**Совместимость и преобразование типов данных**

Курсовой проект по программированию

Составила: Ирина Комарова IT2V

Таллинн 2004 г.

**Введение**

Язык программирования Pascal был разработан в 1968-1971 гг. Никлаусом Виртом в Цюрихском Институте информатики (Швейцария), и назван вчесть Блеза Паскаля – выдающегося математика, философа и физика 17-го века. Первоначальная цель разработки языка диктовалась необходимостью создания инструмента "для обучения программированию как систематической дисциплине". Однако очень скоро обнаружилась чрезвычайная эффективность языка Pascal в самых разнообразных приложениях, от решения небольших задач численного характера до разработки сложных программных систем - компиляторов, баз данных, операционных систем и т.п. К настоящему времени Pascal принадлежит к группе наиболее распространенных и популярных в мире языков программирования:

• существуют многочисленные реализации языка практически для всех машинных архитектур;

• разработаны десятки диалектов и проблемно-ориентированных расширений языка Pascal;

• обучение программированию и научно-технические публикации в значительной степени базируются на этом языке.

**Описание типов данных**

Достоинством языка Паскаль является возможность использования широкого набора разных типов данных. Тип данных определяет возможные значения констант, переменных, функций, выражений и операций, которые могут выполняться над ними.

Типы данных подразделяются на простые и сложные. Простые типы делятся на стандартные (предопределенные) типы и типы определяемые пользователем (пользовательские типы).

Имена стандартных типов являются идентификаторами и действуют в любой точке программы. Они описаны в стандартном модуле System. Так же, как и другие идентификаторы, имена стандартных типов могут быть переопределены в программе.

Однако остается возможность обращения к их первоначальному смыслу с помощью квалифицируемого идентификатора с указанием имени модуля System. Например: System.Integer, System.Real.

К стандартным типам относятся:

• группа целых типов (Shortint, Integer, Longint, Byte, Word);

• группа вещественных типов (Single, Real, Double, Extended, Comp);

• группа логических (булевских) типов (Boolean, ByteBool, WordBool, LongBool);

• символьный тип (Char);

• строковый тип (String, Pchar);

• указательный тип (Pointer);

• текстовый тип (Text).

Символьный тип, целые и булевские типы относят к, так называемым, порядковым типам.

Порядковые типы характеризуются следующими свойствами:

1. Множество допустимых значений любого порядкового типа представляет собой упорядоченную последовательность , каждый элемент которой имеет свой порядковый номер. Порядковый номер представляется целым числом. Первое значение любого порядкового типа имеет номер 0, следующий номер 1 и т.д. Исключение составляют порядковые типы Integer, Shortint, Longint, где порядковым номером значений этих типов является само значение.

2. К любому значению порядкового типа можно применять функции возвращающие номер, предыдущее или последующее значение данного типа.

Пользовательские типы – дополнительные абстрактные типы, характеристики которых программист может определять самостоятельно.

К пользовательским типам относятся:

• перечисляемый тип;

• интервальный тип;

• указательные типы (кроме стандартного типа Pointer);

• структурированные типы;

• процедурный тип.

Перечисляемый и интервальный типы являются порядковыми.

Структура подраздела описания типов выглядит следующим образом:

Форма записи:

type = ;

Пример:

type

vec = integer;

bool = boolean;

Стандартные функции языка Паскаль

Для выполнения часто встречающихся операций и преобразований данных, относящихся к разным типам, существуют заранее определенные функции, которые называются СТАНДАРТНЫМИ. Для обращения к функции необходимо задать ее имя и в скобках список аргументов (параметров).

Прежде чем перейдем к стандартным функциям, сначала ознакомимся с правилами их использования:

1. Имя функции записывается прописными буквами латинского алфавита.

2. Аргумент функции записывается в круглых скобках после имени функции.

