ТУБ №11:
Принцип программного управления и инструменты программиста

Принцип программного управления

Английский писатель-фантаст и футуролог Артур Кларк однажды сказал: «Любая достаточно развитая технология неотличима от магии».

Глядя на современные компьютеры, действительно можно подумать, что современная техника умеет практически всё и про неё не зря говорят «умная» (и говоря о компьютерах мы сразу же подразумеваем смартфоны, планшеты и многочисленные микроконтроллеры, встроенные во многие окружающие нас вещи). Но только хитрые IT-специалисты понимают, что это совсем не так.

Любое «умное» устройство внутри себя содержит процессор или контроллер, которые подчиняются очень простой логике работы, заложенной в них на аппаратном уровне, просто тем, как они устроены. Отсюда возникает и проблема, над решением которой IT-специалисты бьются ещё с каменного века, — языковой барьер между человеком и машиной.

Современные процессоры работают по тем же принципам, что и их собратья 50 лет назад. Один из ключевых принципов — принцип программного управления.

> Сигналы управления работой отдельных частей компьютера вырабатываются внутри компьютера в процессе вычислений. Источником информации о требуемых типах сигналов на каждом шаге вычислений является код команды, считываемый из памяти компьютера.

Как Вам должно быть известно, память компьютера делится на байты, которые в современных компьютерах обычно состоят из 8 бит каждый. Возьмём, например, среднестатистический ноутбук с процессором Intel или AMD на борту. Команда, уменьшающая значение числа на 1, может быть записана в один байт:

> $0100\ 1xxx_{(2)}$

Значения двоичных разрядов, обозначенных «x», будут зависеть от того, где хранится число, которое нужно увеличить. Если проинтерпретировать значение этого байта как целое число, то оно будет в диапазоне от 72(10) до 79(10) включительно.

Команда, увеличивающая значение числа на 1, также может быть записана в 1 байт:

> $0100\ 0xxx_{(2)}$

Для сравнения рассмотрим то же самое действие для устройства под управлением операционной системы Android. Здесь нет специальной команды для увеличения/уменьшения числа на 1, вместо этого придётся использовать команды сложения. Один из способов записи таких команд может быть таким:

> $1101\ 1000_{(2)}$ xxxx $xxxx_{(2)}$ xxxx $xxxx_{(2)}$ $0000\ 0001_{(2)}$

причём группы по 8 бит «xxxx xxxx» должны совпадать, т.к. задают, откуда берётся исходное значение и куда помещается результат.

Приведённый способ записи команд называется *машинным кодом*, или записью в машинном коде.

Как видим, в разных устройствах используются различные обозначения для одних и тех же операций. При этом одна и та же последовательность байтов для каждого устройства может иметь свой смысл. Конечно же, это создаёт серьёзные проблемы для написания программ сразу в понятном устройству виде.

Здесь можно провести аналогию с естественными языками, где существуют так называемые «ложные друзья переводчика» — схожие по написанию и произношению слова в двух разных языках, которые отличаются по значению. Например, с английского языка слово «gift» переводится как *подарок*, а то же самое слово в немецком языке означает *яд*. По-английски «mist» — это *туман*, а по-немецки — *навоз*. С кириллицей тоже не всё гладко: болгарское слово «булка» на русский язык переводится как *невеста*.

При записи команд в машинном коде, т.е. на языке, понятном процессору, таких ситуаций на порядки больше. Неудивительно, что программисты обычно отдают предпочтение использованию специальных языков — языков программирования, — позволяющих записывать программу в форме, приближенной к естественным языкам. А поскольку процессор по-прежнему понимает запись программы только в машинном коде, возникает потребность в использовании целого ряда инструментов, позволяющих выполнять соответствующее преобразование, а также решать другие вспомогательные задачи.

