## Структурное программирование

Подготовленные для программирования материалы – алгоритмы и укрупненные схемы программ, получает руководитель группы программистов. Он распределяет задания между программистами, руководствуясь установленными сроками работы над созданием АСУ, параметрами, характеризующими сложность и трудоемкость разрабатываемых программ, квалификацией программистов и другими факторами. Руководствуясь нормами затрат времени на программирование и необходимого машинного времени ЭВМ, составляют график выполнения работ по этапам. В дальнейшем руководитель группы контролирует соответствие фактических затрат времени предусмотренным в графике, в необходимых случаях консультирует программистов и координирует их работу, участвует в системной отладке.

Временные типовые нормы времени на программирование задач для ЭВМ рассчитаны на комплексы задач или задачи и указаны в человеко-днях при пятидневной рабочей неделе продолжительностью 41 ч в неделю.

Нормы времени определены по факторам, в наибольшей степени влияющим на трудоемкость программирования. К этим факторам относятся: количество наборов данных (макетов) входной информации, включая текущие данные, нормативно-справочную и извлекаемую из банков данных информацию; число типов форм выходной информации; степень новизны комплекса задач; сложность алгоритма; вид используемой информации; сложность контроля входной и выходной информации; язык программирования; степень использования типовых модулей и пакетов прикладных программ.

Предусмотрены четыре степени новизны комплексов задач: А – задачи, предусматривающие применение принципиально новых методов разработки; Б – разработка типовых проектных решений, оригинальных задач; В – привязка типовых проектных решений, требующая их изменения, разработка задач, имеющих аналоги; Г – использование типовых проектных решений, разработка задач, имеющих аналогичные решения.

По сложности алгоритмы разделены на три группы: оптимизации и моделирования систем и объектов; учета, отчетности, статистики и информационного поиска; стандартных методов решения задач, не требующих применения сложных численных и логических методов.

По виду используемой информации определены три группы: переменной или текущей информации; нормативно-справочной информации; информационной базы.

Сложность контроля информации определена четырьмя группами: перекрестного контроля с учетом связей между группами входных данных и документов; формального контроля однотипных входных данных и документов; контроля при выводе данных сложной многоуровневой структуры, разнообразной формы и содержания; вывода данных на машинные носители, печати документов однообразной формы и содержания.

Нормы времени определены для комплексов задач, имеющих степень новизны В и использующих текущую информацию. При задачах с другими характеристиками используют для пересчета поправочные коэффициенты, в частности для задач группы А–2–5; группы Г–0,4–0,8.

Предполагается использование языков верхнего уровня типа ПЛ/1: для языков низкого уровня применяется коэффициент 1,15. При использовании типовых модулей или пакетов прикладных программ применяется коэффициент 0,6-0,8, а при разработке стандартного математического обеспечения– 1,2–1,6.

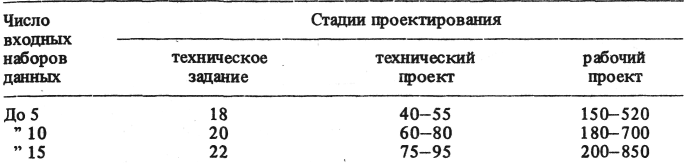
Трудоемкость технорабочего проекта определяется как сумма 85% трудоемкости технического и 100% трудоемкости рабочего проектов.

Общая трудоемкость программирования определяется как сумма трудоемкостей отдельных этапов работы.

Нормы времени учитывают работы, сопутствующие программированию, такие, как согласование работ внутри группы, проверка и приемка работ, внесение исправлений и дополнений по указанию руководителя и т.п., а также время на обслуживание рабочего места, отдых и личные надобности в размере 10% от оперативного времени. Если в организации более прогрессивные нормы времени, их не следует изменять на основании приводимых ниже.

Нормы времени в человеко-днях на различных стадиях проектирования приведены в табл. 1.

Таблица 1

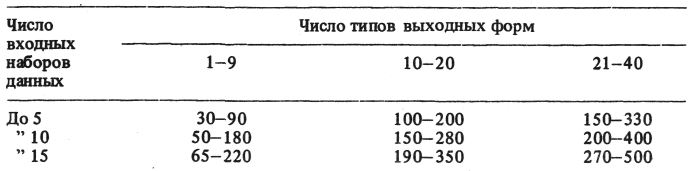


Нормы затрат времени работы ЭВМ определяются в зависимости от тех же приведенных выше основных факторов, которые влияют в наибольшей степени и на трудоемкость программирования, с той разницей, что дополнительно учитывается быстродействие ЭВМ.

