Введение

18. Процедура Assign/AssignFile. Назначение. Формат. Логические имена устройств ввода-вывода. Пример.

19. Файлы с типом. Синтаксис задания. Процедуры открытия, чтения и записи, определенные над файлами с типом. Пример.

20. Организация прямого доступа к элементам файлов с типом. Встроенные функции, определенные над файлами с типом. Закрытие файлов с типом. Примеры.

Пример:

21. Текстовые файлы. Синтаксис задания. Процедуры и функции, обеспечивающие чтение из текстовых файлов, и их особенности по сравнению с файлами с типом. Допустимые типы вводимых переменных. Пример.

22. Процедуры и функции, обеспечивающие запись в текстовые файлы, и их особенности по сравнению с файлами с типом. Допустимые типы выводимых переменных. Размещение информации в строке по умолчанию. Управление размещением информации по позициям строки. Пример.

23. Процедуры, управляющие работой буфера ввода-вывода для текстовых файлов. Пример.

24. Сравнительная характеристика внутренней структуры представления информации в текстовом файле и файле с типом. Достоинства и недостатки использования текстового файла и файла с типом.

25. Файлы без типа. Синтаксис задания. Назначение. Факторы повышения скорости обмена информацией. Процедуры и функции, определенные над файлами без типа. Пример.

26. Проверка операций ввода-вывода. Пример.

27. Ссылочный тип. Назначение. Синтаксис задания. Представление в памяти. Виды указателей. Операции над указателями. Пример.

28. Процедуры New и Dispose. Назначение. Достоинства и недостатки их использования. Пример.

29. Процедуры GetMem и FreeMem. Назначение. Достоинства и недостатки их использования. Пример.

30. Строковый тип в Delphi. Представление в памяти. Автоматическое управление памятью для Delphi-строк.

31. Директива absolute. Принцип работы. Пример.

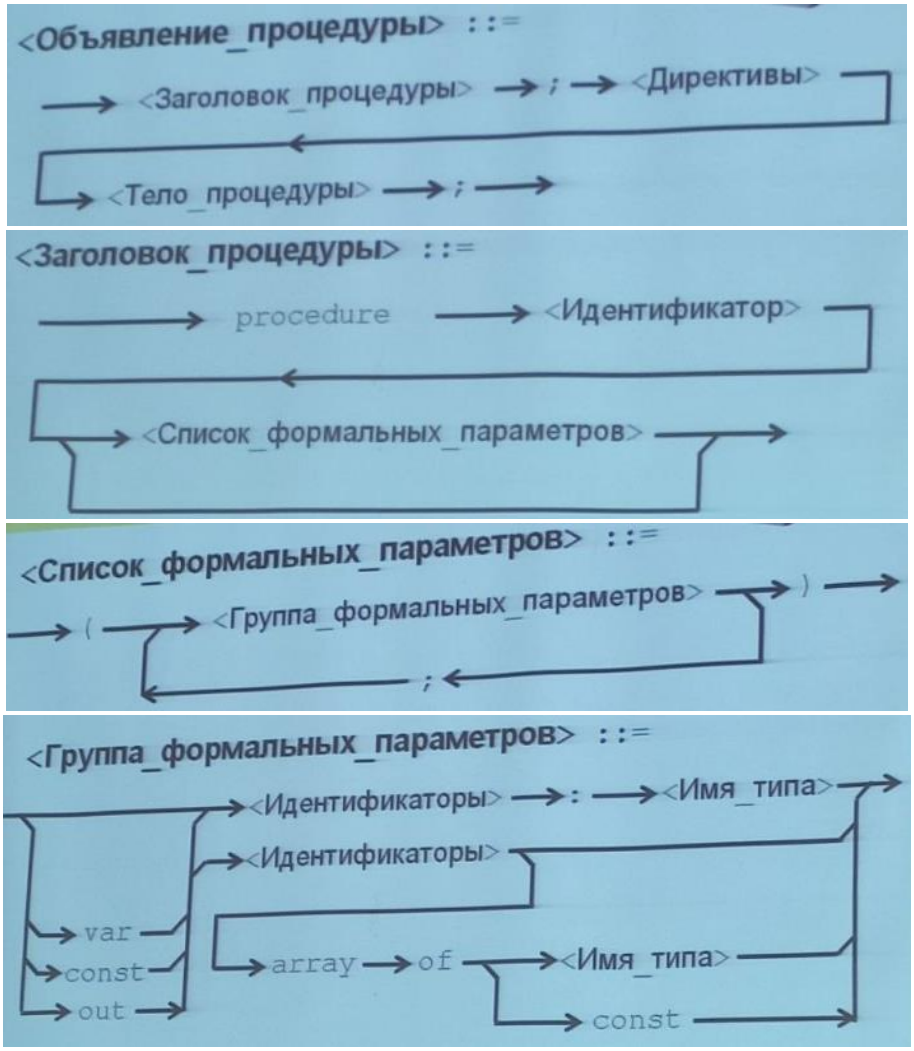
32. Зарезервированное слово packed. Влияние на представление в памяти записей и массивов.

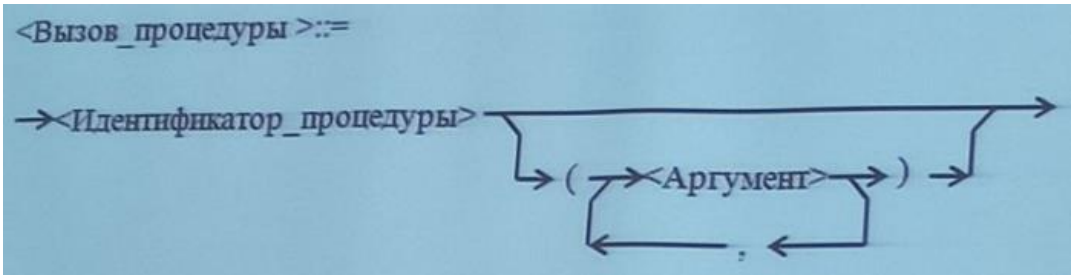
1. Процедуры. Синтаксис объявления процедур. Вызов процедуры. Организация связи по данным при использовании процедур без параметров. Пример.

Подпрограмма - поименованная логически законченная группа операторов языка, которую можно вызвать для выполнения по имени любое количество раз из различных мест программы.

Подпрограммы:

- Процедуры;
- Функции.





Процедуры без параметров используются редко.

- Большинству подпрограмм нужны исходные данные для работы.
- Передача подпрограмме исходных данных через глобальные переменные - это **дурной тон**!

Недостатки передачи параметров через глобальные переменные:

- Процедуру сложно повторно использовать в другой программе;
- Если несколько процедур используют одну и ту же глобальную переменную, логика программы усложняется;
- Меньшая гибкость по сравнению с другими способами передачи параметров.

```
procedure PrintNumb(ANumb: Integer);
begin
    writeln(ANumb);
end.

var
    Numb: Integer;

begin
    …
    Numb := 42;
    PrintNumb(Numb);
    …
end.
```

2. Виды формальных параметров подпрограмм. Параметры-значения. Назначение, синтаксис. Механизм организации взаимосвязи с фактическими параметрами при использовании параметров-значений. Пример.

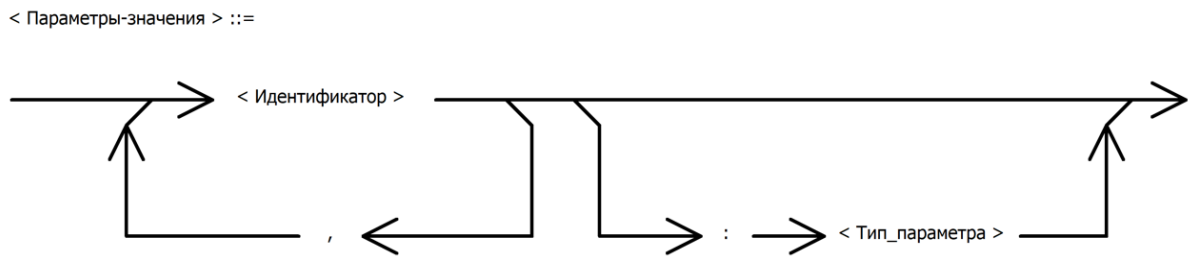
Виды формальных параметров:

- Параметры-значения;
- Параметры-переменные;
- Параметры-константы.

Для параметров-значений при вызове создается локальная переменная:

- Существует только во время выполнения процедуры/функции;
- При каждом вызове это новая переменная.

В заголовке процедуры/функции записывается без модификаторов, указываются только имя и тип данных.



ТУТ ЛИНИИ НАД ": <Тип_параметра>" НЕ ДОЛЖНО БЫТЬ!!!

Порядок передачи:

1. Вычислить значение фактического параметра.
2. Создать локальную переменную с именем формального параметра и присвоить ей значение фактического параметра.

В стек помещается копия значения фактического параметра. Фактическим параметром может быть любое выражение того же типа данных, что и параметр.

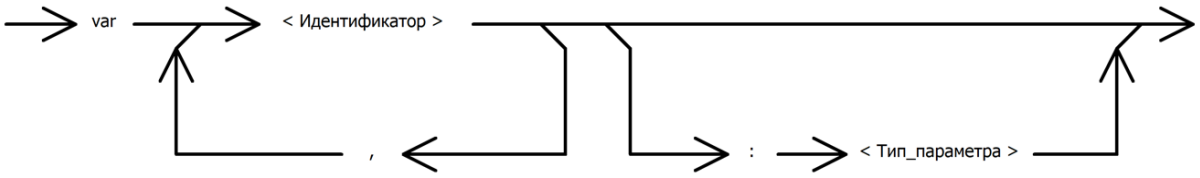
```
procedure PrintNumb(ANumb: Integer);
begin
    writeln(ANumb);
end.

var
    Numb: Integer;

begin
    …
    Numb := 42;
    PrintNumb(Numb);
    …
end.
```

3. Параметры-переменные и параметры-константы. Назначение, синтаксис. Механизм организации взаимосвязи с фактическими параметрами при использовании параметров-переменных и параметров-констант. Пример.

Параметры-переменные:

- В заголовке процедуры записывается с зарезервированным словом var.
- Фактическим параметром может быть только переменная того же типа данных, что и параметр.
- В стек помещается адрес переменной- параметра.
- Изменение формального параметра внутри подпрограммы приводит к изменению фактического параметра.

