

Copyright pluttan&fixii

Привет! Это Трубусы и Уточки, мы создаем свою ботву, этот файл малая ее часть.  
Пользоваться и распространять файлы конечно же можно. Если вы нашли ошибку в файле, можете  
исправить ее в исходном коде и подать на слияние или просто написать в issue.

**Так же вы можете купить распечатанную версию данного файла в виде книжки.**

**По всем вопросам писать в ВК.**

Приятного бота)

---

GitHub: <https://github.com/pluttan>

VK: <https://vk.com/pluttan>

VK: [https://vk.com/f\\_i\\_i\\_x\\_i\\_i](https://vk.com/f_i_i_x_i_i)

# Подготовка к экзамену

## Основы программирования

Над файлом работали:  
pluttan & fixii

## Оглавление

1	Синтаксис и семантика языков программирования. Алфавит языка Pascal. Описание синтаксиса языка: синтаксические диаграммы. Примеры. . . . .	3
2	Представление данных в языке Pascal: константы и переменные. Классификация скалярных типов данных, их внутреннее представление, операции над ними. Примеры. . . . .	4
3	Совместимость типов данных и операции преобразования типов. Примеры. . . . .	7
4	Присваивание, условный оператор, оператор выбора. Синтаксис операторов, их особенности и примеры использования. . . . .	8
5	Операторы циклов языка Pascal. Синтаксис операторов, их особенности и примеры использования. . . . .	9
6	Поисковый цикл. Неструктурная и структурная реализации поискового цикла. . . . .	10
7	Массивы языка Pascal. Описание, внутреннее представление, операции над массивами и их элементами. Примеры. . . . .	12
8	Строки языка Pascal. Описание, внутреннее представление, операции над строками и их элементами. Примеры. . . . .	14
9	Множества языка Pascal. Описание, внутреннее представление, операции над множествами и их элементами. Примеры. . . . .	16
10	Записи языка Pascal. Описание, внутреннее представление, операции над записями и их элементами. Примеры. . . . .	18
11	Процедуры и функции. Определение, описание, особенности. Примеры. . . . .	20
12	Способы передачи данных в подпрограмму на языке Pascal. Примеры. . . . .	21
13	Локальные и глобальные переменные, законы «видимости» идентификаторов. Примеры. . . . .	22
14	Формальные и фактические параметры подпрограмм языка Pascal. Примеры. . . . .	23
15	Параметры-строки, параметры-массивы. Примеры. . . . .	24
16	Принципы разработки универсальных подпрограмм: «открытые» массивы. Примеры. . . . .	25
17	Принципы разработки универсальных подпрограмм: нетипизированные параметры, параметры процедурного типа. Примеры. . . . .	26
18	Структура модуля языка Pascal. Законы видимости идентификаторов. Доступ к «перекрытым» идентификаторам. Примеры. . . . .	27
19	Рекурсия. Виды рекурсии. Особенности программирования. Достоинства и недостатки. Пример. . . . .	28
20	Адресация динамической памяти: понятие адреса, операции получения адреса и разыменования. Процедуры получения памяти и освобождения ее. Примеры. . . . .	29
21	Списковые структуры данных. Классификация и основные приемы работы с ними: создание элемента, добавление элемента к списку, удаление элемента из списка. Область применения списковых структур данных. Пример. . . . .	31
22	Основы файловой системы: файл, каталог, полное имя файла, внутреннее представление информации в файле. Файловая переменная. Операции открытия и закрытия файлов. Примеры. . . . .	35
23	Текстовые файлы. Внутреннее представление информации в файле. Операции над файлами. Пример. . . . .	37
24	Типизированные файлы: внутреннее представление информации в файле. Операции над файлами. Пример. . . . .	39
25	Нетипизированные файлы. Внутреннее представление информации в файле. Операции над файлами. Пример. . . . .	40
26	Классы консольного режима среды Lazarus: описание классов, поля и методы, объявление объектов класса, доступ к полям и методам объекта, ограничение доступа. Пример. . . . .	41
27	Классы консольного режима среды Lazarus: Способы инициализации полей. Неявный параметр Self. Пример. . . . .	42
28	Процедурная и объектная декомпозиция. Диаграммы классов. Отношения между классами. Примеры. . . . .	43
29	Динамические объекты и объекты с динамическими полями в консольном режиме среды Lazarus. Примеры. . . . .	46
30	Технология событийного программирования. События операционной системы, сообщения и события Lazarus. Основные события Lazarus. Примеры. . . . .	47

# 1 Синтаксис и семантика языков программирования. Алфавит языка Pascal. Описание синтаксиса языка: синтаксические диаграммы. Примеры.

Иванова Г.С. Основы программирования 2002, стр. 28 - 30

Иванова Г.С. Курс Основы программирования, през. 1, с. 9, 26

**Синтаксис** - совокупность правил, определяющих допустимые конструкции (слова, предложения) языка, его форму. «Защищенный» синтаксис предполагает, что предложения языка строятся по правилам, которые позволяют автоматически выявлять большой процент ошибок в программах.

**Семантика** - совокупность правил, определяющих смысл синтаксически корректных конструкций языка, его содержание. Ясная или «интуитивно-понятная» семантика – семантика, позволяющая без большого труда определять смысл программы или «читать» ее.

## Алфавит языка Паскаль

1. Латинские буквы без различия строчных и прописных;
2. Арабские цифры: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9;
3. Шестнадцатеричные цифры: 0..9, a..f или A..F;
4. Специальные символы: + - \* / = := ; и т. д.;
5. Служебные слова: do, while, begin, end и т. д.

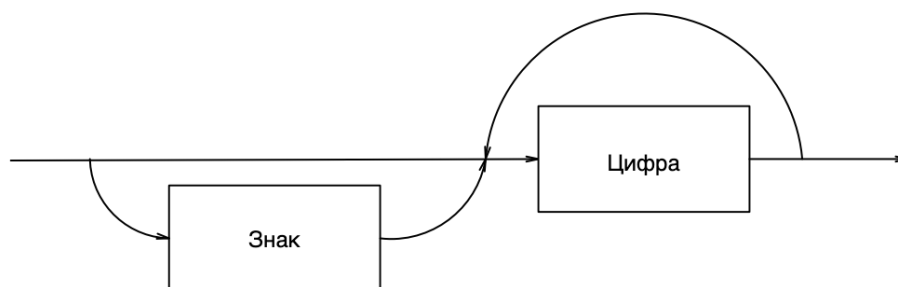
## Описание синтаксиса языка

Включает определение алфавита и правил построения различных конструкций языка из символов алфавита и более простых конструкций. Для этого обычно используют либо форму Бэкуса-Наура (БНФ), либо синтаксические диаграммы.

**Синтаксические диаграммы** отображают правила построения конструкций в более наглядной форме. На такой диаграмме:

- символы алфавита изображают блоками в овальных рамках;
- названия конструкций в прямоугольных рамках;
- правила построения конструкций – в виде линии со стрелками на концах;
- разветвление линии означает, что при построении конструкции есть варианты.

Пример: синтаксическая диаграмма, иллюстрирующая первые два правила описания конструкции <целое>



## 2 Представление данных в языке Pascal: константы и переменные. Классификация скалярных типов данных, их внутреннее представление, операции над ними. Примеры.

Иванова Г.С. Основы программирования 2002, стр. 31 - 40

Иванова Г.С. Курс Основы программирования, през. 1, с. 27 - 43

**Константы** – определяются один раз и не изменяются во время выполнения программы. Используют следующие типы констант:

1. Целые и вещественные десятичные числа;
2. Шестнадцатеричные числа;
3. Логические константы;
4. Символьные константы – (записываются либо в апострофах, либо в виде соответствующих кодов по таблице ASCII);
5. Строки символов – записываются в апострофах;
6. Конструкторы множеств;
7. Нулевой адрес – nil.

Константы используются в двух формах:

- **Литерал** – представляет собой значение константы, записанное непосредственно в программе;
- **Поименованные константы** объявляются в инструкции раздела описаний const. Обращение к ним осуществляется по имени (идентификатору).

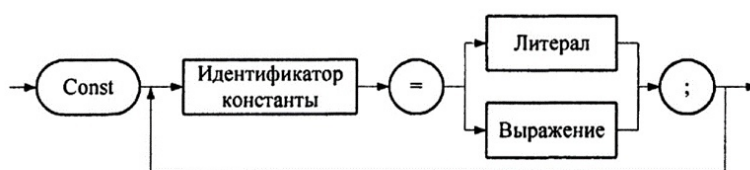


Рис. 2.3. Синтаксическая диаграмма конструкции <Объявление констант>

**Переменные** – поименованные значения, которые могут изменяться в процессе выполнения программы. Их объявление выполняют в разделе описаний программы, причем указывается не только идентификатор переменной, но и ее тип. Обращение к переменным также осуществляют по идентификатору.

**Тип переменной** – определяет возможный набор значений данной переменной, размер ее внутреннего представления и множество операций, которые могут выполняться над переменной.

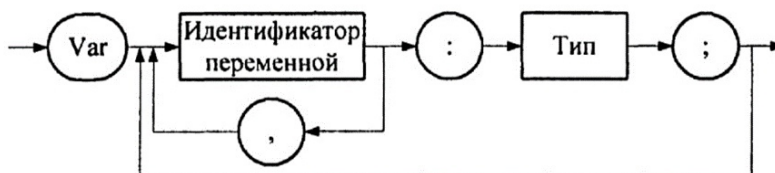
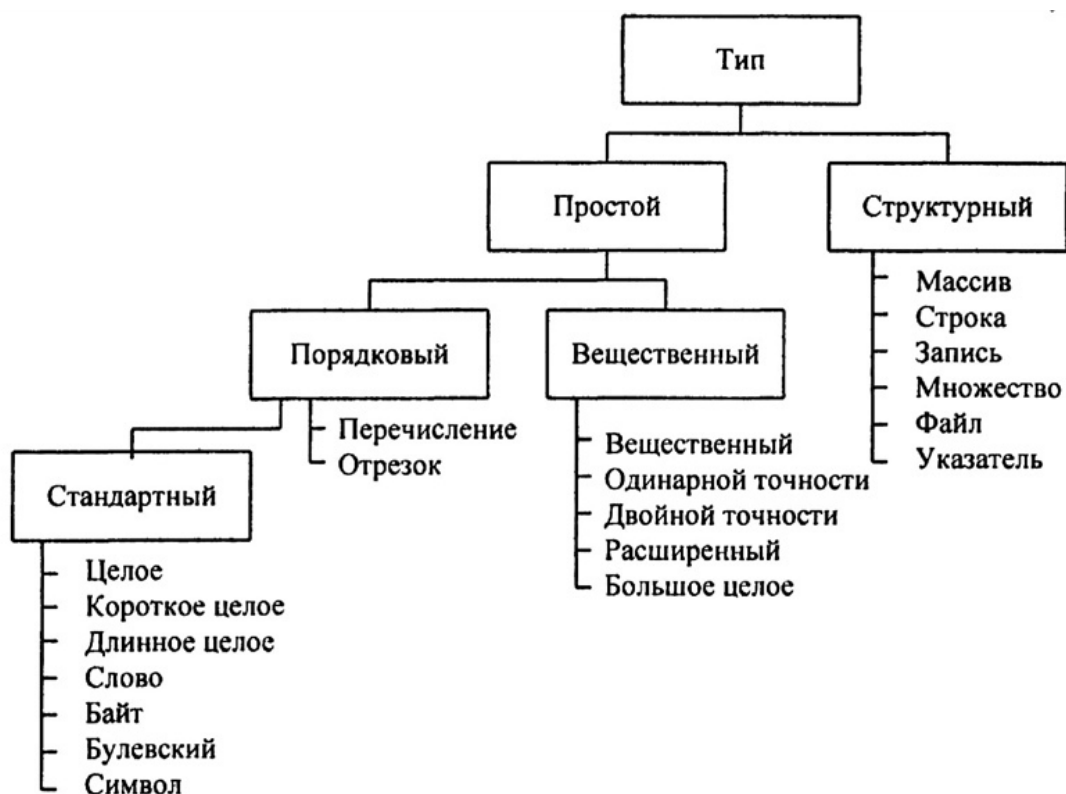


Рис. 2.4. Синтаксическая диаграмма конструкции <Объявление переменных>

Простые (скалярные) типы описывают упорядоченные наборы значений. Они делятся на **порядковые** и **вещественные**.