Инструменты программиста

1) Текстовый редактор (text editor)

В большинстве языков программирования программа представляет собой записанный по определённым правилам текст. Вводить его сразу в двоичном коде — удовольствие не менее сомнительное, чем разработка сложной программы в машинных кодах, поэтому программа, позволяющая набирать текст с клавиатуры и вносить в него изменения по мере необходимости, — едва ли не самый востребованный инструмент программиста.

2) Транслятор (translator)

> Транслятор — это программа, которая занимается переводом (т.е. преобразованием) других программ.

При этом выделяют входной язык — тот, на котором программа написана изначально. Транслятор преобразовывает программу, записанную на входном языке, в эквивалентную ей программу на другом языке.

Выделяют два особых вида трансляторов: компиляторы и интепретаторы.

> Компилятор — это транслятор, который преобразует программу на входном языке в эквивалентный машинный код.

> Интерпретатор — это транслятор, результатом работы которого является не представление исходной программы на другом языке, а непосредственно результаты её работы.

Далеко не все трансляторы можно разделить на компиляторы и интерпретаторы. Например, в настоящее время стал популярным вид трансляторов, который получил название «транспайлер». Исторически этот вид трансляторов никак не назывался, однако с появлением языков наподобие TypeScript и CoffeeScript потребовалось и новое броское название взамен слова «транслятор», кажущегося скучным и устаревшим.

3) Редактор связей (linker)

При разработке крупных программ удобно разделять их на более мелкие самостоятельные части — модули. Например, в первом приближении в современных 3D-играх можно выделить модули, отвечающие за расчёты физических процессов, воспроизведение звуков, отображение графики, управление игровой логикой (подсчёт очков и т.п.), поведение компьютерных персонажей (также часто называется словосочетанием «искусственный интеллект», или AI) и т.п.

Каждый из таких модулей может быть полезен не только в том проекте, для которого он разрабатывается, но и в других, аналогичных проектах. По этой причине бывает удобно выполнять обработку модулей раздельно. Кроме того, такое разделение помогает ускорить формирование (сборку) исполняемого файла в ходе разработки, поскольку между такими сборками многие модули остаются неизменными и можно не выполнять их повторную компиляцию. Особенно это важно для языков программирования, обработка которых традиционно выполняется крайне медленно — C и C++.

Когда модуль записан в виде текста на входном языке, т.е. исходного кода, он называется *исходным модулем*. В результате компиляции он преобразуется в *объектный модуль*. В объектном модуле содержатся команды из исходного модуля, но уже переведённые в машинный код, а также некоторая дополнительная информация (для разных компиляторов она может быть разной). Затем в дело вступает *редактор связей* — программа, отвечающая за объединение объектных модулей в исполняемый файл. Он использует информацию из объектных модулей для того, чтобы понять, как правильно объединить их содержимое для получения готовой к запуску программы.

Результат работы редактора связей традиционно называют исполняемым файлом, однако его более точное название — *загрузочный модуль*, т.е. модуль, который готов к тому, чтобы быть загруженным в память для последующего выполнения. Эту загрузку выполняет уже операционная система на компьютере конечного пользователя. Чаще всего недостаточно просто скопировать содержимое загрузочного модуля в оперативную память: обычно требуется выполнить его дополнительную настройку, чем опять же занимается операционная система. Результатом такой настройки является *абсолютный модуль*. Именно им представлена программа, выполняющаяся на компьютере в данный момент.

4) Библиотека стандартных программ

Многие вспомогательные задачи, например, такие, как взаимодействие с пользователем, обработка строковых данных и т.п., повторяются от программы к программе. Готовые программы, предназначенные для решения таких типовых задач, как правило, включаются в комплект поставки транслятора, а некоторые из них также считаются неотъемлемой частью входного языка.

5) Документация

Самый важный инструмент для опытного программиста. Документация — это своего рода контракт между программистом и разработчиками тех средств, которыми он пользуется для достижения своих целей.