3. Аргументом функции может быть константа, переменная, или арифметическое выражение того же типа

Теперь рассмотрим некоторые стандартные функции:

Функция Действие Тип Х Тип возвращаемого значения

SQRT(X) вычисляет квадратный корень из аргумента Х действительный действительный

SQR(X) вычисляет квадрат аргумента Х целый действи-тельный целый действи-тельный

RANDOM(X) возвращает случайное число, перед вызовом функции желательно использовать в программе оператор RANDOMIZE включающей случайную инициализацию генератора случайных чисел целый, положительный соответствует типу переменной принимающей значение

SIN(X) вычисляет синус аргумента Х действительный действительный

COS(X) вычисляет косинус аргумента Х действительный действительный

ABS(X) вычисляет абсолютное значение (модуль) аргумента Х целый действи-тельный целый действи-тельный

ODD(X) проверяет Х на четность длинное целое логический

ORD(X) определяет порядковый номер символа Х любой тип кроме действительного длинное целое

CHR(X) определяет символ стоящий по порядковому номеру Х byte символьный

PRED(X) определяет предыдущее значение по отношению к Х любой тип кроме действительного тот же тип

SUCC(X) определяет последующее значение по отношению к Х любой тип кроме действительного тот же тип

ARCTAN(X) вычисляет арктангенс аргумента Х действительный действительный

EXP(X) вычисляет экспоненту от аргумента Х действительный действительный

LN(X) вычисляет натуральный логарифм от Х действительный действительный

TRUNC(X) находит целую часть от Х действительный длинное целое

ROUND(X) округляет Х в сторону ближайшего целого действительный длинное целое

INT(X) возвращает целую часть аргумента Х действительный действительный

FRAC(X) возвращает дробную часть аргумента Х действительный действительный

DEC(X,N) уменьшает значение переменной Х на заданное число N любой тип кроме действительного тот же тип

INC(X,N) увеличивает значение переменной Х на заданное число N любой тип кроме действительного тот же тип

PI возвращает значение числа - действительный

Примеры:

1. ORD(‘R’)=82; ORD(5)=5;

2. CHR(68)=’D’; можно вызывать эту функцию через #, если аргумент функции константа (#68='D');

3. PRED(‘N’)=’M’; PRED(87)=86;

4. SUCC(‘S’)=’T’; SUCC(87)=88;

5. PI=3.141592653897932385;

6. ROUND(3.1415)=3;

7. LN(1)=0.000;

8. SQRT(36)=6.000;

9. SIN(90\*pi/180)=1.000.

Замечание:

В тригонометрических функциях аргумент должен быть задан только в радианной мере угла.

Совместимость и преобразование типов данных.

Турбо-Паскаль - типизированный язык, следовательно, все применяемые операции определены только над операндами совместимых типов.

Два типа считаются совместимыми, если

• оба они есть один и тотже тип.

• один тип есть тип-диапазон второго типа.

• оба они являются типами-диапазонами одного и того же базового типа.

• один тип есть строка, а другой - строка или символ.

• оба они есть процедурные типы с одинаковым типом результата (для типа-функции), одинаковым количеством параметров и одинаковым типом взаимно соответствующих параметров.

Когда в тех или иных операциях или операторах присутствуют данные, относящиеся к различным типам, возникает вопрос о совместимости типов. В связи с этим говорят об идентичности типов, совместимости типов и совместимости типов для присваивания. Когда возникают проблемы с соответствием типов данных, можно осуществить преобразование тех или иных типов.

Идентичность типов.

Идентичность типов требуется от формальных параметров процедур и функций и соответствующих им фактических параметров во время вызова.

Два типа Т1 и Т2 идентичны в следующих случаях:

T1 и Т2 – один и тот же идентификатор типа (integer; real и т.д. и т.п.);

Один тип объявляется эквивалентным другому.

type

T1 = boolean;

T2 = T1;

T3 = boolean;

M1 = array [1..5] of integer;

M2 = array [1..5] of integer;

var

V1, V2 = array [1..10] of integer;

Так, типы Т1, Т2, Т3 и boolean – идентичны, а М1 и М2 – не идентичные типы, но тем не менее, переменные V1 и V2 – переменные идентичных типов.

Совместимость типов.

Совместимость типов требуется в выражениях (в том числе и в операциях отношения).

Два типа Т1 и Т2 идентичны в следующих случаях:

Т1 и Т2 – один и тот же тип или они идентичны;

Т1 и Т2 – вещественные типы;

Т1 и Т2 – целые типы;

Один тип – вещественный, а другой – целый;

Один тип представляет собой тип – диапазон другого;

Оба типа являются типами – диапазонами какого-то одного типа;

Оба типа являются типами – множествами с совместимыми базовыми типами;

Один тип является строкой, а другой – символом или строкой.

Совместимость для присваивания.

