Основные понятия алгоритмического языка

1. О С Н О В Н Ы Е П О Н Я Т И Я

А Л Г О Р И Т М И Ч Е С К О Г О Я З Ы К А

 СОСТАВ ЯЗЫКА. Обычный разговорный язык состоит из четырех основных

элементов: символов, слов, словосочетаний и предложений. Алгоритми-

ческий язык содержит подобные элементы, только слова называют элемен-

тарными конструкциями, словосочетания-выражениями, предложения-опера-

торами. Символы, элементарные конструкции, выражения и операторы

составляют иерархическую структуру, поскольку элементарные конструк-

ции образуются из последовательности символов, выражения-это последо-

вательность элементарных конструкций и символов, а оператор-последо-

вательность выражений, элементарных конструкций и символов.

 ОПИСАНИЕ ЯЗЫКА есть описание четырех названных элементов. Описание

символов заключается в перечислении допустимых символов языка. Под

описанием элементарных конструкций понимают правила их образования.

Описание выражений-это правила образования любых выражений, имеющих

смысл в данном языке. Описание операторов состоит из рассмотрения

всех типов операторов, допустимых в языке. Описание каждого элемента

языка задается его СИНТАКСИСОМ и СЕМАНТИКОЙ. Синтаксические определе-

ния устанавливают правила построения элементов языка. Семантика опре-

деляет смысл и правила использования тех элементов языка, для которых

были даны синтаксические определения.

 СИМВОЛЫ языка-это основные неделимые знаки, в терминах которых пи-

шутся все тексты на языке.

 ЭЛЕМЕНТАРНЫЕ КОНСТРУКЦИИ -это минимальные единицы языка, имеющие

самостоятельный смысл. Они образуются из основных символов языка.

 ВЫРАЖЕНИЕ в алгоритмическом языке состоит из элементарных конс-

трукций и символов, оно задает правило вычисления некоторого значе-

ния.

 ОПЕРАТОР задает полное описание некоторого действия, которое необ-

ходимо выполнить. Для описания сложного действия может потребоваться

группа операторов. В этом случае операторы объединяются в СОСТАВНОЙ

ОПЕРАТОР или БЛОК.

 Действия, заданные операторами, выполняются над ДАННЫМИ. Предложе-

ния алгоритмического языка, в которых даются сведения о типах данных,

называются ОПИСАНИЯМИ или неисполняемыми операторами.

 Объединенная единым алгоритмом совокупность описаний и операторов

образует ПРОГРАММУ на алгоритмическом языке.

 В процессе изучения алгоритмического языка необходимо отличать ал-

горитмический язык от того языка, с помощью которого осуществляется

описание изучаемого алгоритмического языка. Обычно изучаемый язык на-

зывают просто языком, а язык, в терминах которого дается описание

изучаемого языка - МЕТАЯЗЫКОМ.

 Синтаксические определения могут быть заданы формальными или не-

формальным способами. Существуют три формальных способа:

 -металингвистическая символика, называемая Бэкуса-Наура формулами;

 -синтаксические диаграммы;

 -скобочные конструкции.

 Мы в последующем изложении будем пользоваться неформальным спосо-

бом.

2. О С Н О В Н Ы Е С И М В О Л Ы

 Основные символы языка-буквы, цифры и специальные символы-состав-

ляют его алфавит. ТУРБО ПАСКАЛЬ включает следующий набор основных

символов:

 1) 26 латинских строчных и 26 латинских прописных букв:

 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

 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

 2) \_ подчеркивание

 3) 10 цифр:

 0 1 2 3 4 5 6 7 8 9

 4) знаки операций:

 + - \* / = <> < > <= >= := @

 5) ограничители:

 . , ' ( ) [ ] (. .) { } (\* \*) .. : ;

 6) спецификаторы:

 ^ # $

 7) служебные (зарезервированные) слова:

ABSOLUTE EXPORTS LIBRARY SET

ASSEMBLER EXTERNAL MOD SHL

AND FAR NAME SHR

ARRAY FILE NIL STRING

ASM FOR NEAR THEN

ASSEMBLER FORWARD NOT TO

BEGIN FUNCTION OBJECT TYPE

CASE GOTO OF UNIT

CONST IF OR UNTIL

CONSTRUCTOR IMPLEMENTATION PACKED USES

DESTRUCTOR IN PRIVATE VAR

DIV INDEX PROCEDURE VIRTUAL

DO INHERITED PROGRAM WHILE

DOWNTO INLINE PUBLIC WITH

ELSE INTERFACE RECORD XOR

END INTERRUPT REPEAT

EXPORT LABEL RESIDENT

 Кроме перечисленных, в набор основных символов входит пробел. Про-

белы нельзя использовать внутри сдвоенных символов и зарезервирован-

ных слов.

3. Э Л Е М Е Н Т А Р Н Ы Е К О Н С Т Р У К Ц И И

 Элементарные конструкции языка ПАСКАЛЬ включают в себя имена, чис-

ла и строки.

 Имена (идентификаторы) называют элементы языка - константы, метки,

типы, переменные, процедуры, функции, модули, объекты. Имя - это пос-

ледовательность букв и цифр, начинающаяся с буквы. В именах может ис-

пользоваться символ \_ подчеркивание. Имя может содержать произвольное

количество символов, но значащими являются 63 символа.

 Не разрешается в языке ПАСКАЛЬ использовать в качестве имен слу-

жебные слова и стандартные имена, которыми названы стандартные конс-

танты, типы, процедуры, функции и файлы.

 Для улучшения наглядности программы в нее могут вставляться пробе-

лы. По крайней мере один пробел требуется вставить между двумя

последовательными именами, числами или служебными и стандартными име-

нами. Пробелы нельзя использовать внутри имен и чисел.

 Примеры имен языка ПАСКАЛЬ:

 A b12 r1m SIGMA gamma I80\_86

 Числа в языке ПАСКАЛЬ обычно записываются в десятичной системе

счисления. Они могут быть целыми и действительными. Положительный

знак числа может быть опущен. Целые числа записываются в форме без

десятичной точки, например:

 217 -45 8954 +483

 Действительные числа записываются в форме с десятичной точкой или

в форме с использованием десятичного порядка, который изображается

буквой Е:

 28.6 0.65 -0.018 4.0 5Е12 -1.72Е9 73.1Е-16

 ПАСКАЛЬ допускает запись целых чисел и фрагментов действительных

чисел в форме с порядком в шестнадцатиричной системе счисления:

 $7F $40 $ABC0

 Строки в языке ПАСКАЛЬ - это последовательность символов, записан-

ная между апострофами. Если в строке в качестве содержательного сим-

вола необходимо употребить сам апостроф, то следует записать два

апострофа. Примеры строк:

 'СТРОКА' 'STRING' 'ПРОГРАММА' 'АД''ЮТАНТ'

4. К О Н Ц Е П Ц И Я Т И П А

Д Л Я Д А Н Н Ы Х

 В математике принято классифицировать переменные в соответствии с

некоторыми важными характеристиками. Производится строгое разграниче-

ние между вещественными, комплексными и логическими переменными, меж-

ду переменными, представляющими отдельные значения и множество значе-

ний и так далее.

 При обработке данных на ЭВМ такая классификация еще более важна. В

любом алгоритмическом языке каждая константа, переменная, выражение

или функция бывают определенного типа.

 В языке ПАСКАЛЬ существует правило: тип явно задается в описании

переменной или функции, которое предшествует их использованию. Кон-

цепция типа языка ПАСКАЛЬ имеет следующие основные свойства:

 -любой тип данных определяет множество значений, к которому при-

надлежит константа, которые может принимать переменная или выражение,

или вырабатывать операция или функция;

 -тип значения, задаваемого константой, переменной или выражением,

можно определить по их виду или описанию;

 -каждая операция или функция требует аргументов фиксированного ти-

па и выдает результат фиксированного типа.

 Отсюда следует, что транслятор может использовать информацию о ти-

пах для проверки вычислимости и правильности различных конструкций.

 Тип определяет:

 -возможные значения переменных, констант, функций, выражений, при-

надлежащих к данному типу;

 -внутреннюю форму представления данных в ЭВМ;

 -операции и функции, которые могут выполняться над величинами,

принадлежащими к данному типу.

 Обязательное описание типа приводит к избыточности в тексте прог-

рамм, но такая избыточность является важным вспомогательным средс-

твом разработки программ и рассматривается как необходимое свойство

современных алгоритмических языков высокого уровня. В языке ПАСКАЛЬ

существуют скалярные и структурированные типы данных.

 К cкалярным типам относятся стандартные типы и типы, определяе-

мые пользователем.

 Стандартные типы включают целые, действительные, символьный,

логические и адресный типы. Типы, определяемые пользователем, - пере-

числяемый и интервальный.

 Структурированные типы имеют четыре разновидности: массивы, мно-

жества, записи и файлы.

 Кроме перечисленных, TURBO PASCAL включает еще два типа - проце-

дурный и объектный.

 Из группы скалярных типов можно выделить порядковые типы,

которые характеризуются следующими свойствами:

 -все возможные значения порядкового типа представляют собой

ограниченное упорядоченное множество;

 -к любому порядковому типу может быть применена стандартная

функция Ord, которая в качестве результата возвращает порядко-

вый номер конкретного значения в данном типе;

 -к любому порядковому типу могут быть применены стандартные

функции Pred и Succ, которые возвращают предыдущее и последую-

щее значения соответственно;

 -к любому порядковому типу могут быть применены стандартные функ-

ции Low и High, которые возвращают наименьшее и наибольшее значения

величин данного типа.

 В языке ПАСКАЛЬ введены понятия эквивалентности и совместимости типов.

 Два типа Т1 и Т2 являются эквивалентными (идентичными), если выполняется

одно из двух условий:

 -Т1 и Т2 представляют собой одно и то же имя типа;

 -тип Т2 описан с использованием типа Т1 с помощью равенства или

последовательности равенств. Например:

 type

 T1 = Integer;

 T2 = T1;

 T3 = T2;

 Менее строгие ограничения определены совместимостью типов. Напри-

мер, типы являются совместимыми, если:

 -они эквивалентны;

 -являются оба либо целыми, либо действительными;

 -один тип - интервальный, другой - его базовый;

 -оба интервальные с общим базовым;

 один тип - строковый, другой - символьный.

 В ТУРБО ПАСКАЛЬ ограничения на совместимость типов можно обойти с

помощью приведения типов. Приведение типов позволяет рассматривать

одну и ту же величину в памяти ЭВМ как принадлежащую разным типам.

Для этого используется конструкция

 Имя\_Типа(переменная или значение).

 Напрмер,

 Integer('Z')

представляет собой значение кода символа 'Z' в двухбайтном представ-

лении целого числа, а

 Byte(534)

даст значение 22, поскольку целое число 534 имеет тип Word и занима-

ет два байта, а тип Byte занимает один байт, и в процессе приведения

старший байт будет отброшен.

5. С Т А Н Д А Р Т Н Ы Е Т И П Ы

Д А Н Н Ы Х

 К стандартным относятся целые, действительные, логические,

символьный и адресный типы.

 ЦЕЛЫЕ типы определяют константы, переменные и функции, значения

которых реализуются множеством целых чисел, допустимых в данной ЭВМ.

 тип диапазон значений требуемая память

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Shortint -128 .. 127 1 байт

Integer -32768 .. 32767 2 байта

Longint -2147483648 .. 2147483647 4 байта

Byte 0 .. 255 1 байт

Word 0 .. 65535 2 байта

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 Над целыми операндами можно выполнять следующие арифметические

операции: сложение, вычитание, умножение, деление, получение остатка

от деления. Знаки этих операций:

 + - \* div mod

 Результат арифметической операции над целыми операндами есть вели-

чина целого типа. Результат выполнения операции деления целых величин

есть целая часть частного. Результат выполнения операции получения

остатка от деления - остаток от деления целых. Например:

 17 div 2 = 8, 3 div 5 = 0.

 17 mod 2 = 1, 3 mod 5 = 3.

 Операции отношения, примененные к целым операндам, дают результат

логического типа TRUE или FALSE ( истина или ложь ).

 В языке ПАСКАЛЬ имеются следующие операции отношения: равенство =,

неравенство <>, больше или равно >=, меньше или равно <=, больше >,

меньше < .

 К аргументам целого типа применимы следующие стандартные (встроен-

ные) функции, результат выполнения которых имеет целый тип:

 Abs(X), Sqr(X), Succ(X), Pred(X),

и которые определяют соответственно абсолютное значение Х, Х в квад-

рате, Х+1, Х-1.

 Следующая группа стандартных функций для аргумента целого типа да-

ет действительный результат:

 Sin(X), Cos(X), ArcTan(X), Ln(X), Exp(X), Sqrt(X).

 Эти функции вычисляют синус, косинус и арктангенс угла, заданного

в радианах, логарифм натуральный, экспоненту и корень квадратный со-

ответственно.

 Результат выполнения функции проверки целой величины на нечетность

Odd(X) имеет значение истина, если аргумент нечетный, и значение

ложь, если аргумент четный:

 X=5 Odd(X)=TRUE , X=4 Odd(X)=FALSE.

 Для быстрой работы с целыми числами определены процедуры:

 Inc(X) X:=X+1

 Inc(X,N) X:=X+N

 Dec(X) X:=X-1

 Dec(X,N) X:=X-N

 ДЕЙСТВИТЕЛЬНЫЕ типы определяет те данные, которые реализуются

подмножеством действительных чисел, допустимых в данной ЭВМ.