- **Группа порядковых типов** объединяет типы переменных, набор значений которых конечен.
- **Группа вещественных типов** объединяет типы с условно бесконечным набором значений.



#### • Стандартные типы

Название	Обозначение	Диапазон значений	Длина внутреннего представления, байт
Целое	Integer	-32768..32767	2 (со знаком)
Короткое целое	ShortInt	-128..127	1 (со знаком)
Длинное целое	LongInt	$-2^{31}..2^{31}-1$	4 (со знаком)
Байт	Byte	0..255	1 (без знака)
Слово	Word	0..65535	2 (без знака)
Логический	Boolean	True, false	1
Символьный	Char	Символы по таблице ASCII	1

\***Логический тип** Boolean включает два значения – true(1) и false(0), но в памяти значения данного типа занимают целый байт;

\***Символьный тип** char определяет набор символов по таблице ASCII.

- **Перечисляемый тип (перечисление)** – формируется из значений, определенных программистом при объявлении типа. Перечень задают через запятую в круглых скобках. Во внутреннем представлении значения перечисляемого типа кодируются целыми числами, начиная с нуля. Например:

Var D : (Mon, The, Wed, Thu, Fri, Sat, Sun); {переменная D принимает одно из перечисленных значений}

Mon = 0; The = 1 ... Sun = 6;

- **Отрезок** – определяется как диапазон значений некоторого уже определенного типа. Например:

Var D : 1..31;

#### • Вещественные типы

Название	Обозначение	Кол-во десятичных значащих цифр	Диапазон изменения порядка	Длина внутреннего представления, байт
Вещественный	Real	11..12	-39..+38	6
Одинарной точности	Single	7..8	-45..+38	4
Двойной точности	Double	15..16	-324..+308	8
Расширенный	Extended	19..20	-4951..4932	10
Большое целое	Comp	19..20	$-2^{63} + 1..2^{63} - 1$	8

### Операции

1. **Арифметические операции** – применяют к вещественным и целым константам и переменным: +, -, \*, / - {вещественное деление}, div - {целочисленное деление}, mod - {остаток от деления}  
\*приоритет операций рассчитывается как в математике (+ скобки).

Операции	Приоритет
@, not	1
*, /, div, mod, and, shr, shl	2
+, -, or, xor	3
<, >, <=, >=, =, <>	4

2. **Операции отношения (больше, меньше, равно и т.д.)** – применяют к числам, символам, строкам – в результате получают логическое значение:  
< {меньше}, > {больше}, = {равно},  
<> {не равно}, <= {меньше или равно}, >= {больше или равно}
3. **Логические операции** – применяют к логическим значениям – результат логическое значение: not, and, or, xor
4. **Поразрядные операции** – выполняются поразрядно, применяют к целым, результат – целое число:  
not, and, or, xor, shr - {сдвиг вправо}, shl - {сдвиг влево}

### 3 Совместимость типов данных и операции преобразования типов. Примеры.

*Иванова Г.С. Основы программирования 2002, стр. 41 - 42*

*Иванова Г.С. Курс Основы программирования, през. 1, сл. 43, 45*

По правилам **совместимы**:

1. все целые типы между собой;
2. все вещественные типы между собой;
3. отрезок базового типа и базовый тип;
4. два отрезка одного и того же базового типа;
5. символ и строка.

Для несовместимых типов результата и переменной, в которую его необходимо занести, при выполнении присваивания необходимо явное преобразование типов, например, посредством специальных функций:

- `trunc(<Вещественное выражение>)` – преобразует вещественное число в целое, отбрасывая дробную часть.
- `round(< Вещественное выражение>)` – округляет вещественное число до целого по правилам арифметики.

Пример: `trunc(4.5) = 4`, `round(4.5) = 5`

- `ord(<Порядковое вып.>)` – преобразует значение в его номер. Пример: `ord('A') = 65`.
- `chr(<Ц. вып.>)` – преобразует номер символа в символ. Пример: `chr(65) = 'A'`.

```
1
2 Var real_: real;
3   integer_: integer;
4 Begin
5   real_ := 12.6;
6   integer_ := trunc(real_);
7   write(integer_, ' ');
8   integer_ := round(real_);
9   write(integer_, ' ');
10
11 End.
```

## 4 Присваивание, условный оператор, оператор выбора. Синтаксис операторов, их особенности и примеры использования.

Иванова Г.С. Основы программирования 2002, стр. 40 – 42, 50, 56

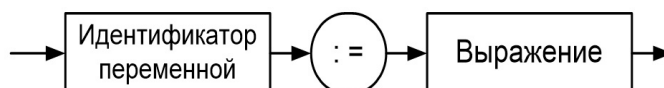
Иванова Г.С. Курс Основы программирования, през. 1, с. 43

Иванова Г.С. Курс Основы программирования, през. 2, с. 2 - 7

### Оператор присваивания

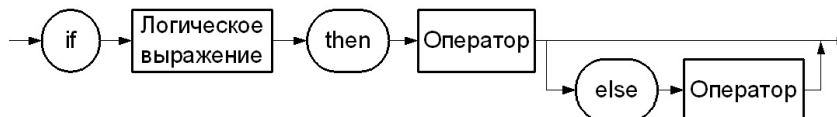
- используется для изменения значений переменных.

Корректное выполнение оператора предполагает, что результат вычисления и переменная правой части одного типа или совместимы по типу.



### Условный оператор (Оператор условной передачи управления)

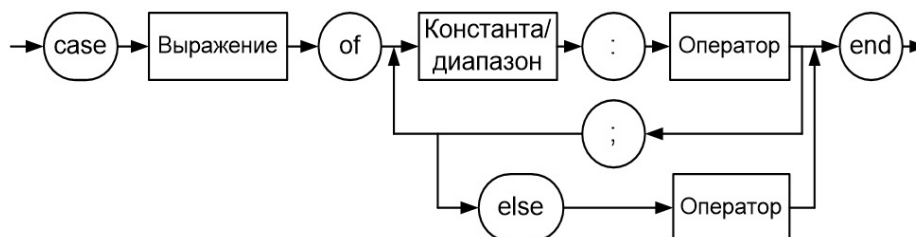
- используется при обработке вариантов вычислений и реализует конструкцию ветвления.



*примечание:* В каждой ветви допускается запись одного оператора (в том числе и другого if) или составного оператора. Составным оператором называют последовательность операторов, заключенную в операторные скобки begin...end. Операторы последовательности отделяют друг от друга точкой с запятой «;». Перед end точку с запятой можно не ставить. Перед else точка с запятой не ставится никогда, так как в этом случае запись условного оператора продолжается.

### Оператор выбора

- позволяет программировать несколько вариантов решения. Используется для реализации нескольких альтернативных вариантов действий, каждый из которых соответствует своим значениям некоторого параметра.



```

1
2 Var a: integer;
3 Begin
4   a := 1;
5   If a >= 1 Then writeln('a >= 1')
6   Else writeln('a < 1');
7   Case a Of
8     0: writeln('a = 0');
9     1: writeln('a = 1');
10    2: writeln('a = 2');
11  End;
12 End.
  
```



## 5 Операторы циклов языка Pascal. Синтаксис операторов, их особенности и примеры использования.

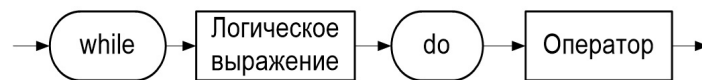
Иванова Г.С. Основы программирования 2002, стр. 58, 59

Иванова Г.С. Курс Основы программирования, през. 2, сл. 10 - 13

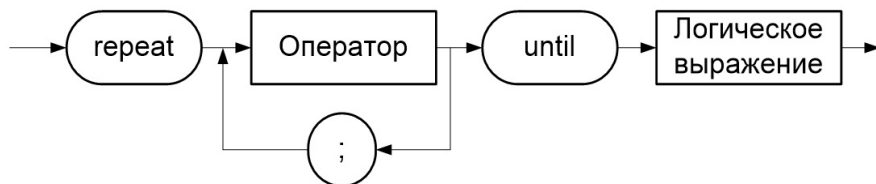
Циклы бывают:

- **Счетный цикл** – цикл, количество повторений которого известно или можно посчитать. Выход из такого цикла программируется по счетчику.
- **Итерационный цикл** – цикл, количество повторений которого неизвестно или считается неизвестным при построении цикла. Выход из цикла программируется по выполнению или нарушению условия.
- **Поисковый цикл** имеет два выхода – «нашли» либо «перебрали все и не нашли».

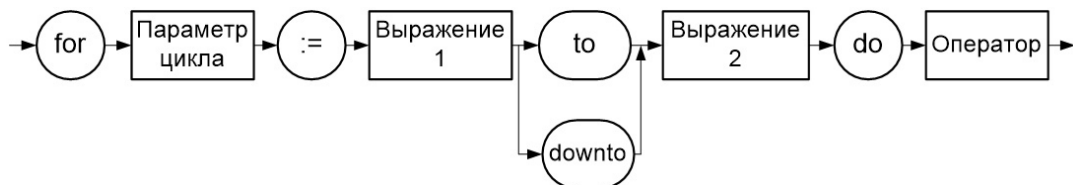
Цикл while (цикл пока)



Цикл repeat..until (цикл до)



Цикл for (счетный цикл)



```

1
2 Var q,u,i,z: longint;
3 Begin
4   z := 10;
5   u := 1;
6   For i:=1 To z Do
7     Begin
8       u := u * i;
9       writeln(i, '!' = ',u);
10    End;
11  While u <> 0 Do
12    Begin
13      q := q + (u Mod 10);
14      u := u Div 10;
15    End;
16  writeln('sum of nums of 10! = ', q )
17 End.
  
```

## 6 Поисковый цикл. Неструктурная и структурная реализации поискового цикла.

Иванова Г.С. Основы программирования 2002, стр. 69 - 76

**Поисковые циклы** – циклические процессы поиска некоторой информации в таблицах, выход из которых происходит при нахождении необходимой информации или по завершению процесса поиска, если необходимая информация не найдена.

**Неструктурная реализация** включает неструктурные передачи управления: оператор безусловной передачи управления и специальные процедуры.

**Структурная реализация**, наоборот, не включает такие передачи управления.

**Оператор безусловной передачи управления** – этот оператор передает управление в точку, определенную специальной меткой. Все метки в программе должны быть описаны инструкцией объявления меток label.

