Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования

**ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ
(ТУСУР)**

Кафедра компьютерных систем в управлении и проектировании
(КСУП)

А.Е. Горяинов

Объектно-ориентированное программирование

Методическое пособие

2016

Горяинов А.Е.

Объектно-ориентированное программирование: Методическое пособие / А.Е. Горяинов. – Томск: Томск. гос. ун-т систем упр. и радиоэлектроники, 2016. – 126 с.

Пособие содержит краткое описание лекций по курсу «Объектно-ориентированное программирование» и методические указания к его изучению для направления направлений 220400.62 «Управление в технических системах» и 230100.62 «Информатика и вычислительная техника» профиль САПР.

Содержание

Введение 5

Конспект лекций 6

Лекция #1. Вводная лекция. Базовые элементы языка Си++ 6
Содержание 6
Ключевые определения 9
Литература по теме лекции 9

Лекция #2. Типы данных, управляющие операторы, указатели 10
Содержание 10
Ключевые определения 10
Литература по теме лекции 12

Лекция #3. Функция, массивы, строки 12
Содержание 12
Ключевые определения 13
Литература по теме лекции 14

Лекция #4. Структуры, перечисления. Динамическая память 14
Содержание 14
Ключевые определения 14
Литература по теме лекции 16

Лекция #5. Динамические структуры данных 16
Содержание 16
Ключевые определения 17
Литература по теме лекции 17

Лекция #6. Объекты и классы 17
Содержание 17
Ключевые определения 19
Литература по теме лекции 20

Лекция #7. Инкапсуляция, наследование 20
Содержание 20
Ключевые определения 21
Литература по теме лекции 22

Лекция #8. Виртуальные функции, полиморфизм 22
Содержание 22
Ключевые определения 23
Литература по теме лекции 24

Лекция #9. Шаблоны 24
Содержание 24
Ключевые определения 25

Литература по теме лекции 25

Лекция #10. Практические задачи по проектированию 26
Содержание 26
Ключевые определения 26
Литература по теме лекции 27

Лекция #11. Критерии качества кода. Рефакторинг 27
Содержание 27
Ключевые определения 28
Литература по теме лекции 28

Лекция #12. Основы тестирования 28
Содержание 28

Лекция #13. Повышение производительности работы 29
Содержание 29

Лекция #14. Экзамен 29
Заключение 30
Список обязательной литературы 31
Список дополнительной литературы 32

Введение

Данное методическое пособие составлено для бакалавров направления 220400.62 «Управление в технических системах» и 230100.62 «Информатика и вычислительная техника» профиль САПР для прохождения курса «Объектно-ориентированное программирование». В данном пособии содержится краткий конспект лекций, а также методические рекомендации по изучению отдельных тем.

Отличием данного пособия является смещение акцента в лекциях с изучения языка программирования Си++ на практические задачи, связанные с проектированием программ. Курс предполагает, что синтаксические особенности языка программирования подробно рассматриваются в рамках лабораторных занятий, а также в дополнительной рекомендуемой литературе. Однако наибольшее затруднение у студентов вызывает непонимание базовых концепций объектного подхода и также конкретных ситуаций и задач, где эти концепции должны быть применены. Таким образом, еще одним отличием данного курса лекций является подача материала по принципу «Описание проблемы – Пример проблемы в практике – Техника программирования – Решение проблемы – Изучение полученного результата».

Данный курс предполагает, что студент в достаточной степени владеет знаниями в области алгоритмизации, способен прочесть блок-схемы, а также способен к самостоятельному составлению алгоритмов.

В ходе лекций не рассматриваются вопросы технической реализации объектного подхода в языке Си++ по причине маловажности данной темы в современных условиях разработки прикладного программного обеспечения.

Изучение дисциплины также предполагает значительную самостоятельную работу студентов с литературой.

Конспект лекций

В данном разделе представлены конспекты лекций по курсу «Объектно-ориентированное программирование», перечень методической литературы для изучения обозначенных тем, а также перечень ключевых понятий, необходимых для понимания материала.

Лекция #1. Вводная лекция. Базовые элементы языка Си++

Содержание

Введение в специальность проектирования. Изготовлению любого технического объекта предшествует проектирования. Проектирование предполагает построение технического объекта и расчет всех его характеристик на бумаге. И только после того, как требуемые технические характеристики достигнуты, чертежи объекта передаются на изготовление. Однако выполнять чертежи и схемы на бумаге неудобно, в чем можно легко убедиться на дисциплине «Инженерная графика». Таким образом, становятся востребованными *системы автоматизированного проектирования* – комплекс программ, направленный на упрощение работы проектировщика за счет автоматизации типовых задач. Благодаря данным программам проектировщик выполняет чертежи за компьютером, хранит документы в электронном виде, большинство необходимых расчетов отдает на выполнение компьютеру. Таким образом, разработка САПР, с одной стороны – критически важная специальность для развития технических направлений в стране и мире, с другой стороны – требует серьезной подготовки как программиста, так как САПР являются одними из сложнейших разновидностей программ. Курс объектно-ориентированного программирования рассматривает общие вопросы разработки программ, все последующие дисциплины («ЛиПоСАПР», «МиМАПР», «РСАПР», «Методы оптимизации») будут посвящены непосредственно различным ас-

пектам разработки САПР. Дисциплина «Объектно-ориентированное программирование» является ***основой всей специальности*** «Разработка САПР». ***Каждый диплом*** выпускника данной специальности является ***программным продуктом***.

