6. Структура управления с условием.

При описании алгоритмов мы перечислили основные ключевые структуры управления, это были: присвоение, последовательность, условное выполнение или выбор и повтор.

При конструировании программ удобнее исходить из дизайна программ/алгоритмов по принципу сверху-вниз. Это означает следующее: сначала надо провести анализ решения в терминах основных составляющих компонентов, выразить решение в виде их последовательности, каждый компонент, в свою очередь, расшифровать и расширить аналогичным образом.

Проиллюстрируем эту идею на примере задачи вычисления окружности и площади круга, когда его радиус задается пользователем. Исходный алгоритм можно представить в виде пронумерованных шагов:

1. Ввести радиус круга;
2. Вычислить окружность и площадь;
3. Вывести значения окружности и площади круга.

Здесь мы использовали идею «последовательности», т.е. все операции будут выполняться строго по порядку, в каком они перечислены. Приведенный алгоритм задает только общую структуру решения проблемы. Теперь каждый шаг должен быть уточнен и расширен.

1. Ввести радиус круга;
   1.1. Запросить пользователя ввести радиус radius;
   1.2. Считать радиус;
2. Вычисления:
   2.1. Вычислить окружность как — $2 \cdot \pi \cdot radius$;
   2.2. Вычислить площадь как — $\pi \cdot radius^2$;
3. Вывод:
   3.1. Вывести значение окружности;
   3.2. Вывести значение площади.

На этом этапе проблема полностью решена независимо от выбора языка программирования. Теперь не составит труда переписать данный алгоритм на любой язык программирования. (Программный код данного алгоритма был приведен в начале четвертой главы в качестве демонстрационного примера.)

К сожалению, не все проблемы решаются так легко. Часто использования только концепта «последовательности» недостаточно для описания решения многих задач.

Рассмотрим задачу решения линейного уравнения:

Написать программу, которой передаются коэффициенты линейного уравнения: ax+b=0. Программа должна выдавать его решение.

На исходной стадии алгоритм решения выглядит так:

1. Ввести коэффициенты линейного уравнения a и b;
2. Вычислить решение: x=-b/a;
3. Вывести решение.

Однако при расширении второго шага недостаточно использования только принципа «последовательности». Эта проблема возникает из-за того, что если коэффициент при неизвестном равен нулю, нельзя использовать общую формулу и этот случай надо выделить как специальный.

Таким образом, в зависимости от истинности логического условия «коэффициент при неизвестном равен нулю», будет либо решаться линейное уравнение, либо выводиться соответствующее сообщение. Мы уже сталкивались с таким условным утверждением при рассмотрении алгоритмов. Он проверял, выполняется или нет определенное условие, и в зависимости от результата проверки, выполнял либо одну операцию, либо другую.

Используя условное утверждение, алгоритмическое решение можно расширить следующим образом:

1.Ввод:

1.1. Запрос — ввести коэффициент *a*;

1.2. Считать величину *a*;
1.3. Запрос — ввести коэффициент *b;*
1.4. Считать величину *b;*
2. Решение уравнения:
2.1. *Если* коэффициент при неизвестном равен нулю
Тогда
2.1.1. Вычислить решение по формуле: *x=-b/a;*
Иначе
{
2.1.2. Вывести сообщение;
2.1.3. Завершить программу;
}
3. Вывести значение вычисленной неизвестной величины.

Здесь важно отметить использование *условной структуры,* которая формируется словами:

Если «условие» *Тогда* A *Иначе* B

и выполняет инструкции, определенные в «А», если «условие» истинно, и выполняется «В», если «условие» ложно.

Почти во всех языках программирования эта структура присутствует в какой-либо форме (обычно If... Then ... Else ...). Его формат на Турбо-Паскале описывается ниже, но предварительно рассмотрим правила формирования логических выражений и условий.

6.1. Условие и логические выражения.

6.1.1. Выражения отношений.

