**7. Порядковые типы данных**

Типы данных описывают множества значений и операции, которые могут быть применены к ним. Типы данных введенные CF Pascal это CHAR и TEXT. Значениями типа CHAR являются множество допустимых символов Паскаль-машины и операциями над этими значениями являются операторы сравнения:

= < > <= >= <>

значениями которых являются соответствующие математические операции, отражающие различное положение символов в алфавитной последовательности. Значения типа данных TEXT – последовательности строк, каждая строка является последовательностью символов. Операции над такими данными:

RESET REWRITE READ WRITE WRITELN OEF EOLN

Типы данных предлагают преимущества *абстрагирования*, *репликации* и *аутентификации*.

Типы данных абстрагируют важные свойства данных. Например, для понимания сравнения

‘A’ < ‘B’

нет необходимости знать, как эти символы представлены в Паскаль-машине. Последовательность сортировки определяет значение этих операций. Пользователю нет необходимости знать количество битов, которыми представлено символьное значение, что происходит с избыточными битами, действительно ли битовое значение для символа **B** больше чем для **A** и т.д.

Модули реализуют разновидность абстракции данных, но типы данных предлагают нечто большее. Объявление переменных типа данных позволяет хранить и манипулировать любым количеством значений типа через эти переменные. Каждое использование переменной в программе дает нам гарантию, что к ней будут применены только допустимые операции.

Контекст операнда в программе подразумевает тип операнда, этот подразумеваемый тип – избыточная информация, которая может быть проверена на соответствие объявленному типу операнда для предотвращения операций над значениями неверного типа. Рассмотрим следующий фрагмент программы:

PROGRAM Typex(INPUT, OUTPUT);

VAR

Ch: CHAR;

F: TEXT;

BEGIN {Typex}

...

IF Ch = F

THEN

...

END. {Typex}

Поскольку операция = определена в CF Pascal только для символьных операндов, появление Ch и F в качестве операндов подразумевает, что оба они типа CHAR. У Ch тип корректен, а F объявлена как TEXT, поэтому будет выявлено несоответствие.

В D Pascal существует две разновидности типов данных: простые типы данных и составные (aggregate) типы данных. Значения простых типов не могут состоять из более мелких частей, с которыми можно работать по отдельности. CHAR – простой тип. Значения сложных типов данных образованы объединением значений простых типов. TEXT – составной тип, потому что строки фалов состоят из символов.

Значения простых типов упорядочены, то есть для каждой пары значений этого типа x, y возможен только один из вариантов: x < y, x = y, x > y. Поэтому эти типы называются порядковыми.

Паскаль содержит три предопределенных порядковых типа, которые задаются идентификаторами CHAR, INTEGER и BOOLEAN. Дополнительно к предопределенным порядковым типам, в Паскале предусмотрено два метода для программиста для определения новых порядковых типов, а именно:

1. Перечислимые типы, значениями которых являются уникальные идентификаторы.
2. Типы диапазон, значениями которых являются последовательные значения другого порядкового типа.

То есть новый порядковые типы данных могут быть определены перечислением констант типа или указанием, что значения типа являются поддиапазоном значений существующего типа. Синтаксис для обозначения таких порядковых типов следующий:

<обозначение типа> ::= <идентификатор типа> | <новый тип>

<идентификатор типа> ::= <идентификатор>

<новый тип> ::= <перечислимый тип> | <тип диапазон>

Способы описания для <перечислимый тип> и <тип диапазон> будут даны в соответствующих разделах ниже.

Когда определяется новый тип, ему может быть дано имя в объявлении типов. Это объявление должно предшествовать объявлению переменных блока.

<блок> ::= <раздел объявлений типов> <раздел объявлений переменных>

<раздел объявлений процедур> <раздел операторов>

<раздел объявлений типов> ::= TYPE <объявления типов> |

Это правило показывает что <раздел объявлений типов> может быть пустым (как это было во всех программах до данного места)

<объявления типов> ::= <объявления типов> <объявление типа> | <объявление типа>

<объявление типа> ::=<идентификатор > = <обозначение типа>

Контекстное правило, которое сопровождает эти синтаксические правила, говорит о том, что только некоторые идентификаторы являются <объявлением типа>:

CR Чтобы быть использованным как <идентификатор типа>, <идентификатор> должен ранее появиться в <объявлениях типов>.

Таким образом для

TYPE

T1 = CHAR;

T2 = T1;

и T1 и T2 являются идентификаторами типа и могут быть использованы в текущем блоке для объявления переменных и формальных параметров, как ранее использовался CHAR.

**7.1. Перечислимые типы**

Часто возникает ситуация когда необходим небольшой набор значений, где каждое значение имеет мнемоническое имя. Вместо использования символьных констант, Паскаль-программисты могут использовать произвольные идентификаторы для обозначения таких констант.

Любой список идентификаторов Паскаля может быть описан как тип данных с одним ограничением, что в данной области видимости каждый идентификатор появится не чаще одного раза. Например, объявления:

TYPE

DayOfWeek = (Monday, Tuesday, Wednesday, Thursday,

Friday, Saturday, Sunday);

VAR

Day : DayOfWeek;

описывают новый тип, называемый день недели, семь констант, входящих в него записаны как идентификаторы, именующие дни недели, и переменная Day может принимать любые из этих константных значений. Day может принимать значения только из списка заданного в DayOfWeek. Значения DayOfWeek упорядочены так, как они перечисляются в объявлении типа:

Monday < Tuesday < ... < Sunday

Таким образом, есть возможность записывать операторы следующим образом:

IF Day < Saturday

THEN {не weekend}

...

