

**Примечание ! Содержание этой книги переведено с румынского на русский язык не совсем компетентными “переводчиками”. Прошу не считать этот перевод официально-правильным. Советую использовать румынский оригинал с сайта или из библиотеки.**

Программирование

ЯЗЫК ПРОГРАММИРОВАНИЯ ПАСКАЛЬ

КУРС ЛЕКЦИЙ

Кишинёв 2005

## Тема 1. История появления компьютера.

Научные исследования, сделанные людьми, требуют много работы и правильный ход мысли. Многие из этих работ требуют вычислений. Во все времена математики пытались создать такие машины, которые бы производили эти вычисления, автоматически давая возможность людям заниматься другими делами. Но, при всех вложенных усилиях, машины автоматического подсчёта или другими словами, компьютеры, появились только в тот момент, когда технология позволила это, к середине 40 годов XX века. Но к этому моменту была создана база вычислительной техники.

В 1641г. математик и философ Блейз Паскаль создаёт вычислительную машину, которая могла производить две простые арифметические операции с шестью десятичными цифрами. Спустя десятки лет, в 1671 году, эта машина была улучшена математиком Вильгельмом Лейбницем, который добавил еще 4-ре арифметические операции.

В 1823 английский математик и экономист Чарльз Баббадж создает дифференциальную машину, которая была создана для математических таблиц необходимых в навигации. Компьютер мог производить только одну операцию. Потому требовалось создание компьютера общего пользования и в 1834 г. он создает аналитическую машину. В 1834 году Баббадж создаёт проект аналитической машины, которая состояла из базовых устройств современного компьютера:

- память;
- устройство подсчёта;
- устройство ввода – вывода.

Эта машина могла запомнить 1000 номеров по 50 десятичных цифр и производила сложение 2-х цифр за секунду, а умножение за одну минуту.

Первое устройство компьютера, имеющее команду «Program» создано немецким учёным Конрадом Зусом в 1941 году. Программа была записана на киноленту и читалась по сериям.

Компьютер был построен из 2600 реле и производил подсчёт 64 цифр по 22 бинарные цифры.

В 1946 году в США фирмой «Bell Telephone» было изобретено несколько компьютеров. Первый компьютер назывался Bell-v. Он был построен из 9000 реле и занимал площадь в 90 м<sup>2</sup> и весил 10 тонн.

В 1943-1946 появляется первый электронный компьютер с названием ENIAC (Electronic Numerical Integrator and Computer). Компьютер состоял из 18 электронных каналов и 1500 реле, он занимал площадь 135 м<sup>2</sup> и весил 30 тонн. Форма и структура работы современного компьютера были предложены американцем Ньюманом. В современный универсальный компьютер должно входить: арифметическое устройство, устройство команд, память и устройство входа-выхода. Таким образом, в память записываются не только обработанные данные, но и сама программа. Инструкция программы загружается в память, таким образом, как и преобразованные данные. После начала вычислительного процесса инструкция из памяти компьютера выводится и проверяется автоматически.

Развитие вычислительной техники со временем познало несколько этапов и несколько поколений компьютеров:

1. Первое поколение составляли те компьютеры, которые были спроектированные на базе электронных каналов (1959). Эти компьютеры имели большие объемы, низкую скорость подсчёта (более 1000 операций в секунду), низкий уровень памяти.
2. Второе поколение составляют компьютеры, произведенные в 1959-1964 гг. Место электронных каналов заняли транзисторы. Правда объём компьютера уменьшился, а скорость работы значительно увеличилась, также объём памяти увеличился, появилась возможность программирования на языках Assembler, Cobol, Fortron; и объём памяти стимулировал работу, направленную на автоматизацию программирования компьютеров. Появились первые операционные системы. Примеры компьютеров 2-го поколения: Минск- 22, Минск-23, БЭСМ-4, БЭСМ-6.
3. Третье поколение – появилось в 1964-1972 гг. На базе которых лежат системы, благодаря которым, размеры компьютеров уменьшились в несколько раз.

Созданные с 1972 по 1980 гг. компьютеры принадлежат четвертому поколению. Базовым элементом служит микропроцессор и сложные системы.

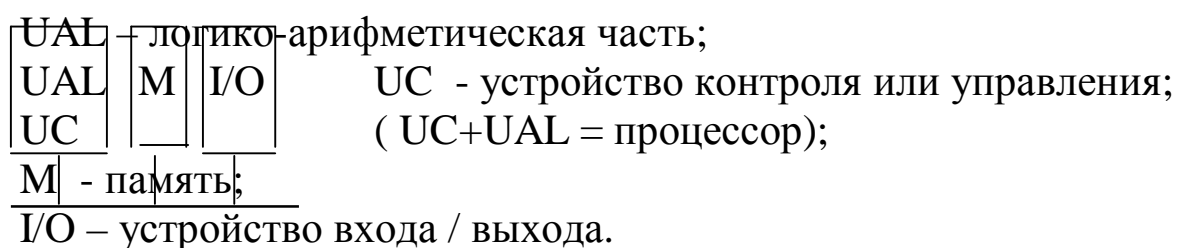
С 1980 года и по сегодняшний день развивается пятое поколение компьютеров, на базе которых лежат сложные системы, возможности для работы очень обширны. Эти системы состоят из сотни тысяч транзисторов маленьких размеров. Параметры, которые характеризуют

современный компьютер: оперативная память «RAM(Mb)», постоянная память «ROM(Gb)», частота «(MHz)».

Тема 2. Главные концепции о компьютерах.

2.1. Принцип работы компьютера и его структура.

Компьютер состоит из устройств и разных видов систем, все устройства сгруппированы и работают точно. Структура компьютера состоит:



Для того чтобы понять принцип работы компьютера проведём базовый анализ программы, веденной в память компьютера. Текст программы, написанный на любом языке, с помощью устройства входа (клавиатура, дискета) вводится в память компьютера, где он читается и проверяется логико-арифметической частью: потом результат выводится, как ответ, через монитор компьютера. Все эти операции находятся под контролем ИС.

Смотря со стороны базовых устройств, современный компьютер состоит из следующих компонентов:

- материнская плата - ей принадлежит функция проверки всей системы;
- монитор - предназначен для вывода текстовых и графических данных;
- клавиатура - позволяет вводить данные в компьютер;
- устройство дисководов;
- жесткий диск.

И кроме базовых компонентов к персональному компьютеру можно подключить:

- принтер – распечатывает информацию на бумаге;
- мышь - предназначена для ввода информации в компьютер;
- джойстик - предназначен для упражнений и для игр на компьютере;
- сканер - переносит изображение или текст с бумаги в память компьютера и другие.

## 2.2 Базовые устройства.

Материнская плата компьютера состоит из центрального микропроцессора, оперативной памяти RAM, контролеров и устройства ввода/вывода(I/O).

Микропроцессор указывает, сколько операций можно выполнить за одну секунду, но существуют слабые микропроцессоры, такие как INTEL8088, INTEL80286, INTEL80386 и для них не существует специальных команд, которые выполняли бы сложные операции, для этого рекомендуют использовать арифметические сопроцессоры, которые увеличивают скорость работы в 5-15раз. С появлением процессоров Pentium эта проблема исчезла, и скорости чувствительно увеличились.

Оперативная память служит для внутренних целей компьютера, из неё процессор берёт данные для обработки и в неё записывает полученные результаты. Содержание оперативной памяти стирается при выключении компьютера. Ёмкость измеряется в мегабайтах. В настоящее время наиболее распространенными объемами оперативной памяти являются 64М, 128М и 256М. Чем больше объем оперативной памяти, тем выше быстродействие компьютера. Собранные компьютеры снабжены оперативной памятью между 8Мб и 256Мб.

Контролеры представляют собой электронные устройства, которые проверяют деятельность периферийных устройств и т.д.

Порты ввода/вывода производят обмен информации между материнской платой и периферийными устройствами. Существуют два вида портов: параллельные (LPT1,LPT2,...) и синхронные (COM1,COM2,...). Параллельные производят операции I/O с большей скоростью, чем серийные порты, но они нуждаются в большем количестве каналов для обмена информации.

К портам входа-выхода можно подключать периферийные устройства - принтер, мышь, сканер, клавиатура, монитор и др.

Дискета - позволяет записывать и считывать информацию с дискеты. Дискеты - это носители информации, которые позволяют переносить информацию с одного компьютера на другой и записывать ее как

копию или архив. Ёмкость дискет измеряется в Мб или Кб. В наши дни самые распространённые дискеты с ёмкостью 1,44 Мб и диаметром 3,5 дюймов (89 mm). Ещё бывают дискеты с диаметром 5,25 дюймов (133mm) с ёмкостью 360Кб, 720Кб, 1,2 Мб. Прежде чем начать работу с дискетами их надо форматировать. Форматирование дискет выполняется с помощью команды *FORMAT* или через SO MS-DOS.

Жесткий диск позволяет сохранять всю информацию в компьютере - программы, текстовые документы, документы, записанные в графической форме. Жесткий диск зафиксирован внутри компьютера. Компьютер IBM PC/XT оснащен жестким диском с ёмкостью в десятки Mbytes, а жесткий диск компьютера IBM AT имеет ёмкость в сто Mbytes. Современные компьютеры снабжены с жесткими дисками с ёмкостью в Gbytes.

Принтер – предназначен для распечатывания информации. Все принтеры могут печатать информацию текстово и графически, черно-белые и цветные. Принтеры бывают разных типов:

- матричные;
- струйные;
- лазерные.

Матричные принтеры состоят из печатной головки множества тонких металлических иголок. Струйные принтеры - печатание документов производится из микроскопических капель, выводимых на бумагу из специальных дырочек. Качество печатания очень высокое. Цены на струйные принтера больше, чем на матричные. Лазерные принтеры имеют самое высокое качество печатания. Скорость печатания намного больше, чем у первых двух принтеров- 3-15 сек. один лист.

Монитор предназначен отображать информацию в удобном для него виде. Они могут работать в двух режимах – текстовом и графическом. При работе в текстовом режиме экран делится на две, так называемые, характерные зоны. Графический режим предназначен для отображения на экране рисунков, графиков, диаграмм и текстов. Экран состоит из точек, каждая из которых имеет свой цвет. Количество точек по вертикали и горизонтали характеризуют тип монитора. Пример: 640x480, 800x600, 1024x768.