< Параметры-переменные > ::=

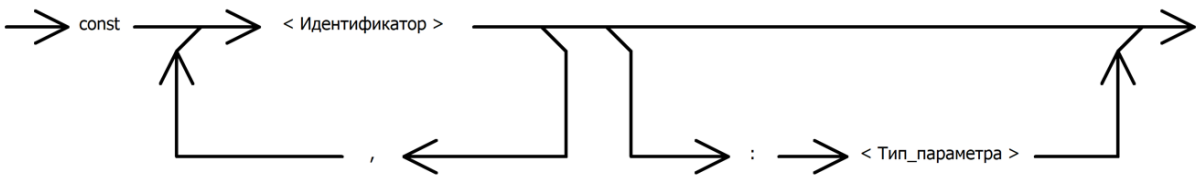


```
procedure ZeroOut(var ANumb: Integer);
begin
    ANumb := 0;
end.

var
    Numb: Integer;

begin
    …
    Numb := 42;
    ZeroOut(Numb);
    …
end.
```

Параметры-константы:

- В заголовке процедуры записывается с зарезервированным словом const.
- Фактическим параметром может быть выражение того же типа данных, что параметр.
- Компилятор запрещает изменение значения внутри подпрограммы.
- Способ передачи выбирается компилятором.
- Рекомендуется использовать при передаче параметров строкового типа и записей.
    - Создание копии значения для этих типов более сложно, чем для остальных.
    - Использование const-параметра позволяет компилятору передать адрес значения, не создавая его копию.

< Параметры-константы > ::=




```
procedure PrintNumb(const ANumb: Integer);
```

```
begin
    writeln(ANumb);
end.

var
    Numb: Integer;

begin
    …
    Numb := 42;
    PrintNumb(Numb);
    …
end.
```

4. Параметры без типа. Назначение, синтаксис. Способы обеспечения совместимости с фактическими параметрами. Примеры.

- Для всех параметров-значений тип должен быть указан.
- Для var- и const-параметров указание типа необязательно.
    - Параметры, у которых тип не указан, называются параметрами без типа (untyped).

Параметры без типа — это группа параметров, перед которыми стоит служебное слово Var или Const и за которыми не следует тип. Фактическими параметрами при вызове подпрограммы в данном случае могут быть переменные любого типа (КОНСТАНТНЫЕ ЗНАЧЕНИЯ НИЗЯ).

2 способа работы с бестипными параметрами

1) зарезервированное слово Absolute
2) поставить в соответствие какой-либо тип (приведение типов)

Absolute позволяет создать переменную, которая будет располагаться в той же области памяти, что и какая-либо другая переменная.

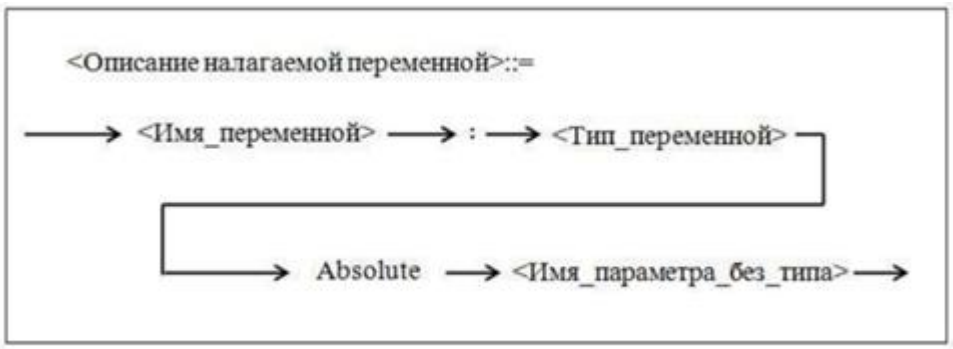


Рисунок 1.6 – Синтаксическая диаграмма описания налагаемой переменной

```
procedure DisplayValue(var param);
var
  x: Integer absolute param;
begin
  Writeln('Значение параметра: ', x);
end;
```

Приведение типов позволяет изменить тип значения переменной или выражения на другой тип данных. Приведение типов может быть неявным (автоматическим) или явным (явным указанием типа).

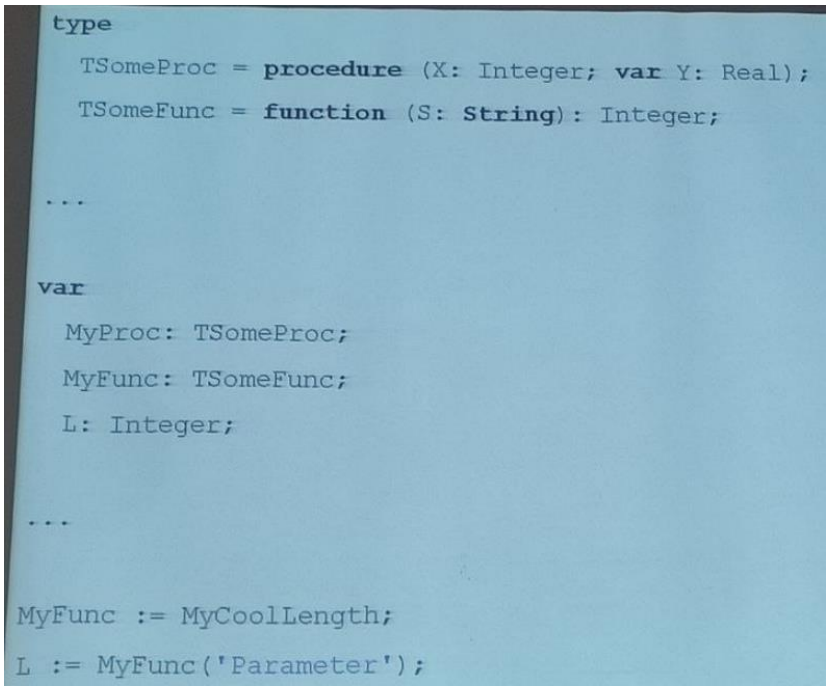
```
procedure DisplayValue(var param);
var
  x: Integer;
begin
  x := Integer(param);

  Writeln('Значение параметра: ', x);
end;
```

5. Параметры процедурного типа. Назначение, синтаксис. Условия совместимости с фактическими параметрами. Пример.

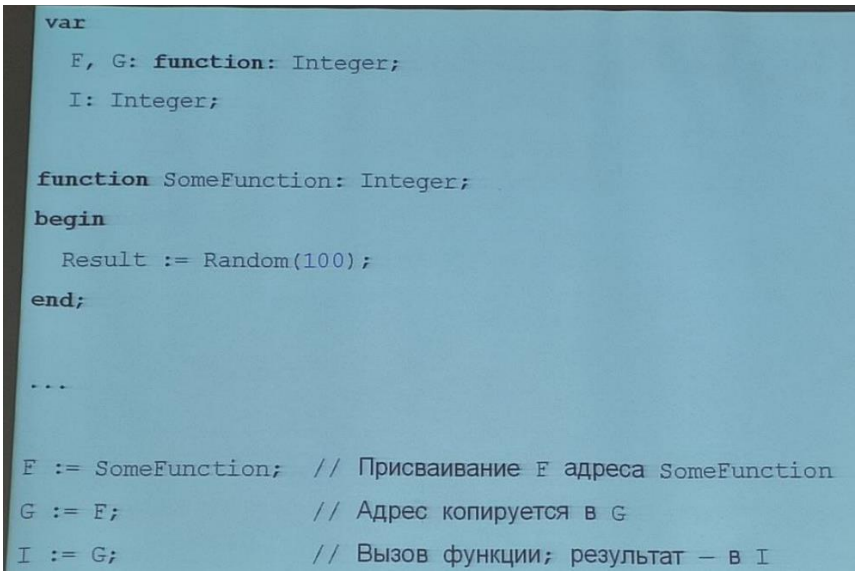
Код, как и данные, - не более, чем последовательность байтов, занимающая участок памяти.
У процедур и функций есть адреса.

- Адрес процедуры/функции - адрес самого первого байта ее кода.

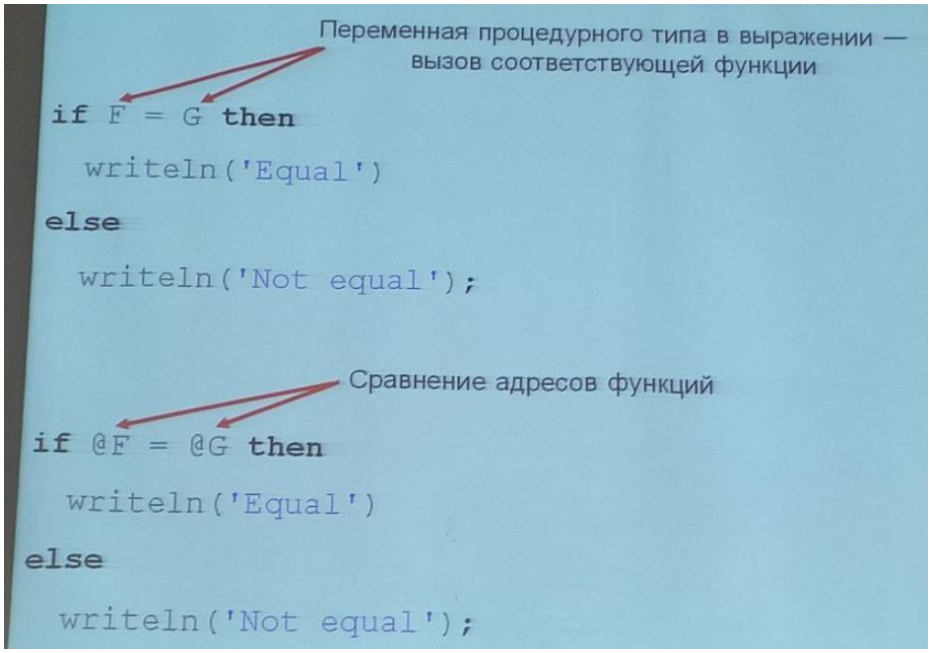
Процедурные типы - типы, значениями которых являются адреса процедур/функций.



```
type
  TSomeProc = procedure (X: Integer; var Y: Real);
  TSomeFunc = function (S: String): Integer;

...

var
  MyProc: TSomeProc;
  MyFunc: TSomeFunc;
  L: Integer;

...

MyFunc := MyCoolLength;
L := MyFunc('Parameter');
```



```
var
  F, G: function: Integer;
  I: Integer;

function SomeFunction: Integer;
begin
  Result := Random(100);
end;

...

F := SomeFunction;  // Присваивание F адреса SomeFunction
G := F;             // Адрес копируется в G
I := G;             // Вызов функции; результат — в I
```



Ограничения процедурных переменных

- Нельзя присваивать адреса стандартных процедур/функций.
- Нельзя присваивать адреса процедур/функций, объявленных с директивой inline.
- Процедура/функция должна точно соответствовать процедурному типу.