Введение в специальность автоматизации. Любое производство стремится к повышению эффективности. Чем меньше затрат требует изготовление единицы продукции (любой продукции), тем эффективнее производство. Одним из способов повышения эффективности является автоматизация. Благодаря автоматизации объём сложной и тяжелой работы, которая выполнялась раньше людьми, теперь может выполняться машинами и техническими устройствами. Автоматизация присутствует повсеместно в нашей жизни: в сложных системах городского освещения, в системах управления транспортом и метро, системах городского водоснабжения или перегонки нефти и газа через всю страну. Автоматизация встречается и в мелочах – автоматически открываемые двери в магазинах, автоматически включаемые и выключаемые электрополотенца, системы автоматического пополнения счета телефона. Автоматизация повышает уровень комфорта человека. Для разработки систем автоматизации специалист должен владеть знанием сборки электрических цепей, знанием специальной аппаратуры, таких как датчики, исполняемые механизмы и микроконтроллеры. И также специалист должен уметь писать программы и алгоритмы для всей описанной техники, должен уметь создавать пользовательские интерфейсы, должен уметь создавать базы данных об объектах автоматизации. Если попытаться оценить, то ***треть вашей будущей профессии заключается в программировании***. Курс объектно-ориентированного программирования позволит вам изучить основные принципы разработки программного обеспечения. Полученные знания потребуются в дисциплинах «Базы данных», «Системное программное обеспечение», «Моделирование систем» и также для дипломных проектов.

Представление рейтинговой системы. Представление необходимой литературы (см. Список рекомендуемой литературы). Краткое содержание дисциплины (см. Содержание).

Сложность программных продуктов растет с каждым годом. Проекты средней сложности в настоящее время исчисляются миллионами строк кода, что делает проблемным ориентирование в программе программистами – сложно отслеживать ошибки, сложно добавлять новую функциональность, сложно новому программисту присоединяться к незнакомому проекту. Таким образом, главным приоритетом в разработке становится ***управление сложностью*** программы. Для управления сложностью разрабатываются целые методологии организации программного кода. Объектно-ориентированное программирование – одна из методологий или ***парадигм*** программирования, целью которой является уменьшение сложности программы за счет строгой организации кода.

Внутренняя организация кода не важна ни для конечного пользователя (пользователь видит только интерфейс программы), ни для компьютерного процессора (процессор просто выполняет последовательности команд). ООП не заставляет работать программы быстрее или стабильнее. ООП не увеличивает продажи программы. Но ООП нужна для разработчиков, чтобы упростить разработку, тестирование и поддержку программ. Чем проще и организованнее устроена программа, тем проще будет дальнейшая её разработка.

Разработка 98% современных программ ведется на основе ООП, следовательно, ООП должен знать каждый современный разработчик.

Изучение ООП будет выполняться на основе языка Си++.

Построчный разбор программы «Hello, World!» на Си++. Этапы сборки программы: препроцессинг, компиляция, компоновка.

Ключевые определения

***Среда разработки* (*интегрированная среда разработки, IDE*)** – комплекс программных средств, используемый программистами для разработки программного обеспечения. Среда разработки обязательно включает в себя текстовый редактор, компилятор, отладчик, но может содержать и иные полезные для разработки утилиты. Примеры сред разработки: Visual Studio, Eclipse, Borland C++, IntelliJ IDEA.

Препроцессор – (в языке Си++) программа, подготавливающая код программы к компиляции за счёт выполнения директив препроцессора.

Компиляция – процесс перевода исходного кода программы, составленной на языке высокого уровня, в низкоуровневый (близкий к машинному) объектный код для дальнейшей компоновки. Результатом компиляции являются файлы формата *.obj.

Компилятор – программа, выполняющая компиляцию. В случае синтаксических ошибок в исходном коде программы, компилятор прерывает компиляцию и, как правило, возвращает перечень обнаруженных ошибок.

Компоновка – процесс перевода объектного кода в машинный код, и сборка на основе полученного кода исполняемого модуля. Результатом компоновки является исполняемый файл формата *.exe или *.dll.

Компоновщик – программа, выполняющая компоновку.

