ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ОРЕНБУРГСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ДЛЯ ОБУЧАЮЩИХСЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

Б1.Б.1.20 Технологии и методы программирования

Направление подготовки *10.05.03 Информационная безопасность автоматизированных систем*

Профиль образовательной программы *Информационная безопасность автоматизированных систем критически важных объектов*

Форма обучения *очная*

СОДЕРЖАНИЕ

1. КОНСПЕКТ ЛЕКЦИЙ

1.1. Лекция № 1, 2 (4 часа)

Тема: «Понятие о программном средстве»

1.1.1. Вопросы лекции:

1) Понятие информационной среды процесса обработки данных.
2) Программа как формализованное описание процесса.
3) Понятие о программном средстве.

1.1.2 Краткое содержание вопросов

Целью программирования является описание процессов обработки данных (в дальнейшем - просто процессов). Согласно ИФИПа: данные (data) - это представление фактов и идей в формализованном виде, пригодном для передачи и переработке в некоем процессе, а информация (information) - это смысл, который придается данным при их представлении. Обработка данных (data processing) - это выполнение систематической последовательности действий с данными. Данные представляются и хранятся на т.н. носителях данных. Совокупность носителей данных, используемых при какой-либо обработке данных, будем называть информационной средой (data medium). Набор данных, содержащихся в какой-либо момент в информационной среде, будем называть состоянием этой информационной среды. Процесс можно определить как последовательность сменяющих друг друга состояний некоторой информационной среды.

Описать процесс - это значит определить последовательность состояний заданной информационной среды. Если мы хотим, чтобы по заданному описанию требуемый процесс порождался автоматически на каком-либо компьютере, необходимо, чтобы это описание было формализованным. Такое описание называется программой. С другой стороны, программа должна быть понятной и человеку, так как и при разработке программ, и при их использовании часто приходится выяснять, какой именно процесс она порождает. Поэтому программа составляется на удобном для человека формализованном языке программирования, с которого она автоматически переводится на язык соответствующего компьютера с помощью другой программы, называемой транслятором. Человеку (программисту), прежде чем составить программу на удобном для него языке программирования, приходится проделывать большую подготовительную работу по уточнению постановки задачи, выбору метода ее решения, выяснению специфики применения требуемой программы, прояснению общей организации разрабатываемой программы и многое другое. Использование этой информации может существенно упростить задачу понимания программы человеком, поэтому весьма полезно ее как-то фиксировать в виде отдельных документов (часто не формализованных, рассчитанных только для восприятия человеком).

Обычно программы разрабатываются в расчете на то, чтобы ими могли пользоваться люди, не участвующие в их разработке (их называют пользователями). Для освоения программы пользователем помимо ее текста требуется определенная дополнительная документация. Программа или логически связанная совокупность программ на носителях данных, снабженная программной документацией, называется программным средством (ПС). Программа позволяет осуществлять некоторую автоматическую обработку данных на компьютере. Программная документация позволяет понять, какие функции выполняет та или иная программа ПС, как подготовить исходные данные и запустить требуемую программу в процесс ее выполнения, а также: что означают получаемые результаты (или каков эффект выполнения этой программы). Кроме того, программная документация помогает разобраться в самой программе, что необходимо, например, при ее модификации.

1.2. Лекция № 3, 4 (4 часа)

Тема: «Источники ошибок в программных средствах»

1.2.1. Вопросы лекции:

1) Интеллектуальные возможности человека, используемые при разработке программных систем.

2) Понятия о простых и сложных системах, о малых и больших системах.

1.2.2. Краткое содержание вопросов:

Дейкстра [2.1] выделяет три интеллектуальные возможности человека, используемые при разработке ПС:

способность к перебору,
способность к абстракции,
способность к математической индукции.

Способность человека к перебору связана с возможностью последовательного переключения внимания с одного предмета на другой, позволяя узнавать искомый предмет. Эта способность весьма ограничена - в среднем человек может уверенно (не сбиваясь) перебирать в пределах 1000 предметов (элементов). Человек должен научиться действовать с учетом этой своей ограниченности. Средством преодоления этой ограниченности является его способность к абстракции, благодаря которой человек может объединять разные предметы или экземпляры в одно понятие, заменять множество элементов одним элементом (другого рода). Способность человека к математической индукции позволяет ему справляться с бесконечными последовательностями.

При разработке ПС человек имеет дело с системами. Под системой будем понимать совокупность взаимодействующих (находящихся в отношениях) друг с другом элементов. ПС можно рассматривать как пример системы. Логически связанный набор программ является другим примером системы. Любая отдельная программа также является системой. Понять систему - значит осмысленно перебрать все пути взаимодействия между ее элементами. В силу ограниченности человека к перебору будем различать простые и сложные системы . Под простой будем понимать такую систему, в которой человек может уверенно перебирать все пути взаимодействия между ее элементами, а под сложной будем понимать такую систему, в которой он этого делать не в состоянии. Между простыми и сложными системами нет четкой границы, поэтому можно говорить и о промежуточном классе систем: к таким системам относятся программы, о которых программистский фольклор утверждает, что "в каждой отлаженной программе имеется хотя бы одна ошибка".

При разработке ПС мы не всегда можем уверенно знать о всех связях между ее элементами из-за возможных ошибок. Поэтому полезно уметь оценивать сложность системы по числу ее элементов: числом потенциальных путей взаимодействия между ее элементами, т.е. n! , где n - число ее элементов. Систему назовем малой, если $n < 7$ ($6! = 720 < 1000$), систему назовем большой, если $n > 7$. При $n=7$ имеем промежуточный класс систем. Малая система всегда проста, а большая может быть как простой, так и сложной. Задача технологии программирования - научиться делать большие системы простыми.

Полученная оценка простых систем по числу элементов широко используется на практике. Так, для руководителя коллектива весьма желательно, чтобы в нем не было больше шести взаимодействующих между собой подчиненных. Весьма важно также следовать правилу: “все, что может быть сказано, должно быть сказано в шести пунктах или меньше”. Этому правилу мы будем стараться следовать в настоящем пособии: всякие

перечисления взаимосвязанных утверждений (набор рекомендаций, список требований и т.п.) будут соответствующим образом группироваться и обобщаться. Полезно ему следовать и при разработке ПС.

1.3. Лекция № 5, 6, 7, 8 (8 часов)

Тема: «Специфика разработки программных средств»

1.3.1. Вопросы лекции:

1) Жизненный цикл программного средства.

2) Понятие качества программного средства.

3) Обеспечение надежности - основной мотив разработки программного средства.

1.3.2. Краткое содержание вопросов:

Разработка программных средств имеет ряд специфических особенностей.

- Прежде всего, следует отметить некоторое противостояние: неформальный характер требований к ПС (постановки задачи) и понятия ошибки в нем, но формализованный основной объект разработки - программы ПС. Тем самым разработка ПС содержит определенные этапы формализации, а переход от неформального к формальному существенно неформален.

- Разработка ПС носит творческий характер (на каждом шаге приходится делать какой - либо выбор, принимать какое - либо решение), а не сводится к выполнению какой - либо последовательности регламентированных действий. Тем самым эта разработка ближе к процессу проектирования каких - либо сложных устройств, но никак не к их массовому производству. Этот творческий характер разработки ПС сохраняется до самого ее конца.