Эта совместимость необходима, когда значение какого-то выражения присваивается переменной, типизированной константе или функции. Если значение объекта типа Т2 присваивается объекту типа Т1, то это возможно в следующих случаях:

Т1 и Т2 – идентичные типы и не являются файловыми типами или структурированными типами, содержащими компоненты файлового типа на любом уровне структурированности;

Т1 и Т2 – совместимые порядковые типы и значение типа Т2 находится в границах возможных значений объекта типа Т1;

Т1 и Т2 – вещественные типы и значение типа Т2 находится в границах возможных значений объекта типа Т1;

Т1 – вещественный тип, а Т2 – целый;

Т1 и Т2 – строки;

Т1 – строка, а Т2 – символ;

Т1 и Т2 – совместимые типы – множества и все компоненты значения типа Т2 находятся в множестве Т1.

Преобразование типов в Паскале может быть явным и неявным. При явном преобразовании типов используются вызовы специальных функций Ord, Trunc, Round, Chr, Ptr (преобразует четырёхбайтный целочисленный аргумент к типу-указателю), аргументы которых принадлежат одному типу, а результат другому.

Преобразование может достигаться применением идентификатора (имени) стандартного типа, или определённого пользователем типа, в качестве идентификатора функции преобразования к выражению преобразуемого типа (так называемое автоопределённое преобразование типов). Например, допустимы следующие вызовы функций:

Type Mytype = (A, B, C, D);

. . . . . . . . . . . . . . . . .

Mytype (2);

Integer (D);

Pointer (Longint (A) + $FF);

Char (127 Mod C);

Byte (K);

При автоопределённом преобразовании типа выражения может произойти изменение длины его внутреннего представления (уменьшение или увеличение).

В Турбо-Паскале есть ещё один явный способ: в ту область памяти, которую занимает переменная некоторого типа, можно поместить значение выражения другого типа, если только длина внутреннего представления вновь размещаемого значения в точности равна длине внутреннего представления переменной. С этой целью вновь используется автоопределённая функция преобразования типов, но уже в левой части оператора присваивания:

Type

Byt = Array [1..2] Of Byte;

Int = Array [1..2] Of Integer;

Rec = Record

X: Integer;

Y: Integer;

End;

Var

VByt: Byt;

VInt: Int;

VRec: Rec;

Begin

Byt (VInt[1])[2]:= 0;

Int (VRec)[1]:= 256;

End.

Данные одного типа могут автоматически (неявно) преобразовываться в данные другого типа перед выполнением операций выражений.

Неявное преобразование типов возможно только в двух случаях:

• выражение из целых и вещественных приводится к вещественным

• одна и та же область памяти трактуется попеременно как содержащая данные то одного, то другого типа.

Совмещение данных может произойти при использовании записей с вариантами, типизированных указателей, содержащих одинаковый адрес, а также при явном размещении данных разного типа в одной области памяти (используется Absolute - за ним помещается либо абсолютный адрес, либо идентификатор ранее определённой переменной).

Абсолютный адрес - пара чисел, разделённых двоеточием - первое - сегмент, второе - смещение.

Пример:

B: Byte Absolute $0000:$0055;

W: Longint Absolute 128:0;

Если за Absolute указан идентификатор переменной, то происходит совмещение в памяти данных разного типа, причём первые байты внутреннего представления данных будут располагаться по одному и тому же абсолютному адресу:

Var

X: Real;

Y: Array [1..3] Of Integer Absolute X;

Эквивалентность типов

Относительно понятия эквивалентности типов существует несколько точек зрения. Рассмотрим три из них. Все они исходят из того, что эквивалентные типы должны допускать одинаковые последовательности операций.

Структурная эквивалентность

Два атрибута типа T1 и T2 называются (структурно) эквивалентными, если

• их базовые типы BT1 и BT2, соответственно, совпадают или

• BT1=arr(M,N,T1'), BT2=arr(M,N,T2') и T1' эквивалентен T2', или

• BT1=rec([F1:T11,...,Fn:T1n]), BT2=rec([F1:T21,...,Fn:T2n]) и T1i эквивалентен T2i для каждого i, или

• BT1=ref(T1'), BT2=ref(T2') и T1' эквивалентен T2' и

• предположение об эквивалентности T1 и T2 не противоречит условиям 1-4.

Несколько странное условие 5 связано с рекурсивностью типов. Оно делает отношение структурной эквивалентности наибольшим среди отношений, удовлетворяющих условиям 1-4.

П р и м е р:

Пусть

T1=rec([info:int,next:T1])

T2=rec([info:int,next:T2])

Применяя только правила 1-4, получим, что T1 и T2 эквивалентны, если T1 и T2 эквивалентны. Правило 5 заставляет сделать вывод, что T1 и T2 действительно эквивалентны (на основании только правил 1-4 можно сделать и обратный вывод).

