Предисловие

> «Программист должен обладать способностью первоклассного математика к абстракции и логическому мышлению в сочетании с эдисоновским талантом сооружать все, что угодно, из нуля и единицы».
>
> *Академик* ***А. П. Ершов***

Данная книга является продолжением первой части пособия [100] и во многом опирается на ее содержание и терминологию, но может использоваться и независимо от нее для углубленного изучения программирования.

В первой части книги рассматриваются основные понятия и базовые методы программирования, иллюстрируемые на примере языка C/C++.

Данная, вторая, часть посвящена более глубокому изучению методов организации данных и построения алгоритмов, а также технологии программирования.

В **разделе 1** кратко описываются базовые понятия и методы хранения сложных структур данных.

Раздел 2 целиком посвящен технике обработки списков. Эти методы играют большую роль в некоторых важных областях программирования, но, как показывает опыт автора, сравнительно мало знакомы большинству программистов. При первом чтении этот раздел можно пропустить без серьезного ущерба для понимания основного материала следующих разделов.

В **разделе 3** рассматриваются вопросы программной реализации и использования простых структур данных — очередей, стеков, деков, строк, массивов, множеств.

Более сложным структурам данных посвящены отдельные разделы. В **разделе 4** изучаются методы программной реализации операций над графами и деревьями, при этом основное внимание уделяется задачам обхода и поиска путей.

В **разделе 5** изучаются методы быстрого поиска информации в таблицах.

В **разделе 6** описываются некоторые базовые схемы построения алгоритмов, среди которых следует отметить перебор вариантов и динамическое программирование.

Раздел 7 представляет собой краткое введение в технику символьной обработки на примере метода рекурсивного спуска и алгоритмов трансляции выражений.

В **разделе 8** излагаются основы технологии модульного программирования и связанные с ними базовые понятия объектно-ориентированного программирования, иллюстрируемые небольшим примером.

В **разделе 9** приведен пример многомодульного проекта — разработки простого компилятора для упрощенного языка С.

Во всех указанных разделах рассматриваются методы решения задач с наиболее распространенными структурами данных и их реализация на языках высокого уровня. Даны алгоритмы и примеры решения таких задач на языке С и на его неформальной версии — псевдокоде. В то же время основное содержание рассматриваемых методов не привязано к языку С, и их можно использовать на любом языке (так, некоторые решения даны на языке Pascal). Для облегчения перехода от одного языка к другому в приложении приведен своего рода краткий «фразеологический словарь» С — Pascal.

Пособие снабжено большим количеством задач различной сложности — от простых упражнений до заданий олимпиадного уровня. Приведены решения наиболее важных и трудных задач (номера таких задач подчеркнуты).

В отличие от имеющейся литературы изложение материала в данном пособии носит практический характер, особое внимание уделено технике программирования. Изучаемые методы лежат в основе образования как системных, так и прикладных программистов.

Большое влияние на содержание учебника оказал многолетний опыт автора в подготовке студентов к олимпиадам по программированию и школьников к олимпиадам по информатике, в том числе участие в чемпионате мира среди студенческих команд. Задачи этих соревнований являются нестандартными, а их решение в полной мере спо-

собствует овладению богатым арсеналом разнообразных методов и алгоритмов во всех областях применения ЭВМ и развитию отточенной техники программирования.

Автор пользуется случаем выразить благодарность людям, без участия и влияния которых эта книга не могла бы появиться на свет, своим учителям и коллегам: Ф. З. Рохлину, В. М. Матросову, Л. И. Ожиганову, Х. Ф. Кулееву, В. И. Медведеву, О. М. Рякину, В. П. Ширикову, А. Р. Бикмурзиной, З. Х. Захаровой и другим, а также, конечно, своим ученикам, для которых она и написана. Особая признательность Ю. М. Баяковскому, консультации и программы которого способствовали первому приобщению автора к профессиональному программированию.

Используемые обозначения

Обозначение	Смысл
`m..n`	Последовательность целых чисел m, $m + 1$, ..., n
`X[m..n]`	Отрезок массива $X[m]$, $X[m + 1]$, ..., $X[n]$
`Очередь <== x;`	Включить в очередь значение x
`Очередь ==> x;`	Исключить из очереди элемент и записать в x
`Очередь ==> ;`	Исключить из очереди элемент (без запоминания)
`Стек <== x;`	Втолкнуть в стек значение x
`Стек ==> x;`	Вытолкнуть из стека элемент и записать в x
`Стек ==> ;`	Вытолкнуть из стека элемент (без запоминания)
`X ↔ Y`	Обменять местами значения X и Y
`for (X ∈ S)`	Повторять цикл для X, равного каждому элементу множества S (в произвольном порядке)
`(условие)`	1, если условие (например, $X > 0$) истинно, 0 — в противном случае
`A ==> B`	Из условия A следует B

Глава 1

Данные и алгоритмы

Программа состоит из обрабатываемых данных и алгоритма:

Программа = Данные + Алгоритм

Разработка программы включает параллельное проектирование этих двух ее компонентов. На каждом этапе проектирования сначала уточняется структура данных, а затем алгоритм, т. е. операции над данными.