- Следует отметить также особенность продукта разработки. Он представляет собой некоторую совокупность текстов (т.е. статических объектов), смысл же (семантика) этих текстов выражается процессами обработки данных и действиями пользователей, запускающих эти процессы (т.е. является динамическим). Это предопределяет выбор разработчиком ряда специфичных приемов, методов и средств.

- Продукт разработки имеет и другую специфическую особенность: ПС при своем использовании (эксплуатации) не расходуется и не расходует используемых ресурсов.

1.4. Лекция № 9, 10, 11 (6 часаов)

Тема: «Понятие внешнего описания».

1.4.1. Вопросы лекции:

1) Определение требований к программному средству.

2) Спецификация качества программного средства.

1.4.2. Краткое содержание вопросов:

Разработчикам больших программных средств приходится решать весьма специфические и трудные проблемы, особенно, если это ПС должно представлять собой программную систему нового типа, в плохо компьютеризированной предметной области. Разработка ПС начинается с процесса формулирования требований к ПС, в котором, исходя из довольно смутных пожеланий заказчика, должен быть создан документ, достаточно точно определяющий задачи разработчиков ПС. Этот документ мы называем внешним описанием ПС.

Очень часто требования к ПС путают с требованиями к процессам его разработки (технологическим процессам). Последние не следует включать во внешнее описание, если только они не связаны с оценкой качества ПС. В случае необходимости требования к технологическим процессам можно оформить в виде самостоятельного документа, который будет использоваться при управлении разработкой ПС.

Внешнее описание ПС играет роль точной постановки задачи, решение которой должно обеспечить разрабатываемое ПС. Более того, оно должно содержать всю информацию, которую необходимо знать пользователю для применения ПС. Оно является исходным документом для трех параллельно протекающих процессов: разработки текстов (конструированию и кодированию) программ, входящих в ПС, разработки документации по применению ПС и разработки существенной части комплекта тестов для тестирования ПС. Ошибки и неточности во внешнем описании, в конечном счете, трансформируются в ошибки самой ПС и обходятся особенно дорого, во-первых, потому, что они делаются на самом раннем этапе разработки ПС, и, во-вторых, потому, что они распространяются на три параллельных процесса. Это требует принятия особенно серьезных мер по их предупреждению.

Исходным документом для разработки внешнего описания ПС является определение требований к ПС. Но так как через этот документ передается от заказчика (пользователя) к разработчику основная информация относительно требуемого ПС, то формирование этого документа представляет собой довольно длительный и трудный итерационный процесс взаимодействия между заказчиком и разработчиком, с которого и начинается этап разработки требований к ПС . Трудности, возникающие в этом процессе, связаны с тем, что пользователи часто плохо представляют, что им на самом деле нужно: использование компьютера в "узких" местах деятельности пользователей может на самом деле потребовать принципиального изменения всей технологии этой деятельности (о чем пользователи, как правило, и не догадываются). Кроме того, проблемы, которые необходимо отразить в определении требований, могут не иметь определенной формулировки, что приводит к постепенному изменению понимания разработчиками этих проблем. В связи с этим определению требований часто предшествует процесс системного анализа, в котором выясняется, насколько целесообразно и реализуемо "заказываемое" ПС, как повлияет такое ПС на деятельность пользователей и какими особенностями оно должно обладать. Иногда бывает полезным разработка упрощенной версии требуемого ПС, называемую прототипом ПС. Анализ "пробного" применения прототипа позволяет выявить действительные потребности пользователей и существенно уточнить требования к ПС.

В определении внешнего описания легко бросаются в глаза две самостоятельные его части. Описание поведения ПС определяет функции, которые должна выполнять ПС, и потому его называют функциональной спецификацией ПС. Функциональная спецификация определяет допустимые фрагменты программ, реализующих декларированные функции. Требования к качеству ПС должны быть сформулированы так, чтобы разработчику были ясны цели, которые он должен стремиться достигнуть при разработке этого ПС. Эту часть внешнего описания будем называть спецификацией качества ПС (в литературе ее часто называют нефункциональной спецификацией [4.1], но она, как правило, включает и требования к технологическим процессам). Она, в отличие от функциональной спецификации, представляется в неформализованном виде и играет роль тех ориентиров, которые в значительной степени определяют выбор подходящих альтернатив при реализации функций ПС, а также определяет стиль всех документов и программ требуемого ПС. Тем самым, спецификация качества играет решающую роль в обеспечении требуемого качества ПС.

Обычно разработка спецификации качества предшествует разработке функциональной спецификации ПС, так как некоторые требования к качеству ПС могут предопределять включение в функциональную спецификацию специальных функций, например, функции защиты от несанкционированного доступа к некоторым объектам информационной среды. Таким образом, структуру внешнего описания ПС можно выразить формулой:

Внешнее описание ПС = определение требований
+ спецификация качества ПС
+ функциональная спецификация ПС

Внешнее описание определяет, что должно делать ПС и какими внешними свойствами оно должно обладать. Оно не отвечает на вопросы, как обеспечить требуемые внешние свойства ПС и как это ПС должно быть устроено. Внешнее описание должно достаточно точно и полно определять задачи, которые должны решить разработчики требуемого ПС. В то же время оно должно быть понято представителем пользователем - на его основании заказчиком достаточно часто принимается окончательное решение на заключение договора на разработку ПС. Внешнее описание играет большую роль в обеспечении требуемого качества ПС, так как спецификация качества ставит для разработчиков ПС конкретные ориентиры, управляющие выбором приемлемых решений при реализации специфицированных функций.

1.5. Лекция № 12, 13, 14 (6 часов)

Тема: «Методы спецификации семантики функций».

1.5.1. Вопросы лекции:

1) Основные подходы к спецификации семантики функций.

2) Табличный подход, метод таблиц решений.

1.5.2. Краткое содержание вопросов:

Табличный подход для определения функций хорошо известен ещё со средней школы. Он базируется на использовании таблиц. В программировании эти методы получили развитие в методе таблиц решений.

Алгебраический подход для определения функций базируется на использовании равенств. В этом случае для определения некоторого набора функций строится система равенств вида:

L1=R1,

. . . (5.1)

Ln=Rn.

где Li и Ri, i=1, ... n, некоторые выражения, содержащие предопределенные операции, константы, переменные, от которых зависят определяемые функции (формальные параметры этих функций), и вхождения самих этих функций. Семантика определяемых функций извлекается в результате интерпретации этой системы равенств. Эта интерпретация может осуществляться по-разному (базироваться на разных системах правил), что порождает разные семантики. В настоящее время активно исследуются операционная, денотационная и аксиоматическая семантики.

Третий подход, логический, базируется на использовании предикатов - функций, у которых аргументами могут быть значения различных типов, а результатами являются логические значения (ИСТИНА и ЛОЖЬ). В этом случае набор функций может

определяться с помощью системы предикатов. Заметим, что система равенств алгебраического подхода может быть задана с помощью следующей системы предикатов:

РАВНО(L1, R1),

. . . (5.2)

РАВНО(Ln, Rn),

где предикат РАВНО истинен, если равны значения первого и второго его аргументов. Это говорит о том, что логический подход располагает не меньшими возможностями для определения функций, однако он требует от разработчиков ПС умения пользоваться методами математической логики, что, к сожалению, не для всех разработчиков оказывается приемлемым. Более подробно этот подход мы рассматривать не будем.

