Глава 3

Как написать программу

В этой главе...

- Прежде чем писать программу
- Написание программы: технические подробности
- Жизненный цикл типичной программы

Вы можете сесть за компьютер и приступить к написанию программы прямо сейчас, безо всякого планирования, однако результат будет таким же плачевным, как и при приготовлении бисквита смешиванием произвольных ингредиентов в произвольной пропорции.

Без предварительной подготовки можно написать простую программу, отображающую на экране имя вашего любимого кота, но для какого-нибудь проекта посложнее программу нужно сначала написать (или по крайней мере спланировать) на бумаге. Однако и это еще не все. Перед написанием собственно программы вы должны тщательно обдумать, как она должна выполнять поставленную перед ней задачу.

Прежде чем писать программу

Постарайтесь определить структуру программы, прежде чем разрабатывать ее. Только тогда вы не потратите время впустую на создание программы, которая не работает или работает неправильно. Планируя создание программы, вы значительно увеличиваете вероятность того, что она будет работать правильно и справится с поставленной перед ней задачей.

Перед созданием программы найдите ответы на четыре вопроса.

- ✓ **Задача.** Какую задачу должна решать программа? Если вы не сможете ясно сформулировать, что должна делать программа, написать ее вам не удастся.
- ✓ **Пользователи.** Кто будет использовать программу?
- ✓ **Целевой компьютер.** Какой компьютер понадобится для выполнения программы? Компьютер, работающий под управлением Windows 98/Me/NT/2000/XP или Mac OS, компьютер Amiga, мэйнфрейм, карманный компьютер, работающий под управлением Palm OS или PocketPC, или настоящий суперкомпьютер?
- ✓ **Автор программы.** Вы планируете писать программу самостоятельно или совместно с другими программистами? Если это будет коллективный труд, то как будет распределена работа над программой? В любом из этих случаев, каков уровень вашей квалификации? Какой "вес" вы можете взять?

Задача

Каждая программа решает некоторую задачу или несколько задач. Например, программа расчета налогов решает задачи организации финансовых данных и заполнения налоговой

декларации. Текстовый процессор решает задачи преобразования последовательности нажатия клавиш в последовательность символов и вывода символов на экран. И даже программа видеоигры решает важную задачу — развлечение человека.

Программа полезна лишь в той степени, в которой ей удается справиться с поставленной задачей. В нашем несовершенном мире почти нет проблем, которые удается решить "раз, навсегда и совсем", и большинство программ всего лишь упрощают и автоматизируют их решение. Например, программа управления финансами не сделает никого богатым при первом же запуске, она лишь поможет организовать обработку финансовых данных и автоматизировать арифметические операции. Целью любой программы является ускорение и облегчение решения реальной задачи. Программа — всего лишь инструмент решения задачи, следовательно, первым этапом разработки программы всегда должен быть ее анализ.

Пользователи программы

Если единственным пользователем новой программы будете вы сами, довольно просто создать программу, которая будет выглядеть и работать так, как хотите именно вы, поскольку вы, будем надеяться, знаете, чего вы хотите. Но если нужно предоставить или продать программу другим людям, обязательно надо знать, кто эти люди.

Узнать предпочтения потенциальных пользователей программы очень важно и зачастую критично для ее успеха. Если по какой-то причине программа придется не по душе пользователям, они вряд ли купят ее. То, как именно работает программа (т.е. ее "внутренности"), часто оказывается несущественным.

Подойдя к проектированию с учетом особенностей потенциальных пользователей, вы значительно увеличите шансы программы на успех, а значит, получите возможность заработать на продаже копий программы.

Даже если вы напишете безупречно работающую программу, пользователи все равно проигнорируют ее, если им не понравится ее графический интерфейс, если им будет непонятно, как заставить программу выполнить то или иное действие, если она работает не так, как программа, к которой они уже привыкли, если в интерфейсе программы используются неудачные цветовые комбинации и т.д. Основная задача программиста — написать программу, которая будет максимально соответствовать пожеланиям пользователей, независимо от того, насколько нелогичными они могут выглядеть (пожелания, а не пользователи, конечно же).

Целевой компьютер

Определив потенциального пользователя, необходимо определить, какой компьютер и операционная система ему потребуются для запуска программы и работы с ней. Тип компьютера и операционной системы в значительной мере определяют выбор языка программирования, на котором будет написана программа. Необходимо также определить требования к оборудованию и минимальный объем оперативной памяти, необходимые для правильной работы программы.