 Тип Диапазон Количество цифр Требуемая

 значений мантиссы память (байт)

---------------------------------------------------------------

Real 2.9e-39 .. 1.7e+38 11 6

Single 1.5e-45 .. 3.4e+38 7 4

Double 5.0e-324 .. 1.7e+308 15 8

Extended 3.4e-4932 .. 1.1e+4932 19 10

Comp -9.2e+18 .. 9.2e+18 19 8

---------------------------------------------------------------

 Тип Real определен в стандартном ПАСКАЛЕ и математическим сопро-

цессором не поддерживается.

 Остальные действительные типы определены стандартом IEEE 457 и ре-

ализованы на всех современных компьютерах.

 Для их использования при наличии сопроцессора или при работе на

ЭВМ типа 80486 необходимо компилировать программу с ключом {$ N+}, а

при отсутствии сопроцессора - с ключами {$N-,E+}.

 Тип Comp хотя и относится к действительным типам, хранит только

длинные целые значения.

 Над действительными операндами можно выполнять следующие арифмети-

ческие операции, дающие действительный результат:

 сложение + , вычитание - , умножение \* , деление / .

 К величинам действительного типа применимы все операции отношения,

дающие булевский результат.

 Один из операндов, участвующих в этих операциях, может быть целым.

 К действительным аргументам применимы функции, дающие действитель-

ный результат:

 Abs(X), Sqr(X), Sin(X), Cos(X), ArcTan(X), Ln(X), Exp(X),

Sqrt(X), Frac(X), Int(X), Pi.

 Функция Frac(X) возвращает дробную часть X, функция Int(X) - целую

часть X.

 Безаргументная функция Pi возвращает значение числа Пи действи-

тельного типа.

 К аргументам действительного типа применимы также функции

 Trunc(X) и Round(X),

дающие целый результат. Первая из них выделяет целую часть действи-

тельного аргумента путем отсечения дробной части, вторая округляет

аргумент до ближайшего целого.

 ЛОГИЧЕСКИЙ тип (Boolean) определяет те данные, которые могут при-

нимать логические значения TRUE и FALSE.

 К булевским операндам применимы следующие логические операции:

 not and or xor.

 Логический тип определен таким образом, что FALSE < TRUE. Это поз-

воляет применять к булевским операндам все операции отношения.

 В ТУРБО ПАСКАЛЬ введены еще разновидности логического типа:

ByteBool, WordBool и LongBool, которые занимают в памяти ЭВМ один, два

и четыре байта соответственно.

 СИМВОЛЬНЫЙ тип (Char) определяет упорядоченную совокупность симво-

лов, допустимых в данной ЭВМ. Значение символьной переменной или

константы - это один символ из допустимого набора.

 Символьная константа может записываться в тексте программы тремя

способами:

 -как один символ, заключенный в апострофы, например:

 'A' 'a' 'Ю' 'ю';

 -с помощью конструкции вида #K, где K - код соответствущего симво-

ла, при этом значение K должно находиться в пределах 0..255;

 -с помощью конструкции вида ^C, где C - код соответствущего управ-

ляющего символа, при этом значение C должно быть на 64 больше

кода управляющего символа.

 К величинам символьного типа применимы все операции отношения.

 Для величин символьного типа определены две функции преобразования

 Ord(C) Chr(K).

 Первая функция определяет порядковый номер символа С в наборе сим-

волов, вторая определяет по порядковому номеру К символ, стоящий на

К-ом месте в наборе символов. Порядковый номер имеет целый тип.

 К аргументам символьного типа применяются функции, которые опреде-

ляют предыдущий и последующий символы:

 Pred(C) Succ(C). Pred('F') = 'E' ; Succ('Y') = 'Z' .

 При отсутствии предыдущего или последующего символов значение со-

ответствующих функций не определено.

 Для литер из интервала 'a'..'z' применима функция UpCase(C), кото-

рая переводит эти литеры в верхний регистр 'A'..'Z'.

 АДРЕСНЫЙ тип (Pointer) определяет переменные, которые могут содер-

жать значения адресов данных или фрагментов программы. Для хранения

адреса требуются два слова (4 байта), одно из них определяет сегмент,

второе - смещение.

 Работа с адресными переменными (указателями) будет рассмотрена

позже, сейчас отметим, что для получения значения адреса какой-либо

переменной введена унарная операция @.

6. К О Н С Т А Н Т Ы

 Тип констант в языке ПАСКАЛЬ определяется по их виду: константы

целого типа - это целые числа, не содержащие десятичной точки, конс-

танты действительного типа - действительные числа, логические конс-

танты - логические значения TRUE и FALSE, символьные константы - либо

строки длиной в один символ, либо конструкции вида #K или ^K.

 Язык ПАСКАЛЬ допускает использовать синонимы для обозначения конс-

тант, в этом случае текст программы содержит раздел описания

констант, например: {}

7. П Е Р Е М Е Н Н Ы Е.

И Н И Ц И А Л И З А Ц И Я П Е Р Е М Е Н Н Ы Х

 Тип переменных определяется пользователем в разделе описания пере-

менных:{}

 В настоящее время в профессиональном программировании принято

записывать имена переменных с использованием так называемой венгерс-

кой нотации.

 Венгерская нотация - это соглашение о наименованиях переменных и

функций. Соглашение широко используется при программировании на язы-

ках PASCAL, C и в среде WINDOWS.

 Венгерская нотация основывается на следующих принципах:

 -имена переменных и функций должны содержать префикс, описывающий

их тип;

 -имена переменных и функций записываются полными словами или сло-

восочетаниями или их сокращениями, но так, чтобы по имени можно было

понять назначение переменной или действие, выполняемое функцией.

 Префиксы записываются малыми буквами, первая буква каждого слова -

заглавная, префиксы и слова записываются либо слитно, либо через сим-

вол \_ (подчеркивание).

 Для языка PASCAL могут быть рекомендованы следующие префиксы для

скалярных переменных и функций:

Префикс Тип

---------------------

by Byte

sh Shortint

i Integer

w Word

l Longint

r Real

si Single

d Double

e Extended

c Comp

ch Char

b Boolean

p Pointer

x,у координаты символа или точки на экране

 Для величин структурированного типа могут быть использованы следу-

ющие префиксы:

a Array

s String

sz Stringz

se Set

re Record

f File

t Text

 Например:

 rV, arVector[1..20], sName, iCount.

 В откомпилированной программе для всех переменных отведено место в

памяти, и всем переменным присвоены нулевые значения.

 Для задания начальных значений переменным (инициализации перемен-

ных) TURBO PASCAL позволяет присваивать начальные значения переменным

одновременно с их описанием. Для этого используется конструкция

 имя переменной: тип = значение;

которая должна быть размещена в разделе описания констант, например:

 const rWeight: Real = 0.4;

8. В Ы Р А Ж Е Н И Я

 Выражение состоит из констант, переменных, указателей функций,

знаков операций и скобок. Выражение задает правило вычисления некото-

рого значения. Порядок вычисления определяется старшинством (приори-

тетом) содержащихся в нем операций. В языке ПАСКАЛЬ принят следующий

приоритет операций:

 1. унарная операция not, унарный минус -, взятие адреса @

 2. операции типа умножения \* / div mod and shl shr

 3. операции типа сложения + - or xor

 4. операции отношения = <> < > <= >= in

 {}

 Выражения входят в состав многих операторов языка ПАСКАЛЬ, а также

могут быть аргументами встроенных функций.

9. О П Е Р А Т О Р П Р И С В А И В А Н И Я

 Тип переменной и тип выражения должны совпадать кроме случая, ког-

да выражение относится к целому типу, а переменная - к действительно-

му. При этом происходит преобразование значения выражения к

действительному типу.{}

10. О П Е Р А Т О Р Ы В В О Д А

И В Ы В О Д А

 Рассмотрим организацию ввода и вывода данных с терминального уст-

ройства. Терминальное устройство - это устройство, с которым работает

пользователь, обычно это экран (дисплей) и клавиатура.

 Для ввода и вывода данных используются стандартные процедуры ввода

и вывода Read и Write, оперирующие стандартными последовательными файлами

INPUT и OUTPUT.

 Эти файлы разбиваются на строки переменной длины, отделяемые друг

от друга признаком конца строки. Конец строки задается нажатием кла-

виши ENTER.

 Для ввода исходных данных используются операторы процедур ввода:

 Read(A1,A2,...AK);

 ReadLn(A1,A2,...AK);

 ReadLn;

 Первый из них реализует чтение К значений исходных данных и прис-

ваивание этих значений переменным А1, А2, ..., АК. Второй оператор

реализует чтение К значений исходных данных, пропуск остальных значе-

ний до начала следующей строки, присваивание считанных значений пере-

менным А1, А2, ..., АК. Третий оператор реализует пропуск строки ис-

ходных данных.

 При вводе исходных данных происходит преобразование из внешней

формы представления во внутреннюю, определяемую типом переменных. Пе-

ременные, образующие список ввода, могут принадлежать либо к целому,

либо к действительному, либо к символьному типам. Чтение исходных

данных логического типа в языке ПАСКАЛЬ недопустимо.

 Операторы ввода при чтении значений переменных целого и

действительного типа пропускает пробелы, предшествующие числу. В то

же время эти операторы не пропускают пробелов, предшествующих значе-

ниям символьных переменных, так как пробелы являются равноправными

символами строк. Пример записи операторов ввода:

 var rV, rS: Real;

 iW, iJ: Integer;

 chC, chD: Char;

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

 Read(rV, rS, iW, iJ);

 Read(chC, chD);

 Значения исходных данных могут отделяться друг от друга пробелами

и нажатием клавиш табуляции и Enter.

 Для вывода результатов работы программы на экран используются

операторы:

 Write(A1,A2,...AK);

 WriteLn(A1,A2,...AK);

 WriteLn;

 Первый из этих операторов реализует вывод значений переменных А1,

А2,...,АК в строку экрана. Второй оператор реализует вывод

значений переменных А1, А2, ..., АК и переход к началу следующей

строки. Третий оператор реализует пропуск строки и переход к началу

следующей строки.

 Переменные, составляющие список вывода, могут относиться к целому,

действительному, символьному или булевскому типам. В качестве элемен-

та списка вывода кроме имен переменных могут использоваться выражения

и строки.

 Вывод каждого значения в строку экрана происходит в соот-

ветствии с шириной поля вывода, определяемой конкретной реализацией

языка.

 Форма представления значений в поле вывода соответствует типу пе-

ременных и выражений: величины целого типа выводятся как целые деся-

тичные числа, действительного типа - как действительные десятичные

числа с десятичным порядком, символьного типа и строки - в виде сим-

волов, логического типа - в виде логических констант TRUE и FALSE.

 Оператор вывода позволяет задать ширину поля вывода для каждого

элемента списка вывода. В этом случае

элемент списка вывода имеет вид А:К, где

А - выражение или строка, К - выражение либо константа целого типа.

Если выводимое значение занимает в поле вывода меньше позиций, чем К,

то перед этим значением располагаются пробелы. Если выводимое значе-

ние не помещается в ширину поля К, то для этого значения будет отве-

дено необходимое количество позиций. Для величин действительного типа

элемент списка вывода может иметь вид А:К:М, где А - переменная или выраже-

ние действительного типа, К - ширина поля вывода, М - число цифр

дробной части выводимого значения. К и М - выражения или константы

целого типа. В этом случае действительные значения выводятся в форме

десятичного числа с фиксированной точкой.

 Пример записи операторов вывода:

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

 var rA, rB: Real; iP,iQ:Integer;

 bR, bS: Boolean; chT, chV, chU, chW: Char;

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

 WriteLn(rA, rB:10:2);

 WriteLn(iP, iQ:8);

 WriteLn(bR, bS:8);

 WriteLn(chT, chV, chU, chW);

11. С Т Р У К Т У Р А П Р О Г Р А М М Ы

 Программа на языке ПАСКАЛЬ состоит из заголовка, разделов описаний

и раздела операторов.

 Заголовок программы содержит имя программы, например:

 Program PRIM;

 Описания могут включать в себя раздел подключаемых библиотек (мо-

дулей), раздел описания меток, раздел описания констант, раздел опи-

сания типов, раздел описания переменных, раздел описания процедур и

функций.

 Раздел описания модулей определяется служебным словом USES и со-

держит имена подключаемых модулей (библиотек) как входящих в состав

системы TURBO PASCAL, так и написанных пользователем. Раздел описания

модулей должен быть первым среди разделов описаний. Имена модулей от-

деляются друг от друга запятыми:

 uses CRT, Graph;

 Любой оператор в программе может быть помечен меткой. В качестве

метки используются произвольные целые без знака, содержащие не более

четырех цифр, либо имена. Метка ставится перед оператором и отделяет-

ся от него двоеточием. Все метки, используемые в программе, должны

быть перечислены в разделе описания меток, например:

 label 3, 471, 29, Quit;

 Описание констант позволяет использовать имена как синонимы конс-

тант, их необходимо определить в разделе описаний констант:

 const K= 1024; MAX= 16384;

 В разделе описания переменных необходимо определить тип всех пере-

менных, используемых в программе:

 var P,Q,R: Integer;

 A,B: Char;

 F1,F2: Boolean;

 Описание типов, прцедур и функций будет рассмотрено ниже. Отдель-

ные разделы описаний могут отсутствовать, но следует помнить, что в

ПАСКАЛЬ - программе должны быть обязательно описаны все компоненты

программы.