Графический подход также известен еще со средней школы. Но в данном случае речь идет не о задании функции с помощью графика, хотя при данном уровне развития компьютерной техники ввод в компьютер таких графиков возможен и они могли бы использоваться (с относительно небольшой точностью) для задания функций. В данном случае речь идет о графическом задании различных схем, выражающих сложную функцию через другие функции, связанными с какими-либо компонентами заданной схемы. Графическая схема может определять ситуации, когда для вычисления представляемой ею функции должны применяться связанные с этой схемой более простые функции. Графическая схема может достаточно точно определять часть семантики функции. Примером такой схемы может быть схема переходов состояний конечного автомата, такая, что в каждом из этих состояний должна выполняться некоторая дополнительная функция, указанная в схеме.

1.6. Лекция № 15, 16, 17 (6 часов)

Тема: «Архитектура программного средства».

1.6.1. Вопросы лекции:

1) Понятие архитектуры программного средства.

2) Виды архитектуры программного средства.

1.6.2. Краткое содержание вопросов:

Архитектура ПС - это его строение как оно видно (или должно быть видно) извне его, т.е. представление ПС как системы, состоящей из некоторой совокупности взаимодействующих подсистем. В качестве таких подсистем выступают обычно отдельные программы. Разработка архитектуры является первым этапом борьбы со сложностью ПС, на котором реализуется принцип выделения относительно независимых компонент.

Основные задачи разработки архитектуры ПС:

- Выделение программных подсистем и отображение на них внешних функций (заданных во внешнем описании) ПС;
- определение способов взаимодействия между выделенными программными подсистемами.

С учетом принимаемых на этом этапе решений производится дальнейшая конкретизация и функциональных спецификаций.

Различают следующие основные классы архитектур программных средств:

* цельная программа;
* комплекс автономно выполняемых программ;

* слоистая программная система;
* коллектив параллельно выполняемых программ.

Цельная программа представляет вырожденный случай архитектуры ПС: в состав ПС входит только одна программа. Такую архитектуру выбирают обычно в том случае, когда ПС должно выполнять одну какую-либо ярко выраженную функцию и ее реализация не представляется слишком сложной. Естественно, что такая архитектура не требует какого-либо описания (кроме фиксации класса архитектуры), так как отображение внешних функций на эту программу тривиально, а определять способ взаимодействия не требуется (в силу отсутствия какого-либо внешнего взаимодействия программы, кроме как взаимодействия ее с пользователем, а последнее описывается в документации по применению ПС).

Комплекс автономно выполняемых программ состоит из набора программ, такого, что:

- любая из этих программ может быть активизирована (запущена) пользователем;
- при выполнении активизированной программы другие про-граммы этого набора не могут быть активизированы до тех пор, пока не закончит выполнение активизированная программа;
- все программы этого набора применятся к одной и той же информационной среде.

Таким образом, программы этого набора по управлению не взаимодействуют - взаимодействие между ними осуществляется только через общую информационную среду.

Слоистая программная система состоит из некоторой упорядоченной совокупности программных подсистем, называемых слоями, такой, что:

- на каждом слое ничего не известно о свойствах (и даже существовании) последующих (более высоких) слоев;
- каждый слой может взаимодействовать по управлению (обращаться к компонентам) с непосредственно предшествующим (более низким) слоем через заранее определенный интерфейс, ничего не зная о внутреннем строении всех предшествующих слоев;
- каждый слой располагает определенными ресурсами, которые он либо скрывает от других слоев, либо предоставляет непосредственно последующему слою (через указанный интерфейс) некоторые их абстракции.

Таким образом, в слоистой программной системе каждый слой может реализовать некоторую абстракцию данных. Связи между слоями ограничены передачей значений параметров обращения каждого слоя к смежному снизу слою и выдачей результатов этого обращения от нижнего слоя верхнему. Недопустимо использование глобальных данных несколькими слоями.

В качестве примера рассмотрим использование такой архитектуры для построения операционной системы. Такую архитектуру применил Дейкстра при построении операционной системы THE. Эта операционная система состоит из четырех слоев. На нулевом слое производится обработка всех прерываний и выделение центрального процессора программам (процессам) в пакетном режиме. Только этот уровень осведомлен о мультипрограммных аспектах системы. На первом слое осуществляется управление страничной организацией памяти. Всем вышестоящим слоям предоставляется виртуальная непрерывная (не страничная) память. На втором слое осуществляется связь с консолью (пультом управления) оператора. Только этот слой знает технические характеристики консоли. На третьем слое осуществляется буферизация входных и выходных потоков данных и реализуются так называемые абстрактные каналы ввода и вывода, так что прикладные программы не знают технических характеристик устройств ввода и вывода.

Коллектив параллельно действующих программ представляет собой набор программ, способных взаимодействовать между собой, находясь одновременно в стадии

выполнения. Это означает, что такие программы, во-первых, вызваны в оперативную память, активизированы и могут попеременно разделять по времени один или несколько центральных процессоров, а во-вторых, осуществлять между собой динамические (в процессе выполнения) взаимодействия, на базе которых производиться их синхронизация. Обычно взаимодействие между такими процессами производится путем передачи друг другу некоторых сообщений.

1.7. Лекция № 18, 19 (4 часа)

Тема: «Разработка структуры программы».

1.7.1. Вопросы лекции:

1) Понятие программного модуля.

2) Основные характеристики программного модуля.

1.7.2. Краткое содержание вопросов:

Приступая к разработке каждой программы ПС, следует иметь в виду, что она, как правило, является большой системой, поэтому мы должны принять меры для ее упрощения. Для этого такую программу разрабатывают по частям, которые называются программными модулями. А сам такой метод разработки программ называют модульным программированием. Программный модуль - это любой фрагмент описания процесса, оформляемый как самостоятельный программный продукт, пригодный для использования в описаниях процесса. Это означает, что каждый программный модуль программируется, компилируется и отлаживается отдельно от других модулей программы, и тем самым, физически разделен с другими модулями программы. Более того, каждый разработанный программный модуль может включаться в состав разных программ, если выполнены условия его использования, декларированные в документации по этому модулю. Таким образом, программный модуль может рассматриваться и как средство борьбы со сложностью программ, и как средство борьбы с дублированием в программировании (т.е. как средство накопления и многократного использования программистских знаний).

Модульное программирование является воплощением в процессе разработки программ обоих общих методов борьбы со сложностью: и обеспечение независимости компонент системы, и использование иерархических структур. Для воплощения первого метода формулируются определенные требования, которым должен удовлетворять программный модуль, т.е. выявляются основные характеристики “хорошего” программного модуля. Для воплощения второго метода используют древовидные модульные структуры программ (включая деревья со сросшимися ветвями).

1.8. Лекция № 20, 21 (4 часа)

Тема: «Разработка программного модуля».

1.8.1. Вопросы лекции:

1) Порядок разработки программного модуля.

2) Структурное программирование и пошаговая детализация.

1.8.2. Краткое содержание вопросов:

При разработке программного модуля целесообразно придерживаться следующего порядка:

изучение и проверка спецификации модуля, выбор языка программирования;
выбор алгоритма и структуры данных;
программирование (кодирование) модуля;
шлифовка текста модуля;
проверка модуля;

компиляция модуля.

