



§ 9

Структурное программирование

9.1. Общее представление о структурном программировании

Программирование как род занятий и сфера деятельности интенсивно развивается со второй половины прошлого века. За это время сложились определённые технологии, способствующие повышению производительности труда программистов, в том числе сокращению числа ошибок, упрощению отладки, модификации и сопровождения программного обеспечения. Особенно это важно при разработке больших и сложных программных комплексов, осуществляемой усилиями целых коллективов программистов.

Одна из таких технологий — структурное программирование — была разработана ещё в начале 70-х гг. прошлого века и связана с именем выдающегося нидерландского учёного Эдсгера Дейкстры (1930–2002).

Структурное программирование — технология разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры логически целостных фрагментов (блоков).



Перечислим некоторые принципы структурного программирования.

1. Любая программа строится из трёх базовых управляющих конструкций: последовательности, ветвления, цикла.
2. В программе базовые управляющие конструкции могут быть вложены друг в друга произвольным образом.
3. Повторяющиеся фрагменты программы можно оформить в виде подпрограмм (процедур и функций). В виде подпрограмм можно оформить логически целостные фрагменты программы, даже если они не повторяются.
4. Все перечисленные конструкции должны иметь один вход и один выход.
5. Разработка программы ведётся пошагово, методом «сверху вниз».

О методе разработки алгоритма «сверху вниз» вы получили представление в курсе информатики основной школы. Напомним его ключевые моменты на примере разработки некоторой программы.

Сначала пишется короткий текст основной программы. В ней вместо каждого логически целостного фрагмента вставляется вызов подпрограммы, которая будет выполнять этот фрагмент. Вместо настоящих, работающих подпрограмм в программу вставляются так называемые заглушки. Как правило, они удовлетворяют требованиям интерфейса заменяемого фрагмента, но не выполняют его функций.

На следующем шаге следует убедиться, что подпрограммы вызываются в правильной последовательности, т. е. верна общая структура программы.

После этого подпрограммы-заглушки последовательно заменяются на полнофункциональные, причём разработка каждой подпрограммы ведётся тем же методом, что и разработка основной программы. На каждом этапе проверяется, что уже созданная программа правильно работает по отношению к подпрограммам более низкого уровня.

Разработка заканчивается тогда, когда ни на одном уровне не останется ни одной заглушки. Полученная программа проверяется и отлаживается.

Такая последовательность гарантирует, что на каждом этапе разработки программист будет иметь дело с обозримым и понятным ему множеством фрагментов, осознавая, что общая структура всех более высоких уровней программы верна.

9.2. Вспомогательный алгоритм

Пример 1. Применим метод «сверху вниз» для разработки алгоритма нахождения периметра треугольника, заданного координатами своих вершин.

Пусть $X_A, X_B, Y_A, Y_B, X_C, Y_C$ — координаты вершин треугольника ABC . Его периметр — сумма длин отрезков AB , BC и AC .

Из курса геометрии вам известна формула для вычисления длины отрезка AB по координатам его концов (рис. 2.11):

$$d = \sqrt{(X_A - X_B)^2 + (Y_A - Y_B)^2}.$$

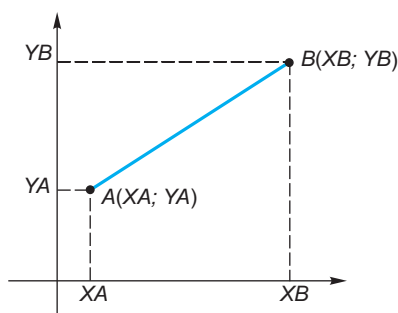


Рис. 2.11. Отрезок AB

Действия по вычислению длины отрезка представляют собой логически целостный фрагмент, который целесообразно оформить в виде вспомогательного алгоритма.

Вспомогательный алгоритм — это алгоритм, целиком используемый в составе другого алгоритма.

