуального метода. Его вызов будет зависеть от конкретного типа или значения объекта. Вызов же мультифункции зависит от нескольких участвующих в вызове объектов.

Языки программирования могут поддерживать мультиметоды через встроенные средства, либо через препроцессоры, либо через расширения библиотек. В этой главе мы рассмотрим библиотечные и небиблиотечные подходы для C++.

**Cmm** [20] – реализация мультиметода на основе препроцессора и генерации кода, на языке C ++. Код в данной реализации является открытым. Cmm доступен в двух версиях версиях, одна из них использует RTTI для того, чтобы найти динамически тип объекта. Другой подход, достигает своей цели с помощью виртуальной функции, переопределенной в каждом классе. О невозможности определения типа, сообщается путем сброса исключения.

**DoubleCpp** [32, 36] – другой подход на основе препроцессора для мультиметодов диспетчеризация по двум параметрам. Реализован на основе паттерна поситетель. DoubleCpp требует доступа к файлам, содержащим определения классов, чтобы добавить соответствующие методы accept и visit.

DoubleCpp, как и любой другой подход, основанный на посетителях, не сообщает об ошибке, связанной с неопределенностью типов, разрешая ее самостоятельно.

**Локи** [21] – библиотека шаблонов программирования, написанная Андреем Александреску. Она предоставляет несколько различных диспетчеров, которые сохраняют баланс между скоростью, гибкостью и размером кода. В данный момент, библиотека поддерживает мультиметоды с двумя аргументами, за исключением статического диспетчера, который допускает больше аргументов. Статический диспетчер обеспечивает разрешение вызова на основе перегрузки, но требует ручного управления иерархией классов, чтобы обнаружить родителя объекта. Все остальные диспетчеры не рассматривают иерархические отношения и требуют явного разрешения всех возможных случаев. В документах вспомогательных функций допускается открытая передача метода. В данный момент, в сообществе, обсуждается идея по расширению данного подхода.

**Yomm11** – эта библиотека реализует открытые мультиметоды для C ++ 11 [40]. Характерными особенностями являются:

Синтаксис: нет никаких ограничений на количество виртуальных аргументов. Виртуальные и не виртуальные аргументы могут быть добавлены произвольно. Виртуальное и множественное наследование  поддерживается.

Таблицы диспетчеризации разбиты на группы классов, что приводит отсутствии избыточности. Поддержка «чужих» иерархий: библиотека может быть использована без внесения изменений в существующий код.

## **Обзор и сравнение реализации мультиметодов на различных языках программирования.**

В Perl 6 [26], как и в предыдущих его версиях, используются протестированные идеи из других языков и системы типов, предлагающие убедительные преимущества в анализе кода со стороны компилятора и мощной семантике путём множественной диспетчеризации. В нём есть как мультифункции, так и мультиподпрограммы. Так как большинство операторов на самом деле является подпрограммами, есть также операторы с мультифункцией. Логотип языка находится на рис. 1.1.



***Рис. 1.1. логотип языка «Perl 6»***

Один из первых широко известных языков для поддержки мульти-методов был CLOS. CLOS линеаризует иерархию классов и использует асимметричную семантику отправки, чтобы избежать конфликтов и неоднозначности. Cecil [29] рассматривает нечеткое определение двусмысленности, как потенциальный источник для ошибок. Поэтому он использует симметричную семантику отправки и обходится без линеаризации иерархии объектов для выявления ошибок во время компиляции. MultiJava [20] реализует модель, которая позволяет раздельно компилировать и устраняет необходимость проверки типа ссылок во время ссылки, но также ограничивает расширяемость.

В языке с поддержкой мультиметодов, таком, как Common Lisp [9], код написать намного проще. При наличии мультиметодов традиционный подход к определению методов в классах и хранению их в объектах становится менее привлекательным, поскольку каждый перегружаемый метод относится к двум различным классам, а не к одному. Таким образом, специальный синтаксис для вызова функции, в общем случае, исчезает, так что вызов функции выглядит точно так же, как и вызов обычного метода. Логотип языка находится на рис. 1.3.

******

***Рис. 1.3. логотип языка «Common Lisp»***

В Python [35] уже есть инструменты для реализации мультиметода. С помощью модуля multipledispatch получается намного более простой синтаксис и скорость увеличивается в несколько раз. Функционально это очень похоже на пример с CLOS, но синтаксис соответствует стандартному синтаксису языка программирования. Логотип языка находится на рис. 1.2.



***Рис. 1.2. логотип языка «Python»***

## **Модель предметной области (AS IS).**

Создание современных приложений характеризуется концентрацией сложности на начальных этапах анализа требований и проектирования спецификаций системы. Именно на этом этапе приходит понимание того, как должна функционировать будущая система, чтобы удовлетворить предъявляемые к ней требования. Нечеткость требований допущенные при анализе и проектировании, приводят к провалу всей работы.

Преодолеть сложности начальных этапов создания системы призван структурный системный анализ, который характеризуется тем, что строится наглядная и модель системы.

Для моделирования бизнес-процессов используется несколько различных методов.