Метка ставится перед любым выполняемым оператором программы, причем на один оператор можно поставить несколько меток, например:

Label M1, M2, M3;...

M1:M2:M3:x:=x+1;

goto M1;

Неструктурную передачу управления также осуществляют ряд процедур:

- Break — реализует выход из цикла любого типа.
- Continue — осуществляет переход на следующую итерацию цикла, игнорируя оставшиеся до конца тела цикла операторы.
- Halt (<Код завершения>) — осуществляет выход из программы, возвращая операционной системе заданный код завершения.

Считается, что программа завершилась нормально, если код завершения равен нулю. Возвращение кода завершения, отличного от нуля, обычно означает, что программа завершена по обнаружении каких-либо ошибок. Коды завершения назначаются программистом, а информация о них помещается в программную документацию.

- Exit — осуществляет выход из подпрограммы. Если процедура использована в основной программе, то она выполняется аналогично Halt.

пример (поиск информации в массиве, см. листинг ниже):

```

1
2 Var a: array [1..10] Of integer;
3   i,f: integer;
4   flag: boolean;
5 Begin
6   randomize;
7   For i:=1 To 10 Do
8     Begin
9       a[i] := random(100);
10      write(a[i], ' ');
11    End;
12   f := a[6];
13   writeln;
14   {Unstructure}
15   For i:=1 To 10 Do
16     Begin
17       If a[i] = f Then
18         Begin
19           writeln(a[i], ' ');
20           break;
21         End;
22     End;
23   {Structure}
24   i := 1;
25   flag := True;
26   While (flag) And (i<10) Do

```

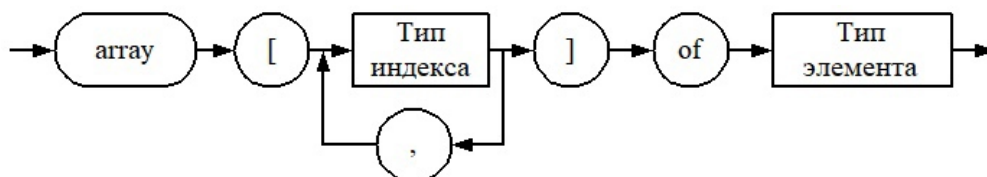
```
27   Begin
28     If a[i] = f Then
29       Begin
30         writeln(a[i], ' ');
31         flag := False;
32       End;
33     i := i + 1;
34   End;
35   If (flag) Then writeln('      ');
36 End.
```

## 7 Массивы языка Pascal. Описание, внутреннее представление, операции над массивами и их элементами. Примеры.

Иванова Г.С. Основы программирования 2002, стр. 77 - 86

Иванова Г.С. Курс Основы программирования, през. 3, с. 10 - 13

**Массив** – это упорядоченная совокупность однотипных данных. Каждому элементу массива соответствует один или несколько индексов порядкового типа, определяющих положение элемента в массиве.



**Тип индекса** – порядковый – определяет доступ к элементу. **Тип элемента** – любой кроме файла, в том числе массивы, строки и т.п. Массив в памяти не может занимать более 2 Гб.

Массивы бывают **одномерными** и **многомерными**, в зависимости от количества измерений (индексов):

1. У одномерного массива одно измерение (обращение к его элементам производится с помощью одного индекса);
2. У многомерного массива несколько измерений. Фактически это «массив массивов ...».

### Внутреннее представление массивов

В памяти элементы массива хранятся по порядку, друг за другом, с некоторым шагом. Этот шаг равен числу байт, которые необходимы на запись одного элемента массива (у всех элементов массива – один тип).

### Операции над массивами

1. Операция присваивания (только для массивов одного типа):

Пример:

```

1 Var a, b: array[boolean] of real;
2
3 ...
4
5 a:=b;

```

2. Доступ к элементу массива

```

1 Var a : array[1..10] of integer;
2
3 ...
4
5 a[i] := 1; {*1}
6 b[2] := 2; {*2}

```

{\*1} Прямой доступ: значение индекса - литерал;

{\*2} Косвенный доступ позволяет реализовать последовательную обработку элементов массивов.

3. Ввод/вывод массивов осуществляется поэлементно.

```

1
2 Var a, b: array[0..3] Of integer;
3   i: integer;
4 Begin
5   For i := 0 To 3 Do
6     readln(b[i]);
7   a := b;
8   writeln(a[0]);

```

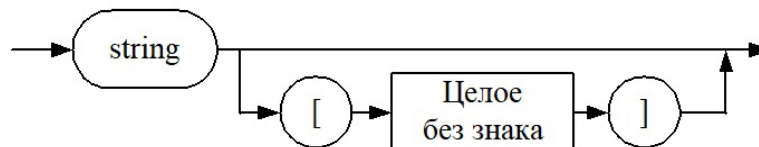
```
9  For i := 0 To 3 Do
10      writeln(a[i]);
11 End.
```

## 8 Строки языка Pascal. Описание, внутреннее представление, операции над строками и их элементами. Примеры.

Иванова Г.С. Основы программирования 2002, стр. 113 - 127

Иванова Г.С. Курс Основы программирования, през. 3, сл. 12 - 22

**Строка** – последовательность символов.



\*Целое – максимальная длина строки.

**Внутреннее представление**



### Операции над строками

1. Присваивание строк:

```
1  S1 := ABCD;
2  S1 := S2;
3  S1 := A;
4  S1 := {      }
```

2. Обращение к элементу:

S1[5] - прямое;

S1[i] - косвенное.

3. Конкатенация (сцепление) строк:

```
1  St := St + 'A';
2  St' := 'A' + 'B';
```

4. Операции отношения – выполняется попарным сравнением кодов символов, результат определяется по отношению кодов первых различных символов:

```
1  b := S1 > S2;
2  'T' < 'Ta'
```

5. Ввод-вывод строк:

```
1  ReadLn(S1);
2  WriteLn(S1);
```

### Стандартные процедуры и функции

1. Функция Length(st):word – возвращает длину строки st:

```
1  n := Length(st1);
```

2. Процедура Delete(st, index, count) – удаляет count символов строки st, начиная с символа с номером index:

```
1 S1: = dddddsssssfffff;
2 Delete(S1,6,5);
```

3. Процедура Insert(St2,St1,index) – вставляет подстроку символов St2 в строку St1, начиная с символа с номером index:

```
1 S1 = dddddddddd;
2 S2 = ;
3 Insert(S2,S1,6);
4 Insert('Pas',S1,6);
```

4. Процедура Str(x[:w[:d]],St) – преобразует результат выражения x, в строку st, содержащую запись этого числа в виде последовательности символов (как при выводе).

```
1 x:=-5.67;
2 Str(x:7:3,s1);
```

5. Процедура Val(St,x,Code) – преобразует строку St с записью числа в виде последовательности символов во внутреннее представление целого или вещественного числа и помещает его в переменную x. В целочисленной переменной Code процедура возвращает код ошибки

6. Функция Copy(St,index,count):string – возвращает фрагмент строки St, длиной count символов, начиная с символа с номером index:

```
1 S1 = qqEEEEEEEEuuuuu;
2 S:= Copy(S1,4,6);
```

7. Функция Pos(St2,St1):integer – возвращает номер позиции первого вхождения подстроки St2 в строку St1. Если вхождение не найдено, то функция возвращает 0:

```
1 S1 = qqEEEEEEEEuuuuu;
2 i:= Pos(EE,S1);
```

8. Функция UpCase(ch):char – возвращает символ, соответствующий символу верхнего регистра для ch, если таковой имеется, либо сам символ ch, если для него не определен символ верхнего регистра.

```
1
2 Var a: string[20];
3 x,code: integer;
4 Begin
5 a := 'abc';
6 writeln(a[2]);
7 a := a + 'd';
8 If 'abce' > a Then writeln('e > d')
9 Else writeln('e < d');
10 writeln(length(a));
11 delete(a, 1, 2);
12 writeln(a);
13 insert('ab', a, 1);
14 writeln(a);
15 str(1234, a);
16 writeln(a);
17 val(a, x, code);
18 writeln(x);
19 writeln(copy(a, 2, 2));
20 writeln(pos('3', a));
21 writeln(upcase('a'));
22 End.
```

## 9 Множества языка Pascal. Описание, внутреннее представление, операции над множествами и их элементами. Примеры.

Иванова Г.С. Основы программирования 2002, стр. 127 - 135

Иванова Г.С. Курс Основы программирования, през. 3, с. 23 - 28

**Множество** – неупорядоченная совокупность неповторяющихся элементов.

**Тип элементов** – порядковый, кроме Word, Integer, SmallInt, LongInt. Количество элементов не должно превышать 256.

**Конструкторы множеств** – константы множественного типа:

- [] – пустое множество;
- [2,3,5,7,11] – множество чисел;
- ['a','d','f','h'] – множество символов;
- [1,k] – множество чисел, переменная k должна содержать число;
- [2..100] – множество содержит целые числа из указанного интервала;
- [k..2\*k] – интервал можно задать выражениями;
- [red,yellow,green] – множество перечисляемого типа

**Инициализация множеств при объявлении:**

```
1 Type setnum = set of byte;
2 Var S:setnum = [1..10];
```


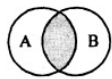
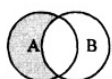
### Операции над множествами

1. Присваивание:

A:=B; A:=[];

2. Объединение, пересечение и дополнение:

- A+B ( $A \cup B$ ) – объединение множеств A и B – множество, состоящее из элементов, принадлежащих множествам A и B
- A\*B ( $A \cap B$ ) – пересечение множеств A и B – множество, состоящее из элементов, принадлежащих одновременно и множеству A и множеству B.
- A-B ( $A \setminus B$ ) – дополнение множества A до B – множество, состоящее из тех элементов множества A, которые не принадлежат множеству B.

Математическая запись	Операция Borland Pascal	Геометрическая интерпретация	Результат операции
$A \cup B$	$A + B$		Объединение множеств A и B – множество, состоящее из элементов, принадлежащих множествам A и B
$A \cap B$	$A * B$		Пересечение множеств A и B – множество, состоящее из элементов, принадлежащих одновременно и множеству A и множеству B
$A \setminus B$	$A - B$		Дополнение множеств A и B – множество, состоящее из тех элементов множества A, которые не принадлежат множеству B

3. Операции отношения:

A = B – проверка совпадения множеств A и B (если совпадают – true).

A <> B – проверка не совпадения множеств A и B (не совпадают – true).

A <= B – проверка нестрогого вхождения A в B (если входит – true).

A > B – проверка строгого вхождения B в A (если входит – true).



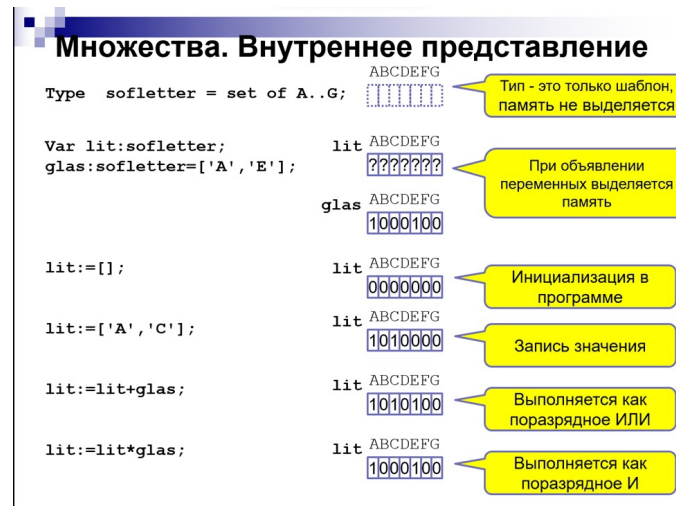
4. Проверка вхождения элемента во множество:

5. Значения множественного типа нельзя вводить и выводить!