Первый шаг разработки программного модуля в значительной степени представляет собой смежный контроль структуры программы снизу: изучая спецификацию модуля, разработчик должен убедиться, что она ему понятна и достаточна для разработки этого модуля. В завершении этого шага выбирается язык программирования: хотя язык программирования может быть уже предопределен для всего ПС, все же в ряде случаев (если система программирования это допускает) может быть выбран другой язык, более подходящий для реализации данного модуля (например, язык ассемблера).

На втором шаге разработки программного модуля необходимо выяснить, не известны ли уже какие-либо алгоритмы для решения поставленной и или близкой к ней задачи. И если найдется подходящий алгоритм, то целесообразно им воспользоваться. Выбор подходящих структур данных, которые будут использоваться при выполнении модулем своих функций, в значительной степени предопределяет логику и качественные показатели разрабатываемого модуля, поэтому его следует рассматривать как весьма ответственное решение.

На третьем шаге осуществляется построение текста модуля на выбранном языке программирования. Обилие всевозможных деталей, которые должны быть учтены при реализации функций, указанных в спецификации модуля, легко могут привести к созданию весьма запутанного текста, содержащего массу ошибок и неточностей. Искать ошибки в таком модуле и вносить в него требуемые изменения может оказаться весьма трудоемкой задачей. Поэтому весьма важно для построения текста модуля пользоваться технологически обоснованной и практически проверенной дисциплиной программирования. Впервые на это обратил внимание Дейкстра, сформулировав и обосновав основные принципы структурного программирования. На этих принципах базируются многие дисциплины программирования, широко применяемые на практике.

Следующий шаг разработки модуля связан с приведением текста модуля к завершенному виду в соответствии со спецификацией качества ПС. При программировании модуля разработчик основное внимание уделяет правильности реализации функций модуля, оставляя недоработанными комментарии и допуская некоторые нарушения требований к стилю программы. При шлифовке текста модуля он должен отредактировать имеющиеся в тексте комментарии и, возможно, включить в него дополнительные комментарии с целью обеспечить требуемые примитивы качества. С этой же целью производится редактирование текста программы для выполнения стилистических требований.

Шаг проверки модуля представляет собой ручную проверку внутренней логики модуля до начала его отладки (использующей выполнение его на компьютере), реализует общий принцип, сформулированный для обсуждаемой технологии программирования, о необходимости контроля принимаемых решений на каждом этапе разработки ПС.

И, наконец, последний шаг разработки модуля означает завершение проверки модуля (с помощью компилятора) и переход к процессу отладки модуля.

1.9. Лекция № 22 (2 часа)

Тема: «Доказательство свойств программ».

1.9.1. Вопросы лекции:

1) Понятие обоснования программ.

2) Формализация свойств программ.

1.9.2. Краткое содержание вопросов:

Для повышения надежности программных средств весьма полезно снабжать программы дополнительной информацией, с использованием которой можно существенно повысить уровень контроля ПС. Такую информацию можно задавать в форме неформализованных или формализованных утверждений, привязываемых к различным фрагментам программ. Будем называть такие утверждения обоснованиями программы. Неформализованные обоснования программ могут, например, объяснять мотивы принятия тех или иных решений, что может существенно облегчить поиск и исправление ошибок, а также изучение программ при их сопровождении. Формализованные же обоснования позволяют доказывать некоторые свойства программ как вручную, так и контролировать (устанавливать) их автоматически.

Одной из используемых в настоящее время концепций формальных обоснований программ является использование так называемых триад Хоора. Пусть S - некоторый обобщенный оператор над информационной средой IS, а P и Q - некоторые предикаты (утверждения) над этой средой. Тогда запись {P}S{Q} и называют триадой Хоора, в которой предикат P называют предусловием, а предикат Q - постусловием относительно оператора S. Говорят, что оператор (в частности, программа) S обладает свойством {P}S{Q}, если всякий раз, когда перед выполнением оператора S истинен предикат P, после выполнения этого оператора S будет истинен предикат Q.

Простые примеры свойств программ:

(9.1) {n=0} n:= n+1{n=1},
(9.2) {n<m} n:= n + k {n<m+k},
(9.3) {n<m+k} n:=3* n {n<3* (m + k)},
(9.4) {n>0} p:=1; m:=1;
ПОКА m <> n ДЕЛАТЬ
m:=m+1; p:= p* m
ВСЕ ПОКА
{p= n!}.

Для доказательства свойства программы S используются свойства простых операторов языка программирования (мы здесь ограничимся пустым оператором и оператором присваивания) и свойствами управляющих конструкций (композиций), с помощью которых строится программа из простых операторов (мы здесь ограничимся тремя основными композициями структурного программирования). Эти свойства называют обычно правилами верификации программ.

1.10. Лекция № 23, 24 (4 часа)
Тема: «Тестирование и отладка программного средства».
1.10.1. Вопросы лекции:
1) Стратегия проектирования тестов.
2) Правила отладки.

1.10.2. Краткое содержание вопросов:
В разработке программ традиционно выделяют следующие этапы: формализация постановки (перевод задачи на язык математической символики), выбор либо разработка методов решения, разработка алгоритма, кодирование (перевод алгоритма на алгоритмический язык в соответствии с его синтаксисом), тестирование, отладка, защита,

сдача в эксплуатацию, сопровождение (авторский надзор). Формализация и алгоритмизация часто носят нестандартный характер и не имеют единых методов. Остановимся подробнее на отладке и тестировании.

Отладка — процесс выявления, локализации и устранения ошибок в алгоритме и реализующей его программе — осуществляется с помощью тестирования. Не будем останавливаться на выявлении синтаксических ошибок — это с разной эффективностью реализуется трансляторами; речь пойдет о программе, которая благополучно транслируется, принимает исходные данные и выдает (хоть какие-то) результаты. Задача — убедиться в корректности алгоритма, то есть получить уверенность, что программа выдает результаты, соответствующие задаче и исходным данным. Показано, что посредством испытаний либо теоретически невозможно доказать правильность программы; можно лишь спровоцировать появление и устранить в ней как можно больше ошибок.

Тест — совокупность исходных данных для программы вместе с ожидаемыми результатами (с учетом формы представления последних). Тесты разрабатываются до, а не вовремя или после разработки программы, дабы избежать провокационного влияния стереотипов алгоритма на тестирование. Готовится не один тест, а их совокупность — *набор тестов,* призванный охватить максимум ситуаций. Испытание программы проводится сразу на всем наборе с протоколированием и анализом результатов.

Набор тестов называется *полным,* если он позволяет активизировать все ветви алгоритма. Набор тестов назовем *не избыточным,* если удаление из него любого теста лишает его полноты. Таким образом, искусство тестирования сводится к разработке полного и не избыточного набора тестов, а технология — к испытанию программы на всем наборе после внесения в нее каждого исправления. Удачно подобранные тесты позволяют не только констатировать факт наличия ошибок, но и *локализовать* их, то есть найти место в программе, виновное в получении неверных результатов.