На рис. 2.12 представлены:

- 1) блок-схема алгоритма вычисления периметра треугольника, предполагающая вызов вспомогательного алгоритма Отрезок;
- 2) блок-схема вспомогательного алгоритма Отрезок.

При вызове вспомогательного алгоритма указываются его параметры (входные данные и результаты). Параметрами вспомогательного алгоритма *Отрезок* являются величины $X1, Y1, X2, Y2, D$. Это формальные параметры, они используются при описании алгоритма. При конкретном обращении к вспомогательному алгоритму формальные параметры заменяются фактическими параметрами, т. е. именно теми величинами, для которых будет исполнен вспомогательный алгоритм. Типы, количество и порядок следования формальных и фактических параметров должны совпадать.

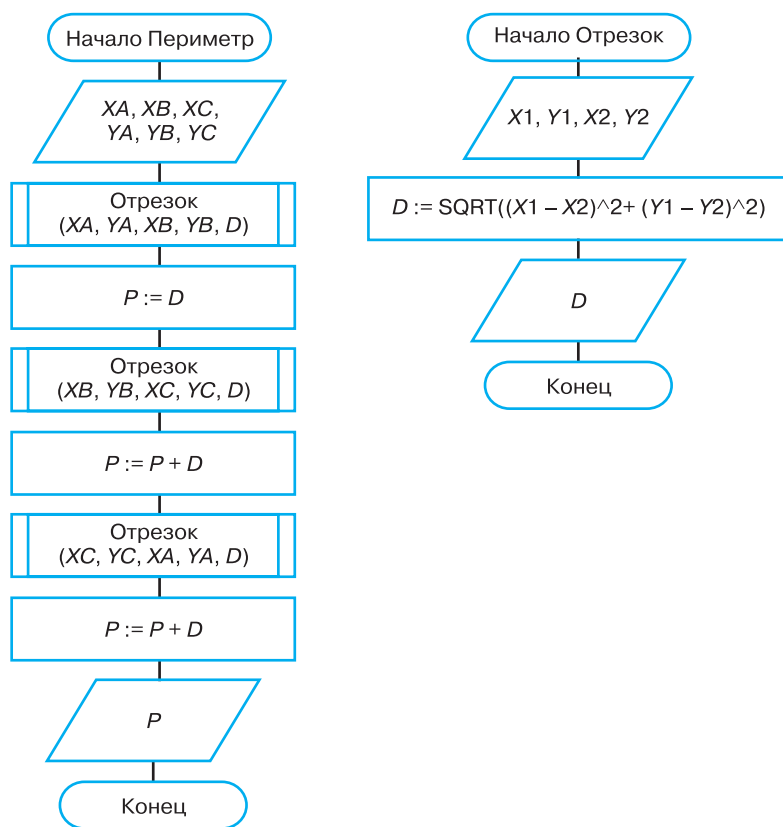


Рис. 2.12. Алгоритм вычисления периметра треугольника и вспомогательный алгоритм *Отрезок*

Команда вызова вспомогательного алгоритма выполняется следующим образом:

- 1) формальные входные данные вспомогательного алгоритма заменяются значениями фактических входных данных, указанных в команде вызова вспомогательного алгоритма;
- 2) для заданных входных данных исполняются команды вспомогательного алгоритма;
- 3) полученные результаты присваиваются переменным с именами фактических результатов;
- 4) осуществляется переход к следующей команде основного алгоритма.

Каким будет результат работы алгоритма при следующих входных данных: $XA = 1$, $XB = 2$, $XC = 3$, $YA = 1$, $YB = 3$, $YC = 1$?



9.3. Рекурсивные алгоритмы

Алгоритм называется **рекурсивным**, если на каком-либо шаге он прямо или косвенно обращается сам к себе.



Пример 2. Как известно, факториал натурального числа n определяется следующим образом: $n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n$; $0!$ считается равным единице ($0! = 1$).