Примеры:

```
1 [1,2]+[3,4] = [1,2,3,4];
2 [1..10]*[3,8,9,15,23,45] = [3,8,9];
3 [1..15]-[3,8,9,15,23,45] = [1,2,4..7,10..14];
4 [red,blue,green,black]*[blue,magenta,yellow] = [blue]
```

### Внутреннее представление



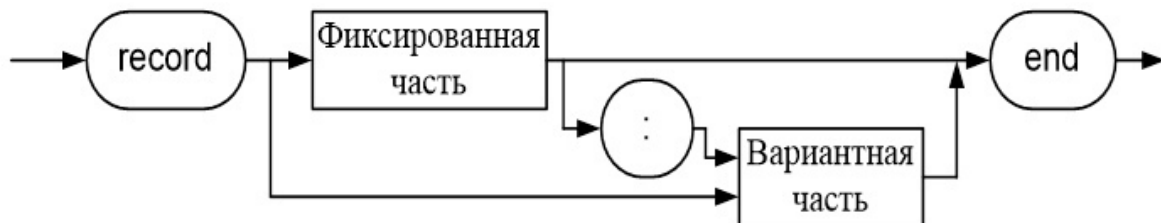
```
1
2 Var s: set Of 0..100;
3   i: integer;
4 Begin
5   s := [1,2,3,4,5];
6   For i := 0 To 100 Do
7     If i In s Then write(i, ' ');
8   writeln;
9   s := s + [3..10];
10  For i := 0 To 100 Do
11    If i In s Then write(i, ' ');
12  writeln;
13  s := s * [2..4];
14  For i := 0 To 100 Do
15    If i In s Then write(i, ' ');
16  writeln;
17  s := [1..10] - s;
18  For i := 0 To 100 Do
19    If i In s Then write(i, ' ');
20  writeln;
21  If s <> [1,5..10] Then writeln('err');
22 End.
```

## 10 Записи языка Pascal. Описание, внутреннее представление, операции над записями и их элементами. Примеры.

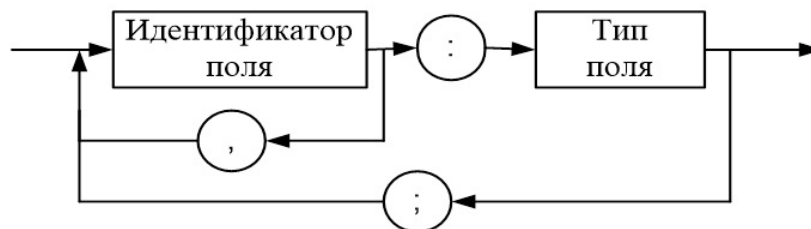
Иванова Г.С. Основы программирования 2002, стр. 136 - 143

Иванова Г.С. Курс Основы программирования, през. 3, сл. 33 - 38

**Запись** – это структура данных, образованная фиксированным числом разнотипных компонентов, называемых полями записи.



**Фиксированная часть:**



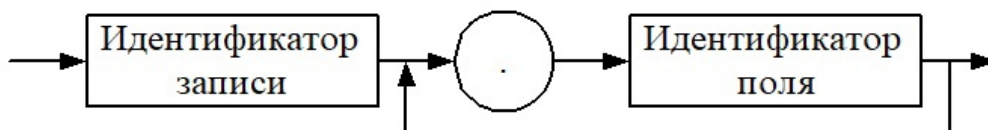
**Операции над записями:**

1. Присваивание записей одного типа:

```
1 Var A,B: record Day:1..31; Month: 1..12; Year: word; end;
2 A:=B;
```

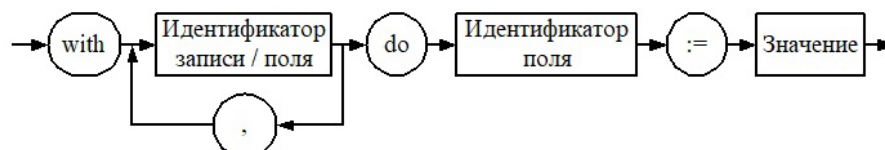
2. Доступ к полям записи:

• Точечная нотация:



```
1 A.Day := 21; {dot notation}
```

• Оператор доступа:



```
1 with A do Day := 21; {Access operator}
```

3. Ввод и вывод записей осуществляется по полям.

## Вариантная часть

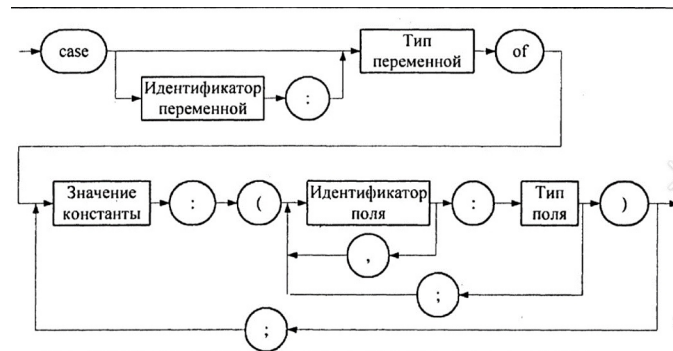


Рис. 4.37. Синтаксическая диаграмма &lt;Вариантная часть записи&gt;

- может содержать несколько альтернатив, в каждой из которых задается список полей, присущих данному варианту.

Каждой альтернативе предшествует константа, идентифицирующая соответствующий вариант.

```

1
2 Type rec = Record
3   info: integer;
4   symv: string[20];
5 End;
6
7 Var r,q: rec;
8 Begin
9   r.info := 1;
10  r.symv := 'p';
11  q := r;
12  writeln('o', q.symv);
13 End.

```

## 11 Процедуры и функции. Определение, описание, особенности. Примеры.

Иванова Г.С. Основы программирования 2002, стр. 144 - 150

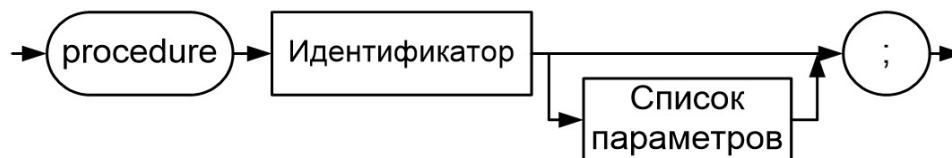
Иванова Г.С. Курс Основы программирования, през. 4, сл. 2 - 3

**Процедуры и функции** – самостоятельные фрагменты программы, соответствующим образом оформленные и вызываемые по имени (программные блоки).

Программный блок оформляется следующим образом:



**Процедура**



**Функция**



**Особенности процедур**

- могут иметь несколько результатов, или не иметь их вовсе. Результаты могут быть любого типа (осуществляется с помощью передачи параметров по ссылке).
- команда вызова процедуры – отдельная команда, которая употребляется самостоятельно.

**Особенности функций**

- имеет только один результат, тип которого указывается в объявлении функции. Этот результат возвращается функцией в точку вызова (никто не мешает передавать в функцию параметры по ссылке, по аналогии с процедурами, но тогда нет смысла использовать функцию).
- обращение к функции может использоваться как компонент выражения соответствующего типа.

```

1 Procedure iwanttoclosemycompandgoSLEEP(a:integer);
2 Begin
3   writeln('youmustprepareforexams');
4 End;
5 Function iwantoget30bofexam(a:integer): integer;
6 Begin
7   writeln('ifyoureallywantityoumastprepare');
8   iwantoget30bofexam := a;
9 End;
10 Begin
11   iwanttoclosemycompandgoSLEEP(1);
12   writeln(iwantoget30bofexam(2));
13 End.
  
```

## 12 Способы передачи данных в подпрограмму на языке Pascal. Примеры.

Иванова Г.С. Основы программирования 2002, стр. 145 - 147

Иванова Г.С. Курс Основы программирования, през. 4, сл. 5 - 9

Из основной программы данные могут быть получены:

1. **Неявно** – с использованием глобальных констант и переменных;
2. **Явно** – через параметры.

### Неявная передача данных

Каждой подпрограмме доступны все ресурсы программного блока, в разделе описаний которого эта подпрограмма объявлена. Однако в обратную сторону это не работает: локальные ресурсы, объявленные в разделе описаний подпрограммы, из программного блока, в разделе описания которого она определена, не доступны. Если имя одного из локальных ресурсов подпрограммы совпадет с именем одного из глобальных ресурсов программного блока, в разделе описания которого определена подпрограмма, то глобальный ресурс будет недоступен (он перекроется локальным).

### Недостатки неявной передачи данных:

1. жестко связывает подпрограмму и данные;
2. приводит к большому количеству ошибок.

### Передача данных через параметры

Список параметров описывается в заголовке (объявлении) подпрограммы.

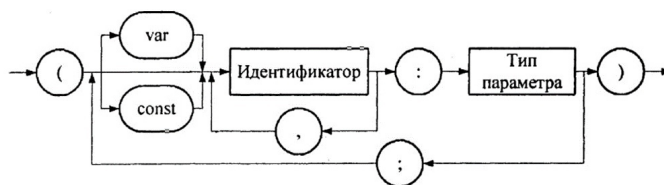


Рис. 5.4. Синтаксическая диаграмма конструкции <Список параметров>

Передача данных в параметры осуществляется тремя способами:

1. **Как значения** - в подпрограмму передаются копии значений параметров, и никакие изменения этих копий не повлияют на оригиналы (изменения не возвращаются в вызывающую подпрограмму)
2. **Как переменные** – в подпрограмму передаются адреса фактических параметров, соответственно все изменения этих параметров в подпрограмме на самом деле происходят с переменными, переданными в качестве фактических параметров; такие параметры при описании помечаются служебным словом `var`; в этом случае нельзя использовать литералы
3. **Как неизменяемые переменные (именованные константы)** – в подпрограмму, так же как и в предыдущем случае, передаются адреса фактических параметров, но при попытке изменить значение параметра компилятор выдаст сообщение об ошибке; такие параметры при описании помечаются служебным словом `const`

## 13 Локальные и глобальные переменные, законы «видимости» идентификаторов. Примеры.

*Иванова Г.С. Основы программирования 2002, стр. 145*

**Локальная переменная** - переменная, объявленная внутри какой-либо функции или процедуры. Областью видимости локальных переменных является тело функции или процедуры, в которой эта переменная объявлена. Локальная переменная может быть проинициализирована константой или выражением, соответствующими ее типу.

**Глобальная переменная** - переменная, объявленная за пределами всех функций и процедур. Областью видимости глобальных переменных является вся программа.

