Программное обеспечение и технология программирования

Системы программирования
Структурное программирование
Чтение структурированных программ
Стратегии решения задач

Повтор: этапы технологии создания прикладных программ

1. Постановка задачи (определяем цель),
2. Математическое описание задачи и выбор метода ее решения..
3. Алгоритмизация решения задачи.
4. Составление программы решения задачи.
5. Тестирование,
6. Отладка.

Типы ошибок:

- *синтаксические* - некорректная запись отдельных конструкций языка программирования.
- *Логические* – ошибки в логике работы программы на исходных данных.

Системы и языки программирования

Системы программирования предназначены для совершенствования процесса разработки и отладки программ.

Включают в себя:

- входной язык системы программирования (исходный язык);
- транслятор, обеспечивающий перевод программы с входного языка системы на внутренний (машинный) язык;
- библиотеку стандартных программ, подключаемую в процессе подготовки программы к выполнению;
- документацию.

Классификация языков программирования

Языки программирования делятся на *декларативные* и *операторные*.

Декларативный язык программирования - язык программирования высокого уровня, построенный на описании данных и на описании искомого результата.

Декларативные языки подразделяются на функциональные и логические языки.

- Логический язык программирования - язык программирования, позволяющий выполнить описание проблемы в терминах фактов и логических формул, а собственно решение проблемы выполняет система с помощью механизмов логического вывода.
- Функциональный язык программирования - язык программирования, позволяющий задавать программу в виде совокупности определений функций.

В функциональных языках программирования:

- функции обмениваются между собой данными без использования промежуточных переменных и присваиваний;
- переменные, однажды получив значение, никогда его не изменят;
- циклы заменяются аппаратом рекурсивных функций.

К *декларативным* языкам относятся языки Пролог, Кобол, Лисп.

Пролог – язык логического программирования, предназначен для разработки интеллектуальных систем.

Лисп – язык для обработки списков, также применяется для проектирования интеллектуальных систем.

Операторные 👉👉👉👉👉👉👉👉👉

Операторные языки

Операторный язык – это такой способ кодирования алгоритма, в результате которого получается понятная для компьютера запись алгоритма - программа. Этот способ требует, как минимум, знания правил записи выражений средствами того или иного языка программирования.

Операторные языки разделяются на непроцедурные и процедурные.

Непроцедурные языки приближаются к естественным языкам и ориентированы на пользователей, не являющихся программистами. К ним относятся:

QBE (программирование на примерах),

Форт (применяется при решении сложных задач имитационного моделирования, в системах искусственного интеллекта в графических системах).

Процедурные 👉 👉 👉 👉 👉 👉 👉 👉 👉 👉

Процедурные языки

Процедурный язык программирования - Язык программирования, синтаксис которого ориентирован на разработку программ в соответствии с технологией структурного программирования. (Pascal, Algol, Си).

С помощью *процедурных* языков программист определяет, что необходимо получить в результате решения задачи и как этого можно достичь (пишет программу).

Они делятся на *машинно-зависимые* языки и *машинно-независимые* языки.

- *Машинно-зависимые* языки это машинные языки и машинно-ориентированные языки. *Машинные* языки представляют собой набор команд конкретного процессора ЭВМ. *Машинно-ориентированные* языки – это автокоды, языки символического кодирования и ассемблеры. Все эти языки учитывают специфику организации и принципы работы конкретной ЭВМ.
- *Машинно-незавимые* языки (языки высокого уровня) делятся на процедурно-ориентированные (универсальные), проблемно-ориентированные и объектно-ориентированные.

Машинно-незавимые языки

- *Процедурно-ориентированный язык* - язык программирования высокого уровня, в основу которого положен принцип описания (последовательности) действий, позволяющей решить поставленную задачу. Обычно процедурно-ориентированные языки задают программы как совокупности процедур или подпрограмм.

  Процедурно-ориентированные эффективны для описания алгоритмов решения широкого класса задач. Наиболее известны Фортран, Кобол, ПЛ/1, Паскаль, Ада, Бейсик, Си.
- *Проблемно-ориентированные* языки предназначены для описания процессов обработки информации в более узкой, специфической области:

  РПГ – язык для генерации отчетов, АПЛ – язык для статистической обработки массивов, GPSS – язык моделирования.
- *Объектно-ориентированный язык* - язык программирования, поддерживающий понятие объектов, их свойств и методов обработки, язык программирования, поддерживающий наследование и полиморфизм, язык, ориентированный на разработку программных приложений для широкого круга разнообразных задач, имеющих общность в реализуемых компонентах.