Например, если программа будет запускаться на компьютере Macintosh, то можно рассчитывать на такие преимущества, как отличные звуковая и графическая подсистемы, большой жесткий диск и значительный объем памяти. Однако, если программа должна работать на карманном компьютере Palm, ее придется переписать практически с нуля, так как возможности такого компьютера для воспроизведения звука и отображения графики значительно скромнее.

Если программу можно запустить, просто скопировав с компьютера одного типа на компьютер другого типа вообще без изменений или с небольшими изменениями, то ее принято называть *переносимой*, или *кроссплатформенной*. Переносимость программы в значительной мере зависит от того, какой язык использовался для ее написания. Именно по причине лучшей переносимости такие языки программирования, как C и C++, используются гораздо чаще, чем другие.

Переносимость, или кроссплатформенность

В большинстве случаев программисты стараются создавать такие программы, которые смогут работать на компьютерах многих типов, например на компьютерах, работающих под управлением Mac OS и Windows. Любая программа, которая работает на компьютерах нескольких типов, считается *кроссплатформенной*. Например, программа Microsoft Word кроссплатформенная, поскольку вы можете приобрести ее версию как для Mac OS, так и для Windows.

Программа, способная работать на компьютерах нескольких типов, значительно расширяет круг потенциальных заказчиков, однако и приводит к увеличению потенциальных проблем, с которыми вы рискуете столкнуться. Например, вам придется предоставлять техническую поддержку пользователям каждой версии программы и пытаться заставить все версии программы работать совершенно одинаково, несмотря на то, что они работают под управлением различных операционных систем и на компьютерах с совершенно различными возможностями.

Программные инструменты REALbasic, Revolution и Java создавались таким образом, чтобы созданные с их помощью программы можно было без изменений запускать на разных платформах. В значительной степени эта цель достигнута, однако во многих случаях программу все же приходится немного изменять для учета особенностей разных операционных систем. Несмотря на необходимость небольших изменений, это все же большое достижение, поскольку для переноса на другую платформу программ, написанных на других языках, необходима их радикальная переделка.

Ваша квалификация

При проектировании любой программы вам не следует забывать о собственном уровне подготовки. Вы можете вынашивать просто гениальные идеи, но если вы начинающий программист с небольшим опытом, то на создание программы вы потратите уйму времени (если вы сразу не бросите это занятие).

Ваш уровень подготовки и опыт написания программ определяют, какому именно языку программирования вы отдадите предпочтение. Опытные программисты практически всегда пишут программы на языке C, C# или C++. Новичок же стоит перед выбором: либо потратить много времени на изучение одного из этих языков, прежде чем написать первую программу, либо применить более простой язык, например BASIC.

Некоторые новички тратят слишком много времени на изучение сложных языков, таких как C и C++, после чего приступают к написанию своей первой программы. Другие предпочитают более легкий подход и выбирают простой язык программирования, например Visual Basic, чтобы немедленно приступить к созданию (и продаже) программ. Не надо бояться начинать с изучения сложного языка программирования, но не стоит также избегать языка попроще. Основная ваша цель — написать программу, которую вы сможете или использовать сами, или продавать другим.

Многие непрофессиональные программисты создают программу на таком простом языке, как Visual Basic, а затем нанимают профессиональных программистов, чтобы те переписали программу на более сложный язык, например C или C++, благодаря чему программа заработает быстрее и эффективнее. Однако в большинстве случаев такой метод не приводит к положительному эффекту, поскольку в разных языках одни и те же операции выполняются одинаковыми способами. Положительный эффект происходит не из-за смены языка, а благодаря тому, что профессиональный программист в некоторых случаях может обнаружить и переделать нерационально запрограммированные операции.

Опасайтесь "золотых наручников"!

Вместо того чтобы изучать программирование самостоятельно, многие люди предпочитают нанимать других для написания программ. Будьте осторожны! Многие программисты — свободные художники, живущие по принципу: "Заплатите побольше, а когда закончу работу — хоть потоп".

Вот как это происходит. Вы нанимаете человека для написания необходимой программы. Сначала человек получает деньги, а затем пишет программу, которая работает не совсем так, как вы хотели. Затем, вместо того, чтобы махнуть рукой на потерянные деньги, вы платите программисту дополнительную сумму, после чего он пишет новый вариант программы, который все равно работает не так, как надо. "Золотые наручники" защелкнулись (название означает следующее: золото — программисту, наручники — вам).