В объявлении:

TYPE

DayOfWeek = (Sunday, Monday, Tuesday, Wednesday, Thursday,

Friday, Saturday);

WeekEndDay = (Saturday, Sunday); {недопустимо}

VAR

Friday : CHAR; {недопустимо}

идентификаторы типа WeekEndDay и переменная Friday являются недопустимыми дубликатами идентификаторов Friday, Saturday и Sunday, которые ранее появились в объявлении типа DayOfWeek. То есть идентификаторы должны быть уникальными для всех их возможных вариантов использования в качестве имен типов, переменных, процедур, или констант в перечислимом типе.

Перечислимые типы являются “суперсимволами” Паскаля. Они могут быть использованы также как и обычные символы Паскаля и к ним применимы те же операции, но программист может изобрести любое их количество, желательно, чтобы идентификаторы соответствовали значению. Ранее мы использовали для таких целей символьные константы, но перечислимые типы лучше. Например, вместо возвращения значений Y или N для обозначения успеха, может быть использован перечислимый тип с идентификаторами Yes и No или даже Success и Failure.

Поскольку буквы алфавита идентификаторы, они могут быть использованы для задания типа данных:

TYPE

Alphabet = (A, B, C, D, E, F, G, H, I, J, K, L, M,

N, O, P, Q, R, S, T, U, V, W, X, Y, Z);

VAR

Letter: Alphabet;

Что позволяет следующие выражения:

IF Letter = A

THEN

Letter := E;

но при этом недопустимо:

IF Letter = ‘A’

THEN

Letter := ‘E’;

Поскольку A и E – константы типа Alphabet и соответствуют типу Letter, но ‘A’ и ‘E’ типа CHAR и это другой тип, чем у Letter.

Во многих задачах порядок, определенный перечислимым типом может быть полезен. Например, при работе с датами, важен порядок месяцев в календарном году. Поскольку трехсимвольные сокращения названий месяцев (Jan, Feb и т.д.) легко распознаются и компьютером и людьми, эти сокращения используются как строки, имеющие следующий лексикографический порядок.

Apr, Aug, Dec, Feb, Jan, Jul, Jun, Mar, May, Nov, Oct, Sep

который бесполезен в календарном смысле. Однако следующий перечислимы тип:

TYPE

Month = (Jan, Feb, Mar, Apr, May, Jun,

Jul, Aug, Sep, Oct, Nov, Dec);

Предоставляет автоматически способ сравнения значений переменных в соотвествии с календарем, если они заданы типом Month.

**7.1.1. Ввод-вывод для перечислимых типов.**

Не существует стандартного способа чтения/записи значений перечислимых типов с помощью операторов READ/WRITE, такого как для типа CHAR. Для того, чтобы проиллюстрировать как организуется ввод-вывод, будет разработан модуль чтения-записи для перечислимого типа Month, представляющего месяцы года. Этот модуль будет содержать две процедуры ReadMonth и WriteMonth, но не будет содержать объявлений данных. То есть это чисто вычислительный модуль.

ReadMonth преобразует три символа в файле открытом для чтения в значение типа Month, если это возможно. Например, если следующие три символа ‘J’, ‘A’, ‘N’, тогда ReadMonth вернет значение Jan типа Month. Однако если для трех считанных символов нет соответствующего значения типа Month, будет сообщено об ошибке. Простой и полезный способ сообщения об ошибке – расширить тип Month значением NoMonth. Тогда, если три считанных символа не являются сокращенным названием месяца, ReadMonth возвращает значение NoMonth.

Объявление типа будет следующим:

TYPE

Month = (NoMonth, Jan, Feb, Mar, Apr, May, Jun,

Jul, Aug, Sep, Oct, Nov, Dec);

Процедуры модуля будут в основном использоваться для чтения из INPUT и записи в OUTPUT, но дисциплина проектирования модулей требует чтобы в процедуры передавались как параметры файловые переменные, иначе придется работать с данными за пределами модуля.

PROCEDURE ReadMonth(VAR FIn: TEXT; VAR Mo: Month)

{Fin.3 = R и length(Fin.2) >= 3 ->

читать три символа из Fin.2, затем присвоить Mo

соответствующее значение типа Month, если это возможно,

иначе присвоить Mo значение NoMonth}

VAR

Ch1,Ch2,Ch3 : CHAR;

BEGIN{ReadMonth}

READ(FIn,Ch1,Ch2,Ch3);