 Раздел операторов представляет собой составной оператор, который

содержит между служебными словами

 begin.......end

последовательность операторов. Операторы отделяются друг от друга

символом ;.

 Текст программы заканчивается символом точка.

 Кроме описаний и операторов ПАСКАЛЬ - программа может содержать

комментарии, которые представляют собой произвольную последователь-

ность символов, расположенную между открывающей скобкой комментариев

{ и закрывающей скобкой комментариев }.

 Текст ПАСКАЛЬ - программы может содержать ключи компиляции, кото-

рые позволяют управлять режимом компиляции. Синтаксически ключи ком-

пиляции записываются как комментарии. Ключ компиляции содержит символ

$ и букву-ключ с последующим знаком + (включить режим) или - (выклю-

чить режим). Например:

 {$E+} - эмулировать математический сопроцессор;

 {$F+} - формировать дальний тип вызова процедур и функций;

 {$N+} - использовать математический сопроцессор;

 {$R+} - проверять выход за границы диапазонов.

 Некоторые ключи компиляции могут содержать параметр, например:

 {$I имя файла} - включить в текст компилируемой программы назван-

ный файл.

 Пример записи простой программы:

 Program TRIANG;

 var A, B, C, S, P: Real;

 begin

 Read(A,B,C);

 WriteLn(A,B,C);

 P:=(A+B+C)/2;

 S:=Sqrt(P\*(P-A)\*(P-B)\*(P-C));

 WriteLn('S=',S:8:3)

 end.

12. Б И Т О В А Я А Р И Ф М Е Т И К А

 Битовая или поразрядная арифметика введена в TURBO PASCAL для

обеспечения возможности работы с двоичными разрядами (битами). Опера-

ции битовой арифметики применимы только к целым типам.

 Первая группа операций - логические операции not, and, or и xor.

 Операция not является одноместной, она изменяет каждый бит целого

числа на обратный.

 Операции and, or и xor - двуместные, операнды этих операций - целые

величины одинаковой длины. Операции выполняются попарно над всеми

двоичными разрядами операндов.

 Вторая группа операций - это операции сдвига влево shl и сдвига

вправо shr:

 I shl N

 I shr N.

 Эти операции сдвигают двоичную последовательность значения I влево

или вправо на N двоичных разрядов. При этом биты, уходящие за пределы

разрядной сетки, теряются, а освободившиеся двоичные разряды заполня-

ются нулями. При сдвиге вправо отрицательных значений освободившиеся

разряды заполняются единицами.

13. О П Е Р А Т О Р П Е Р Е Х О Д А

 Обычно операторы в программе выполняются в том порядке, в каком

они записаны. Оператор перехода прерывает естественный порядок выпол-

нения программы и указывает, что дальнйшее выполнение должно продол-

жаться, начиная с оператора, помеченного меткой, указанной в операто-

ре перехода. Пример записи оператора перехода:

 goto 218;

14. Э Л Е М Е Н Т Ы С Т Р У К Т У Р Н О Г О

П Р О Г Р А М М И Р О В А Н И Я

 Структуризованная программа (или подпрограмма) - это программа,

составленная из фиксированного множества базовых конструкций. Расс-

мотрим основные определения и способы образования этих конструкций в

схемах алгоритмов.

{}

 Из операций, развилок и слияний строятся базовые конструкции: сле-

дование, ветвление, цикл. Применяя только эти три конструкции, можно

реализовать алгоритм решения любой задачи.

 Конструкция, представляющая собой последовательное выполнение двух

или более операций, называется следованием.

 Конструкция, состоящая из развилки, двух операций и слияния, назы-

вается ветвлением. Одна из операций может отсутствовать.

 Конструкция, имеющая линии управления, ведущие к предидущим опера-

циям или развилкам, называется циклом.

 Конструкции следование, ветвление и цикл можно представить как

операции, так как они имеют единственный вход и единственный выход.

Произвольную последовательность операций можно представить как одну

операцию.

 Операция может быть реализована любым оператором языка ПАСКАЛЬ

(простым или составным), либо группой операторов, за исключением опе-

ратора перехода GOTO.

 В языке ПАСКАЛЬ количество базовых конструкций увеличено до шести,

это:

 -следование;

 -ветвление;

 -цикл с предусловием;

 -цикл с постусловием;

 -цикл с параметром;

 -вариант.

 Далее рассмотрим, как эти базовые конструкции реализуются в языке

ПАСКАЛЬ.

15. У С Л О В Н Ы Й О П Е Р А Т О Р

 {}

 Условный оператор в короткой форме работает по правилу: если бу-

левское выражение B истинно, то выполняется оператор ОР1, далее вы-

полняется оператор, следующий за условным. Если булевское выражение B

ложно, то будет выполняться оператор, следующий за этим условным опе-

ратором.

16. Ц И К Л С П Р Е Д У С Л О В И Е М{}

17. Ц И К Л С П О С Т У С Л О В И Е М{}

18. Ц И К Л С П А Р А М Е Т Р О М{}

19. О П Е Р А Т О Р Ы З А В Е Р Ш Е Н И Я Ц И К Л А

 Для всех операторов цикла выход из цикла осуществляется как

вследствие естественного окончания оператора цикла, так и с

помощью операторов перехода и выхода.

 В версии ТУРБО ПАСКАЛЬ 7.0 определены стандартные процедуры Break и

Continue. Процедура Break выполняет безусловный выход из цикла. Проце-

дура Continue обеспечивает переход к началу новой итерации цикла.

20. О П Е Р А Т О Р В А Р И А Н Т А{}

21. П Е Р Е Ч И С Л Я Е М Ы Й Т И П Д А Н Н Ы Х

 Перечисляемый тип представляет собой ограниченную упорядоченную

последовательность скалярных констант, составляющих данный тип. Зна-

чение каждой константы задается ее именем. Имена отдельных констант

отделяются друг от друга запятыми, а вся совокупность констант, сос-

тавляющих данный перечисляемый тип, заключается в круглые скобки.

 Программист объединяет в одну группу в соответствии с каким - либо

признаком всю совокупность значений, составляющих перечисляемый тип.

Например, перечисляемый тип Rainbow(РАДУГА) объединяет скалярные значения

RED, ORANGE, YELLOW, GREEN, LIGHT\_BLUE, BLUE, VIOLET (КРАСНЫЙ,

ОРАНЖЕВЫЙ, ЖЕЛТЫЙ, ЗЕЛЕНЫЙ, ГОЛУБОЙ, СИНИЙ, ФИОЛЕТОВЫЙ). Пе-

речисляемый тип Traffic\_Light (СВЕТОФОР) объединяет скалярные

значения RED, YELLOW, GREEN (КРАСНЫЙ, ЖЕЛТЫЙ, ЗЕЛЕНЫЙ).

 Перечисляемый тип описывается в разделе описания типов, который

начинается со служебного слова type, например:

type

 Rainbow = (RED, ORANGE, YELLOW, GREEN, LIGHT\_BLUE, BLUE, VIOLET);

 Каждое значение является константой своего типа и может принадле-

жать только одному из перечисляемых типов, заданных в программе. Нап-

ример, перечисляемый тип Traffic\_Light не может быть определен в одной

программе с типом Rainbow, так как оба типа содержат одинаковые конс-

танты.

 Описание переменных, принадлежащих к скалярным типам, которые объ-

явлены в разделе описания типов, производится с помощью имен типов.

Например:

 type Traffic\_Light= (RED, YELLOW, GREEN);

 var Section: Traffic\_Light;

 Это означает, что переменная Section может принимать значения RED,

YELLOW или GREEN.

 Переменные перечисляемого типа могут быть описаны в разделе описа-

ния переменных, например:

 var Section: (RED, YELLOW, GREEN);

 При этом имена типов отсутствуют, а переменные определяются сово-

купностью значений, составляющих данный перечисляемый тип.

 К переменным перечисляемого типа может быть применим оператор

присваивания:

 Section:= YELLOW;

 Упорядоченная последовательность значений, составляющих перечисля-

емый тип, автоматически нумеруется, начиная с нуля и далее через еди-

ницу. Отсюда следует, что к перечисляемым переменным и константам мо-

гут быть применены операции отношения и стандартные функции Pred,

Succ, Ord.

 Переменные и константы перечисляемого типа не могут быть элемента-

ми списка ввода или вывода.

22. И Н Т Е Р В А Л Ь Н Ы Й Т И П Д А Н Н Ы Х

 Отрезок любого порядкового типа

может быть определен как интервальный или ограниченный

тип. Отрезок задается диапазоном от минимального до максимального

значения констант, разделенных двумя точками. В качестве констант мо-

гут быть использованы константы, принадлежащие к целому, символьному,

логическому или перечисляемому типам. Скалярный тип, на котором стро-

ится отрезок, называется базовым типом.

 Минимальное и максимальное значения констант называются нижней и

верхней границами отрезка, определяющего интервальный тип. Нижняя

граница должна быть меньше верхней.

 {}

 Над переменными, относящимися к интервальному типу, могут выпол-

няться все операции и применяться все стандартные функции, которые

допустимы для соответствующего базового типа.

 При использовании в программах интервальных типов данных может осущест-

вляться контроль за тем, чтобы значения переменных не выходили за

границы, введенные для этих переменных в описании интервального типа.

23. М А С С И В Ы

 Массивы представляют собой ограниченную упорядоченную совокупность

однотипных величин. Каждая отдельная величина называется компонентой

массива. Тип компонент может быть любым, принятым в языке ПАСКАЛЬ,

кроме файлового типа. Тип компонент называется базовым типом.

 Вся совокупность компонент определяется одним именем. Для обозна-

чения отдельных компонент используется конструкция, называемая пере-

менной с индексом или с индексами:

 A[5] S[k+1] B[3,5].

 В качестве индекса может быть использовано выражение. Тип индексов

может быть только интервальным или перечисляемым. Действительный и

целый типы недопустимы. Индексы интервального типа, для которого ба-

зовым является целый тип, могут принимать отрицательные, нулевое и

положительные значения.{}

 В операторной части

программы один массив может быть присвоен другому, если их типы иден-

тичны, например:

 R1:=Z.

 Для ввода или вывода массива в список ввода или вывода помещается

переменная с индексом, а операторы ввода или вывода выполняются в

цикле.

{}

 Первый индекс определяет номер строки, второй - номер столбца.

Двумерные массивы хранятся в памяти ЭВМ по строкам.

 Инициализация массивов (присвоение начальных значений всем компо-

нентам массивов) осуществляется двумя способами.

 Первый способ - с использованием типизированных констант, напри-

мер:

 type Dim10= Array[1..10] of Real;

 const

 raM10: Dim10 = ( 0, 2.1, 4, 5.65, 6.1, 6.7, 7.2, 8, 8.7, 9.3 );

 При инициализации двумерных массивов значения компонент каждого из

входящих в него одномерных массивов записывается в скобках:

 type Dim3x2= Array[1..3,1..2] of Integer;

 const

 iaM3x2: Dim3x2= ( (1, 2)

 (3, 4)

 (5, 6) );

 Второй способ инициализации - использование разновидности процеду-

ры FillChar:

 FillChar( var V; NBytes: Word; B: Byte );

Эта процедура заполняет участок памяти однобайтовым значением. Напри-

мер, для обнуления массива A[1..10] of Real можно записать:

 FillChar(A, 40, 0);

или

 FillChar(A, SizeOf(A), 0);

{}

24. С Т Р О К И

 Особое место в языке ПАСКАЛЬ занимают массивы символов. Стандарт-

ный ПАСКАЛЬ допускает два способа хранения символьных массивов в па-

мяти ЭВМ: распакованный и упакованный. Распакованные массивы символов

хранятся в памяти ЭВМ по одному символу в машинном слове, упакованные

- по одному символу в байте. При описании упакованного массива симво-

лов используют служебное слово PACKED, например:

 var MAS: Packed Array[1..20] of Char;

Описание распакованного массива символов имеет вид:

 var M: Array[1..20] of char;

 Для преобразования символьного массива из распакованной формы в

упакованную и наоборот, из упакованной в распакованную, в язык ПАС-

КАЛЬ введены две стандартные функции Pack, UnPack.

 Упакованный массив символов образует символьную строку. Символьная

строка может быть либо строковой константой, либо строковой перемен-

ной. Строковая константа, или строка, представляет собой совокупность

символов, заключенную в апострофы. Строка - это элементарная конс-

трукция языка ПАСКАЛЬ. Строковые константы могут входить в состав вы-

ражений. Как и числовые константы, они могут быть описаны в разделе

описания констант.

 Строковые переменные - это одномерные упакованные массивы симво-

лов, для описания которых в TURBO PASCAL введен тип String.

 Например, если строка содержит до 30 символов, ее тип будет опре-

делен как

 type s= String[30];

 Длина строки не может содержать более, чем 255 символов.

 В TURBO PASCAL определено понятие строки переменной длины, в этом

случае ее описание задается как

 type s= String;

 Тип String без указания длины совместим со всеми типами строк.

 Особенностью строковых переменных является то, что к ним можно об-

ращаться как к скалярным переменным, так и к массивам. Во втором слу-

чае применяется конструкция "переменная с индексом", что обеспечивает

доступ к отдельным символам строки. При этом нижняя граница идекса

равна 1. Отдельный символ строки совместим с типом Char.