Иначе это можно записать так:

$$F(n) = 1 \text{ при } n \leq 1;$$

$$F(n) = F(n - 1) \cdot n \text{ при } n > 1.$$

В определении факториала через рекурсию имеется условие $n \leq 1$, при достижении которого вызов рекурсии прекращается.

В рекурсивном определении должно присутствовать ограничение (**граничное условие**), при выходе на которое дальнейшая инициация рекурсивных обращений прекращается.



Пример 3. Определим функцию $S(n)$, вычисляющую сумму цифр в заданном натуральном числе n :

$$S(n) = n \text{ при } n < 10;$$

$$S(n) = S(n \text{ div } 10) + n \text{ mod } 10 \text{ при } n \geq 10.$$





Самостоятельно определите функцию $K(n)$, которая возвращает количество цифр заданного натурального числа n .



Пример 4. Алгоритм вычисления значения функции $F(n)$, где n — натуральное число, задан следующими соотношениями:

$$F(n) = 1 \text{ при } n \leq 2;$$

$$F(n) = F(n-1) + 3 \cdot F(n-2) \text{ при } n > 2.$$

Требуется выяснить, чему равно значение функции $F(7)$.

По условию, $F(1) = F(2) = 1$.

$$F(3) = F(2) + 3 \cdot F(1) = 1 + 3 \cdot 1 = 4.$$

$$F(4) = F(3) + 3 \cdot F(2) = 4 + 3 \cdot 1 = 7.$$

$$F(5) = F(4) + 3 \cdot F(3) = 7 + 3 \cdot 4 = 19.$$

$$F(6) = F(5) + 3 \cdot F(4) = 19 + 3 \cdot 7 = 40.$$

$$F(7) = F(6) + 3 \cdot F(5) = 40 + 3 \cdot 19 = 97.$$

Подобные вычисления можно проводить в уме, а их результаты фиксировать в таблице:

n	1	2	3	4	5	6	7
$F(n)$	1	1	4	7	19	40	97



Пример 5. Исполнитель Плюс имеет следующую систему команд:

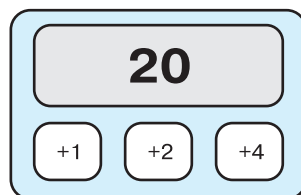
1. прибавь 1
2. прибавь 2
3. прибавь 4

С помощью первой из них исполнитель увеличивает число на экране на 1, с помощью второй — на 2, с помощью третьей — на 4. Программа для исполнителя Плюс — это последовательность команд. Выясним, сколько разных программ, преобразующих число 20 в число 30, можно составить для этого исполнителя.

Количество программ, с помощью которых можно получить некоторое число n , будем рассматривать как функцию $K(n)$.

Число, меньшее 20, при заданных начальных условиях и системе команд исполнителя Плюс получить невозможно. Следовательно, при $n < 20$ $K(n) = 0$.

Для начального числа 20 количество программ равно 1: существует только одна пустая программа, не содержащая ни одной команды. Можем записать: $K(n) = 1$ при $n = 20$.



Любое число $n > 20$ может быть получено из чисел $n - 1$, $n - 2$ и $n - 4$ одной из трёх команд, входящих в систему команд исполнителя: прибавь 1, прибавь 2 и прибавь 4 соответственно. При этом каждая программа получения из исходного числа чисел $n - 1$, $n - 2$ и $n - 4$ удлинится на одну команду и будет приводить к числу n . Следовательно:

$$K(n) = K(n - 1) + K(n - 2) + K(n - 4).$$

Запишем все соотношения, определяющие функцию $K(n)$:

$$K(n) = 0 \text{ при } n < 20;$$

$$K(n) = 1 \text{ при } n = 20;$$

$$K(n) = K(n - 1) + K(n - 2) + K(n - 4) \text{ при } n > 20.$$

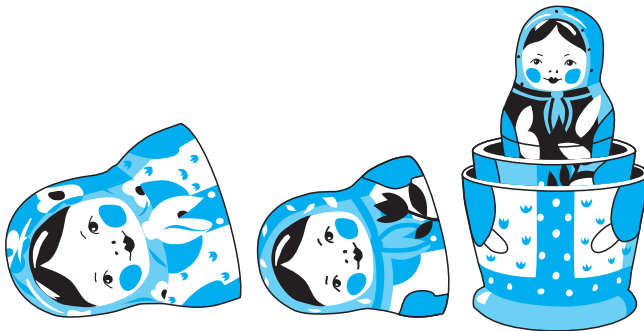
Заполним по этой формуле таблицу для всех значений n от 20 до 30:

n	20	21	22	23	24	25	26	27	28	29	30
$K(n)$	1	1	2	3	6	10	18	31	55	96	169

Итак, существует 169 различных программ, с помощью которых исполнитель Плюс может преобразовать число 20 в 30.

Любой объект, который частично определяется через самого себя, называется рекурсивным. Нас окружает множество рекурсивных объектов. Приведём примеры только некоторых из них.

1. Матрёшка — русская деревянная игрушка в виде расписной куклы, внутри которой находятся подобные ей куклы меньшего размера.



2. Два зеркала, поставленные друг напротив друга, — в них образуются два коридора из затухающих отражений. Это, например, можно наблюдать в спальном железнодорожном вагоне.



3. Примером рекурсивной структуры является замечательное стихотворение Р. Бернса «Дом, который построил Джек» в переводе С. Маршака.
4. Рекурсивную природу имеют геометрические фракталы. На рисунке представлено построение одного из геометрических фракталов — треугольника Серпинского. Чтобы его получить, нужно взять равносторонний треугольник с внутренней областью, провести в нём средние линии и «выкинуть» центральный из четырёх образовавшихся маленьких треугольников. Далее эти же действия нужно повторить с каждым из оставшихся трёх треугольников и т. д.



9.4. Запись вспомогательных алгоритмов на языке программирования

Запись вспомогательных алгоритмов в языках программирования осуществляется с помощью подпрограмм. В языках Pascal и Python различают два вида подпрограмм: процедуры и функции.

Описание **процедуры** на языке Pascal имеет вид:

```
procedure <имя_процедуры> (<описание параметров-значений>;  
    var: <описание параметров-переменных>);  
begin  
    <операторы>  
end;
```

В заголовке процедуры после её имени приводится перечень формальных параметров и их типов. Для вызова процедуры достаточно указать её имя со списком фактических параметров. При этом между фактическими и формальными параметрами должно быть полное соответствие по количеству, порядку следования и типу.

Описание процедуры на языке Python имеет вид:

```
def <имя процедуры> () :
    <операторы>
```

Процедура начинается со служебного слова **def** (от англ. *define* — определить). После этого записывается имя процедуры, скобки и двоеточие. Операторы, которые входят в тело процедуры, записываются с отступом. Так мы показываем, какие команды входят в процедуру.

Пример 6. Запишем на двух языках программирования программу нахождения периметра треугольника, заданного координатами его вершин. Вспомогательный алгоритм оформим с помощью процедуры.



Программа на языке Pascal

```
program perimetr;
var xa, ya, xb, yb, xc, yc, d,p: real;
procedure otrezok(x1, y1, x2, y2: real; var d: real);
begin
    d := sqrt(sqr(x1 - x2) + sqr(y1 - y2)); end;
begin
    p := 0;
    writeln('Ввод координат концов отрезка:');
    writeln('xa, ya');
    read(xa, ya);
    writeln('xb, yb');
    read(xb, yb);
    writeln('xc, yc');
    read(xc, yc);
    otrezok(xa, ya, xb, yb, d);
    p := p + d;
    otrezok(xa, ya, xc, yc, d);
    p := p + d;
    otrezok(xc, yc, xb, yb, d);
    p := p + d;
    writeln('p=', p)
end.
```