  К ним относятся Ада 93, Java, Object Pascal, Visual Basic, Си++.

Языки программирования высокого уровня

Декларативные

Операторные

Функциональные

Логические

Непроцедурные

Процедурные

Пролог,Кобол, Лисп

Машинно-зависимые

Машинно-независимые

Процедурно-ориентированный

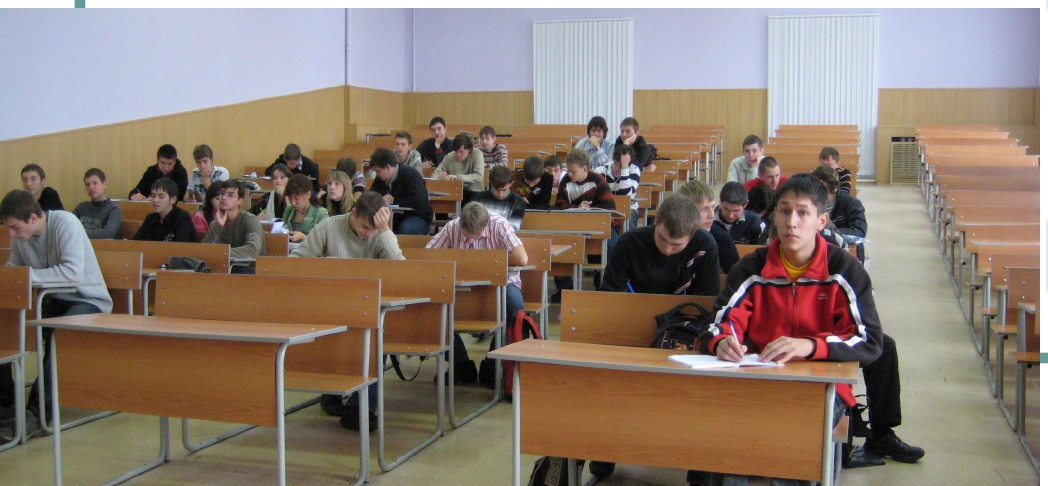
Фортран, Кобол, ПЛ/1, Паскаль, Ада, Бейсик, Си

Проблемно-ориентированный

РПГ, АПЛ, GPSS

Объктно-ориентированный

Ада93, Java, Object Pascal, Visual Basic, Cr++



Трансляторы

Для перевода исходного текста программы на машинный язык используются *трансляторы*. Существует два типа трансляторов: интерпретаторы и компиляторы.

Интерпретация подразумевает пооперационную трансляцию и последующее выполнение оттранслированного оператора исходной программы. Поддерживают диалоговый режим, который удобен в процессе отладки программы.

При *компиляции* сначала исходная программа полностью переводится на машинный язык – в результате получается *модуль объектного кода*. Затем этот модуль обрабатывается загрузчиком (редактором связей) для получения загрузочного (выполняемого) модуля. Загрузочный модуль запускается на выполнение.

Структурное программирование

Структурное программирование – методология разработки программного обеспечения, предложенная в 70-х года XX века Дейкстрой и разработанная и дополненная Виртом.

Методология структурного программирования появилась как следствие возрастания сложности решаемых на компьютерах задач и соответственного усложнения программного обеспечения, поэтому потребовалась какая-то *систематизация* процесса разработки и структуры программ.

!!! В соответствии с данной методологией любая программа представляет собой структуру, построенную из трёх типов базовых конструкций:

- последовательное исполнение – однократное выполнение операций в том порядке записи;
- ветвление – однократное выполнение одной из двух или более операций, в зависимости от условия;
- цикл – многократное исполнение одной и той же операции до тех пор, пока выполняется некоторое заданное условие.

Базовые конструкции могут быть вложены друг в друга произвольным образом, но никаких других средств управления последовательностью выполнения операций не предусматривается.

Повторяющиеся фрагменты программы (либо не повторяющиеся, но представляющие собой логически целостные вычислительные блоки) могут оформляться подпрограмм (процедур или функций). В текст основной программы вместо помещённого в подпрограмму фрагмента вставляется инструкция вызова подпрограммы. При выполнении такой инструкции выполняется вызванная подпрограмма, после чего исполнение программы продолжается со следующей за командой вызова подпрограммы инструкции.

Разработка программы

Основным принципом технологии структурного программирования является нисходящее программирование - это программирование с использованием подпрограмм, которое позволяет вести разработку приложения пошагово «сверху вниз».

