

## *Лекция 8.* **РАЗРАБОТКА ПРОГРАММНОГО МОДУЛЯ**

*Порядок разработки программного модуля. Структурное программирование и пошаговая детализация. Понятие о псевдокоде. Контроль программного модуля.*

### **8.1. Порядок разработки программного модуля.**

При разработке программного модуля целесообразно придерживаться следующего порядка [8.1]:

- 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 [8.2], сформулировав и обосновав основные принципы структурного программирования. На этих принципах базируются многие дисциплины программирования, широко применяемые на практике [8.3-8.6]. Наиболее распространенной является дисциплина пошаговой детализации [8.3], которая подробно обсуждается в разделах 8.2 и 8.3.

Следующий шаг разработки модуля связан с приведением текста модуля к завершеному 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 [8.1]. 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 (см. лекцию 3). Методы проверки модуля обсуждаются в разделе 8.4.

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

## **8.2. Структурное программирование.**

При программировании модуля следует иметь в виду, что программа должна быть понятной не только компьютеру, но и человеку: и разработчик модуля, и лица, проверяющие модуль, и тестовики, готовящие тесты для отладки модуля, и сопроводители ПС, осуществляющие требуемые изменения модуля, вынуждены будут многократно разбирать логику работы модуля. В современных

языках программирования достаточно средств, чтобы запутать эту логику сколь угодно сильно, тем самым, сделать модуль трудно понимаемым для человека и, как следствие этого, сделать его ненадежным или трудно сопровождаемым. Поэтому необходимо принимать меры для выбора подходящих языковых средств и следовать определенной дисциплине программирования. В связи с этим Дейкстра [8.2] и предложил строить программу как композицию из нескольких типов управляющих конструкций (структур), которые позволяют сильно повысить понимаемость логики работы программы. Программирование с использованием только таких конструкций назвали *структурным*.

