Предисловие

Подготовку профессиональных программистов важно вести на базе общего подмножества языков С и С++. Книга поможет перейти к использованию языков С и С++ также тем, кто овладел языком Pascal.

Первая часть учебного пособия посвящена основным понятиям и базовым методам алгоритмизации и программирования, знакомству с типовыми средствами языков высокого уровня на примере общего подмножества двух языков, называемого в учебнике «язык С/С++».

Во второй части пособия более глубоко рассматриваются методы и приемы разработки и программной реализации структур данных и алгоритмов, составляющие основу техники и технологии программирования.

Основное содержание данного пособия составляют методы систематического построения алгоритмов и их реализации в виде компьютерных программ. Такой подход неизбежно затрагивает необходимые разработчику алгоритмов разнообразные и взаимосвязанные методы многочисленных смежных дисциплин: математики и информатики, принципов работы устройств компьютера, языков программирования, операционных систем, технологии программирования и др.

Тесная взаимосвязь этих разделов информатики затрудняет их последовательное изложение и изучение, постоянно требует ссылок вперед на еще не рассмотренные понятия, возврата к уже изученному материалу, а также поиска справок по многочисленным вопросам. Некоторые средства и методы зачастую используются в примерах до их последующего формального рассмотрения.

Учитывая все это, автор старался построить книгу так, чтобы ее можно было использовать не только в качестве учебника для начальной и углубленной подготовки, но и как справочник, в том числе при самостоятельном изучении программирования.

Глава 1 первой части книги посвящена первоначальному знакомству с основными понятиями алгоритмизации и программирования, необходимыми для понимания остальных разделов. Здесь, в частности, дается поверхностный обзор важнейших проблем программирования, а также рекомендации по методике его изучения.

В ***главе 2*** рассматриваются основные конструкции базового языка программирования. Язык при этом рассматривается как главный инструмент программиста, но не как самоцель вводного курса программирования. Как правило, основные обозначения языка вводятся по мере необходимости, а овладение деталями приходит постепенно с практикой решения задач, которое необходимо начинать как можно раньше. Поэтому рекомендуется при первом чтении лишь бегло ознакомиться с содержанием этой главы, а в дальнейшем использовать ее как справочник по языку, наряду с главами 5, 6, 8–10, посвященными более сложным средствам языка, и приложениями.

Глава 3 посвящена основным элементам технологии программирования и в первую очередь важнейшим методам систематического построения алгоритмов, вызывающего наибольшие трудности у начинающих программистов.

В ***главе 4*** рассматриваются методы однопроходной последовательной обработки данных.

В ***главе 5*** изучаются методы решения разнообразных задач с использованием массивов.

Глава 6 посвящена средствам и методам использования подпрограмм.

Глава 7 представляет собой краткий обзор алгоритмов сортировки информации, которые не только нужны для практики, но и являются хорошими примерами использования разнообразных методов и приемов программирования.

В ***главе 8*** кратко рассматриваются базовые средства языка C/C++ для обработки файлов и связи с операционной средой.

В ***главе 9*** приведены несколько примеров программирования вычислительных задач и обработки файлов.

Глава 10 содержит краткое введение в макросы и средства условной трансляции, имеющие большое значение в технологии программирования.

Каждый раздел снабжен упражнениями и задачами, в том числе олимпиадного характера (поскольку автор считает решение олимпиадных задач превосходным средством изучения методов программирования). Для наиболее важных и сложных задач приводятся решения либо указания к решению (номера таких задач подчеркнуты).

Приложения включают справочник по синтаксису и стандартным функциям языка C/C++, а также небольшой «фразеологический словарь» C — Pascal, облегчающий переход от одного из этих языков к другому.

Глава 1

Основные понятия программирования

1.1. Программное обеспечение ЭВМ

В наше время большинство населения развитых стран составляют *пользователи ЭВМ*, т. е. люди, в той или иной мере использующие компьютеры для решения разнообразных задач в сферах науки, проектирования, производства, экономики, управления, образования, медицины, культуры, развлечений и многих других. Но даже далекие от информатики люди знают, что сам по себе компьютер не может решить ни одной задачи. Это всего лишь автомат, быстро и точно выполняющий заданные ему программы, в которых заключен весь его «интеллект» — детальное описание последовательностей операций по решению задач обработки информации.