Применение

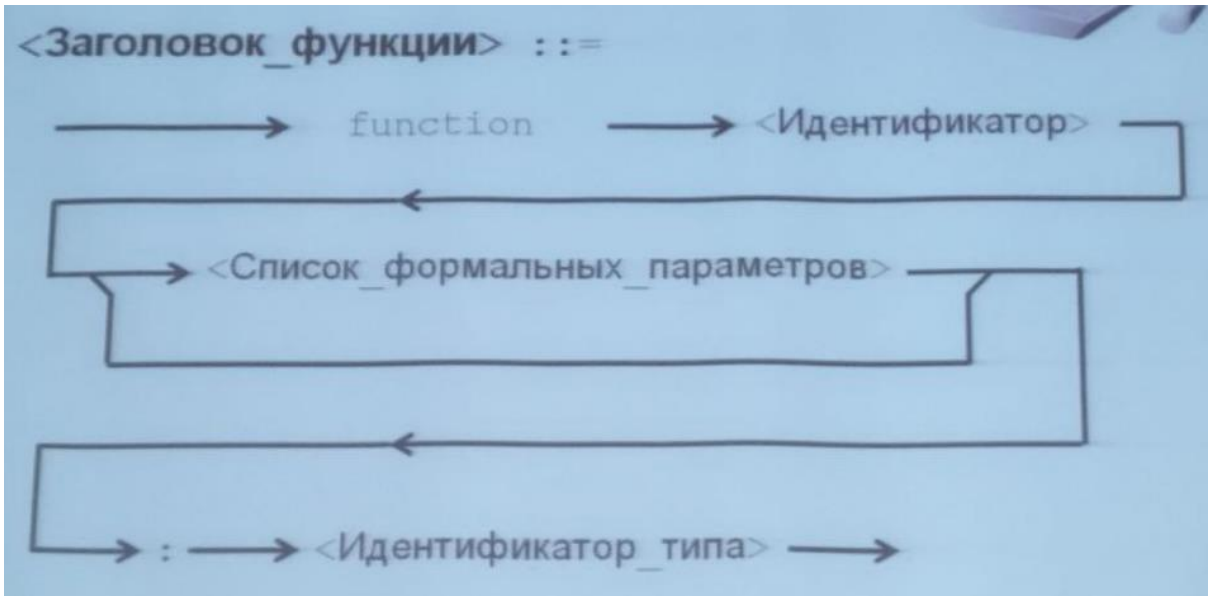
```
type
  TCompareFunc = function (X1, X2: Real): Boolean;
  TArray = array [1..100] of Real;

procedure Sort(A: TArray; CompareFunc: TCompareFunc);
var
  ...
begin
  ...
  if CompareFunc(A[i], A[j]) then
  ...
end;
```

6. Функции. Описание функций. Вызов функции. Возврат значения из функции. Пример.

Функция - процедура, возвращающая значение.

- Или процедура - функция, не возвращающая значения

Функции аналогичны процедурам, за исключением нескольких отличий.



Функции могут использовать те же механизмы передачи параметров и результатов, что и процедуры.
Кроме того, каждая функция вычисляет так называемое возвращаемое значение функции.

Возвращаемое значение:

Типом возвращаемого значения:

- может быть:
  - стандартным типом;
  - предварительно описанным типом;
- не может быть:
  - заданием типа.
    (НЕЛЬЗЯ function GetMax (X1, X2: Real): array of Integer;)

Классический способ:

```
function GetMax (X1, X2: Real): Real;
begin
```

```
    if x1 > X2 then
      GetMax := X1;
    else
      GetMax := X2;
end;
```

Delphi-способ:

```
function GetMax (X1, X2: Real): Real;
begin
    if x1 > X2 then
      Result := X1;
    else
      Result := X2;
end;
```

Оба способа присваивания возвращаемого значения могут использоваться в одной и той же функции.

Если возвращаемое значение функцией не было присвоено, оно является неопределённым.

Функции (в отличие от процедур) могут использоваться в составе выражений.

- Сам вызов функции является выражением, его тип - тип возвращаемого значения функции.

```
MaxValue := GetMax(5, 10.3) + Z;
```

7. Рекурсивные подпрограммы. Виды рекурсии. Достоинства и недостатки рекурсивной записи подпрограмм. Явная рекурсия. Пример.

Рекурсия - определение какого-либо понятия через само это понятие.

Классический пример:

$$n! = \begin{cases} 1, \text{если } n = 0, \\ n \cdot (n-1)!, \text{если } n > 0. \end{cases}$$

Этот же пример на Delphi:

```
function Fact (N:Integer): Integer;
begin
    if N = 0 then
        Result := 1
    else
        Result := N * Fact(N-1);
end;
```

При каждом рекурсивном вызове создается новый набор:

- фактических параметров;
- локальных переменных.

На время выполнения текущего рекурсивного вызова данные всех предыдущих вызовов становятся недоступными.

- Рекурсивность - это не свойство задачи, а особенность её реализации.
- Ту же подпрограмму можно реализовать без рекурсии: n! = 1*2*3*.....*(n-1).

```
function Fact (N: Integer): Integer;
var
    I: Integer;
begin
    Result := 1;
    for I := 1 to N do
        Result := Result * I;
end;
```

Преимущества:

- Как правило, запись такой программы короче и нагляднее.

Недостатки:

- Как правило, такая программа выполняется медленнее, чем эквивалентная ей нерекурсивная.

Рекурсия:

- явная - обращение к подпрограмме содержится в теле самой подпрограммы;
- неявная (взаимная) - обращение к подпрограмме содержится в теле другой подпрограммы, к которой происходит обращение из данной подпрограммы.

Реинферабельность:

Важным аспектом рекурсии является ее реинферабельность, то есть возможность использования рекурсивной подпрограммы для различных задач, не изменяя ее код. Реинферабельность достигается путем использования параметров и возвращаемых значений, чтобы передавать данные между различными вызовами подпрограммы.

Возьмем наш пример с вычислением факториала. Он реализует рекурсивный алгоритм для вычисления факториала числа. Функция принимает один параметр N, представляющий число, для которого нужно вычислить факториал. Внутри функции проверяется базовое условие: если N равно 0, то возвращается 1. В противном случае вызывается рекурсивный вызов функции, передавая N - 1 в качестве аргумента, и результат умножается на N.

Пример демонстрирует реинферабельность рекурсивной подпрограммы. Мы можем использовать эту подпрограмму для вычисления факториала разных чисел, передавая разные значения параметра n, и каждый раз получать правильный результат, не изменяя код.

8. Директивы подпрограмм. Неявная рекурсия. Пример.

Директива inline

Обычная подпрограмма:

- формируется машинный код для тела подпрограммы;
- в местах вызова вставляются инструкции вызова этого кода.

Inline-подпрограмма:

- тело подпрограммы вставляется прямо в месте вызова.

Пример:

```
function Min (X1, X2: Integer): Integer; inline;
begin
    If x1 < X2 then
        Result := X1
    Else
        Result := X2;
end;
```

Директива forward

Описание подпрограммы:

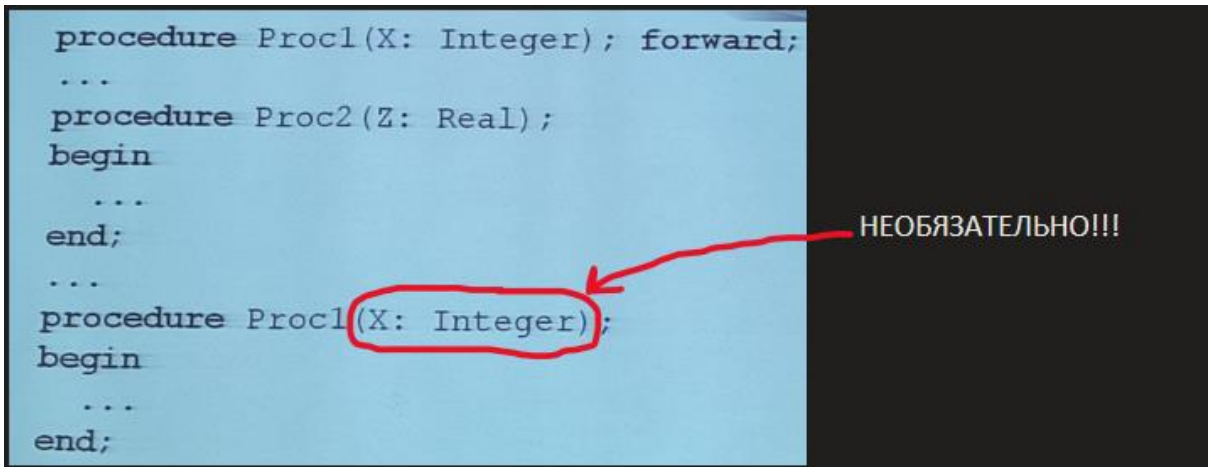
- Опережающие;
- Определяющее.

Опережающее описание:

- Заголовок подпрограммы;
- Директива forward.

Определяющее описание:

- Заголовок подпрограммы;
- Блок (тело подпрограммы).



Assembler: Директива assembler позволяет вставлять инструкции на языке ассемблера непосредственно в код на Delphi. Это позволяет использовать низкоуровневые операции и оптимизировать производительность программы.

External: Директива external используется для объявления внешней функции или процедуры. Она указывает, что реализация данной функции или процедуры находится внутри отдельного модуля, такого как DLL (динамическая библиотека). При использовании external необходимо указать имя модуля и имя функции/процедуры

Far: Директива far используется для определения процедур, которые должны быть скомпилированы с использованием модели памяти "far". Модель памяти "far" позволяет обращаться к данным и коду, находящимся в других сегментах памяти, что полезно при работе с большими объемами памяти.

Interrupt: Директива interrupt используется для определения процедур, которые будут вызываться в ответ на прерывания аппаратных устройств или программные прерывания. Процедуры, помеченные как interrupt, предназначены для обработки прерываний и выполняют определенные действия в ответ на возникновение прерывания. Обработка прерываний может потребовать использования специфичных для платформы функций или процедур.

Неявная рекурсия

Неявная (взаимная) рекурсия - обращение к подпрограмме содержится в теле другой подпрограммы, к которой происходит обращение из данной подпрограммы.