IF (Ch1='J')AND (Ch2='A')AND (Ch3='N') THEN Mo:=JAN ELSE

IF (Ch1='F')AND (Ch2='E')AND (Ch3='B') THEN Mo:=FEB ELSE

IF (Ch1='M')AND (Ch2='A')AND (Ch3='R') THEN Mo:=MAR ELSE

IF (Ch1='A')AND (Ch2='P')AND (Ch3='R') THEN Mo:=APR ELSE

IF (Ch1='M')AND (Ch2='A')AND (Ch3='Y') THEN Mo:=MAY ELSE

IF (Ch1='J')AND (Ch2='U')AND (Ch3='N') THEN Mo:=JUN ELSE

IF (Ch1='J')AND (Ch2='U')AND (Ch3='L') THEN Mo:=JUL ELSE

IF (Ch1='A')AND (Ch2='U')AND (Ch3='G') THEN Mo:=AUG ELSE

IF (Ch1='S')AND (Ch2='E')AND (Ch3='P') THEN Mo:=SEP ELSE

IF (Ch1='O')AND (Ch2='C')AND (Ch3='T') THEN Mo:=OCT ELSE

IF (Ch1='N')AND (Ch2='O')AND (Ch3='V') THEN Mo:=NOV ELSE

IF (Ch1='D')AND (Ch2='E')AND (Ch3='C') THEN Mo:=DEC

ELSE Mo:=NoMonth

END; {ReadMonth}

PROCEDURE WriteMonth(VAR FOut: TEXT; VAR Mo: Month);

{Fin.3 = Ц и Mo <> NoMonth ->

вывести три символа соответствующие значению Mo, в Fout.1}

VAR

Ch1,Ch2,Ch3 : CHAR;

BEGIN {WriteMonth}

IF Mo=JAN THEN WRITE('Jan') ELSE

IF Mo=FEB THEN WRITE('Feb') ELSE

IF Mo=MAR THEN WRITE('Mar') ELSE

IF Mo=APR THEN WRITE('Apr') ELSE

IF Mo=MAY THEN WRITE('May') ELSE

IF Mo=JUN THEN WRITE('Jun') ELSE

IF Mo=JUL THEN WRITE('Jul') ELSE

IF Mo=AUG THEN WRITE('Aug') ELSE

IF Mo=SEP THEN WRITE('Sep') ELSE

IF Mo=OCT THEN WRITE('Oct') ELSE

IF Mo=NOV THEN WRITE('Nov') ELSE

IF Mo=DEC THEN WRITE('Dec')

END; {WriteMonth}

**7.1.2. Сравнение дат.**

Этот модуль может быть использован для определения календарного порядка месяцев, представленных в виде сокращений в INPUT.

DP 1

PROGRAM CalendarOrder(INPUT, OUTPUT);

{Распознать, если возможно два трехсимвольных сокращения

для месяцев и вывести их в календарном порядке.}

TYPE

Month = (NoMonth, Jan, Feb, Mar, Apr, May, Jun,

Jul, Aug, Sep, Oct, Nov, Dec);

VAR

M1, M2: Month;

{включить модуль для Month}

BEGIN {CalendarOrder}

ReadMonth(INPUT, M1);

ReadMonth(INPUT, M2);

{Сравнить M1 и M2 и вывести результаты}

END. {CalendarOrder}

После того как оба вызова процедуры ReadMonth будут выполнены M1 и M2 будут содержать значения типа Month, возможно NoMonth. Оставшийся код очевиден:

DP 1.1.

BEGIN {Сравнить M1 и M2 и вывести результаты}

IF (M1 = NoMonth) OR (M2 = NoMonth)

THEN

WRITELN(‘Входные данные записаны неверно’)

ELSE

IF M1 = M2

THEN

BEGIN

WRITE(‘Оба месяца ’);

WriteMonth(OUTPUT, M1);

WRITELN

END

ELSE

{Сравнить M1 и M2 и вывести результат}

END

DP 1.1.1.

BEGIN {Сравнить M1 и M2 и вывести результат}

WriteMonth(OUTPUT, M1);

IF (M1 < M2)

THEN

WRITE (‘ предшествует ’)

ELSE

WRITE (‘ следует за ’);

WriteMonth(M2);

WRITELN

END

Выполнение:

INPUT: APRJUL

OUTPUT: Apr предшествует Jul

INPUT: AprJul

OUTPUT: Входные данные записаны неверно

INPUT: JULAPR

OUTPUT: Jul следует за Apr

INPUT: DECDEC

OUTPUT: Оба месяца DEC

**7.2. Тип BOOLEAN**

Логические операции NOT, AND и OR были введены в главе 3. Эти операции работают с значениями типа BOOLEAN из D Pascal. Несколько иным определением синтаксиса логических выражений мы можем специфицировать ассоциативность и приоритет операций.

Тип BOOLEAN использует константные значения FALSE и TRUE и операторы NOT, AND, OR и операторы сравнения. Два константных идентификатора типа BOOLEAN работаю как если бы они были объявлены в перечислимом типе:

TYPE

BOOLEAN = (FALSE, TRUE);

(Но это объявление не присутствует явно.) Операторы сравнения определены на операндах типа BOOLEAN таким образом, что значение

FALSE < TRUE

равно TRUE.

NOT – унарный префиксный оператор, а AND и OR – бинарные инфиксные операторы. Функции, вычисляемые AND, OR и NOT, представлены в разделе 3.3.

Паскаль гарантирует порядок, в котором применяются операторы, но не для каждого конкретного оператора. Многие Паскаль-машины выполняют «ленивое» (lazy) вычисление логических выражений. Если первый операнд для OR имеет значение TRUE, второй операнд не вычисляется, потому что

TRUE OR x = TRUE

для любого x. Аналогично, если первый операнд для AND имеет значение FALSE, второй операнд не вычисляется, потому что все выражение будет FALSE. Этот метод вычисления возвращает корректные значения для выражений, которые в ином случае могли бы быть не определены.