Клавиатура – предназначена для введения информации в компьютер. Клавиатура состоит из разных типов клавиш:

- базовые (а, б, в, ...я, 0, 1, ..., 9, (, ., :, \*, @, \$, и др.);
- функциональные (F1-F10);
- вспомогательные (Shift, Alt, Ctrl, Enter).

Мышь – направлена к удобной работе на компьютере. Мышь бывает с 2 или 3 кнопками. Самые распространённые - с 2 клавишами. Щелчок

левой кнопкой означает выбор данной команды. Щелчок правой кнопки открывает дополнительное меню, у каждой программы оно разное.

## 2.3 Арифметические основы вычислительной техники.

### 2.3.1 Системы счисления.

В цифровых компьютерах информация любого вида представляется, хранится и обрабатывается в числовой форме. Числа представляются с помощью элементарных символов, называемых цифрами. Приведём несколько примеров систем счисления:

-десятичная система-это система счисления по основанию 10 и с количеством используемых цифр равным 10,соответственно 0,1,2,...,9;

-двоичная система - это система счисления по основанию 2 и с количеством используемых цифр равным 2,то есть 0 и 1;

Соответствующие цифры называются двоичными цифрами или битами;

-троичная система - это система счисления по основанию 3 и с количеством используемых цифр равным 3, соответственно 0,1,2;

-восьмеричная система - это система счисления по основанию 8 и содержащая 8 цифр: 0,1,2,...7;

- шестнадцатеричная система- это система счисления по основанию 16, которая содержит 16 цифр:0,1,...,9,A (десять), B, C, D, E, F(15).

Правило представления чисел в десятичной системе видно из следующего примера:

$(3856,43)_{10} = 3 * 10^3 + 8 * 10^2 + 5 * 10^1 + 6 * 10^0 + 4 * 10^{-1} + 3 * 10^{-2}$ . Заметим, что в этом представлении значение каждой цифры зависит от положения, которое она занимает в числе. Предположим, что число N имеет целую часть, состоящую из n+1 цифры, а дробную часть из m цифр:

$$N = C_n * C_{n-1} \dots C_1 * C_0 C_{-1} \dots C_m.$$

Значение этого числа вычисляется в зависимости от основания системы следующим образом:  $(N)_b = C_n b^n + C_{n-1} b^{n-1} + \dots + C_0 b^0 + C_{-1} b^{-1} + \dots + C_{-m} b^{-m}$ .

Путем соответствующих вычислений выполняются преобразование числа  $(N)_b$  по основанию b в десятичную систему счисления.

Например:

$$(110.11)_{10} = 1 * 10^2 + 1 * 10^1 + 0 * 10^0 + 1 * 10^{-1} + 1 * 10^{-2} = 110.15;$$

$$(110.11)_2 = 1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 1 * 2^{-1} + 1 * 2^{-2} = 6.75;$$

$$(110.01)_3 = 1 \cdot 2^2 + 1 \cdot 3^1 + 0 \cdot 3^0 + 1 \cdot 3^{-1} + 1 \cdot 3^{-2} = 12.444\dots;$$

$$(110.01)_8 = 1 \cdot 8^2 + 1 \cdot 8^1 + 0 \cdot 8^0 + 1 \cdot 8^{-1} + 1 \cdot 8^{-2} = 72,140625;$$

$$(110.01)_{16} = 1 \cdot 16^2 + 1 \cdot 16^1 + 0 \cdot 16^0 + 1 \cdot 16^{-1} + 1 \cdot 16^{-2} = 272,06640625;$$

Формально, десятичная система не имеет никаких особых преимуществ перед другими системами счисления. Предполагается, что это система была принята в древние времена благодаря тому, что в процессе счета в качестве инструмента использовались пальцы рук. Компьютер может работать в любой системе счисления. В ходе развития вычислительной техники было установлено, что наиболее удобна двоичная система. Предпочтение ей можно отдать по следующим причинам:

- простота правил для арифметических и логических операций;
- физическое представление цифр, с целью хранения и обработки чисел, осуществляется гораздо легче для двух символов, чем для десяти: перфорирован - не перфорирован, контакт замкнут - контакт разомкнут, наличие или отсутствие тока и т.д.;
- схемы, которые должны различать только два состояния, более надёжны в работе, чем схемы, которые должны различать десять состояний.

В процессе развития цивилизаций были созданы и непозиционные системы счисления. Ярким примером может служить римская система, использующая цифры I, V, X, L, C, D, M. Поскольку правила представления чисел и выполнения арифметических операций в таких системах очень сложны, то они имеют ограниченное применение.

### 2.3.2 Перевод чисел из одной системы в другую.

Перевод числа  $(N)_b$  в его десятичный эквивалент осуществляется в соответствии с формулой:

$$(N)_b = C_n b^n + C_{n-1} b^{n-1} + \dots + C_0 b^0 + C_{-1} b^{-1} + \dots + C_{-m} b^{-m}.$$

Перевод десятичного числа  $(N)_{10}$  в его эквивалент осуществляется по основанию  $b$  и осуществляется по следующим правилам:

- делением на соответствующее основание целой части и целочисленных частных после каждой операции деления до получения нулевого частного;
- результат перевода целой части составляется из целочисленных остатков, записанных в порядке, обратном их вычислению;
- умножение на основании дробной части, а затем и дробных частей, полученных в предшествующих умножениях, до тех пор, пока дробная часть очередного произведения не станет равной

- нулю или до получения требуемого количества цифр дробной части результата;
- результат преобразования дробной части состоит из целых частей произведений, записанных в порядке их вычисления.

Проанализируем несколько примеров.

1) Перевести десятичное число 37.0625 в его двоичный эквивалент.

$$37:2=18+1/2;$$

$$18:2=9+1/2;$$

$$9:2=4+1/2;$$

$$4:2=2+1/2;$$

$$2:2=1+0/2;$$

$$1:2=0+1/2;$$

Следовательно, целая часть двоичного числа будет (100101,0001);

$$0.0625*2=0.125; 0.125*2=0.250; 0.250*0.5; 0.5*2=1.0;$$

2) Перевести число 43.9 из десятичной системы в двоичную.

Целая часть:  $43:2=21+1/2; 21:2=10+1/2; 10:2=5+0/2; 5:2=2+1/2; 2:2=1+0/2; 1:2=0+1/2.$

Дробная часть:  $0.9*2=1.8; 0.8*2=1.6; 0.6*2=1.2; 0.2*2=0.4; 0.4*2=0.8; 0.8*2=1.6; \dots$

Заметим, что операция может быть продолжена до бесконечности, то есть не существует точного перевода анализируемого числа. Следовательно  $(43.9)_{10}=(101011,111001100\dots)_2.$

3) Осуществить перевод числа  $(1456,40625)_{10}$  из десятичной системы в восьмеричную.

$$1456:8=182+0/8; 182:2=22+6/8; 22:8=16+6/84; 16:8=2+0/8; 2:8=0+2/8; 0.40625*8=3.25; 0.25*8=2.0;$$

Следовательно:  $(1456,40625)_{10}=(20660,32)_8;$

4) Перевести число 3786.25 из десятичной системы в шестнадцатеричную:

$$3786:16=236+10/16; 236:16=14+12/16; 14:16=0+14/16; 0.25*16=4.$$

Следовательно:  $(3786,25)_{10}=(ECA,4)_{16}.$

2.3.3 Перевод чисел из двоичной системы счисления в восьмеричную, шестнадцатеричную и обратно.

Поскольку  $8=2^3$ , двоичный - восьмеричный и восьмерично - двоичный перевод может быть осуществлён напрямую. Любая восьмеричная цифра представляется тремя двоичными цифрами:

0=000;      4=100;  
 1=001;      5=101;  
 2=010;      6=110;  
 3=011;      7=111.

Если дано восьмеричное число, то для его перевода в двоичное каждая восьмеричная цифра заменяется тремя двоичными.

Примеры:

$$(352,451)_8=(011\ 101\ 010\ 100\ 101\ 001)_2;$$

$$(126.23)_8=(001\ 010\ 110\ 010\ 011)_2;$$

$$(5.065)_8=(101\ 000\ 110\ 101)_2;$$

Если рассматривать двоичное число, то для перевода его в восьмеричное группируем по три двоичные цифры, начиная с позиции запятой влево, для целой части, и вправо, соответственно, для дробной части, находя эквиваленты этих трех цифр в восьмеричной системе. При дополнении некоторой группы до трех двоичных цифр, добавление нулей в начале числа для целой части и, соответственно, в конце числа для дробной части не меняет значение числа.

Примеры:

$$(11,011101)_2=(011,011\ 101)_2=(3,35)_8;$$

$$(10,11011)_2=(010\ 001\ 110)_2=(2,66)_8;$$

$$(1001,01011)_2=(011\ 001,010\ 110)_2=(11,26)_8.$$

Аналогично поступаем и в случае шестнадцатеричной системы, основание которой  $16=2^4$ . Любая шестнадцатеричная цифра представляется четырьмя двоичными цифрами:

0=0000; 1=0001; 2=0010; 3=0011; 4=0100; 5=0101; 6=0110; 7=0111;  
 8=1000; 9=1001; A=1010; B=1011; C=1100; D=1101; E=1110; F=1111;

Примеры:  $(4B5F,B7)_{16} = (0100\ 1011\ 0101\ 1111,1011\ 0111)_2;$

$$(A51, 3 DE)_{16} = (1010\ 0101\ 0001, 0011\ 1101\ 1110)_2;$$

$$(1011, 100111)_2 = (1011, 1001\ 1100) = (B, 9C)_{16};$$

$$(10, 11011001)_2 = (0010, 1101\ 1001) = (2, D9)_{16};$$

### 2.3.4 Арифметические операции в двоичной системе счисления.

Арифметические операции над двоичными числами очень просты. Правила действий в двоичной системе представлены в таблицах.

Двоичное сложение

Двоичное вычитание

Двоичное умножение

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10

0 - 0 = 0
1 - 0 = 1
1 - 1 = 0
10 - 1 = 1

0 * 0 = 0
0 * 1 = 0
1 * 0 = 0
1 * 1 = 1

Примеры:

1.Выполняем сложения десятичных чисел 29 и 43 в двоичной системе исчисления:

$$(29)_{10}=(11101)_2;$$

$$(43)_{10}=(101011)_2$$

$$\begin{array}{r} 11101 \\ + \\ 101011 \\ \hline 1001000 \end{array}$$

проверка:  $(1001000)_2=(72)_{10}$ , результат верен, поскольку  $(29)_{10}+(43)_{10}=(72)_{10}$ .