### Локальные и глобальные переменные

Классы переменных	Время жизни	Доступность
<b>Глобальные</b> – объявленные в основной программе	От запуска до завершения программы – все время работы программы	Из любого места программы, включая подпрограммы*
<b>Локальные</b> – объявленные в подпрограмме	От вызова подпрограммы до возврата управления – время работы подпрограммы	Из подпрограммы и подпрограмм, вызываемых из нее*

### Законы видимости идентификаторов

1. В подпрограмме можно обращаться к внешним идентификаторам, если в ней нет локального идентификатора с тем же именем.
2. Если таковой имеется, то он перекрывает внешний идентификатор
3. Однако к внешнему идентификатору можно будет обратиться из подпрограммы, если дописать к нему спереди имя содержащей его программы

```

1 Function ye(b: integer): integer;
2 Begin
3     ye := b;
4 End;
5
6 Var a: integer;
7
8 Function ne(): integer;
9 Begin
10    ne := a;
11 End;
12
13 Begin
14    a := 1;
15    writeln(ye(a));
16    writeln(ne());
17 End.
```

## 14 Формальные и фактические параметры подпрограмм языка Pascal. Примеры.

*Иванова Г.С. Основы программирования 2002, стр. 146*

Параметры, описанные в заголовке – **формальные**.

При вызове подпрограммы необходимо определить **фактические** значения этих параметров – аргументы (константы и переменные).

Формальные и фактические параметры должны соответствовать по количеству, типу и порядку.

## 15 Параметры-строки, параметры-массивы. Примеры.

Иванова Г.С. Курс Основы программирования, през. 4, сл. 14

Структурные типы параметров (массивы, строки, множества, записи, указатели, файлы) должны быть **предварительно объявлены**.

```
1
2 Type ar = array [1..112] Of integer;
3
4 Function i_not_unless(a:ar): integer;
5 Begin
6     i_not_unless := a[111];
7 End;
8
9 Var a: ar;
10    i: integer;
11 Begin
12     For i:=1 To 112 Do
13         a[i] := i * 2;
14         writeln(i_not_unless(a));
15 End.
```



## 16 Принципы разработки универсальных подпрограмм: «открытые» массивы. Примеры.

*Иванова Г.С. Основы программирования 2002, стр. 159 - 162*

*Иванова Г.С. Курс Основы программирования, през. 4, сл. 21 - 23*

**Открытый массив** – конструкция описания типа массива без указания типа индексов. Используется при объявлении формальных параметров.

Индексы открытых массивов всегда начинаются с 0;

Размер такого массива можно:

- передать через дополнительный параметр;
- получить, используя функцию High(<Идентификатор массива>).

Чтобы в программе выделить память под открытый массив, следует воспользоваться процедурой `setlength`, которая принимает два фактических параметра – имя открытого массива и устанавливаемое количество элементов в нем. В результате работы `setlength` в памяти выделяется столько байт, сколько необходимо для хранения n-го количества элементов определенного типа. Чтобы освободить выделенную память, используется `nil`.

```
1
2 Var mas: array Of integer;
3 Begin
4   setlength(mas,100);
5   writeln(high(mas));
6 End.
```

## 17 Принципы разработки универсальных подпрограмм: нетипизированные параметры, параметры процедурного типа. Примеры.

Иванова Г.С. Основы программирования 2002, стр. 162 - 168

Иванова Г.С. Курс Основы программирования, през. 4, сл. 24 - 35

**Нетипизированные параметры** – параметры-переменные, тип которых при объявлении не указан.

Для приведения нетипизированного параметра к определенному типу можно использовать:

1. автоопределенное преобразование типов:

```
1 Procedure Proc(Var a); ...  
2 ...b := Integer () + 10; ...
```

2. наложенное описание переменной определенного типа:

```
1 Procedure Proc(Var a); ...  
2 Var r:real absolute a;...
```

**Параметры процедурного типа** используются для передачи в подпрограмму имен процедур и функций. Для объявления процедурного типа используется заголовок подпрограммы, в котором отсутствует имя:

```
1 Type proc = procedure (a,b,c:real;Var d:real);  
2 func = function(x:real):real;
```

Значениями переменных процедурных типов являются идентификаторы процедур и функций с соответствующими заголовками:

```
1 Var f:func;  
2 ...  
3 f:=fun1;...
```

## 18 Структура модуля языка Pascal. Законы видимости идентификаторов. Доступ к «перекрытым» идентификаторам. Примеры.

*Иванова Г.С. Основы программирования 2002, стр. 156 - 159*

*Иванова Г.С. Курс Основы программирования, през. 4, сл. 16 - 20*

**Модуль** – это автономно компилируемая коллекция программных ресурсов, предназначенных для использования другими модулями и программами.

**Ресурсы** – переменные, константы, описания типов и подпрограммы.

Все ресурсы, определенные в модуле, делят на:

1. внешние – предназначенные для использования другими программами и модулями.
2. внутренние – предназначенные для использования внутри модуля.

Структура модуля:

**Unit** <Имя модуля (= имя файла, в котором он описан)>;

**Interface**

<Интерфейсная секция>

**Implementation**

<Секция реализации>

**[Initialization**

<Секция инициализации>

**[Finalization**

<Секция завершения>]]

**End.**

**Правило видимости имен ресурсов модуля**

- Ресурсы модуля перекрываются ресурсами программы и ранее указанными модулями.
- Для доступа к перекрытым ресурсам модуля используют точечную нотацию:  
<Имя модуля>.<Имя ресурса>

```
1
2 Unit MathFx;
3
4 Interface
5
6 Type f = Function (x : real) : real;
7 Function P(f1, f2, f3 : f; x : real) : real;
8
9
10 Implementation
11 Function P(f1, f2, f3 : f; x : real) : real;
12 Begin
13     result := f1(x)*sqr(x) - f2(x)*x + f3(x);
14 End;
15 End.
```

## 19 Рекурсия. Виды рекурсии. Особенности программирования. Достоинства и недостатки. Пример.

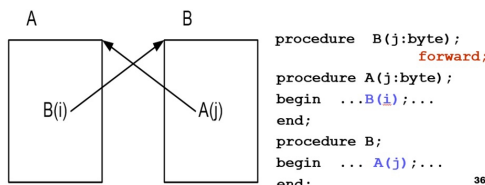
Иванова Г.С. Основы программирования 2002, стр. 168 - 179

Иванова Г.С. Курс Основы программирования, през. 4, с. 36 - 53

**Рекурсия** – организация вычислений, при которой процедура или функция обращаются к самим себе.

Различают **явную** и **косвенную** рекурсии. При **явной** – в теле подпрограммы существует вызов самой себя, при **косвенной** – вызов осуществляется в подпрограммах, вызываемых из рассматриваемой.

Косвенная рекурсия требует предопределения forward:



### Достоинства

1. Наглядное, понятное и компактное решение задачи.
2. Облегчает отладку.
3. Неизменяемость переменных.

### Недостатки

1. Потребление памяти.
2. Потребление процессорного времени.

### Замечания:

1. Рекурсию предпочитают итерации, если рекурсия естественно отражает задачу и её результаты, т.е. когда рекурсивный подход нагляден.
2. Если требуется повысить эффективность программы, то следует избегать использования рекурсий.

```

1 Procedure rec(n, k:integer);
2 Begin
3     If n <> 1 Then rec(n Div 2, k + 1)
4     Else writeln(k);
5 End;
6 Begin
7     rec(1024, 0);
8 End.

```

```

1 Procedure rec2(a,b:integer);
2 forward;
3 Procedure rec1(a,b:integer);
4 Begin
5     If a > b Then rec2(a,b)
6     Else rec2(b,a);
7 End;
8 Procedure rec2(a,b:integer);
9 Begin
10    If a = b Then writeln(a)
11    Else rec1(a - b, b);
12 End;
13 Begin
14    rec1(24,16);
15 End.

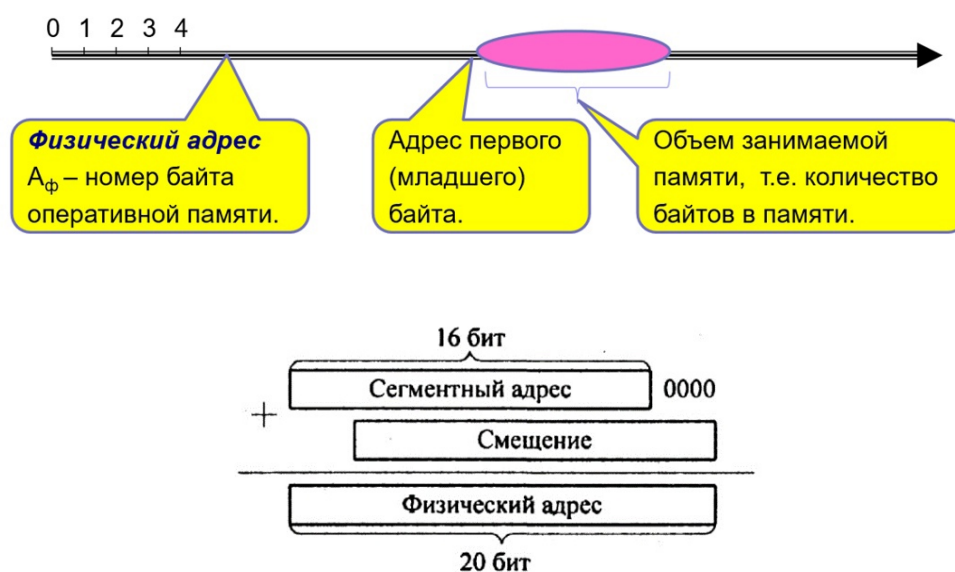
```

## 20 Адресация динамической памяти: понятие адреса, операции получения адреса и разыменования. Процедуры получения памяти и освобождения ее. Примеры.

Иванова Г.С. Основы программирования 2002, стр. 212 - 222

Иванова Г.С. Курс Основы программирования, през. 6, сл. 2 - 13

Минимальная адресуемая единица памяти большинства современных процессоров – **байт**. Байты памяти нумеруют, начиная с нуля. Непрерывный участок памяти, имеющий длину не более 64 КБ и начинающийся с адреса, кратного 16 (0,16,32, ...), называют **сегментом**. Адрес начала сегмента принимают за базу для всего сегмента. Адрес базы сегмента без последних четырех бит называют **сегментным**. Сегментный адрес и смещение имеют размер по 16 бит (слово). Физический адрес, получаемый при их сложении с учетом отброшенных четырех бит (рис. 7.2), имеет размер 20 бит и может адресовать память объемом  $2^{20}$  байт или 1 МБ.

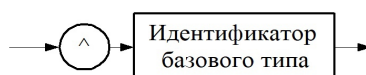


Для хранения и работы с адресами переменных используются **указатели**. Данные этого типа включают два поля word и хранят соответственно сегментный адрес и смещение.

Различают указатели:

- **типизированные** – адресующие данные конкретного типа;
- **нетипизированные** – не связанные с данными определенного типа.

Объявление типизированного указателя:



Объявление нетипизированного указателя: pointer

Взятие адреса

```
Data := Addr (NodeNumber) ; <-> Data := @NodeNumber ;
```

Разыменовывание:  $\text{Data}^{\wedge} = \langle \text{значение} \rangle$ ;

Управление выделением и освобождением памяти осуществляется посредством специальных процедур и функций:

1. Процедура **New(var P: <тип>)** – выделяет память для размещения переменной, размер определяется типом указателя.
2. Процедура **Dispose(var P: <тип>)** – освобождает выделенную память.
3. Процедура **GetMem(var P: Pointer; Size: Integer)** – выделяет указанное количество памяти и помещает ее адрес в указатель.
4. Процедура **FreeMem(var P: Pointer; Size: Integer)** – освобождает выделенную память.
5. Функция **SizeOf(X): Integer** – возвращает размер переменной в байтах.

```
1
2 Type m = ^point;
3   point = integer;
4
5 Var i: integer;
6   p: ^integer;
7   q: m;
8 Begin
9   p := @i;
10  new(q);
11  dispose(q);
12 End.
```

## 21 Списковые структуры данных. Классификация и основные приемы работы с ними: создание элемента, добавление элемента к списку, удаление элемента из списка. Область применения списковых структур данных. Пример.