В наборе тестов выделяют три группы:

• «тепличные» — проверяющие программу при корректных, нормальных исходных данных самого простого вида;

• «экстремальные» — на границе области определения, в ситуациях, которые могут произойти и на которые нужно корректно реагировать;

• «запредельные» — за границей области определения — ситуации, бессмысленные с точки зрения постановки задачи, но которые могут произойти из-за ошибок пользователя или программ, поставляющих исходные данные для тестируемой программы.

Требование надежности программирования: принимать данные, если они корректны, и получать для них правильные результаты либо отвергать их как некорректные, по возможности с анализом некорректности.

Подготовка тестов может оказаться довольно трудоемкой работой; но в некоторых случаях этот процесс можно автоматизировать. Так, некоторые задачи некритичны к содержимому входных массивов; правильность результата можно проверить визуальным сопоставлением с исходными данными, каковы бы они ни были. Один из приемов — *рандомизация* — подготовка случайных исходных данных.

Практически во всех трансляторах реализован *датчик случайных чисел,* — как правило, стандартная функция с именем Random, возвращающая случайное число из интервала [0;1). С помощью простейшего датчика можно получать практически любые распределения. Так, если ξ равномерно распределено на [0;1), то выражение $x=(b-a)\xi+a$ дает действительное число, равномерно распределенное на отрезке *[a;b),* а конструкция на Паскале k:=Ord(p>Random) присваивает переменной k единицу с вероятностью p и нуль — с вероятностью $(1 - p)$. Аналогично можно генерировать монотонные (возрастающие или убывающие) последовательности (например обращение в цикле: a[i]:=a[i-1]+Random

позволяет получить случайный неубывающий массив), а также «замусоренные» массивы — частично искаженные случайным шумом.

Другой прием — программная подготовка тестов в виде массивов или файлов существенной размерности, обладающих заданными свойствами, с известным ожидаемым результатом основной программы. Для этого необходимо написать специальную программу — генератор тестов, но это менее трудоемко, чем подготовка тестов вручную.

Таким образом, защита разработки сводится к демонстрации работоспособности программы на совокупности тестов, а именно к совместной защите набора тестов, алгоритма и программы.

1.11. Лекция № 25 (2 часа)

Тема: «Обеспечение функциональности и надежности программного средства».

1.11.1. Вопросы лекции:

1) Функциональность и надежность как обязательные критерии качества программного средства.

2) Обеспечение завершенности программного средства.

1.11.2. Краткое содержание вопросов:

Завершенность ПС является общим примитивом качества ПС для выражения и функциональности и надежности ПС, причем для функциональности она является единственным примитивом.

Функциональность ПС определяется его функциональной спецификацией. Завершенность ПС как примитив его качества является мерой того, в какой степени эта спецификация реализована в разрабатываемом ПС. Обеспечение этого примитива в полном объеме означает реализацию каждой из функций, определенной в функциональной спецификации, со всеми указанными там деталями и особенностями. Все рассмотренные ранее технологические процессы показывают, как это может быть сделано.

Однако в спецификации качества ПС могут быть определены несколько уровней реализации функциональности ПС: может быть определена некоторая упрощенная (начальная или стартовая) версия, которая должна быть реализована в первую очередь; могут быть также определены и несколько промежуточных версий. В этом случае возникает дополнительная технологическая задача: организация наращивания функциональности ПС. Здесь важно отметить, что разработка упрощенной версии ПС не есть разработка его прототипа. Прототип разрабатывается для того, чтобы лучше понять условия применения будущего ПС, уточнить его внешнее описание. Он рассчитан на избранных пользователей и поэтому может сильно отличаться от требуемого ПС не только выполняемыми функциями, но и особенностями пользовательского интерфейса. Упрощенная же версия разрабатываемого ПС должна быть рассчитана на практически полезное применение любыми пользователями, для которых предназначено это ПС. Поэтому главный принцип обеспечения функциональности такого ПС заключается в том, чтобы с самого начала разрабатывать ПС таким образом, как будто требуется ПС в полном объеме, до тех пор, когда разработчики будут иметь дело непосредственно с теми частями или деталями ПС, реализацию которых можно отложить в соответствии со спецификацией его качества. Тем самым, и внешнее описание и описание архитектуры ПС должно быть разработано в полном объеме. Можно отложить лишь реализацию тех программных подсистем (определенных в архитектуре разрабатываемого ПС), функционирования которых не требуется в начальной версии этого ПС. Реализацию же самих программных подсистем лучше всего производить методом целенаправленной конструктивной реализации, оставляя в начальной версии ПС подходящие имитаторы тех программных модулей, функционирование которых в этой версии не требуется. Допустима также упрощенная реализация некоторых программных модулей, опускающая

реализацию некоторых деталей соответствующих функций. Однако такие модули с технологической точки зрения лучше рассматривать как своеобразные их имитаторы (хотя и далеко продвинутые).

Достигнутый при обеспечении функциональности ПС уровень его завершенности на самом деле может быть не таким, как ожидалось, из-за ошибок, оставшихся в этом ПС. Можно лишь говорить, что требуемая завершенность достигнута с некоторой вероятностью, определяемой объемом и качеством проведенного тестирования. Для того чтобы повысить эту вероятность, необходимо продолжить тестирование и отладку ПС. Однако, оценивание такой вероятности является весьма специфической задачей, которая пока еще ждет соответствующих теоретических исследований.

1.12. Лекция № 26 (2 часа)

Тема: «Обеспечение качества программного средства».

1.12.1. Вопросы лекции:

1) Реализация пользовательского интерфейса и обеспечение легкости применения программного средства.

2) Обеспечение эффективности программного средства.

1.12.2. Краткое содержание вопросов:

Легкость применения, в значительной степени, определяется составом и качеством пользовательской документации, а также некоторыми свойствами, реализуемыми программным путем.

С пользовательской документацией связаны такие примитивы качества ПС, как П-документированность и информативность. Обеспечением ее качества занимаются обычно технические писатели. Этот вопрос будет обсуждаться в следующей лекции. Здесь лишь следует заметить, что там речь будет идти об автономной по отношению к программам документации. В связи с этим следует обратить внимание на широко используемый в настоящее время подход информирования пользователя в интерактивном режиме (в процессе применения программ ПС). Такое информирование во многих случаях оказывается более удобным для пользователя, чем с помощью автономной документации, так как позволяет пользователю без какого-либо поиска вызывать необходимую информацию за счет использования контекста ее вызова. Такой подход к информированию пользователя является весьма перспективным.

Программным путем реализуются такие примитивы качества ПС как коммуникабельность, устойчивость и защищенность. Обеспечение устойчивости и защищенности уже было рассмотрено в предыдущей лекции. Коммуникабельность обеспечивается соответствующей реализацией обработки исключительных ситуаций и созданием подходящего пользовательского интерфейса.