Программа на языке Python

```

def otrezok(x1, y1, x2, y2):
    global d
    d = ((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 0.5
p = 0
print('Ввод координат концов отрезка:')
print('xa, ya')
xa, ya = map(float, input().split())
print('Ввод координат концов отрезка:')
print('xb, yb')
xb, yb = map(float, input().split())
print('Ввод координат концов отрезка:')
print('xc, yc')
xc, yc = map(float, input().split())
otrezok(xa, ya, xb, yb)
p += d
otrezok(xa, ya, xc, yc)
p += d
otrezok(xc, yc, xb, yb)
p += d
print('p =', p)

```



Выполните программу на компьютере.

Подумайте, каким образом можно модифицировать программу, чтобы вычислять с её помощью периметр n -угольника. Каким образом при решении этой задачи можно использовать массивы?



Функция — подпрограмма, имеющая единственный результат, записываемый в ячейку памяти, имя которой совпадает с именем функции.

Описание функции на языке Pascal имеет вид:

```

function <имя_функции> (<описание входных данных>):
    <тип_функции>;
begin
    <операторы>
end;

```

В заголовке функции после её имени приводится описание входных данных — указывается перечень формальных параметров и их типов. Там же указывается тип самой функции, т. е. тип результата. В блоке функции обязательно должен присутствовать оператор

```
<имя_функции> := <результат>;
```

Для вызова функции достаточно указать её имя со списком фактических параметров в любом выражении, в условиях (после слов **if**, **while**, **until**) или в операторе **write** главной программы.

Описание функции на языке Python имеет вид:

```
def <имя функции>():
    <операторы>
    return <результат>
```

Функция начинается со служебного слова **def**. После этого записывается имя функции, скобки, в которых может быть перечень формальных параметров, и двоеточие. Операторы, которые входят в тело функции, записываются с отступом. После оператора **return** записывается результат, который возвращает функция.

Пример 7. Запишем программу нахождения периметра треугольника, заданного координатами его вершин. Вспомогательный алгоритм оформим с помощью функции.



Программа на языке Pascal

```
program perimetr;
  var xa, ya, xb, yb, xc, yc, p: real;
function d(x1, y1, x2, y2: real): real;
begin
  d := sqrt(sqr(x1 - x2) + sqr(y1 - y2));
end;
begin
  writeln('Ввод координат концов отрезка:');
  writeln('xa, ya');
  read(xa, ya);
  writeln('xb, yb');
  read(xb, yb);
  writeln('xc, yc');
  read(xc, yc);
  p := p + d(xa, ya, xb, yb) + d(xa, ya, xc, yc) + d(xc, yc, xb, yb);
  writeln('p=', p)
end.
```

Программа на языке Python

```
def otr(x1, y1, x2, y2):  
    return((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 0.5  
print('Ввод координат концов отрезка:')  
print('xa, ya')  
xa, ya = map(float, input().split())  
print('xb, yb')  
xb, yb = map(float, input().split())  
print('xc, yc')  
xc, yc = map(float, input().split())  
print('p=', otr(xa, ya, xb, yb) + otr(xa, ya, xc, yc) + otr(xc, yc, xb, yb))
```



Выполните программу на компьютере.

На основе этой программы напишите функцию, вычисляющую площадь треугольника по целочисленным координатам его вершин. Используйте эту функцию для вычисления площади n -угольника.

САМОЕ ГЛАВНОЕ

Структурное программирование — технология разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры логически целостных фрагментов (блоков).

Основные принципы структурного программирования заключаются в том, что:

- 1) любая программа строится из трёх базовых управляющих конструкций: последовательности, ветвления, цикла;
- 2) в программе базовые управляющие конструкции могут быть вложены друг в друга произвольным образом;
- 3) повторяющиеся фрагменты программы можно оформить в виде подпрограмм (процедур и функций). В виде подпрограмм можно оформить логически целостные фрагменты программы, даже если они не повторяются;
- 4) все перечисленные конструкции должны иметь один вход и один выход;
- 5) разработка программы ведётся пошагово, методом «сверху вниз».

