**Семантика оператора “case”.**

Одной из форм оператора выбора в языке Pascal является оператор сase. Синтаксис этого оператора приведен ниже.

сase <выражение> of

<список констант>:<оператор> {; <список констант>:<оператор>}

end

Например:

сase i mod 3 of

0: m:=0;

1: m:=-1;

2: m:=1;

end

ñase sym of

'=': k:=k+1;

'\*', '+', '/', '-': ;

'!': l:=l+1;

':', ';': p:=p+1;

end

3. сase день of

ПН, ВТ, СР, ЧТВ, ПТН: writeln('Рабочий день');

СБ, ВСКР: writeln('Выходной день');

end

Ясно, что в этих примерах переменная i - типа integer, переменная sym - типа char, день - перечислимого типа (ПН, ВТ, СР, ЧТВ, ПТН, СБ, ВСКР).

Действие оператора сase из примера 1 можно было бы описать так:

if i mod 3=0 then m:=0 else

if i mod 3=1 then m:=-1 else

if i mod 3=2 then m:=1;

Перепишем эту последовательность вложенных условных операторов в следующем виде:

if

i mod 3=0 → m:=0;

i mod 3=1 → m:=-1;

i mod 3=2 → m:=1;

end {if}

В этой форме условия трех альтернатив просматриваются последовательно сверху вниз и выполняется первая альтернатива, условие которой выполнено.

Обобщим теперь эту запись на большее число альтернатив следующим образом:

if

B1 → S1;

B2 → S2;

 . . .

Bk → Sk

end {if}

В этом обобщении последовательно сверху вниз просматриваются логические выражения Bi и для того i, где первым будет получено значение T, будет выполнен оператор Si. Причем обязательно хоть одно Bi должно принять значение T. Эту обобщенную форму условного оператора обозначим IF. Отсюда получаем.

Определение 11.1. Семантика оператора сase:

wp(ñase, R)=(∃ i: 1≤i≤k: Bi) ∧ (∀ i: 1≤i≤k: Bi ⇒ wp(Si, R))

Отсюда должно быть видно, что:

Всегда на текущем состоянии выполняется хотя бы одно Bi. Другими словами, предусловие этого оператора должно имплицировать любое Bi.

Если на текущем состоянии выполняется Bi, то соответствующий Si перерабатывает это состояние в такое состояние, где должно выполняться постусловие R.

Рассмотрим пример. Написать предусловие для оператора

Sñase:

ñase a of

1: b:=c+a;

2: b:=a+1;

3: b:=a-c

end

такое, что если этот оператор начинает работать в состоянии, удовлетворяющем этому предусловию, то он обязательно закончит свою работу и после ее выполнения мы получим состояние, удовлетворяющее условию R ≡ b>1.

Другими словами, надо вычислить

wp(Sñase, b>1).

Выпишем в соответствии с определением

wp(Scase , b>1) = ((a=1)∨(a=2)∨(a=3)) ∧

((a=1) ⇒ wp(b:=c+a, b>1)) ∧

((a=2) ⇒ wp(b:=a+1, b>1)) ∧

((a=3) ⇒ wp(b:=a-c, b>1))

Преобразуем каждый из предикатов (1)-(3) в соответствии с определением семантики оператора присваивания.

Получим:

(a=1) ⇒ (c+a>1) ≡ c>0

(a=2) ⇒ (a+1>1) ≡ 3>1 ≡ T

(a=3) ⇒ (a-c>1) ≡ c<2

В итоге с использованием правил эквивалентных преобразований из таблицы 5.8 получим

((a=1) ∧ (c>0)) ∨ (a=2) ∨ ((a=3) ∧ (c<2))

**Семантика оператора цикла.**

Здесь нам надо определить, как оператор цикла в языке Pascal меняет состояние вычислительного процесса. Другими словами, если задано постусловие R, то каково должно быть для него предусловие wp, чтобы в результате выполнения оператора получить R.