«*Условие*» — это такое *логическое выражение,* которое может принимать только два значения: «истинно» (True) или «ложно» (False). Самое простое логическое выражение – это *выражение отношения:*

x < y

которое истинно, если значение переменной *x* меньше значения переменной *y*. Общая форма выражения отношения имеет вид:

Операнд_1 Оператор_отношения *Операнд_2*

Операнды могут быть переменными, константами или арифметическими выражениями. Если операнд – выражение, то сначала вычисляется его текущее значение и затем это значение берется в качестве операнда.

В Турбо-Паскале определены следующие *операторы отношений:*

< — Меньше чем
> — Больше чем
<= — Меньше чем или равняется
>= — Больше чем или равняется
= — равно
<> — не равно

Заметьте, что отношение равенства «=» отличается от знака присвоения «:=».

Условие принимается истинным, если операнды удовлетворяют оператору отношения, а иначе — оно ложно.

6.1.2. Примеры использования операторов отношения.

```
i < 10
sum >= 1000.0
count <> n
discriminant < 0
x * x + y * y <= r*r
```

Очевидно, что в зависимости от текущих значений, эти логические выражения могут быть истинными или ложными. Например, при

```
x=2
y=5
```

r = 10

последнее выражение истинно, тогда как при

x=8
y=9
r = 10

то же самое выражение ложно.

Генерируемые в Турбо-Паскале значения логических выражений принимают значения *True* и *False*.

6.1.3. Логические выражения.

Можно составить более сложные условия, которые являются комбинацией операторов отношений. Составные логические выражения формируются с помощью *логических операторов*, т.н. *логических связок:*

Not – Не
And — Да
Or – Или
Xor Исключающее Или

Логические связки приведены по порядку приоритета их выполнения.

Только оператор «Not» является унарным, т.е. воздействует только на один операнд, остальные операторы — бинарные, т.е. соединяют два операнда. При формировании составных логических выражений в Турбо-Паскале простые выражения—операнды заключаются в круглые скобки.

Логическая связка «Not» означает отрицание «НЕТ». Она применяется к одному операнду и возвращает значение «ложно», если значение операнда истинно, и «истина», если значение операнда ложно.

Оператор «И» («And») соединяет два выражения. Возвращает значение «истина», только если оба операнда истинны. Если оба или хотя бы один из операндов имеет значение «ложно», то и значение целого выражения будет «ложно».

Оператор «ИЛИ» («Or») также бинарный. Возвращает значение «истина», если хотя бы один из операндов имеет значение «истина». Значение «ложно» генерируется, только если оба операнда имеют значение «ложно».

Бинарный оператор «исключающее ИЛИ» («Xor») возвращает значение «истина», только если операнды принимают противоположные значения, т.е. если одно составляющее выражение истинно, второе должно быть ложным, и наоборот. В противном случае полное выражение будет ложным.

При использовании нескольких логических операторов порядок их выполнения (если не используются дополнительные скобки) определяется приоритетом каждой логической связки. Например,

(5>=3) Or (3<0) And Not (5=10).

Здесь сначала выполнится оператор «Not», затем «And» и в конце «Or».

Для наглядности определения значений составных логических выражений используется т.н. *таблица истинности.* Пусть «A» и «B» — два простых логических выражения. Тогда таблица истинности будет иметь вид (таблица 6.1):

Таблица 6.1.

A	B	Not A	A And B	A Or B	AXorB
True	True	False	True	True	False
True	False	False	False	True	True
False	True	True	False	True	True
False	False	True	False	False	False

Таким образом, с помощью этой таблицы можно вычислять значения более сложных логических выражений.

Например, если i=15, а j=10, тогда для вычисления выражения

(i>10) and (j>0)

сначала надо найти значение выражения (i>10) (которое истинно), затем выражения (j>0) (которое также истинно), и, наконец, по таблице истинности — значение полного выражения, которое в данном случае является истинным. Если бы мы имели j=-1, тогда второе выражение было бы ложным, и, следовательно, полное выражение также было бы ложным. Если бы i=5, тогда полное выражение было бы ложным вне зависимости от второго выражения и значения переменной *j*. В таких случаях второе выражение в Турбо-Паскале даже не вычисляется, точно так же, как при вычислении значения выражения со связкой «Or» игнорируется второе выражение, если первое истинно. Такой подход т.н. укороченной оценки делает более эффективным вычисление значений сложных логических выражений.