*Иванова Г.С. Основы программирования 2002, стр. 223 - 237*

*Иванова Г.С. Курс Основы программирования, през. 6, с. 14 - 30*

**Список** – способ организации данных, предполагающий использование указателей для определения следующего элемента.

Элемент списка состоит из двух частей: **информационной** и **адресной**:

- **Информационная часть** содержит поля данных.
- **Адресная** – включает от одного до  $n$  указателей, содержащих адреса следующих элементов.

Количество связей, между соседними элементами списка определяет его **связность**: односвязные, двусвязные,  $n$ -связные.

**Основные приемы работы (на примере односвязного списка):** Подробнее на страницах 226 - 237 учебника.

- **Создание элемента:** для каждого элемента память выделяется динамически (процедура `new`).
- **Добавление элемента к списку:**
  1. Находим место вставки;
  2. Создаем новый элемент и в его адресное поле записываем указатель из адресного поля элемента, предшествующего вставке;
  3. В адресное поле предшествующего элемента записываем указатель на новый элемент;
- **Удаление элемента из списка:**
  1. Находим место удаления;
  2. Сохраняем адрес следующего элемента;
  3. Освобождаем память;
  4. Заносим в указатель предшествующего элемента сохраненный адрес.

```

1
2 Type list = ^Node;
3   Node = Record
4     Data : char;
5     pNext : list;
6   End;
7
8 Procedure append(Var sp :list; Const Data : char);
9 Begin
10  If sp = Nil Then
11    Begin
12      new(sp);
13      sp^.Data := Data;
14      sp^.pNext := Nil;
15    End
16  Else append(sp^.pNext, Data);
17 End;
18
19 Procedure append0(Var sp: list; Const Data : char);
20
21 Var new_elem : list;
22 Begin

```

```
23  If sp = Nil Then
24      Begin
25          new(sp);
26          sp^.Data := Data;
27          sp^.pNext := Nil;
28      End
29  Else
30      Begin
31          new(new_elem);
32          new_elem^.Data := Data;
33          new_elem^.pNext := sp;
34          sp := new_elem;
35      End;
36  End;
37
38  Procedure init(Var sp :list; Const size : integer);
39
40  Var i : integer;
41  Begin
42      randomize;
43      For i := 1 To size Do
44          append(sp, char(random(26) + 65));
45      End;
46
47  Procedure show(Var sp :list);
48  Begin
49      If sp = Nil Then writeln('Your sp is empty!')
50      Else
51          If sp^.pNext <> Nil Then
52              Begin
53                  write(sp^.Data, ' ');
54                  show(sp^.pNext);
55              End
56          Else writeln(sp^.Data);
57      End;
58
59  Procedure delete(Var sp :list);
60  Begin
61      If sp <> Nil Then
62          Begin
63              delete(sp^.pNext);
64              dispose(sp);
65              sp := Nil;
66          End;
67      End;
68
69  Function size(sp :list) : integer;
70
71  Var i : integer;
72  Begin
73      i := 1;
74      If sp = Nil Then size := 0
75      Else
76          Begin
77              i := 1;
78              While sp^.pNext <> Nil Do
79                  Begin
80                      inc(i);
81                      sp := sp^.pNext;
82                  End;
83              size := i;
```



```

84     End;
85 End;
86
87 Procedure insert(Var sp: list; Const Data :char; Const index : integer);
88
89 Var i : integer;
90     home, new_elem : list;
91 Begin
92     home := sp;
93     If (index <= 1) Then
94         append0(sp, Data)
95     Else
96         If (index >= size(sp)) Then append(sp, Data)
97     Else
98         Begin
99             For i := 1 To index - 2 Do
100                 Begin
101                     sp := sp^.pNext;
102                 End;
103                 new(new_elem);
104                 new_elem^.Data := Data;
105                 new_elem^.pNext := sp^.pNext;
106                 sp^.pNext := new_elem;
107                 sp := home;
108             End;
109 End;
110
111 Procedure pop0(Var sp :list);
112
113 Var del : list;
114 Begin
115     del := sp;
116     sp := sp^.pNext;
117     dispose(del);
118     del := Nil;
119 End;
120
121 Procedure pop(Var sp :list);
122
123 Var del, home : list;
124 Begin
125     home := sp;
126     While (sp^.pNext)^.pNext <> Nil Do
127         sp := sp^.pNext;
128     del := sp^.pNext;
129     sp^.pNext := del^.pNext;
130     dispose(del);
131     del := Nil;
132     sp := home;
133 End;
134
135 Procedure popI(Var sp :list; Const index :integer);
136
137 Var i : integer;
138     home, del : list;
139 Begin
140     If (index <= 1) Then pop0(sp)
141     Else If (index >= size(sp)) Then pop(sp)
142     Else
143         Begin
144             home := sp;

```

```
145     For i := 1 To index - 2 Do
146         sp := sp^.pNext;
147     del := sp^.pNext;
148     sp^.pNext := del^.pNext;
149     dispose(del);
150     del := Nil;
151     sp := home;
152 End;
153 End;
154
155 Var sp : list;
156 Begin
157     init(sp, 10);
158     write('size = ', size(sp), '; init: ');
159     show(sp);
160     writeln();
161
162     Append0(sp, '!');
163     write('size = ', size(sp), '; Append0: ');
164     show(sp);
165     writeln();
166
167     Append(sp, '!');
168     write('size = ', size(sp), '; Append: ');
169     show(sp);
170     writeln();
171
172     insert(sp, '!', 4);
173     write('size = ', size(sp), '; insert: ');
174     show(sp);
175     writeln();
176
177     pop0(sp);
178     write('size = ', size(sp), '; pop0: ');
179     show(sp);
180     writeln();
181
182     pop(sp);
183     write('size = ', size(sp), '; pop: ');
184     show(sp);
185     writeln();
186
187     popI(sp, 3);
188     write('size = ', size(sp), '; popI: ');
189     show(sp);
190     writeln();
191
192
193     delete(sp);
194     write('size = ', size(sp), '; delete: ');
195     show(sp);
196     writeln();
197     readln;
198 End.
```

## 22 Основы файловой системы: файл, каталог, полное имя файла, внутреннее представление информации в файле. Файловая переменная. Операции открытия и закрытия файлов. Примеры.

Иванова Г.С. Основы программирования 2002, стр. 188 - 196

Иванова Г.С. Курс Основы программирования, през. 5, сл. 2 - 15

**Файл** – поименованная последовательность элементов данных (компонентов файла), хранящихся, как правило, во внешней памяти.

Как исключение данные файла могут не храниться, а вводиться с внешних устройств (ВУ), например клавиатуры или выводиться на ВУ, например экран.

Полное имя файла включает:

<Имя диска>:<Список имен каталогов><Имя файла>.<Расширение>

Имя файла в Windows составляют из строчных и прописных букв латинского и русского алфавитов, арабских цифр и некоторых специальных символов, например, символов подчеркивания «\_» или доллара «\$»

Расширение определяет тип хранящихся данных, например:

- **COM, EXE** – исполняемые файлы (программы);
- **PAS, BAS, CPP** – исходные тексты программ на алгоритмических языках ПАСКАЛЬ, БЭЙСИК и C++;
- **BMP, JPG, PIC** – графические файлы (рисунки, фотографии);
- **WAV, MP3, WMA** – музыкальные файлы.

**внутреннее представление информации в файле**

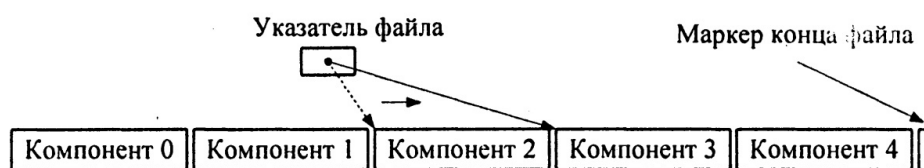


Рис. 6.2. Организация файла

**Описание файловых переменных:**

1. **Типизированные файлы:** file of <Тип компонента>, где <Тип компонента> – любой тип данных, кроме файлового.
2. **Текстовые файлы:** text
3. **Нетипизированные файлы:** file

Процедура Assign или AssignFile (Var f; st:string) – связывает файловую переменную f с файлом, имя которого указано в строке st. Если файл находится в текущем каталоге, то достаточно задать имя файла и его расширение. В противном случае необходимо указать полное имя файла.

**Операция открытия файла**

При открытии файла необходимо задать направление передачи данных: запись или чтение. Кроме того текстовый файл можно открыть для добавления компонентов.

1. Процедура **ReSet(Var f)** – открывает файл для чтения данных. Устанавливает указатель файла на первый компонент. Если файл не существует, выдается сообщение об ошибке.
2. Процедура **ReWrite(Var f)** – открывает файл для записи. Если указанный файл существовал, то он уничтожается без выдачи предупреждения пользователю, иначе он создается и указатель устанавливается на начало.
3. Процедура **AppEnd(Var f:text)** – открывает текстовый файл для добавления данных. Указатель файла устанавливается на конец файла.

**Контроль операций ввода-вывода**

Функция `IOResult:Word` – возвращает код завершения операции ввода-вывода: 0 – если операция прошла нормально, код ошибки, если нет. Функция применяется при отключенном контроле операций ввода-вывода `{SI-}`.

**Операция закрытия файла**

Процедура `Close` или `CloseFile(Var f)` - выполняет закрытие файла. При этом вновь созданный файл регистрируется в каталоге. Процедура закрытия файла обеспечивает вывод оставшихся компонентов из буфера в файл. Связь файловой переменной с файлом при закрытии сохраняется, поэтому при продолжении обработки повторно процедуру `AssignFile()` можно не выполнять.

\*По способу доступа к компонентам файлы делятся на файлы прямого доступа и файлы последовательного доступа.

- Файлы последовательного доступа считываются от первой записи до последней по порядку;
- Файлы прямого доступа могут считывать определенную запись по номеру.

```
1
2 Var F,G,H: text;
3
4 Begin
5     assign(F,'F');
6     assign(G,'G');
7     assign(H,'H');
8     reset(F);
9     rewrite(G);
10    append(H);
11    close(H);
12    close(F);
13    close(G);
14 End.
```

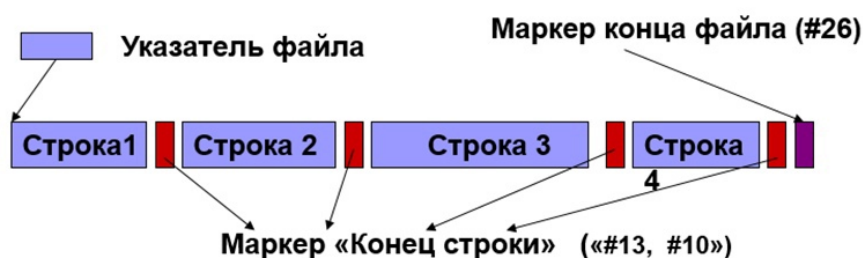
## 23 Текстовые файлы. Внутреннее представление информации в файле. Операции над файлами. Пример.