Переменные типа BOOLEAN могут быть объявлены, и им может быть присвоено значение, являющееся результатом вычисления логического выражения. Например, для данного объявления:

VAR

EndWord: BOOLEAN;

следующее присваивание присваивает EndWord TRUE или FALSE:

EndWord := (Ch = ‘#’) OR (Ch = ‘ ‘)

**7.2.1. Ввод-вывод логических значений.**

Логичесике переменные не могут появляться в операторах READ, но логические выражения могут появляться в операторах WRITE. Логические значения преобразуются в символы и выравниваются по правой стороне строки, длина которой зависит от реализации Паскаль-машины (например, 10 символов).

Последовательность операторов:

Condition := TRUE;

WRITE(Condition);

Добавляет строку † TRUE† в OUTPUT.

**7.2.2. Поиск в файле.**

Программа SarahRevere в части 2 распознает слова land и sea в INPUT. Разделы проекта этой программы переработаны с использованием логических перемнных и собраны вновь ниже.

PROGRAM SarahRevere(INPUT, OUTPUT);

VAR

W1, W2, W3, W4: CHAR;

Looking, Land, Sea: BOOLEAN;

BEGIN {SarahRevere}

BEGIN {Инициализируем W1, W2, W3, W4}

W1 := ‘ ‘;

W2 := ‘ ‘;

W3 := ‘ ‘;

W4 := ‘ ‘;

Looking := TRUE;

Land := FALSE;

Sea := FALSE;

END;

WHILE Looking AND NOT (Land OR Sea)

DO

BEGIN

BEGIN {движение окна, проверка на конец данных}

W1 := W2;

W2 := W3;

W3 := W4;

READ(W4);

Looking := W4 <> ‘#’

END;

BEGIN {проверка окна на land}

Land := (W1 = ‘l’) AND (W2 = ‘a’) AND

(W3 = ‘n’) AND (w4 = ‘d’)

END;

BEGIN {проверка окна на sea}

Sea := (W1 = ‘s’) AND (W2 = ‘e’) AND (W3 = ‘a’)

END;

END;

BEGIN {создание сообщения Sarah}

IF Land

THEN

WRITELN(‘The british are coming by land.’)

ELSE

IF Sea

THEN

WRITELN(‘The british are coming by sea.’)

ELSE

WRITELN(‘Sarah didn’t say.’)

END

END. {SarahRevere}

Оператор WHILE управляется выражением:

Looking AND NOT (Land OR Sea)

которое разрешает телу цикла выплняться до тех пор пока значение Looking равно TRUE и оба значения Land и Sea равны FALSE. Looking присваивается значение False когда встречается #:

Looking := W4 <> ‘#’

Land и Sea присваивается TRUE если W1, W2, W3, W4 содержат соотвествующую строку.

Использование логических переменных сократило программу с 57 строк до 43.

**7.3. Тип INTEGER.**

Тип данных INTEGER упрощает задачи программирования связанные с вычислениями. Доступны все арифметические операции. Арифметика с использованим операндов типа INTEGER работает быстро, потому что практически все компьютеры имеют специальную поддержку этого типа данных. Однако платой за скорость является ограничение размера целочисленных данных.

Тип данных INTEGER имеет значения, которые могут быть положительными и отрицательными целыми числами и нулем. Любая Паскаль-машина ограничивает целые значения до подмножества целых чисел:

{-MAXINT, …, -2, -1, 0, 1, 2, …, MAXINT}

MAXINT – предопределенная целочисленная константа (объчно степень двойки) соответствующая наибольшему целому значению котрое может быть представлено на данной Паскаль-машине. Переменным объявленным с типом INTEGER может быть присвоено одно из этих значений.

VAR

I,J: INTEGER;

Операторами для типа INTEGER являются операторы сравнения и:

+ - \* DIV MOD

для прибавления, вычитания, умножения, деления с остатком и остатка от деления соответственно.

Добавление, вычитание и умножение могут вызывать *переполнение (*см. ниже), а DIV и MOD – не могут. Операторы + и – могут использоваться как унарные префиксы и как бинарные инфиксные операторы. Таким образом, выражение J – I может быть записано как –I + J. Деление с остатком обычно округляет до нуля как в следующем примере:

|  |  |
| --- | --- |
| Операция | Значение |
| 5 DIV 3  -7 DIV 3  10 DIV –3  -5 DIV -2 | 1  -2  -3  2 |

В выражении I DIV J результат неопределен, если значение J нуль, результат равен нулю, если модуль I меньше модуля J. Иначе знак результата положительняй, если I и J имеют тот же знак и отрицательный, если у I и J знак различный.

Результат выражения I MOD J неопределен, если значение J меньше или равно нулю, иначе он равен остатку от деления I на J. Например:

|  |  |
| --- | --- |
| Операция | Значение |
| 5 MOD 3  -7 MOD 3 | 2  -1 |

Остаток от деления определяется через деление с остатком, например для –7 MOD 3:

-7 – (-7 DIV 3\*3)

Нормальные правила вычисления арифметических выражений действую и здесь: операции со скобками вычисляются в первую очередь, умножение и деление имеют приоритет перед сложением и вычитанием, операторы равного приоритета вычисляются слева направо.