Заметим еще раз, что при записи сложных логических выражений в Турбо-Паскале, простые выражения должны быть заключены в круглые скобки.

6.1.4. Примеры использования логических связок.

```
(i<10) And (j>0)
((x+y)<=15) Or (i==5)
Not((i>= 10) 0r (j<=0))
(i< 10) And False
```

Заметим, что последнее выражение всегда будет ложным независимо от значения переменной *i*, так как выражение *False* всегда ложно.

В этих примерах круглые скобки использовались для определения порядка выполнения операторов. Однако приоритеты выполнения арифметических и логических операторов строго предопределены. Полная таблица приоритетов сверху-вниз приводится в табл.6.2. Операторы с наивысшим приоритетом выполняются в первую очередь, а дальше — по убыванию приоритета.

Таблица 6.2.

Таблица приоритетов операторов	
( )	Скобки
Not	Логическое «НЕТ»
Div Mod	Целочисленное деление, остаток
* /	Умножение, деление
+ -	Сложение, вычитание
< <= > >=	Меньше чем, меньше чем или равняется, больше чем, больше чем или равняется
=<>	Равно, не равно
And	Логическое «И»
Or Xor	Логическое «ИЛИ», «исключающее ИЛИ»
:=	Оператор присвоения

При записи будьте осторожны и не перепутайте оператор присвоения «:=» с логическим оператором равенства «=».

Рассмотрим еще несколько примеров. Пусть x и y – действительные переменные, и пусть они обозначают пространственные координаты. Тогда с помощью составных логических выражений можно задавать различные пространственные области на плоскости, например:

```
Not (x<=0)                          : правая полуплоскость
(x>=0) And (y>=0)                   : первый квадрант
(x>=0) Or (y>=0)                    : I, II, и IV квадранты
(x>=0) Xor (y>=0)                   : II и IV квадранты
(y=0) And (x>=0) And (x<=1)         : Единичный интервал [0,1 ]
```

В Турбо-Паскале определена логическая функция

```
Odd(x:LongInt):Boolean;
```

которая используется для определения четности числа. Если аргумент функции – нечетное число, то значение будет «истинно», а если четное — «ложно». В описании функции использовались два

новых типа, а именно *LongInt* – сокращенное от Long Integer, «длинное целое число», разрешающий более широкий диапазон значений для целых чисел, чем тип Integer, и *Boolean* – логический тип с диапазоном значений *True* и *False*. Следовательно, в Турбо-Паскале вполне допустимо следующее объявление и присвоения:

```
Var boo, logvar : Boolean;
...
boo:= True;
logvar:=5>10;
```

Здесь значение переменной *logvar* будет *False*.

Мы описали принципы формирования логических выражений (условий). Сейчас уже можно рассмотреть *условные структуры* управления в Турбо-Паскале.

6.2. Оператор «If».

Как уже было отмечено, для реализации алгоритмов нужно иметь механизм, позволяющий выполнять по выбору тот или иной набор операторов, в зависимости от определенного условия. Оператор If является самым простым, т.н. *условным оператором* или *оператором выбора*. Рассмотрим следующий фрагмент:

```
If(x > 0) Then Writeln(' X - положительная величина!');
```

Если текущее значение переменной *x* положительно, и, следовательно, логическое условие *(x > 0)* истинно, выполнится оператор «Writeln» и на экране появится сообщение «X– положительная величина!».

Общий формат оператора If имеет вид:

```
If (Условие) Then Оператор;
```

где (Условие) — это допустимое логическое условие, а после «Then» стоит один оператор, простой или составной, т.е. если после «Then» по контексту выполняется несколько операторов, соответствующий

фрагмент должен быть заключен в программные скобки «Begin End;»:

```
If (Условие) Then
Begin
    Оператор — 1;
    Оператор — 2;
    …
    Оператор—n;
End;
```

Как все выполняемые операторы в Турбо-Паскале, оператор If тоже должен завершиться символом – «точка с запятой». Если условие – простое логическое выражение, круглые скобки можно опустить, но они обязательно присутствуют в составных логических выражениях согласно правилам их формирования.