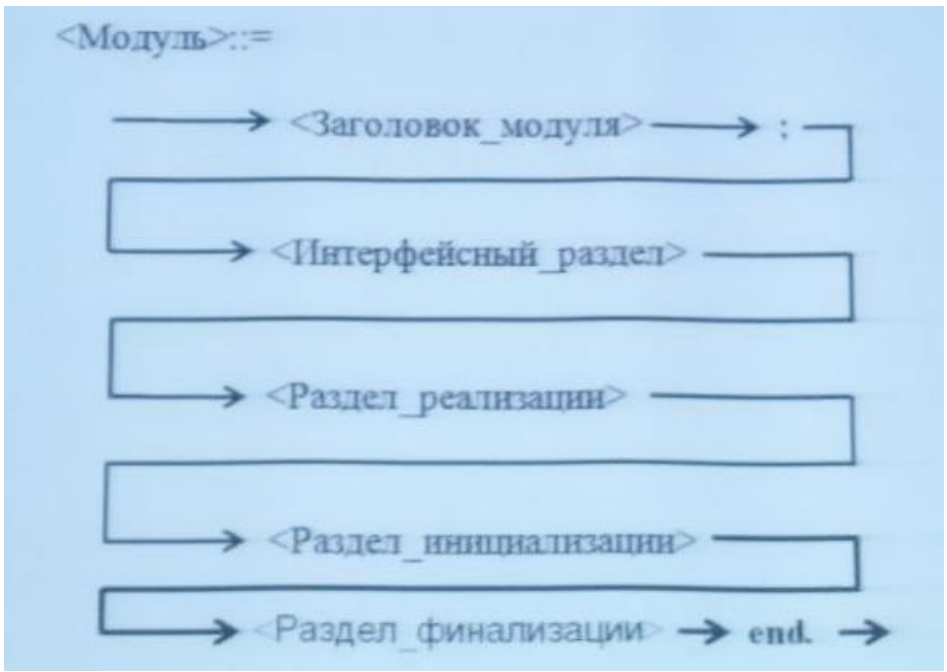
```
procedure ProcedureB(ACounter: Integer); forward;
procedure ProcedureA(ACounter: Integer);
begin
  Writeln('Procedure A: ', ACounter);
  if ACounter > 0 then
    ProcedureB(ACounter - 1);
```

```
end;

procedure ProcedureB(ACounter: Integer);
begin
  Writeln('Procedure B: ', ACounter);
  if ACounter > 0 then
    ProcedureA(ACounter - 1);
end;
```

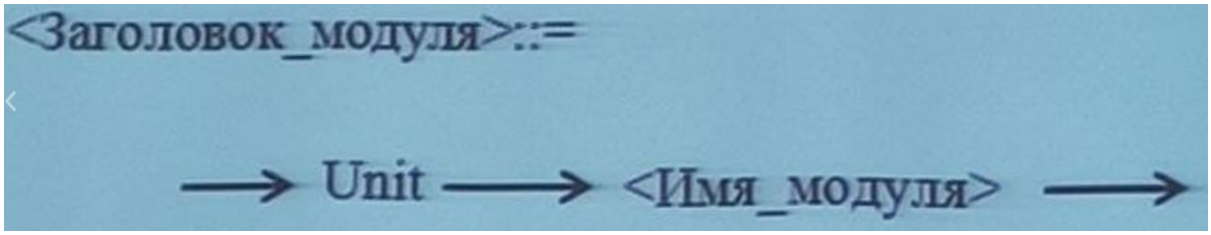
9. Библиотечные модули пользователя. Назначение модуля. Структура модуля. Синтаксис и назначение разделов модуля. Пример.

Файлы в Delphi:

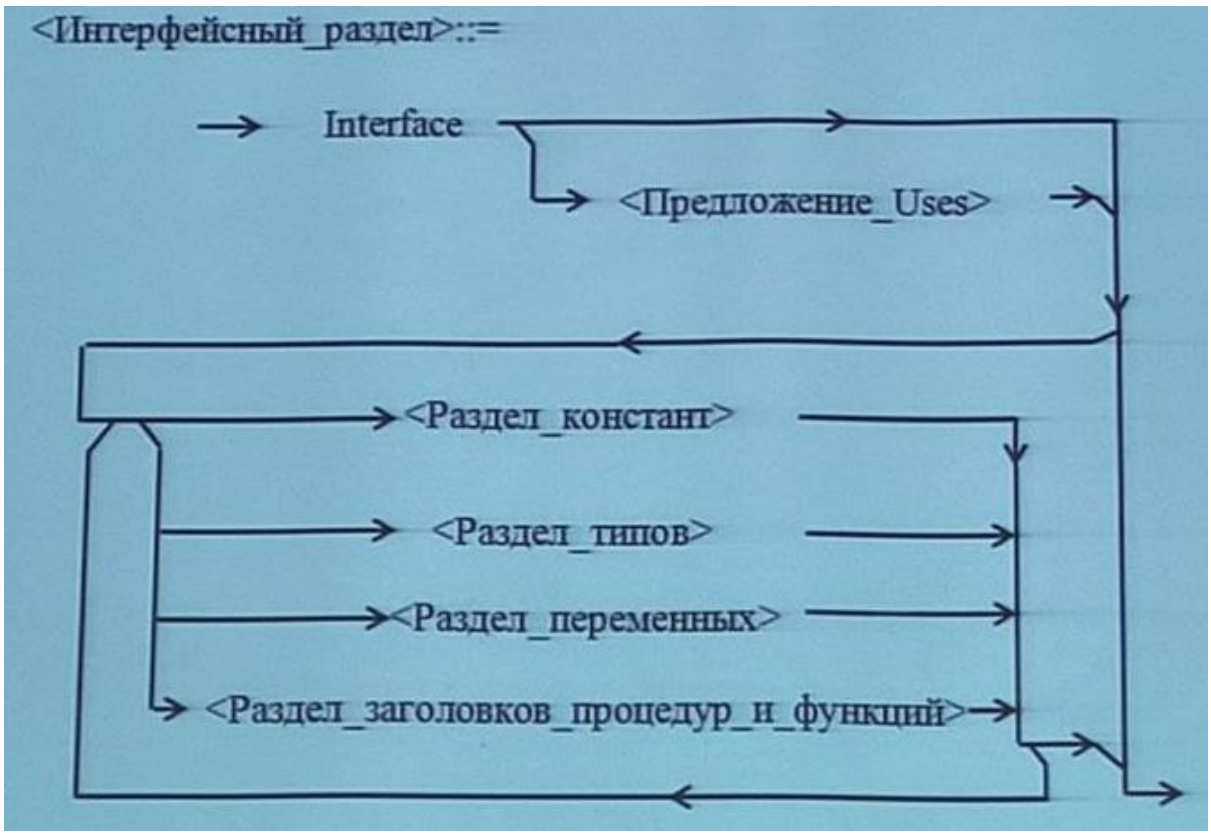
- файл проекта (.dpr) - программный модуль;
- исполняемый файл (.exe) — результат компиляции проекта;
- вспомогательные модули (.pas);
- скомпилированные модули (.dcu).



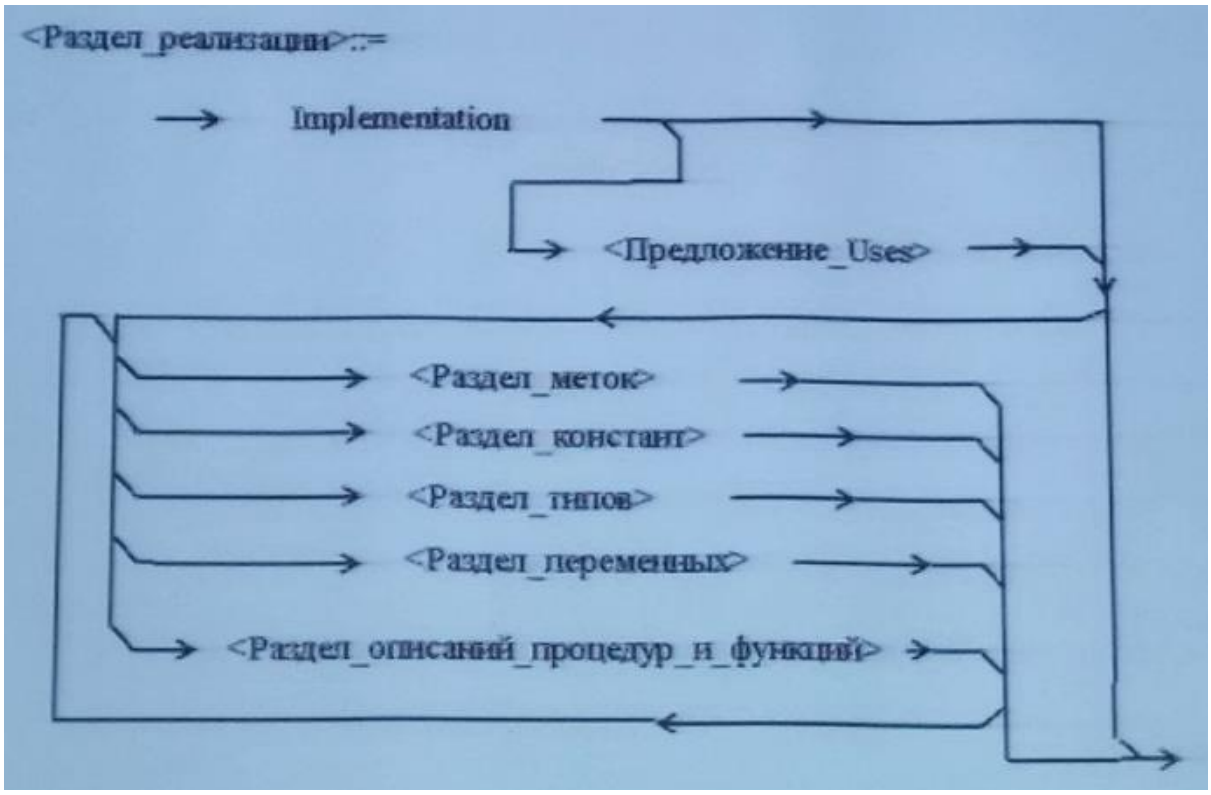
<Заголовок_модуля>::=



<Интерфейсный_раздел>::=



<Раздел_реализации>::=



В секции **Interface** (интерфейсном разделе) описываются внешние элементы – глобальные типы, константы, переменные, процедуры и функции. Это те элементы, которые доступны основной программе или другому модулю. Описания могут следовать в любом порядке (как и в программном модуле).

Если при объявлении типов, данных или подпрограмм интерфейсной секции используются элементы, введенные в других модулях, то эти модули должны быть перечислены в директиве Uses сразу после слова Interface. В директиве Uses данной секции рекомендуется указывать только те модули, которые необходимы именно в этой секции.