Таким образом, значение предыдущего выражения будет:

|  |  |
| --- | --- |
| Выражение | Вычисление |
| -7 – (-7 DIV 3 \* 3)  -7 – (-2\*3)  -7 – (-6)  -1 | начинаем с выражения в скобках  DIV имеет тот же приоритет что и \*  Вычисляем выражение в скобках |

Информация о целочисленных операторах обобщена в таблице ниже.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Оператор | Domain | Range | Ассоциативность | Приоритет |
| унарный – +  \* DIV MOD  + –  сравнения | integer  integer x integer  integer x integer  integer x integer | integer  integer  integer  boolean | правая  левая  левая  нет | наивысший  низший |

**7.3.1. Подсчет пробелов.**

В разделе 13.1 ContingBlanksInText выполняла подсчет пробелов во входном файле с использованием модуля счетчика.

PROGRAM CountingBlanksInText(INPUT, OUTPUT);

VAR

Ch, X100, X10, X1: CHAR;

{Включить модуль счетчика}

BEGIN {CountingBlanksInText}

Start; {обнулить счетчик}

WHILE NOT EOF

DO

BEGIN

WHILE NOT EOLN

DO

BEGIN

READ(Ch);

IF Ch = ‘ ‘

THEN

BEGIN

Bump; {Увеличиваем счетчик на едеинцу}

Ch := ‘#’;

END;

WRITE(Ch);

END;

READLN;

WRITELN

END;

WRITELN;

Value(X100, X10, X1); {получаем значение счетчика}

IF (X100 = ‘9’) AND (X10 = ‘9’) AND (X1 = ‘9’)

THEN

WRITELN(‘Количество пробелов как минимум 999’)

ELSE

WRITELN(‘Количество пробелов ’, X100, X10, X1)

END. {CountingBlanksInText}

Модуль определял абстракцию счетчика состоящего из трех переменных. Start присваивала символ 0 каждой из трех переменных, Bump изображала добавление единицы к к символьному представлению целочисленного значения, Value возвращала значение счетчика. Эти процедуры добавляли порядка 50 строк к программе, делая ее общую длину более 75 строк. С использованием типа данных INTEGER счетчик может быть реализован как переменная программы.

VAR

Counter: INTEGER;

а процедуры Start и Bump становятся операторами присваивания:

Counter := 0; {Start}

Counter := Counter + 1; {Bump}

Value заменяется ссылкой на Counter и Counter может быть сравнен с целой константой 999. Полная программа приведена ниже (теперь каких-то 25 строк).

PROGRAM CountingBlanksInText(INPUT, OUTPUT);

VAR

Ch: CHAR;

Counter: INTEGER;

BEGIN {CountingBlanksInText}

Counter := 0; {обнулить счетчик}

WHILE NOT EOF

DO

BEGIN

WHILE NOT EOLN

DO

BEGIN

READ(Ch);

IF Ch = ‘ ‘

THEN {Увеличиваем счетчик на единицу}

Counter := Counter +1;

WRITE(Ch);

END;

READLN;

WRITELN

END;

WRITELN;

IF Counter >= 999

THEN

WRITELN(‘Количество пробелов как минимум 999’)

ELSE

WRITELN(‘Количество пробелов ’, Counter)

END. {CountingBlanksInText}

INPUT:

Now is

the time for

all good men.

OUTPUT:

Now is

the time for

all good men.

Количество пробелов 9

INPUT:

OUTPUT:

Количество пробелов 0

Тест на более чем трехзначные значения был оставлен, чтобы программа была похода на ту которая использовала модуль счетчика, хотя с целочисленной переменной действительным ограничением является MAXINT. Частной и небезопасной практикой является игнорирование существования MAXINT и надежда на то, что Паскаль-машина сама разберется с этими проблемами.

**7.3.2. Ввод-вывод целочисленных значений.**

Вывод в CountingBlanksInText выглядит несколько ст ранно – множество пробелов отделяет конец строки. Количество пробелов зависит от выведенного целочисленного значения.

Когда целые значения выводятся в файлы типа TEXT, они преобразуются в символы и выравниваются по правой границе строки с фиксированной длиной (обычно длиной 10 символов). Лидирующие нули удаляются. Существует разница между операторами WRITE(‘5’) и WRITE(5). Первый добавляет строку †5† в OUTPUT, тогда как второй добавляет строку † 5†, потому что это были константы типов CHAR и INTEGER соответственно.

Паскаль имеет возможность управлять размером поля, в котором печатается значение типа INTEGER. Любые целые выражения в операторе WRITE могут быть дополнены двоеточием и целочисленным выражением которое обозначает размер поля для выражения. Например, если X и Y целые переменные со значениями 123 и 5, соответственно, тогда оператор WRITE(X:Y) добавляет строку † 123† в OUTPUT.

Последовательные целочисленные значения могут быть считаны из файла типа TEXT если они разделены пробелами или маркерами строк. Если Int1 целочисленная переменная, выражение READ(Int1) будет ожидать последовательность символов в INPUT, которая образует знаковое целое. Строка символов будет преобразована в целое значение перед сохранением в Int1. Синтаксис таких последовательностей символов в INPUT такой же как синтаксис констант типа INTEGER в Паскаль-программах.

<знаковое целое> ::= <знак><беззнаковое целое>| <беззнаковое целое>

<знак> ::= + | –

<беззнаковое целое> ::= <последовательность цифр>

<последовательность цифр> ::= <последовательность цифр><цифра> | <цифра>

<цифра> ::= 0|1|2|3|4|5|6|7|8|9

Когда целочисленная переменная появляется в операторе READ, пробелы и маркеры, если необходимо, пропускаются пока не будет встречен непробельный символ. Оператор READ принимает те символы, которые могут быть частью целочисленной константы включая предшествующие знаки и останваливается перед первым символом, который не может быть частью этой константы. Целочисленное значение, соответствующее принятым символам сохраняется в переменной.