Иванова Г.С. Основы программирования 2002, стр. 196 - 201

Иванова Г.С. Курс Основы программирования, през. 5, с. 16 - 25

**Текстовый файл** – файл, компонентами которого являются символьные строки переменной длины, заканчивающиеся специальным маркером – маркером «Конец строки».

**Внутреннее представление информации в файле:**



**Операции над текстовыми файлами:**

- Текстовые файлы используют для хранения и обработки символов, строк, символьных массивов. Числовые и логические данные при записи в текстовые файлы должны преобразовываться в символьные строки.
- Текстовый файл можно открыть для записи, чтения и добавления записей в конец. Файл, открытый для записи, не может использоваться для чтения и наоборот.

**Процедуры и функции обработки текстовых файлов:**

1. Функция `EOLn([Var f])`: Boolean – возвращает TRUE, если во входном текстовом файле достигнут маркер конца строки; при отсутствии файловой переменной проверяется файл INPUT, связанный с клавиатурой.
  - При работе с клавиатурой функция `EOLn` возвращает TRUE, если последним считанным был символ #13.
  - При работе с диском функция `EOLn` возвращает TRUE, если следующим считанным будет символ #13.
2. Процедура `Read([Var f;text;]v1,v2,...vn)` – обеспечивает ввод символов, строк и чисел. При вводе чисел пробелы и символы табуляции игнорируются. Если файловая переменная не указана, то ввод осуществляется из файла INPUT.
3. Процедура `ReadLn([Var f;][v1,v2,...,vn])` – осуществляет ввод символов, строк и чисел. После чтения последней переменной оставшаяся часть строки до маркера конца строки пропускается так, что следующее обращение к `ReadLn` или `Read` начинается с первого символа новой строки.
4. Процедура `Write([Var f;]v1,v2, ...,vn)` – осуществляет вывод одного или более выражений типа CHAR, STRING, BOOLEAN, а также целого или вещественного типов. При выводе числовых значений последние преобразуются в символьное представление. Если файловая переменная не указана, то вывод осуществляется в файл OUTPUT.
5. Процедура `WriteLn([Var f;][v1,v2, ...,vn])` – осуществляет вывод в текстовый файл. Если файловая переменная не указана, то вывод осуществляется в файл OUTPUT.  
Выводимая строка символов завершается маркером конца строки. Если список вывода не указан, то в файл передается только маркер конца строки.
6. Функция `SeekEOLn([Var f]):boolean` – пропускает пробелы и знаки табуляции до маркера конца строки или до первого значащего символа и возвращает TRUE, при обнаружении маркера. Если файловая переменная не указана, то функция проверяет файл INPUT.
7. Функция `SeekEOF([Var f]):boolean` – пропускает все пробелы, знаки табуляции и маркеры конца строки до маркера конца файла или до первого значащего символа и возвращает TRUE при обнаружении маркера. Если файловая переменная отсутствует, то функция проверяет файл INPUT.

```
1
2 Var Fmas, Gmas: array Of Integer;
3   F,G: text;
4   Fi,Gi,Fk,Gk: integer;
5   fl: boolean;
6
7 Begin
8   assign(F,'F');
9   assign(G,'G');
10  assign(H,'H');
11  reset(F);
12  reset(G);
13  While Not eof(F) Do
14    Begin
15      readln(F);
16      Fk := Fk + 1;
17    End;
18  While Not eof(G) Do
19    Begin
20      readln(G);
21      Gk := Gk + 1;
22    End;
23  setlength(Fmas,Fk);
24  setlength(Gmas,Gk);
25  reset(F);
26  reset(G);
27  writeln('          F');
28  For Fi := 0 To Fk - 1 Do
29    Begin
30      readln(F,Fmas[Fi]);
31      write(Fmas[Fi], ' ');
32    End;
33  writeln();
34  writeln('          G');
35  For Gi := 0 To Gk - 1 Do
36    Begin
37      readln(G,Gmas[Gi]);
38      write(Gmas[Gi], ' ');
39    End;
40  writeln();
41  close(F);
42  close(G);
43 End.
```

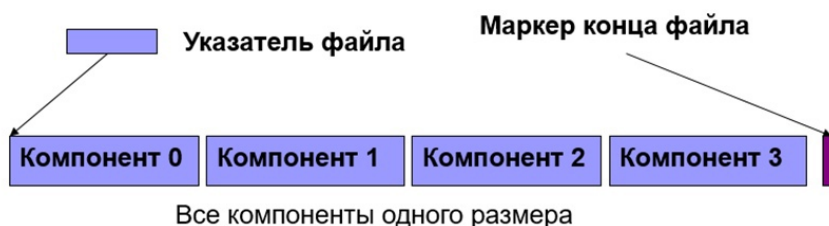
## 24 Типизированные файлы: внутреннее представление информации в файле. Операции над файлами. Пример.

Иванова Г.С. Основы программирования 2002, стр. 201 - 207

Иванова Г.С. Курс Основы программирования, през. 5, с. 26 - 43

**Типизированный файл** – файл, все компоненты которого одного типа, заданного при объявлении файловой переменной. Компоненты хранятся на диске во внутреннем (двоичном) формате. При этом числа хранятся в формате, в котором над ними выполняются операции.

**Внутреннее представление информации в файле:**



\*Типизированные файлы могут быть созданы только программным путем.

Типизированный файл можно открыть для записи и чтения. Файл, открытый для записи, может использоваться для чтения. В файл, открытый для чтения, можно писать. Поскольку размер компонентов одинаков, принципиально возможен не только последовательный, но и **прямой доступ** (о типах доступа к компонентам файла подробнее в п.22).

**Процедуры и функции обработки типизированных файлов**

1. Процедура **Read(Var f; c1,c2,...,cn)** – осуществляет чтение компонентов типизированного файла. Список ввода содержит одну или несколько переменных того же типа, что и компоненты файла. Если файл исчерпан, обращение к процедуре вызывает ошибку ввода-вывода.
2. Процедура **Write(Var f; c1,c2,...,cn)** – осуществляет запись компонентов в типизированный файл. Список вывода содержит одно или более выражений того же типа, что и компоненты файла.
3. Процедура **Seek(Var f; numcomp:longint)** – осуществляет установку указателя файла на компонент с номером numcomp.
4. Функция **FileSize(Var f):longint** – возвращает количество компонентов файла. Может использоваться для установки на конец файла совместно с Seek():(f);
5. Функция **FilePos(Var f):longint** – возвращает порядковый номер компонента, который будет обрабатываться следующим.
6. Процедура **Truncate(Var f)** – выполняет «усечение» файла

## 25 Нетипизированные файлы. Внутреннее представление информации в файле. Операции над файлами. Пример.

Иванова Г.С. Основы программирования 2002, стр. 207 - 211

Иванова Г.С. Курс Основы программирования, през. 5, сл. 44 - 49

**Нетипизированными** называют файлы, объявленные без указания типа компонентов.

\*Нетипизированные файлы могут быть созданы только программным путем.

Операции чтения и записи с такими файлами осуществляются блоками, что позволяет организовать высокоскоростной обмен данными между диском и памятью. Отсутствие типа делает эти файлы совместимыми с любыми другими, однако обрабатывать такие файлы существенно сложнее.

Нетипизированные файлы, как и типизированные, допускают организацию **прямого доступа**, но к записям.

Нетипизированный файл можно открыть для записи и для чтения:

- ReSet(Var f:[reclsize:word]);
- ReWrite(Var f:[reclsize:word]);

, где reclsize – размер записи файла в байтах. Длину записи задают кратной 512 байт, например: 1024, 2048. Если длина записи не указана, она принимается равной 128.

Процедуры и функции обработки нетипизированных файлов:

1. Процедура BlockRead(Var f:file; Var buf;Count:word[;Var res:word])– осуществляет чтение блока записей из файла в буфер buf. Параметр res будет содержать количество фактически обработанных записей. Если последняя запись – неполная, то значение параметра res ее не учтет.
2. Процедура BlockWrite(Var f:file;Var buf;Count:word[;Var res:word])– осуществляет запись блока из буфера buf в файл.

```
1
2 Var f: file;
3   buf: integer;
4 Begin
5   assign(f,'f.bin');
6   reset(f);
7   BlockRead(f, buf, 2);
8   rewrite(f);
9   BlockWrite(f,buf, 2);
10 End.
```



## 26 Классы консольного режима среды Lazarus: описание классов, поля и методы, объявление объектов класса, доступ к полям и методам объекта, ограничение доступа. Пример.

*Иванова Г.С. Основы программирования 2002, стр. 303 - 321*

*Иванова Г.С. Курс Основы программирования, през. 7, сл. 17 - 25*

**Класс** – это структурный тип данных, который включает описание **полей данных**, а также процедур и функций, работающих с этими полями данных. Применительно к классам такие процедуры и функции получили название **методов**.

– это шаблон кода, по которому создается какой-то объект

– это тип данных, задающий реализацию некоторой абстракции данных (сущности), характерной для моделируемой предметной области.

### Описание классов

```
Type <имя класса> = object
    <описание полей класса>
    <прототипы методов>
end; ...
```

```
Procedure <имя класса>.<имя метода>;
    <локальные ресурсы процедуры>
Begin
    <тело процедуры>
End; ...
```

**Объект-переменная или объект класса** – переменная типа «класс». Описав класс, мы можем объявить любое количество объектов этого класса, причем можно объявить отдельные объекты, массивы объектов и указатели на объекты данного класса.

```
1 type class_identifier = object; {NOT CLASS!!!}
2   private
3     field1 : field_type1;
4     field2 : field_type2;
5     ...
6
7   public
8     constructor create(...);
9     procedure proc1;
10    function f1(): function_type;
11 end;
12
13 constructor class_identifier.create(...);
14 ...
15 procedure class_identifier.proc1;
16 ...
17
18 var classvar : class_identifier;
```

### модификаторы доступа к полям (основные):

- **private** – доступ только в контексте класса.
- **protected** – доступ только в контексте класса и классов-наследников.
- **Public** – доступ есть всегда, когда доступен сам класс.

## 27 Классы консольного режима среды Lazarus: Способы инициализации полей. Неявный параметр Self. Пример.

*Иванова Г.С. Основы программирования 2002, стр. 314 - 319*

*Иванова Г.С. Курс Основы программирования, през. 7, сл. 18, 20 - 22*

### Способы инициализация полей:

1. Инициализация полей прямой записью в поле:

```
1 A.length:=3.5;  
2 A.width:=5.1;
```

2. Инициализация при объявлении объекта:

```
1 Var A:TRoom = (length:3.5; width:5.1);
```

3. Инициализация посредством метода:

```
1 procedure Init(l,w:single);  
2 Procedure TRoom.Init;  
3 Begin length:=l; width:=w; End;
```

Любой метод (при его вызове) неявно получает **параметр Self** – ссылку (адрес) на поля объекта (вызывающего этот метод), и обращение к полям происходит через это имя.