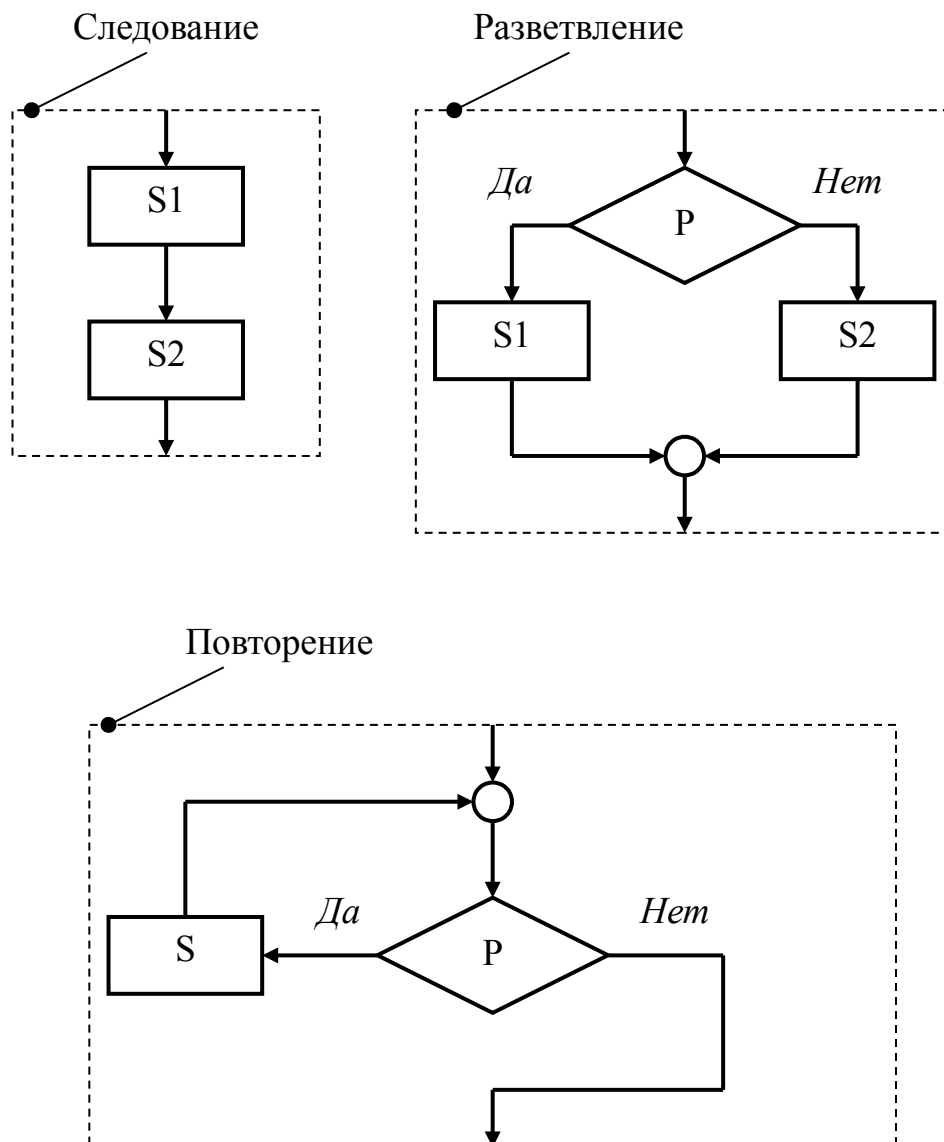


Рис. 8.1. Основные управляющие конструкции структурного программирования.

Основными конструкциями структурного программирования являются: следование, разветвление и повторение (см. Рис. 8.1). Компонентами этих конструкций являются обобщенные операторы (узлы обработки [8.5]) S, S1, S2 и условие (предикат) P. В качестве обобщенного оператора может быть либо простой оператор используемого языка программирования (операторы присваивания, ввода, вывода, обращения к процедуре), либо фрагмент программы, являющийся композицией основных управляющих конструкций структурного программирования. Существенно, что каждая из этих конструкций имеет по управлению только один вход и один выход. Тем самым, и обобщенный оператор имеет только один вход и один выход.

Весьма важно также, что эти конструкции являются уже математическими объектами (что, по существу, и объясняет причину успеха структурного программирования). Доказано, что для каждой неструктурированной программы можно построить функционально эквивалентную (т.е. решающую ту же задачу) структурированную программу. Для структурированных программ можно математически доказывать некоторые свойства, что позволяет обнаруживать в программе некоторые ошибки. Этому вопросу будет посвящена отдельная лекция.

Структурное программирование иногда называют еще "программированием без GO TO". Однако дело здесь не в операторе GO TO, а в его беспорядочном использовании. Очень часто при воплощении структурного программирования на некоторых языках программирования (например, на ФОРТРАНе) оператор перехода (GO TO) используется для реализации структурных конструкций, что не нарушает принципов структурного программирования. Запутывают программу как раз "неструктурные" операторы перехода, особенно переход к оператору, расположенному в тексте модуля выше (раньше) выполняемого оператора перехода. Тем не менее, попытка избежать оператора перехода в некоторых простых случаях может привести к слишком громоздким структурированным программам, что не улучшает их ясность и содержит опасность появления в тексте модуля дополнительных ошибок. Поэтому можно рекомендовать избегать употребления оператора перехода всюду, где это возможно, но не ценой ясности программы [8.1].

К полезным случаям использования оператора перехода можно отнести выход из цикла или процедуры по особому условию, "досрочно" прекращающего работу данного цикла или данной процедуры, т.е. завершающего работу некоторой структурной единицы (обобщенного оператора) и тем самым лишь локально нарушающего структурированность программы. Большие трудности

(и усложнение структуры) вызывает структурная реализация реакции на возникающие исключительные (часто ошибочные) ситуации, так как при этом требуется не только осуществить досрочный выход из структурной единицы, но и произвести необходимую обработку (исключение) этой ситуации (например, выдачу подходящей диагностической информации). Обработчик исключительной ситуации может находиться на любом уровне структуры программы, а обращение к нему может производиться с разных нижних уровней. Вполне приемлемой с технологической точки зрения является следующая "неструктурная" реализация реакции на исключительные ситуации [8.7]. Обработчики исключительных ситуаций помещаются в конце той или иной структурной единицы и каждый такой обработчик программируется таким образом, что после окончания своей работы производит выход из той структурной единицы, в конце которой он помещен. Обращение к такому обработчику производится оператором перехода из данной структурной единицы (включая любую вложенную в нее структурную единицу).