Программное обеспечение (ПО) — совокупность программ для ЭВМ — играет основную роль в успехе применения компьютеров.

Сравнительно простые и рутинные задачи компьютер может решать в автоматическом *пакетном режиме*, выполняя последовательность (пакет) программ с заранее известными исходными данными без вмешательства человека. Однако в наиболее важных, интересных и сложных случаях используется *интерактивный (диалоговый) режим* работы программы с непосредственным участием человека, который занимается принципиальными творческими проблемами и ведет с компьютером *диалог* — обмен информацией, вопросами и ответами в процессе решения задачи.

Для решения задач в диалоговом режиме создаются разнообразные *автоматизированные (компьютерные)*

системы, включающие аппаратуру, программы и людей. Примерами являются автоматизированные системы для обучения, научных исследований, проектирования, управления технологическими процессами производства и т. п.

Программное обеспечение вместе с аппаратурой ЭВМ образует *вычислительную систему* и является ее основной частью. Достаточно сказать, что доля затрат на программное обеспечение во всем мире постоянно растет и составляет до 80–90 % всех затрат на вычислительные системы. Говорят даже, что она относится к затратам на аппаратуру «как стоимость товара к стоимости его упаковки».

Программное обеспечение делится на прикладное, системное и инструментальное. Иногда инструментальное ПО относят к системному ПО.

Прикладное ПО содержит программы и программные комплексы, предназначенные для решения конкретных прикладных задач.

Системное (общее) ПО включает программы, обеспечивающие функционирование вычислительной системы как единого целого и необходимые для решения всех задач, независимо от конкретной области применения [38].

Основной частью системного ПО является *операционная система* — комплекс программ, управляющий устройствами вычислительной системы и выполнением всех остальных программ, в том числе их взаимодействием с аппаратурой, другими программами и пользователями. Операционная система вместе с аппаратурой ЭВМ создает *операционную среду — программно-аппаратную платформу* для выполнения прикладных программ и непосредственной работы пользователей.

К системному ПО также относят сервисные и антивирусные программы, испытательные программы для проверки исправности аппаратуры и т.п.

В *инструментальное ПО* входят средства для разработки программ: системы программирования, инструментальные комплексы и отдельные программы для автоматизации разных этапов создания программного обеспечения.

Система программирования включает язык программирования, транслятор для него, библиотеку программ,

текстовый редактор, редактор связей, загрузчик, средства отладки и другие вспомогательные программы.

Текстовый редактор представляет собой программу для создания и изменения в памяти ЭВМ различных текстов: программ, писем, книг и т. п.

Язык программирования — это система понятий и обозначений для записи программ. Существуют тысячи языков программирования, из которых лишь десятки широко используются специалистами. Около десяти языков известны миллионам людей во всем мире: Basic, Pascal, C, C++, Java, Fortran, PL/1, Ada и др.

В данном пособии используются языки программирования C и C++. Язык C (произносится как «Си» — английское название латинской буквы «C») по своим возможностям близок к языку Pascal. Язык C++ является развитием языка C и включает усовершенствованный его вариант, а также ряд дополнительных средств. В данном пособии под названием «C/C++» понимается общее подмножество языков C++ и C; при этом из средств, отсутствующих в языке C, используются только строчный комментарий, потоковый ввод-вывод и ссылочные параметры функций.

Перечисленные языки являются *машинно-независимыми языками высокого уровня*. Машинная независимость означает возможность использовать язык для ЭВМ разных типов. *Уровень языка* определяется степенью его близости к машинному языку, непосредственно воспринимаемому компьютером. Чем большее число машинных команд требуется в среднем для выполнения одной команды некоторого языка, тем выше уровень этого языка.

К *машинно-зависимым (машинно-ориентированным)* языкам относят в основном *языки ассемблера* — для программирования на уровне машинных команд компьютеров определенного типа. Их называют языками уровня 1:1 («один к одному»), так как команда такого языка обычно соответствует одной машинной команде.

