УВАЖАЕМЫЕ СТУДЕНТЫ! Изучите приведенную лекцию, ответьте *письменно* на контрольные вопросы.

Ответы на вопросы, фотоотчет, предоставить преподавателю на e-mail: r.bigangel@gmail.com до 23.01.2023.

При возникновении вопросов по приведенному материалу обращаться по следующему номеру телефона: (072)111-37-59, (Viber, WhatsApp), vk.com: https://vk.com/daykini

ВНИМАНИЕ!!! При отправке работы, не забывайте указывать ФИО студента, наименование дисциплины, дата проведения занятия (по расписанию).

Лекция 25

Тема: Модульное программирование

Цель: Изучить основные характеристики и приемы модульного программирования

Программу для ее упрощения разрабатывают по частям, которые называются программными модулями. Такой метод разработки программ называют модульным программированием.

Программный модуль – фрагмент описания процесса, оформляемый как самостоятельный программный продукт, пригодный для использования в описаниях разных процессов. Программный модуль программируется, компилируется и отлаживается отдельно; может включаться в состав разных программ; является средством борьбы со сложностью программ; является средством борьбы с дублированием в программировании.

Основными характеристиками программного модуля являются:

- размер;
- связность;
- сцепление с другими модулями;
- рутинность.

Размер модуля измеряется числом содержащихся в нем операторов или строк. Модуль не должен быть слишком маленьким или слишком большим. Маленькие модули приводят к громоздкой модульной структуре программы. Большие модули неудобны для изучения и изменений, могут существенно увеличить суммарное время повторных трансляций программы при ее отладке.

Отладка модуля размером в одну страницу может быть в разы проще отладки модуля размером в одну страницу и еще 4-5 строк на другой странице. Это связано с принципами организации человеческой памяти. Есть сверхоперативная память, связанная, в основном, со зрением. Эта память имеет очень быстрый доступ, но очень мала – 7-9 позиций. Существенно больше оперативная память, в которой и происходит вся основная мыслительная деятельность, но данные в ней не могут храниться долго. Наконец, самая большая — долговременная память. Человеку непросто заложить туда данные, но хранятся они долго.

С устройством памяти связан принцип *центрального зрения*. Человек хорошо воспринимает какую-то точку и то, что ее окружает. Если при отладке программы автор должен обозревать больше, чем одну небольшую страницу текста, он не может полноценно воспринять программу - *листать вредно*.

Связность модуля - мера зависимости его частей, внутренняя характеристика. Чем выше связность модуля, тем больше связей он скрывает от внешней части программы и больший вклад в упрощение программы вносит. Для оценки степени связности модуля используется семь типов связности :

Рутинность модуля. Модуль называется *рутинным*, если результат обращения к нему зависит только от значений его параметров. Модуль называется *зависящим от предыстории*, если результат обращения к нему зависит от внутреннего состояния этого модуля, изменяемого в результате предыдущих обращений к нему. Не рекомендуется использовать зависящие от предыстории модули, так как они провоцируют появление в программах неуловимых ошибок.

Методы разработки структуры программы

В качестве модульной структуры программы принято использовать древовидную структуру. В узлах такого дерева размещаются программные модули, а направленные дуги показывают подчиненность модулей.

В процессе разработки программы ее модульная структура может формироваться и использоваться по-разному для определения порядка программирования и отладки модулей, указанных в этой структуре. Обычно рассматриваются два метода [1, 4, 7]:

- метод восходящей разработки;
- метод нисходящей разработки.

Метод *восходящей разработки* заключается в следующем. Сначала строится модульная структура программы в виде дерева. Затем поочередно программируются модули программы, начиная с модулей самого нижнего уровня, в таком порядке, чтобы для каждого программируемого модуля были уже запрограммированы все модули, к которым он может обращаться. После того, как все модули программы запрограммированы, производится их тестирование и отладка в порядке, в каком велось их программирование.

Технологией программирования не рекомендуется восходящий порядок разработки программы по следующим причинам.

- Каждая программа подчиняется некоторым внутренним для нее, но глобальным для ее модулей соображениям. При восходящей разработке эта глобальная информация для модулей нижних уровней еще не ясна в полном объеме. Часто приходится перепрограммировать модули, когда уточняется эта информация.
- При восходящем тестировании для каждого модуля приходится создавать ведущую программу, которая подготавливает для тестируемого модуля необходимое состояние информационной среды и производит требуемое обращение к нему. Это приводит к большому объему *отладочного* программирования.