### **8.3. Пошаговая детализация и понятие о псевдокоде.**

Структурное программирование дает рекомендации о том, каким должен быть текст модуля. Возникает вопрос, как должен действовать программист, чтобы построить такой текст. Часто программирование модуля начинают с построения его блок-схемы, описывающей в общих чертах логику его работы. Однако современная технология программирования не рекомендует этого делать без подходящей компьютерной поддержки. Хотя блок-схемы позволяют весьма наглядно представить логику работы модуля, при их ручном кодировании на языке программирования возникает весьма специфический источник ошибок: отображение существенно двумерных структур, какими являются блок-схемы, на линейный текст, представляющий модуль, содержит опасность искажения логики работы модуля, тем более, что психологически довольно трудно сохранить высокий уровень внимания при повторном ее рассмотрении. Исключением может быть случай, когда для построения блок-схем используется графический редактор и они формализованы настолько, что по ним автоматически генерируется текст на языке программирования (как, например, это делается в Р-технологии [8.6]).

В качестве основного метода построения текста модуля современная технология программирования рекомендует *пошаговую детализацию* [8.1, 8.3, 8.5]. Сущность этого метода заключается в

разбиении процесса разработки текста модуля на ряд шагов. На первом

шаге описывается общая схема работы модуля в обозримой линейной текстовой форме (т.е. с использованием очень крупных понятий), причем это описание не является полностью формализованным и ориентировано на восприятие его человеком. На каждом следующем шаге производится уточнение и детализация одного из понятий (будем называть его *уточняемым*), в каком либо описании, разработанном на одном из предыдущих шагов. В результате такого шага создается описание выбранного уточняемого понятия либо в терминах базового языка программирования (т.е. выбранного для представления модуля), либо в такой же форме, что и на первом шаге с использованием новых уточняемых понятий. Этот процесс завершается, когда все уточняемые понятия будут *уточнениями* (т.е. в конечном счете будут выражены на базовом языке программирования). Последним шагом является получение текста модуля на базовом языке программирования путем замены всех вхождений уточняемых понятий заданными их описаниями и выражение всех вхождений конструкций структурного программирования средствами этого языка программирования.

Пошаговая детализация связана с использованием частично формализованного языка для представления указанных описаний, который получил название *псевдокода* [8.5, 8.8]. Этот язык позволяет использовать все конструкции структурного программирования, которые оформляются формализованно, вместе с неформальными фрагментами на естественном языке для представления обобщенных операторов и условий. В качестве обобщенных операторов и условий могут задаваться и соответствующие фрагменты на базовом языке программирования.

Головным описанием на псевдокоде можно считать внешнее оформление модуля на базовом языке программирования, которое должно содержать:

- начало модуля на базовом языке, т.е. первое предложение или заголовок (спецификацию) этого модуля [8.1];
- раздел (совокупность) описаний на базовом языке, причем вместо описаний процедур и функций - только их внешнее оформление;
- неформальное обозначение последовательности операторов тела модуля как одного обобщенного оператора (см. ниже), а также неформальное обозначение тела каждого описания процедуры или функции как одного обобщенного оператора;
- последнее предложение (конец) модуля на базовом языке [8.1].

Внешнее оформление описания процедуры или функции представляется аналогично. Впрочем, если следовать Дейкстре [8.2], раздел описаний лучше также представить здесь неформальным обозначением, произведя его детализацию в виде отдельного описания.

Неформальное обозначение обобщенного оператора на псевдокоде производится на естественном языке произвольным предложением, раскрывающим в общих чертах его содержание. Единственным формальным требованием к оформлению такого обозначения является следующее: это предложение должно занимать целиком одно или несколько графических (печатных) строк и завершаться точкой (или каким-либо другим знаком, специально выделенным для этого).