2.Выполняем вычитание десятичного числа 37 из десятичного числа 46 в двоичной системе счисления:

$$(37)_{10}=(100101)_2;$$

$$(46)_{10}=(101110)_2;$$

$$\begin{array}{r} 101110 \\ - \\ 100101 \\ \hline 1001 \end{array}$$

проверка:  $(1001)_2=(9)_{10}$ , результат верен, поскольку  $(46)_{10}-(37)_{10}=(9)_{10}$ .

3.Выполняем умножение десятичных чисел 3.25 и 7.125 в двоичной системе счисления:

Пример:

$$(3.25)_{10}=(11,01)_2;$$

$$(7.125)_{10}=(111,001)_2;$$

$$\begin{array}{r} 11,01_x \\ 111,001 \\ \hline 1101 \\ 0000 \\ 000 \\ 1101 \\ 1101 \\ 1101 \end{array}$$

10111,00101

проверка:  $(10111,00101)_2 = (23,15625)_{10}$ , результат верен, поскольку  $(3,25)_{10} * (7,125)_{10} = (23,15625)_{10}$ .

4. Выполняем деление десятичного числа 211 на десятичное число 3 в двоичной системе счисления:

$$(211)_{10} = (11010011)_2;$$

$$(3)_{10} = (11)_2;$$

11010011

11

00100

11

11

11

01

11

1000110

Следовательно,  $11010011 : 11 = 1000110$ , остаток 1.

Проверка:  $(1000110)_2 = (70)_{10}$ , результат верен, поскольку  $(211)_{10} : (3)_{10} = (70)_{10} + (1)_{10}$ .

Заметим что, как при умножении, так и при делении, установка запятой, отделяющей целую часть от дробной, осуществляется, так же как и в десятичной системе исчисления.

Тема 3. Метод решения задач с помощью компьютера.

3.1. Этапы решения задач с помощью компьютера.

Любая задача предназначенная решению с помощью компьютера, в процессе решения должна пройти 6 этапов:

1. Выбор математического модуля
2. Создания алгоритма
3. Проверка алгоритма
4. Алгоритм решения
5. Проверка программы
6. Документация программ.

1. Выбор математического модуля. Каждая задача, в независимости от своей сложности или предмета, может быть математически сформулирована перед решением. Данная формулировка представляет собой математическую копию задач. Задача по своей сути может быть сложной или простой. При выборе математического модуля воздействуют следующие факторы:

- уровень знаний программиста.
- удобный метод указания команд.

- простые команды.
- возможности вычислительной техники.

После выбора математического модуля, решаемая задача выбирает нужные математические термины.

2.Создания алгоритма. Любой алгоритм состоит из 3 базовых свойств:

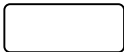

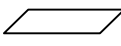

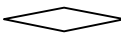





-Общее (то есть решает задачу в общем, виде).

-Реализация состоит в том, чтобы все выразить правильно.

-Финал.

Алгоритм имеет данные входа, после которых получаются конечные результаты для выхода. Алгоритм решения программы состоит из нескольких вариантов. Самые распространенные это - блок-схемы и с помощью кодового языка.

Представление алгоритма с помощью блок-схемы:

1. START/STOP.Указывает начало/конец схемы. 
2. Блок присвоения. 
3. Блок входа. 
4. Блок выхода. 
5. Блок условия. 
- 6 . Блок процедура 
7. Блок цикла FOR 
8. Узел 
- 9.Элемент перехода на другой лист. 
- 10 .Стрелка соединения между блоками. 

Между элементами логической схемы счисления существует группы, показывающие более сложные процессы.

3. Проверка алгоритма. Это один из сложных этапов. Представляет собой контроль хода решения, проверку алгоритма по разным методам решения. Алгоритм описывается по пронумерованным этапам (0...n) и проводится анализ каждого шага.

4.Реализация алгоритма. Представляет собой углублённый анализ алгоритма, выбор программного языка и перевод на нужный язык.

5.Проверка программы. Для проверки программы существуют используемые методы:

- а) аналитический метод;

в) конструктивный метод;

с) тестированный метод и др.

6. Документация программы. Преследует метод описания алгоритма программы, всех используемых данных, блок-схем, полученных результатов и данных для работы с программой.

## Тема 4: Язык программирования Паскаль

### 4.1 Введение

#### 4.1.1 Алфавит, словарь и синтаксис языка.

Язык программирования - это язык, который диктует компьютеру метод решения задач. Метод решения задач называется алгоритмом. Целая теория информатики занимается созданием новых компьютеров, языков программирования и алгоритмов. Благодаря любому языку высшего уровня, которые созданы для решения любой задачи, язык программирования Паскаль, как и другой язык программирования, имеет свой алфавит и специфику использования символов. Алфавитом одного из языков программирования называется группа символов, позволяющая использовать и узнаваемая компьютером, с чьей помощью можно создать размеры, выражения, и операторы этого языка программирования. Алфавит любого языка программирования содержит самые простые элементы литературного смысла, которые с элементами словаря получают правильные фразы.

Словарные элементы состоят из символов. Любой символ представлен в компьютере, в одном экземпляре, и расположен среди номеров от 0 до 127 и названой кодом ASCII.

Алфавит языка программирования Паскаль состоит:

1. Символы, использующиеся для создания идентификаторов:

- маленькие и большие буквы латинского алфавита,

- арабские цифры от 0 до 9,

- символ подчеркивания (код ASCII: 95).

2. Символы раздела:

- символ « пробел » (код ASCII: 32). Функция разделения ключевых слов и номеров.

- Управляемые символы (ASCII: 0...31), могут использоваться для описания констант. Табуляция символов (ASCII: 9) и символ перехода на другую строку.

a:= b + c -D тоже самое  $\left\{ \begin{array}{l} a:= b \\ +c - D \end{array} \right.$

3. Специальные символы состоят из одного или нескольких символов.

+ \* - / = [ ] { } ( ) < > ? # % @ . : ; ' ^

4. Сложные символы - группа символов, понимаемые компьютером как одно целое.

<= => := ( . .) (\* \*) .. ,

5. Ключевые слова являются зарезервированными и не могут использоваться с целью, отличной от той, которая предназначена им по определению языка.

Лексические единицы, рассматриваемые в одном параграфе, могут быть определены следующими формулами БНФ:

- Специальный символ: = + I - I \* I / I = I < I > I # I

[ I { I } I : I ; I ...I

- Ключевое слово and, array, begin, case, const, div, mod, down to, else, end, file, for, function, go to, if, in, label, mod, nil, not, of, or, packet, procedure, program, record, repeat, set, then, to, type, until, VAR, while, with.

Символы {, }, [ \* ], используемые в формулах БНФ, являются одновременно и элементами языка ПАСКАЛЬ. Чтобы избежать двусмысленности, данные символы, как элементы словаря, могут быть представлены через эквивалентные символы (\*, \*) и соответственно (., .).

Идентификаторы – это лексические единицы, которые выступают в качестве имён переменных, констант, функций, программ и т.д.

Любой идентификатор начинается с буквы, за которой может следовать любая комбинация из букв и цифр. Длина идентификаторов не ограничена, но только первые 63 символа являются значимыми.

Строка представляет собой последовательность печатных символов, заключенную в одиночные кавычки. Если в состав строки нужно включить сам символ одиночной кавычки, то этот символ печатается два раза. Отметим, что в строках строчные и прописные буквы являются различными символами.

В отличие от других лексических единиц языка Паскаль, в строках могут использоваться и буквы русского алфавита. Для этого необходимо, чтобы на компьютере была установлена программа-драйвер, обеспечивающая ввод, вывод и печать таких букв.

#### 4.1.2 Общая структура одной программы.

Программы, написанные в языке программирования BORLAND PASCAL 7.0, состоят в соответствии с несколькими правилами расширенной версии стандартного языка PASCAL. Общая структура одной программы можно разделить на несколько основных частей:

- штамп программы,
- декларативная часть,
- процедуры и функции,
- главный блок,

которые показаны в программе следующим образом:

##### 1. Стандарт

Имя программы;

##### 2. Декларативная часть.

{ \$... } глобальная директива компилятора.

USES- библиотеки.

LABEL- объявление глобальных частей

CONST- объявление глобальных констант

TYPE- декларирование глобальных типов

VAR- объявление глобальных переменных

##### 3. Процедуры и функции.

Процедура (функция)- стандарт процедура (функция)

LABEL- объявление локальных частей

CONST- объявление локальных констант

TYPE- объявление локальных типов

VAR- декларирование локальных переменных

BEGIN- главный блок процедуры (функции)

END;

##### 4. Часть главного блока

BEGIN- главный блок программы

END.

## 4.2 Концепция данных

Информация, подлежащая обработке, представлена в компьютере в виде данных. Данные состоят из цифр, букв, знаков, чисел и др.

В машинном коде компьютера данные представляются как последовательности двоичных цифр. В программах на языке Паскаль данные представляются в виде величин: переменных и констант. Для того, чтобы избавить пользователя от деталей внутреннего

представления данных. Под типом данных понимается множества значений и множества операций, которые можно к ним применять.

#### 4.2.1 Простые типы данных

Существуют 3 типа простых данных: перечисляемые, интервальные и порядковые.

##### 4.2.1.1 Предопределенные данные

Существуют 5 простых предопределенных типов данных: INTEGER, REAL, BOOLEAN, CHAR, TEXT.

Тип данных INTEGER.

Множество значений типа данных Integer состоит из элементов подмножества целых чисел определяемого при реализации. Максимальные значения определяются константой MaxInt, известной любой программе на языке Паскаль. Обычно максимальным значениям в изучаемом типе данных является – MaxInt или (MaxInt+1).

Очевидно, что результаты операций '+', '-', '\*', над числами также должны принадлежать множеству значений типа Integer.

Тип данных REAL – состоит из элементов подмножества вещественных чисел, определяемого при реализации. Операции, которые можно применять к значениям вещественного типа - '+', '-', '\*', '/' и др. Операции над вещественными числами в общем случае являются приближенными из-за погрешностей округления. Естественно, результаты данных операций также должны принадлежать множеству значений типа Real.