В секции **Implementation** (разделе реализации) описываются локальные элементы (типы, константы, переменные, метки). Здесь также содержатся тела процедур и функций, заголовки которых описаны в секции Interface.
Все элементы, описанные в интерфейсной секции, доступны в секции реализации. Все, что описано в секции реализации, недоступно программе или другим модулям. Это скрытая часть модуля.
Описанные в разделе реализации типы, константы, переменные являются глобальными по отношению к подпрограммам этого раздела, а также операторам раздела инициализации.
Если в телах подпрограмм или при объявлении типов, переменных в разделе реализации используются имена, объявленные в других модулях, и эти модули не попали в предложение Uses раздела Interface, то они перечисляются в предложении Uses после слова Implementation.

Все разделы модуля являются необязательными, но служебные слова, начинающие разделы, должны присутствовать обязательно.

```
Unit Hollow; {Пустой модуль}
 Interface
 Implementation
 End.
```

10. Особенности работы с библиотечными модулями пользователя. Пример.

1. Подключение модулей происходит в порядке их перечисления в предложении Uses: слева направо. В этом же порядке срабатывают разделы инициализации модулей.
   Инициализация происходит только при работе программы. При подключении модуля к модулю инициализация не выполняется.
2. Порядок подключения влияет на доступность библиотечных типов, данных, процедур и функций. Например, имеются два модуля U1 и U2. В каждом из этих модулей в интерфейсной секции описаны одноименные тип Ned, переменная D, процедура SetNed. Но они реализованы по-разному.

Если в программе записано предложение использования Uses U1, U2; то обращения к элементам Ned, D, SetNed будут эквивалентны обращениям к модулю U2.

Если в главной программе также объявляются эти имена, то они заменяют собой имена, относящиеся к модулю.

В этом случае, чтобы обеспечить доступ к содержимому нужного модуля, используются составные имена, в первой части которых указывается имя модуля. Например, U1.Ned, U1.D, U1.SetNed(X)

3. Решение проблемы **закольцованности** (циклических ссылок модулей друг на друга). Например, модуль U1 использует элементы из модуля U2, а модуль U2 использует элементы из модуля U1.

Если оба модуля подключают друг друга в разделе реализации, то закольцованность автоматически разрешается компилятором.
Но если хотя бы один из модулей подключает другой в разделе Interface, то проблема закольцованности может быть решена только программным путем. В этом случае обычно используется третий модуль.
Достоинства использования модулей Unit:

1. Наличие модулей позволяет использовать модульное программирование, то есть представлять программу в виде

модулей и при необходимости корректировать отдельные модули, а не всю программу в целом.
2. Модули компилируются независимо друг от друга; при подключении модуля к другой программе он не компилируется заново. Таким образом, сокращается время компиляции больших программ. Это же справедливо и при корректировке отдельных модулей – заново компилируются только зависящие от них модули.
3. Наличие модулей позволяет создавать большие программы с суммарным размером исполняемого кода большим, чем 64К.

Директива компилятора

Позволяет подставить в текст модуля содержимое внешнего текстового файла.
{$I SomeFile.inc}

- при его включении на место директивы {$I …} он должен вписаться в структуру и смысл программы без ошибок;
- он должен содержать законченный смысловой фрагмент, то есть блок от Begin до End (например, тело процедуры) должен храниться целиком в одном файле;
- включаемый файл не может быть указан в середине раздела операторов.
- Включаемые файлы сами могут содержать директивы {$I …}. Максимальный уровень такой вложенности равен восьми.

К **недостаткам** такого подключения к программе внешнего файла по сравнению с использованием библиотечных модулей можно отнести следующее:

- подключаемые файлы каждый раз компилируются заново. Это увеличивает время компиляции;
- размер программы не может превышать 64К.

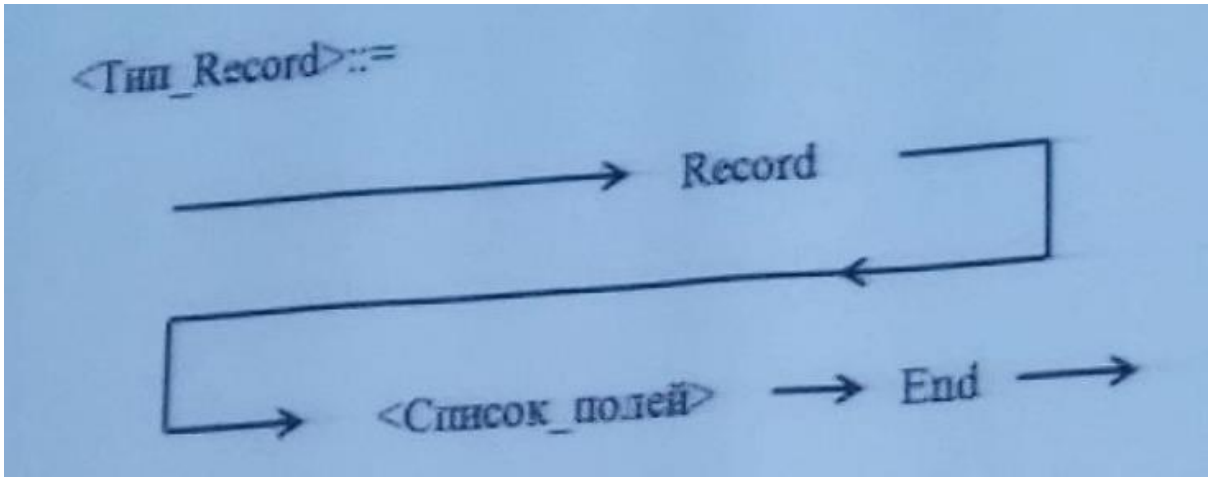
11. Записи. Синтаксис задания. Записи без вариантной части. Операции над записями и над полями. Пример.

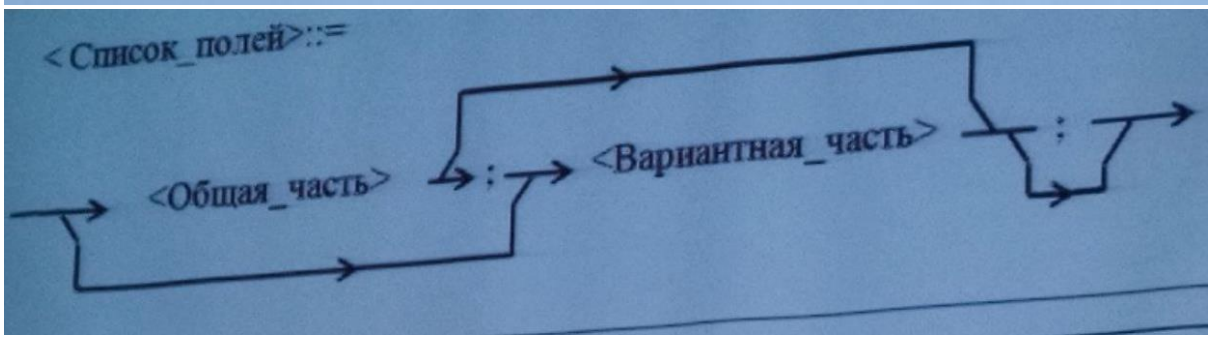
Запись - структура данных, состоящая из упорядоченных разнородных компонентов

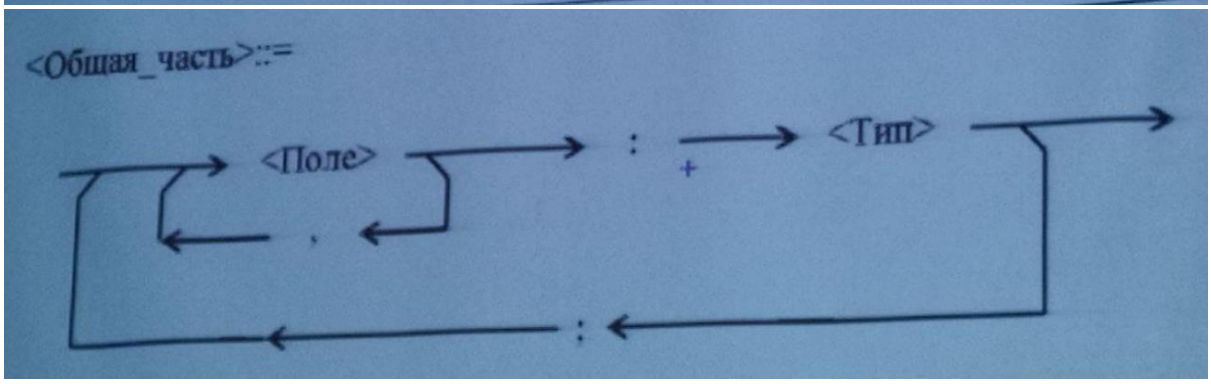
Компоненты записи - поля

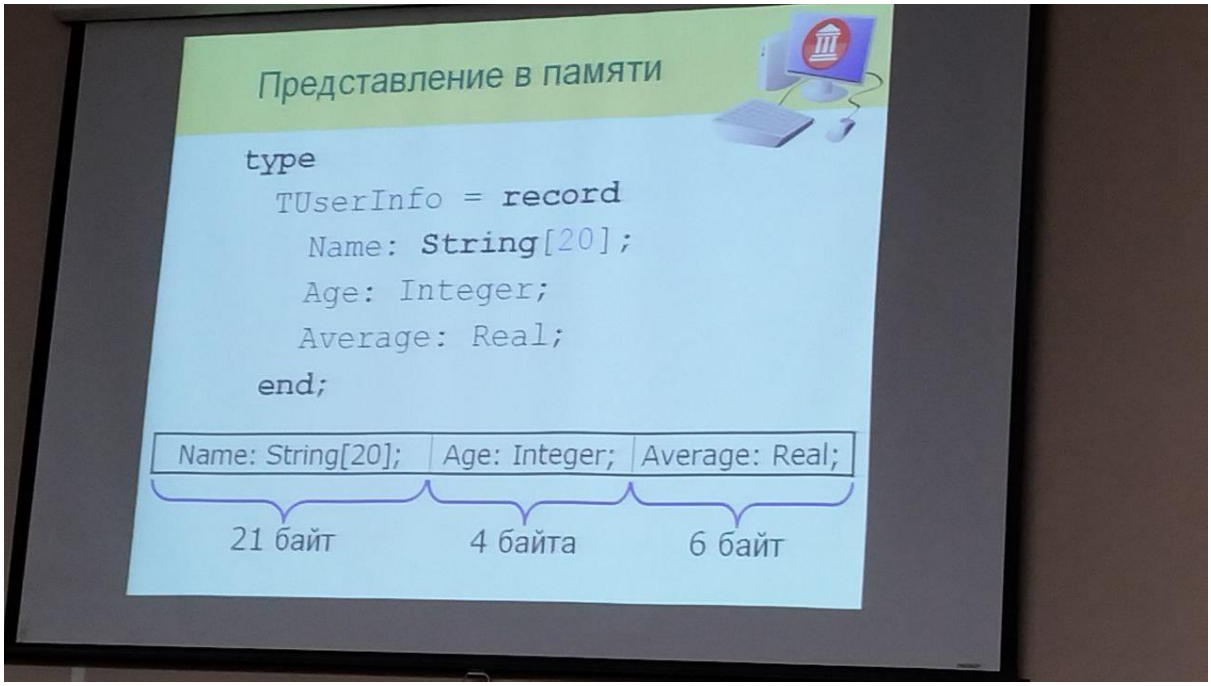
Другие названия:

- комбинированный тип
- тип record
- структура (в С-подобных языка)









Объём памяти, занимаемой переменной записью, складывается из размеров полей нижнего уровня вложенности.