Приведенные в табл. 1 для ориентации нормы времени в часах определены для комплексов задач группы В, стандартных методов решения и языков программирования высокого уровня.

Время работы ЭВМ на стадии ввода в эксплуатацию составляет от 10 до 25% времени, указанного в табл. 2. Если внедрение осуществляется заказчиком, затраты машинного времени на отладку должны быть снижены на 10–25% по усмотрению руководителя работ.

Таблица 2



Эффективным методом повышения качества комплексов программ является использование при их подготовке принципов структурного программирования. При организации проектирования программ и процесса кодирования это позволяет предотвратить большинство логических ошибок и обнаружить те, которые допущены. Структурное программирование включает три составляющие: проектирование сверху – вниз; модульное программирование; структурное копирование.

Проектирование сверху – вниз. По существу это иерархический подход к решению поставленной задачи. Метод предусматривает на первом этапе определение задачи в общих чертах и последовательное уточнение ее структуры путем детализации основных функций. На каждом шаге детализации выясняются основные функции, т.е. задача разбивается на ряд подзадач, пока эти подзадачи не станут настолько простыми, что каждая из них может быть представлена несложным программным модулем, действие которого описывается одной фразой.

Метод предусматривает описание данных, их структуры и основных процессов обработки. Описание данных должно включать тщательно отобранные примеры, демонстрирующие выполнение основных функций системы и их наиболее существенные варианты. При описании модуля должны быть описаны тестовые данные.

При проектировании сверху - вниз создание проекта системы по уровням, начиная с верхнего, облегчает реализацию поставленных целей и выявление ошибок на ранних стадиях создания программного обеспечения.

Модульное программирование. В его основе лежит разделение программы на логические части (модули) и последовательное программирование каждой части. Если проведено проектирование всей задачи сверху – вниз, то она разбивается на подзадачи, соответствующие возможным модулям. Разбиение задачи на модули должно обеспечивать независимость реализации каждого программного модуля. Последующее формирование комплекса программ не должно требовать знаний о внутренней работе модуля, т.е. необходимо обеспечить простоту сопряжений программных модулей комплекса.

При разбиении задачи на модули следует учитывать их размер. Обычно размер модуля составляет около 60 строк. Такая длина удобна для восприятия, так как она легко охватывается и запоминается. Это число строк помещается на одной странице, его легко прочесть на терминале вычислительной машины. Однако целесообразно выбирать размеры модуля так, чтобы он соответствовал одной подзадаче.

Следует стремиться к независимости программных модулей. Для этого необходимо, чтобы модуль не зависел от источника входных данных, места назначения выходных данных и от предыстории. Каждый модуль должен иметь свое назначение, отличающееся от назначения других модулей. Это должен быть замкнутый блок, вход и выход которого точно определены. Функциональная независимость модулей сокращает вероятность воздействия изменений в одной подпрограмме на остальные подпрограммы комплекса.

Воздействие изменений в одном модуле на другие модули называют волновым эффектом. Этот эффект можно уменьшить, сведя к минимуму связь между модулями, т.е. сократив количество путей, вдоль которых изменения или ошибки влияют на другие части программы. Простейший путь уменьшения волнового эффекта состоит в сокращении использования глобальных переменных и в сокращении размеров модуля. Минимизация взаимосвязей между модулями проводится обычно за счет усиления связей между элементами одного модуля (модульной прочности). Высокая степень независимости, характерная для структурного программирования, и минимальная связь модулей приводят к ограничению влияния ошибок в пределах отдельных модулей и к уменьшению как функциональной сложности модуля, так и сложности связей.

Формализация и оптимизация процесса разбиения системы программного обеспечения на модули с учетом приведенных факторов позволяет существенно повысить качество создаваемых комплексов программ и сократить вероятность появления ошибок на дальнейших этапах разработки.

Рассмотрим постановку и решение задачи разбиения сложной системы программного обеспечения на функциональные модули, имеющие минимальное число информационных связей, при ограничении на общее число выделяемых модулей, на число процедур и информационных элементов в каждом модуле, на сложности связей между отдельными модулями. Исходными данными для задачи являются: множество различных типов входных, промежуточных и выходных переменных информационных элементов; множество необходимых процедур обработки (преобразования) информационных элементов; взаимосвязи процедур обработки информационных элементов.