Например, для данной строки в INPUT:

† -12 3/32/†

где / представляет маркер конца строки, и программного фрагмента

VAR

I1, I2, I3 : INTEGER;

BEGIN

READ(I1, I2, I3);

…

выполнение оператора READ произведет состояние выполнения со следующим частичным содержанием:

{I1•-12, I2•3, I3•32}

При чтении файла типа TEXT, содержащего неизвестное количество целых чисел, финальный маркер конца строки остается за последним считанным целым числом. Таким образом, EOF не может быть использован для определения завершения, как по ошибке пытается делать программа ReadInts:

PROGRAM ReadInts(INPUT, OUTPUT);

{Ошибочная попытка считать все целые числа в INPUT}

VAR

Val: INTEGER;

BEGIN {ReadInts}

WHILE NOT EOF

DO

BEGIN

READ(Val);

WRITELN(Val)

END

END. {ReadInts}

После того как последнее целое будет считано, остается маркер конца строки и EOF будет все еще FALSE, поэтому программа пытается считать снова когда целых чисел уже нет, значение оператора READ неопределено.

Следующий вариант лучше:

PROGRAM ReadInts2(INPUT, OUTPUT);

{Читать файл с целыми числами обращая внимание

на последний маркер конца строки }

VAR

Val: INTEGER;

BEGIN {ReadInts2}

WHILE NOT EOF

DO

BEGIN

IF EOLN

THEN

READLN;

IF NOT EOF

THEN

BEGIN

READ(Val);

WRITELN(Val)

END

END

END. {ReadInts2}

Однако и эта программа выдаст ошибку, если финальный маркер конца строки не следует непосредственно за последним числом, например для строки †123x† или †123 †. Легче читать целые числа из файла типа TEXT с использованием охраняющего значения для обозначения, когда входные данные закончились (например –99999 или 0, какое-то значение, которое недопустимо в данном файле).

Для иллюстрации вычислений с целыми числами, следующая программа получает целое число N из входного файла, потом вычисляет сумму первых N целых чисел:

PROGRAM Sum (INPUT, OUTPUT);

{(N >= 0 -> I, S := N+1, 1+2+…+ N)}

VAR

I, N, S: INTEGER;

BEGIN {Sum}

READ(N);

S := 0;

I := 1;

WHILE I <= N

DO

BEGIN

S := S+I;

I := I+1;

END;

WRITELN(‘Сумма первых’, N:2

‘ положительных целых чисел’, S:3 )

END. {Sum}

Выполнение:

INPUT: 6

OUTPUT: Сумма первых 6 положительных целых чисел 21

**7.3.3. Переполнение.**

Целочисленные операции Паскаля производят корректные результаты только тогда, когда операнды и результа этих операций находится в пределах [-MAXINT, MAXINT]. Для операции сложения, график показывает пунктиром область корректности результатов для X + Y.

MAXINT

MAXINT

- MAXINT

- MAXINT

За пределами границ обозначенных MAXINT, сложение не производит корректных результатов. Из графика видно, что для двух произвольно выбранных целых чисел в одном случае из четырех результат для целочисленных операций моежт оказаться некорректным – ¼ часть области ограниченной +/- MAXINT не производит корректные результаты.

Аналогично для корректных результатов операции X-Y показанных на следующем графике.

MAXINT

MAXINT

- MAXINT

- MAXINT

Снова здесь вероятность ¼ что произойдет переполнение с двумя произвольными целыми числами.

Поскольку умножение может произвести большие знаечния легче, чем сложение и вычитание, вероятность преполнения здесь выше. Область корректных результатов для операции X\*Y представлена на графике ниже.

MAXINT

MAXINT

- MAXINT

- MAXINT

(1, 0)

Границы области представлены гиперболами, график выполнен с искажением масштаба. На самом деле площадь области корректных результатов существенно меньше. В одном квадранте, площадь подкривой Y = MAXINT/X задается:

= MAXINT + MAXINT ln MAXINT – MAXINT ln 1

Таким образом, площадь подкривой будет MAXINT + MAXINT ln MAXINTб тогда как площадь области ограниченной операндами MAXINT2, поэтому в случае произвольного выбора операндов, переполнение не произойдет с вероятностью:

 = 

Для MAXINT = 216 = 65535, вероятность не переполнения в случае произвольного выбора операндов будет:



или менее 2 на 10000.

В противоположность сложению, вычитанию и умножению, DIV и MODне производят переполнения:

Очевидно, что тип INTEGER представляет только небольшую, частичную модель математических целых чисел. При соблюдении ограничений операции будут отлично выполняться, но Паскаль-машина не занимается соблюдением таких ограничений, это должен делать программист.

Реализация работы с MAXINT зависит от реализации Паскаль-машины. То, что значение MAXINT может быть определено опытным путем на каждой машине можно скорее отнести к недостаткам Паскаля.Поэтому для разработки стабильного кода нужно быть внимательным. На некоторых машина вычисления останавливаются при переполнении, поэтому его можно безопасно игнорировать, пока программа не начнет давать сбои. Однако на большистве машин результат просто не будет соотвествтвовать значению соотвествующей арифметической операции.