* Метод функционального моделирования SADT (IDEF0). Функциональная модель SADT отображает функциональную структуру объекта, т.е. производимые им действия и связи между этими действиями [2].
* Моделирование потоков данных DFD. Требования представляются в виде иерархии функциональных компонентов (процессов), связанных потоками данных. Цель моделирования – продемонстрировать, как каждый процесс преобразует входные данные в выходные.
* Диаграмма вариантов использования (Use Case). Определяет поведение системы с точки зрения пользователя. Показывает набор элементов Use Case, актеров и их отношений [2].

Для моделирования данной системы была выбрана методология Use Case, исходя из следующих характеристик:

* диаграммы вариантов использования предназначены для отображения внешнего функционирования проектируемой системы и ее взаимодействия с внешним миром, пользователями;
* главная задача таких диаграмм – выяснение требований к системе на начальных этапах проектирования, когда решаются наиболее общие задачи предназначения разрабатываемой системы.

Результатом моделирования бизнес-процессов является модель «AS-IS (как есть)» существующего состояния системы. Ее анализ позволяет понять, где находятся наиболее слабые места, в чем будут состоять преимущества новых бизнес-процессов и насколько глубоким изменениям подвергнется существующая структура. Найденные в модели «AS-IS» недостатки можно исправить при создании модели «TO-BE (как должно быть)».

Диаграмма вариантов использования системы представлена на рис. 1.3.

Основным действующим лицом (актером) является библиотека мультиметодов, которая выполняет два основных действия:

* выводит ошибку, при отсутствии реализации перегруженной функции;
* отображение объекта на функциональность;

Библиотека объектом занимается хранения объектов и их свойств, а приложение занимается взаимодействием с пользователем.

|  |  |
| --- | --- |
| C:\Users\Julia\AppData\Local\Microsoft\Windows\INetCache\Content.Word\11.png | ***Рис. 1.3. «Диаграмма вариантов использования»*** |

1. Техническое задание

В данной работе необходимо разработать программную систему множественной диспетчеризации.

* 1. Основания для разработки

Основанием для разработки программной системы множественной диспетчеризации является задание на диссертационную работу, выданное к.т.н., доц. Израилевым В. Я.

* + 1. Назначение разработки

Назначение разработки– увеличение скорости и качества разработки программ, использующие мультиметоды.

Для того чтобы оценить, насколько часто мультиметоды используются на практике, Мушевичи с соавторами исследовал программы, использующие динамическую диспетчеризацию [18]. Они проанализировали девять приложений, в основном компиляторов, написанных на шести различных языках программирования: C++, Lisp, Cecil, MultiJava, Dylan, Diesel и Nice. Результаты показывают, что от 13% до 32% обобщенных методов используют динамическую типизацию 1 аргумента, в то время как от 2,7% до 6,5% методов используют динамическую типизацию нескольких аргументов.

Поэтому готовое решение, поможет уменьшить время разработки, за счет существования готового решения.

### Требования к программе

**Требования к функциональным характеристикам.** Разрабатываемая система обеспечивать возможность выполнения перечисленных ниже функций:

* принимать больше, чем два параметра;
* возможность независимого подключения;
* возможность использования любого типа входного параметра;
* параметр может быть ссылкой или значением;
* Консольное приложение должно удовлетворять следующим требованиям:
* программа должна содержать подсказки на русском языке, для удобства навигации пользователя;
* организовано общение с пользователем, на русском языке;
* отображение времени работы библиотеки;

**Требования к надежности.** Важность обеспечения надежности разрабатываемой системы является очевидной.

Надежность информации должна обеспечиваться с точки зрения:

* физической целостности, т.е. наличия или отсутствия искажений или уничтожения элементов этой информации;
* доверия к информации, т.е. наличия или отсутствия в ней подмены (несанкционированной модификации) ее элементов при сохранении целостности;
* безопасности информации, т.е. наличия или отсутствия несанкционированного получения ее лицами или процессами, не имеющими на это соответствующих полномочий.

Физическую целостность необходимо обеспечить посредством проведения следующих мероприятий:

* конфигурирование аппаратных и программных средств в соответствии с техническими требованиями;
* использование источников бесперебойного питания.

**Условия эксплуатации.** Для нормальной работы компьютерного оборудования и разрабатываемой системы необходимо соблюдение условий эксплуатации. Эксплуатационные режимы работы не должны превышать значений, указанных в технической характеристике оборудования. Прочие условия должны соответствовать санитарным нормам и правилам эксплуатации компьютеров. Программная система учета и анализа списков поступающих в вуз рассчитана на обычного пользователя ПК.

**Требования к составу и параметрам технических средств.** В состав технических средств должен входить IВМ-совместимый персональный компьютер (ПЭВМ), выполняющий роль сервера, включающий в себя:

* процессор Intel или AMD 2.0hz;
* оперативную память объемом, 1Гигабайт;
* жесткий диск объемом 200 Мб.

На ПК обязательно должна быть установлены:

* лицензионная операционная система семейства Windows или Linux;

Пользователь может быть не подключен к сети Интернет.