Отображение множества процедур на множество информационных элементов удобно представить в виде двудольного графа, дуги которого соединяют процедуры с соответствующими информационными элементами, причем дуги, инцидентные различным информационным элементам, могут быть помечены различными цветами (рис. 1).

Задача разбиения программного обеспечения на модули может быть сформулирована как задача разбиения заданного множества процедур обработки данных на непересекающиеся подмножества, имеющие минимальное число попарно общих информационных элементов. Наиболее наглядно такая постановка иллюстрируется с использованием графа, вершинами которого являются процедуры, а связывающие их дуги различного цвета соответствуют имеющимся информационным элементам. В этом случае в графовой интерпретации данная задача формулируется как задача разбиения раскрашенного графа на подграфы (модули), обеспечивающие минимум суммарного числа дуг различного цвета, связывающих выделенные подграфы, при ограничении на общее число выделяемых подграфов, число вершин и дуг в каждом подграфе, число связей между ними.

Для удобства постановки, описания методов и результатов решения задачи сведем информацию о связи процедур и данных (элементов записей), содержащуюся в графовой модели, в специальную табличную форму. По строкам таблицы расположим выполняемые системой функции или процедуры (запись, сортировка, контроль, обновление, выборка, печать и т.д.), а по столбцам - элементы множества идентификаторов исходных, промежуточных и выходных данных, связанных с выполнением этих процедур. На пересечении столбца и строки таблицы записывается "1" – признак зависимости или связи рассматриваемой функции и информационного элемента, если таковая существует.

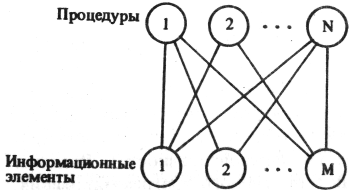


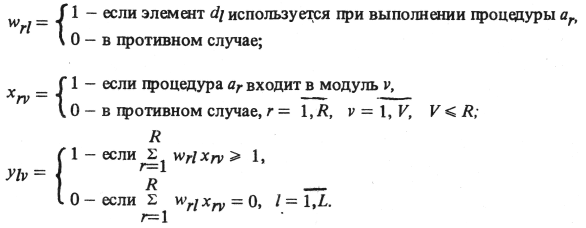
Рис. 1

В этом случае критерий оптимального разбиения на модули информационного и программного обеспечения, содержание которого описано в табличной форме, формулируется следующим образом. Найти разбиение множества функций или процедур таблицы на непересекающиеся подмножества, минимизирующее суммарное число информационных связей между модулями при ограничениях: на число выделяемых модулей; на число информационных элементов, входящих в один модуль, определяющее сложность разработки блок-схемы подсистемы; на число процедур каждого модуля, определяющее, в основном, сложность кодирования блок-схемы; на число информационных связей между выделяемыми модулями; на совместимость отдельных процедур.

Введем необходимые обозначения:

D = {d1, d2, ..., dl, ..., di}

-множество информационных элементов системы; - матрица взаимосвязи процедур и информационных элементов, где



Суммарное число S различн. информационных элементов, являющихся попарно общими для различных модулей, равно



При разработке программного обеспечения АСУ требуется минимизировать S при следующих основных технологических ограничениях: на число процедур в модулях



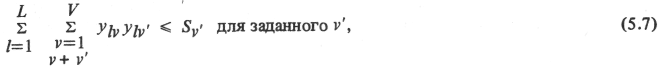
где Мv - мощность множества процедур в v-м модуле;

на число информационных элементов, используемых модулями,



где Lv - максимальное число информационных элементов, используемых v-м модулем;

на интерфейс между отдельным модулем и другими модулями системы



где Sv' - максимально возможное число информационных связей между модулем v' и остальными модулями системы;

на интерфейс между двумя заданными модулями системы



где Svv' - максимально возможное число информационных связей между модулями v и v';

на разделение отдельных процедур системы по различным модулям

xry + xr'v ≤ 1,

где v = 1, 2, ..., V для заданных r и r'. Поставленная задача является задачей квадратичного целочисленного программирования и может быть сведена к линейной форме путем линеаризации выражений и введением связи между вспомогательными и основными переменными. В этом случае она может быть решена с использованием имеющихся пакетов прикладных программ.