Вы оказываетесь в затруднительном положении. Продолжите ли вы платить программисту, который, как вы уже поняли, никогда не выполнит свою работу, или смиритесь с потерей денег? Хуже того: вы не сможете нанять другого программиста для продолжения работы над программой, так как ее исходный код находится у первого программиста, ее автора, а значит, никто другой не сможет внести в нее изменения. Единственный выход из ситуации — нанять нового программиста и начать все сначала. Однако нет худа без добра. Перспектива оказаться в подобной ситуации заставляет опытных заказчиков высоко ценить и хорошо оплачивать работу, выполненную быстро и качественно.

Написание программы: технические подробности

Создать программу за один день невозможно. Программы эволюционируют со временем. Ввод команд языка требует много времени, и при этом часто возникают проблемы, поэтому опытный программист всегда старается не приступать к написанию программы до тех пор, пока не будет абсолютно уверен в том, что он знает, что делает.

Создание прототипов

Чтобы убедиться в том, что многие месяцы (или даже годы) не будут потрачены на разработку программы, которая будет работать не так, как нужно, программисты часто начинают с создания *прототипа* программы. Точно так же, как архитекторы начинают с создания картонных или пластиковых моделей небоскребов, программисты, прежде чем приступать к реальному строительству, начинают с макета (прототипа) будущей программы.

Прототип программы обычно содержит графический интерфейс пользователя (раскрывающиеся меню, кнопки, диалоговые окна). Выглядит прототип, как настоящая программа,

однако щелчки на командах меню и кнопках не приводят ни к каким результатам. Основная идея прототипа — показать заказчику (и увидеть самому), как будет выглядеть программа и как она будет работать, не тратя пока что время на создание программы.

Сконструировав прототип программы, программист может продолжить работу, используя прототип для создания окончательного варианта программы.

Многие программисты используют языки RAD (например, C# или Visual Basic), поскольку они позволяют легко и быстро создавать прототипы программ. Создав на Visual Basic прототип, который продемонстрирует работу графического интерфейса программы, смело приступайте к добавлению команд и превращению прототипа в настоящую полноценную программу.

Выбор языка программирования

Любую программу можно написать на любом языке программирования, однако каждый язык приспособлен для создания программ определенного класса.

Выбор языка программирования часто разделяет людей так же сильно, как и принадлежность к определенной религии или политической партии. Идеального языка программирования, подходящего для любой ситуации, не существует, поэтому рассмотрите несколько языков. Если программа работает хорошо, абсолютно никого не будет интересовать, на каком языке она написана. В следующих подразделах рассматриваются причины использования для написании программы того или иного языка программирования. О преимуществах и недостатках разных языков см. в главе 2.

Универсальные и специализированные языки

На универсальном языке программирования (например, на C, C#, C++, BASIC, Pascal, ассемблер) можно создать практически любую программу, однако это не значит, что именно так и нужно делать. Для решения специфических задач создано огромное количество специализированных языков, позволяющих решить задачу намного быстрее и легче, чем с помощью универсального языка.

Например, малоизвестный язык SNOBOL специально приспособлен для манипулирования текстом. Если программа должна только управлять текстами, то написать ее на SNOBOL значительно легче, чем на C или C++. Если же программа должна делать еще что-либо, кроме манипулирования текстом, то язык SNOBOL для ее создания абсолютно непригоден.

Еще пример. Языки LISP и Prolog созданы для разработки программ искусственного интеллекта и логического анализа. Для этого в них встроены команды логического вывода. Эти же программы можно разрабатывать и на любом из универсальных языков, однако тогда для имитации одной команды LISP или Prolog, автоматически выполняющей анализ ситуации, придется писать довольно длинный фрагмент кода.

Следовательно, выбор языка, наиболее подходящего для данного класса задач, может намного облегчить разработку программы.

Как должна работать программа

Выбрав подходящий язык программирования, не спешите приступать к вводу текста программы. Точно так же, как программисты создают прототипы графического интерфейса будущей программы, они часто создают и прототипы (макеты) инструкций, которые четко описывают, как должна работать программа. Подобные макеты инструкций называют *псевдокодами*.

Использование нескольких языков

Программа не обязательно должна быть написана с помощью одного языка программирования (например, C++). Некоторые компиляторы способны преобразовать исходный код в специальный файл, который называется *объектным файлом*. Основное преимущество объектных файлов состоит в том, что один программист может использовать для создания программы, например, C++, второй — язык ассемблера, а третий — Pascal. Каждый из них пишет свою часть программы на своем любимом языке и сохраняет ее в виде отдельного объектного файла. Затем все объектные файлы связываются и образуют одну большую программу. Программа, которая отвечает за связывание объектных файлов, называется *компоновщиком*, или *редактором связей*.