Полная переменная - переменная, имеющая тип записи верхнего уровня вложенности.

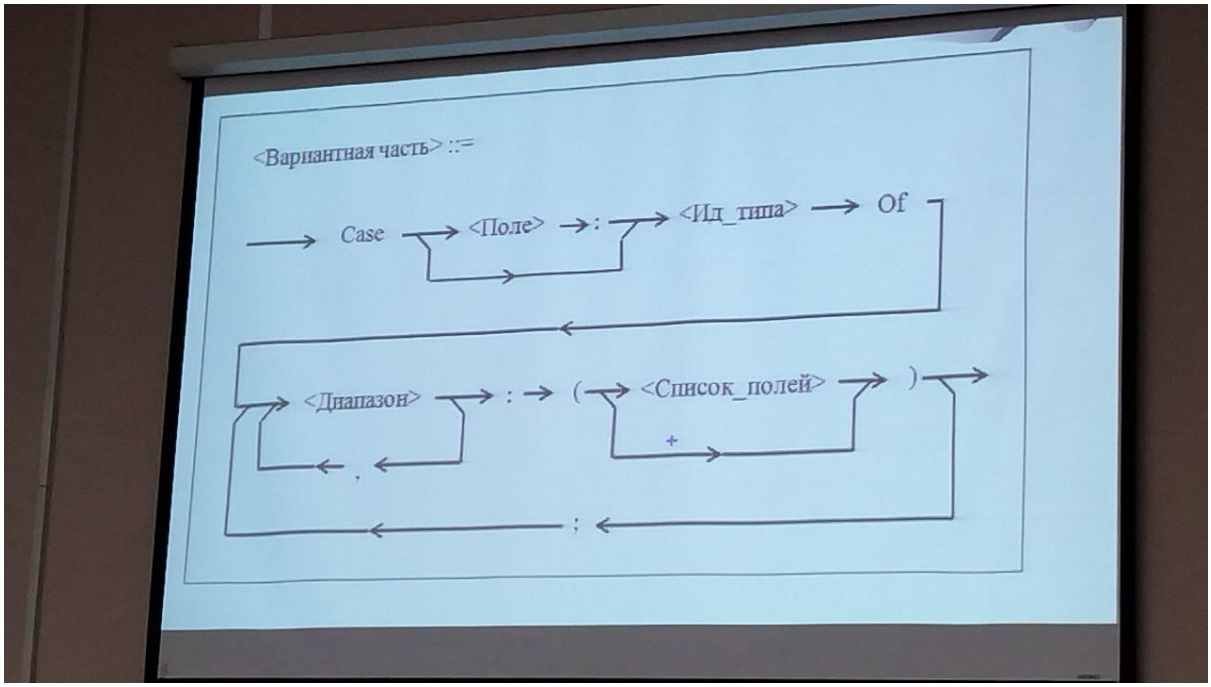
Для полных переменных существует одна операция:

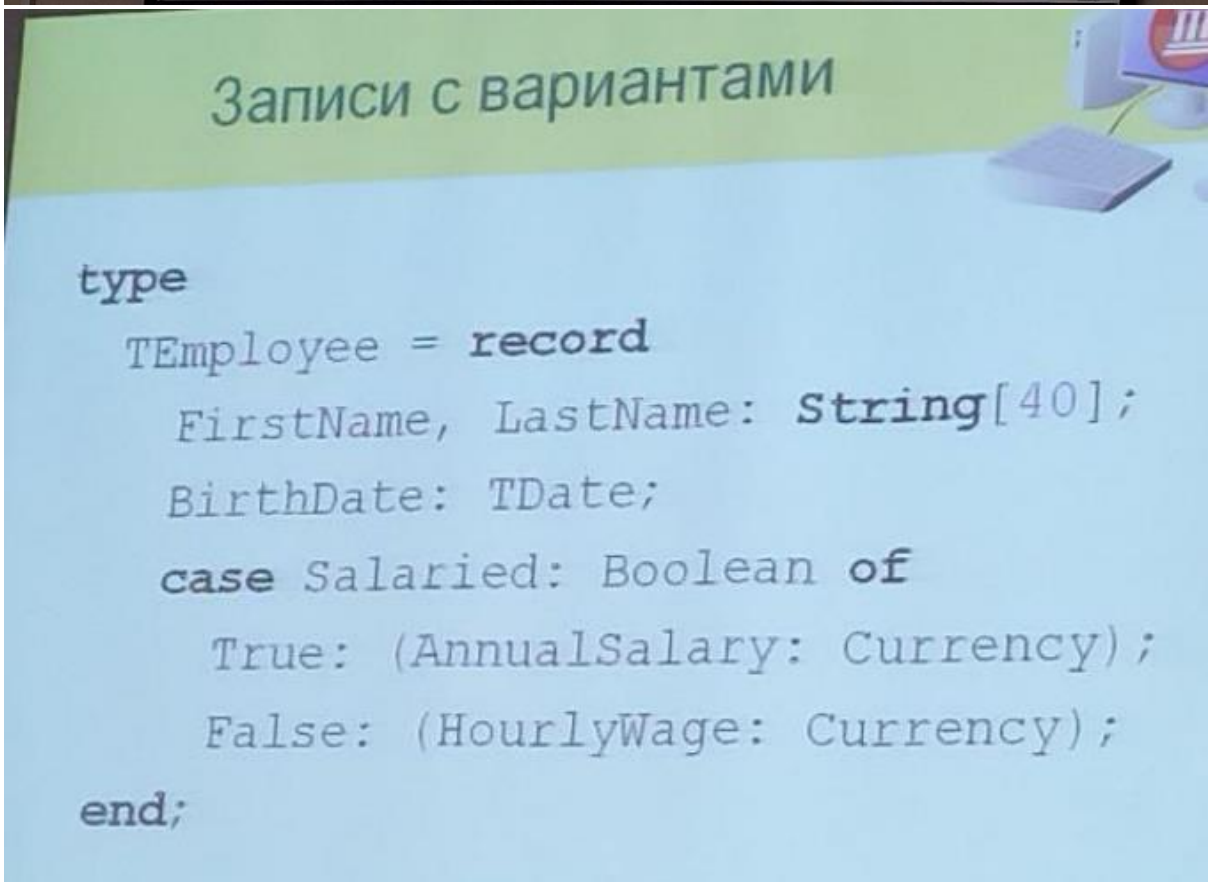
- Операция присваивания

Для определения значения полной переменной необходимо присвоить значения всем ее полям.

Возможные операции для полей зависят от их типа данных

12. Записи с вариантами. Синтаксис задания. Особенности задания записей с полем признака и без него. Пример.





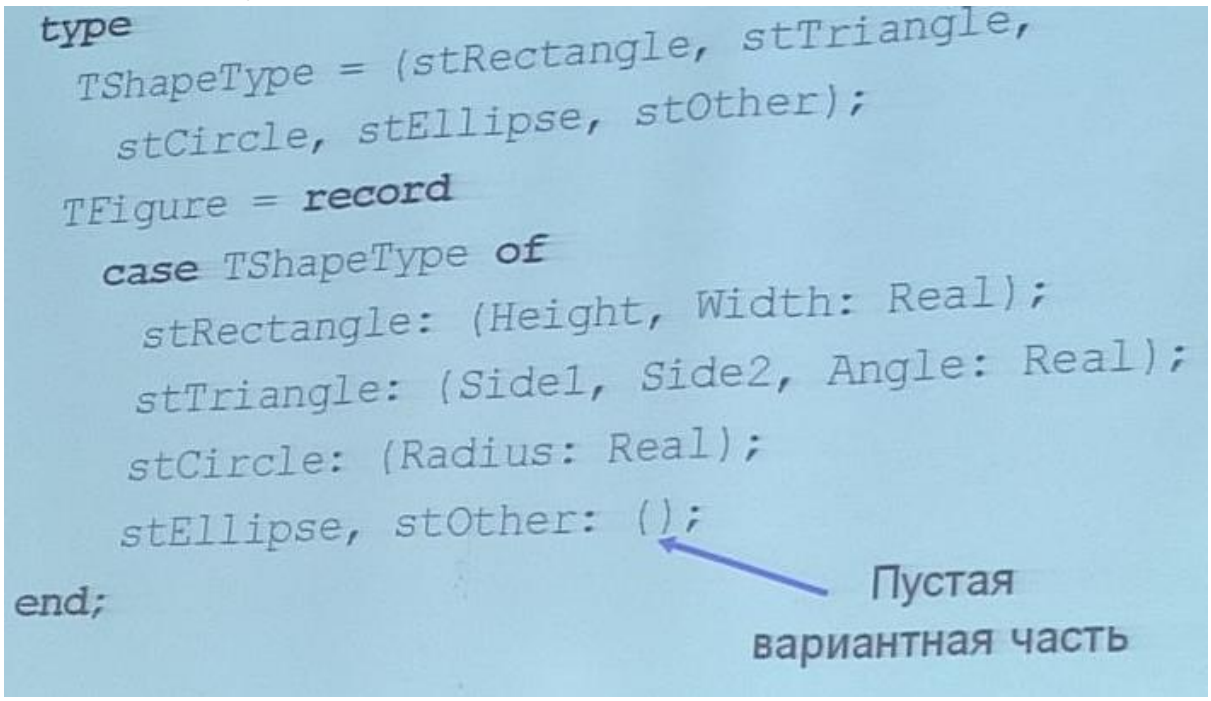
Поля каждого варианта занимают одну и ту же область памяти.
Размер этой области памяти определяется по наибольшему варианту.