Если бы не было ссылочных и, следовательно, рекурсивных типов (как в Фортране или Алголе 60), то определение структурной эквивалентности сводилось бы к условию 1, т.е. к равенству базовых типов.

При допущении же рекурсивных ссылочных типов для проверки структурной эквивалентности двух типов используется алгоритм нахождения всех пар эквивалентных состояний некоторого конечного автомата. Можно использовать следующий метод построения этого автомата.

Сначала строится праволинейная грамматика:

1. Для каждого описания идентификатора типа, переменной или параметра declared(I,B,Inf), где Inf равно type(T), var(T) или par(T), типу T ставится в соответствие новый нетерминал t.

2. Если нетерминалу t соответствует базовый тип arr(m,n,T1), то типу T1 ставится в соответствие новый нетерминал t1 и вводится правило t -> arr m n t1.

3. Если нетерминалу t соответствует базовый тип rec([f1:T1,...,fn:Tn]), то типу Ti ставится в соответствие новый нетерминал ti для каждого i и вводятся правила t -> ref i fi ti.

4. Если нетерминалу t соответствует базовый тип ref(T1), где T1=int или T1=real, то вводится правило t -> T1.

5. Если нетерминалу t соответствует базовый тип ref(tid(I,B)), а типу tid(I,B) уже сопоставлен нетерминал t1, то вводится правило t -> ^ t1.

Нетерминалы этой грамматики эквивалентны, если и только если соответствующие типы структурно эквивалентны.

Остаётся преобразовать эту грамматику к автоматному виду и применить алгоритм нахождения всех пар эквивалентных состояний.

Отношение структурной эквивалентности действительно является эквивалентностью, т.к. оно рефлексивно, транзитивно и симметрично. Кроме того, два типа структурно эквивалентны тогда и только тогда, когда они допускают одни и те же последовательности операций. Это делает такую эквивалентность простой и понятной программистам. Её основным и весьма существенным недостатком является сложный и громоздкий алгоритм проверки эквивалентности. Насколько известно автору, структурная эквивалентность была принята только в языке Алгол 68.

Предикат consist в этом случае определим следующим образом:

consist(T1,T2):base\_type(T1,BT1),base\_type(T2,BT2),

(BT1=int,BT2=real ; % приводимость

equiv(BT1,BT2) ; % эквивалентность

error("Несовместимые типы")).

Именная эквивалентность

При стандартизации языка Паскаль была принята именная эквивалентность. Согласно её определению эквивалентными могут быть только именованные типы, т.е. типы с атрибутами int, real или tid(\_,\_):

1. Именованный тип эквивалентен сам себе.

2. Если тип T=tid(I,B) имеет описание declared(I,B,T1), где T1 – именованный тип, то типы T и T1 эквивалентны.

Это очень ограничительное определение. Даже в Паскале допустимо присваивание

U:=V,

если переменные U и V описаны как

var U,V: array[1..10]of real,

хотя имеют неименованный тип "массив". Но в том же Паскале этот оператор недопустим из-за неэквивалентности типов переменных, если они описаны как

var U: array[1..10]of real;

var V: array[1..10]of real;

Чтобы охватить все эти случаи, компилятор для каждого вхождения выражения типа, начинающегося с array, record или ^ (т.е. не являющегося идентификатором типа), вводит уникальное имя типа – псевдоним, благодаря чему разные вхождения одного и того же выражения типа оказываются неэквивалентными в смысле именной эквивалентности.

В соответствии со сказанным следует внести изменения в правила DC-грамматики для нетерминала type, определяющие атрибут типа. В них включается теперь порождение и описание псевдонимов типа. Для генерации новых "имен" можно использовать самые разные методы; мы здесь воспользуемся предикатом recorda, генерирующем в качестве псевдонима уникальную ссылку на пустой терм, записываемый по ключу alias. описание этого псевдонима типа включается в виде предиката declared.

type(B,tid(A,B)) -->

[array,`[,n(M),`:,n(N),`],of],type(B,T),

{recorda(alias,\_,A),

assert(declared(A,B,type(arr(M,N,T)))}.

type(B,tid(A,B)) -->

[record],field(B,F),fields(B,LF),

{correct(F,LF),

recorda(alias,\_,A),

assert(declared(A,B,type(rec([F|LF])))},

[end].

type(B,tid(A,B)) -->

[`^,id(I)],

{(type\_id2(I,B,B1,type(\_));

assert(declared(I,B,type(referred))),

B1=B),

recorda(alias,\_,A),

assert(declared(A,B,type(ref(I,B1)))}.