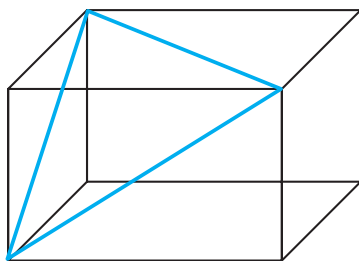
Вспомогательный алгоритм — это алгоритм, целиком используемый в составе другого алгоритма.

Алгоритм называется рекурсивным, если на каком-либо шаге он прямо или косвенно обращается сам к себе.

Запись вспомогательных алгоритмов в языках программирования осуществляется с помощью подпрограмм. В языках Pascal и Python различают два вида подпрограмм: процедуры и функции.

Вопросы и задания

1. В чём заключается сущность структурного программирования? Какие преимущества обеспечивает эта технология?
2. Какой алгоритм называется вспомогательным?
3. Вспомните, в чём состоит суть метода последовательного построения (уточнения) алгоритма. Как этот метод называется иначе?
4. Опишите основные шаги разработки программы методом «сверху вниз».
5. Дан прямоугольный параллелепипед, длины рёбер которого равны a , b и c . Требуется определить периметр треугольника, образованного диагоналями его граней. Какой алгоритм целесообразно использовать при решении этой задачи в качестве вспомогательного?



6. Какой вспомогательный алгоритм называется рекурсивным? Что такое граничное условие и каково его назначение в рекурсивном алгоритме?

7. Алгоритм вычисления значения функции $F(n)$, где n — натуральное число, задан следующими соотношениями:

$$F(n) = 2 \text{ при } n \leq 0;$$

$$F(n) = F(n - 2) + F(n - 1) + F(n \operatorname{div} 2) \text{ при } n > 0.$$

Требуется выяснить, чему равно значение функции $F(10)$.

8. Исполнитель Калькулятор имеет следующую систему команд:

1. прибавь 1
2. умножь на 2

С помощью первой из них исполнитель увеличивает число на экране на 1, с помощью второй — в 2 раза.

- 1) Выясните, сколько разных программ, преобразующих число 1 в число 20, можно составить для этого исполнителя.
- 2) Сколько среди них таких программ, у которых в качестве промежуточного результата обязательно получается число 15?
- 3) Сколько среди них таких программ, у которых в качестве промежуточного результата никогда не получается число 12?

9. Попробуйте найти рекурсивные синтаксические структуры:

- а) в поэме А. Блока «Двенадцать»;
- б) в стихотворении М. Лермонтова «Сон»;
- в) в романе М. Булгакова «Мастер и Маргарита»;
- г) в фольклоре.

10. Найдите информацию о таких геометрических фракталах, как снежинка Коха, Т-квадрат, Н-фрактал, кривая Леви, драконова ломаная.

11. Напишите программу вычисления значения функции $F(n)$, рассмотренной в примере 4 этого параграфа. Вычислите с её помощью значение функции $F(7)$.

12. Напишите программу вычисления $C_n^k = \frac{n!}{(n-k)! \cdot k!}$. Используйте подпрограмму.

13. Дана программа на двух языках программирования:

Программа на языке Pascal	Программа на языке Python
<pre> program rek; procedure F(n: integer); begin if n > 0 then begin F(n-4); writeln(n); F(n div 3) end; end; begin F(9) end. </pre>	<pre> def F(n) : if n > 0: F(n-4) print(n) F(n//3) F(9) </pre>

Не выполняя программу на компьютере, выясните, что получится в результате её работы.

Проверьте свой результат, выполнив программу на компьютере.

Дополнительные материалы к главе смотрите в авторской мастерской.