Метод *нисходящей разработки* заключается в следующем. Как и в

предыдущем методе сначала строится модульная структура программы в виде дерева. Затем поочередно программируются модули программы, начиная с верхнего уровня. Переход к программированию следующего модуля происходит в том случае, если запрограммирован модуль, который к нему обращается. После того, как все модули программы запрограммированы, производится их поочередное тестирование и отладка в таком же порядке.

Первым тестируется головной модуль программы при том состоянии информационной среды, при котором начинает выполняться эта программа. Те модули, к которым может обращаться головной, заменяются их имитаторами. Имитатор модуля представляется программой, которая сигнализирует о факте обращения к имитируемому модулю.

После завершения тестирования и отладки головного и любого последующего модуля производится переход к тестированию модулей, которые представлены имитаторами. Для этого имитатор заменяется самим этим модулем и добавляются имитаторы тех модулей, к которым он может обращаться. При этом каждый такой модуль будет тестироваться при тех состояниях информационной среды, которые возникают к моменту обращения к этому модулю при выполнении тестируемой программы. Таким образом, большой объем *отладочного* программирования при восходящем тестировании заменяется программированием простых имитаторов.

Особенностью классических методов восходящей и нисходящей разработок является требование, чтобы модульная структура программы была разработана до начала программирования модулей. Это требование соответствует водопадному подходу к разработке ПС. Однако, не всегда до программирования модулей можно точно и содержательно разработать структуру программы. *Конструктивный* и *архитектурный* подходы к разработке программ предлагают формирование модульной структуры в процессе программирования модулей.

Конструктивный подход к разработке программы представляет собой модификацию нисходящей разработки, при которой модульная древовидная структура программы формируется в процессе программирования модулей. Разработка программы при конструктивном подходе начинается с программирования головного модуля исходя из спецификации программы в целом. Если эта программа большая, выделяются подзадачи.

Для каждой выделяемой подзадачи создается спецификация реализующего ее фрагмента программы (поддерева модулей). В головном модуле программы строится обращение к головному модулю указанного поддерева. Таким образом, на первом шаге разработки программы (при программировании ее головного модуля) формируется верхняя начальная часть дерева. Аналогичные действия производятся при программировании любого другого модуля, который выбирается из текущего состояния дерева программы.

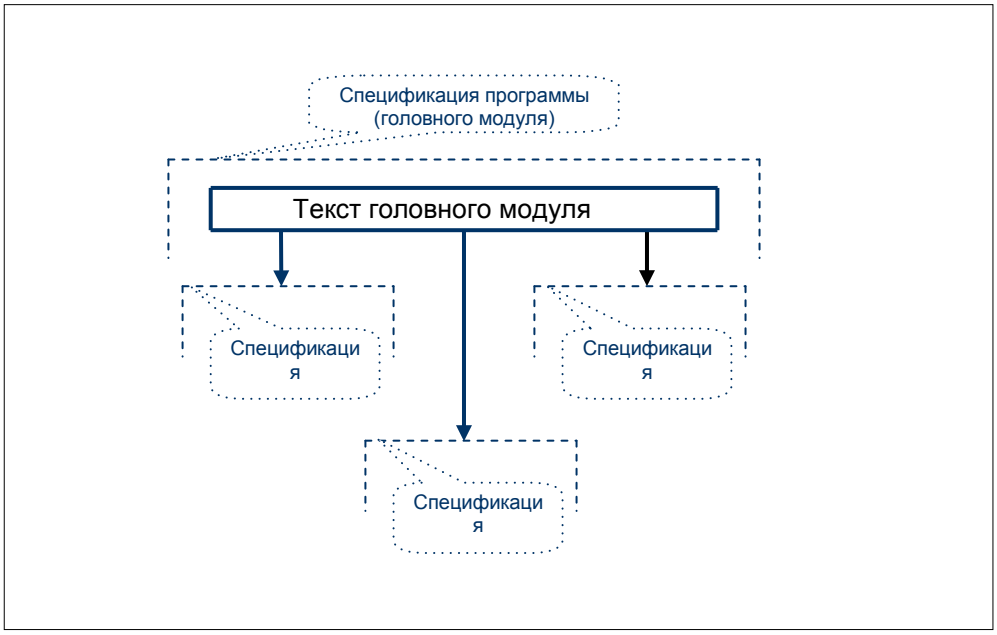

Рис. 1. Первый шаг формирования модульной структуры программы при конструктивном подходе