 В памяти ЭВМ строка занимает количество байтов, на единицу большее

ее длины. Нулевой байт строки содержит ее длину.

 Для строк определены операции присваивания, слияния (конкатенации)

и сравнения.

 Для сравнения строк применяются все операции отношения. Сравнение

строк происходит посимвольно, начиная с первого символа. Строки рав-

ны, если имеют одинаковую длину и посимвольно эквивалентны.

 Строки могут быть элементами списка ввода - вывода, при этом запи-

сывается имя строки без индекса.

 При вводе строковых переменных количество вводимых символов может

быть меньше, чем длина строки. В этом случае вводимые символы разме-

щаются с начала строки, а оставшиеся байты заполняются пробелами. Ес-

ли количество вводимых символов превышает длину строки, лишние

символы отбрасываются.

 Инициализация строк может производиться как с помощью типизирован-

ных констант:

 const sName: String[9]= 'IBM PC/AT';

так и с использованием второй разновидности функции FillChar:

 FillChar( var V; NBytes: Word; C: Char );

например:

 FillChar(A, SizeOf(A), '0');

 Для работы со строками в TURBO PASCAL включены процедуры и функ-

ции, которые обеспечивают редактирование и преобразование строк.

{}

25. П Р О Ц Е Д У Р Ы И Ф У Н К Ц И И

 Алгоритм решения задачи проектируется путем декомпозиции всей за-

дачи в отдельные подзадачи. Обычно подзадачи реализуются в виде подп-

рограмм.

 Подпрограмма - это последовательность операторов, которые опреде-

лены и записаны только в одном месте программы, однако их можно

вызвать для выполнения из одной или нескольких точек программы. Каж-

дая подпрограмма определяется уникальным именем. В языке ПАСКАЛЬ су-

ществуют два типа подпрограмм - процедуры и функции.

 Процедура и функция - это именованная последовательность описаний

и операторов. При использовании процедур или функций ПАСКАЛЬ - прог-

рамма должна содержать текст процедуры или функции и обращение к про-

цедуре или функции. Тексты процедур и функций помещаются в раздел

описаний процедур и функций.

{}

 Процедура может содержать такие - же разделы описаний, что и ПАС-

КАЛЬ - программа, а именно: разделы описания модулей, меток, конс-

тант, типов, переменных, процедур и функций.

{}

 ПЕРЕДАЧА ИМЕН ПРОЦЕДУР И ФУНКЦИЙ В КАЧЕСТВЕ ПАРАМЕТРОВ. Во многих

задачах, особенно в задачах вычислительной математики, необходимо пе-

редавать имена процедур и функций в качестве параметров. Для этого в

TURBO PASCAL введен новый тип данных - процедурный или функциональ-

ный, в зависимости от того, что описывается.

 Описание процедурных и функциональных типов производится в разделе

описания типов:

 type

 FuncType = Function(z: Real): Real;

 ProcType = Procedure (a,b: Real; var x,y: Real);

 Функциональный и процедурный тип определяется как заголовок проце-

дуры и функции со списком формальных параметров, но без имени. Можно

определить функциональный или процедурный тип без параметров, напри-

мер:

 type

 Proc = Procedure;

 После объявления процедурного или функционального типа его можно

использовать для описания формальных параметров - имен процедур и

функций.

 Кроме того, необходимо написать те реальные процедуры или функции,

имена которых будут передаваться как фактические параметры. Эти про-

цедуры и функции должны компилироваться в режиме дальней адресации с

ключом {$F+}.

 Пример. Составить программу для вычисления определенного интеграла

 tk

 2t

 I= S--------------- dt

 sqrt(1-sin2t)

 tn

по методу Симпсона. Вычисление подинтегральной функции реализовать с

помощью функции, имя которой передается как параметр. Значение опре-

деленного интеграла по формуле Симпсона вычисляется по формуле:

 ISimps=2\*h/3\*(0.5\*F(A)+2\*F(A+h)+F(A+2\*h)+2\*F(A+3\*h)+...

 +2\*F(B-h)+0.5\*F(B))

где A и B - нижняя и верхняя границы интервала интегрирования,

 N - число разбиений интервала интегрирования,

 h=(B-A)/N, причем N должно быть четным.

Program INTEGRAL;

 type

 Func= function(x: Real): Real;

 var

 I,TN,TK:Real;

 N:Integer;

{$F+}

 Function Q(t: Real): Real;

 begin

 Q:=2\*t/Sqrt(1-Sin(2\*t));

 end;

{$F-}

 Procedure Simps(F:Func; a,b:Real; N:Integer; var INT:Real);

 var

 sum, h: Real;

 j:Integer;

 begin

 if Odd(N) then N:=N+1;

 h:=(b-a)/N;

 sum:=0.5\*(F(a)+F(b));

 for j:=1 to N-1 do

 sum:=sum+(j mod 2+1)\*F(a+j\*h);

 INT:=2\*h\*sum/3

 end;

 begin

 WriteLn(' ВВЕДИ TN,TK,N');

 Read(TN,TK,N);

 Simps(Q,TN,TK,N,I);

 WriteLn('I=',I:8:3)

 end.

{}

26. О П Е Р А Т О Р Ы В Ы Х О Д А

 Для завершения работы программ, процедур и функций без предвари-

тельного перехода по меткам к закрывающему end в TURBO PASCAL введены

процедуры Exit и Halt.

 Вызов Exit завершает работу своего программного блока и передает

управление вызывающей программе. Если Exit выполняется в подпрограм-

ме, то выполнение этой подпрограммы прекратится, и далее будет выпол-

няться следующий за вызовом этой подпрограммы оператор. Если Exit вы-

полняется в основной программе, выход из нее будет эквивалентен ее

нормальному завершению.

 Вызов процедуры Halt, где бы она не находилась, завершает работу

программы и передает управление операционной системе.

 Процедура Halt имеет структуру Halt(n), где n - код возврата, ко-

торый может быть проанализирован операционной системой с помощью ко-

манды IF ERRORLEVEL. Значение n=0 соответствует нормальному заверше-

нию работы программы. Вызов процедуры Halt без параметра эквивалентен

вызову Halt(0).

27. М О Д У Л И

 Модуль (UNIT) в TURBO PASCAL - это особым образом оформленная биб-

лиотека подпрограмм. Модуль в отличие от программы не может быть за-

пущен на выполнение самостоятельно, он может только участвовать в

построении программ и других модулей.

 Модули позволяют создавать личные библиотеки процедур и функций и

строить программы практически любого размера.

 Модуль в TURBO PASCAL представляет собой отдельно хранимую и неза-

висимо компилируемую программную единицу.

 В общем случае модуль - это совокупность программных ресурсов,

предназначенных для использования другими программами. Под программ-

ными ресурсами понимаются любые элементы языка TURBO PASCAL: констан-

ты, типы, переменные, подпрограммы. Модуль сам по себе не является

выполняемой программой, его элементы используются другими программны-

ми единицами.

 Все программные элементы модуля можно разбить на две части:

 - программные элементы, предназначенные для использования другими

программами или модулями, такие элементы называют видимыми вне моду-

ля;

 - программные элементы, необходимые только для работы самого моду-

ля, их называют невидимыми или скрытыми.

 В соответствии с этим модуль, кроме заголовка, содержит две основ-

ные части, называемые интерфейсом и реализацией.

 В общем случае модуль имеет следующую структуру:

 unit <имя модуля>; {заголовок модуля}

 interface

 { описание видимых программных элементов модуля }

 { описание скрытых программных элементов модуля }

 begin

 { операторы инициализации элементов модуля }

 end.

 В частном случае модуль может не содержать части реализации и час-

ти инициализации, тогда структура модуля будет такой:

 unit <имя модуля>; {заголовок модуля}

 interface

 { описание видимых программных элементов модуля }

 implementation

 end.

 Использование в модулях процедур и функций имеет свои особенности.

Заголовок подпрограммы содержит все сведения, необходимые для ее вы-

зова: имя, перечень и тип параметров, тип результата для функций, эта

информация должна быть доступна для других программ и модулей. С дру-

гой стороны, текст подпрограммы, реализующий ее алгоритм, другими

программами и модулями не может быть использован. Поэтому заголовок

процедур и функций помещают в интерфейсную часть модуля, а текст - в

часть реализации.

 Интерфейсная часть модуля содержит только видимые (доступные для

других программ и модулей) заголовки процедур и функций (без служеб-

ного слова forward). Полный текст процедуры или функции помещают в

часть реализации, причем заголовок может не содержать список формаль-

ных параметров.

 Исходный текст модуля должен быть откомпилирован с помощью дирек-

тивы Make подменю Compile и записан на диск. Результатом компиляции

модуля является файл с расширением .TPU (Turbo Pascal Unit). Основное

имя модуля берется из заголовка модуля.

 Для подключения модуля к программе необходимо указать его имя в

разделе описания модулей, например:

 uses CRT, Graph;

 В том случае, если имена переменных в интерфейсной части модуля и

в программе, использующей этот модуль, совпадают, обращение будет

происходить к переменной, описанной в программе. Для обращения к пе-

ременной, описанной в модуле, необходимо применить составное имя,

состоящее из имени модуля и имени переменной, разделенных точкой.

 Например, пусть имеется модуль, в котором описана переменная К:

 unit M;

 interface

 var K: Integer;

 implementation

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

 end.

 Пусть программа, использующая этот модуль, также содержит перемен-

ную К:

 Program P;

 uses M;

 var K: Char;

 begin

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

 end.

 Для того, чтобы в программе P иметь доступ к переменной K из моду-

ля M, необходимо задать составное имя M.K.

 Использование составных имен применяется не только к именам пере-

менных, а ко всем именам, описанным в интерфейсной части модуля.

 Рекурсивное использование модулей запрещено.

 Если в модуле имеется раздел инициализации, то операторы из этого

раздела будут выполнены перед началом выполнения программы, в которой

используется этот модуль.

28. М Н О Ж Е С Т В А

 Понятие множества в языке ПАСКАЛЬ основывается на математическом

представлении о множествах: это ограниченная совокупность различных

элементов. Для построения конкретного множественного типа использует-

ся перечисляемый или интервальный тип данных. Тип элементов, состав-

ляющих множество, называется базовым типом.

 Множественный тип описывается с помощью служебных слов Set of,

например:

 type M= Set of B;

Здесь М - множественный тип, В - базовый тип.

 Пример описания переменной множественного типа:

 type

 M= Set of 'A'..'D';

 var

 MS: M;

 Принадлежность переменных к множественному типу может быть опреде-

лена прямо в разделе описания переменных:

 var

 C: Set of 0..7;

 Константы множественного типа записываются в виде заключенной в

квадратные скобки последовательности элементов или интервалов базово-

го типа, разделенных запятыми, например:

 ['A', 'C'] [0, 2, 7] [3, 7, 11..14].

Константа вида

 [ ]

означает пустое подмножество.

 Множество включает в себя набор элементов базового типа, все подм-

ножества данного множества, а также пустое подмножество. Если базовый

тип, на котором строится множество, имеет К элементов, то число подм-

ножеств, входящих в это множество, равно 2 в степени К. Пусть имеется

переменная Р интервального типа:

 var P: 1..3;

 Эта переменная может принимать три различных значения - либо 1,

либо 2, либо 3. Переменная Т множественного типа

 var T: Set of 1..3;

может принимать восемь различных значений:

 [ ] [1,2]

 [1] [1,3]

 [2] [2,3]

 [3] [1,2,3]

 Порядок перечисления элементов базового типа в константах безраз-

личен.

 Значение переменной множественного типа может быть задано конс-

трукцией вида [T], где T - переменная базового типа.

 К переменным и константам множественного типа применимы операции

присваивания(:=), объединения(+), пересечения(\*) и вычитания(-):

 ['A','B'] + ['A','D'] даст ['A','B','D']

 ['A'] \* ['A','B','C'] даст ['A']

 ['A','B','C'] - ['A','B'] даст ['C'].

 Результат выполнения этих операций есть величина множественного

типа.

 К множественным величинам применимы операции: тождественность (=),

нетождественность (<>), содержится в (<=), содержит (>=). Результат

выполнения этих операций имеет логический тип, например:

 ['A','B'] = ['A','C'] даст FALSE

 ['A','B'] <> ['A','C'] даст TRUE

 ['B'] <= ['B','C'] даст TRUE

 ['C','D'] >= ['A'] даст FALSE.

 Кроме этих операций для работы с величинами множественного типа в

языке ПАСКАЛЬ используется операция

 in

проверяющая принадлежность элемента базового типа, стоящего слева

от знака операции, множеству, стоящему справа от знака операции. Ре-

зультат выполнения этой операции - булевский. Операция проверки при-

надлежности элемента множеству часто используется вместо операций от-

ношения, например:

 A in ['A', 'B'] даст TRUE,

 2 in [1, 3, 6] даст FALSE.

 При использовании в программах данных множественного типа

выполнение операций происходит над битовыми строками данных. Каждому

значению множественного типа в памяти ЭВМ соответствует один двоичный

разряд. Например, множество

 ['A','B','C','D']

представлено в памяти ЭВМ битовой строкой

 1 1 1 1.