Для использования языка программирования на ЭВМ его нужно *реализовать* на ЭВМ этого типа, т. е. разработать для него *транслятор* — программу для перевода программ с одного языка на другой. Существуют различные

виды трансляторов: компилятор, интерпретатор, ассемблер, редактор связей, загрузчик и др.

Компилятор — это транслятор для перевода программ с языка высокого уровня на язык, близкий к машинному языку, без непосредственного выполнения команд этих программ. Последующее исполнение полученной машинной программы происходит уже без участия компилятора. Таким образом, перевод программы и ее выполнение разделены во времени. Программу на языке программирования называют *исходным модулем,* а результат ее трансляции (компиляции) — *объектным модулем* (от английского слова *object* — цель).

Интерпретатор — это транслятор, который читает, анализирует и немедленно выполняет каждую команду исходной программы без получения объектного модуля. В этом случае перевод программы и ее выполнение происходят параллельно.

Компилятор и интерпретатор в чистом виде представляют собой два крайних подхода к реализации языка программирования. В любом реальном трансляторе обычно сочетаются элементы компиляции и интерпретации.

Редактор связей (компоновщик) представляет собой программу для преобразования объектного модуля в *загрузочный (исполнимый) модуль*, т. е. программу, пригодную для загрузки в память и выполнения. Компоновщик связывает несколько раздельно транслированных объектных модулей в один загрузочный модуль, присоединяя к ним необходимые объектные модули из программных библиотек.

Загрузчик загружает программу (загрузочный модуль) в оперативную память для выполнения. *Перемещающий загрузчик* может настраивать загружаемую программу на работу в заданном месте памяти, а *связывающий загрузчик* может также выполнять функции редактора связей.

Данные — это информация, представленная в машинной форме, воспринимаемой устройствами компьютера. Программы, их исходные данные и результаты, а также другая информация хранятся в компьютерах в виде *файлов* — именованных совокупностей данных на магнитном

диске или другом носителе информации (обычно вне оперативной памяти).

В каждой операционной системе имеются свои правила именования файлов. Например, в распространенных на персональных компьютерах операционных системах UNIX, Windows или MS-DOS если исходный модуль, т. е. программа на языке C, находится в файле **p.c** (или **p.cpp** для языка C++), то файл с результатом ее компиляции (объектным модулем) обычно называется **p.obj**, а файл с исполнимым модулем — **p.exe** (англ. *execute* — исполнять).

1.2. Программирование

Составление программ для ЭВМ называют программированием. В более широком смысле *программирование* — это раздел информатики, изучающий теорию, методы и средства разработки, производства и эксплуатации программного обеспечения ЭВМ. В более узком смысле под программированием понимают также один из этапов разработки программы — перевод алгоритма на язык программирования.

Высокие затраты на программное обеспечение объясняются большой трудоемкостью разработки программ. Современные программы содержат много тысяч или даже миллионы команд. Обычно программист разрабатывает за год программы общим размером от 1000 до 10000 команд, т. е. производительность его труда составляет в среднем не более 5 команд в час.

Такая производительность может показаться очень низкой. Ведь команда — это одна строчка, которую можно написать быстрее, чем за минуту. Дело, однако, в том, что трудоемкость разработки очень быстро растет при увеличении программы. Трудоемкость приблизительно пропорциональна длине программы в степени от 1,5 до 2 (т. е. при увеличении количества команд в 10 раз трудоемкость возрастает в 30–100 раз).

Программирование требует практически безошибочной работы. Ошибка в одной единственной цифре может совершенно изменить смысл команды и полностью нарушить работу всей программы, содержащей огромное число

команд. Поэтому мало написать программу, ее еще надо тщательно проверить и отладить — найти и исправить в ней ошибки, что требует огромного труда.

Вот почему создание программного обеспечения обходится очень дорого — в среднем от 5 до 50 и более долларов за одну команду.

Программное обеспечение включает в себя в основном не одиночные программы, а сложные программные комплексы из большого количества взаимосвязанных программ. Большие программные системы — это самые сложные технические объекты, когда-либо создававшиеся человечеством. Количество команд в больших программах сопоставимо с числом элементарных деталей в таких сложных технических системах как, например, самолет, ракета или ЭВМ, а связаны команды между собой гораздо сложнее. Например, комплект документации программного обеспечения современного компьютера по объему в десятки раз может превосходить документацию по его аппаратной части.