6.2.1. Примеры использования оператора «If».

Следующий оператор увеличивает значение переменной *sum* на величину *x*, если *x* положительна:

```
If (x > 0.0) Then sum := sum + x;
```

Следующий фрагмент выполняет то же самое, но вдобавок оператор увеличивает значение переменной *count* на единицу (подсчет положительных чисел):

```
If x > 0 Then
Begin
    sum := sum + x;
    count := count + 1;
End;
```

Здесь в качестве тела условного оператора используется составной оператор. Что бы произошло, если бы мы опустили программные скобки, т.е.:

```
If x>0 Then sum := sum + x;
count := count + 1;
```

В этом случае, если *x* больше нуля, то выполнился бы следующий оператор и значение переменной *sum* увеличилось бы на *x*. Однако оператор «count := count + 1;» воспринимался бы как следующая инструкция программы, а не составная часть оператора If, и выполнился бы в любом случае вне зависимости от значения переменной «x». Разница выявится только в случае, если *x* отрицательна.

Составной оператор может содержать любой дозволенный оператор Турбо-Паскаля, в том числе и другой оператор If.

6.3. Оператор «If-Else».

Оператор If позволяет сделать простой выбор: выполнить или нет последующий оператор, если условие истинно. В Турбо-Паскале поддерживается еще один вариант оператора If, который позволяет сделать двойной выбор: если условие истинно – выполнить один оператор, а если ложно – другой. Эту функцию могут выполнить два оператора If со взаимно исключающими условиями, например,

```
If disc>=0 Then Writeln('Корни действительные!');
If disc<0 Then Writeln('Корни комплексные!');
```

Но здесь можно использовать другую, расширенную версию «If», т.н. «If-Else» формат, в результате фрагмент запишется в виде:

```
If disc>=0 Then Writeln('Корни действительные!')
           Else Writeln('Корни комплексные!');
```

Общий формат «If-Else» оператора имеет вид:

```
If (Условие) Then Оператор — 1
             Else Оператор — 2;
```

Если Условие истинно, выполняется Оператор— 1, который может быть либо простым, либо составным, а если Условие ложно, выполняется Оператор—2. Символ «;» ставится в конце после Оператор—2, а перед «Else» точка с запятой не ставится, так как она будет восприниматься как конец простого оператора «If» без «Else».

Пример оператора «If-Else».

В следующем фрагменте обрабатывается значение переменной *x:* если *x* не отрицательна, ее значение прибавляется переменной *sum_pos,* которая представляет сумму неотрицательных чисел, а значение счетчика *count_pos* неотрицательных чисел увеличивается на единицу. Аналогично, если *x* отрицательна, ее значение прибавляется переменной *sum_neg* (сумма отрицательных чисел), а значение счетчика *count_neg* отрицательных чисел увеличивается на единицу.

```
If x>=0 Then Begin
    sum_pos:=sum_pos+x;
    count_pos:=count_pos+1
    End
 Else Begin
    sum_pos:=sum_neg+x;
    count_neg:=count_neg+1
     End;
```

Программа-пример: Вычисление периметра и площади прямоугольника.

Задается длина и ширина прямоугольника. Программа вычисляет его периметр и площадь. Если стороны прямоугольника равны, желательно, чтобы вывести также сообщение, что это квадрат.

Алгоритм:

Ввести *длину* и *ширину*, Вычислить *периметр* и *площадь*;
Если длина = ширина
Тогда: Вывести: «Периметр и площадь квадрата = »
Иначе: «Периметр и площадь прямоугольника = »
Вывести *периметр* и *площадь*.