Сначала пишется текст основной программы, в котором вместо каждого связного логического фрагмента текста вставляется вызов подпрограммы, которая будет выполнять этот фрагмент.

Вместо настоящих, работающих подпрограмм, в программу вставляются "заглушки", которые ничего не делают.

Полученная программа проверяется и отлаживается. После того, как программист убедится, что подпрограммы вызываются в правильной последовательности (то есть общая структура программы верна), подпрограммы-"заглушки" последовательно заменяются на реально работающие, причём разработка каждой подпрограммы ведётся тем же методом, что и основной программы.

Разработка заканчивается тогда, когда не останется ни одной "заглушки ", которая не была бы удалена. Такая последовательность гарантирует, что на каждом этапе разработки программист одновременно имеет дело с обозримым и понятным ему множеством фрагментов и может быть уверен, что общая структура всех более высоких уровней программы верна.

При сопровождении и внесении изменений в программу выясняется, в какие именно процедуры нужно внести изменения, и они вносятся, не затрагивая непосредственно не связанные с ними части программы. Это позволяет гарантировать, что при внесении изменений и исправлении ошибок не выйдет из строя какая-то часть программы, находящаяся в данный момент вне зоны внимания программиста.

Структурированные алгоритмы

Основная идея структурного программирования и состоит в том, что структура программы должна отражать структуру решаемой задачи, чтобы алгоритм программы был ясно виден из исходного текста. Следовательно, надо разбить программу на последовательность модулей, каждый из которых выполняет одно или несколько действий.

Требование к модулю – чтобы его выполнение начиналось с первой команды и заканчивалось последней. *Модульность* – это основная характеристика структурного программирования. А для этого надо иметь средства для создания программы не только с помощью трех простых операторов, но и с помощью средств более точно отражающих конкретную структуру алгоритма.

С этой целью в программирование и введено понятие *подпрограммы* – набора операторов, выполняющих нужное действие и не зависящих от других частей исходного кода. Каждая из подпрограмм выполняет одно из действий исходного кода. Комбинируя блоки, удается сформировать итоговый алгоритм уже не из операторов, а из законченных блоков. Обращаться к блокам надо по названиям, а название несет смысловую нагрузку. Например, Call Summa, означает обращение к подпрограмме с именем Summa, Call - вызов. При структурном подходе к составлению алгоритмов и программ используются три основные типа алгоритмов: условные, циклические алгоритмы и подпрограммы.

Структурированными считаются алгоритмы и программы, составленные с использованием только этих трех типов алгоритмов, при этом для записи циклов и условий должна использоваться *ступенчатая запись*.

Чтение структурированных программ

Подпрограммы бывают двух видов: процедуры и функции. Процедуры просто выполняют последовательность операторов, а функции вычисляют значение и передают его в главную программу.

Подпрограмма – процедура или подпрограмма – функция - это отдельный блок операторов, начинающийся с заголовка и заканчивающийся признаком конца процедуры или функции. Чтобы подпрограмма имела смысл ей надо получить какие-то значения, которые называются параметрами.

Параметры, которые принимаются в подпрограмме, описываются в заголовке и называются *формальными.* Следование принципам структурного программирования сделало тексты программ, даже довольно крупных, нормально читаемыми. Серьёзно облегчилось понимание программ, появилась возможность разработки программ в нормальном промышленном режиме, когда программу может без особых затруднений понять не только её автор, но и другие программисты. Это позволило разрабатывать достаточно крупные программные комплексы силами коллективов разработчиков, и сопровождать эти комплексы в течение многих лет. Структурный подход предполагает отказ от оператора GOTO, т.к. использование произвольных переходов в тексте программы приводит к получению запутанных программ, по тексту которых практически невозможно понять порядок исполнения и взаимозависимость фрагментов.

Структурированная программа должна содержать четкое, удобное для чтения форматирование (систему отступов и правил оформления записей программных конструкций) и необходимое количество комментариев.

Пример

Процедура *Summa*(*a,b*) – это заголовок подпрограммы - процедуры, *имя* которой *Summa*, а в скобках указываются *формальные параметры a* и *b.*

Обращение из главной программы к процедуре осуществляется по имени подпрограммы-процедуры с перечнем в скобках параметров, которые ей передаются, например, *Call Summa*(*x,y*) – означает обратиться к процедуре *Summa* и передать ей параметры *x* и *y,* которые называются *фактическими параметрами*.