**7.3.4. Преобразование символов в числа.**

Рассмотрим строку цифр длиной N:

c1, c2, …, cN-1, cN

Соответствие между строкой и числом, которое она представляет, задается формулой:



где B – основание системы исчисления, а c – численное значение *i*-го символа в строке.

Для числа 0101 получаем следующие десятичные значения в выбранных системах исчисления:

01012 = 0x23 + 1x22 + 0x21+1x20 = 0 + 4 + 0 + 1 = 510

01018 = 0x83 + 1x82 + 0x81+1x80 = 0 + 64 + 0 + 1 = 6510

010110 = 0x103 + 1x102 + 0x101+1x100 = 0 +100 + 0 + 1 = 10110

010116 = 0x163 + 1x162 + 0x161+1x160 x= 0 + 256 + 0 + 1 = 25710

Цифры систем счисления больше 10 обычно представляются буквами, например цифры шестнадцатиричной системы исчисления следующие:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Лучшая стратегия поиска десятичного значения выражения в системе исчисления с основанием B:

c1 x BN-1 + c2 x BN-2 + … + cN x B0

умножать его, используя правило Хорнера (Horner’s rule)

(…(c1 x B + c2) x B + c3) … x B + cN

Процедура StringToInt преобразовывает строки символов в системе счисления представленной Base в целочисленные значения в Result, используя правило Хорнера.

**DP 1**

PROCEDURE StringToInt (VAR F: TEXT; VAR Base, Result: INTEGER);

{F=<,x#y, R>, где x – строка цифр, Base – целое число,

2 <= Base <= 16, задающее их систему исчисления -->

F, Result := <x#, y, R>, N(x), где N – целое значение,

имеющее символьное представление x}

VAR<

Ch: CHAR;

Digit: INTEGER;

BEGIN {StringToInt}

Result := 0;

READ(F, Ch);

WRITE(Ch);

WHILE(Ch <> ‘#’) AND (Ch <> ‘ ’)

DO

BEGIN

{Digit := число, соотвествующее Ch в Base}

Result := Result \* Base + Digit;

READ(F, Ch);

WRITE(Ch)

END

END. {StringToInt}

Код для нахождения целого, соответствующего Ch простой, но длинный, поэтому он оформлен в виде процедуры.

**DP 1.1.**

PROCEDURE CharToDigit (VAR Ch: CHAR; VAR Result: INTEGER);

{(‘0’ <= Ch <= ‘9’ OR ‘A’ <= Ch <= ‘F’ -->

Result := число соответствующее Ch ) |

(NOT (‘0’ <= Ch <= ‘9’ OR ‘A’ <= Ch <= ‘F’) -->

Result := 0)}

BEGIN { CharToDigit}

IF Ch = ‘0’ THEN Result := 0 ELSE

IF Ch = ‘1’ THEN Result := 1 ELSE

IF Ch = ‘2’ THEN Result := 2 ELSE

IF Ch = ‘3’ THEN Result := 3 ELSE

IF Ch = ‘4’ THEN Result := 4 ELSE

IF Ch = ‘5’ THEN Result := 5 ELSE

IF Ch = ‘6’ THEN Result := 6 ELSE

IF Ch = ‘7’ THEN Result := 7 ELSE

IF Ch = ‘8’ THEN Result := 8 ELSE

IF Ch = ‘9’ THEN Result := 9 ELSE

IF Ch = ‘A’ THEN Result := 10 ELSE

IF Ch = ‘B’ THEN Result := 11 ELSE

IF Ch = ‘C’ THEN Result := 12 ELSE

IF Ch = ‘D’ THEN Result := 13 ELSE

IF Ch = ‘E’ THEN Result := 14 ELSE

IF Ch = ‘F’ THEN Result := 15

ELSE

BEGIN

Result := 0

END

END. { CharToDigit}

Для тестирования этого проекта он помещается в программу, которая сначала считывает основание системы счисления в десятичном формате, в котором должна быть интерпретирована следующая строка.

PROGRAM Convert (INPUT, OUTPUT);

VAR

Base, Number: INTEGER;

{Включить PROCEDURE StringToInt

(VAR F: TEXT; VAR Base, Result: INTEGER);

F=<,x#y, R>, где x – строка цифр, Base – целое число,

2 <= Base <= 16, задающее их систему исчисления -->

F, Result := <x#, y, R>, N(x), где N – целое значение,

имеющее символьное представление x}

BEGIN {Convert}

READ(Base);

StringToInt(INPUT, Base, Number);

WRITELN;

WRITELN(‘Десятичное значение:’, Number)

END. {Convert}

Выполнение:

INPUT: 8 0101#

OUTPUT: 8 0101#

Десятичное значение: 65

INPUT: 16 1F#

OUTPUT: 16 1F#

Десятичное значение: 31

**7.4. Тип диапазон.**

Когда переменная должна принимать значения в определенном диапазоне значений, он может быть ограничен до поддиапазона другого порядкового типа. Это ограничение помогает находить ошибки выхода за пределы допустимого диапазона.

Новые типы Паскаля также могут быть образованы указанием значений, которые лежат внутри поддиапазона значений существующего порядкового типа. Поддиапазон определяется указанием минимальной и максимальной констант, которые входят в поддиапазон.