### Требования к программной документации

Для облегчения использования разрабатываемого программного продукта, необходимо составить подробную техническую документацию. Разрабатываемая система должна сопровождаться руководством пользователя.

**Руководство пользователя.** Руководство пользователя содержит следующую информацию:

* описание функций, выполняемых программой;
* описание пользовательского интерфейса.

### Стадии и этапы разработки

**Стадии разработки.** Разработка должна быть проведена в три стадии:

**1 стадия** – анализ;

**2 стадия** – проектирование;

**Этапы разработки.** На стадии анализа должен быть выполнен этап разработки, согласования и утверждения технического задания.

На стадии проектирования должны быть выполнены перечисленные ниже этапы работ:

1. разработка программы;
2. разработка программной документации;
3. тестирование программы.

На стадии внедрения должен быть выполнен этап подготовки и передачи программы.

**Содержание работ по этапам.** На этапе разработки технического задания должны быть выполнены перечисленные ниже работы:

1. постановка задачи;
2. определение требований к техническим средствам;
3. определение требований к программе;
4. определение стадий, этапов разработки программы и документации на нее;
5. согласование и утверждение технического задания .

На этапе разработки программы должны быть выполнены перечисленные ниже работы:

1. выбор архитектуры программной системы;
2. выбор средств разработки (среда разработки, язык программирования);
3. разработка программного обеспечения;
4. отладка программной системы.

На этапе разработки программной документации должна быть выполнена разработка программных документов в соответствии с требованиями к составу документации.

## Выводы

**Целью** работы реализация нескольких алгоритмов мультиметода, используя инструменты С++ (11–17 версии).

Поставленная в работе цель достигается путем решения следующих **задач**:

* анализ предметной области;
* составление технического задания для разрабатываемой системы;
* исследование алгоритмов мультиметодов и выделение закономерностей;
* проектирование и реализация системы по описанным требованиям;
* тестирование работоспособности разработанного приложения.

**Объектом** исследования является множественная диспетчеризация.

**Предметом** исследования является реализация алгоритмов мультифункций на языке с++.

В качестве **методов исследования** были рассмотрены объектно-ориентированный язык программирования, объектно-ориентированный инструмент проектирования.

**Теоретическая значимость** диссертационной работы заключается   
в нахождении оптимального алгоритма множественной диспетчеризации.

С точки зрения **практической значимости** диссертационной работы разработана библиотека, с помощью которой можно использовать мультифункции, реализации, которых нет в самом языке.

1. ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ
   1. Цель и задачи исследования

**Цель** исследования – выявить взаимосвязь направлений подготовки, которые указывали абитуриенты при подаче заявлений входе приёмной кампании в Брянском государственном техническом университете в 2016 году.

Чтобы лучше понять поведение абитуриентов необходимо попытаться разделить их на типовые группы, определить основные варианты поведения абитуриентов и установить взаимосвязь между группами и вариантами поведения.

С задачей разбиения множества объектов на группы (кластеры) на основе некоторой меры сходства отлично справляются методы и алгоритмы кластеризации.

К **задачам**, путем решения которых достигается поставленная цель, можно отнести:

* выбор объект анализа;
* исследование задачи кластерного анализа;
* изучение и выделение существующих методов для анализа;
* применение этих методов для выявления особенностей выбора направлений подготовки абитуриентами БГТУ [34].
  1. Описание исследования
     1. Литература

В своей книге "Дизайн и эволюция C++" [33] Страуструп упоминает, что ему нравится концепция мультиметодов и что он рассматривал возможность их реализации в C++, но утверждает, что ему не удалось найти пример эффективной (в сравнении с виртуальными функциями) их реализации и решить некоторые возможные проблемы неоднозначности типов. Он далее утверждает, что, хотя было бы хорошо реализовать поддержку этой концепции, она может быть приближенно реализована двойной диспетчеризацией или таблицей поиска на основе типов, как описано в примере на C/C++ выше, поэтому эта задача имеет низкий приоритет при разработке будущих версий языка.

Мультиметоды можно сделать с использованием RTTI и dynamic\_cast'ов. Пример описан в статье "Новый подход с использованием шаблонов и RTTI" [34]. В данном методе, есть проблема при описание новых фигур.

Можно обойтись без RTTI и dynamic\_cast'ов. Например, в статье "MultiMethods in C++: Finding a complete solution" [27]. Но код, в данном случае, получается довольно большой по размеру.

Про мультиметоды можно почитать у Александреску в "Современном проектирование на С++" [21], глава 11.

* 1. Паттерн «Визитор»

Паттерн Посетитель является частным случаем множественной диспетчеризации. Шаблон демонстрирует классический приём восстановления информации о потерянных типах, не прибегая к понижающему приведению типов [37].