Подмножества этого множества представлены строками:

 ['A','B','D'] 1 1 0 1

 ['B','C'] 0 1 1 0

 ['D'] 0 0 0 1

 Величины множественного типа не могут быть элементами списка вво-

да - вывода.

 В каждой конкретной реализации транслятора с языка ПАСКАЛЬ коли-

чество элементов базового типа, на котором строится множество, огра-

ничено. В TURBO PASCAL количество базовых элементов не должно превы-

шать 256.

 Инициализация величин множественного типа производится с помощью

типизированных констант:

 const seLit: Set of 'A'..'D'= [];

 Проиллюстрируем применение данных множественного типа на примере.

 Пример. Составить программу, которая вырабатывает и выводит на эк-

ран дисплея наборы случайных чисел для игры в "Спортлото 5 из 36".

 Для заполнения каждой карточки спортлото необходимо получить набор

из пяти псевдослучайных чисел. К этим числам предъявляются два требо-

вания:

 -числа должны находиться в диапазоне 1..36;

 -числа не должны повторяться.

 Program Lotto;

 var

 nb, k: Set of 1..36;

 kol, l, i, n: Integer;

 begin

 Randomize;

 WriteLn('ВВЕДИ kol');

 ReadLn(kol);

 nb:=[1..36];

 for i:=1 to kol do

 begin

 k:=[];

 for l:=1 to 5 do

 begin

 repeat

 n:=Random(36)

 until (n in nb) and not (n in k);

 k:=k+[n];

 Write(n:4)

 end;

 WriteLn

 end

 end.

29. З А П И С И

 Запись представляет собой совокупность ограниченного числа логи-

чески связанных компонент, принадлежащих к разным типам. Компоненты

записи называются полями, каждое из которых определяется именем. Поле

записи содержит имя поля, вслед за которым через двоеточие указывает-

ся тип этого поля. Поля записи могут относиться к любому типу, допус-

тимому в языке Паскаль, за исключением файлового типа.

 Описание записи в языке ПАСКАЛЬ осуществляется с помощью

служебного слова RECORD, вслед за которым описываются компоненты за-

писи. Завершается описание записи служебным словом END.

 Например, записная книжка содержит фамилии, инициалы и номера те-

лефона, поэтому отдельную строку в записной книжке удобно представить

в виде следующей записи:

 type Row=Record

 FIO: String[20];

 TEL: String[7]

 end;

 var str: Row;

 Описание записей возможно и без использования имени типа, напри-

мер:

 var str: Record

 FIO: String[20];

 TEL: String[7]

 end;

 Обращение к записи в целом допускается только в операторах присва-

ивания, где слева и справа от знака присваивания используются имена

записей одинакового типа. Во всех остальных случаях оперируют отдель-

ными полями записей. Чтобы обратиться к отдельной компоненте записи,

необходимо задать имя записи и через точку указать имя нужного поля,

например:

 str.FIO, str.TEL

 Такое имя называется составным. Компонентой записи может быть так-

же запись, в таком случае составное имя будет содержать не два, а

большее количество имен.

 Обращение к компонентам записей можно упростить, если воспользо-

ваться оператором присоединения with.

 Он позволяет заменить составные имена, характеризующие каждое по-

ле, просто на имена полей, а имя записи определить в операторе присо-

единения:

 with M do OP;

 Здесь М - имя записи, ОР - оператор, простой или составной.

Оператор ОР представляет собой область действия оператора присоедине-

ния, в пределах которой можно не использовать составные имена.

 Иногда содержимое отдельной записи зависит от значения одного из

ее полей. В языке ПАСКАЛЬ допускается описание записи, состоящей из

общей и вариантной частей. Вариантная часть задается с помощью конс-

трукции

 case P of,

где Р - имя поля из общей части записи. Возможные значения, прини-

маемые этим полем, перечисляются так же, как и в операторе варианта.

Однако вместо указания выполняемого действия, как это делается в опе-

раторе варианта, указываются поля варианта, заключенные в круглые

скобки. Описание вариантной части завершается служебным словом end.

 Тип поля Р можно указать в заголовке вариантной части, например:

 case P: Integer of

 Инициализация записей осуществляется с помощью типизированных

констант:

 type

 RecType= Record

 x,y: Word;

 ch: Char;

 dim: Array[1..3] of Byte

 end;

 const

 Rec: RecType= ( x: 127; y: 255;

 ch: 'A';

 dim: (2, 4, 8) );

{}

30. Ф А Й Л Ы

 Введение файлового типа в язык ПАСКАЛЬ вызвано необходимостью

обеспечить возможность работы с периферийными (внешними) устройствами

ЭВМ, предназначенными для ввода, вывода и хранения данных.

 Файловый тип данных или файл определяет упорядоченную совокупность

произвольного числа однотипных компонент.

 Общее свойство массива, множества и записи заключается в том, что

количество их компонент определено на этапе написания программы, тог-

да как количество компонент файла в тексте программы не определяется

и может быть произвольным.

 Понятие файла достаточно широко. Это может быть обычный файл на

диске, коммуникационный порт ЭВМ, устройство печати, клавиатура или

другие устройства.

 При работе с файлами выполняются операции ввода - вывода. Операция

ввода означает перепись данных с внешнего устройства (из входного

файла) в основную память ЭВМ, операция вывода - это пересылка данных

из основной памяти на внешнее устройство (в выходной файл).

 Файлы на внешних устройствах часто называют физическими файлами.

Их имена определяются операционной системой. В программах на языке

Паскаль имена файлов задаются с помощью строк. Например, имя файла на

диске может иметь вид:

 'A:LAB1.DAT'

 'c:\ABC150\pr.pas'

 'lab3.pas'.

 Операционная система MS-DOS не делает особого различия между фай-

лами на дисках и лентах и устройствами ЭВМ и портами коммуникаций. В

TURBO PASCAL могут использоваться имена устройств и портов, опреде-

ленные в MS-DOS, например:

 'CON', 'LPT1', 'PRN', 'COM1', 'AUX', 'NUL'.

 С файловой системой TURBO PASCAL связано понятие буфера ввода -

вывода. Ввод и вывод данных осуществляется через буфер. Буфер - это

область в памяти, которая выделяется для каждого файла. При записи в

файл вся информация сначала направляется в буфер и там накапливается

до тех пор, пока весь объем буфера не будет заполнен. Только после

этого или после специальной команды сброса происходит передача данных

на внешнее устройство. При чтении из файла данные вначале считываются

в буфер, причем данных считывается не столько, сколько запрашивается,

а сколько поместится в буфер.

 Механизм буферизации позволяет более быстро и эффективно обмени-

ваться информацией с внешними устройствами.

 Для работы с файлами в программе необходимо определить файловую

переменную. TURBO PASCAL поддерживает три файловых типа: текстовые

файлы, компонентные файлы, бестиповые файлы.

 Описание файловых переменных текстового типа производится с по-

мощью служебного слова Text, например:

 var tStory: Text;

 Описание компонентных файлов имеет вид:

 var fComp: File of T;

где T - тип компоненты файла. Примеры описания файловой переменной

компонентного типа:

 type M= array[1..500] of Longint;

 var f1: File of Real;

 f2: File of Integer;

 fLi: File of M;

 Бестиповые файлы описываются с помощью служебного слова File:

 var f: File;

 Файловые переменные, которые описаны в программе, называют логи-

ческими файлами. Все основные процедуры и функции, обеспечивающие

ввод - вывод данных, работают только с логическими файлами. Физичес-

кий файл должен быть связан с логическим до выполнения процедур отк-

рытия файлов.

 TURBO PASCAL вводит ряд процедур и функций, применимых для любых

типов файлов: Assign, Reset, Rewrite, Close, Rename, Erase, Eof,

IOResult.

 Процедура Assign( var f; FileName: String ) связывает логический

файл f с физическим файлом, полное имя которого задано в строке

FileName.

 Процедура Reset( var f ) открывает логический файл f для последую-

щего чтения данных или, как говорят, открывает входной файл. После

успешного выполнения процедуры Reset файл готов к чтению из него пер-

вого элемента.

 Процедура Rewrite( var f ) открывает логический файл f для после-

дующей записи данных (открывает выходной файл). После успешного вы-

полнения этой процедуры файл готов к записи в него первого элемента.

 Процедура Close( var f ) закрывает открытый до этого логический

файл. Вызов процедуры Close необходим при завершении работы с файлом.

Если по какой-то причине процедура Close не будет выполнена, файл

все-же будет создан на внешнем устройстве, но содержимое последнего

буфера в него не будет перенесено. Для входных файлов использование

оператора закрытия файла необязательно.

 Логическая функция EOF( var f ): Boolean возвращает значение TRUE,

когда при чтении достигнут конец файла. Это означает, что уже прочи-

тан последний элемент в файле или файл после открытия оказался пуст.

 Процедура Rename( var f; NewName: String ) позволяет переименовать

физический файл на диске, связанный с логическим файлом f. Переимено-

вание возможно после закрытия файла.

 Процедура Erase( var f ) уничтожает физический файл на диске, ко-

торый был связан с файловой переменной f. Файл к моменту вызова про-

цедуры Erase должен быть закрыт.

 Функция IOResult: Integer возвращает целое число, соответствующее

коду последней ошибки ввода - вывода. При нормальном завершении опе-

рации функция вернет значение 0. Значение функции IOResult необходимо

присваивать какой - либо переменной, так как при каждом вызове функ-

ция обнуляет свое значение. Функция IOResult работает только при вык-

люченном режиме проверок ошибок ввода - вывода или с ключом компиля-

ции {$I-}.

31. Т Е К С Т О В Ы Е Ф А Й Л Ы

 Особое место в языке ПАСКАЛЬ занимают текстовые файлы, компоненты

которых имеют символьный тип. Для описания текстовых файлов в языке

определен стандартный тип Тext:

 var TF1, TF2: Text;

 Текстовые файлы представляют собой последовательность строк, а

строки - последовательность символов. Строки имеют переменную длину,

каждая строка завершается признаком конца строки.

 С признаком конца строки связана функция EOLn(var T:Text):Boolean,

где Т - имя текстового файла. Эта функция принимает значение TRUE,

если достигнут конец строки, и значение FALSE, если конец строки не

достигнут.

 Для операций над текстовыми файлами, кроме перечисленных, опреде-

лены также операторы обращения к процедурам:

 ReadLn(T) - пропускает строку до начала следующей;

 WriteLn(T) - завершает строку файла, в которую производится за-

пись, признаком конца строки и переходит к началу следующей.

 Для работы с текстовыми файлами введена расширенная форма операто-

ров ввода и вывода. Оператор

 Read(T,X1,X2,...XK)

 эквивалентен группе операторов

 begin

 Read(T,X1);

 Read(T,X2);

 ...........

 Read(T,XK)

 end;

 Здесь Т - текстовый файл, а переменные Х1, Х2,...ХК могут быть ли-

бо переменными целого, действительного или символьного типа, либо

строкой. При чтении значений переменных из файла они преобразуются из

текстового представления в машинное.

 Оператор

 Write(T,X1,X2,...XK)

эквивалентен группе операторов

 begin

 Write(T,X1);

 Write(T,X2);

 ...........

 Write(T,XK)

 end;

 Здесь Т - также текстовый файл, но переменные Х1,Х2,...ХК могут

быть целого, действительного, символьного, логического типа или стро-

кой. При записи значений переменных в файл они преобразуются из внут-

реннего представления в текстовый.

 К текстовым файлам относятся стандартные файлы INPUT, OUTPUT.

 Рассмотренные ранее операторы ввода - вывода являются частным слу-

чаем операторов обмена с текстовыми файлами, когда используются стан-

дартные файлы ввода - вывода INPUT, OUTPUT.

 Работа с этими файлами имеет особенности:

 -имена этих файлов в списках ввода - вывода не указываются;

 -применение процедур Reset, Rewrite и Close к стандартным файлам

ввода - вывода запрещено;

 -для работы с файлами INPUT, OUTPUT введена разновидность функции

EOLn без параметров.

 TURBO PASCAL вводит дополнительные процедуры и функции, применимые

только к текстовым файлам, это SetTextBuf, Append, Flush, SeekEOLn,

SeekEOF.

 Процедура SetTextBuf( var f: Text; var Buf; BufSize: Word ) служит

для увеличения или уменьшения буфера ввода - вывода текстового файла

f. Значение размера буфера для текстовых файлов по умолчанию равно

128 байтам. Увеличение размера буфера сокращает количество обращений

к диску. Рекомендуется изменять разиер буфера до открытия файла. Бу-

фер файла начнется с первого байта переменной Buf. Размер буфера за-

дается в необязательном параметре BufSize, а если этот параметр от-

сутствует, размер буфера определяется длиной переменной Buf.

 Процедура Append( var f: Text ) служит для специального открытия

выходных файлов. Она применима к уже существующим физическим файлам и

открывает из для дозаписи в конец файла.

 Процедура Flush( var f: Text ) применяется к открытым выходным

файлам. Она принудительно записывает данные из буфера в файл незави-

симо от степени его заполнения.

 Функция SeekEOLn( var f: Text ): Boolean возвращает значение True,

если до конца строки остались только пробелы.

 Функция SeekEOF( var f: Text ): Boolean возвращает значение True,

если до конца файла остались строки, заполненные пробелами.

32. К О М П О Н Е Н Т Н Ы Е Ф А Й Л Ы

 Компонентный или типизированный файл - это файл с объявленным ти-

пом его компонент. Компонентные файлы состоят из машинных представле-

ний значений переменных, они хранят данные в том же виде, что и па-

мять ЭВМ.