Возбуждение исключительной ситуации во многих случаях означает, что возникла необходимость информировать пользователя о ходе выполнения программы. При этом выдаваемая пользователю информация должна быть простой для понимания (см. лекцию 4). Однако исключительные ситуации возникают обычно на достаточно низком уровне модульной структуры программы, а создать понятное для пользователя сообщение можно, как правило, на более высоких уровнях этой структуры, где известен контекст, в котором были активизированы действия, приведшие к возникновению исключительной ситуации. Обработку исключительных ситуаций внутри модуля мы уже обсуждали в лекции 8. Для обработки возникшей исключительной ситуации в другом модуле приходится принимать не простые решения. Применяемый часто способ передачи информации о возникшей исключительной ситуации по цепочке обращений к программным модулям (в обратном направлении) является тяжеловесным: он требует дополнительных проверок после возврата из модуля и часто усложняет само обращение к этим модулям за счет задания дополнительных параметров. Приемлемым решением является включение в

операционную среду выполнения программ (в исполнительную поддержку) возможностей прямой передачи этой информации обработчикам исключительных ситуаций по динамически формируемой очереди таких обработчиков.

Пользовательский интерфейс представляет средство взаимодействия пользователя с ПС. При разработке пользовательского интерфейса следует учитывать потребности, опыт и способности пользователя. Поэтому потенциальные пользователи должны быть вовлечены в процесс разработки такого интерфейса. Большой эффект здесь дает его прототипирование. При этом пользователи должны получить доступ к прототипам пользовательского интерфейса, а их оценка различных возможностей используемого прототипа должна существенно учитываться при создании окончательного варианта пользовательского интерфейса.

В силу большого разнообразия пользователей и видов ПС существует множество различных стилей пользовательских интерфейсов, при разработке которых могут использоваться разные принципы и подходы. Однако следующие важнейшие принципы следует соблюдать всегда:

- пользовательский интерфейс должен базироваться на терминах и понятиях, знакомых пользователю;
- пользовательский интерфейс должен быть единообразным;
- пользовательский интерфейс должен позволять пользователю исправлять собственные ошибки;
- пользовательский интерфейс должен позволять получение пользователем справочной информации: как по его запросу, так и генерируемой ПС.

2. МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ПРОВЕДЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

2.1. Лабораторная работа №1, 2 (4 часа).

Тема: «Понятие о программном средстве».

2.1.1. Вопросы к занятию:

1) Понятие ошибки в программном средстве.

2) Надежность программного средства.

3) Технология программирования как технология разработки надежных программных средств.

2.1.2. Краткое описание проводимого занятия:

2.1.2.1. Ответы на вопросы.

2.1.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Фактический аргумент – это:

1) конкретное значение, присвоенное этой переменной вызывающей программой

2) переменная в вызываемой программе

3) строка, которая пишется в скобках функции

4) строка, которая пишется в скобках процедуры

2. Тип функции определяется:

1) типом ее аргументов

2) использованием в программе

3) типом ее описания

4) типом возвращаемого ею значения

2.2. Лабораторная работа № 3, 4 (4 часа).

Тема: «Источники ошибок в программных средствах».

2.2.1. Вопросы к занятию:

1) Неправильный перевод информации из одного представления в другое - основная причина ошибок при разработке программных средств.

2) Модель перевода и источники ошибок.

2.2.2. Краткое описание проводимого занятия:

2.2.2.1. Ответы на вопросы.

2.2.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Автоматические объекты:

1) существуют во время выполнения блока и теряют свои значения при выходе из него

2) хранятся вне любой функции и существуют в течение выполнения всей программы

3) являются объектами статического класса памяти

4) можно инициализировать только выражениями с константами и с указателями на ранее описанные объекты

2. Макровызов должен состоять:

1) из списка макросов

2) из списка макропеременных

3) из списка макроимен

4) из макроимени и заключенного, в круглые скобки списка аргументов

2.3. Лабораторная работа № 5, 6, 7, 8 (8 часов).

Тема: «Специфика разработки программных средств».

2.3.1. Вопросы к занятию:

1) Методы борьбы со сложностью.

2) Обеспечение точности перевода.

3) Преодоление барьера между пользователем и разработчиком.

2.3.2. Краткое описание проводимого занятия:

2.3.2.1. Ответы на вопросы.

2.3.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Альтернатива – это:
   1) композиция разных действий
   2) вариант
   3) конструкция ветвления
   4) шаг выполнения программы
2. Итерация – это:
   1) шаг выполнения программы
   2) циклическая конструкция алгоритма
   3) язык программирования
   4) функция прерывания

2.4. Лабораторная работа № 9, 10, 11 (6 часов).

Тема: «Понятие внешнего описания».

2.4.1. Вопросы к занятию:

1) Основные примитивы качества программного средства.

2) Функциональная спецификация программного средства.

2.4.2. Краткое описание проводимого занятия:

2.4.2.1. Ответы на вопросы.

2.4.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Дедуктивный принцип – это:

1) когда определяется связь между входными, выходными данными и процессами обработки

2) принцип построения модели от частного к общему

3) упрятывание информации и абстрактных типов данных

4) принцип построения модели от общего к частному

2. Индуктивный принцип – это:

1) когда определяется связь между входными, выходными данными и процессами обработки

2) принцип построения модели от частного к общему

3) упрятывание информации и абстрактных типов данных

4) принцип построения модели от общего к частному

2.5. Лабораторная работа № 12, 13, 14 (6 часов).

Тема: «Методы спецификации семантики функций».

2.5.1. Вопросы к занятию:

1) Алгебраический подход: операционная, денотационная и аксиоматическая семантики.

2) Языки спецификаций.

2.5.2. Краткое описание проводимого занятия:

2.5.2.1. Ответы на вопросы.

2.5.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Композиция – это:

1) циклическая конструкция алгоритма, состоящая из многократного повторения одного блока действий

2) линейная конструкция алгоритма, состоящая из последовательно следующих друг за другом функциональных вершин

3) конструкция ветвления, имеющая предикатную вершину

4) механизм языка, позволяющий описать новый класс на основе существующего (родительского) класса

2. Тестирование программы – это:

1) оценивание ресурсов компьютера, на котором будет работать программа

2) перевод проекта в форму программы для конкретного компьютера

3) системный подход к построению алгоритма с использованием декомпозиции и синтеза

4) процесс исполнения программы с целью выявления ошибок

2.6. Лабораторная работа № 15, 16, 17 (6 часов).

Тема: «Архитектура программного средства».

2.6.1. Вопросы к занятию:

1) Взаимодействие между подсистемами и архитектурные функции.

2) Контроль архитектуры программных средств.

2.6.2. Краткое описание проводимого занятия:

2.6.2.1. Ответы на вопросы.

2.6.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Инспекция при тестировании – это:

1) надзор за изменением состояний переменных

2) отслеживание логических ошибок

3) набор процедур и приемов обнаружения ошибок

4) надзор за соответствием типов и атрибутов переменных

2. Граничные условия в тестах – это:

1) ситуации, возникающие на, выше или ниже границ входных и выходных классов эквивалентности

2) тестовые задания, имеющие наивысшую вероятность обнаружения ошибок

3) выход индексов заданий за пределы допустимых

4) начальные и конечные условия границы применимости теста

2.7. Лабораторная работа № 18, 19 (4 часа).

Тема: «Разработка структуры программы».

2.7.1. Вопросы к занятию:

1) Методы разработки структуры программы.

2) Спецификация программного модуля.

2.7.2. Краткое описание проводимого занятия:

2.7.2.1. Ответы на вопросы.