Директива препроцессора – (в языке Си++) командная строка в исходном коде, имеющая следующий формат: #ключевое_слово параметры.

Сборка – последовательное выполнение препроцессинга, компиляции и компоновки средой разработки с целью получения исполняемого модуля программы из исходного кода.

Литература по теме лекции

[Гради Буч] гл. 1 (стр. 33-59).

[Подбельский] раздел 1.1 (стр. 7-10).

[Лафоре] гл.2 частично (стр. 53-55).

[Вирт] гл.1.2-1.3 (стр.20-26).

Лекция #2. Типы данных, управляющие операторы, указатели

Содержание

Типы данных в языке Си++, принимаемые ими значения и выполняемые операции. Объявление переменных и инициализация. Оператор присваивания. Разница между операторами присваивания и сравнения. Модификаторы переменных const и unsigned. Вывод значений переменных на экран и считывание значений переменных с клавиатуры. Явное и неявное преобразование типов.

Управляющие операторы if, switch, for, while, do-while, continue, break. Блок кода и область видимости переменных. Представление алгоритма в виде блок-схемы.

Хранение переменных в оперативной памяти. Значение, адрес и тип данных. Указатель как специальный тип данных. Объявление и инициализация адреса. Операция взятия адреса и операция разыменования.

Ключевые определения

Тип данных – множество значений и операций на этих значениях

Переменная – поименованная, либо адресуемая иным способом область памяти, адрес которой можно использовать для осуществления доступа к данным. Данные, находящиеся в переменной (то есть по данному адресу памяти), называются значением этой переменной.

Объявление переменной – создание поименованной области памяти (переменной).

Инициализация переменной – присвоение объявленной переменной значения.

Оператор ветвления if - оператор, конструкция языка программирования, обеспечивающая выполнение определённой команды (набора команд) только при условии истинности некоторого логического выражения.

Оператор ветвления switch - оператор, конструкция языка программирования, обеспечивающая выполнение одной из нескольких команд (наборов команд) в зависимости от значения некоторого выражения.

Оператор параметрического цикла for - цикл, в котором некоторая переменная изменяет своё значение от заданного начального значения до конечного значения с некоторым шагом, и для каждого значения этой переменной тело цикла выполняется один раз.

Оператор цикла с предусловием while - цикл, который выполняется, пока истинно некоторое условие, указанное перед его началом.

Оператор цикла с постусловием do-while - цикл, в котором условие проверяется после выполнения тела цикла. Отсюда следует, что тело всегда выполняется хотя бы один раз.

Оператор управления break – команда, выполняющая досрочное завершение выполнения цикла, в котором условие выхода ещё не достигнуто.

Оператор управления continue – оператор, осуществляющий безусловный переход к следующей итерации с пропуском всех команд до конца тела цикла в текущей итерации.

Область видимости – область программы, в пределах которой идентификатор (имя) некоторой переменной продолжает быть связанным с этой переменной и возвращать её значение. За пределами области видимости тот же самый идентификатор может быть связан с другой переменной, либо быть свободным (не связанным ни с какой из них).

Блок кода – множество операторов языка Си++, окруженных фигурными скобками и обособляющие уникальную область видимости.

Указатель – переменная, диапазон значений которой состоит из адресов ячеек памяти или специального значения — нулевого адреса

Операция взятия адреса – унарная операция, выполняемая над переменной любого типа данных, и возвращающая адрес переменной.

Операция разыменования – унарная операция, выполняемая над указателем (или значением типа «адрес»), для обращения к значению по указанному адресу.

Литература по теме лекции

[Лафоре] гл.2, 3 (стр.48-138).

[Лафоре] гл. 10 частично (стр. 411-436).

[Википедия] Блок-схема.

Лекция #3. Функция, массивы, строки

Содержание

Проблема дублирования кода. Создание функций. Необходимость передачи переменных в функцию. Передача переменной в функцию по значению. Необходимость возврата значения из функции. Возврат значений из функции. Оператор return. Что будет, если попытаться изменить значение переменной, переданной в функцию по значению? Объяснение механизма создания локальной переменной. Необходимость изменения значения входной переменной (функция расчета корней квадратного уравнения). Передача переменной в функцию по указателю. Разница в назначении двух способов передачи переменных в функцию.

Правильная декомпозиция на функции. Причины для декомпозиции – повышение читаемости, уменьшение вложенности, уменьшение количества строк кода функции, уменьшение количества переменных в функции, повышение универсальности функции для возможности повторного использования. Атомарность задачи, выполняемой функцией.

Массивы как структура данных. Представление в оперативной памяти. Для чего нужны массивы. Работа с массивами с помощью циклов for. Передача массивов в функции. Двумерные массивы, лестничные массивы.

Строки как структура данных. Отличие от массива. Нуль-символ. Понятие длины строки и максимальной длины строки. Работа со строками.