Ясно поэтому, что разработка крупных программ приобрела индустриальный характер и требует участия больших коллективов программистов. Современное программирование — это скорее коллективный, чем индивидуальный труд.

Программирование является, с одной стороны, технологией и даже отраслью промышленности, а с другой стороны, может рассматриваться как ветвь прикладной математики. В этом рассмотрении программирование неразрывно связано с теорией алгоритмов. При этом традиционные методы математики оказались плохо применимыми к таким сложным объектам, как программы. Потребовалась разработка специальной «машинной» математики, и сейчас теория программирования интенсивно развивается.

Удивительная многоплановость программирования требует от программиста разносторонних способностей и знаний. Это очень точно и образно выразил крупнейший российский специалист в данной области академик А. П. Ершов, сказавший, что «программист должен обладать способностью первоклассного математика к абстрак-

ции и логическому мышлению в сочетании с эдисоновским талантом сооружать все, что угодно, из нуля и единицы. Он должен сочетать аккуратность бухгалтера с проницательностью разведчика, фантазию автора детективных романов с трезвой практичностью экономиста. А кроме того, программист должен иметь вкус к коллективной работе, понимать интересы пользователя и многое другое... Трудность заключается в том, что именно программисты непосредственно упираются в пределы человеческого познания в виде алгоритмически неразрешимых проблем и глубоких тайн работы человеческого мозга».

До половины и более всех затрат на разработку программы уходит на ее *отладку* — обнаружение ошибок в программе, их локализацию (поиск причины и местонахождения) и исправление.

Основным методом обнаружения ошибок в программе является *тестирование* — выполнение программы вручную или на ЭВМ с целью обнаружить в ней ошибки или изучить механизм ее работы. Для этого используются контрольные примеры (тесты). *Тест* — это пример исходных данных программы вместе с ожидаемым правильным результатом ее работы (или каким-либо средством проверки правильности результата, например, проверяющей программой).

Поскольку на примерах невозможно проверить все сочетания исходных данных, **тестирование позволяет показать лишь наличие ошибок в программе, но не их отсутствие!** Поэтому все реальные программы, как бы тщательно они ни отлаживались, как правило, содержат ошибки, которые могут привести к неправильному результату при некоторых исходных данных. Важнейшей проблемой программирования является уменьшение вероятности таких случаев — повышение *надежности* программ.

Другим методом борьбы с ошибками является *верификация* — доказательство правильности программы — не на примерах, а для общего случая, как доказывают математические теоремы и формулы. Такие методы интенсивно разрабатываются, однако они также имеют ряд недостатков:

1) доказательство очень трудоемко, по объему превышает программу и может содержать ошибки. Правда, ошибки в доказательстве обнаружить гораздо легче, чем в программе, и этот процесс можно автоматизировать, но ведь программа проверки доказательства сама может быть ошибочной!
2) доказать можно правильность решения программой только четко определенной математически сформулированной задачи, однако наиболее неприятные ошибки как раз и возникают при постановке (формулировке) задач, а обнаруживаются такие ошибки только тестированием.

Поэтому верификация не может полностью заменить тестирование, и эти две методики отладки необходимо применять комплексно.

1.3. Критерии качества программ

Требования к программам и критерии (показатели) их качества в основном вытекают из необходимости уменьшить затраты времени, труда, материальных, финансовых и других средств для решения задач на ЭВМ.

Существуют десятки взаимосвязанных *критериев оценки программ*, основные из которых следующие.

1. *Функциональность* — выполняемые программой функции, ее полезность.
2. *Надежность* — вероятность безошибочной работы программы в течение определенного времени.
3. Время решения задачи — *эффективность по времени*.
4. Требуемая память для программы и обрабатываемых данных — *эффективность по памяти*.
5. *Удобство эксплуатации* — определяется качествами самой программы и ее документации и зависит от многих факторов. Очень важна, например, *дружественность программы* — удобный и естественный для пользователя *интерфейс* (способ взаимодействия с программой), наличие в ней защиты от ошибок и развитых средств подсказки и диалоговой документации.