Тип данных Boolean.

Значения типа данных Boolean являются истинностные значения false и true. Логические операции, которые можно применять к данным логического типа And- конъюнкция, Or- дизъюнкция, Not- отрицание.

Конъюнкция- And

X	Y	X&Y
1	1	1
1	0	0
0	1	0

Дизъюнкция- Or

X	Y	XvY
1	1	1
1	0	1
0	1	1

Отрицание- Not

X	X
1	0
0	1

## Тип данных Char

Множества значений данного типа являются конечными упорядоченными данного множества символами. Значения рассматриваемого типа обозначаются символами и заключены в одиночные кавычки. Как правило, символы конкретной версии языка Паскаль упорядочены согласно таблице кодов ASCII. Тип данных Char используется для создания более сложных структур данных, в частности строк символов.

### 4.2.1.2 Перечисляемые типы данных.

Перечисляемый тип содержит множество значений, определяемых посредством идентификаторов.

Пример: `type` месяц = (Январь, Февраль, Март, Апрель, Май, Июнь, Июль, Август, Сентябрь, Октябрь, Ноябрь, Декабрь);

`Var` a : месяц ; B: real;

### 4.2.1.3 Интервальные типы данных

Интервальный тип данных включает подмножество значений уже известного типа данных, называемого базовым. Имя интервального типа данных наименьшее и наибольшее значение указываются в разделе описаний программ после ключевого слова `type`.

Пример:

```
1) type индекс = 1..10;  
   буквы = 'А'...'Я';  
   цифры = '0'...'9';  
   дни = (п,вт,ср,чт,пт,сб,вс);  
   дни рабочие = п...пт;  
var  
и: индекс;  
б: буква;  
д: дни рабочие;
```

Множество значений типа «индекс» является подмножеством значений предопределенного типа `integer`. Множества значений типов «литера» и «цифра» являются подмножествами предопределенного типа `char`.

Множество значений типов Рабочие Дни и Дни Отдыха являются подмножествами перечисляемого типа Дни определенного пользователем.

Переменные интервального типа объявляются с помощью ключевого слова VAR, переменная интервального типа обладает всеми свойствами переменных базового типа, но ее значения должны находиться в соответствующем диапазоне. В противном случае возникает ошибка и программа останавливается. Использование интервальных типов данных улучшает наглядность программ и упрощает их проверку. Следуют подчеркнуть, что в языке Паскаль нельзя определять интервальные типы на базе типа real, так как его значения не имеют порядковых номеров.

Пример:

```
Program tekst;  
Type цифра = 0...9;  
Var c1, c2, c3: цифра;  
Begin  
{R+} c1 := 5;  
c2 := c1+3;  
{R-} c3 := 25;  
{R+} c3 := 30;  
end.
```

#### 4.2.2 Структурированный тип данных в Паскаль

До данного момента компьютеры работали с простыми типами данных (в особенности скалярными). Каждое значение любого типа данных известна и состоит из одного компонента. В случае структурированных типов данных каждое значение представляет собой структуру с невыявленным значением, только если это значение имеет больше чем один компонент. В тоже время, любое конкретное число представляет собой структуру данных. Структурированный тип данных представляет структуру, состоящую из нескольких элементов такого же типа, в простом случае каждый простого типа.

##### 4.2.2.1 Тип данных массив(Array)

Множество массивов типа данных array состоит из массивов (таблиц). Массивы составляются из фиксированного числа компонент одного и того же типа, которые называется базовым. Ссылка на компоненты осуществляется с помощью индексов.

В случае компонентов типа real имеем вектор, т.е. имеем массив. В случае компонентов типа char массив, может быть выполнен как один

ряд текста. Обычно каждому массиву использованному в программе должно присваиваться имя. Это имя назовем переменной, потому что ее значение является целым массивом. Каждый компонент массива должен быть оценен через показания названного массива, после которого следует искатель компонента: индекс взятый []. Значит для ссылки к одному элементу массива используется синтаксис <имя массив> [<индекс>];

Пример: x[i] называется частичной переменной, потому что ее значение часть компонента массива. В общем случае индекс массива можно использовать как выражение, значение которого определяет число компонентов массива. В то же время в рамке программы это выражение может получать разные параметры. В таком случае одна и та же переменная с индексом может указывать о разных компонентах массива.

Заметка: Тип индекса или выражение отображающее индекс должно быть обязательно порядковым. Но самую большую возможность для обработки массива предлагает тип индекс: подобласть целого типа. Декларированный синтаксис массива:

Var: <имя>: array [C1..C2] of <тип>.

где <имя> - имя массива; C1,C2-ограничение величины массива,<тип> - тип компонента массива.

Пример: var: x: array [1...4] of real;

В тоже время компоненты массива могут быть любого типа, индекс массива должен быть подобластью целого типа. Существование массива x из 4 элементов. X[1]-указывает первые элементы массива. X[2]- вторые. X[i]- элемент с индексом i. Именно этот индекс использован в цикле For для обработки массива: For i: =1 to N do;

В Паскале не существует ограничений в отношении типов элемента массива. Необходимо только что бы все элементы были одного и такого же типа. И если они составлены из скалярных элементов, то эти массивы - матрица. Если к следующему этапу элементы массива являются тем же самым массивом, тогда этот массив двоичный.

Синтаксис: ARRAY [<tip index >] OF <tip element>;

Потому что каждый элемент массива является тем же самым массивом, имеем ARRAY [<tip index>] OF ARRAY [<tip index>] OF <tip scalar>; этот синтаксис приобретает форму: ARRAY [C1...C2, C3...C4] OF < tip scalar>; где C1, C2, C3, C4-размер массива.

Пример: X: ARRAY [1...10, 1...20] OF real;

i        j

#### 4.2.2.2 Тип строка символов

Для переработки комплектующего шрифта, слов, предложений в Турбо Паскале используется тип данных названный STRING. Результат переменной этого типа состоит из определенного числа символов.

Из-за этого переменная этого типа имеет, как результат несколько элементов простого типа (тип char). Тип string-сложный тип.

Тип строка символов имеет базу string:

```
Var a: string [длина]; b: string;
```

В первую очередь 'длина' обозначает максимальную длину строки символов, от 0 до 255. Длина строки в некоторых случаях неизвестна.

Она может быть изменена на протяжении выполнения программы или выполнение программы выполняется без объявления длины. Для определения длины какой-то строки в Паскале используется функция Length.

```
Пример: Var a1: string [10]; a2: string; n1, n2: integer;
```

```
Begin a1:= 'Programare'; a2:= 'Radioelektronika';
```

```
n1:= Length (a1); {n1=10} n2:= Length (a2); {n2=16}
```

```
Writeln ('длина a1:' n1); Writeln ('длина a2:' n2); end.
```

Как видим, значение функции Length является целый тип (integer), которую содержит переменная типа string. Это значение может быть модифицирована программистом и может иметь форму: строка[0]:= #nr или строка[0]:= chr(ord(nr)); Если номер имеет значение 0, то строка является пустой. Строка может быть использована полностью, или наполовину. Строка должна быть сформирована так, чтобы выражение было в интервале от 0 до объявленной длины строки.

```
Пример: Var a: string [15];
```

```
Begin a:= 'студент'; n: integer;
```

```
Writeln ('первый символ:' a [1]);
```

```
n:= Length (a); Writeln ('последний символ:' a[n]);
```

```
Writeln ('средний символ:' a [n-(ndiv2)] ;
```

Над строкой символов может выполняться конкретная операция, отмеченная +.

Операторы отношения =, <>, <, >, =, >=, и <= сравнивает строки символов, в соответствии с таблицей ASCII.

#### 4.2.2.3 Тип данных множества (SET)

Тип данных множество (set) определяется по отношению к базовому типу, который должен быть порядковым:

```
<Тип множество>:: [packed] set of <Тип>
```

Значениями типа данных множество являются множества, состоящие из значений базового типа. Если базовый тип имеет n значений, то тип множество будет иметь  $2^n$  значений. Значение n ограничено:  $n \leq 256$ .

В языке Паскаль элементы множества могут перечисляться в квадратных скобках [и], которые являются аналогом фигурных скобок в математике.

Запись [ ] означает пустое множество.

Type Indice = 1..10;

Дни = (По, Вт, Ср, Чет, Пят, Сб, Вс);

Множество Индексов = set of Индекс;

День Присутствия =set of День;

Var МИ: Множество Индексов;

ДП: День Присутствия;

Порядковый тип Индекс имеет n=10 значений: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

Тип MulteIndcii имеет  $2^{10}=1024$  значений, а именно:

[ ], [1], [2], ..., [1, 2], [1, 3], ..., [1, 2, 3, 4, 5, 6, 7, 8, 9, 10].

Таким образом, переменная MI может принимать любое из этих значений, например:

MI := [1, 3].

Порядковый тип ZI имеет n=7 значений Пн, Вт, Ср, Чт, Пт, Сб, Вс. Тип

День Присутствия имеет  $2^7=128$  значений, а именно [ ], [Пн], [Вт],

[Ср],... [Пн, Вт], [По, Ср],..., [ Пн, Вт, Ср, Чт, Пт, Сб, Вс]. Значения

типа множества может определяться через конструктор множества.

Значения данных выражений представляют собой верхний и нижний

пределы интервала. К значениям типа данных множества можно

применять обычные операций:

+ объединения;

-разность;

\*пересечения,

результат, которых относится к типу множества, и операций отношения:

= равенство;

< > неравенство;

<= , => включения;

: принадлежность,

результаты, которых относится к типу boolean.

#### 4.2.2.4 Тип артикль.

До этого момента были изучены сложные типы данных. В случае одного массива, все элементы массива были одного и того же типа(integer, real, char, и т.д.) Но часто возникают ситуации, когда нужно, что бы переработка и хранение информации были сложными: информация о личности, об успеваемости студента, расписание уроков и т.д. Но типы данных различаются между собой: имя и группа будут типа string, оценка о успеваемости- типа integer, а средний балл- типа

real. Впоследствии мы не можем группировать компоненты, как таблицу, которая изображает однородную структуру. Это группирование допускает структуре регистрироваться в Паскале и названному типу артикль.

Тип артикль является сложным типом, образованным из компонентов, названных полями. Число компонентов бывает точным или переменным. В первом случае имеем точную структуру, во втором - структуру с вариантами.