Ключевые определения

Декомпозиция – разделение целого на части.

Функция – фрагмент программного кода (подпрограмма), к которому можно обратиться из другого места программы. Функция, как правило, обладает уникальным идентификатором (именем функции), набором входных параметров и возвращаемым значением (возможно, пустым).

Возвращаемое значение функции – значение, полученное в результате выполнения функции, и возвращаемое в точку вызова функции.

Локальная переменная – переменная, существующая в определенной области видимости. Чаще, под локальной переменной подразумевают переменную, объявленную внутри функции.

Глобальная переменная – переменная, доступная в любой точке кода программы или сборки.

Массив – тип или структура данных в виде набора компонентов (элементов массива), расположенных в памяти непосредственно друг за другом.

Строка – тип данных, значениями которого является произвольная последовательность (строка) символов алфавита.

ASCII-код – целочисленный код символа в специальной таблице символов ASCII для представления символов в виде чисел.

Нуль-символ – специальный управляющий символ, используемый для обозначения конца строки (или конца полезной информации в строке), и обозначается «\0».

Литература по теме лекции

[Лафоре] гл.5 (стр.168-215).

[Лафоре] гл.7 (стр.261-307).

Лекция #4. Структуры, перечисления. Динамическая память

Содержание

Необходимость типов данных с пользовательским количеством принимаемых значений. Перечисления как способ ограничения количества принимаемых значений. Удобство использования перечислений. Повышение читаемости за счет использования перечислений.

Растущее количество переменных в коде – увеличение сложности программы. Пример на системе отдела кадров. Составные типы данных как средство уменьшения сложности программы. Синтаксис создания. Работа с переменными структур. Работа с переменными структур через указатели. Упрощение системы отдела кадров за счёт ввода структур данных. UML-представление структур. Агрегирование. Агрегация и композиция. Создание структур на основе реальных объектов и фиктивных.

Задачи на создание структур данных, определения лишних элементов в структуре, создания иерархии объектов. Представление программы в оперативной памяти. Проблема неэффективного использования оперативной памяти. Необходимость создания механизма для временного резервирования оперативной памяти. Динамическая память. Работа с динамической памятью в Си++. Ошибки работы с оперативной памятью.

Ключевые определения

Перечисление – тип данных, чьё множество значений представляет собой ограниченный список идентификаторов.

Структура – программная единица (тип данных), позволяющая хранить и обрабатывать множество однотипных и/или логически связанных данных в вычислительной технике.

Агрегирование – методика создания нового типа данных из уже существующих типов путём их включения в качестве полей нового типа данных.

Агрегация – отношение «часть-целое» между двумя равноправными объектами, когда один объект (контейнер) имеет ссылку на другой объект. Оба объекта могут существовать независимо: если контейнер будет уничтожен, то его содержимое — нет.

Композиция – более строгий вариант агрегирования, когда включаемый объект может существовать только как часть контейнера. Если контейнер будет уничтожен, то и включённый объект тоже будет уничтожен.

Структура (***архитектура***) ***программы*** – совокупность важнейших решений об организации программной системы. Архитектура включает: 1) выбор структурных элементов и их интерфейсов, с помощью которых составлена система, а также их поведения в рамках сотрудничества структурных элементов; 2) соединение выбранных элементов структуры и поведения во всё более крупные системы; 3) архитектурный стиль, который направляет всю организацию — все элементы, их интерфейсы, их сотрудничество и их соединение.

Сегмент кода – область оперативной памяти, выделенной для исполняемого приложения, где располагаются исполняемые инструкции программы.

Сегмент данных – область оперативной памяти, выделенной для исполняемого приложения, где располагаются значения статических, глобальных и константных переменных

Стек – область оперативной памяти, выделенной для исполняемого приложения, где располагаются значения локальных переменных, выполняемых в настоящий момент функций программы.

Куча – свободная область оперативной памяти, доступная для динамического распределения среди исполняемых приложений.

Утечка памяти – процесс неконтролируемого уменьшения объёма свободной оперативной памяти компьютера, связанный с ошибками в работающих программах, вовремя не освобождающих ненужные уже участки памяти.

Литература по теме лекции

[Лафоре] гл. 4 (стр. 142-165).

[Лафоре] гл. 10 частично (стр.437-471).

[УМП] гл. 5 частично (стр. 63-76).

[Фаулер] гл. 3, 5 (стр.62-80, 92-112).

[Вирт] гл. 1.6 (стр.29-31).

Лекция #5. Динамические структуры данных

Содержание

Проблемы хранения данных в массивах: неэффективность работы с оперативной памятью, проблема выделения единого участка памяти, невозможность динамического расширения массива.

Таблица указателей: преимущества, недостатки, реализация.

Список: преимущества, недостатки, реализация.

Стек: назначение, реализация.