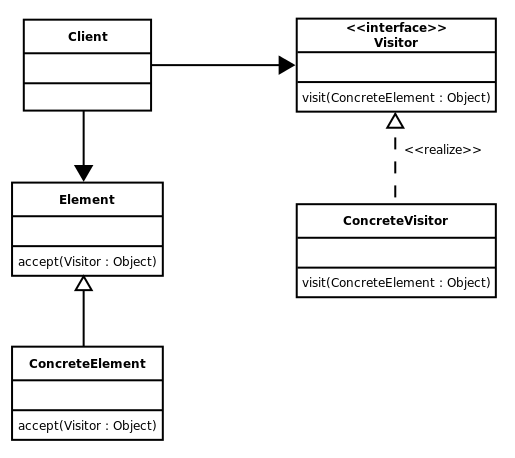
Он позволяет определить операцию для объектов других классов без изменения этих классов. При использовании паттерна Посетитель определяются две иерархии классов: одна для элементов, для которых надо определить новую операцию, и вторая иерархия для посетителей, описывающих данную операцию.

Основным назначением паттерна Visitor является введение абстрактной функциональности для совокупной иерархической структуры объектов "элемент", а именно, паттерн Visitor позволяет, не изменяя классы Element, добавлять в них новые операции. Для этого вся обрабатывающая функциональность переносится из самих классов Element (эти классы становятся "легковесными") в иерархию наследования Visitor.

При этом паттерн Visitor использует технику "двойной диспетчеризации". Обычно при передаче запросов используется "одинарная диспетчеризация" – то, какая операция будет выполнена для обработки запроса, зависит от имени запроса и типа получателя. В "двойной диспетчеризации" вызываемая операция зависит от имени запроса и типов двух получателей.

Реализация метода представлена ниже.

* Добавление метода accept(Visitor) в иерархию «элемент».
* Создание базового класса Visitor и определение методов visit() для каждого типа элемента.
* Создание производных классов Visitor для каждой операции, исполняемой над элементами.
* Клиент создаёт объект Visitor и передаёт его в вызываемый метод accept().



***Рис. 2.1. «Паттерн Посетитель»***

В случае использования классического паттерна проектирования Посетитель возникает циклическая зависимость между Посетителем и посещаемыми объектами.

Чтобы объявить интерфейс Посетителя, необходимо знать список посещаемых объектов. В свою очередь, чтобы объявить посещаемый объект, необходимо знать интерфейс посетителя. Отложенный Диспетчер позволяет частично разорвать эту связь и дает возможность объявить и реализовать посещаемые объекты до того, как будет известен интерфейс Посетителя.

Интерфейс Отложенного Диспетчера и его конкретные реализации генерируются на основе списка типов посещаемых объектов. Отложенный

Диспетчер используется для определения конкретного типа посещаемого объекта. Каждый конкретный диспетчер принимает в виде шаблонного параметра шаблона тип получателя информации о типе посещенного объекта, что позволяет вкладывать диспетчеры один в другой. Данный механизм позволяет определить тип n объектов, участвующих в вызове n-арного мультиметода. В качестве последнего, nH1-го уровня, указывается класс, позволяющий на основе уже известной информации о типах всех участвующих в вызове мультиметода объектах, вызвать наиболее подходящую специализацию.

Нужно написать реализацию данного метода на языке С++ более поздней версии, так как в основных книгах использует стандарт 1998 года.

* 1. Статический метод

Данный метод, имеет не большое количество кода. Статистический метод имеет в шаблоне шесть параметров.

1. Executer – это тип объекта, который выполняет нужное нам действие - в нашем примере (в главе номер 2), это пересечение машин.
2. FirstType, SecondType являются характеристиками базовых объектов.
3. FirstBaseType, SecondBaseType – это списки типов, содержащие возможные производные типы для предыдущих двух аргументов.

Функция статической диспетчеризации выполняет динамический приведение к первому типу, найденному в списке типов FirstBaseType. Если динамическое приведение не выполнимо[5], то функция переходит к хвосту FirstBaseType (с помощью рекурсии).

Конечным результатом является то, что функция выполняет набор инструкций if-else, которые применяют dynamic\_cast к каждому типу в списке типов. Когда найдено первое совпадение, то главная функция вызывает SecondBaseType, который аналогичным образом находит второй параметр.

* 1. Статический метод c симметрией

Пересечения двух машин может зависеть от того, как именно пересекаются фигуры: гоночная машина сталкивается с грузовиком, либо наоборот. Иногда это не имеет значения, и функция в обоих случаях должна оставаться одинаковой. В этом случае нужен симметричный мулътиметод, который не зависит от порядка следования его аргументов.

Симметрия применяется, только если типы обоих параметров идентичны (в нашем случае класс FirstType совпадает с классом SecondType, a FirstBaseType совпадает с классом SecondBaseType).

Метод в прошлом разделе, реализованный в рамках грубого подхода, является асимметричным. Это значит, что он не поддерживает симметричные мультиметоды. Даже несмотря на то, что с точки зрения класса, не имеет никакого значения, какой из этих параметров является левым, а какой— правым, метод настаивает на том, чтобы объекты передавались в определенном порядке.

Поменяв аргументы местами, и применив другую перегрузку в клиентском коде, можно исправить этот недостаток. В идеале класс должен был бы сам предоставлять эту возмож­ность с помощью дополнительного булевского шаблонного параметра. Для этого нуж­но, чтобы в некоторых ситуациях класс менял порядок следования аргументов при обратном вызове.