В отличие от таблицы, поля могут быть сложными типами. Каждое поле имеет свое имя (идентификатор полей) и один тип (название типа поля), подобно, как и структура, артикль имеет свое имя.

Основная форма с точным артиклем:

```
имя_ артикль = рекорд;  
имя_ поле _1: тип_ поле_2;  
имя_ поле_2: тип_ поле_2;  
... ..  
имя_ поле_ n: тип_ поле_n;  
End.
```

Пример:

```
Тип data = рекорд  
год: 1900...2000;  
месяц: (январь, февраль, апрель, март, июнь, июль, август, сентябрь,  
октябрь, ноябрь, декабрь);  
день: 1...31; end; var сегодня: data;
```

Основная форма с переменным артиклем:

```
Тип встреча = рекорд  
Когда: data;  
Час: real;  
Где: string[20];  
End;
```

Var a: встреча;

Название поля должно быть только в одном единственном артикле, тот который был определен. Значит, если в программе имеем объявленные артикли, тогда можно имена полей из артиклей повторять, при условии, что имя артикля было единственным.

Пример:

```
Тип студент 1 = рекорд  
Имя: string; Группа: string; Оценка 1: integer; Оценка 2: integer; Средняя:  
real; end;  
Студент 2 = рекорд
```

Имя: string; Группа: string; Оценка 1: integer; Оценка 2: integer; Оценка 3: integer; Средняя: real;

End;

Для объявления редактирования программ, объявленные поля того же типа одного артикля выполняются простые переменные того же типа-они разделяются запятой.

Пример:

Type студент = рекорд

Имя, группа: string; оценка 1, оценка 2, оценка 3: integer; средняя: real;End.

Переменная величина типа артикль может принимать участие в разных операциях, решениях, формулах. Но эти величины не являются величинами всего типа, а конкретными величинами полей артикля. В случае одного артикульного поля через названные переменные и поля, разделено точкой. В таких случаях используют функцию with. Она разрешает короткую адресованность полей одного артикля.

Например:

Type calendar = record

An: 1900...2000; Luna: 1...12; ziua: 1...31; end.

Var data: calendar;

Begin with data do

If luna = 12 then begin

Luna: =1; an: = an + 1;

End.

Else luna = luna + 1;

End.

Эта функция эквивалентна следующей:

If data. Luna = 12 then begin

Data. Luna: = 1;

Data.an: = data. An +1; end

Else data. Luna: = data. Luna +1;

Внутри инструкции with, одно наименование переменной, в первый раз ставится на место как названное поле одной переменной.

Например:

Type punkt = record

x, y: integer; end;

Var x: punkt; y: integer;

В нашем случае как x, так и y могут изображать одну переменную, так как в инструкции with x do begin x: = 1; y: = 2; end; то опознаватель x установлен между with и do адресован к переменной типа integer, то

есть в поле артикля. В случае необходимости with с множеством переменных тип артикль поступает так:

With v1, v2... Vn do инструкция;

Эта инструкция равна следующей:

With v1 do with v2 do { ... } with Vn do инструкция;

#### 4.2.2.5 Тип reper

Важнейшей проблемой сегодня в информатики является проблема памяти. В процессе развития компьютерная техника даёт потребителю большие возможности. Вместе с развитием компьютерной техники, развивается с гигантскими шагами и её Soft обеспечение. Артикль может принимать

Таким образом, появляются новые разработки с наибольшими потребностями к резервам компьютера. Чаще всего проблема использования этих разработок состоит в недостаточной оперативной памяти RAM. Такие ситуации могут появиться и в программах Паскаль, когда разрабатывается большое количество данных, и программы становятся большими и сложными, требуя больше памяти RAM. Для таких ситуаций Паскаль имеет свой инструмент *variabile dinamice*. В Турбо Паскаль переменные могут быть динамические или статистические.

Статистические переменные находятся в памяти во время компиляции, а их место в памяти не может быть изменено во время выполнения; они сохраняются на протяжении всего исполнения блока (программы процедура функция). Но в Паскале есть переменные, которые могут быть сформированы и уничтожены динамически во время работы программы. Такая переменная называется динамическая переменная.

Создание и уничтожение динамической переменной протекают с помощью процедуры *New* и *GetMem* соответственно *Dispose* и *FreeMem*. Эти процедуры дают и соответственно освобождают место в памяти для динамических переменных. Адрес зоны памяти, для динамических переменных будет находиться в переменных специального типа называемой *Reper*. Длина зоны памяти данной переменной зависит от типа динамической переменной. В зависимости от типа переменной даётся разное количество ячеек для динамических переменных.

Например, если тип динамической переменной `integer` резервируются 2 ячейки, `real` резервируются 6 ячеек.

В результате переменной типа `gereg`, которая будет иметь адрес зоны данной переменной, которая должна передаться в память локационным процессом типа динамической переменной.

Надо отметить, что динамическая переменная находится в специальной зоне памяти, называемой „heap”. Можно устанавливать тип `gereg` в секции `type` в следующем порядке:

```
Type имя _ gereg =^tip_ динамическая переменная;
```

Где `^` обозначает адрес. Множество динамических переменных типа, имя `gereg`” является неограниченным количеством адресов, каждый адрес устанавливает “tip\_динамическая \_переменная”. Сюда добавляется специальная величина, которая называется `nil` и которая не устанавливает ни одного типа.

```
Type rep =^art;
```

```
art= record x, y : integer; end;
```

```
var r1,r2:rep;
```

```
r3:^integer;
```

```
r4: ^char;
```

Другой плюс языка является в том, что можно использовать типы, которые сами определяют себя:

```
Type lista =^ articol ; articol=record
```

```
a , b: integer ; urmator: lista; end;
```

```
var : l : lista ;
```

После создания динамических переменных, адрес, которых ставится в переменных типа `gereg`, она может быть доступна так называемым: после динамических переменных типа `gereg` находится знак `^`. Этот знак также может находиться и после другого кодификатора.

Пример простых операций с динамическими переменными:

```
Type pc =^char ; pintrg=^integer ; pcmp=^art;
```

```
Art=record x: integer; y: arraya[1..2] of integer; end;
```

```
Var c: pc; intrg:pintrg ; cmp:pcmp;
```

```
Begin new(c); {crearea unei variabile dinamice tip character }
```

```
c^ :='*'; {incarcarea variabilei create }
```

```
new(intrg); {crearea unei variabile dinamice }
```

```
intrg^:123; {de tip intreg si incarcarea ei }
```

```
new(cmp); {crearea unei variabile dinamice }
```

```
cmp^.x:=4; {de tip articol si incarcarea }
```

```
cmp^.y[1]:=5; {cimpurilor variabilei }
```

```
cmp^.y[2]:=6; {... }
```

```
writeln(c^); writeln(intr^); write(cmp^.x); write(' ', cmp^.y[1]);
```

```
writeln (' ',cmp^.y[2]); {varibilele dinamice nu mai sint necesare}
dispose (cmp);           {distrugerea variabilelor dinamice}
dispose(intrg);
dispose (c);             {se poate reutiliza spatiul de memorie ocupat
anterior}
{...} end.
```

#### 4.2.2.6 Тип pointer

Множество значений предопределенного типа pointer состоит из адресов и специального значения nil. По соглашению тип данных pointer совместим с любым ссылочным типом. Переменная типа pointer вводится описанием вида:

Var p: pointer; Так как любое описание не содержит информацию о базовым типе, тип динамической переменной p<sup>^</sup> неизвестен. Следовательно, на переменные типа pointer нельзя ссылаться с помощью символа <sup>^</sup>, наличие которого после такой переменной является ошибкой.

Пример:

Program test;

Type e = integer ; pe=<sup>^</sup>e; pin=<sup>^</sup>integer;

Var alfa: e; beta : integer; p,p1:pointer; vpe: pe; vpi: pin;

Begin writeln('Тест с типом pointer') ;

Alfa:=5; beta:=6;

Vpe:=@alfa; vpi=@ beta ;

Writeln('alfa= ',p2<sup>^</sup>); {ошибка 64: Cannot Read or Write variables of this type}

{(Не могут быть прочитаны или написаны переменные данного типа)}

{alfa 1<sup>^</sup>:=p1<sup>^</sup>};

p:=@beta ; vpi:=p; Vpe:=p1; writeln('alfa = ', vpe<sup>^</sup>, 'beta=', vpi<sup>^</sup>);

readln; end.

Пример:

alfa=5 beta=6

Alfa=5 beta=6

## 4.2.2.7 Файлы в Паскале

### 4.2.2.7.1 Свойства файлов

Под файлом понимают структуру данных, которая состоит из последовательности компонент. Все компоненты файла относятся к одному и тому же типу, который называется базовым. Число компонентов файлов является произвольным, однако конец файла обозначается специальным символом EOF. Файл, который не содержит ни одного элемента, называется пустым файлом. Отметим что символ EOF, который означает конец файла, не является компонентом файла. Переменные FN, FC, FE файлового типа называется файлами языка Паскаль или просто файлами. В отличие от остальных типов данных, значения, которых хранятся во внутренней памяти компьютера, данные файлов, хранятся на периферийных устройствах - носителях информации. В Турбо Паскали связь файловой переменной f с внешним файлом осуществляется вызовом процедуры

```
assign (f , s)
```

где s- это выражения типа string, задающая имя внешнего файла.

Примеры:

```
assign (FN, ' A:\результат \ R. DAT') -
```

файл FN связывает с внешним файлом R.DAT, находящимся в каталоге Rezultat на диске A.;

```
assign (FC, ' C:\ A.CHR' )-
```

файл FC связывает с внешним файлом A.CHR, находящимся в корневом каталоге диска C.;

```
write('Введите имя файла :');
```

```
readln (str);
```

```
assign (FE, str)-
```

файл FE связывается с внешним файлом, имя которого считывается с клавиатуры в переменную str типа string.

После выполнения оператора assign (f, s) все операций, осуществляемые над файлом f, фактически будут выполняться над внешним файлом s.

Самыми распространенными операциями, выполняемые над файлами, являются считывания компонентов из файла и их запись в файл.

Считывание текущей компоненты из файла осуществляется с помощью оператора вызова процедуры

`read (f, v)`

где *v* – переменная, которая относится к базовому типу файла *f*

Запись следующей компоненты в файл осуществляется с помощью оператора вызова процедуры:

`Write (f, e),`

Где *e*- выражение, относящееся к базовому типу файла *f*.