Очередь: назначение, реализация.

Словарь (отображение): назначение, реализация.

Граф: назначение, реализация.

Дерево: назначение, реализация.

Объяснение задачи на контрольную работу №1: определения, теоретический вопрос, написание функции, рефакторинг, задача на абстрагирование и агрегацию в UML-диаграмме.

Ключевые определения

Список (***связный список***) – базовая динамическая структура данных, состоящая из узлов, каждый из которых содержит как собственно данные, так и одну или две ссылки («связки») на следующий и/или предыдущий узел списка.

Стек – абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»).

Очередь – абстрактный тип данных с дисциплиной доступа к элементам «первый пришёл — первый вышел» (FIFO, First In — First Out).

Словарь (***ассоциативный массив***) – абстрактный тип данных, позволяющий хранить пары вида «(ключ, значение)» и поддерживающий операции добавления пары, а также поиска и удаления пары по ключу. Предполагается, что словарь не может хранить две пары с одинаковыми ключами.

Граф – совокупность непустого множества вершин и связей между вершинами.

Дерево – это связный ациклический граф.

Литература по теме лекции

[Лафоре] гл.15 (стр.681-749).

[УМП] гл. 5 частично (стр. 76-85).

[Вирт] гл. 4. частично (стр. 167-210).

Лекция #6. Объекты и классы

Содержание

Благодаря составным типам данных и агрегированию, в программах появляется ***архитектура***, что значительно уменьшает сложность программ, делает изучение и ориентирование в коде проще, а, следовательно, облегчает

разработку и поддержку продукта. Наличие архитектуры – главное преимущество структурной парадигмы программирования перед процедурной парадигмой. Однако, строгой организации подлежат только типы данных, когда функции программы остаются никак не систематизированными.

Способом решения данной проблемы является набор понятий, методологий и механизмов языка программирования, объединенных в концепции ***объектно-ориентированного программирования***. Ключевая идея заключается в объединении данных и функций, обрабатывающих эти данные, в единой сущности – ***объекте***. Таким образом, функции программы распределяются по типам данных в существующей архитектуре. Объединение данных и функций в единой архитектуре систематизирует программу и упрощает её изучение.

Однако, ключевой является проблема – каким образом выполнить разделение функций по структурам данных? Для ответа на данных вопрос вводятся следующие понятия: объект, класс, сообщение, состояние, поведение, абстрагирование.

Описание объектной модели. Представление предметной области в виде взаимодействующих объектов. Класс как описание множества объектов со схожими характеристиками. Сообщения как способ взаимодействия объектов. Понятие состояния и поведения объектов. Понятие экземпляра класса и идентичности. Виды сообщений. Абстракции и абстрагирование. Зависимость абстракции от предметной области и решаемой задачи. Типовые задачи. Агрегирование объектов. Реализация класса на языке Си++: поля, методы, конструктор, деструктор. UML-диаграммы классов.

Ключевые определения

Объектно-ориентированное программирование – методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Объект – это отдельный представитель класса, имеющий конкретное состояние и поведение, полностью определяемое классом. Каждый объект обладает состоянием, четко выраженным поведением и идентичностью.

Класс – это способ описания сущности, определяющий состояние и поведение, зависящее от этого состояния, а также правила для взаимодействия с данной сущностью.

Сообщение – обращение одного объекта к другому с возможной передачей данных и ожиданием ответа. Объект, отправивший сообщение, приостанавливает свою работу до тех пор, пока объект-адресат не вернет ответ.

Состояние – перечень значимых характеристик объекта, а также их текущее значение.

Поведение – перечень сообщений (функций), который может выполнить объект, а также их реализация.

Идентичность (***индивидуальность***) – свойства объекта, которые отличают его от всех других объектов. Следует понимать, что идентичность определяется не только состоянием. Два объекта одного класса, имеющие одинаковые значения состояния не являются идентичными, так как остаются двумя разными объектами.

Абстрагирование – процесс создание абстракции некоторого объекта путём выделения значимых в данном контексте характеристик объекта и отбрасывания незначимых.

Поле класса – переменная, характеризующая состояние объекта класса и являющаяся его членом.

Метод класса – функция, реализующая поведение класса и являющаяся его членом.

Конструктор – метод класса, определяющий действия, выполняемые при создании экземпляра класса. Конструктор используется для создания объектов.

Деструктор – метод класса, определяющий действия, выполняемые при уничтожении экземпляра класса. Деструктор используется для «разрушения» объектов.

Литература по теме лекции

[Гради Буч] гл.2-4 (стр. 59-175)

[Лафоре] гл.6 (стр.217-257)

Лекция #7. Инкапсуляция, наследование

Содержание

Проблема #1: возможность изменения состояния объекта может привести к ошибкам во внутренней работе объекта. Проблема #2: возможность вызова извне методов объекта, обеспечивающих его внутреннее функционирование, может привести к ошибкам в работе объекта. Также, доступность методов, необходимых для внутреннего функционирования, увеличивает связность класса с другими частями программы, а также усложняет изучение использования данного класса.