Данные — это информация, представленная в форме, воспринимаемой устройствами ЭВМ.

Под *структурой данных* понимают определенным образом организованную совокупность данных. Структура данных состоит из элементов (*полей*) и определяется правилами, устанавливающими отношения (связи) между элементами. Элемент может состоять из более мелких элементов, т. е. сам может являться структурой данных. Таким образом, возможны иерархические (многоуровневые) структуры данных.

Иногда полем называют наименьший элемент, имеющий определенный смысл. Для этого используют также термины: реквизит, атрибут, терм, признак, скалярный элемент и др.

Пример. Структура данных «адрес человека» включает скалярные элементы: «фамилия», «имя», «отчество» и «домашний адрес», который сам является структурой данных и включает поля: «город», «улица», «дом», «квартира».

Первоначально применение ЭВМ ограничивалось в основном вычислительными задачами, и структуры данных были очень простыми — использовались числа и числовые

массивы. Соответственно, основную трудность в создании программ представляло проектирование алгоритма.

В дальнейшем — сначала при разработке трансляторов и операционных систем, а затем и при создании прикладных программ, особенно систем с элементами искусственного интеллекта и экспертных систем использовались все более сложные структуры данных, и центр тяжести в разработке программ смещался к данным. Особенно наглядно это проявилось с появлением автоматизированных систем, построенных на основе концепции базы данных. *База данных* — это общая структура данных нескольких программ, составляющая основное ядро всей информационной системы.

Проектирование структуры данных является наиболее трудной и ответственной проблемой в создании сложных программ. Выбор структуры данных и способа их представления в памяти ЭВМ во многом (а иногда — почти однозначно) определяет структуру алгоритма. Овладение соответствующими методами стало обязательной частью обучения не только системных, но и прикладных программистов.

В этом пособии рассматриваются методы организации и обработки данных в оперативной памяти ЭВМ. Аналогичные методы лежат в основе проектирования баз данных во внешней памяти.

1.1. Уровни описания данных

В конечном счете все данные в памяти ЭВМ хранятся в виде последовательности битов, но такое детальное представление трудно разработать сразу. Кроме того, его трудно корректировать при изменениях программы. Поэтому при проектировании структур данных, так же как и при разработке алгоритмов, используют основной метод борьбы со сложностью больших систем — иерархию, т. е. разбиение системы на уровни.

Часто, например, используют три уровня описания структуры данных: функциональный, логический и физический.

1. На *функциональном уровне* определяются необходимые для решения задачи операции над структурой данных и их свойства. Получается информационная модель предметной области решаемой задачи.
2. На *логическом уровне* структура данных разбивается на элементы, а операции над ней выражаются через операции над ее элементами.
3. На *физическом уровне* определяется способ представления данных и операций в выбранном языке программирования (т. е. в ЭВМ). Достаточно выразить данные и операции решаемой задачи через базовые структуры и понятия языка программирования. Более детальное представление в машинном языке определяется транслятором автоматически. Программист же имеет дело с абстрактной машиной, «понимающей» язык высокого уровня.

На функциональном и логическом уровнях приходится иметь дело с абстрактными структурами данных, организованными в соответствии с решаемой задачей без детального учета особенностей ЭВМ (или языка программирования).

Абстрактная структура данных — это структура данных, рассматриваемая с точки зрения применяемых к ней операций без описания способа ее представления в памяти. Она близка к понятию абстрактного типа данных.

Абстрактный тип данных — тип данных, определяемый функционально: только через операции над объектами этого типа без описания способа представления их значений.

На физическом уровне имеют дело *со структурами хранения*, т. е. структурами данных, легко реализуемыми в языке программирования. В качестве базовых структур хранения в этом пособии рассматриваются *вектор, список* и *сеть*.

Множество абстрактных структур данных бесконечно, так как для каждой задачи можно придумывать свои структуры. В этом пособии описаны наиболее распространенные в самых разных задачах абстрактные структуры данных: *очередь, стек, дек, строка, граф, дерево, табли-*

ца, массив и *множество*. Они изучаются по единому плану: определение, применение, типовые операции, способы представления данных и реализации операций.