Операторы цикла в любом языке программирования реализуют операцию композиции повторения (см. лекция 3). В языке Pascal есть три вида оператора цикла:

while B do S;

for i:=E1 to E2 do S; (for i:=E1 downto E2 do S)

repeat S1 ; S2 ; … ; Sk until B;

где B - логическое выражение,

E1, E2 - выражения,

S1 ; S2 ; … ; Sk - операторы.

С циклом вида (1) мы уже встречались в примере 9.1 вычисления n членов гармонического ряда. Фрагмент программы (строки 5-8) показан на рис. 11.1

{Вычисление суммы ряда}

{Инициализация цикла}

s:=0;

i:=1;

{Вычисление суммы первых n членов гармонического ряда}

while i≤n do begin s:=s+1;

i:=i+1 end {while}

Рис. 11.1.

На рис. 9.3 были показаны все состояния вычислительного процесса программы Harmonic для n=5. Фактически каждая итерация этого цикла представляет из себя выполнение оператора if\_endif на рис. 11.2, а операторы в строках 5,6 устанавливают значения для s и i, необходимые для начала цикла.

if

B1 → s:=s+1/i; i:=i+1;

 . . . . .

B5 → s:=s+1/i; i:=i+1;

endif ,

где ∀j: 1≤j≤n: Bj = i=j.

Рис. 11.2.

Оператор IF, представленный на рис. 11.2, надо выполнять до тех пор, пока для очередного значения i истинен хоть один Bi ≡ i=n. Как только при очередном выполнении IF ни одно из условий не будет выполнено, то более повторять выполнение этого IF не надо.

Обозначим

BB=B1 ∨ B2 ∨ B3 ∨ ... ∨ Bk ,

тогда оператор IF на рис. 9.2 надо выполнять до тех пор, пока выполняется BB.

Теперь рассмотрим циклы вида (2). Для этого вернемся к примеру 10.1, программа для которого представлена на рис. 10.1. Интересующий нас в данный момент фрагмент дан на рис. 11.3.

begin fctrl:=1;

for i:=2 to n do fctrl:=fctrl\*i

end {if n=0}

Рис. 11.3

Здесь был использован цикл вида (2). Все состояния вычислительного процесса, соответствующие этому циклу, были пока

заны на рис. 10.2. Там же, в колонке, где указан номер итерации, указано и условие, при выполнении которого надо выполнять тело цикла на данной итерации.

Рассмотрим оператор IF на рис. 11.4.

i:=2 ; if

B1 → fctrl:=fctrl\*i; i:=i+1;

B2 → fctrl:=fctrl\*i; i:=i+1;

 . . . . .

B5 → fctrl:=fctrl\*i; i:=i+1

endif

где ∀k: 1≤ k ≤5 Bk ≡ i=k.

Рис. 11.4

Если его выполнять до тех пор, пока выполняется BB, то мы получим конструкцию, эквивалентную той, что показана на рис. 11.3 с той оговоркой, что значение параметра цикла i после выхода из цикла на рис. 11.3 считается не определенным. Для того, чтобы убедиться в эквивалентности этих конструкций, достаточно построить вычислительный процесс, соответствующий циклическому выполнению IF с рис. 11.4 и сравнить его с вычислительным процессом на рис.10.2.

Таким образом, мы видим, что циклы вида (2) тоже сводятся к последовательному выполнению операторов типа IF, пока выполняется условие BB.

Рассмотрим теперь циклы вида (3):

repeat S until B.

На интуитивном уровне действие операторов цикла этого вида можно описать так: после каждого выполнения оператора S надо проверять условие B. Если оно не выполняется, то надо выполнить S еще раз, если выполняется, то цикл заканчивается. Другими словами, оператор цикла вида (3) можно выразить оператор цикла вида (1) следующим образом:

repeat S until B;

≡

S; while ¬B do S.

Однако, как мы уже установили, оператор цикла вида (1) выражается через повторные выполнения оператора IF в соответствующей форме. Здесь уместно вспомнить, что при построении операции композиции повторения мы использовали операцию композиции выбора также.