В мире Microsoft Windows возможность написания программы на нескольких языках базируется на применении динамически подключаемых библиотек DLL, т.е. специальных фрагментов программ. При создании трех файлов DLL могут использоваться три разных языка, например C, Java и COBOL. Затем четвертый программист пишет программу на Visual Basic, которая обеспечивает интерфейс пользователя, а остальные команды извлекаются из трех файлов DLL.

Третий способ состоит в написании программы с использованием любимого языка (например, Pascal). После этого на ассемблере пишут инструкции, которые невозможно или тяжело написать на языке Pascal. Только учитывайте, что переключаться между разными языками в рамках одной программы позволяют далеко не все компиляторы.

И наконец, невозможно не упомянуть о многоязыковой платформе .NET, разработанной компанией Microsoft. В среде программирования .NET в рамках одного проекта каждый программист может разрабатывать свой модуль на своем любимом языке (Visual Basic, Visual C# и др.). Затем каждый модуль транслируется в файл IL (Intermediate Language — промежуточный язык). После этого уже не имеет значения, на каком языке создавался модуль, все они будут автоматически состыкованы друг с другом компилятором IL. Поэтому в среде .NET каждую часть программы можно писать на языке, наиболее приспособленном именно для этой части. В принципе то же самое можно делать и с помощью объектных файлов, однако для создания таких модных сейчас графических интерфейсов пользователя объектные файлы практически непригодны, в то время как платформа .NET для этого весьма удобна, поскольку она разрабатывалась специально для этого.

Например, если нужно написать программу, которая будет управлять полетом ядерной ракеты, направляя ее к какому-нибудь городу для уничтожения любых признаков жизни в радиусе 100 км, псевдокод программы будет выглядеть следующим образом:

```
1.Получить координаты цели.
2.Получить текущие координаты ракеты.
3.Вычислить траекторию ракеты для попадания в цель.
4.Взорвать ядерную боеголовку.
```

Используя псевдокод, вы сможете быстрее выявить оплошности в логической последовательности действий еще до написания программы, так как в тексте программы такие недочеты легко скрываются за сложным синтаксисом команд.

Каждая инструкция псевдокода в предыдущем примере требует дополнительных уточнений для того, чтобы можно было приступать к написанию программы. Вы не можете просто дать компьютеру команду “Получить координаты цели”, поскольку он сразу “поинтересуется” у вас, а откуда он получит эти координаты. Поэтому псевдокод придется переписать следующим образом:

```
1.Получить координаты цели.
  а) Попросить технический персонал ввести координаты цели.
  б) Убедиться в том, что указанные координаты цели имеют
     допустимые значения.
```

```
  в) Сохранить координаты цели в памяти.
2.Получить текущие координаты ракеты.
3.Вычислить траекторию ракеты для попадания в цель.
4.Взорвать ядерную боеголовку.
```

Однако инструкции псевдокода нуждаются в еще большей детализации. Например, пункт 1б нужно уточнить следующим образом:

```
1.Получить координаты цели.
  а) Попросить технический персонал ввести координаты цели.
  б) Убедиться в том, что указанные координаты цели имеют
     допустимые значения, для этого
        -убедиться в том, что координаты указаны полностью,
        -убедиться в том, что указанные координаты
         достижимы для ракеты,
        -убедиться в том, что указанные координаты
         не приведут к поражению дружественных территорий.
  в) Сохранить координаты цели в памяти.
2.Получить текущие координаты ракеты.
3.Вычислить траекторию ракеты для попадания в цель.
4.Взорвать ядерную боеголовку.
```

Определение общей задачи, с которой должна справляться программа, и детализация каждого пункта программы называется *нисходящим проектированием программы*. Другими словами, проектирование начинается с самого верха — с формулировки общей задачи, поставленной перед программой. После чего, спускаясь вниз, программист конкретизирует каждую задачу, разбивая ее на подзадачи. Затем он разбивает каждую подзадачу. Процесс повторяется до тех пор, пока не будет описана каждая элементарная операция, которую может выполнить компьютер.

Написание псевдокода требует больших затрат времени. Однако немедленное написание программы безо всякого планирования аналогично тому, что вы сядете в автомобиль и немедленно поедете, не взглянув на карту и не спланировав маршрут.

Псевдокод — полезный инструмент для описания общей структуры программы. С его помощью легче определить, какие данные потребуются программе при работе. Основная идея состоит в том, чтобы сначала написать пошаговую инструкцию (т.е. псевдокод) на обычном “человеческом” языке, а затем на ее основе написать программу на языке программирования (C, C#, C++, FORTRAN, Pascal, Java или любом другом предпочтительном для вас).