Предикат consist в этом случае определяется следующим образом:

consist(T1,T2):T1=int,T2=real ; % приводимость

equiv(T1,T2) ; % эквивалентность

error("Несовместимые типы").

equiv(T,T).

equiv(tid(I,B),tid(I1,B1)):declared(I,B,type(tid(I1,B1)));

declared(I1,B1,type(tid(I,B))).

Именная эквивалентность сравнительно просто реализуется. Но это – отношение (рефлексивное и симметричное) не транзитивно, и поэтому не является эквивалентностью ни в математическом, ни в привычном, обыденном смысле. Понятие псевдонима типа обычно не даётся программистам, Поэтому начинающие программисты на Паскале часто делают ошибки, вроде указанных в примере. Мотивацией для введения именной эквивалентности в 1970-е годы послужило желание избежать ошибок программирования, вроде присваивания "яблокам" "крабов", когда и те, и другие описаны как целые. С развитием объектно-ориентированного программирования подобные ухищрения стали излишними, а именная эквивалентность осталась в некоторых языках как анахронизм.

Структурно-именная эквивалентность

Этот тип эквивалентности самый простой: эквивалентными считаются типы, имеющие одинаковый базовый тип. Предикат consist в этом случае определяется следующим образом:

consist(T1,T2):base\_type(T1,BT1),base\_type(T2,BT2),

(BT1=int,BT2=real ; % приводимость

BT1=BT2 ; % эквивалентность

error("Несовместимые типы")).

При этом нет необходимости вводить псевдонимы типа, как в случае именной эквивалентности. Вместо предиката acc\_type при анализе доступа можно применять предикат base\_type.

Отношение структурно-именной эквивалентности рефлексивно, симметрично и транзитивно. По вложению оно лежит строго между структурной и именной эквивалентностями. Им легко пользоваться на практике.

Примеры

1.

С помощью несложной программы мы сможем узнать внутренний код произвольного символа.

Program Code\_pf\_Char;

{Программа читает символ с клавиатуры и выводит на экран

этот символ и соответствующий ему внутренний код}

var

ch: Char; {В эту переменную читается символ}

begin

Write('Введите любой символ: ');

ReadLn(ch); {Читаем один символ}

WriteLn(ch, ' = ',ord(ch)); {Преобразуем его к целому и выводим на экран}

END.

Обращаем внимание: при вызове

WriteLntch,' = ',ord(ch));

третьим параметром обращения указан вызов функции ORD (СН) , что с точки зрения языка является выражением; во многих случаях при вызове процедур и функций в качестве параметров вызова можно указывать не только переменные или константы, но и выражения с их участием.

2.

Давайте посмотрим, как можно реализовать на универсальном языке программирования простейшие инструкции лингвиста (даже не используюшие этих понятий), касающиеся этапа преобразования орфографического текста в фонетическую транскрипцию. Далее приведен пример задачи и ее решения на языке Паскаль.

Непроизносимые гласные:

Сочетания "вств" и "стн" транскрибируются как [ств] и [сн] соответственно, т. е.:

1) /вств/ -> [ств]

2) /стн/ -> [лнц]

Реализация двух данных правил на языке Паскаль:

program transcription;

var

Word: String;

I, J, K: Integer;

begin

Write('Введите слово: '); ReadLn(Word);

K := Pos('вств', word);

while (K 0) do

begin

delete(Word, K, 1);

K := Pos('вств', Word);

end;

K := Pos('стн', Word);

while (K 0) do

begin

delete(Word, K+1, 1);

K := Pos('стн', Word);

end;

WriteLn('Транскрипция: ', Word);

end.

Как видно из примера, два правила, изложенные в лингвистическом описании, реализуются программой в двадцать с лишним строк. Нетрудно заметить, что а) такая запись очень громоздка, б) она трудна для восприятия, в) она совершенно не имеет ничего общего с записью, естественной для лингвиста и поэтому плохо отражает суть происходящих преобразований. А если бы мы чуть-чуть усложнили правила, введя несколько слов-исключений, требуя проверить, не является ли транскрибируемое слово производным от одного из них, программа увеличилась бы в два раза или более.