Мы приходим к выводу, что все наши три вида операторов цикла сводятся к повторному выполнению соответствующего оператора IF. Назовем эту конструкцию, лежащую в основе всех этих операторов цикла, циклом вида DO. Ее вид представлен на рис. 11.5.

DO

if

B1 → S1;

B2 → S2;

 . . .

Bn → Sn

endif

ENDO

Рис 11.5. Цикл вида DO.

Еще раз напомним, как он работает. Последовательно просматриваются сверху вниз условия Bi, и для первого выполненного условия выполняется надлежащий оператор Si. После этого опять cверху вниз, начиная с первого, просматриваются условия Bi. Выполнение оператора if заканчивается, когда при очередном просмотре ни одно Bi не будет выполняться.

Итак, для определения семантики трех видов оператора цикла нам достаточно определить семантику оператора DO! Теперь, придя к пониманию того, что в основе всех циклов лежит конструкция DO, попытаемся выписать ее семантику, то есть определить вид предиката

wp(DO, R)

Для любого цикла нам важно обеспечить выполнение двух условий:

Оператор цикла обязательно заканчивается.

Работа этого оператора заканчивается в состоянии, удовлетворяющим постусловию R.

Обозначим C0(R) предикат следующего вида:

C0(R)=¬BB∧R,

где BB - конъюнкция условий в операторе DO, а R - постусловие.

Очевидно, что C0(R) выражает условия 1,2 для оператора цикла, т.е. что цикл заканчивается (это описывает условие ¬BB) и заканчивается в состоянии R (это описывает второй конъюнкт R).

Обозначим

C1(R)= C0(R) ∨ wp(IF, C0(R)),

где wp(IF, C0(R)) - предусловие для оператора IF в операторе DO такое, что за одну итерацию цикла DO оператор IF переработает это состояние в состояние, удовлетворяющее условию C0(R), т.е. что цикл заканчивается и заканчивается в состоянии R.

Таким образом, условие C1(R) описывает состояние, на котором цикл DO закончится не более чем за одну итерацию в состоянии, удовлетворяющем R. Теперь обобщим предикат C1(R) для случая не одной итерации цикла, а k итераций,

Ck(R)= C0(R) ∨ wp(IF, Ck-1(R)), где k>0 .

Этот предикат выражает условие, что либо на текущем состоянии цикл заканчивается (член C0(R)), либо за одну итерацию цикла мы получим состояние, начиная с которого цикл закончится не более чем за k-1 итерацию (член wp(IF, Ck-1(R))).

Таким образом, нижеприведенный предикат:

∃ k: k≥0: Ck(R) (11.1)

выражает условия 1,2 для циклов. Однако, он мало что нам дает для понимания внутреннего устройства цикла, семантики итераций.

Как мы уже сказали, важно гарантировать, что цикл закончится и непременно в состоянии, удовлетворяющем постусловию. Обеспечивает эту гарантию особое утверждение (предикат), сохраняющий истинность на любом состоянии, порождаемом в цикле. Этот предикат называется инвариантом цикла или просто инвариантом. Другими словами, цикл DO устроен таким образом, что состояния, которые мы получаем в конце каждой итерации, носят не произвольный характер, а подчиняются некоторому единому условию, которое называется инвариантом цикла.

Вернемся к нашим примерам 9.1 и 10.1. В примере 9.1 нетрудно заметить, что условие

s=(Σ k: 1≤k≤i: 1/k) (11.2)

сохраняет истинное значение в течение всей работы цикла. Здесь запись

(Σ k: 1≤k≤i: 1/k)

означает .

Для цикла из примера 10.1, приведенному на рис. 11.3, инвариант выглядит так:

P: s=(Πk: 1≤k<i: k)) (11.3)

где s=Πk: 1≤k<i :k означает .

Таким образом, мы можем добавить к условию (11.1) утверждение

(∀i: 0≤i<k: P∧BB ⇒ (IF, P)) ∧ (P ∧ ¬BB) ⇒ R ,

где P - инвариант цикла.