Программа:

```
Program rectungle;
{ Программа считывает длину и ширину прямоугольника }
{ Вычисляет его периметр и площадь }
{ Различает квадрат от прямоугольника }
Var breadth, height: Integer; { Входные данные }
    perimeter, area:Integer; { Выходные данные }
Begin
        { Ввод длины и ширины }
      Write(' Введите длину и ширину:');
                  Read(breadth, height);
{ Вычисление периметра и площади }
perimeter:= 2*(breadth+height);
area;==breadth*height;
if (breadth = height)
     Then Write(' Периметр и площадь прямоугольника = ')
             Else Write(' Периметр и площадь квадрата = ');
 { Вывод периметра и площади }
  Writeln(perimeter,' ', area);
End.
```

Обратите внимание, что инструкции алгоритма в программе используются в качестве комментариев.

Программа-пример: Вычисление зарплаты.

Написать программу, которой передаются количество рабочих часов в неделю и почасовая оплата служащего, программа должна выдавать его недельный оклад. Принцип оплаты такой — первые 35

часов служащему оплачивают по нормальному курсу, затем за каждый час платят в полтора раза больше.

Алгоритм:

1. Ввод:
   1.1. Запрос о количестве рабочих часов в неделю;
   1.2. Считать величину количества часов;
   1.3. Запрос о почасовом окладе;
   1.4. Считать величину почасового оклада;
2. Вычисление недельного оклада:
   2.1. *Если* количество рабочих часов в неделю ≤ 35
   Тогда
   2.1.1. Вычислить оклад «по норме»;
   Иначе
   2.1.2. Вычислить оклад по правилу «свыше нормы»;
3. Вывести значение годового оклада.

Программа:

```
Program wages;
{ Вычисляет недельную зарплату }
Const hoursLimit =35; { Лимит нормы для рабочих часов }
      overtimeFactor=1.5; { Коэффициент «сверх нормы» }
Var hourRate:Real;     { Оклад за час }
    workedHours:Real;  { Рабочие часы — за неделю }
    wage: Real;        { Недельный оклад }
Begin
    { Ввод проработанных часов за неделю }
    Writeln(' Введите количество рабочих часов:');
Read(workedHours);

    Writeln(' Введите величину почасовой оплаты:');
    Read(hourRate);
    { Вычисление оклада}
    If workedHours <= hoursLimit
```

```
    Then wage := workedHours * hourRate
        Else wage := (hoursLimit + (workedHours - hoursLimit)
                * overtimeFactor) * hourRate;
{ Вычисление оклада }
Writeln( 'Оклад за ', workedHours:8:3,' часов с ', hourRate:8:3,
                ' рублей за час = ',wage:8:3);
End.
```

Отметим, что комментарии здесь использовались для описания назначения каждой переменной и константы, а также основных шагов решения задачи.

Программа-пример: Триплеты Пифагора.

Рассмотрим задачу: пусть программа считывает два положительных числа в переменных m и n, где $m>n$ вычисляет и выводит соответствующий триплет Пифагора, т.е. числа: m^2-n^2, $2mn$ и m^2+n^2. При вводе чисел нужно проверить, являются ли они дозволенными, т.е. выполняется ли условие: $m>n$.

Алгоритм:

Ввести значения для *m* и *n;*
Если $m>n$
Тогда {
Вычислить числа Пифагора для *m* и *n;*.
Вывести триплет Пифагора;
}
Иначе Вывести сообщение об ошибке.
Программа:

```
Program pythagorean_triples;
Var n,m:Integer; { Для представления вводимых чисел }
    t1,t2,t3:Integer; { Для представления вводимых чисел }
BEGIN
    Write(' Введите значения для m и n, m>n: ');
    Read(m,n);
```

```
If m>n Then Begin
   tl := m*m-n*n;
   t2:=2*m*n;
   t3 := m*m+n*n;
        Writeln;
        Writeln('Триплет Пифагора для чисел', m,' и ', n, '
                : (', tl,',',t2,',',t3,').');
   End
Else Writeln ('Нелегальный ввод: первое число>второго!');
END.
```

Заметим, что при выводе печатаются и значения начальных данных, т.е. *m* и *n*. Это считается хорошим стилем – давать определенную исходную информацию, которая привела к данному результату. В приведенном примере данные печатаются полностью, так как их мало, но в других случаях полный вывод может оказаться слишком громоздким.

6.4. Вложенные операторы «If» и «If-Else».