Абстрактную структуру данных можно реализовать разными способами на базе других более простых структур данных. Основными критериями выбора метода реализации являются:

1) возможность реализации необходимых для задачи операций над данными;
2) время выполнения операций;
3) требуемая память;
4) простота программирования.

Относительная важность этих критериев зависит от конкретной ситуации. Указанные критерии часто противоречат друг другу, и разработчику приходится искать разумный компромисс между ними.

1.2. Методы хранения данных

Возможные методы хранения данных определяются организацией памяти ЭВМ.

В большинстве современных ЭВМ оперативная память построена по адресному принципу и представляет собой пронумерованную последовательность ячеек одинакового размера. При этом номер ячейки называется ее *адресом*, а содержимое ячейки — *машинным словом*. Количество ячеек (объем, емкость памяти) обычно находится в пределах от 10^4 до 10^{10}, а размер (длина) ячейки — от 8 до 64 бит.

Адресом данных, занимающих несколько соседних ячеек, считают адрес первой из них. В языках программирования адреса называют также *указателями* или *ссылками*.

Минимальным элементом хранения является *бит* — двоичная цифра, принимающая одно из двух значений, например, 0 или 1. Бит используется также и как единица количества информации. Количество информации в битах

равно минимальному количеству двоичных цифр, необходимому для представления этой информации.

Машинное слово может представлять собой команду или данные. Первоначально ЭВМ использовались преимущественно для обработки числовой информации. Ячейка памяти при этом содержала одно число и имела длину от 16 до 64 бит. Это неудобно для представления символьной (текстовой) информации, так как код символа (*байт*) в зависимости от размера алфавита содержал от 6 до 8 бит, и в ячейке приходилось размещать несколько символов, что затрудняло доступ к каждому из них.

С повышением роли обработки символьной информации стали применять байтовую организацию памяти, когда ячейка содержит один байт, равный 8 битам, а для представления числа используются несколько соседних байтов. Байт также рассматривается как единица количества информации, равная одному символу текста.

Таким образом, оперативная память ЭВМ имеет линейную (одномерную) организацию, и для хранения многомерных массивов и других сложных структур данных их необходимо «линеаризовать».

Существуют три основные группы методов хранения структур данных.

1. **Последовательное (сплошное) представление данных.** Элементы структуры располагаются в памяти друг за другом без промежутков. Наиболее часто используемой структурой хранения является *вектор*.
2. **Связанное (цепное) представление данных**. Элементы структуры могут размещаться в памяти в произвольном порядке — не обязательно подряд, причем каждый элемент содержит указатели (адреса) одного или нескольких других элементов, позволяющие отыскивать их в памяти. Основные структуры хранения — *список* и *сеть*.
3. **Адресная арифметика**. Элементы структуры располагаются в памяти в произвольном порядке с возможными промежутками, но существует определенная зако-

номерность, позволяющая вычислять адрес элемента, например, в зависимости от его номера или другого параметра. Адресная арифметика может рассматриваться как обобщение последовательного и связанного представлений и в общем виде используется сравнительно редко.

Вектор — это конечная последовательность элементов одинакового размера, расположенных в памяти подряд (без промежутков). Вектор определяется *базой* (адресом первого элемента), *длиной* (количеством элементов) и *размером элемента*. По сути дела, вектор представляет собой одномерный массив. Главное достоинство вектора — *прямой доступ* к элементу по его порядковому номеру — *индексу*:

```
адрес(V[J]) = адрес(V[0])+d*J =
            = адрес(V[1])+d*(J-1),        (1.1)
```

где $V[J]$ — элемент с индексом J, d — размер элемента (количество ячеек, занимаемых одним элементом); считаем, что $d \geqslant 1$, целое (случай $d < 1$ рассмотрен в разделе 3.3.3).

Элементы вектора одинаково доступны, и их можно обрабатывать в любом порядке.

Связанный список (далее — просто *список*) — это последовательность элементов, каждый из которых кроме других данных содержит указатель (адрес) следующего элемента. На рисунках указатель изображают в виде стрелки, соединяющей элементы списка. На рис. 1.1 показан список, каждый элемент которого содержит один символ. Последний элемент списка содержит *пустой указатель* («адрес» несуществующего объекта), показанный на рис. 1.1 в виде поля, перечеркнутого крестиком (X).

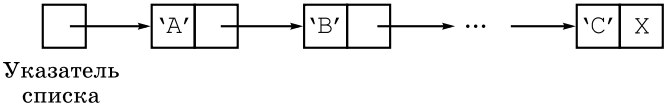

Рис. 1.1. Список