Другими словами, если P - инвариант, то он должен удовлетворять следующим условиям (11.4):

P должен выполняться непосредственно перед входом в цикл.

∀i: 1≤i≤n: {P∧Bi} Si {P}.

P ∧ ¬BB ⇒ R

В итоге получаем:

wp(DO, R) = ∃k: k≥0: Ck(R) ∧ (∀i: 0≤i<k: P∧BB ⇒ (IF, P)) ∧ (P ∧ ¬BB ⇒ R) ,

где P - инвариант цикла.

Заметим, что не любое утверждение на множестве состояний итераций цикла является инвариантом цикла. Инвариант должен конструктивно помогать доказательству корректности цикла, т.е. что цикл заканчивается, т.е.

P ⇒ wp(DO, P ∧ ¬BB) (11.5)

и что заканчивается в состоянии, удовлетворяющем R, т.е.

(P ∧ ¬BB) ⇒ R (11.6)

В качестве упражнения докажите, что инварианты (11.2) и (11.3) обладают свойствами (11.5) и (11.6).

Рассмотрим следующий пример:

Доказать, что в следующем цикле P - инвариант и цикл заканчивается в нужном состоянии R.

{{T} - т.е. начинается с произвольного состояния}

i:=10, s:=0;

{P: 0≤i≤10 ∧ s = (Σk: i+1≤k≤10: b[k])}

while i≥1 do begin s:=s+b[i]; i:=i-1; end

{R: s = (Σi: 1≤i≤10: b[i])}

Для того, чтобы доказать, что P - инвариант этого цикла, нам надо показать, что выполняются условия (11.4) 1-3. То, что P выполняется перед входом в цикл, доказать нетрудно. Для этого достаточно подставить в P значения i и s и учесть, что запись

(Σ k: i+1≤k≤10: b[k])

при i=10 даст 0.

Докажем, что выполняется условие (11.4) 2., т.е.

∀k: 0≤k≤10: {P∧Bk} S {P},

где Bk= k+i=10 ∧ i ≥1; s: s:=s+b[i]; i:=i-1.

Это несложно сделать с помощью метода математической индукции.

Теперь нам осталось показать (11.4)3, что

P∧¬BB⇒R,

это условие можно записать так

P∧i<1 ⇒R.

Учитывая, что в условии Р переменная i от 0 до 10, следовательно i<1, только при i=0. Отсюда, поставив в Р значение i=0, получим требуемое утверждение R.

 Теперь, имея представление о семантике основных операторов языка Pascal, рассмотрим,что же нам дает такое понимание семантики для создания программ? Здесь, конечно, уместно было бы спросить, что мы понимаем под словами “создание программ”? Но мы ответим на этот вопрос чуть позже. А пока вернемся к нашим программам 9.1 и 10.1 и попытаемся на них понять, что же нам дает знание семантики.

 Перепишем программу из примера 9.1, но в качестве комментариев будем записывать предусловия и постусловия для операторов этой программы. Результат этого преобразования показан на рис. 11.1.

Program Harmonic (input, output);

{ Предусловие: (n∈ N)∧(n>0);

 Постусловие: s = }

 var n, i : integer;

 s : real;

begin

{ T }

write(’Введите значение n =’);

readln( n ); { n∈ N }

if n>0 then

begin {(n∈ N)∧(n>0)}

s:=0 ;

i:=1; {i=1 ∧ s=0 ∧ n>0}

while i≤n do

begin {s=Σk : 0<k<i : 1/k}

begin s:=s+1/i ; i:=i+1 end

{i=n , , n>0};

writeln (’Сумма’, п , ’ членов

гармонического ряда =’’, s)

end {}

else {n≤0} writeln (’Ошибка в исходных данных.

Должно быть’, n,’ >0’)

end {}

Рис. 11.1.