Поле, объявленное в case, - поле признака.
Поле признака относится к общей части записи.

Никаких дополнительных проверок в зависимости от значения поля признака не производится

К любым полям вариантной части можно обращаться независимо от значения поля признака.

Запись может не иметь общей части
В этом случае вместо поля признака указывается только имя любого перенумерованного типа.



У case в записи с вариантами нет отдельного служебного слова end.
Слово end заканчивает всю конструкцию описания записи.

Вариантные поля должны находится после общей части.

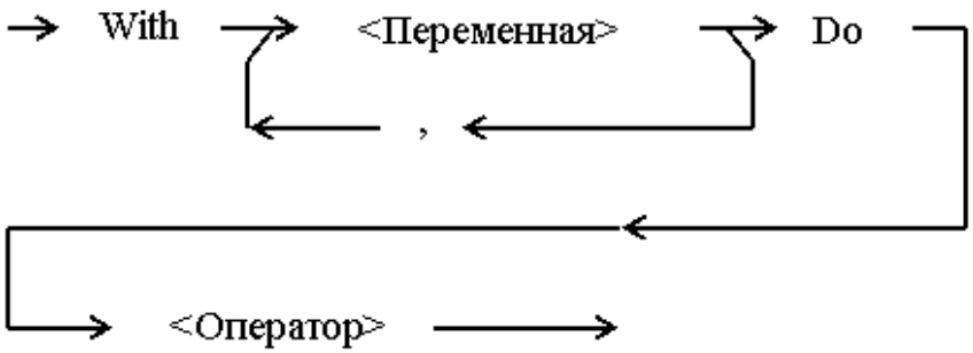
Имена полей должны быть уникальными:

- Даже если поля относятся к разным вариантам.

13. Оператор присоединения. Назначение. Формат. Полная и сокращенная формы оператора присоединения. Примеры использования.

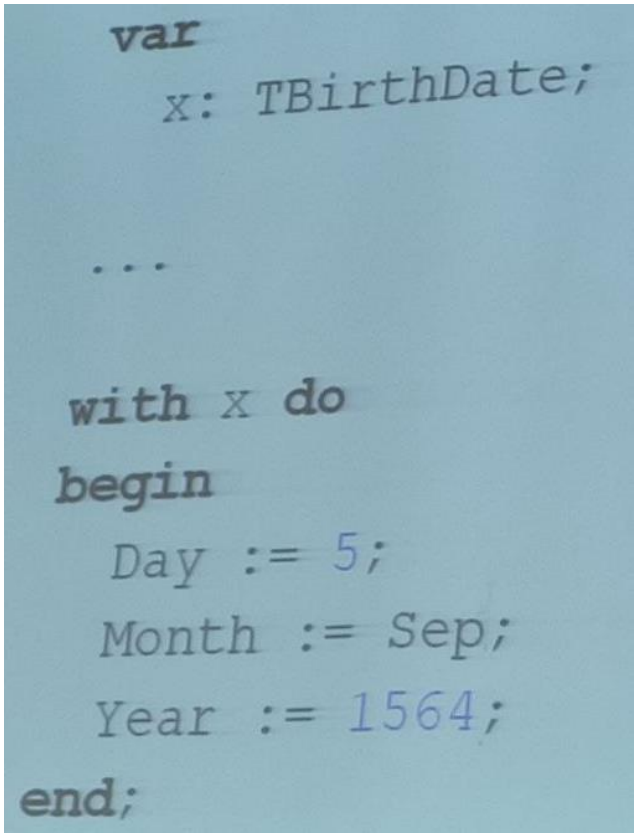
При работе с полями в их составном имени необходимо писать путь к полю через все уровни иерархии, начиная от полного имени записи. С учетом того, что количество уровней иерархии может достигать девяти, составное имя поля может содержать девять компонент. Работа с такими именами неудобна, а программа оказывается очень громоздкой. Для сокращения составного имени поля может быть использован оператор присоединения With.

<Оператор_With>::=



В операторе With указывается список переменных типа Record. Оператор With облегчает доступ к полям этих записей и минимизирует повторные адресные вычисления. Внутри <Оператора>, вложенного в оператор With, к полям этих записей можно обращаться как к простым переменным.

Адрес переменной типа Record вычисляется до выполнения оператора With. Любые модификации переменных, влияющие на вычисленное значение адреса, до завершения оператора With не отражаются на значении вычисленного ранее адреса.



```
var
  x: TBirthDate;

...

with x do
begin
  Day := 5;
  Month := Sep;
  Year := 1564;
end;
```

with var1, var2, var3 do
работает, как вложенные with:

```
with var1 do
    with var2 do
        with var3 do
            …
```

При обращении к полю, которое есть в нескольких переменных, перечисленных в операторе with, предпочтение отдаётся переменной, указанной ближе к концу списка.

Если необходимо изменить это поведение или вообще обратиться к одноимённой переменной, необходимо записать её полное имя.

14. Множественный тип. Синтаксис задания. Базовый тип множества. Представление в памяти. Конструктор множества. Пример.

Множественный тип (set) соответствует понятию множества в математике:

- Неупорядоченный набор значений.

Макс кол-во элементов - 256.

Все элементы должны быть одного и того же типа - базового типа множества.

Базовый тип

Любой скалярный тип:

- Кроме вещественных
- Для остальных типов есть ограничения.

Общее правило

- Ord(x) для любого x базового типа должно попадать в диапазон от 0 до 255.

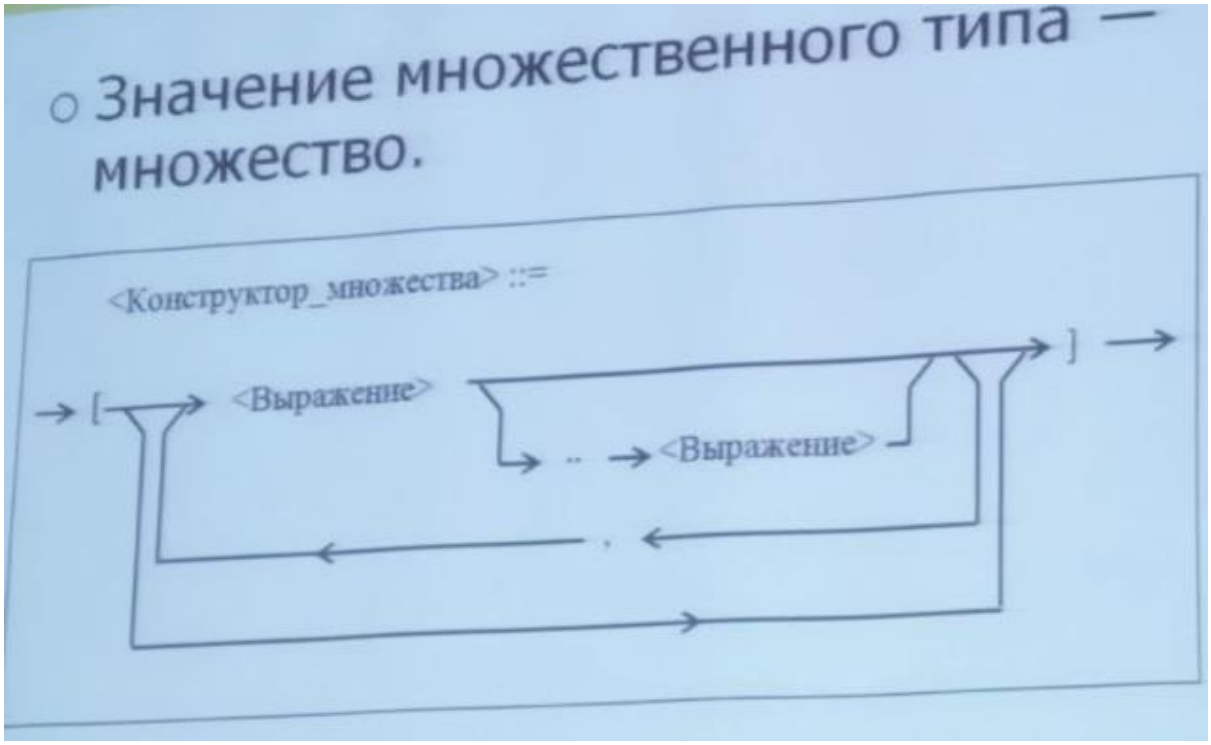
Представление в памяти

Занимает от 1 до 32 байт.

- В зависимости от количества элементов.

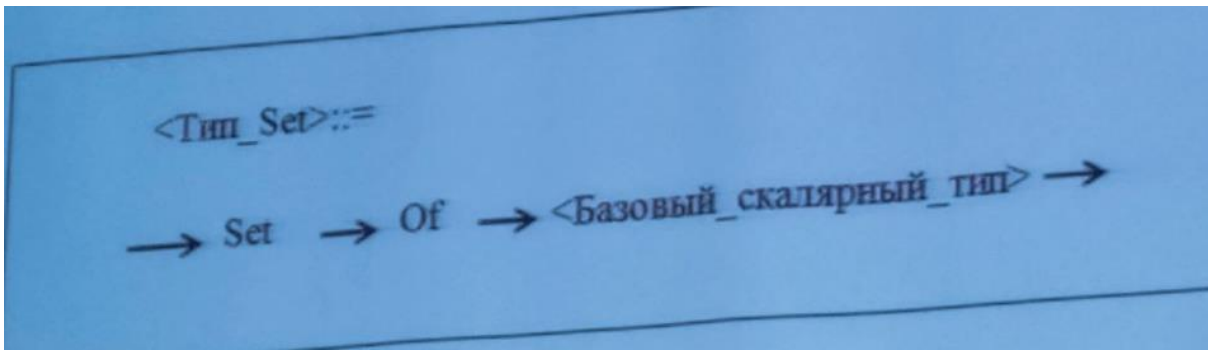
Каждому из возможных элементов ставится в соответствие 1 бит.