 Описание величин файлового типа имеет вид:

 type M= File Of T;

где М - имя файлового типа, Т - тип компоненты. Например:

 type

 FIO= String[20];

 SPISOK=File of FIO;

 var

 STUD, PREP: SPISOK;

 Здесь STUD, PREP - имена файлов, компонентами которых являются

строки.

 Описание файлов можно задавать в разделе описания переменных:

 var

 fsimv: File of Char;

 fr: File of Real;

 Компонентами файла могут быть все скалярные типы, а из структури-

рованных - массивы, множества, записи. Практически во всех конкретных

реализациях языка ПАСКАЛЬ конструкция "файл файлов" недопустима.

 Все операции над компонентными файлами производятся с помощью

стандартных процедур:

 Reset, Rewrite, Read, Write, Close.

 Для ввода - вывода используются процедуры:

 Read(f,X);

 Write(f,X);

 где f - имя логического файла, Х - либо переменная, либо массив,

либо строка, либо множество, либо запись с таким же описанием, какое

имеет компонента файла.

 Выполнение процедуры Read(f,X) состоит в чтении с внешнего уст-

ройства одной компоненты файла и запись ее в X. Повторное применение

процедуры Read(f,X) обеспечит чтение следующей компоненты файла и за-

пись ее в X.

 Выполнение процедуры Write(f,X) состоит в записи X на внешнее уст-

ройство как одной компоненты. Повторное применение этой процедуры

обеспечит запись X как следующей компоненты файла.

 Для работы с компонентными файлами введена расширенная форма опе-

раторов ввода и вывода:

 Read(f,X1,X2,...XK)

 Write(f,X1,X2,...XK)

 Здесь f - компонентный файл, а переменные Х1, Х2,...ХК должны

иметь тот-же тип, что и объявленный тип компонент файла f.

33. Б Е С Т И П О В Ы Е Ф А Й Л Ы

 Бестиповые файлы позволяют записывать на диск произвольные участки

пвмяти ЭВМ и считывать их с диска в память. Операции обмена с бести-

повыми файлами осуществляется с помощью процедур BlokRead и

BlockWrite. Кроме того, вводится расширенная форма процедур Reset и

Rewrite. В остальном принципы работы остаются такими же, как и с ком-

понентными файлами.

 Перед использованием логический файл

 var f: File;

должен быть связан с физическим с помощью процедуры Assign. Далее

файл должен быть открыт для чтения или для записи процедурой Reset

или Rewrite, а после окончания работы закрыт процедурой Close.

 При открытии файла длина буфера устанавливается по умолчанию в 128

байт. TURBO PASCAL позволяет изменить размер буфера ввода - вывода,

для чего следует открывать файл расширенной записью процедур

 Reset(var f: File; BufSize: Word )

или

 Rewrite(var f: File; BufSize: Word )

 Параметр BufSize задает число байтов, считываемых из файла или за-

писываемых в него за одно обращение. Минимальное значение BufSize - 1

байт, максимальное - 64 К байт.

 Чтение данных из бестипового файла осуществляется процедурой

 BlockRead( var f: File; var X; Count: Word; var QuantBlock: Word );

 Эта процедура осуществляет за одно обращение чтение в переменную X

количества блоков, заданное параметром Count, при этом длина блока

равна длине буфера. Значение Count не может быть меньше 1. За одно

обращение нельзя прочесть больше, чем 64 К байтов.

 Необязательный параметр QuantBlock возвращает число блоков (буфе-

ров), прочитанных текущей операцией BlockRead. В случае успешного за-

вершения операции чтения QuantBlock = Count, в случае аварийной ситу-

ации параметр QuantBlock будет содержать число удачно прочитанных

блоков. Отсюда следует, что с помощью параметра QuantBlock можно

контролировать правильность выполнения операции чтения.

 Запись данных в бестиповой файл выполняется процедурой

 BlockWrite( var f: File; var X; Count: Word; var QuantBlock: Word );

которая осуществляет за одно обращение запись из переменной X коли-

чества блоков, заданное параметром Count, при этом длина блока равна

длине буфера.

 Необязательный параметр QuantBlock возвращает число блоков (буфе-

ров), записанных успешно текущей операцией BlockWrite.

34. П О С Л Е Д О В А Т Е Л Ь Н Ы Й И П Р Я М О Й

Д О С Т У П

 Смысл последовательного доступа заключается в том, что в каждый

момент времени доступна лишь одна компонента из всей последователь-

ности. Для того, чтобы обратиться (получить доступ) к компоненте с

номером К, необходимо просмотреть от начала файла К-1 предшествующую

компоненту. После обращения к компоненте с номером К можно обращаться

к компоненте с номером К+1. Отсюда следует, что процессы формирования

(записи) компонент файла и просмотра (чтения) не могут произвольно

чередоваться. Таким образом, файл вначале строится при помощи после-

довательного добавления компонент в конец, а затем может последова-

тельно просматриваться от начала до конца.

 Рассмотренные ранее средства работы с файлами обеспечивают после-

довательный доступ.

 TURBO PASCAL позволяет применять к компонентным и бестиповым фай-

лам, записанным на диск, способ прямого доступа. Прямой доступ озна-

чает возможность заранее определить в файле блок, к которому будет

применена операция ввода - вывода. В случае бестиповых файлов блок

равен размеру буфера, для компонентных файлов блок - это одна компо-

нента файла.

 Прямой доступ предполагает, что файл представляет собой линейную

последовательность блоков. Если файл содержит n блоков, то они нуме-

руются от 1 через 1 до n. Кроме того, вводится понятие условной гра-

ницы между блоками, при этом условная граница с номером 0 расположена

перед блоком с номером 1, граница с номером 1 расположена перед бло-

ком с номером 2 и, наконец, условная граница с номером n находится

после блока с номером n.

 Реализация прямого доступа осуществляется с помощью функций и про-

цедур FileSize, FilePos, Seek и Truncate.

 Функция FileSize( var f ): Longint возвращает количество блоков в

открытом файле f.

 Функция FilePos( var f ): Longint возвращает текущую позицию в

файле f. Позиция в файле - это номер условной границы. Для только что

открытого файла текущей позицией будет граница с номером 0. Это зна-

чит, что можно записать или прочесть блок с номером 1. После чтения

или записи первого блока текущая позиция переместится на границу с

номером 1, и можно будет обращаться к ьлоку с номером 2. После проч-

тения последней записи значение FilePos равно значению FileSize.

 Процедура Seek( var f; N: Longint) обеспечивает назначение текущей

позиции в файле (позиционирование). В параметре N должен быть задан

номер условной границы, предшествующей блоку, к которому будет произ-

водиться последующее обращение. Например, чтобы работать с блоком 4,

необходимо задать значение N, равное 3. Процедура Seek работает с от-

крытыми файлами.

 Процедура Truncate( var f ) устанавливает в текущей позиции приз-

нак конца файла и удаляет (стирает) все последующие блоки.

 Пример. Пусть на НМД имеется текстовый файл ID.DAT, который содер-

жит числовые значения действительного типа по два числа в каждой

строке - значения аргумента и функции соответственно. Количество пар

чисел не более 200. Составить программу, которая читает файл, значе-

ния аргумента и функции записывает в одномерные массивы, подсчитывает

их количество, выводит на экран дисплея и записывает в файл компо-

нентного типа RD.DAT.

 Program F;

 var

 rArg, rF: Array[1..200] of Real;

 inf: Text;

 outf: File of Real;

 n, l: Integer;

 begin

 Assign(inf,'ID.DAT');

 Assign(outf,'RD.DAT');

 Reset(inf);

 Rewrite(outf);

 n:=0;

 while not EOF(inf) do

 begin

 n:=n+1;

 ReadLn(inf,rArg[n],rF[n])

 end;

 for l:=1 to n do

 begin

 WriteLn(l:2,rArg[l]:8:2,rF[l]:8:2);

 Write(outf,rArg[l], rF[l]);

 end;

 close(outf)

 end.

35. У К А З А Т Е Л И.

 Операционная система MS - DOS все адресуемое пространство делит на

сегменты. Сегмент - это участок памяти размером 64 К байт. Для зада-

ния адреса необходимо определить адрес начала сегмента и смещение от-

носительно начала сегмента.

 В TURBO PASCAL определен адресный тип Pointer - указатель. Пере-

менные типа Pointer

 var p: Pointer;

содержат адрес какого - либо элемента программы и занимают 4 байта,

при этом адрес хранится как два слова, одно из них определяет сег-

мент, второе - смещение.

 Переменную типа указатель можно описать другим способом.

 type NameType= ^T;

 var p: NameType;

 Здесь p - переменная типа указатель, связанная с типом Т с помощью

имени типа NameType. Описать переменную типа указатель можно непос-

редственно в разделе описания переменных:

 var p: ^T;

 Необходимо различать переменную типа указатель и переменную, на

которую этот указатель ссылается. Например если p - ссылка на пере-

менную типа Т, то p^ - обозначение этой самой переменной.

 Для переменных типа указатель введено стандартное значение NIL,

которое означает, что указатель не ссылается ни к какому объекту.

Константа NIL используется для любых указателей.

 Над указателями не определено никаких операций, кроме проверки на

равенство и неравенство.

 Переменные типа указатель могут быть записаны в левой части опера-

тора присваивания, при этом в правой части может находиться либо

функция определения адреса Addr(X), либо выражение @ X, где @ - унар-

ная операция взятия адреса, X - имя переменной любого типа, в том

числе процедурного.

 Переменные типа указатель не могут быть элементами списка ввода -

вывода.

36. Д И Н А М И Ч Е С К И Е П Е Р Е М Е Н Н Ы Е

 Статической переменной (статически размещенной) называется описан-

ная явным образом в программе переменная, обращение к ней осуществля-

ется по имени. Место в памяти для размещения статических переменных

определяется при компиляции программы.

 В отличие от таких статических переменных в программах, написанных

на языке ПАСКАЛЬ, могут быть созданы динамические переменные. Основ-

ное свойство динамических переменных заключается в том, что они соз-

даются и память для них выделяется во время выполнения программы.

Размещаются динамические переменные в динамической области памяти

(heap - области).

 Динамическая переменная не указывается явно в описаниях переменных

и к ней нельзя обратиться по имени. Доступ к таким переменным осу-

ществляется с помощью указателей и ссылок.

 Работа с динамической областью памяти в TURBO PASCAL реализуется с

помощью процедур и функций New, Dispose, GetMem, FreeMem, Mark,

Release, MaxAvail, MemAvail, SizeOf.

 Процедура New( var p: Pointer ) выделяет место в динамической об-

ласти памяти для размещения динамической переменной p^ и ее адрес

присваивает указателю p.

 Процедура Dispose( var p: Pointer ) освобождает участок памяти,

выделенный для размещения динамической переменной процедурой New, и

значение указателя p становится неопределенным.

 Проуедура GetMem( var p: Pointer; size: Word ) выделяет участок

памяти в heap - области, присваивает адрес его начала указателю p,

размер участка в байтах задается параметром size.

 Процедура FreeMem( var p: Pointer; size: Word ) освобождает учас-

ток памяти, адрес начала которого определен указателем p, а размер -

параметром size. Значение указателя p становится неопределенным.

 Процедура Mark( var p: Pointer ) записывает в указатель p адрес

начала участка свободной динамической памяти на момент ее вызова.

 Процедура Release( var p: Pointer ) освобождает участок динамичес-

кой памяти, начиная с адреса, записанного в указатель p процедурой

Mark, то-есть, очищает ту динамическую память, которая была занята

после вызова процедуры Mark.

 Функция MaxAvail: Longint возвращает длину в байтах самого длинно-

го свободного участка динамической памяти.

 Функция MemAvail: Longint полный объем свободной динамической па-

мяти в байтах.

 Вспомогательная функция SizeOf( X ): Word возвращает объем в бай-

тах, занимаемый X, причем X может быть либо именем переменной любого

типа, либо именем типа.

 Рассмотрим некоторые примеры работы с указателями.

 var

 p1, p2: ^Integer;

 Здесь p1 и p2 - указатели или пременные ссылочного типа.

 p1:=NIL; p2:=NIL;

 После выполнения этих операторов присваивания указатели p1 и p2 не

будут ссылаться ни на какой конкретный объект.

 New(p1); New(p2);

 Процедура New(p1) выполняет следующие действия:

 -в памяти ЭВМ выделяется участок для размещения величины целого

типа;

 -адрес этого участка присваивается переменной p1:

 г=====¬ г=====¬

 ¦ \*--¦--------->¦ ¦

 L=====- L=====-

 p1 p1^

 Аналогично, процедура New(p2) обеспечит выделение участка памяти,

адрес которого будет записан в p2:

 г=====¬ г=====¬

 ¦ \*--¦--------->¦ ¦

 L=====- L=====-

 p2 p2^

 После выполнения операторов присваивания

 p1^:=2; p2^:=4;

в выделенные участки памяти будут записаны значения 2 и 4 соответ-

ственно:

 г=====¬ г=====¬

 ¦ \*--¦--------->¦ 2 ¦

 L=====- L=====-

 p1 p1^

 г=====¬ г=====¬

 ¦ \*--¦--------->¦ 4 ¦

 L=====- L=====-

 p2 p2^

 В результате выполнения оператора присваивания

 p1^:=p2^;

в участок памяти, на который ссылается указатель p1, будет записано

значение 4:

 г=====¬ г=====¬

 ¦ \*--¦--------->¦ 4 ¦

 L=====- L=====-

 p1 p1^

 г=====¬ г=====¬

 ¦ \*--¦--------->¦ 4 ¦

 L=====- L=====-

 p2 p2^

 После выполнения оператора присваивания

 p2:=p1;

оба указателя будут содержать адрес первого участка памяти:

 г=====¬ г=====¬

 ¦ \*--¦--------->¦ 4 ¦

 L=====- --->L=====-

 p1 ¦ p1^ p2^

 ¦

 г=====¬ ¦

 ¦ \*--¦-------

 L=====-

 p2

 Переменные p1^, p2^ являются динамическими, так как память для них

выделяется в процессе выполнения программы с помощью процедуры New.