Программист, не умеющий читать документацию, обречён быть вечно отстающим, ведь именно документация является первоисточником информации о языке программирования, трансляторе, редакторе связей, операционной системе, аппаратных ключах и микроконтроллерах, с которыми будет общаться программа и т.д. Если программист не умеет работать с документацией, ему приходится ждать выхода книг наподобие «C++ за 10 минут», которые являются всего лишь вольным пересказом (и зачастую с ошибками!) той самой документации — например, спецификации языка программирования.

6) Средства отладки (debugging tools)

Английский поэт XVIII века Александр Поуп однажды написал: «To err is human, to forgive divine». В переводе это означает: «Человеку свойственно ошибаться, а Богу — прощать», или чуть ближе к оригиналу — «Ошибаться человечно, прощать божественно». Перефразируем его и скажем: «To err is human, to debug divine».

Продолжим цитировать великих. Как сказал кто-то из древних, «безглючен только begin…end, и то не всегда». Если Вы написали программу и она сразу заработала правильно, остановитесь и подумайте, очень уж это подозрительно. Редкая программа обходится без допущенных при её написании ошибок. И зачастую, даже зная об ошибке, бывает сложно обнаружить её причину.

Отладочные средства (чаще всего это программа, которую называют просто «отладчик») позволяют программисту проанализировать ход выполнения написанной им программы и сравнить получаемые результаты с ожидаемыми. А дальше всё просто: скорее всего, причина ошибки кроется именно там, где ожидания перестают совпадать с реальностью. Есть, конечно, и исключения из этого правила, но они редки и грамотный программист с лёгкостью таких ситуаций избегает.

7) И др.

Самый многочисленный класс программ. Всё, что не попадает в другие группы, оказывается здесь. В зависимости от того, какую именно программу нужно разработать, набор инструментов может существенно варьироваться. Например, при разработке мобильных приложений часто применяют специальные программы-эмуляторы, а при разработке desktop-приложений может быть полезно иметь под рукой программу, позволяющую перехватывать и отображать отдельно отладочный вывод других программ, или программу, позволяющую проанализировать имеющиеся в системе окна других программ. А ещё часто нужны графические редакторы, чтобы нарисовать красивые значки для использования в программе, средства управления ресурсами программы, системы контроля версий и т.д.

Среды программирования

Конечно, работать с таким многообразием программ по отдельности крайне затруднительно, поэтому в XXI веке уже никого не удивить таким явлением, как система (или среда) программирования.

> Система (среда) программирования — комплекс программ, включающий в себя инструменты для разработки программ.

Строго говоря, если подобрать некоторую комбинацию названных выше программ так, чтобы результаты работы одних из них могли быть использованы другими, — это уже система программирования. Не получится системы программирования разве что в том случае, когда, например, выбранный редактор связей не умеет работать с объектными модулями, которые формирует выбранный компилятор. Хотя такие несоответствия — отнюдь не редкость.

Тем не менее, даже в 80-х годах прошлого века уже активно пользовались интегрированными средами программирования.

> Интегрированная среда программирования — это среда программирования, в которой отдельные её компоненты (программы) настроены так, чтобы обмениваться информацией между собой.

Если в Вашей среде программирования можно набрать текст программы, поставить курсор на какое-либо ключевое слово и тут же получить подсказку по соответствующей команде, а затем запустить программу и тут же, глядя на исходный код программы, проследить её выполнение по шагам — принимайте поздравления, Ваша среда программирования уже вполне себе интегрированная.

Разумеется, прогресс пошёл дальше — и сегодня уже в порядке вещей такие достижения цивилизации, как RAD, CASE-средства, языки 4-го и 5-го поколений (сюда относится, например, Delphi) и т.п. Впрочем, многие программисты предпочитают оставаться в каменном веке и вести большую часть разработки по старинке — например, задавать пользовательский интерфейс своей программы исходным кодом, а не визуально, настраивать все взаимосвязи в программе вручную — поэтому по-прежнему полезно знать и понимать, какие инструменты могут пригодиться для повседневной работы в этой профессии.