Для решения данных проблем в объектно-ориентированном программировании придуман механизм ***сокрытия реализации***, позволяющий разделять поля и методы на доступные извне (интерфейс) и доступные для вызова только внутри класса (реализация). Сокрытие реализации в языке Си++ обеспечивается использованием ***модификаторов доступа***. Сокрытие реализации является частью более общего понятия – ***инкапсуляции***. Обозначение сокрытия реализации на UML-диаграммах классов.

Проектирование класса должно выполняться таким образом, чтобы как можно больше полей и методов класса были сокрыты. Негласное правило: не больше трёх открытых полей, не большей пяти открытых методов.

Часто при программировании возникает ситуация, когда необходимо создать класс, который копирует реализацию другого класса, но при этом добавляет к ней несколько собственных полей или методов. Первое приходящее в голову решение – скопировать исходный код одного класса в другой. Однако такое решение приводит к дублированию кода, а это: а) увеличение кода, требующего поддержки; б) копирование потенциальных ошибок.

Для отсутствия дублирования исходная проблема решается механизмом ***наследования***, при котором один класс полностью приобретает реализацию другого класса. Важно отметить, что класс-наследник приобретает как открытые методы, так и закрытые методы класса-родителя. Наследование в языке Си++. Наследование на UML-диаграммах классов.

При изменении класса-родителя (добавлении или удалении полей и методов) изменяется и наследованная реализация класса-наследника. Таким образом, не возникает проблема дублирования кода, а поддержка кода становится проще.

Проблема доступа к закрытой реализации класса-родителя со стороны класса-наследника. Модификатор доступа protected.

Ключевые определения

Сокрытие реализации – механизм языка программирования, предназначенный для управления видимостью частей кода, например, членов класса.

Интерфейс класса - это набор методов класса, доступных для использования другими классами.

Реализация класса – алгоритмы и внутренние данные, необходимые для обеспечения работы интерфейса класса.

Модификаторы доступа – ключевые слова языка программирования, управляющие областью видимости членов класса. В языке Си++ существуют 3 модификатора доступа: public, protected, private.

Инкапсуляция – это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе и скрыть детали реализации от пользователя.

Наследование – это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью.

Класс-родитель **(*базовый класс*)** – класс, от которого производится наследование.

Класс-наследник **(*дочерний класс*)** – класс, который производит наследование.

Литература по теме лекции

[Лафоре] гл.9 (стр.361-407).

Лекция #8. Виртуальные функции, полиморфизм

Содержание

Проблема: есть множество классов, выполняющих схожие задачи, но имеющие различную реализацию. Несмотря на схожесть в интерфейсах, код использования объектов данных классов дублируется. Решением проблемы могла бы стать некоторая сущность, которая позволяла бы обращаться к объектам различных классов через единый интерфейс. Проблема схожа с наследованием, однако при наследовании классом приобретается вся реализация класса-родителя, когда в новой задаче необходимо приобретение только интерфейса класса-родителя с возможностью создания собственной реализации под интерфейс.

Решением данной проблемы является ***полиморфизм***, основанный на использовании следующих понятий: виртуальная функция, наследование и указатель на базовый класс.

Создание и применение виртуальной функции на языке Си++. Чисто виртуальная функция. Понятие абстрактного класса. Отображение виртуальных функций на UML-диаграммах классов.

Указатель на базовый класс и его возможности. Доступ к реализации класса-наследника через интерфейс класса-родителя. Уменьшение дублирования кода за счет применения полиморфизма. Различие в понятиях абстрактного базового класса и интерфейса как сущности. Отображение полиморфных объектов на UML-диаграммах классов.

Ключевые определения

Виртуальная функция – метод класса, реализация которого может быть переопределена в дочерних классах

Чисто виртуальная функция – виртуальная функция, не имеющая реализацию в базовом классе.

Абстрактный класс – класс, содержащий хотя бы одну чисто виртуальную функцию. («Абстракным» класс называется в данном случае потому, что невозможно создать экземпляр абстрактного класса)

Полиморфизм – это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта

Указатель на базовый класс – переменная, являющаяся типом базового класса, но способная также хранить адреса объектов производных (дочерних) классов и предоставлять доступ к их реализации методов, относящихся к интерфейсу базового класса.

Интерфейс-сущность – сущность предметной области, не имеющая состояния, но определяющая интерфейс класса из чисто виртуальных функций

для их реализации в производных классах. (К сожалению, в ООП существуют два понятия с одинаковым названием, но отличающиеся по смыслу. Поэтому в контексте следует различать интерфейс отдельного класса и интерфейс как самостоятельную сущность).