Целенаправленная конструктивная реализация предлагает при обходе дерева программы сначала реализовать те модули, которые необходимы для простейшего варианта программы. Вместо модулей, на которые в такой программе имеются ссылки, в нее вставляются их имитаторы. Затем к этой программе добавляются реализации этих имитаторов. Этот процесс продолжается поэтапно до полной реализации требуемой программы. Таким образом, обход дерева программы производится с целью кратчайшим путем реализовать вариант действующей программы. Достоинством этого метода является то, что уже на ранней стадии создается работающий вариант разрабатываемой программы.

Архитектурный подход к разработке программы представляет собой модификацию восходящей разработки. Для заданной предметной области выделяются типичные функции, каждая из которых может использоваться при решении разных задач в этой области. Они программируются отдельными программными модулями. Это позволяет сократить трудозатраты на разработку программы путем подключения к ней заранее заготовленных модульных структур. Так как такие структуры могут многократно использоваться в разных программах, то архитектурный подход рассматривается как путь борьбы с дублированием в программировании.

Контроль структуры программы

Для контроля структуры программы используются три метода:

Статический контроль состоит в оценке структуры программы, насколько хорошо программа разбита на модули с учетом значений основных характеристик модуля.

Сквозной контроль - это мысленное прокручивание структуры программы при выполнении заранее разработанных тестов. Он является видом динамического контроля так же, как и имитация функциональной спецификации или архитектуры ПС.

Смежный контроль сверху - это контроль со стороны разработчиков архитектуры и внешнего описания ПС.

Смежный контроль снизу - это контроль спецификации модулей со стороны разработчиков этих модулей.

Разработка программного модуля

При разработке программного модуля принято придерживаться следующего порядка:

1. Изучение и проверка спецификации модуля, выбор языка программирования;
2. выбор алгоритма и структуры данных;
3. программирование модуля;
4. шлифовка текста модуля;
5. проверка модуля;
6. компиляция модуля.

Первый шаг разработки представляет собой смежный контроль структуры программы снизу. В завершении этого шага выбирается язык программирования. Хотя язык программирования может быть уже предопределен для всего ПС.

На втором шаге разработки выясняют, какие известны алгоритмы для решения поставленной задачи. Выбор структур данных, которые будут использоваться при выполнении модулем своих функций, предопределяет логику и качественные показатели модуля.

На шаге программирования модуля осуществляется построение текста модуля на выбранном языке программирования. Важно для построения текста модуля пользоваться технологически обоснованной дисциплиной программирования. Наиболее распространенной является дисциплина пошаговой детализации, базирующаяся на принципах структурного программирования.

При программировании модуля разработчик основное внимание уделяет правильности реализации функций модуля, оставляя недоработанными комментарии и допуская нарушения требований к стилю программы. При шлифовке текста модуля он должен отредактировать имеющиеся в тексте комментарии и включить в него дополнительные комментарии с целью обеспечить требуемые примитивы качества.

Шаг проверки модуля представляет собой ручную проверку внутренней логики модуля до начала его отладки (выполнение его на компьютере).

Последний шаг разработки модуля означает завершение проверки модуля (с помощью компилятора) и переход к процессу отладки модуля.

Структурное программирование

Программа должна быть понятной не только компьютеру, но и человеку. Поэтому необходимо следовать определенной дисциплине программирования. Дийкстра предложил строить программу как композицию из нескольких типов управляющих конструкций, которые позволяют повысить понятность логики работы программы. Программирование с использованием только таких конструкций назвали *структурным*. Основными конструкциями структурного программирования являются: *следование, разветвление, повторение* [1, 4].

Для структурированных программ можно математически доказывать определенные свойства, что позволяет обнаруживать в программе некоторые

ошибки.

Структурное программирование иногда называют *программированием без GO TO*. Рекомендуется избегать употребления оператора перехода, который запутывает программу. К полезным случаям использования оператора перехода можно отнести выход из цикла или процедуры по особому условию, т.е. завершающего работу некоторой структурной единицы и тем самым лишь локально нарушающего структурированность программы.