* 1. Динамическая диспетчеризация

Проблема метода статической диспетчеризации в том, что в этом случае придется перебирать все случаи, пока приведение типов не покажет, что мы достигли желаемого результата. Кроме того, излишняя сложность в применении шаблонов, и увеличенное время компиляции также относятся к недостаткам этого метода.

Метод динамической диспетчеризации предполагает использование некоторой коллекции для хранения вызываемых методов [18]. Мы можем выбирать метод поиска в этой коллекции. Для больших коллекций возможно выбрать хеширование, для небольших — бинарный поиск. Это дает нам большую гибкость и может значительно повысить скорость поиска по сравнению со статической диспетчеризацией.

Выбор ключа в этой коллекции — важный аспект. Благодаря конструкции typeid() мы имеем возможность узнать TypeInfo для произвольного объекта [21]. Причем, что является ключевым моментом, позволяющим применять этот метод, операция typeid() – полиморфна. Эта значит, что если мы имеем ссылку на тип базового класса, которая на самом деле указывает на производный, typeid() вернет TypeInfo соответствующее производному. Это позволяет «распознать» настоящий тип. TypeInfo - отличный кандидат для построения составного ключа, количество элементов которого — будет равно количеству аргументов в хранимых функциях.

Одна из проблем которую здесь приходится решать — это как сохранить в одной коллекции методы с различными типами вызываемых параметров. Решение с «трамплинными функциями» позволяет «запаковать» методы с различными параметрами внутри методов с одинаковыми параметрами. Это позволяет хранить их в одной коллекции и пользовательский код так же не нуждается в приведении типов.

Вторая проблема — вызов симметричных функций, так же может быть легко решена. Действительно, часто бывает так, что отношение между двумя объектами симметрично, и в этом случае не желательно в пользовательском коде объявлять дополнительную функцию с аргументами в обратном порядке с вызовом функции с аргументами в прямом порядке. Шаблонный параметр simmetric = true приводит к добавлению в коллекцию функций, обратной функции, что решает эту проблему.

* 1. Динамическая диспетчеризация с ассоциативным массивом

Это самый быстрый диспетчер, однако, для его применения нужно модифицировать классы (добавить макрос в каждый класс, к которому применяется диспетчеризация). Для такой диспетчеризации нужно выполнить два виртуальных вызова, набор числовых проверок и операцию доступа к элементу матрицы.

* 1. Выводы и результаты исследования

В этой главе были отображены идеи основных диспетчеров, которые были ограничены двойной диспетчеризацией. Так же обсудили паттерн «Посетитель», основанный на основных принципах ООП.

Объект, выполняющий вызов соответствующей функции, называется двойным диспетчером. Существует несколько типов диспетчеров.

* Статический диспетчер. Он полагается на информацию о статических типах (предоставленную в виде списка типов) и выполняет линейный развернутый поиск соответствующего типа. Как только требуемый тип найден, диспетчер вызывает перегруженную функцию-член для обработки объекта.
* Диспетчер, управляемый ассоциативным массивом. Он использует ассоциативный массив, ключами которого являются объекты класса std: :type\_info. В этом массиве хранятся обратные вызовы (либо указатели на функции, либо функторы). Алгоритм распознавания типа выполняет бинарный поиск.
* Диспетчер, управляемый матрицей. Быстрый диспетчер с модификацией классов.

На основе указанных выше диспетчеров можно реализовать следующие функциональные возможности.

* Автоматизированное преобразование типов. Для вызова диспетчеров необходимо выполнить приведение базовых типов к производным.
* Симметрия. Некоторые варианты использования двойной диспетчеризации являются симметричными по своей природе. Они применяются к одинаковым типам, и порядок следования параметров для них не важен. Реализация симметрии в библиотеке позволяет уменьшить размер клиентского кода и избежать многих ошибок.

В следующей главе, происходит программная реализация данных пунктов на языке С++ 11–17 версии.

1. РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ
   1. Описание модели ЖЦПО

В качестве модели жизненного цикла была выбрана «Каскадная модель (водопад)».

Алгоритм данного метода, представлен на рис.3.1.

|  |
| --- |
|  |
| ***Рис. 3.1. Схема каскадной модели жизненного цикла программного обеспечения*** |

Каскадная модель обладает следующими преимуществами:

* последовательное выполнение этапов проекта в строгом фиксированном порядке;
* позволяет оценивать качество продукта на каждом этапе .

Положительные стороны применения каскадного подхода согласуются с этапами выполнения дипломного проектирования. Это плюс для нас.

Для большинства проектов недостатком может быть отсутствие обратных связей между этапами. Недостатком для нас это не является, поэтому был выбран этот тип модели.

* 1. Разработка архитектуры ПО.

При разработке структуры программного продукта (ПП) необходимо сформулировать критерии ее формирования. Важнейшими критериями являются модифицируемость, корректность кода и удобство управления разработкой ПП, обеспечение возможности контролируемого изменения конфигурации, состава и функций компонент с сохранением основы структурного построения базовых версий ПП. В зависимости от особенностей проблемной области критериями при выборе архитектуры ПП могут быть также: эффективное использование памяти или производительности реализующей ЭВМ, трудоемкость или длительность разработки и другие показатели.