Оператор «If-Else» позволяет сделать выбор между двумя возможными альтернативными операторами. Иногда нужно сделать выбор между более чем двумя альтернативами. Например, математическая функция *sign* определена как:

$$sign(x) = \begin{cases} -1, & если \quad x < 0; \\ 0, & если \quad x = 0; \\ 1, & если \quad x > 0. \end{cases}$$

На Турбо-Паскале ее можно представить следующим образом:

```
If(x<0)Then sign:=-l Else
If(x=0) Then sign:=0 Else sign:=l;
```

Здесь фактически используется один оператор «If-Else», в котором после «Else» в роли выполняемого оператора выступает

другой оператор «If-Else». Можно было бы добиться того же эффекта с помощью трех последовательных операторов «If»:

```
If (x<0) Then sign:= –1;
If(x=0)Then sign:=0;
If(x>0)Then sign:= 1;
```

Такую запись часто используют начинающие программисты, однако она не эффективна, так как каждый раз проверяются все три условия, а кроме того, неочевидно, что должно выполниться только одно присваивание.

Слишком большая глубина вложенных «If-Else» операторов может оказаться трудно читабельной, В этом случае соблюдается правило:

Каждый «Else» относится к ближайшему «If» без «Else».

6.5. Оператор выбора «Case».

В предыдущем разделе для реализации многократного выбора были использованы вложенные «If-Else» операторы. Однако, если вариантов выбора слишком много, такая запись кажется слишком громоздкой и нечитабельной. В Турбо-Паскале поддерживается другой оператор для выполнения той же задачи, это – оператор *выбора* «Case». Общий формат записывается в виде:

```
Case i of
    1: Оператор – 1;
    2: Оператор – 2;
    3: Оператор – 3;
Else Оператор – n { Необязательный }
End;
```

Здесь *i* – так называемый «селектор», переменная порядкового типа, т.е. целочисленного, символьного, логического, перечисляемого или интервального (см. гл. 10). В зависимости от текущего значения селектора будет выполняться либо оператор под *меткой* 1.

либо 2 и т.д. Следовательно, тип метки должен совпадать с типом селектора, а их значения должны быть различными. После метки может стоять только один оператор, простой или составной. Если значение селектора не совпадает ни с одной меткой, тогда выполняется оператор в разделе «Else». Этот раздел необязательный атрибут. Если он будет отсутствовать и если селектор не совпадет ни с одной меткой, то оператор «Case» будет просто игнорироваться.

Рассмотрим примеры.

Следующий оператор выводит на экран название дня недели по номеру, предполагая что первый день недели — понедельник.

```
Var day:Integer;
...
Case day of
    1: Writeln(' Понедельник');
    2: Writeln(' Вторник');
    3: Writeln('Среда');
    4: Writeln('Четверг');
    5: Writeln('Пятница');
    6: Writeln('Суббота');
    7: Writeln('Воскресенье');
Else Writeln(Неправильный номер для дня недели');
End;
...
```

Можно рассмотреть такой случай: пусть требуется определить, является ли данный день выходным. Тогда оператор будет иметь вид:

```
Var day:Integer;
...
Case day of
    1,2,3,4,5:;
    6,7: WriteIn ('Выходной!');
  Else Writeln(' Неправильный номер для дня недели ');
End;
...
```

Здесь отметим два момента. Во-первых, метки альтернатив можно перечислять через запятую, а, во-вторых, метка может быть «пустой», т.е. содержать пустой оператор.

Рассмотрим еще один пример, когда селектор – символьного типа. В некоторых университетах успеваемость студента оценивается на символьной шкале. Пусть «А» соответствует отличнику, т.е. оценке 5, «В» – оценке 4, и т.д. Запишем оператор, который переводит символьную шкалу оценок в численную:

```
Var numberGrade:Integer;
        charGrade:Char;
…
Case char Grade of
    'A': numberGrade:= 5;
    'B': numberGrade:= 4;
    'C': numberGrade:= 3;
    'D': numberGrade:= 2;
    Else Writeln(' Неправильный символ оценки');
End;
…
```