Подпрограмма - функция оформляется таким образом: Функция *Длина(a, b, c, d),* где *Длина* – имя функции, а в скобках указаны формальные параметры.

Подпрограмма–функция возвращает только одно значение, которое обязательно присваивается названию функции в теле подпрограммы–функции. Так как функция возвращает значение, то обращение к ней из основной программы может входить в выражение, как операнд.

При выполнении процедуры или функции формальные параметры временно заменяются на фактические.

Стратегии решения задач

Решение задач может быть отнесено к наиболее характерной деятельности человека. Глупо советовать человеку, столкнувшемуся с задачей, спланировать ее решение, если он понятия не имеет, как это делается. Казалось бы, что тут сложного? Нужно только разрабатывать одно за другим возможные решения и затем проверять их. А что если вы не можете придумать ни одного решения? Существует несколько стратегий, которые при правильном использовании могут помочь вам генерировать решения, Несмотря на то что ни одна отдельно взятая стратегия не может гарантировать вам универсальных решений на все случаи жизни, умение применять некоторые стратегии придаст направленность и уверенность вашим действиям при решении новых задач.

Многие математики и ученые при решении стоящих передними задач прибегают к определенным стратегиям и правилам. Многие из них уверены, что если бы студенты приобрели некоторые базовые навыки, они бы решали задачи с большим успехом. Кроме того, исследователи обнаружили, что обучение, направленное на приобретение соответствующих навыков, может повысить способность человека решать возникающие задачи.

Развитие навыков решения задач

Предположим, вы один едете в машине ночью по длинному неосвещенному участку шоссе, по которому лишь изредка проносятся машины, и вдруг слышите знакомое «чап-чап» — звук, издаваемый совершенно спущенной шиной. Вы сворачиваете на обочину и начинаете малоприятную процедуру замены колеса при свете одной лишь луны да редких вспышек фар. Аккуратно отвинтив крепежные гайки, вы кладете их в ящик для инструментов, стоящий рядом с машиной. Но тут мимо проносится какой-то лихач, задевает ящик и все гайки разлетаются по темной дороге за пределы вашей видимости. И вот вы сидите: в руках запасное колесо, другое, спущенное, прислонено к машине, крепежных гаек нет, кругом только ночная тьма и пустынное шоссе. В довершение всего начинает моросить холодный дождь. Как бы вы поступили? Пока незадачливый водитель сидел и обдумывал сложившуюся ситуацию, к решетчатому забору заведения подошли несколько его «старожилов». Один из них предложил водителю следующее решение задачи: снять по одной крепежной гайке с каждого из трех оставшихся колес и использовать их для крепления запасного колеса. Три гайки обеспечат достаточную надежность для того, чтобы добраться до автозаправки. Обрадованный водитель поблагодарил обитателя «психушки», а затем, не удержавшись, спросил: «Как это вы додумались до такого гениального решения задачи?» На что тот ответил: «Я же не дурак, а просто сумасшедший!»

Все согласятся, что водителю был предложен отличный выход из затруднительного положения. Почему же он сам не смог найти решение? Почему оно кажется таким простым и очевидным после того, как уже найдено? Как до такого удачного решения додумался посторонний наблюдатель?

Развитие навыков

Логические рассуждения. Анализ аргументации. Мышление как проверка гипотез. Вероятность и неопределенность. Практическое мышление. Принятие решений. Творческое мышление.

Стратегии решения задач, разработанные для эффективного механического доказательства теорем, в настоящее время связывают с компьютерным моделированием решения задач человеком.

Одинаковые стратегии поиска решений оказываются применимыми как человеком при решении задач, сформулированных на естественных языках, так и машинами при решении задач, сформулированных в символической форме.

Стратегии предполагают два уровня:

1) последовательность действий, которые шаг за шагом ведут к решению рассматриваемой задачи,
2) набор детерминистических объяснений этой последовательности действий.

Разработка общей теории решения задач привела к появлению новой теории – теория поиска вывода.

Теория поиска вывода

Теория поиска вывода - сравнительно молодая логическая теория. Ее основные идеи были сформулированы в 70-х г.г. XX века ленинградским логиком С.Ю.Масловым.

Он определил ее как область математической логики, занимающуюся «выявлением по исчислению и объекту в языке исчисления структуры возможных выводов этого объекта».

Другими словами, теория поиска вывода исследует возможные способы решения задач в различных исчислениях. Поэтому помимо активного практического применения, связанного с машинным поиском вывода, она находит интересные теоретические приложения в философии и психологии творчества.