• Выбранная архитектура приложения – «монолитная».

• Тип приложения – насыщенное клиентское приложение.

* 1. Выбор средств разработки

Для разработки был выбран язык программирования С++.

* + 1. Язык программирования

В качестве языка программирования выбран C++ [21]. Он имеет следующие преимущества:

* простота;
* объектная ориентированность;
* типовая защищенность;
* скорость разработки;
* большая популярность среди разработчиков.

С++ предоставляет множество возможностей, некоторые из них:

* полная поддержка классов и объектно-ориентированного программирования, включая наследование реализации и интерфейсов, виртуальные функции и перегрузку операций;
* согласованный и четко определенный набор базовых типов;
* автоматическая очистка динамически распределяемой памяти.
  1. Обоснование выбора среды программирования

Microsoft Visual Studio Community 2017 – последняя версия на текущий период известного пакета библиотеки. Данное обеспечение является необходимым всех современных и устаревших систем. От актуальности директории зависит, насколько правильно будут работать приложения, их запуск и совместимость с текущей ОС. В новой версии 2017 года реализован полностью язык программирования «С++17», что дает значительно больше преимуществ софту. Наличие Microsoft Visual Studio Community 2017 на компьютере необходимо, а в некоторых случаях неотъемлемо.

Пакет устанавливается из одного файла запуска, после чего через рабочий лаунчер можно проверять и обновлять компоненты. Также в текущей сборке уже вшиты «NET Framework», «JavaScript», «Visual Basic» и т.д. Это обеспечивает полное техническое обновление системы, позволяет аппаратной части работать эффективнее. Достаточно скачать Visual studio 2017 community и установить единым пакетом. В таком случае у пользователя не будет проблем с реализацией или с совместимостью новой программы. Поддерживается мультиплатформенность, перенос данных на другую систему, обновление из указанной папки. Компоненты «Visual studio» для Виндовс позволяют использовать на максимум аппаратную составляющую системы, благодаря умным алгоритмам и усовершенствованным технологиям.

Есть минус в данной IDE, если версия продукта устарела, тогда новые программы, приложения и игры не смогут работать. У них просто не будет информации о том, как использовать текущую систему. Приложения не «увидят» процессор последнего поколения и «не узнают», как с ним обращаться.

Установщик на русском языке прост и понятен. Достаточно одного клика для начала распаковки, после чего все временные файлы будут удалены в целях экономии места. Рекомендуется сохранить пакет данных отдельно, он может пригодиться и будет уже в наличии.

Основные характеристики представлены ниже (табл.3.1).

Таблица 3.1

*Основные свойства Visual Studio 2017* *Community*

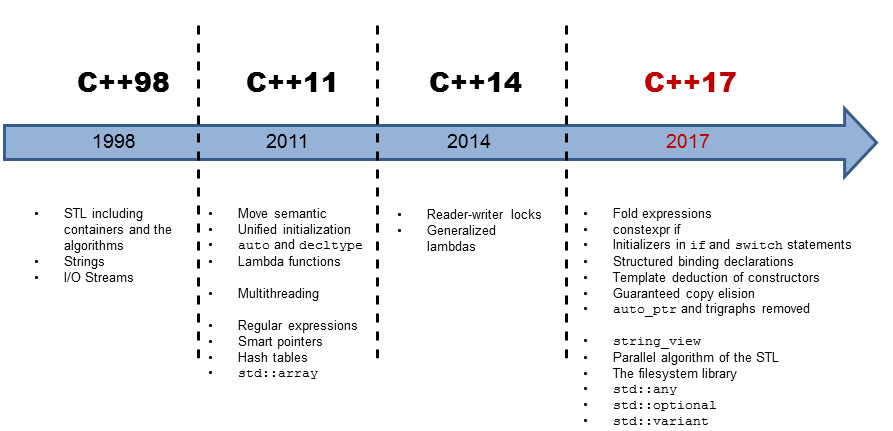
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Разработчик | Процессор | Разрядность | ОП | Лицензия | Язык |
| Microsoft Corporation | 1 GHz | 32 и 64 bit | 1024 Mb | Бесплатно | Русский |

* 1. Описание нововведений языка С++

C++11 – это стандарт языка С++ утвержденный Международной организацией по стандартизации (ISO) в 2011-м году [39]. Предыдущие версии стандарта обычно называют C++98 или C++03. Различия между С++98 и С++03 настолько специфические и их настолько мало, что на них можно не обращать внимания (рис. 3.2.).