Литература по теме лекции

[Лафоре] гл.11 (стр.476-531).

[Фаулер] гл. 5 (стр. 92-112).

Лекция #9. Шаблоны

Содержание

Проблема: есть две или более функций (методов), имеющие одинаковую реализацию, но отличающиеся только используемыми типами данных. Фактически, это дублирование кода. Решением проблемы является ***шаблонные функции***.

Реализация шаблонных функций на языке Си++. Применение шаблонных функций для исключения дублирования кода (функции перестановки значений указателей, функции сортировки массивов). Ограничения, накладываемые на алгоритмы в шаблонных функциях. Шаблонные функции и перегруженные функции с конкретной реализацией.

Обобщенная задача: есть два или более классов, имеющих одинаковую реализацию, но отличающиеся только используемыми типами данных. Фактически, это дублирование кода. Решением проблемы является ***шаблонные классы***.

Реализация шаблонных классов на языке Си++. Применение шаблонных классов на примере классических структур данных (списках, словарях). Ограничения, накладываемые на алгоритмы в шаблонных классах. Наследование от шаблонных классов.

На данном этапе рассмотрены все базовые механизмы и понятия объектно-ориентированного программирования. Все они направлены либо на уменьшение сложности программы за счёт строгой организации кода, либо на уменьшение дублирования кода для упрощения поддержки программ. Однако, для эффективного использования данных механизмов, необходимо понимать, на решение какой конкретной проблемы они направлены.

Описание изменений в объектной архитектуре программы в сравнении с процедурной и структурной. Перечисление ключевых достоинств и недостатков объектно-ориентированного подхода. Повторение проблем, на которые направлены техники ООП, с представлением их на языке Си++ или UML-диаграммах. Умение определять проблемы кода и выбирать правильную технику для их решения.

Ключевые определения

Шаблонная (обобщенная) функция – функция, реализация которой способна работать с различными типами данных, определяемыми при вызове функции.

Шаблонный (обобщенный) класс – класс, состояние и/или поведение которого может быть определено различными типами данных, конкретизированных при создании экземпляра класса.

Перегрузка функций – создание нескольких функций с единым идентификатором (именем функции), но различным набором входных параметров.

Литература по теме лекции

[Лафоре] гл.14 частично (стр.640-658)

Лекция #10. Практические задачи по проектированию

Содержание

Методология проектирования программ с применением объектно-ориентированного подхода:

1) Понимание задач, стоящих перед пользователем программы (постановка задачи).
2) Составление алгоритмов по решению задач пользователя (блок-схемы или диаграммы вариантов использования).
3) Выделение базовых сущностей и данных в составленных алгоритмах.
4) Конкретизация полученных сущностей, определение их атрибутов (полей), создание иерархии классов.
5) Выделение в составленных алгоритмов действий (функций) по обработке данных.
6) Распределение функций по существующим сущностям архитектуры. Возможная конкретизация архитектуры.

Пример использования методологии для решения конкретных задач в различных предметных областях.

Объяснение задачи на контрольную работу №2: определения, теоретический вопрос, написание класса, задача на абстрагирование, инкапсуляцию, наследование и полиморфизм в UML-диаграмме.

Ключевые определения

Системный архитектор – должность в компаниях, связанных с разработкой программного обеспечения с долгим циклом разработки и поддержки, в обязанности которой является долгосрочное планирование архитектуры разрабатываемых систем и контроль реализации запланированной архитектуры командой разработчиков. Системный архитектор является высшим звеном

развития разработчика, так как требует многолетнего (порядка 15-20 лет) опыта практической разработки в различных проектах, на различных платформах, в различных предметных областях.

Литература по теме лекции

[Лафоре] гл. 16 (стр. 752-794)

[Гради Буч] гл. 6 (стр. 281-338)

Лекция #11. Критерии качества кода. Рефакторинг

Содержание

Анализ качества кода по UML-диаграммам. Критерии оценки архитектуры:

- Минимальная сложность.
- Простота сопровождения (поддержки).
- Слабое сопряжение.
- Расширяемость.
- Возможность повторного использования.
- Высокий коэффициент объединения по входу.
- Низкий коэффициент разветвления по выходу.
- Портируемость.
- Достаточность функциональности.
- Стратификация.
- Соответствие стандартным методикам.

Признаки плохой архитектуры в UML-диаграммах классов. Перепроектирование.

Анализ качества исходного кода. Признаки плохого исходного кода. Рефакторинг.

Методология проведения рефакторинга.

Ключевые определения

Сопряжение – количество и мощность связей одного класса с другими классами.

Рефакторинг – изменение во внутренней структуре программного обеспечения, имеющие целью облегчить понимание его работы и упростить модификацию, не затрагивая наблюдаемого поведения.