По типам операций, применяемых компонентов, файлы подразделяются:

- входные
- выходные
- рабочие

По методу доступа к компонентам файлы подразделяются:

- файлы последовательного доступа или последовательные
- файлы прямого доступа

Отметим, что в стандартном языке допустимы только входные и выходные файлы последовательного доступа.

Тип файла и метод доступа задаются при открытии файла. В случае стандартом языке существуют следующие процедуры для открытия файлов:

`Reset (f)`- открывает существующий файл для чтения;

`Rewrite (f)`- создает пустой файл для записи.

После завершения обработки компонент файл нужно закрыть. При закрытии файла операционная система записывает, если необходимо, символ EOF; регистрируя только что созданный файл в соответствующем каталоге и т.д.

#### 4.2.2.7.2 Файл с типом

Файлы с типом содержит множество компонентов, каждый компонент имеет тот же тип называемый базовый тип файла. Переменная данного типа декларируется через следующую форму:

`Var имя файла: of базовый тип`, где базовый тип взят произвольно, исключая типовой файл .

Пример:

`Program файл с типом;`

`type устроенный = record`

`марка: integer; имя:string[20]; зарплата: integer; end;`

`var f:файл of устроенный;`

```

a: устроенный;   продолжение: char; k: integer;
Begin
assign(f,'персона.txt');
rewrite(f);
repeat  writeln('марка=');readln(a.марка);
writeln('имя=');readln(a.имя);
writeln('зарплата=');readln(a.зарплата);
write(f,a);
writeln('Продолжаем? d/n');readln(продолжение);
until upcase(продолжение)='N';
k:=filesize(f)
writeln('Число компонентов в файле =' ,k);
close(f);
End.

```

Пример 2:

```

Program основной доступ;
type  устроенный = rekord
марка: integer; имя:string [20]; salar : integer ;end;
var f:файл of устроенный ; a: устроенный; fs: integer; nr: integer;
begin
assign(f,'персона.txt')
reset(f);
fs: =filesize(f);
writeln('nr.компонента ='); readln(nr);
if nr >fs then writeln('ошибка, наименьший файл')else
begin seek(f,nr);
read(f,a);
writeln('марка =' ,a.марка);  writeln('имя = ' ,a.имя);
Writeln ('зарплата = ' , A.зарплата);
end; close(f);
end.

```

#### 4.2.2.7.3 Текстовый файл.

Известно, что данные файлов хранятся на внешних носителях информации. В случае, когда файлы описываются в виде file of T, элементы типа T записываются на соответствующих носителях в виде последовательности двоичных цифр. Такой способ представления данных удобен для внешней памяти. Для устройства ввода \ вывода соответствующие данные должны быть представлены во внешней форме, то есть через строки символов.

Для того чтобы упростить общения пользователя и компьютера, в языке Паскаль информация, предназначенная для пользователя, представляется в виде текстовых файлов. Текстовый файл состоит из последовательности символов, разделенных на строки. Длина строк произвольна. Конец каждой строки обозначается специальным символом EOL. Так как длина строк произвольна, то позицию некоторой строки в файле нельзя знать заранее.

Обработка текстовых файлов может осуществляться с помощью известных процедур, применимых к любым типам файлов: assign, reset, rewrite, read, write, close. В дополнение к ним в языке есть специальные процедуры для обработки элементов EOL.

Для ввода и вывода данных используется, как правило, предопределенные текстовые файлы

Input и Output. Файл Input предназначен только для чтения и связан со стандартным устройством ввода операционной системы. Файл Output предназначен только для записи и связан со стандартным устройством вывода. Эти файлы открываются, и закрываются автоматически в начале и соответственно в конце выполнения программы.

```
Program crtext;
```

```
Var c :char ; f: text;
```

```
begin
```

```
assign (f, 'книга.txt); {сравнение с переменной f с внешним файлом  
книга.txt}
```

```
rewrite (f);           {открытие нового файла для печатание }
```

```
while not e of do      {контроль последнего файла: eof(Input)}
```

```
begin while not eoln do {контроль последней линий: eoln (Output)}
```

```
begin read(c);         {чтения с клавиатуры: read (Input(c)}
```

```
write(f,c);           {переменная c будет написана в файле} end;
```

```
readln;               {readln(Input)}
```

```
writeln(f)            {напиши EOLN в f} end;
```

```
Close (f)             {напиши EOF в f и закрой} end.
```

#### 4.2.2.7.4 Переменные файлы

В Турбо Паскаль с переменными типа file of T можно проводить как операции чтения, так как и записи, независимо от того, были ли они открыты с помощью процедуры rewrite или reset. Для этого каждый элемент файла имеет свой номер, называемый номером элемента. Первому элементу соответствует номер ноль, второму один и т.д. Количество элементов файла f можно определить с помощью стандартной функции.

## FileSize(f)

Обычно доступ к элементам файла является последовательным. Это означает, что после считывания /записи некоторого элемента указатель текущей позиции в файле перемещается на элемент со следующим порядковым номером. Прямой доступ основывается на процедуре поиска Seek(f, i), которая перемещает указатель текущей позиции в файле на элемент с номером i. После чего элемент, выбранный таким образом, можно считывать/записывать. Стандартная функция FilePos(fi) возвращает номер текущей позиции в файле.

Известно, что данные файлов хранятся на внешних носителях информации. В случае, когда файлы описываются в виде file of T, элементы типа T записываются на соответствующих носителях в виде последовательности двоичных цифр. Такой способ представления данных удобен для внешней памяти (магнитные диски и магнитные ленты, оптические диски и др.). Для устройства ввода /вывода (клавиатура, экран, принтер и т.п.) соответствующие данные должны быть представлены во внешней форме, то есть через строки символов.

Для того чтобы упростить общения пользователя и компьютера, в языке Паскаль информация, предназначенная для пользователя, представляется в виде текстовых файлов. Текстовый файл состоит из последовательности символов, разделённых на строки. Длина строк произвольна. Конец каждой строки обозначается специальным символом EOL(End of Line-конец строки). Так как длина строк произвольна, то позицию некоторой строки в файле нельзя знать заранее. Следовательно, доступ к элементам текстового файла может быть только последовательным.

Пример:

```
Program копирование;  
Var sursa, dest: file;  
Citit, scrib: word; tampon: array [1...2048] of char;  
Numsurs, numdest: string [14];  
Begin writeln ('fisierul sursa :'); readln (numsurs);  
Assign (sursa, numsurs);  
Reset (sursa, 1);  
Writeln ('fisierul destinatie :'); readln (numdest);  
Assign (dest, numdest); rewrite (dest, 1);  
Writeln ('copiere de', FileSize (sursa), 'octezi...');  
Repeat  
BlokRead (sursa, tampon, 2048, citit)  
BlokWrite (dest, tampon, citit, scrib);  
Until (citit = 0) or (scrib < > citit); close (sursa); close (dest);
```

End.

### 4.3 Операторы.

Программа на языке Паскаль состоит из двух частей: раздела описаний и раздела операторов. В разделе описаний описываются данные, которыми будет оперировать программа, а в разделе операторов описываются выполняемые над этими данными действия.

Действия, необходимые для обработки данных, и порядок их выполнения задаются с помощью операторов. Существуют две категории операторов:

- 1) Простые операторы
- 2) Сложные операторы

#### 4.3.1 Простые операторы

Называются простые операторы те операторы, которые не содержат других операторов.

##### 4.3.1.1 Оператор присваивания

Оператор присваивания имеет вид:

<Оператор присваивания>:=<Переменная>:=<Выражение>

При выполнении оператора присваивания происходит следующее:

- а) вычисляется выражения, которое стоит в правой части;
- б) полученное значение присваивается переменной, стоящей в левой части;

Присваивание возможно только тогда, когда переменная и результат вычисления выражения совместимы с точки зрения присваивания. В противном случае возникает ошибка.

Переменная и результат вычисления выражения являются, совместимы точки зрения присваивания, если справедливо одно из следующих утверждений:

- 1) тип переменной и тип результат идентичны;
- 2) тип результата является интервалом типа переменной;
- 3) оба типа являются интервалом одного и того же типа, а тип результата принадлежит интервальному типу переменной;
- 4) тип переменной real, а тип результата integer или его интервал.

##### 4.3.1.2 Оператор процедуры

Процедура представляется собой подпрограмму, к которой в процессе выполнения можно обращаться произвольное число раз. Каждая процедура имеет свое имя, например:

Read, writeln, CitireDate.

Оператор процедуры вызывает процедуру с соответствующим именем. Тип каждого фактического параметра и порядок его появления в списке указываются разделе описаний соответствующих процедур.

#### 4.3.1.3 Оператор перехода

Программа представлена в виде блоков программного кода. Оператор безусловного перехода позволяет прерывать ход программного кода и перейти в другую точку программы, без каких либо условий. В общем, виде инструкция goto записывается следующим образом:

Goto e

где e — это идентификатор, находящийся перед инструкцией, которая должна быть выполнена после инструкции goto.

Метка, используемая в инструкции goto, должна быть объявлена в разделе меток, который начинается словом label и располагается перед разделом объявления переменных.

В программе метка ставится перед инструкцией, к которой должен быть выполнен переход в результате выполнения инструкции goto. Сразу после метки ставится двоеточие.

Пример: label 5,8

```
Const a = 1.4; b = 7, 03;
```

```
Var x, y, z: real
```

```
Begin
```

```
Writeln ('y'); readln(y); X: = sgr(y);
```

```
5: if x > 5 then goto 8
```

```
Z: = x+a+b;
```

```
X: = sgrt (z);
```

```
Goto 5;
```

```
8: end;
```

#### 4.3.1.4 Пустой оператор

Выполнение данного оператора никак не влияет на значения переменных используемых в программе. Обычно пустой оператор используется на этапах разработки отладки сложных программ. Хотя пустой оператор не выполняет никаких действий, его присутствие может повлиять на ход программы.

#### 4.3.2 Структурированный оператор

Оператор называется структурированный, который содержит в своем составе несколько простых операторов (2 и более).

#### 4.3.2.1 Составной оператор BEGIN- END

Иногда для правильного решения программы в языке Паскаль нужно чтоб в нужном месте присутствовал только один оператор, во время решения после алгоритма в том же месте должны присутствовать множества операторов. Ключевое слова множества операторов это <begin> и <end>.