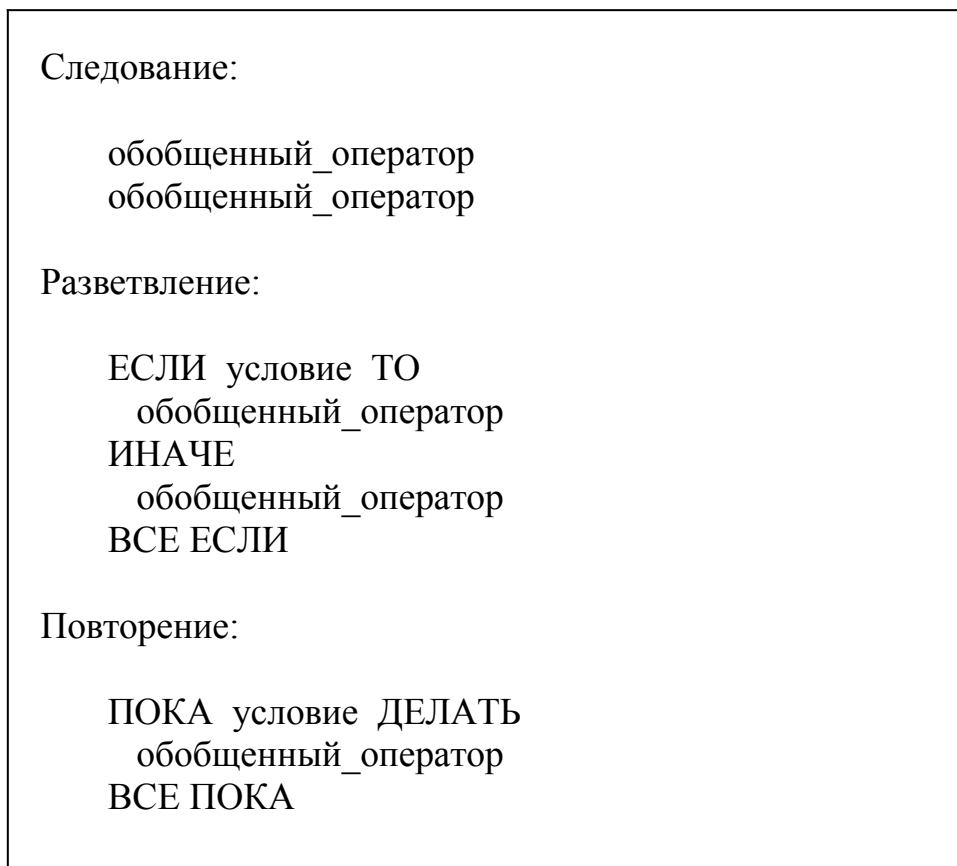


Рис. 8.2. Основные конструкции структурного программирования на псевдокоде.

Для каждого неформального обобщенного оператора должно быть создано отдельное описание, выражающее логику его работы (детализирующее его содержание) с помощью композиции основных конструкций структурного программирования и других обобщенных операторов. В качестве заголовка такого описания должно быть

неформальное обозначение детализируемого обобщенного оператора. Основные конструкции структурного программирования могут быть представлены в следующем виде (см. рис. 8.2). Здесь условие может быть либо явно задано на базовом языке программирования в качестве булевского выражения, либо неформально представлено на естественном языке некоторым фрагментом, раскрывающим в общих чертах смысл этого условия. В последнем случае должно быть создано отдельное описание, детализирующее это условие, с указанием в качестве заголовка обозначения этого условия (фрагмента на естественном языке).

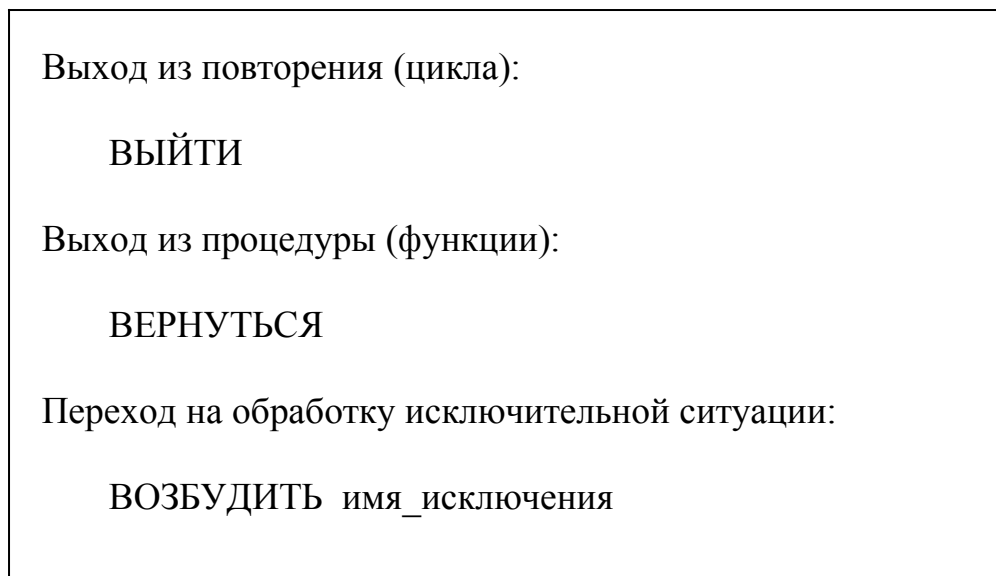


Рис. 8.3. Частные случаи оператора перехода в качестве обобщенного оператора.