Литература по теме лекции

[МакКоннелл] гл. 5.2 частично (стр. 74-99)

[Рефакторинг] гл. 3 (стр. 85-101)

Лекция #12. Основы тестирования

Содержание

Проблема: при написании программы каждая новая функция требует тестирование, в том числе и повторного тестирования, что при выполнении тестирования вручную ведет к большим временным затратам. Решение проблемы заключается в автоматизации проведения тестирования. Одним из видов автоматизированного тестирования является ***модульное тестирование***, или, более распространенное название, ***юнит-тестирование***.

Принцип составления и работы юнит-тестов. Пример написания юнит-тестов в реальной практике и экономия времени тестирования.

Причины для написания юнит-тестов и определение необходимого объёма покрытия кода тестами.

Лекция #13. Повышение производительности работы

Содержание

Инструментальные средства повышения эффективности работы разработчика: средства Visual Studio, инструменты рефакторинга, системы версионного контроля.

Ответы на вопросы по экзаменационным билетам

Лекция #14. Экзамен

Экзамен проводится в письменной форме в течение 2 академических часов. В билете содержатся вопросы на знание определений, применения базовых механик и понятий объектно-ориентированного подхода, а также практическая задача на составление архитектуры заданной предметной области.

Заключение

В пособии представлены курс лекций, а также рекомендации по изучению дисциплины «Объектно-ориентированное программирование». В рамках лекций рассмотрены типовые проблемы, связанные с разработкой ПО, а также способы их решения. Успешное освоение принципов ООП, а также понимание задач, на решение которых направлены эти принципы, является важными навыками разработчика программного обеспечения.

Данный курс рекомендуется завершить курсовыми проектами, направленными на разработку приложений в конкретной предметной области и использовании полученных знаний на практике.

Список обязательной литературы

1. Лафоре Р. Объектно-ориентированное программирование в C++. – 4-е изд. – С.-П.: Питер, 2004. – 924 с.
2. [Гради Буч] Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений, 3-е изд / Г. Буч, Р.А. Максимчук, М.У. Энгл, Б.Д. Янг, Д. Коналлен, К.А. Хьюстон // пер. с англ. и ред. Д.А. Клюшина. – М.: ООО «И.Д. Вильямс», 2008. – 720 с.
3. [УМП] Горяинов А. Введение в программирование на языке Си++: учебно-методическое пособие / А.Е. Горяинов. – Томск: Томск. гос. ун-т упр. и радиоэлектронники, 2015. – 126 с.

Список дополнительной литературы

1. [Вирт] Вирт Н. Алгоритмы и структуры данных. – Пер. с англ. Ткачев Ф.В. – С.-П.: ДМК Пресс, 2010. – 274 с.
2. [Бадд] Бадд Т. Объектно-ориентированное программирование в действии / Т. Бадд, изд. «Addison Wesley», пер. с англ. изд «Питер». – СПб.: «Питер», 1997. – 296 с.
3. [Фаулер] Фаулер М. UML. Основы, 3-е издание./ М. Фаулер, пер. с англ. – СПб: Символ-Плюс, 2004. – 192 с.
4. [Рефакторинг] Фаулер М. Рефакторинг. Улучшение существующего кода / М. Фаулер, пер. с англ. – СПб: Символ-Плюс, 2004.
5. [Шилдт] Шилдт Г. C++: базовый курс. - 3-е изд., пер. с англ. – М.: Издательский дом «Вильямс», 2010. – 624 с.
6. [Подбельский] Подбельский В.В. Язык Си++: учеб. пособие. – 5-е изд. – М.: Финансы и статистика, 2003. – 560 с.
7. [Страуструп] Страуструп Б. Язык программирования C++. – М.: Бином, 2011. – 1136 с.
8. [VS] Visual Studio: офиц. страница [Электронный ресурс]. – Режим доступа: https://www.visualstudio.com/ - (Дата обращения: 15.11.2015)
9. [msdn] Microsoft Developer Network: форум поддержки разработчиков Microsoft MSDN [Электронный ресурс]. – Режим доступа: https://www.social.msdn.microsoft.com/ - (Дата обращения: 15.11.2015)
10. [Википедия] http://ru.wikipedia.org (только статьи, на которые идут прямые ссылки в методическом пособии)
11. [ОС ТУСУР] Работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления: образовательный стандарт вуза ТУСУР 01-2013. – Томск: ТУСУР, 2013. – 57 с.
12. [rsdn] Russian Software Developer Network: Соглашения по оформлению кода команды RSDN [Электронный ресурс]. -

http://rsdn.ru/article/mag/200401/codestyle.xml/ - (Дата обращения: 15.11.2015)

13. [МакКоннелл] МакКоннелл С. Совершенный код / С. Макконнелл, пер. с англ. под ред. В.Г. Вшивцева. – С.-П.: Питер, 2005. – 896с.