2.7.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Если данные размещены на внешнем носителе, то доступ к ним возможен:
1) моментальный
2) прямой
3) последовательный
4) выборочный
2. Процедура линейного поиска – это:
1) просмотр массива с конца
2) просмотр массива с середины
3) сравнение эталона осуществляется с элементом, расположенным в середине массива
4) последовательный просмотр всех элементов массива и сравнение их с эталоном

2.8. Лабораторная работа № 20,21 (4 часа).
Тема: «Разработка программного модуля».
2.8.1. Вопросы к занятию:
1) Понятие о псевдокоде.
2) Контроль программного модуля.

2.8.2. Краткое описание проводимого занятия:
2.8.2.1. Ответы на вопросы.
2.8.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
1. Дан алгоритм сортировки: определяется минимальный элемент сред всех и меняется местами с первым и т. д., начиная со второго. Вид сортировки:
1) метод прямого включения
2) метод прямого выбора
3) пузырьковый метод
4) с помощью «дерева»
2. Деструктивность процесса тестирования проявляется в следующем:
1) тест удачный, если обнаружена ошибка
2) тест удачный, если проведен без ошибок
3) тест неудачный, если обнаружена еще не выявленная ошибка
4) тест неудачный, если все задания некорректны

2.9. Лабораторная работа № 22 (2 часа).
Тема: «Доказательство свойств программ».
2.9.1. Вопросы к занятию:
1) Правила для установления свойств оператора присваивания, условного и составного операторов.
2) Правила для установления свойств оператора цикла, понятие инварианта цикла.

2.9.2. Краткое описание проводимого занятия:
2.9.2.1. Ответы на вопросы.
2.9.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
1. Тестирование программы как черного ящика заключается в следующем:
1) знаем, какие данные будут на выходе
2) не знаем, какие данные подаем на входе
3) анализ входных данных и результатов работы программы
4) управляем логикой программы, используя ее внутреннюю структуру
2. Тестирование программы как белого ящика заключается в следующем:
1) не знаем, какие данные будут на выходе

2) не знаем, как получаются данные на выходе
3) анализ входных данных и результатов работы программы
4) управляем логикой программы, используя ее внутреннюю структуру

2.10. Лабораторная работа № 23, 24 (4 часа).
Тема: «Тестирование и отладка программного средства».
2.10.1. Вопросы к занятию:
1) Автономная отладка и тестирование программного модуля.
2) Комплексная отладка и тестирование программного средства.

2.10.2. Краткое описание проводимого занятия:
2.10.2.1. Ответы на вопросы.
2.10.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
1. Цель декомпозиции является:
1) определение связи между модулями
2) процедурное описание программы
3) создание модулей, которые взаимодействуют друг с другом по определенным правилам
4) неформальное описание модуля: обзор действий
2. В методологии Джексона предусматривается
1) определение структуры программы структурой данных, подлежащих обработке
2) связь между входными, выходными данными и процессом обработки с помощью иерархической декомпозиции
3) техника упрятывания или сокрытия информации и абстрактных типов данных
4) объектно-ориентированное программирование, в основе которого лежит понятия объекта и класса

2.11. Лабораторная работа № 25 (2 часа).
Тема: «Обеспечение функциональности и надежности программного средства».
2.11.1. Вопросы к занятию:
1) Защитное программирование и обеспечение устойчивости программного модуля.
2) Виды защиты и обеспечение защищенности программного средства.

2.11.2. Краткое описание проводимого занятия:
2.11.2.1. Ответы на вопросы.
2.11.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
1. Метод иерархических диаграмм предусматривает:
1) определение структуры программы структурой данных, подлежащих обработке
2) связь между входными, выходными данными и процессом обработки с помощью иерархической декомпозиции
3) техника упрятывания или сокрытия информации и абстрактных типов данных
4) объектно-ориентированное программирование, в основе которого лежит понятия объекта и класса
2. Оператор – это:
1) функция, которая оперирует с данными
2) законченная фраза языка, предписание, команда
3) алгоритм действия программы, написанной на данном языке
4) процедура обработки данных

2.12. Лабораторная работа № 26 (2 часа).
Тема: «Обеспечение качества программного средства».
2.12.1. Вопросы к занятию:
1) Обеспечение сопровождаемости и управление конфигурацией программного средства.
2) Аппаратно-операционные платформы и обеспечение мобильности программного средства.

2.12.2. Краткое описание проводимого занятия:
2.12.2.1. Ответы на вопросы.
2.12.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
1. Текстовый поток – это:
   1) логическое понятие, которое система может относить от дисковых файлов до терминалов
   2) последовательность символов, которая организуется в строки, завершающиеся символами новой строки
   3) последовательность символов, которая организуется в списке слов, завершающиеся точкой с запятой
   4) последовательность символов, в которых символы преобразуются согласно требованиям среды
2. Формальный аргумент – это:
   1) конкретное значение, присвоенное этой переменное вызывающей программой
   2) переменная в вызываемой программе
   3) строка, которая пишется в скобках функции
   4) строка, которая пишется в скобках процедуры

3. МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ПРОВЕДЕНИЮ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

3.1. Практическое занятие №1, 2 (4 часа).

Тема: «Понятие о программном средстве».

3.1.1. Вопросы к занятию:

1) Понятие ошибки в программном средстве.

2) Надежность программного средства.

3) Технология программирования как технология разработки надежных программных средств.

3.1.2. Краткое описание проводимого занятия:

3.1.2.1. Ответы на вопросы практического занятия.

3.1.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1.1. Что такое информационная среда программы?

1.2. Что такое программное средство (ПС)?

1.3. Что такое ошибка в ПС?

1.4. Что такое надежность ПС?

1.5. Что такое технология программирования?

3.2. Практическое занятие № 3, 4 (4 часа).

Тема: «Источники ошибок в программных средствах».

3.2.1. Вопросы к занятию:

1) Неправильный перевод информации из одного представления в другое - основная причина ошибок при разработке программных средств.

2) Модель перевода и источники ошибок.

3.2.2. Краткое описание проводимого занятия:

3.2.2.1. Ответы на вопросы практического занятия.

3.2.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

2.1. Что такое простая и сложная системы?

2.2. Что такое малая и большая системы?

3.3. Практическое занятие № 5, 6, 7, 8 (8 часов).

Тема: «Специфика разработки программных средств».

3.3.1. Вопросы к занятию:

1) Методы борьбы со сложностью.

2) Обеспечение точности перевода.

3) Преодоление барьера между пользователем и разработчиком.

3.3.2. Краткое описание проводимого занятия:

3.3.2.1. Ответы на вопросы практического занятия.

3.3.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

3.1. Что такое жизненный цикл программного средства (ПС)?

3.2. Что такое внешнее описание ПС?

3.3. Что такое сопровождение ПС?

3.4. Что такое качество ПС?

3.5. Что такое смежный контроль?

3.4. Практическое занятие № 9, 10, 11 (6 часов).

Тема: «Понятие внешнего описания».

3.4.1. Вопросы к занятию:

1) Основные примитивы качества программного средства.

2) Функциональная спецификация программного средства.

3.4.2. Краткое описание проводимого занятия:

3.4.2.1. Ответы на вопросы практического занятия.

3.4.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

4.1. Что такое определение требований к программному средству (ПС)?

4.2. Что такое спецификации качества ПС?

4.3. Что такое устойчивость (*robustness*) *ПС?*

4.4. Что такое защищенность (*defensiveness*) *ПС?*

4.5. Что такое коммуникабельность (*communicativeness*) *ПС?*

4.6. Что такое функциональная спецификация ПС?