Составной оператор begin – end может состоять из одного или множества операторов.

Пример:

```
Begin a: = b+3 end; begin y: =s +cos s; z: = y-4ac; end;
```

```
Begin I: =1; begin c: =a + b; y: = sin c end; end;
```

#### 4.3.2.2 Условный оператор if.

Простой оператор выбора if выполняет одно из двух возможных действий в зависимости от значения некоторого условия – логического выражения. Синтаксис данного оператора:

<Оператор if>::=

if<Логическое выражение типа>

then <Оператор>

[else <Оператор>]

Выполнение оператора if начинается с проверки условия. Если результатом проверки является true ,то выполняется оператор стоящий после ключевого слова then. Если условие принимает значение false,то выполняется оператор, стоящий после ключевого слова else (если оно есть), или управление передаётся оператору, следующему непосредственно за оператором if.

В следующей программе оператор if используется для определения максимума из двух чисел X и y, считываемых с клавиатуры.

```
Program pi;  
{ Определение max из двух чисел }  
var x,y,max:real;  
begin  
  writeln ( ' Введите два числа: ' );  
  write('x='); readln(x);  
  write('y='); readln(y);  
  if x>=y then max:=x else max:=y;  
  writeln('max=',max);  
  readln;  
end.
```

Отметим, что в языке Паскаль символ "; " является частью оператора, а используется в качестве разделителя. Следовательно, если в операторе:

```
If B then s
```

перед S поставить пустой оператор

```
if B then ; s
```

тогда s не будет входить в состав условного оператора. В таком случае s будет выполняться независимо от значения B.

Если в операторе

```
If B then I else J
```

после I поставить символ " ", то получим синтаксически неправильно составленную программу:

```
if B then I ; else J
```

В данном случае текст else J интерпретируется как оператор , стоящий после условного оператора.

Оператор выбора Case

Оператор выбора case состоит из выражения, называемого переключателем (селектором), списка констант и соответствующего ему списка операторов. Каждому оператору из списка соответствует одна или несколько констант выбора.

Синтаксис рассматриваемого оператора:

```
Case I OF
```

```
C1:<оператор1>;
```

```
C2:<оператор2>;      или
```

```
.....
```

```
Cn:<оператор n>;
```

```
Otherwise<оператор>;
```

```
End;
```

```
Case I OF
```

```
C1:=<оператор>;
```

```
C2:=<оператор>;
```

```
.....
```

```
Cn:<оператор n>;
```

```
ELSE<оператор>;
```

```
End;
```

Селектор должен относиться к порядковому типу. Константы выбора должны быть совместимы с типом селектора и не могут повторяться:

Пример:

```
Var I : integer; c:char; a,b,y:real;
```

```
Case I of
```

```
0, 2, 4, 6, 8 : writeln ( ,Четная цифра ');
```

```
1, 3, 5, 7, 9 : writeln ('нечетная цифра');
```

```
end;
```

```
case c of
```

```
'+' : y:=a+b;
```

```
'-' : y:=a-b;
```

```
'*' : y:=a*b;
```

```
 '/' : y:=a/b;
```

end;

Выполнения оператора case начинается с проверки селектора. Если селектор принимает одно из значений констант выбора, то выполняется оператор, соответствующий этой константе.

Отметим, что в некоторых версиях языка синтаксис и семантика оператора case были изменены. Список вариантов может включать оператор, которому предшествует ключевое слово else (в некоторых версиях otherwise). Константы выбора можно заменить интервалами вида

<Константа>.. <Константа>

#### 4.3.2.3 Операторы цикла

1) Оператор FOR

2) Оператор REPEAT

3) Оператор WHILE

Все эти операторы исполняют свою функцию только один раз.

Оператор FOR предназначен для повторного выполнения другого оператора в зависимости от значения управляющей переменной. Синтаксис рассматриваемого оператора:

<Оператор for> ::=

for <Переменная> ::= <Выражение>

<Шаг> <Выражение>

do <Оператор>

< Шаг> ::= to/ downto

Переменная, находящаяся после ключевого слова for, называется управляющей переменной или параметром цикла. Эта переменная должна принадлежать некоторому порядковому типу.

Значения выражений, входящих в состав оператора for, должны быть совместимыми, с точки зрения присваивания, с параметром цикла. Эти выражения проверяются один раз в начале цикла. Первое выражение указывает исходное значение параметра цикла, а второе – конечное значение.

Оператор, стоящий после ключевого слова do, выполняется для каждого значения из диапазона, определяемого начальным и конечным значениями.

Если в операторе for используется шаг to, то значения параметра цикла увеличивается при каждом повторении, переходя к следующему значению. Если исходное значение больше конечного,

оператор, расположенный после ключевого слова `do`, не выполняется ни разу. Если в операторе `for` используется шаг `downto`, то значения параметра цикла уменьшается при каждом повторении, переходя к значению, предшествующему текущему. Если начальное значения меньше конечно то оператор, расположенный после ключевого слова `do`, не выполняется ни разу.

Пример

```
Program P 10;  
{оператор for}  
var i : integer; c: char;  
begin  
  for i:=0 to 9 do write (i : 2);  
  writeln;  
  for i:=9 downto 0 do write (i : 2);  
  writeln;  
  for c:='A' to 'Z' do write (c : 2);  
  writeln;  
  for c:='Z' to 'A' do write (c : 2);  
  writeln;  
  readln;  
end.
```

Значения параметра цикла не могут быть изменены внутри цикла, т.е.

- 1) параметру цикла не присваиваются никакие значения;
- 2) параметр цикла не может быть параметром цикла другого вложения оператора `for`;
- 3) нельзя называть процедуры `read`, `readln`, в которых указывается параметр цикла

После выхода из оператора `for` значения параметра цикла не определено, за исключением случая, когда выход из цикла осуществляется принудительно, через оператора безусловного перехода `goto`.

Оператор `repeat`.

Оператор `repeat` указывает на многократное выполнение последовательности операторов в зависимости от значения некоторого логического выражения. Операторы, расположенные между ключевыми словами `repeat` и `until`, выполняется многократно до тех пор, пока логическое выражение сохраняет значения `false`. Как

только логическое выражение становится истинным, управление переходит следующему оператору. Очевидно, что операторы, стоящие между ключевыми словами `repeat` и `until`, будут выполнены, по крайней мере, один раз, так как логическое выражение проверяется лишь после выполнения последовательности операторов.

Как правило, оператор `repeat` используется вместо оператора `while` в тех случаях, когда проверка логического выражения, управляющего повторением, должна производиться после выполнения соответствующей последовательности операторов. Выполнения оператора `repeat` завершается, когда пользователь вводит `i:=0`.

### Оператор `while`

Оператор `while` состоит из логического выражения, которая управляет многократным выполнением других операторов. Оператор, расположенный после ключевого слова `do`, выполняется до тех пор, пока логическое выражение принимает значение `true`. Как только логическое значение принимает значения `false`, оператор, стоящий после ключевого слова `do`, перестает выполняться.

Обычно оператор `while` используется для организаций повторных вычислений в циклах, где управляющей является переменная типа `real`.

### 4.4 Формальные параметры функция/процедура.

Часто в ходе написания программы появляются ситуации, когда надо, чтобы одна и та же часть программы была выполнена много раз, используя различные данные. В таких ситуациях используются подпрограммы. Одна подпрограмма, правильно записанная в программе только один раз и содержащая тип и необходимое число параметров может быть вызвана множество раз, из разных частей программы, с разными значениями параметров, но совершая один и тот же алгоритм.

В Паскале существует 2 вида подпрограмм: процедуры и функции. Разница между ними состоит в том, что из числа значений рассчитанных и возвращенных на те места, из которых они были вызваны. Процедура рассчитывает множество значений, в то время как функция только одно, позволяя функции выполняться прямо из выражения, которое должно быть рассчитано. Процедуры и функции определяются в

декларированную часть подпрограммы и являются последними декларированными из множества данных. Такого вида декларированные данные ассоциируются с идентификаторами с одной частью программы, т.о. она может быть активизирована с помощью параметров функции в любой момент.

#### 4.4.1 Процедуры

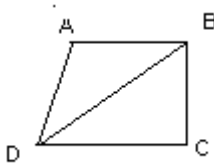
Объявление процедур расположено в части процедуры функции и имеет следующий синтаксис:

*Procedure* имя (список формальных параметров);

*Var* [Список локальных переменных];

*Begin* [блок процедур] *end*;

Для понятного объяснения воспользуемся конкретным примером. Предположим надо вычислить площадь трапеции, в которой известны все 4 стороны и 1 диагональ. Для вычисления площади прямоугольника вычислим площадь каждого треугольника отдельно, используя формулу Герона:



$S = \sqrt{p(p-a)(p-b)(p-c)}$ , где  $p = (a+b+c)/2$ ;  $a, b, c$  – длины сторон треугольника.

Без использования процедур решение этой задачи будет выглядеть так:

*Program* *patrulater1*;

*Var* *AB, BC, CD, DA, BD, a, b, c, p, s, s1, s2, : real*;

*Begin* *readln (AB, BC, CD, DA, BD)*;

*a:=AB; b:=DA; c:=BD; p:=(a+b+c)/2*;

*S!:=sqrt(p\*(p-a)\*(p-b)\*(p-c))*;

*a:=BC; b:=CD; c:=BD; p:=(a+b+c)/2*;

*S2:=sqrt (p\*(p-a)\*(p-b)\*(p-c))*;

*S:=S1+S2; writeln ('S=', S)*; *end*.

Замечаем повторение некоторых инструкций, а именно вычисление  $p$  и  $S$ . Именно эта ситуация позволяет использовать процедуры. Использование в программе процедур представлена ниже:

*Program* *patrulater2*;

*Var* *AB, BC, CD, DA, BD, a, b, c, p, s, s1, s2, : real*;

```

Procedure aria;
Begin p:=(a+b+c)/2; S:=sqrt(p*(p-a)*(p-b)*(p-c)); end;
Begin readln (AB, BC, CD, DA, BD);
a:=AB; b:=DA; c:=BD; aria;
S1:=S; a:=BC; b:=CD; c:=BD; aria;
S2:=S; S:=S1+S2; writeln ('S=', S);
End.