В условиях, когда задано информационное обеспечение, т.е. определено содержание каждого информационного массива и его динамические характеристики, решаются динамические задачи разбиения системы программного обеспечения на модули. В качестве критерия оптимизации используются минимум общего времени обмена с внешней памятью и минимум межмодульного интерфейса.

Исходными данными для постановки и решения динамических задач синтеза системы модулей являются множество информационных массивов системы, для которых определены: входные, выходные и промежуточные данные; множество альтернативных процедур обработки данных, входящих в состав алгоритмов, обеспечивающих решение задач системы; возможные последовательности выполнения процедур в процессе обработки; способы обмена с внешней памятью модульной системы обработки данных; взаимосвязи процедур с информационными массивами; временные характеристики обращения к массивам.

Задачи синтеза систем модулей программного обеспечения СОД сведены к задачам нелинейного целочисленного программирования. Для их решения предложены алгоритмы, использующие схему "ветвей и границ" и особенности функционирования модульных систем обработки данных.

Структурное кодирование. Третьим составным элементом структурного программирования является структурное кодирование, которое представляет собой метод написания хорошо структурированных программных модулей, удобных для тестирования, модификации и использования. Метод предусматривает написание программных модулей произвольного размера и сложности на основе ограниченного множества базисных логических структур. Метод аналогичен принципу, положенному в основу проектирования схем, где любая логическая структура может быть создана из элементарных структур И, ИЛИ и НЕ. Структурное кодирование базируется на строго доказанной теореме о структурировании, которая утверждает, что любую правильную программу (с одним входом и одним выходом, без зацикливаний и недостижимых команд) можно написать с использованием следующих логических структур: последовательности двух или более операторов; выбора одного из двух операторов (IF THEN, ELSE); повторения (или управления циклом) оператора, пока выполняется некоторое условие (DO WHILE).

На рис. 2 показаны основные логические структуры: последовательности (рис. 2, а); выбора (рис. 2, б); повторения (рис. 2, в). Каждая структура имеет один вход и один выход. Комбинации правильных программ, полученные с использованием трех основных логических структур, также являются правильными программами. При использовании только указанных структур отпадает необходимость в безусловных переходах и метках. Применение структурного кодирования в значительной степени уменьшает сложность программ. Существенным моментом является то, что при замене любого функционального блока в схеме правильной программы на любую из трех основных логических структур программа остается в классе правильных программ. При структурном кодировании структура последовательности формализует тот факт, что операторы программы выполняются в порядке их появления, структура выбора - выбор одного из двух действий исходя из выполнения некоторого условия (оператор IF THEN, ELSE), структура повторения – выполнение группы операторов до тех пор, пока выполняется некоторое условие (оператор DO WHILE или итеративный DO).

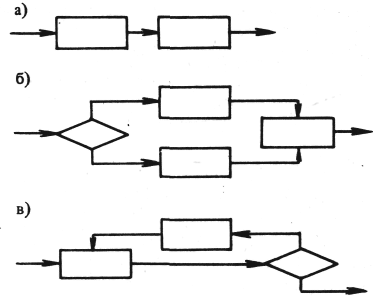


Рис. 2

Хотя теоретически возможно написать все правильные программы, используя только перечисленные структуры, можно расширить это ограничение добавлением структур, имеющих один вход и один выход и формализующих операторы DO UNTIL и CASE.

Таким образом, чтобы получить структурированную программу, следует избегать операторов GO TO. Каждый программный модуль должен состоять из последовательности операторов IF, THEN, ELSE, циклов или операторов CASE. В случае, если язык не имеет достаточного количества языковых конструкций для структурного программирования (ФОРТРАН, КОБОЛ), рекомендуется применять операторы GO TO только для переходов вперед и там, где возможно заменять их операторами CALL или PERFORM. В языках ПЛ/1 и АЛГОЛ имеется большая часть рекомендованных конструкций, что позволяет легко писать структурированные программы.

Применение принципов структурного программирования позволяет существенно уменьшить сложность проектируемых комплексов программ и облегчить процесс их отладки за счет более детального проектирования. Основным преимуществом структурного программирования является то, что программа может быть проанализирована путем проверки структуры и становится возможным обнаруживать ошибки уже на этапе программирования, что значительно снижает затраты на процесс отладки.