 Динамические переменные могут входить в состав выражений, напри-

мер:

 p1^:=p1^+8; Write('p1^=',p1^:3);

 Пример. В результате выполнения программы:

 Program DemoPointer;

 var p1,p2,p3:^Integer;

 begin

 p1:=NIL; p2:=NIL; p3:=NIL;

 New(p1); New(p2); New(p3);

 p1^:=2; p2^:=4;

 p3^:=p1^+Sqr(p2^);

 writeln('p1^=',p1^:3,' p2^=',p2^:3,' p3^=',p3^:3);

 p1:=p2;

 writeln('p1^=',p1^:3,' p2^=',p2^:3)

 end.

на экран дисплея будут выведены результаты:

p1^= 2 p2^= 4 p3^= 18

p1^= 4 p2^= 4

37. Д И Н А М И Ч Е С К И Е С Т Р У К Т У Р Ы

Д А Н Н Ы Х

 Структурированные типы данных, такие, как массивы, множества, за-

писи, представляют собой статические структуры, так как их размеры

неизменны в течение всего времени выполнения программы.

 Часто требуется, чтобы структуры данных меняли свои размеры в ходе

решения задачи. Такие структуры данных называются динамическими, к

ним относятся стеки, очереди, списки, деревья и другие. Описание ди-

намических структур с помощью массивов, записей и файлов приводит к

неэкономному использованию памяти ЭВМ и увеличивает время решения за-

дач.

 Каждая компонента любой динамической структуры представляет собой

запись, содержащую по крайней мере два поля: одно поле типа указа-

тель, а второе - для размещения данных. В общем случае запись может

содержать не один, а несколько укзателей и несколько полей данных.

Поле данных может быть переменной, массивом, множеством или записью.

 Для дальнейшего рассмотрения представим отдельную компоненту в ви-

де:

 г=====¬

 ¦ D ¦

 ¦=====¦

 ¦ p ¦

 L=====-

где поле p - указатель;

 поле D - данные.

 Описание этой компоненты дадим следующим образом:

 type

 Pointer = ^Comp;

 Comp = record

 D:T;

 pNext:Pointer

 end;

здесь T - тип данных.

 Рассмотрим основные правила работы с динамическими структурами

данных типа стек, очередь и список, базируясь на приведенное описание

компоненты.

38. С Т Е К И

 Стеком называется динамическая структура данных, добавление компо-

ненты в которую и исключение компоненты из которой производится из

одного конца, называемого вершиной стека. Стек работает по принципу

 LIFO (Last-In, First-Out) -

поступивший последним, обслуживается первым.

 Обычно над стеками выполняется три операции:

 -начальное формирование стека (запись первой компоненты);

 -добавление компоненты в стек;

 -выборка компоненты (удаление).

 Для формирования стека и работы с ним необходимо иметь две пере-

менные типа указатель, первая из которых определяет вершину стека, а

вторая - вспомогательная. Пусть описание этих переменных имеет вид:

 var pTop, pAux: Pointer;

где pTop - указатель вершины стека;

 pAux - вспомогательный указатель.

 Начальное формирование стека выполняется следующими операторами:

 г=====¬ г=====¬

 New(pTop); ¦ \*--¦---¬ ¦ ¦

 L=====- ¦ ¦=====¦

 pTop L-->¦ ¦

 L=====-

 г=====¬ г=====¬

 pTop^.pNext:=NIL; ¦ \*--¦---¬ ¦ ¦

 L=====- ¦ ¦=====¦

 pTop L-->¦ NIL ¦

 L=====-

 г=====¬ г=====¬

 pTop^.D:=D1; ¦ \*--¦---¬ ¦ D1 ¦

 L=====- ¦ ¦=====¦

 pTop L-->¦ NIL ¦

 L=====-

 Последний оператор или группа операторов записывает содержимое

поля данных первой компоненты.

 Добавление компоненты в стек призводится с использованием вспо-

могательного указателя:

 г=====¬ г=====¬ г=====¬

 New(pAux); ¦ \*--¦---¬ ¦ ¦ ----¦--\* ¦

 L=====- ¦ ¦=====¦ ¦ L=====-

 pTop ¦ ¦ ¦<--- pAux

 ¦ L=====-

 ¦

 ¦ г=====¬

 ¦ ¦ D1 ¦

 ¦ ¦=====¦

 L-->¦ NIL ¦

 L=====-

 г=====¬ г=====¬ г=====¬

 pAux^.pNext:=pTop; ¦ \*--¦---¬ ¦ ¦ ----¦--\* ¦

 L=====- ¦ ¦=====¦<--- L=====-

 pTop ¦ ¦ \*--¦-¬ pAux

 ¦ L=====- ¦

 ¦ ¦

 ¦ г=====¬ ¦

 ¦ ¦ D1 ¦ ¦

 ¦ ¦=====¦ ¦

 L-->¦ NIL ¦<-

 L=====-

 г=====¬ г=====¬ г=====¬

 pTop:=pAux; ¦ \*--¦---¬ ¦ ¦ ----¦--\* ¦

 L=====- ¦ ¦=====¦<--- L=====-

 pTop L-->¦ \*--¦-¬ pAux

 L=====- ¦

 ¦

 г=====¬ ¦

 ¦ D1 ¦ ¦

 ¦=====¦ ¦

 ¦ NIL ¦<-

 L=====-

 г=====¬ г=====¬

 pTop^.D:=D2; ¦ \*--¦---¬ ¦ D2 ¦

 L=====- ¦ ¦=====¦

 pTop L-->¦ \*--¦-¬

 L=====- ¦

 ¦

 г=====¬ ¦

 ¦ D1 ¦ ¦

 ¦=====¦ ¦

 ¦ NIL ¦<-

 L=====-

 Добавление последующих компонент производится аналогично.

 Рассмотрим процесс выборки компонент из стека. Пусть к моменту на-

чала выборки стек содержит три компоненты:

 г=====¬ г=====¬

 ¦ \*--¦---¬ ¦ D3 ¦

 L=====- ¦ ¦=====¦

 pTop L-->¦ \*--¦-¬

 L=====- ¦

 ¦

 г=====¬ ¦

 ¦ D2 ¦ ¦

 ¦=====¦ ¦

 --¦--\* ¦<-

 ¦ L=====-

 ¦

 ¦ г=====¬

 ¦ ¦ D1 ¦

 ¦ ¦=====¦

 L>¦ NIL ¦

 L=====-

 Первый оператор или группа операторов осуществляет чтение данных

из компоненты - вершины стека. Второй оператор изменяет значение ука-

зателя вершины стека:

 г=====¬ г=====¬

 D3:=pTop^.D; ¦ \*--¦---¬ ¦ D3 ¦

 pTop:=pTop^.pNext; L=====- ¦ ¦=====¦

 pTop ¦ ¦ ¦

 ¦ L=====-

 ¦

 ¦ г=====¬

 ¦ ¦ D2 ¦

 ¦ ¦=====¦

 L-->¦ \*--¦-¬

 L=====- ¦

 ¦

 г=====¬ ¦

 ¦ D1 ¦ ¦

 ¦=====¦ ¦

 ¦ NIL ¦<-

 L=====-

 Как видно из рисунка, при чтении компонента удаляется из стека.

 Пример. Составить программу, которая формирует стек, добавляет в

него произвольное количество компонент, а затем читает все компоненты

и выводит их на экран дисплея, В качестве данных взять строку симво-

лов. Ввод данных - с клавиатуры дисплея, признак конца ввода - строка

символов END.

 Program STACK;

 uses Crt;

 type

 Alfa= String[10];

 PComp= ^Comp;

 Comp= Record

 sD: Alfa;

 pNext: PComp

 end;

 var

 pTop: PComp;

 sC: Alfa;

 Procedure CreateStack(var pTop: PComp; var sC: Alfa);

 begin

 New(pTop);

 pTop^.pNext:=NIL;

 pTop^.sD:=sC

 end;

 Procedure AddComp(var pTop: PComp; var sC: Alfa);

 var pAux: PComp;

 begin

 NEW(pAux);

 pAux^.pNext:=pTop;

 pTop:=pAux;

 pTop^.sD:=sC

 end;

 Procedure DelComp(var pTop: PComp; var sC:ALFA);

 begin

 sC:=pTop^.sD;

 pTop:=pTop^.pNext

 end;

 begin

 Clrscr;

 writeln(' ВВЕДИ СТРОКУ ');

 readln(sC);

 CreateStack(pTop,sC);

 repeat

 writeln(' ВВЕДИ СТРОКУ ');

 readln(sC);

 AddComp(pTop,sC)

 until sC='END';

 writeln('\*\*\*\*\*\* ВЫВОД РЕЗУЛЬТАТОВ \*\*\*\*\*\*');

 repeat

 DelComp(pTop,sC);

 writeln(sC);

 until pTop = NIL

 end.

39. О Ч Е Р Е Д И

 Очередью называется динамическая структура данных, добавление ком-

поненты в которую производится в один конец, а выборка осуществляется

с другого конца. Очередь работает по принципу:

 FIFO (First-In, First-Out) -

поступивший первым, обслуживается первым.

 Для формирования очереди и работы с ней необходимо иметь три пере-

менные типа указатель, первая из которых определяет начало очереди,

вторая - конец очереди, третья - вспомогательная.

 Описание компоненты очереди и переменных типа указатель дадим сле-

дующим образом:

 type

 PComp=^Comp;

 Comp=record

 D:T;

 pNext:PComp

 end;

 var

 pBegin, pEnd, pAux: PComp;

где pBegin - указатель начала очереди, pEnd - указатель конца очере-

ди, pAux - вспомогательный указатель.

 Тип Т определяет тип данных компоненты очереди.

 Начальное формирование очереди выполняется следующими операторами:

 г=====¬ г=====¬ г=====¬

 New(pBegin); ¦ \*--¦---¬ ¦ ¦ ¦ ¦

 L=====- ¦ ¦=====¦ L=====-

 pBegin L-->¦ ¦ pEnd

 L=====-

 г=====¬ г=====¬ г=====¬

 pBegin^.pNext:=NIL; ¦ \*--¦---¬ ¦ ¦ ¦ ¦

 L=====- ¦ ¦=====¦ L=====-

 pBegin L-->¦ NIL ¦ pEnd

 L=====-

 г=====¬ г=====¬ г=====¬

 pBegin^.D:=D1; ¦ \*--¦---¬ ¦ D1 ¦ ¦ ¦

 L=====- ¦ ¦=====¦ L=====-

 pBegin L-->¦ NIL ¦ pEnd

 L=====-

 г=====¬ г=====¬ г=====¬

 pEnd:=pBegin; ¦ \*--¦---¬ ¦ D1 ¦ ----¦--\* ¦

 L=====- ¦ ¦=====¦ ¦ L=====-

 pBegin L-->¦ NIL ¦<--- pEnd

 L=====-

 Добавление компоненты в очередь производится в конец очереди:

 New(pAux);

г=====¬ г=====¬ г=====¬ г=====¬ г=====¬

¦ \*--¦---¬ ¦ D1 ¦ ----¦--\* ¦ ¦ ¦ ----¦--\* ¦

L=====- ¦ ¦=====¦ ¦ L=====- ¦=====¦ ¦ L=====-

pBegin L-->¦ NIL ¦<--- pEnd ¦ ¦<--- pAux

 L=====- L=====-

 pAux^.pNext:=NIL;

г=====¬ г=====¬ г=====¬ г=====¬ г=====¬

¦ \*--¦---¬ ¦ D1 ¦ ----¦--\* ¦ ¦ ¦ ----¦--\* ¦

L=====- ¦ ¦=====¦ ¦ L=====- ¦=====¦ ¦ L=====-

pBegin L-->¦ NIL ¦<--- pEnd ¦ NIL ¦<--- pAux

 L=====- L=====-

 pBegin^.pNext:=pAux;

г=====¬ г=====¬ г=====¬ г=====¬ г=====¬

¦ \*--¦---¬ ¦ D1 ¦ ----¦--\* ¦ ¦ ¦ ----¦--\* ¦

L=====- ¦ ¦=====¦ ¦ L=====- ¦=====¦ ¦ L=====-

pBegin L-->¦ \* ¦<--- pEnd ¦ NIL ¦<--- pAux

 L=====- L=====-

 ¦ ^

 ¦ ¦

 L----------------------------

 pEnd:=pAux;

г=====¬ г=====¬ г=====¬ г=====¬ г=====¬

¦ \*--¦---¬ ¦ D1 ¦ ¦ \*--¦---¬ ¦ ¦ ----¦--\* ¦

L=====- ¦ ¦=====¦ L=====- ¦ ¦=====¦ ¦ L=====-

pBegin L-->¦ \* ¦ pEnd L-->¦ NIL ¦<--- pAux

 L=====- L=====-

 ¦ ^

 ¦ ¦

 L----------------------------

 pEnd^.D:=D2;

г=====¬ г=====¬ г=====¬ г=====¬

¦ \*--¦---¬ ¦ D1 ¦ ¦ D2 ¦ ----¦--\* ¦

L=====- ¦ ¦=====¦ ¦=====¦ ¦ L=====-

pBegin L-->¦ \*--¦-------------------->¦ NIL ¦<--- pEnd

 L=====- L=====-

 Добавление последующих компонент производится аналогично.