4.7. Что такое ручная имитация внешнего описания ПС?

3.5. Практическое занятие № 12, 13, 14 (6 часов).

Тема: «Методы спецификации семантики функций».

3.5.1. Вопросы к занятию:

1) Алгебраический подход: операционная, денотационная и аксиоматическая семантики.

2) Языки спецификаций.

3.5.2. Краткое описание проводимого занятия:

3.5.2.1. Ответы на вопросы практического занятия.

3.5.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

5.1. Функции

function F(x, y: integer): integer;

function G(x, y: integer): integer;

function R(x, y: integer): integer;

определены с помощью операционной семантики равенствами:

R(x, y) = x(y – 1),*

F(x, y) = R(x + 1, y) - R(x, y - 1),

G(x, y) = F(x, R(x, y)).

Найти значения G(3, 3).

5.2. Функции

function F(n: integer): integer;

function G(n: integer): integer;

определены с помощью операционной семантики равенствами:

F(0)=1,

G(0)=2,

F(n)=G(n-1),

G(n)=F(n-1) + G(n-1).

Найти значения F(3) и G(3).

5.3. Формальные языки E и T определены над алфавитом

{'a', '', '&', '<', '>'}*

с помощью денотационной семантики равенствами

E= T □ '' T □ E '&' T,*

T= 'a' □ 'a' □ '<' E '>'*

Какие из следующих строк
*'*a&*a*&a*' ,*
*'*a&<a&a*>',*
'<*a*&a>&<*a*>*'*
принадлежат языку E и какие из них не принадлежат языку E.
5.4. Тип R определён с помощью следующей аксиоматической семантики.
Описания:
type R= record P1, P2, P3: CHAR end;
function READ(S: R): CHAR; {READ: R → CHAR}
function SHIFT(S: R): R; {SHIFT: R → R}
*function ADD(S: R, C: CHAR): R; {ADD: R * CHAR → R}*
function REMOVE(S: R): R; {REMOVE: R → R}
var X, Y, Z: CHAR;
U: R;
Аксиомы:
SHIFT(ADD(ADD(ADD(U, X), Y), Z)) =
ADD(ADD(ADD(U,Y), Z), X);
REMOVE(U) = SHIFT(ADD(U, '#'));
READ(SHIFT(ADD(U, X))) = X;
Найти значение:
READ(SHIFT(SHIFT(REMOVE(ADD(ADD(U, 'a'), 'b'))))) =

3.6. Практическое занятие № 15, 16 (4 часа).
Тема: «Архитектура программного средства».
3.6.1. Вопросы к занятию:
1) Взаимодействие между подсистемами и архитектурные функции.
2) Контроль архитектуры программных средств.

3.6.2. Краткое описание проводимого занятия:
3.6.2.1. Ответы на вопросы практического занятия.
3.6.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
6.1. Что такое архитектура программного средства?
6.2. Что такое архитектурная функция?

3.7. Практическое занятие № 17, 18, 19, 20 (8 часов).
Тема: «Разработка структуры программы».
3.7.1. Вопросы к занятию:
1) Методы разработки структуры программы.
2) Спецификация программного модуля.

3.7.2. Краткое описание проводимого занятия:
3.7.2.1. Ответы на вопросы практического занятия.
3.7.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
7.1. Что такое программный модуль?
7.2. Что такое прочность программного модуля?
7.3. Что такое сцепление программного модуля?

3.8. Практическое занятие № 21, 22, 23, 24 (8 часов).
Тема: «Разработка программного модуля».
3.8.1. Вопросы к занятию:

1) Понятие о псевдокоде.
2) Контроль программного модуля.

3.8.2. Краткое описание проводимого занятия:
3.8.2.1. Ответы на вопросы практического занятия.
3.8.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
8.1. Что такое структурное программирование?
8.2. Что такое пошаговая детализация программного модуля?
8.3. Что такое псевдокод?

3.9. Практическое занятие № 25, 26 (4 часа).
Тема: «Доказательство свойств программ».
3.9.1. Вопросы к занятию:
1) Правила для установления свойств оператора присваивания, условного и составного операторов.
2) Правила для установления свойств оператора цикла, понятие инварианта цикла.

3.9.2. Краткое описание проводимого занятия:
3.9.2.1. Ответы на вопросы практического занятия.
3.9.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
9.1. Что такое триада Хоора?
9.2. Что такое свойство программы?
9.3. Пусть заданы описания
const n= <конкретное целое значение>;
var k, m: integer;
x: array[1..n] of integer;

Доказать свойство программы:
{n>0}
m:= x[1]
k:=1;
ПОКА k<n ДЕЛАТЬ
k:= k+1;
ЕСЛИ x[k]<m ТО
m:= x[k]
ВСЕ ЕСЛИ
ВСЕ ПОКА;
{n>0 & m<= x[i] для всех i, 1<=i<= n}

3.10. Практическое занятие № 27, 28, 29, 30 (8 часов).
Тема: «Тестирование и отладка программного средства».
3.10.1. Вопросы к занятию:
1) Автономная отладка и тестирование программного модуля.
2) Комплексная отладка и тестирование программного средства.

3.10.2. Краткое описание проводимого занятия:
3.10.2.1. Ответы на вопросы практического занятия.
3.10.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости

10.1. Что такое отладка программного средства?

10.2. Что такое тестирование программного средства?

10.3. Что такое автономная отладка программного средства?

10.4. Что такое комплексная отладка программного средства?

10.5. Что такое ведущий отладочный модуль?

10.6. Что такое отладочный имитатор программного модуля?

3.11. Практическое занятие № 31, 32, 33 (6 часов).

Тема: «Обеспечение функциональности и надежности программного средства».

3.11.1. Вопросы к занятию:

1) Защитное программирование и обеспечение устойчивости программного модуля.

2) Виды защиты и обеспечение защищенности программного средства.

3.11.2. Краткое описание проводимого занятия:

3.11.2.1. Ответы на вопросы практического занятия.

3.11.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

11.1. Что такое защитное программирование?

11.2. Какие виды защиты программного средства от искажения информации Вы знаете?

11.3. Какие требования предъявляются к компьютеру, чтобы можно было обеспечить защиту программы от отказов другой программы в мультипрограммном режиме?

11.4. Что такое компьютерная подпись?

11.5. Что такое компьютерная печать?

3.12. Практическое занятие № 34, 35 (4 часа).

Тема: «Обеспечение качества программного средства».

3.12.1. Вопросы к занятию:

1) Обеспечение сопровождаемости и управление конфигурацией программного средства.

2) Аппаратно-операционные платформы и обеспечение мобильности программного средства.

3.12.2. Краткое описание проводимого занятия:

3.12.2.1. Ответы на вопросы практического занятия.

3.12.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

12.1. Какие задачи приходится решать при обеспечении коммуникабельности ПС?

12.2. Какие возможности предоставляет пользователю графический пользовательский интерфейс?

12.3. Как нужно действовать для обеспечения эффективности ПС?

12.4. Что такое инсталятор программного средства (ПС)?

12.5. Что такое управление конфигурацией ПС?

12.6. Что такое ядро ПС?

12.7. Что такое оболочка ПС?