В первом комментарии сформулированы пост- и пред- условия, определяющие множество допустимых исходных данных и множество возможных результатов. Комментарий {T} между строками 2 и 3 утверждает, что в этой точке программы допустимо любое состояние. Действительно, мы ничего пока еще не можем сказать в этой точке о значениях переменных этой программы.

 Однако, после оператора readln (n) (строка 4) мы уже можем утверждать, что n∈ N, т.е. целое. Как Вы уже, наверное, изучали на семинарских занятиях, Pascal - это строго типизированный язык программирования. Это означает, что каждой переменной в программе как бы сопоставляется предикат, определяющий тип этой переменной. Всякий раз, когда происходит запись нового значения в эту переменную, происходит вычисление значения этого предиката. Если оно T, то запись выполняется, если F, то выдается сообщение об ошибке, и выполнение программы прекращается. Поэтому, после оператора readln (n), т.е. после его нормального завершения, мы можем быть уверены, что n - целое значение. Но мы ничего не можем сказать о величине этого значения. Поэтому, нам надо убедиться,что n > 0 , прежде чем начать вычисления.

 В строке 5 логическое выражение n>0 обеспечивает проверку исходных данных на соответствие предусловию программы. Поэтому, после строки 5 мы можем быть уверены, что любое состояние, полученное в этой точке, будет удовлетворять предикату, написанному в виде комментария - {(n∈ N)∧(n>0)}.

 Комментарий в конце 7 строки определяет состояния вычислительного процесса непосредственно перед входом в цикл. Комментарий после заголовка цикла (строка 8) - { s=Σk : 0<k<i : 1/k} - определяет инвариант цикл. То, что это действительно так, мы убедились ранее, при рассмотрении семантики цикла.

 Комментарий после строки 9 определяет состояние вычислительного процесса, непосредственно после окончания цикла. Он как раз утверждает то, что в этом состоянии переменная s будет иметь в качестве своего значения сумму ряда .

 Комментарий в строке 10, после end показывает, что в этой точке состояние вычислительного процесса не изменится, если не принимать в расчет переменную П - указатель номера выполняемого оператора (см. лекцию 2).

 Комментарий в строке 11 утверждает, что в этой точке у любого состояния вычислительного процесса n≤0. Ну и , наконец, последний комментарий утверждает, что в любом заключительном состоянии вычислительного процесса, соответствующего корректным исходным данным (предусловие выполнено), в этой программе , т.е. постусловие для программы выполнено.

 Если теперь сравнить программу на рис. 11.1 с двумя другими формами этой же программы, представленными на рис. 9.1 и рис. 9.2 , то мы должны согласиться, что это новая форма, новая программа. Согласно определению 9.1 один и тот же алгоритм может иметь разные программы, его реализующие. Различия между ними могут отражать различия во взглядах, потребностях создателя, исполнителя и пользователя этими программами. Что демонстрирует программа на рис. 11.1?

 Представим, что у компьютера, на котором мы будем исполнять эту программу, есть

кнопка, нажав которую, мы заставляем его выполнять очередной оператор программы и, выполнив его, остановиться;

после очередного нажатия, на специальном дисплее высветить текущие значения всех переменных программы.

Тогда, имея программу вычисления суммы n первых членов гармонического ряда в форме как на рис. 11.1, мы легко сможем проверить после каждого нажатия кнопки: действительно ли программа работает верно. Для этого достаточно подставить в предикат, определяющий текущее состояние, значения переменных из этого состояния, получить высказывания и вычислить его истинность. Если мы получим значение Т, то программа работает правильно, если F, то не правильно.

Заметим также, что, зная семантику операторов, мы могли бы доказать, что в случае программы из примера 9.1, для каждого из предусловий соответствующий опреатор приводит состояние, удовлетворяющее требуемому постусловию. И, что в итоге, мы получим состояние, удовлетворяющее постусловию для программы.

Таким образом, мы приходим к выводу, что программа на рис. 11.1 отражает взгляд на программу, показанную на рис. 9.1, того, кто будет проверять правильность этой программы.