Вот несколько новых объявленных возможностей языка С++ :

1. ссылки на временные объекты и семантика переноса (rvalue reference);
2. обобщённые константные выражения (ключевое слово constexpr);
3. внешние шаблоны — возможность запретить компилятору инстанцировать шаблон в единице трансляции (extern template class);
4. ключевое слово auto для задания типа переменной на этапе компиляции;
5. цикл for по коллекции данных;
6. lambda-функции;
7. введена отдельная константа нулевого указателя nullptr;
8. шаблоны с переменным числом параметров ([variadic templates](http://en.wikipedia.org/wiki/Variadic_templates));
9. thread-local хранилище, модель памяти с поддержкой потоков;
10. изменения в стандартной библиотеке: включение hash tables, регулярных выражений, smart pointers, [элементов синхронизации потоков](http://en.wikipedia.org/wiki/C%2B%2B11#Threading_facilities)



***Рис. 3.2 Новые возможности языка C++***

* 1. Посетитель

До появления C++11 шаблоны (классов или функций) могли принимать только заданное число аргументов, определяемых при первоначальном объявлении шаблона. C++11 позволяет определять шаблоны с переменным числом аргументов любого типа [34]. В данном разделе, мы будем использовать, метод «трех путей». Этот метод включает в себя: внешнее объявление, объявление и частичная специализация шаблона. Его цель заключается в использовании VARIADIC параметров, для реализации циклических событий.

Этот пример объясняет легкий способ реализации шаблона посетителя:

|  |
| --- |
| // Первая часть  template<typename... Types>  struct NewVisitor;    // Вторая часть  template<typename First, typename... Types>  struct NewVisitor<First, Types...>: NewVisitor<Types...>  {  using NewVisitor<Types...>::visit;  virtual void visit(const First&) = 0;  };    // Третья часть  template<typename First>  struct NewVisitor<First>  {  virtual void visit(const First&) = 0;  }; |

***Код. 3.7.1 Часть реализации шаблона «посетитель»***

Объявление NewVisitor является рекурсивным. Этот рекурсивность должен быть остановлена, поэтому нужно использовать третью часть программы, объявление шаблона с одним параметром. Код выше, объявляет чисто виртуальные методы. Теперь можно посмотреть на функции, которые используют данный шаблон.

|  |
| --- |
| // Объявление  struct FirstParam;  struct SecondParam;    /// Основной класс  struct BaseVisit  {  typedef NewVisitor<FirstParam, SecondParam> MyVisitor;    virtual void accept(MyVisitor&) = 0;  };    /// Double  struct FirstParam: BaseVisit  {  double x;    void accept(BaseVisit::MyVisitor& v)  {  v.visit(\*this);  }  };    /// Int  struct SecondParam: BaseVisit  {  int x;    void accept(BaseVisit::MyVisitor& v)  {  v.visit(\*this);  }  };    /// Действие посетителя  struct Show: BaseVisit::MyVisitor  {  void visit(const FirstParam& x)  {  std::cout << "FirstParam: " << x.x << std::endl;  }    void visit(const SecondParam& x)  {  std::cout << "SecondParam: " << x.x << std::endl;  }  }; |

***Код. 3.7.2 Часть реализации шаблона «посетитель»***

Тело MyVisitor является абстрактным и он содержит две чистые виртуальные методы, один для FirstParam и второй для SecondParam.

Основной может выглядеть следующим образом:

|  |
| --- |
| int main(int, char\*\*)  {  PrintVisitor vis;  FirstParam md;  md.x = 1.2;  SecondParam mi;  mi.x = 3;    Base\* b = &md;  b->accept(vis);  b = &mi;  b->accept(vis);  } |

***Код. 3.7.3. Вызов шаблона «посетитель»***

Вывод:

|  |
| --- |
| 1.2  3 |

***Код. 3.7.4 Вывод итоговых значений***

«Variandic» ничего нового не принёс, ни в стратегию, ни в архитектуру. Но при этом, кода стало намного меньше, что упрощает понимание шаблона проектирования.

# ЗАКЛЮЧЕНИЕ

Результатом выполнения магистерской диссертации является разработка мультиметодов на языке С++.

В рамках диссертационной работы был произведен подробный анализ предметной области, рассмотрена проблема формирования списков. В работе был проведен сравнительный анализ программ учета и обработки заявлений абитуриентов, выделены их достоинства и недостатки, которые были учтены при разработке системы.

В ходе исследования были составлены требования к программной системе, построены ее проектные модели.

В рамках разработки были соблюдены все требования, описанные в задании на разработку. Программная система была протестирована в различных условиях, в результате чего были выявлены и устранены некоторые существующие неполадки нарушавшие работу системы.

Основные результаты работы и научные выкладки представлялись на следующих научных конференциях:

1. Название: «Мультифункции в языке программирования С++». Номер издания: 6, март 2017 г. , ISSN 2409-1677, Наименование издания: Журнал «Высшая школа», главный редактор Хисматуллин Д.Р.
2. Название: «Сравнение мультиметодов в разных языках программирования». Номер издания: 4, апрель 2017 г. , ISSN 2541-8041, Наименование издания: Журнал «Методы науки», главный редактор Хисматуллин Д.Р.

СПИСОК ЛИТЕРАТУРЫ

1. Мультифункции в языке программирования С++ [Журнал] / авт. Ю.С. Куликова // Высшая школа / ред. Д.Р. Хисматуллин. - Омск : [б.н.], 6 марта 2017 г.. - ISSN 2409-1677.