Усложнение структуры вызывает обработка исключительных ситуаций, так как при этом требуется не только осуществить досрочный выход из структурной единицы, но и произвести обработку этой ситуации. Обработчики исключительных ситуаций помещаются в конце структурной единицы, и каждый такой обработчик программируется таким образом, что после окончания своей работы производит выход из этой структурной единицы.

Пошаговая детализация

Иногда программирование модуля начинают с построения его блок-схемы, наглядно описывающей логику его работы [4]. Однако современная ТП не рекомендует этого делать, так как при его кодировании возникают ошибки из-за отображения двумерных структур блок-схем, на линейный текст модуля, что может исказить логику работы модуля.

Исключением может быть случай, когда для построения блок-схем используется графический редактор, и блок-схемы становятся формализованы настолько, что по ним автоматически генерируется текст на языке программирования (Р - технологии).

В качестве основного метода построения текста модуля ТП рекомендуется *пошаговая детализация*. Сущность этого метода заключается в разбиении процесса разработки текста модуля на ряд шагов [1, 4].

На первом шаге описывается общая схема работы модуля в линейной текстовой форме (с использованием крупных понятий). Это описание не является формализованным и ориентировано на восприятие его человеком.

На каждом следующем шаге производится детализация понятий, разработанных на предыдущих шагах. В результате создается описание выбранного уточняемого понятия либо в терминах базового ЯП, либо в такой же форме, что и на первом шаге с использованием новых уточняемых понятий.

Этот процесс завершается, когда все уточняемые понятия будут в конечном счете выражены на базовом ЯП.

Пример: Удаление в файле записей до первой, удовлетворяющей заданному фильтру.

```
Установить начало файла ПОКА не конец файла ДЕЛАТЬ
    Прочитать очередную запись
    ЕСЛИ очередная запись удовлетворяет фильтр
    ТО
    ВЫЙТИ
    ИНАЧЕ
    Удалить очередную запись из файла.
    ВСЕ ЕСЛИ
    ВСЕ ПОКА
```

ЕСЛИ записи не удалены ТО
НАПЕЧАТАТЬ "Записи не удалены".
ИНАЧЕ
НАПЕЧАТАТЬ "Удалено N записей".
ВСЕ ЕСЛИ

Рекомендуется на каждом шаге детализации создавать достаточно содержательное описание, но легко обозримое, так чтобы оно размещалось на одной странице текста. В результате можно получить описание логики работы по наглядности вполне конкурентное с блок-схемами, но обладающее преимуществом - сохраняется линейность описания.

Контроль программного модуля

Применяются следующие методы контроля программного модуля

Статическая проверка текста модуля предполагает прочитывание его с начала до конца с целью найти ошибки в модуле. Обычно для такой проверки привлекают, кроме разработчика модуля, еще одного или даже нескольких программистов.

Сквозное прослеживание представляет собой один из видов динамического контроля модуля на некотором наборе тестов.

Контрольные вопросы

1. В чем заключается *цель модульного программирования*?
2. Назовите основные *характеристики программного модуля*.
3. Что определяет *связность модуля*?
4. Перечислите названия модулей с разными *степенями связности* по степени их возрастания.
5. Что такое *сцепление модуля*?
6. Какой модуль называется *рутинным*? Какой модуль зависит от *предыстории*?
7. Какая *модульная структура программы* используется в ТП?
8. Перечислите *классические методы разработки* структуры программы. Назовите предпочтительный метод.
9. Суть *конструктивного подхода разработки* структуры программы?
10. Суть *целенаправленной конструктивной реализации* разработки структуры программы?
11. Суть *архитектурного подхода разработки* структуры программы?
12. Какие существуют *методы контроля структуры программы*?
13. Перечислите *шаги разработки программного модуля*.
14. Основная суть *структурного программирования*?
15. Основные *конструкции структурного программирования*?
16. Почему в структурном программировании не рекомендуется использовать *оператор GOTO*?
17. Когда *оператор GOTO* рекомендуется использовать?
18. Почему построение *блок-схем* не рекомендуется при программировании модуля?
19. В каких случаях построение *блок-схем* эффективно при программировании модуля?

20. Суть *метода пошаговой детализации* при построении текста модуля?

21. Что описывается на *первом шаге пошаговой детализации* при построении текста модуля?

22. Чем завершается *метод пошаговой детализации* построения текста модуля?

23. В чем заключается *статическая проверка текста модуля*?

24. Суть *сквозного прослеживания текста модуля*?