В качестве обобщенного оператора на псевдокоде можно использовать указанные выше частные случаи оператора перехода (см. рис. 8.3). Последовательность обработчиков исключительных ситуаций (исключений) задается в конце модуля или описания процедуры (функции). Каждый такой обработчик имеет вид:

```

ИСКЛЮЧЕНИЕ имя_исключения
    обобщенный_оператор
ВСЕ ИСКЛЮЧЕНИЕ
  
```

Отличие обработчика исключительной ситуации от процедуры без параметров заключается в следующем: после выполнения процедуры управление возвращается к оператору, следующему за



обращением к ней, а после выполнения исключения управление возвращается к оператору, следующему за обращением к модулю или процедуре (функции), в конце которого (которой) помещено данное исключение.

Рекомендуется на каждом шаге детализации создавать достаточно содержательное описание, но легко обозримое (наглядное), так чтобы оно размещалось на одной странице текста. Как правило, это означает, что такое описание должно быть композицией пяти-шести конструкций структурного программирования. Рекомендуется также вложенные конструкции располагать со смещением вправо на несколько позиций (см. рис. 8.4). В результате можно получить описание логики работы по наглядности вполне конкурентное с блок-схемами, но обладающее существенным преимуществом - сохраняется линейность описания.

```

УДАЛЕНИЕ В ФАЙЛЕ ЗАПИСЕЙ ДО ПЕРВОЙ,
УДОВЛЕТВОРЯЮЩЕЙ ЗАДАННОМУ ФИЛЬТРУ:

    УСТАНОВИТЬ НАЧАЛО ФАЙЛА.
    ПОКА НЕ КОНЕЦ ФАЙЛА ДЕЛАТЬ
        ПРОЧИТАТЬ ОЧЕРЕДНУЮ ЗАПИСЬ.
        ЕСЛИ ОЧЕРЕДНАЯ ЗАПИСЬ УДОВЛЕТВОРЯЕТ
            ФИЛЬТРУ 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

```

Рис. 8.4. Пример одного шага детализации на псевдокоде.

Идею пошаговой детализации приписывают иногда Дейкстре [8.1]. Однако Дейкстра предлагал принципиально отличающийся метод построения текста модуля [8.2], который нам представляется более глубоким и перспективным. Во-первых, вместе с уточнением

операторов он предлагал постепенно (по шагам) уточнять (детализировать) и используемые структуры данных. Во-вторых, на каждом шаге он предлагал создавать некоторую виртуальную машину для детализации и в ее терминах производить детализацию всех уточняемых понятий, для которых эта машина позволяет это сделать. Таким образом, Дейкстра предлагал, по существу, детализировать по горизонтальным слоям, что является перенесением его идеи о слоистых системах (см. лекцию 6) на уровень разработки модуля. Такой метод разработки модуля поддерживается в настоящее время пакетами языка АДА [8.7] и средствами объектно-ориентированного программирования [8.9].

#### **8.4. Контроль программного модуля.**

Применяются следующие методы контроля программного модуля:

- статическая проверка текста модуля;
- сквозное прослеживание;
- доказательство свойств программного модуля.

При статической проверке текста модуля этот текст просматривается с начала до конца с целью найти ошибки в модуле. Обычно для такой проверки привлекают, кроме разработчика модуля, еще одного или даже нескольких программистов. Рекомендуются ошибки, обнаруживаемые при такой проверке исправлять не сразу, а по завершению чтения текста модуля.

Сквозное прослеживание представляет собой один из видов динамического контроля модуля. В нем также участвуют несколько программистов, которые вручную прокручивают выполнение модуля (оператор за оператором в той последовательности, которая вытекает из логики работы модуля) на некотором наборе тестов.

Доказательству свойств программ посвящена следующая лекция. Здесь следует лишь отметить, что этот метод применяется пока очень редко.

#### **Упражнения к лекции 8.**

- 8.1. *Что такое структурное программирование?*
- 8.2. *Что такое пошаговая детализация программного модуля?*
- 8.3. *Что такое псевдокод?*

#### **Литература к лекции 8.**

- 8.1. Г.Майерс. Надежность программного обеспечения. - М.: Мир, 1980. - С. 127-154.

- 8.2. Э.Дейкстра. Заметки по структурному программированию / У.Дал, Э.Дейкстра, К.Хоор. Структурное программирование. - М.: Мир, 1975. - С. 24-97.
- 8.3. Н.Вирт. Систематическое программирование. - М.: Мир, 1977. - С. 94-164.