1. Применение UML 2.0 и шаблонов проектирования. Введение в объектно-ориентированный анализ, проектирование и итеративную разработку [Конференция] / авт. Ларман Крэг. - СПб : [б.н.], 2013. - Т. 2.
2. Программирование: принципы и практика использования C++ [Книга] / авт. Страуструп Бьёрн. - 2011. - стр. 1248.
3. Сравнение мультиметодов в разных языках программирования [Журнал] / авт. Ю.С. Куликова // Методы науки / ред. Д.Р. Хисматуллин. - [б.м.] : ISSN 2541-8041, 4 апреля 2017 г..
4. Язык программирования C++ [Книга] / авт. Страуструп Бьёрн. - СПб : 3, 1999. - стр. 991.
5. Draft proposal for adding multimethods to C++ [Книга] / авт. Smith J.. - 2003.
6. Update on the C++-0x Language Standard [Книга] / авт. Caves Jonathan. - 2010.
7. A calculus for overloaded functions with subtyping [Книга] / авт. Castagna Giuseppe и Longo Giorgio Ghelli & Giuseppe. - 1995. - стр. 115-135.
8. ANSI Common Lisp [Книга] / авт. Graham Paul. - СПб : Символ-Плюс, 1995. - стр. 448.
9. Auto – A Necessary Evil? [Конференция] / авт. Orr Roger. - 2013.
10. C – A Reference Manual [Книга] / авт. III Samuel P. Harbison. - [б.м.] : 5. - стр. 251.
11. C++ Coding Standard. [Книга] / авт. Martin Lockheed. - 2005.
12. C++ Front End [Журнал] / авт. Group Edison Design. - 2006 г..
13. C++ The Complete Reference Third Edition [Книга] / авт. Schildt Herbert. - Osborne McGraw-Hill : 3, 1998.
14. Clang - C++98, C++11, and C++14 Status [Конференция] / авт. Clang.llvm.org. - 2014.
15. Decltype and auto, Programming Language C++ [Журнал] / авт. Stroustrup Bjarne и Gregor Douglas. - 2003 г..
16. Double dispatch in C++ [Журнал] / авт. L. Bettini S. Capecchi, and B. Venneri. - 2006 г.. - стр. 581 – 613.
17. Efficient Multimethods in a Single Dispatch Language [Книга] / авт. B. Foote R. Johnson, and J. Noble. - Scotland, : [б.н.], 2005.
18. Efficient multiple and predicated [Журнал] / авт. Chen C. Chambers and W.. - New York : [б.н.], 1999 г.. - стр. 238–255.
19. General Constant Expressions for System Programming Languages [Журнал] / авт. Reis Gabriel Dos и Stroustrup Bjarne. - 2010 г..
20. Modern C++ Design: Generic Programming and [Книга] / авт. Alexandrescu A.. - 2001.
21. Modular statically typed multimethods [Книга] / авт. Chambers T. D. Millstein and C.. - UK : [б.н.], 1999. - стр. 279-303.
22. MultiJava:modular open classes and symmetric multiple dispatch for Java [Книга] / авт. C. Clifton G. T. Leavens, C. Chambers, and T. Millstein. - New York : ACM, 2000. - стр. 145.
23. Multiple Dispatch [Журнал] / авт. Pescio Dr. Carlo. - 1998 г..
24. Multiple dispatch in practice [Книга] / авт. Muschevici R.. - Nashville : [б.н.], 2008. - стр. 563–582.
25. Multiple dispatch in practice [Книга] / авт. Muschevici Radu [и др.]. - Nashville : [б.н.], 2008. - стр. 563–582.
26. Multipledispatch [В Интернете]. - http://www.eclipse.org/xtend/#multipledispatch.
27. N4661 Editors' Report - Programming Languages - C++ [В Интернете] / авт. Smith R.. - 2017 г.. - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4661.html.
28. Object-oriented multi-methods in Cecil [Журнал] / авт. Chambers C.. - London : [б.н.], 1992 г..
29. Optimizing multi-method dispatch using compressed dispatch tables [Журнал] / авт. E. Amiel O. Gruber, and E. Simon. - New York : [б.н.], 1994 г.. - стр. 244–258.
30. Programming languages C++ [Книга] / авт. American National Standards Institute. - 1998.
31. Reconciling encapsulation and dynamic dispatch via accessory functions [Книга] / авт. Wonnacott C. B. Flynn and D.. - 1999. - стр. 387.
32. The Design and Evolution of C++ [Книга] / авт. Stroustrup Bjarne. - Indianapolis : [б.н.], 1994.
33. The Itanium C++ ABI [В Интернете] / авт. Technical report. - 2001 г..
34. The Python Language Reference Manual [Журнал] / авт. Rossum G. van. - 2003 г..
35. Using accessory functions to generalize dynamic dispatch in single-dispatch object-oriented languages [Книга] / авт. Wonnacott D.. - [б.м.] : USENIX COOTS, 2001. - стр. 93–102.
36. Visitor Pattern Versus Multimethods [В Интернете]. - http://nice.sourceforge.net/visitor.html.
37. Visual C++ 2010: полный курс [Книга] / авт. Хортон Айвор. - 2010.
38. We have C++14! [Конференция] / авт. Sutter Herb. - 2014.