## 28 Процедурная и объектная декомпозиция. Диаграммы классов. Отношения между классами. Примеры.

Иванова Г.С. Основы программирования 2002, стр. 303 - 312

Иванова Г.С. Курс Основы программирования, през. 7, сл. 26 - 55

**Процедурная декомпозиция** - структурное программирование: разбиение программы на подпрограммы, решающие отдельные подзадачи.

**Объектная декомпозиция** - в предметной области задачи выделяют отдельные элементы, поведение которых программно моделируется, а затем, уже из готовых объектов собирается сложная программа.

**Диаграмма классов** (англ. **class diagram**) — диаграмма, демонстрирующая общую структуру иерархии классов системы, их коопераций, атрибутов (полей), методов, интерфейсов и взаимосвязей между ними.

Описание класса на диаграмме:

Имя класса
Атрибуты (состояние)
Методы

**Отношения между классами:**

- **Наследование** – механизм конструирования новых более сложных производных классов из уже имеющихся базовых посредством добавления полей и методов.

Описание наследования на диаграмме:

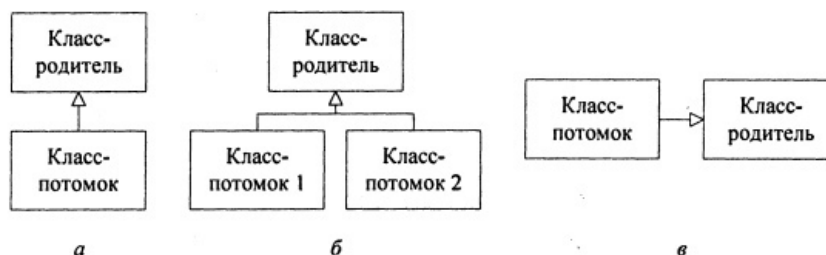


Рис. 9.3. Примеры иерархий классов:

*a* – с одним потомком; *б* – с двумя потомками; *в* – с нестандартным расположением классов

- **Композиция** – включение объектов одного класса в объекты другого. Реализуется механизмом поддержки объектных полей.

Описание композиции на диаграмме:

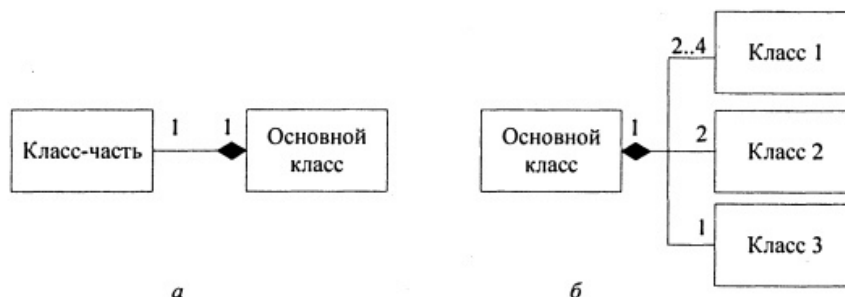


Рис. 9.4. Примеры диаграмм классов, изображающих композицию:

*a* – с одним объектным полем; *б* – с несколькими объектными полями различных типов

- **Наполнением или агрегацией** называют такое отношение между классами, при котором точное количество объектов одного класса, включаемых в другой класс, не ограничено и может меняться от 0 до достаточно больших значений. Физически наполнение реализуется с использованием указателей на объекты.

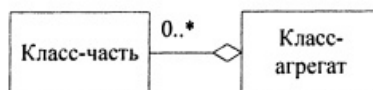


Рис. 9.5. Пример диаграммы классов, изображающей агрегацию или наполнение

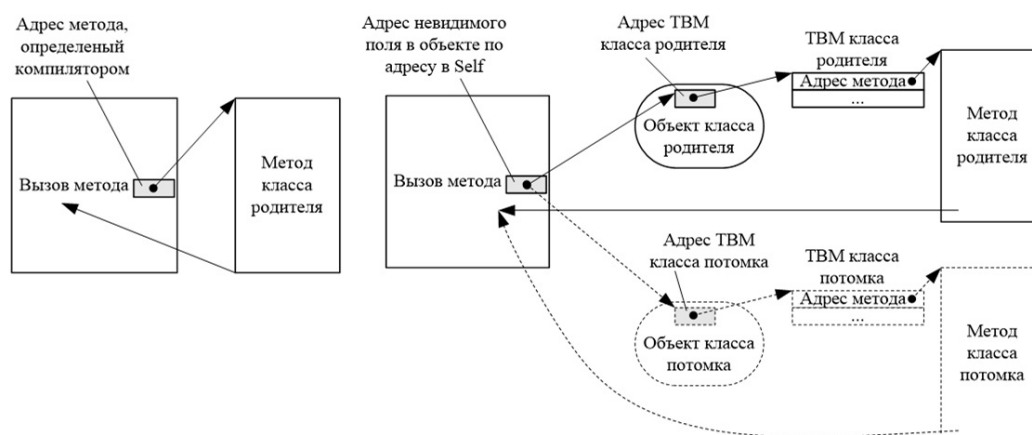
- **Простой (статический) полиморфизм** – механизм переопределения методов при наследовании, при котором связь метода с объектом выполняется на этапе компиляции (раннее связывание).

\*Раннее связывание – адрес метода определяется на этапе компиляции по объявленному типу переменной.

- **Сложный полиморфизм** – механизм, используемый, если при вызове переопределенного метода необходимо уточнить, какой метод должен быть подключен: метод родителя или метод потомка, так как объект, для которого вызывается переопределенный метод, может быть как объектом класса родителя, так и объектом класса потомка. В этом случае нужный метод определяется на этапе выполнения программы (позднее связывание), когда тип объекта точно известен.

\*Позднее связывание – адрес метода определяется на этапе выполнения по фактическому типу объекта через таблицу виртуальных методов класса, адрес которой хранится в объекте.

#### Различие раннего и позднего связывания



#### Для организации сложного полиморфизма необходимо:

1. переопределяемые методы описать служебным словом **virtual**;
2. к методам класса с виртуальными полиморфными методами добавить специальный метод-процедуру – **конструктор**, в котором служебное слово `procedure` заменено служебным словом `constructor`;
3. вызвать конструктор прежде, чем произойдет первое обращение к виртуальным полиморфным методам. Подключение осуществляется с использованием таблицы виртуальных методов (ТБМ), которая создается при выполнении конструктора.

#### 3 случая обязательного использования сложного полиморфизма:

1. Если наследуемый метод для объекта производного класса вызывает метод, переопределенный в производном классе.
2. Если объект производного класса через указатель базового класса обращается к методу, переопределенному производным классом.
3. Если процедура вызывает переопределенный метод для объекта производного класса, переданного в процедуру через параметр-переменную, описанный как объект базового класса («процедура с полиморфным объектом»).

**Свойства виртуальных методов класса:**

1. позднее связывание требует построения ТВМ, а следовательно больше памяти;
2. вызов виртуальных полиморфных методов происходит через ТВМ, а следовательно медленнее;
3. список параметров одноименных виртуальных полиморфных методов должен совпадать, а статических полиморфных – не обязательно;
4. статический полиморфный метод не может переопределить виртуальный полиморфный метод.

## 29 Динамические объекты и объекты с динамическими полями в консольном режиме среды Lazarus. Примеры.

*Иванова Г.С. Основы программирования 2002, стр. 348 - 353*

*Иванова Г.С. Курс Основы программирования, през. 7, сл. 56 - 62*

### **Создание полиморфных объектов:**

Функция `New(<Тип указателя>[, <Вызов конструктора>])` – возвращает адрес размещенного и, при вызове конструктора, сконструированного объекта. Если вызов конструктора в `New` отсутствует, то после выделения памяти необходим вызов конструктора.

**Деструктор** – метод класса, который используется для корректного уничтожения полиморфного объекта, содержащего невидимое поле. Деструктор можно использовать для освобождения памяти динамических полей и переопределять при наследовании.

### **Уничтожение полиморфных объектов:**

Процедура `Dispose(<Указатель>[, <Вызов деструктора>])` – если есть вызов деструктора, то устанавливается размер полиморфного объекта и корректно освобождается память

**Динамические поля** в статических и динамических полиморфных объектах:

Если при разработке классов для реализации полиморфных объектов используют динамические поля (объектные или нет), то запрос памяти под них обычно помещают в конструктор, а освобождение памяти - в деструктор. Тогда при уничтожении объекта автоматически освобождается память, отведенная для его полей.

## 30 Технология событийного программирования. События операционной системы, сообщения и события Lazarus. Основные события Lazarus. Примеры.

*Иванова Г.С. Основы программирования 2002, стр. 391 - 412*

*Иванова Г.С. Курс Основы программирования, през. 8, с. 2 - 18*

**Событийно-ориентированное программирование (СОП)** – парадигма программирования, в которой выполнение программы определяется событиями – действиями пользователя (мышь, клавиатура, сенсорный экран и тд), сообщениями других программ и потоков, событиями операционной системы.

**События операционной системы** – любые изменения состояния ОС: запуск системы (ПК), вход в систему любого пользователя, любые запускаемые процессы и тд.

### **Порядок обработки событий ОС:**

1. В системе происходит событие, например, пользователь передвинул мышь или нажал на клавишу клавиатуры, в результате генерируется сообщение об этом событии – сообщение ОС.
2. Сообщение ОС диспетчируется конкретному приложению.
3. В приложении сообщение передается активному компоненту (окну или управляющему элементу).
4. Метод обработки сообщения ОС компонента инициирует заранее предусмотренные события среды.
5. Если в приложении предусмотрен соответствующий обработчик события среды, то он вызывается, если нет – то продолжается обработка сообщения ОС.

### **Основные события Lazarus**

1. При изменении состояния формы:
  - OnCreate – в начальной стадии создания формы - используется при необходимости задания параметров (цвет или размер);
  - OnActivate – при получении формой фокуса ввода (окно становится активным и ему адресуется весь ввод с клавиатуры);
  - OnShow – когда форма (окно) становится видимой;
  - OnPaint – при необходимости нарисовать или перерисовать форму;
  - OnResize – при изменении размеров формы на экране;
  - OnDeactivate – при потере формой фокуса ввода (окно становится неактивным);
  - OnHide – при удалении формы с экрана (окно становится невидимым);
  - OnCloseQuery – при попытке закрыть форму - обычно используется для создания запроса-подтверждения необходимости закрытия окна;
  - OnClose – при закрытии формы;
  - OnDestroy – при уничтожении формы;
2. От клавиатуры и мыши:
  - OnKeyPressed – при нажатии клавиш, которым соответствует код ANSI;
  - OnKeyDown, OnKeyUp – при нажатии и отпускании любых клавиш;
  - OnClick, OnDbClick – при обычном и двойном нажатии клавиш мыши;
  - OnMouseMove – при перемещении мыши (многократно);
  - OnMouseDown, OnMouseUp – при нажатии и отпускании клавиш мыши;
3. При перетаскивании объекта мышью:
  - OnDragDrop – в момент опускания объекта на форму;
  - OnDragOver – в процессе перетаскивания объекта над формой (многократно);
4. Другие:
  - OnHelp – при вызове подсказки;
  - OnChange – при изменении содержимого компонент.

**Вот и кончился наш прекрасный файл! Спасибо, что прочитали его.**

**Да прибудет с вами полиморфизм и стек!**

**Хорошего зимнего настроения и отличных оценок)**

*Риторический терзает душу мне вопрос*

*Как пропылесосить пылесос*