```

В данном примере 2 раза обращаемся к процедуре *aria*, но перед каждым обращением к процедуре выполняется несколько инструкций присваивания, которые определяют значение переменных *a*, *b*, *c*. Каждое обращение вызывает процедуру и в результате, получаем площадь треугольника, для которого были заданы параметры. Так связь между процедурой и основной программой выполняется посредством переменных *a*, *b*, *c* и *S*. Но процедура еще содержит и переменную *p*, которая является локальной процедурой и не входит в основной блок программы. В такой ситуации объявление и описание переменной *p* может быть выполнено внутри процедуры. Такие переменные называются локальными переменными (для процедуры). Они не могут воздействовать на глобальные переменные (с основной программы) под тем же именем. Разница состоит в видимости: глобальная переменная (для компилятора) видимо всей программе. Но локальная переменная, видима только внутри процедуры.

Процедура описывается следующим образом:

```

Procedure p(x1; x2...; xn);
D;
Begin
.....
End;

```

где

*P*.- имя процедуры;

(*x*; *x*2...*x*n)-произвольный список формальных параметров.

Тело процедуры включает:

*D*-произвольные локальные описания, сгруппированные таким же образом, как и в случае функций.

*Begin*---*end* –составной оператор , в котором имя процедуры не появляется в операциях присваивания.

Процедура может возвращать несколько результатов, но не через своё имя, а через специально предназначенные для этого переменные (следующие за словом *var*) из списка формальных параметров.

Параметры их списка, вводимые через описания вида

$v_1, v_2, \dots, v_k: tp$

называются параметрами – значениями. Они служат для передачи значений из основной программы в процедуру.

Формальные параметры, введенные в список через описания вида

`Var v1, v2, ...; tp`

называются параметрами – переменных и служат для возвращения результатов из процедуры в основную программу.

Запуск процедуры выполняется путем ее вызова:

`P (a1, a2, ..., an),`

где  $(a_1, a_2, \dots, a_n)$ - список фактических параметров. В отличие от функции, обращение к процедуре является оператором. Операторы вызова процедур вставляются в основную программу там, где требуется выполнить соответствующую обработку данных.

Для параметра – значения в качестве фактического параметра можно использовать любое выражение соответствующего типа, в частности константу или переменную. Изменения параметров-значений не передаются во внешнее окружение подпрограммы.

Для параметра переменной в качестве фактического параметра можно использовать только переменную. Все изменения указанных параметров будут переданы в программу, которая вызвала соответствующую процедуру.

#### 4.4.2 Функции

На языке Паскаль функция описывается следующим образом:

`Function f(x1, x2...; xn): tr;`

`D;`

`Begin`

`...`

`F:=e;`

`....`

`end;`

Первая строка это заголовок функции, состоящий из:

f-имя функций:

$(x_1, x_2, \dots, x_n)$ -произвольный список формальных параметров, являющихся аргументами функции;

tr-тип результата; может быть простым или ссылочным.

За заголовком следует тело функции, состоящее из произвольных локальных описаний D и составного оператора `begin...end`.

Локальные описания состоят из следующих разделов (некоторые из которых могут отсутствовать): `label, const, type, var, function, procedure`.

Имя функции  $f$  должно появиться хотя бы один раз в левой части некоторого оператора присваивания:  $f:=e$ . Последнее значение, присвоенное функции  $f$ , возвращается в основную программу.

Как правило, формальный параметр из списка  $(x_1, x_2, \dots, x_n)$  имеет вид:

$v_1, v_2, \dots, v_k:tp,$

где  $v_1, v_2, \dots, v_k$ -идентификаторы, а  $tp$ -имя типа.

Обращение к функции  $f$  осуществляется следующим образом:

$f(a_1, a_2, \dots, a_n)$ - список фактических параметров. Обычно фактическими параметрами являются выражения, значения которых передаются функции. Связь между фактическим и формальным параметрами определяется их позицией в рассматриваемых списках. Фактический параметр должен быть совместимым с точки зрения присваивания с типом формального параметра.

Program pf;

Var z:integer; s:real;

Function fact(n:integer):integer;

Var I,k:integer;

Begin K:=1; for I:=1 to n do k:=k\*I; fact:=k; end;

Begin writeln('Выберает j целое число'); readln(z);

Writeln(z,'!=',fact(z));

S:=135+sin(62)\*fact(z)/10;

End.

Заметим что, функция `fact` имеет два формальных параметра:  $s$  типа `real` и  $z$  типа `integer`. Функция возвращает значение типа `real`. В теле функции описаны локальные переменные  $I$  и  $k$ .

При обращении к функции `fact` ( $z, s$ ) значение фактических параметров  $z$  и  $s$  передаются формальным параметрам соответственно  $I, k$ .

Синтаксис функции:

Function имя(формальные параметры ): тип функций;

### Побочные эффекты

Любая функция возвращает единственный результат-значение функции. Как правило, значение аргументов передаются функции через параметры-значения, а результат возвращается на место вызова через имя функции. Помимо этого, язык Паскаль допускает передачу аргументов через глобальные переменные параметры-переменные.

Под побочным эффектом понимается присваивание некоторого значения глобальной переменной или формальному параметру-

переменной. Подобные эффекты могут непредвидимым образом влиять на ход программы, усложняя тем самым процесс ее отладки.

Пример:

```
Program P6;  
Var a,b:integer;  
Function par (x:integer):integer;  
Begin b:=b-x; par:=x*x; end;  
Begin b:=10;  
A:=par(b);  
Writeln(a,b);  
End.
```

Ответ: (100 0)

Побочные эффекты вносят отклонения в стандартный процесс связи программа-подпрограмма, при котором используемые переменные указываются явно в качестве формальных параметров описания и фактических параметров при выводе. Последствие побочных эффектов могут распространяться на области видимости глобальных описаний и взаимодействовать с подобными эффектами, возникающих при выполнении других процедур и функций. В таких условиях использование глобальных переменных становится рискованным. Поэтому для составления сложных программ рекомендуется:

- Связь функции со средой вызова осуществлять посредством передачи данных в функцию через формальные параметры-значения, возврат единственного результата – через ее имя.
- Связь процедур со средой вызова осуществлять посредством передачи данных в процедуру через формальные параметры-значения или параметры-переменные, а возврат результатов - через параметры переменные.
- Глобальные переменные можно использовать для передачи данных в программы, однако внутри подпрограмм их значения не должны меняться.

#### 4.4.3 Процедуры и функции

Определения на тип процедура или тип функция имеет динамическую манипуляцию с подпрограммами (код подпрограммы). Это значит, что одно имя подпрограммы может быть отнесено к одному параметру типа процедура или функция. Имя подпрограммы может быть представлено как параметр для функции. Определение типа процедура определяет имена типов, а так же имена и типы параметров; определение типа функции определяет и типы значений возвращенных в функции.

Пример: type tipf = функция(x,y:integer):integer;  
tipr = процедура (var n: real; v: integer);  
Var f1, f2: tipf;  
p; tipr;

Имеем определение функции:  
Функция f(x,y:integer):integer;  
Begin f: =x+y+1; end.

Пример. В этом примере определен один тип функции, названный tipf. Делается прямой и непрямой вызов процедуры apelparf с параметрами на тип процедуры.

```
{ $F+ }  
Program tip;  
Type tipf: функция(x, y: integer): integer;  
Var varf: tipf;  
Процедура apelparf(fpar: tipf; x,y: integer);  
Begin writeln (fpar(x, y)); end;  
Собирательная функция (x,y: integer): integer;  
Begin собирательная: = x + y; end;  
Сниженная функция( x,y: integer): integer;  
Begin сниженная: =x+y; end;  
Begin  
Apelparf (ссобирательная, 2, 3); apelparf (сниженная, 2, 3);  
Varf: = собирательная; apelparf (varf, 4, 5);  
Varf: = сниженная; apelparf (varf, 4, 5); end.
```

Замечание:

- Использование параметров на тип процедура или функция производится тогда, когда компилятор выполняется посредством директивы.

#### 4.4.4 Рекурсия.

Рекурсией называется прямое или косвенное обращение подпрограммы к самой себе.

Подпрограмма, которая вызывает саму себя, называется рекурсивной.

```
Program pr;  
Function степень (база: integer): integer;  
Begin if база <= 0 then степень :=1 else степень: =база*  
Степень(база -1); end;  
Begin writeln('2^4=',степень(2,4));end.
```

При любом вызове программы в память компьютера записывается следующая информация:

- текущие значения параметров, передаваемые через значение;
- местонахождение параметров-переменных;
- адрес возврата, т.е. адрес оператора, который следует за оператором вызова.

Таким образом, при рекурсивном вызове занимаемая область памяти увеличивается очень быстро, что ведет к риску переполнения памяти компьютера{\$S +}. Это можно избежать с помощью операторов (for, while, repeat).

## БИБЛИОГРАФИЯ

Nr	Название работы	автор	Год издан
1	Pascal и Turbo Pascal	Bălănescu T. ...	1992
2	Введение в Turbo Pascal	Kalisz E.	1997
3	Pascal в интересах каждого	Anghel, Florin Stînga	1992
4	Turbo Pascal 6.0 Гид для пользования	Sandur Covax	1993
5	Turbo Pascal 6.0 Программы	Lucian Vasiu ...	1994
6	Обработка файлов в Pascal	Roşca I.	1994
7	Îndrumare metodică de laborator N510	FCIM	1996
8	Персональные компьютеры	Gremalschi A.	1997
9	Structura calculatoarelor numerice	Gremalschi A.	1996
10	Языки C и C++ для начинающих	Negrescu L.	1996
11	Turbo C++	Cojocaru C.O.	1994
12	Введение в язык Паскаль	Абрамов В.Г.	1988
13	Введение в программирование на языке Паскаль	Эрбс, Хайнц-Эрих, Штольц, Отто.	1989
14	Вычислительная техника и программирование	Алексеев В.Е.	1991
15	Англо-русский словарь по программированию и информатике	Борковский А.Б.	1992
16	Русско-англо-немецко-французский словарь по вычислительной технике	Масловский Е.К.	1990
17	Язык программирования Си	Керниган Б. Ритчи Д.	1992
18	Программирование на языке Си для	Григорьев А.	1990

	персональных компьютеров		
19	Программирование на языке Си, справочное пособие	Котлинская Г.П.	1991