Определение 11.1. Назовем процесс формулировки пред и постусловий как для программы вцелом, так и для отдельных ее частей, вплоть до опраторов, процессом спецификации программы.

Определение 11.2. Набор пред и постусловий в порядке их выполнения, как для программы вцелом, так и для отдельных ее частей, вплоть до операторов, назовем спецификацией программы.

Теперь мы можем сказать, что на рис. 11.1 представлена программа с рис. 9.2 со спецификацией.

На рис. 11.2 показана программа и ее спецификация для примера 10.1. В отличие от программы на рис. 11.1, в этой программе, после проверки корректности исходных данных, мы сначала выдаем сообщение об ошибке и только потом приступаем к обработке. В программе на рис. 11.1 сделано наоборот.

 Итак, мы обнаружили с Вами три разных формы представления, вообще говоря, одной и той же программы:

для пользовтеля, кто будет читать и стараться понять как работает эта программа;

для компьютера, который будет выполнять эту программу;

для того, кто будет проверять правильность этой программы.

Program Factorial (input, output);

{ Предусловие: (n∈ N)∧(n ≥ 0)

 Постусловие: (Fctrl=) }

var i , n , fctrl : integer;

begin { T }

write(′Введите значение n = ′);

readln( n ); { n∈ N }

if n < 0 then { n < 0 }writeln (′Ошибка: n не может быть < 0′)

else

begin { n ≥ 0 }

 if n = 0 then {n=0} fctrl:=1 {n=0∧fctrl=1∧(i не определенно)}

else

begin { n > 0 }

fctrl : =1 ;

{n=0 ∧ fctrl=1}

for i : =2 to n do fctrl : = fctrl \* i

{∀i: 2 ≤ i ≤ n : Πk : 1≤k<i : k};

{ n>0 ∧ fctrl=(Πi : 1≤i<n : i) }

writeln (′ При n = ′ , n , ′ n! = ′ , fctrl )

end

{ n ≥ 0 ∧ fctrl= n! };

end

end {Program}

Рис. 11.2.

Упражнения:

|  |  |
| --- | --- |
| {?}1. if a=1 then b:=a else b:=a+1; {b=1} {i=n ∧ j=m} | wp (S5 , a > 0 ∧ b > 0) = ?S5: if a > b then a:=a-b else if b < a then b:=b-a; |
| 2. if i=0 then j:=0 else j:=r;{?} | wp (S6 , x ≤ y) = ?S6: if x > y then begin t:=x;x:=y;y:=t end; |
|  {i=n ∧ j=m} if i=0 then j:=0;{?} | Вычислить:wp(x:=2\*y+3, х=13)wp(x:=х+y, х<2\*y)wp(j:=j+1, 0<j ∧(∀i: 0 ≤ i ≤ j: b[i]=5))wp(x:=х\*y , х\*y=c) |
| 4. if w ≤ r then begin r:=r-w; q:=q+1; end ; {q \* w+r = x ∧r ≥0} | case c of′ + ′ : p:=p+1;′ - ′ : m:=m+1;′(′, ′)′ : b:=b+1;′\*′,′/′ : p:=b+1; end {p>0, m>0, b=0} |

Цикл.

Проверка цикла.

Доказать, что Р до входа в цикл

∀i: 1 ≤ i ≤ n : {P∧Βi} S {P}

P∧¬BB⇒R

Доказать,что в следующих циклах Р - инвариант и цикл заканчивается в нужном состоянии R:

1.

{0 ≤ n }

i:=0 ;

{P: 0 ≤ i ≤ n ∧ x∉ b [0: i-1]}

while (i < n) and (x≠ b[i]) do i:=i+1;

{R: (0 ≤ i ≤ n ∧ x= b[i])∨( i = n ∧ x∉ b [0: i-1])}

2.

{0 < n }

i:=1 ;

{P: 0 < i ≤ n ∧ (∃p: i=2p)}

while 2\*i ≤ n do i:=2\*i ;

{R: 0 < i ≤ n< 2\*i ∧ (∃p: i=2p)}.