 Выборка компоненты из очереди осуществляется из начала очереди,

одновременно компонента исключается из очереди. Пусть в памяти ЭВМ

сформирована очередь, состоящая из трех элементов:

г=====¬ г=====¬ г=====¬ г=====¬ г=====¬

¦ \*--¦---¬ ¦ D1 ¦ ¦ D2 ¦ ¦ D3 ¦ ----¦--\* ¦

L=====- ¦ ¦=====¦ ¦=====¦ ¦=====¦ ¦ L=====-

pBegin L-->¦ \*--¦------>¦ \*--¦------>¦ NIL ¦<--- pEnd

 L=====- L=====- L=====-

 Выборка компоненты выполняется следующими операторами:

 D1:=pBegin^.D;

 pBegin:=pBegin^.pNext;

г=====¬ г=====¬ г=====¬ г=====¬ г=====¬

¦ \*--¦---¬ ¦ D1 ¦ ¦ D2 ¦ ¦ D3 ¦ ----¦--\* ¦

L=====- ¦ ¦=====¦ ¦=====¦ ¦=====¦ ¦ L=====-

pBegin ¦ ¦ ¦ --->¦ \*--¦------>¦ NIL ¦<--- pEnd

 ¦ L=====- ¦ L=====- L=====-

 ¦ ¦

 L--------------

 Пример. Составить программу, которая формирует очередь, добавляет

в нее произвольное количество компонент, а затем читает все компонен-

ты и выводит их на экран дисплея. В качестве данных взять строку сим-

волов. Ввод данных - с клавиатуры дисплея, признак конца ввода -

строка символов END.

 Program QUEUE;

 uses Crt;

 type

 Alfa= String[10];

 PComp= ^Comp;

 Comp= record

 sD:Alfa;

 pNext:PComp

 end;

 var

 pBegin, pEnd: PComp;

 sC: Alfa;

 Procedure CreateQueue(var pBegin,pEnd: PComp; var sC: Alfa);

 begin

 New(pBegin);

 pBegin^.pNext:=NIL;

 pBegin^.sD:=sC;

 pEnd:=pBegin

 end;

 Procedure AddQueue(var pEnd:PComp; var sC:Alfa);

 var pAux: PComp;

 begin

 New(pAux);

 pAux^.pNext:=NIL;

 pEnd^.pNext:=pAux;

 pEnd:=pAux;

 pEnd^.sD:=sC

 end;

 Procedure DelQueue(var pBegin: PComp; var sC: Alfa);

 begin

 sC:=pBegin^.sD;

 pBegin:=pBegin^.pNext

 end;

 begin

 Clrscr;

 writeln(' ВВЕДИ СТРОКУ ');

 readln(sC);

 CreateQueue(pBegin,pEnd,sC);

 repeat

 writeln(' ВВЕДИ СТРОКУ ');

 readln(sC);

 AddQueue(pEnd,sC)

 until sC='END';

 writeln(' \*\*\*\*\* ВЫВОД РЕЗУЛЬТАТОВ \*\*\*\*\*');

 repeat

 DelQueue(pBegin,sC);

 writeln(sC);

 until pBegin=NIL

 end.

40. Л И Н Е Й Н Ы Е С П И С К И

 В стеки или очереди компоненты можно добавлять только в какой -

либо один конец структуры данных, это относится и к извлечению компо-

нент.

 Связный (линейный) список является структурой данных, в произволь-

но выбранное место которого могут включаться данные, а также изымать-

ся оттуда.

 Каждая компонента списка определяется ключом. Обычно ключ - либо

число, либо строка символов. Ключ располагается в поле данных компо-

ненты, он может занимать как отдельное поле записи, так и быть частью

поля записи.

 Основные отличия связного списка от стека и очереди следующие:

 -для чтения доступна любая компонента списка;

 -новые компоненты можно добавлять в любое место списка;

 -при чтении компонента не удаляется из списка.

 Над списками выполняются следующие операции:

 -начальное формирование списка (запись первой компоненты);

 -добавление компоненты в конец списка;

 -чтение компоненты с заданным ключом;

 -вставка компоненты в заданное место списка (обычно после компо-

ненты с заданным ключом);

 -исключение компоненты с заданным ключом из списка.

 Для формирования списка и работы с ним необходимо иметь пять пере-

менных типа указатель, первая из которых определяет начало списка,

вторая - конец списка, остальные- вспомогательные.

 Описание компоненты списка и переменных типа указатель дадим сле-

дующим образом:

 type

 PComp= ^Comp;

 Comp= record

 D:T;

 pNext:PComp

 end;

 var

 pBegin, pEnd, pCKey, pPreComp, pAux: PComp;

где pBegin - указатель начала списка, pEnd - указатель конца списка,

pCKey, pPreComp, pAux - вспомогательные указатели.

 Начальное формирование списка, добавление компонент в конец списка

выполняется так же, как и при формировании очереди.

г=====¬ г=====¬ г=====¬ г=====¬ г=====¬ г=====¬

¦ \*--¦-¬ ¦ D1 ¦ ¦ D2 ¦ ¦ DN1 ¦ ¦ DN ¦ --¦--\* ¦

L=====- ¦ ¦=====¦ ¦=====¦ ¦=====¦ ¦=====¦ ¦ L=====-

pBegin L-->¦ \*--¦--->¦ \*--¦-....->¦ \*--¦--->¦ NIL ¦<-- pEnd

 L=====- L=====- L=====- L=====-

 Для чтения и вставки компоненты по ключу необходимо выполнить по-

иск компоненты с заданным ключом:

 pCKey:=pBegin;

 while (pCKey<>NIL) and (Key<>pCKey^.D) DO

 pCKey:=pCKey^.pNext;

Здесь Key - ключ, тип которого совпадает с типом данных компоненты.

 После выполнения этих операторов указатель pСKey будет определять

компоненту с заданным ключом или такая компонента не будет найдена.

 Пусть pCKey определяет компоненту с заданным ключом. Вставка новой

компоненты выполняется следующими операторами:

 New(pAux); г===¬

 pAux^.D:= DK1; ---¦-\* ¦

 ¦ L===-

 ¦ pCKey

 ¦

г===¬ г===¬ г===¬ г===¬ г===¬ г===¬

¦ \*-¦--¬ ¦D1 ¦ ¦Key¦ ¦KK1¦ ¦DN ¦ ---¦-\* ¦

L===- ¦ ¦===¦ ¦===¦ ¦===¦ ¦===¦ ¦ L===-

pBegin L->¦ \*-¦-...->¦ \*-¦---->¦ \*-¦-...->¦NIL¦<-- pEnd

 L===- L===- L===- L===-

 г===¬ г===¬

 ¦DK1¦ ---¦-\* ¦

 ¦===¦ ¦ L===-

 ¦ ¦<-- pAux

 L===-

 pAux^.pNext:=pCKey^.pNext;

 pCKey^.pNext:=pAux;

 г===¬

 ---¦-\* ¦

 ¦ L===-

 ¦ pCKey

 ¦

г===¬ г===¬ г===¬ г===¬ г===¬ г===¬

¦ \*-¦--¬ ¦D1 ¦ ¦Key¦ ¦KK1¦ ¦DN ¦ ---¦-\* ¦

L===- ¦ ¦===¦ ¦===¦ ¦===¦ ¦===¦ ¦ L===-

pBegin L->¦ \*-¦-...->¦ \* ¦ ¦ \*-¦-...->¦NIL¦<-- pEnd

 L===- L===- L===- L===-

 ¦ ^

 ¦ ¦ г===¬ г===¬

 ¦ ¦ ¦DK1¦ ---¦-\* ¦

 ¦ L----------¦===¦ ¦ L===-

 L------------------->¦-\* ¦<-- pAux

 L===-

 Для удаления компоненты с заданным ключом необходимо при поиске

нужной компоненты помнить адрес предшествующей:

 pCKey:=pBegin;

 while (pCKey<>NIL) and (Key<>pCKey^.D) do

 begin

 pPreComp:=pCKey;

 pCKey:=pCKey^.pNext

 end;

Здесь указатель pCKey определяет компоненту с заданным ключом, указа-

тель pPreComp содержит адрес предидущей компоненты.

 Удаление компоненты с ключом Key выполняется оператором:

 pPreComp^.pNext:=pCKey^.pNext;

 pPreComp pCKey

 г===¬ г===¬

 ¦ \* ¦ ¦ \* ¦

 L===- L===-

 ¦ ¦

 ¦ ¦

 ¦ ¦

г===¬ г===¬ г===¬ г===¬ г===¬ г===¬ г===¬

¦ \*-¦--¬ ¦D1 ¦ ¦KK1¦ ¦Key¦ ¦KK2¦ ¦DN ¦ ---¦-\* ¦

L===- ¦ ¦===¦ ¦===¦ ¦===¦ ¦===¦ ¦===¦ ¦ L===-

pBegin L->¦ \*-¦-...->¦ \*-¦-¬ ¦ \*-¦--->¦ \*-¦-...->¦NIL¦<-- pEnd

 L===- L===- ¦ L===- L===- L===-

 ¦ ^

 ¦ ¦

 L---------------

 Пример. Составить программу, которая формирует список, добавляет в

него произвольное количество компонент, выполняет вставку и удаление

компоненты по ключу, а затем читает и выводит весь список на экран

дисплея. В качестве данных взять строку символов. Ввод данных - с

клавиатуры дисплея, признак конца ввода - строка символов END.

 Program LISTLINKED;

 uses Crt;

 type

 Alfa= String[10];

 PComp= ^Comp;

 Comp= record

 sD:Alfa;

 pNext:PComp

 end;

 var

 pBegin, pEnd, pAux, pCKey, pPreComp: PComp;

 sC, sKey: Alfa;

 bCond: Boolean;

 Procedure CreateLL(var pBegin,pEnd: PComp; var sC: Alfa);

 begin

 New(pBegin);

 pBegin^.pNext:=NIL;

 pBegin^.sD:=sC;

 pEnd:=pBegin

 end;

 Procedure AddLL(var pEnd: PComp; var sC: Alfa);

 var pAux: PComp;

 begin

 New(pAux);

 pAux^.pNext:=NIL;

 pEnd^.pNext:=pAux;

 pEnd:=pAux;

 pEnd^.sD:=sC

 end;

 Procedure Find(var sKey: Alfa; var pBegin,pCKey,pPreComp: PComp;

 var bCond: Boolean);

 begin

 pCKey:=pBegin;

 while (pCKey <> NIL) and (sKey <> pCKey^.D) do

 begin

 pPreComp:=pCKey;

 pCKey:=pCKey^.pNext

 end;

 if (pCKey = NIL) and (sKey <> pCKey^.sD) then bCond:=FALSE

 else bCond:=TRUE

 end;

 Procedure InsComp(var sKey,sC: Alfa);

 var pAux:PComp;

 begin

 Find(sKey,pBegin,pCKey,pPreComp,bCond);

 New(pAux);

 pAux^.sD:=sC;

 pAux^.pNext:=pCKey^.pNext;

 pCKey^.pNext:=pAux

 end;

 Procedure DelComp(var sKey: Alfa; var pBegin: PComp);

 begin

 Find(sKey,pBegin,pCKey,pPreComp,bCond);

 pPreComp^.pNext:=pCKey^.pNext

 end;

 begin

 ClrScr;

 writeln(' ВВЕДИ СТРОКУ ');

 readln(sC);

 CreateLL(pBegin,pEnd,sC);

 repeat

 writeln('ВВЕДИ СТРОКУ ');

 readln(sC);

 AddLL(pEnd,sC)

 until sC='END';

 writeln(' \*\*\*\*\* ВЫВОД ИСХОДНОГО СПИСКА \*\*\*\*\*');

 pAux:=pBegin;

 repeat

 writeln(pAux^.sD);

 pAux:=pAux^.pNext;

 until pAux=NIL;

 writeln;

 writeln('ВВЕДИ КЛЮЧ ДЛЯ ВСТАВКИ СТРОКИ');

 readln(sKey);

 writeln('ВВЕДИ ВСТАВЛЯЕМУЮ СТРОКУ');

 readln(sC);

 InsComp(sKey,sC);

 writeln;

 writeln('ВВЕДИ КЛЮЧ УДАЛЯЕМОЙ СТРОКИ');

 readln(sKey);

 DelComp(sKey,pBegin);

 writeln;

 writeln(' \*\*\*\*\* ВЫВОД ИЗМЕНЕННОГО СПИСКА \*\*\*\*\*');

 pAux:=pBegin;

 repeat

 writeln(pAux^.sD);

 pAux:=pAux^.pNext;

 until pAux=NIL

 end.