Жизненный цикл типичной программы

В некоторых случаях программист пишет программу, передает ее заказчику, а затем оставляет ее без всякого внимания. Однако намного чаще программа все-таки проходит через различные циклы, в течение которых она постоянно обновляется, пока необходимость в ней не отпадет сама собой. Например, компании, разрабатывающие текстовые процессоры, постоянно обновляют их, а пользователи с удовольствием приобретают новые версии, хотя алфавит остается неизменным на протяжении столетий.

В самом общем случае почти каждая программа проходит через цикл разработки (в течение которого ее создают и продают), цикл сопровождения (когда разработчик как можно быстрее исправляет ошибки, обнаруженные пользователями при работе с программой) и цикл обновления (когда в программу добавляются новые средства, чтобы продать ее дороже).

Цикл разработки

Каждая программа начинается с пустого экрана перед глазами программиста. Во время разработки программист проходит путь от идеи до работающей программы. Процесс разработки состоит из нескольких этапов.

1. **Формулировка общей идеи программы.**
2. **Принятие решения о потенциальных пользователях программы.**
3. **Принятие решения о типе компьютера, на котором программа будет выполняться.**
4. **Выбор языка программирования.**
5. **Проектирование структуры программы с помощью псевдокода или другого инструмента.**
6. **Написание программы.**
7. **Тестирование программы без участия пользователей.**

   Этот этап называют альфа-тестированием.
8. **Исправление ошибок, обнаруженных во время альфа-тестирования.**

   Этапы 7 и 8 повторяются многократно.
9. **Передача копий программы пользователям для ее тестирования "в полевых условиях".**

   Этот этап называют бета-тестированием.
10. **Исправление ошибок, обнаруженных во время бета-тестирования.**

    Этапы 9 и 10 повторяются многократно.
11. **Выпуск окончательной версии программы. Лишь с этого момента разработчики гарантируют безупречную работу программы (естественно, гарантия не 100%-ная).**

Цикл сопровождения

Многие программисты предпочитают писать новые программы, а не сопровождать или изменять программы, написанные другими людьми. Это вполне естественно, поскольку копаться в чужой программе намного тяжелее, чем в своей. Однако написать заново сложную программу еще тяжелее. Представьте, что было бы, если бы каждый программист, вновь нанятый компанией Microsoft, начал заново создавать программу Word.

Ниже приведен приблизительный список этапов, выполняемых при сопровождении существующей программы.

1. **Просмотр всех отчетов об обнаруженных ошибках. Необходимо выяснить, какая часть программы обусловила их появление.**
2. **Исправление ошибок.**
3. **Проверка работы программы, чтобы убедиться в том, что все обнаруженные ошибки исправлены, а новые не появились.**
4. **Исправление всех ошибок, обнаруженных во время проверки.**
5. **Повторение этапов 1–4, до тех пор, пока не будут исправлены все ошибки.**

   Такова уж природа программного обеспечения: при исправлении старых ошибок обязательно появляются новые.

6. **Передача исправленной программы пользователям.**

   Если объем программы большой, то иногда пользователям передают не исправленную программу, а так называемую "заплатку" — исправленный фрагмент программы и средства его записи в старую программу.

Цикл обновления

Компании зарабатывают деньги не на том, что выпускают "заплатки" для программ, а на том, что продают новые версии программ, снабжая их новыми функциями и параметрами.

Независимо от того, будете вы работать на компанию или на себя, вам довольно часто придется обновлять программы, написанные вами или другими людьми. Цикл обновления программы обычно состоит из этапов, приведенных ниже.

1. **Определение новых функций, которые нужно добавить в программу.**
2. **Планирование работы новых функций (с помощью псевдокода или других средств).**
3. **Внесение изменений в программу.**
4. **Проверка работоспособности новых функций (этап альфа-тестирования).**
5. **Исправление ошибок, выявленных при альфа-тестировании.**
6. **Передача копий программы пользователям для бета-тестирования.**
7. **Исправление ошибок, выявленных при бета-тестировании.**
8. **Повторение этапов 1–7 для каждой функции, добавляемой в программу.**
9. **Выпуск новой версии программы и ожидание сообщений об ошибках. При первых же сообщениях начинается цикл поддержки программы.**

Несмотря на существование многочисленных научных дисциплин по компьютерной тематике и громкие должности наподобие "самый главный программист", программирование до сих пор остается больше искусством, чем наукой. Для написания и обновления программ не нужны ни высокий IQ (коэффициент интеллекта), ни ученая степень по математике. Для этого нужны настойчивость и воображение. Писать программы можно любым способом, однако, чтобы уменьшить проблемы с ними в будущем, к их написанию следует относиться как можно более организованно и методично.