<тип диапазон> ::= <константа> .. <константа>

<константа> ::= <знак> <беззнаковое число> | <беззнаковое число>

| <идентификатор константы> | <строка символов>

Рассмотрим следующее объявление:

TYPE

Letter = ‘A’ .. ‘Z’;

SmallInt = 0 .. 100;

SmallerInt = 1 .. 10;

VAR

L: Letter;

X: SmallInt;

Y: SmallerInt;

Тип, которому принадлежат константы, задающие диапазон, называется *владеющим типом* (host type). Все операции владеющего типа могут быть применены к операндам его поддиапазона. В примере выше владеющим типом для Letter является CHAR, а для SmallInt и SmallerInt – INTEGER. Таким образом, символьные операции могут быть применены к L, а целочисленные к SmallInt и SmallerInt.

BEGIN

...

IF L = ‘A’ {сравниваем Letter и CHAR}

THEN

...

WRITE(X+Y); {складываем целые значения}

...

END

После определения перечислимого типа, его константы могут быть использованы для определения диапазонов, владеющим типом которых является перечислимых тип. Объявление таких новых типов не ведут к дублированию идентификаторов, использованых при определении диапазона.

TYPE

DayOfWeek = (Monday, Tuesday, Wednesday, Thursday,

Friday, Saturday, Sunday);

WeekDay = Monday .. Friday;

WeekEndDay = Saturday, Sunday;

**7.4.1. Присваивания для диапазонов.**

Не всегда возможно определить заранее будет ли выражение содержащее значения из диапазона нарушать его границы. Рассмотрим следующий оператор присваивания, включающий переменную X типа SmallInt и Y типа SmallerInt, объявленные выше.

Y := Y + X

Поскольку владеющий тип в данном случае INTEGER, операция + допустима для операндов Y и X. Сложение дает значение типа INTEGER, которое может быть присвоено Y, как значение владеющего типа. Однако если результат сложения за пределами объявленного диапазона значений для Y, т.е. 0 ... 10, работа программы завершится с сообщением об ошибке.

Тип диапазонов требуют переопределения значения оператора присваивания, таким образом, что функция оператора присваивания не определена, когда значение выражения находится за пределами диапазона переменной в левой части оператора.

V := E = {<s, t>: E(s) ∈ Vals(V), t = (s – {<V, c>: c ∈ Vals(V)}) ∪ {<V, E(s)>}}

где Vals(V) – множество всех значений типа V.

**7.4.2. Преобразование температур.**

Следующий пример преобразует температуры в точке кипения воды из градусов по Фаренгейту в градусы Цельсия. Обе переменные могли быть объявлены как целые, однако знание проблемной области предлагает значимое ограничение значений.

PROGRAM Temperature(INPUT, OUTPUT);

VAR

Farenheit: -459 .. 212;

Celsius: -273 .. 100;

BEGIN {Temperature}

READ(Farenheit);

Celsius := (Farenheit – 32)\*5 DIV 9;

WRITELN(Farenheit:4, ‘degrees Farenheit or ’,

Celsius:4, ‘degrees Celsius’)

END. {Temperature}

Выполнение:

INPUT: 212

OUTPUT: 212 degrees Farenheit or 100 degrees Celsius

INPUT: 32

OUTPUT: 32 degrees Farenheit or 0 degrees Celsius

INPUT: -459

OUTPUT: -459 degrees Farenheit or –272 degrees Celsius

INPUT: 1000

OUTPUT: Value 1000 is out of range.

Последний выход был произведен не программой, а Паскаль-машиной, которая определила, что 1000 за пределами допустимого диапазона для значений типа Farenheit.

Тип диапазон помогает документировать программу и сдвигать груз проверок допустимости присваиваний на Паскаль-машину. Этот часто приводит к более компактным, хорошо читаемым программам. Дополнительно, значения типа диапазон могут занимать меньше памяти, чем аналогичные значения владеющего типа.

**7.5. Заключение.**

В этой главе были рассмотрены порядковые типы данных Паскаля. Операции, которые могут быть применены к каждому типу, обобщены в следующей таблице.

|  |  |  |
| --- | --- | --- |
| **Оператор** | **Операция** | **Функциональность** |
| NOT | инверсия | логический -> логический |
| AND | коньюнкция | логический x логический |
| OR | дизъюнкция |  |
| = | эквивалентность | порядковый x порядковый -> логический |
| <> | неравенство |  |
| < | меньше |  |
| <= | меньше или равно |  |
| > | больше |  |
| >= | больше или равно |  |
| + | унарный плюс | целый -> целый |
| - | унарный минус |  |
| + | бинарное сложение | целый x целый -> целый |
| - | бинарное вычитание |  |
| \* | умножение |  |
| DIV | целочисленное деление |  |
| MOD | остаток от деления |  |

Каждый порядковый тип используется по-своему. Переменные типа BOOLEAN могут быть использованы для хранения сложных условий для дальнейшего использования. Переменные типа INTEGER позволяют легко выполнять подсчет при условии, что выполняется ограничение [-MAXINT, MAXINT]. Перечислимые типы хороши, когда нужно зафиксировать небольшой набор значений, каждое со своим мнемоническим именем. Типы диапазоны позволяют программисту объявлять границы их значений, чтобы они проверялись автоматически.

Методы анализа, использованные в CF Pascal, расширены до использования с порядковыми типами без изменений.