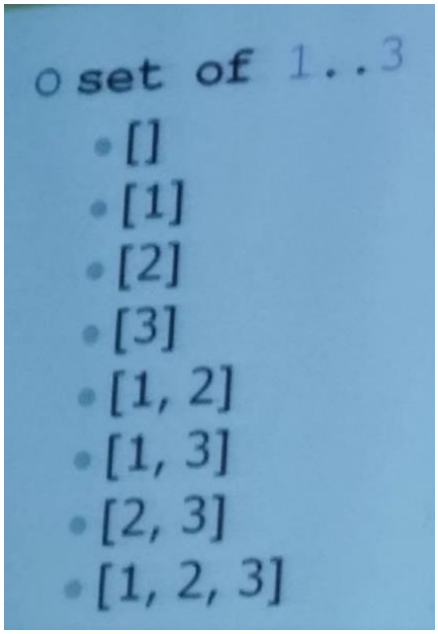
- Если бит содержит 1, элемент включен во множество.
- Если бит содержит 0,элемент не входит во множество.




```
[]
[2, 3, 5]
['A'..'Z']
[1..10, 100..110]
[Jan, Feb, Dec]
```

Задание множественного типа:

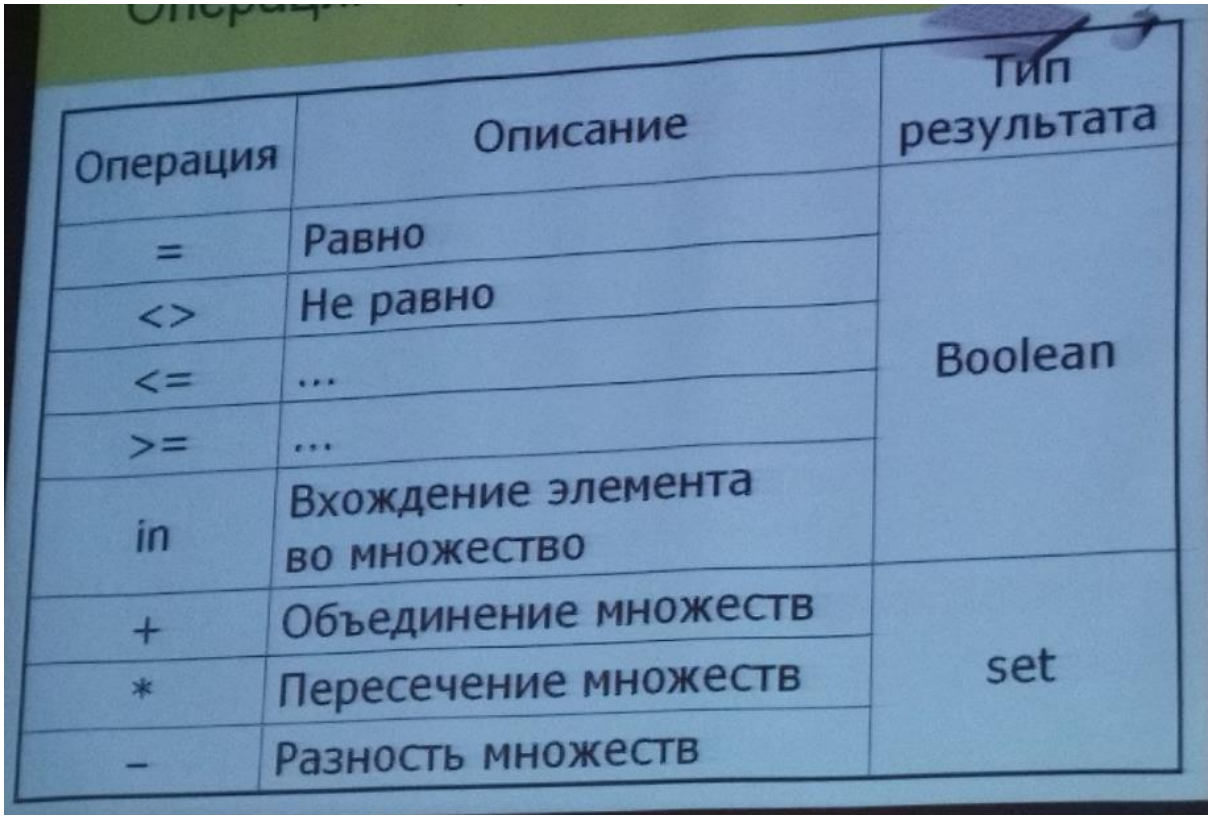




15. Множественные выражения. Операции и встроенные функции над множествами. Ввод-вывод множественных переменных. Пример.

Множественное выражение - выражение, значением которого является множество

Операции над множествами:



Операция	Описание	Тип результата
=	Равно	Boolean
<>	Не равно	
<=	...	
>=	...	
in	Вхождение элемента во множество	
+	Объединение множеств	set
*	Пересечение множеств	
-	Разность множеств	

(Еще есть Not - дополнение множества (одноместная операция), Xor - исключающее объединение множеств. A Xor B = A + B - A * B. У двух операций тип результата является set)

Над множественными переменными определена одна встроенная функция – Sizeof(X), указывающая количество байт для представления значения X множественной переменной. (Возвращаемый параметр Integer)

В языке Delphi нет прямого способа ввода и вывода множества в целом. Множество представляет собой набор элементов, и для ввода и вывода нужно работать с отдельными элементами множества. Вот пример, как организовать ввод и вывод элементов множества в Delphi:

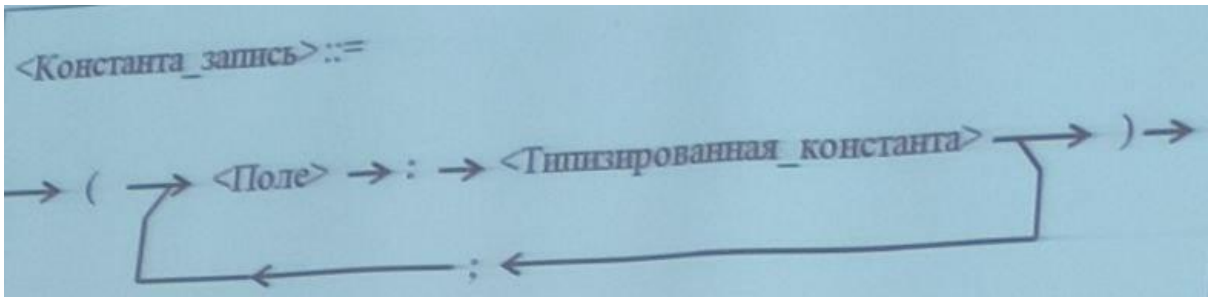
```
var
  A: set of 1..5;
  X, I: Integer;

begin
  for I := 1 to 5 do
  begin
```

```
    ReadLn(X);
    A := A + X;
  end;

  for I := 1 to 5 do
    if I in A then
      WriteLn(I);
end.
```

16. Типизированные константы-записи (с вариантами и без) и константы-множества. Назначение. Синтаксис задания. Примеры использования.

1) Типизированные константы-записи

Назначение: Типизированные константы-записи позволяют определить константу, имеющую структуру записи с заданными полями и значениями.




```
type
  TPerson = record
    Name: string;
    Age: Integer;
  end;

const
  PersonConst: TPerson = (Name: 'John Smith'; Age: 30);
```

2) Константы-записи с вариантной частью

Назначение: Константы-записи с вариантной частью позволяют определять константы, которые могут содержать различные наборы полей в зависимости от условий.

```
type
  TShape = record
    ShapeType: Integer;
    case Integer of
      0: (Width, Height: Integer);
      1: (Radius: Integer);
  end;

const
  Rectangle: TShape = (ShapeType: 0; Width: 10; Height:
                       20);
  Circle: TShape = (ShapeType: 1; Radius: 5);
```

3) Константы-множества

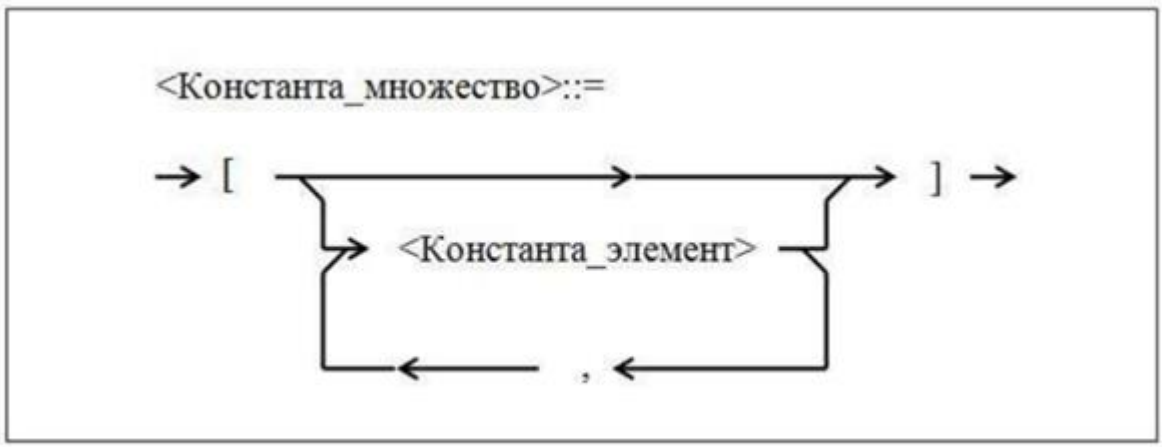


Рисунок 4.8 – Синтаксическая диаграмма константы-множества

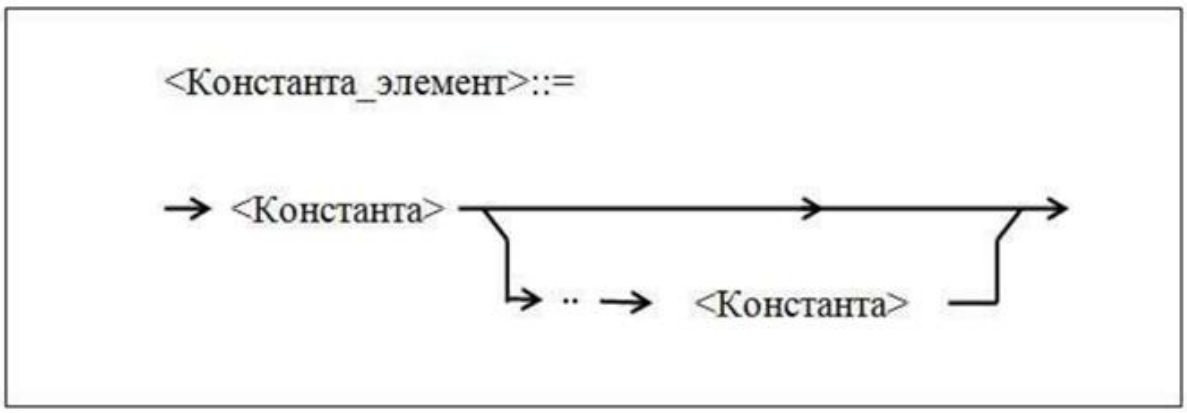


Рисунок 4.9 – Синтаксическая диаграмма константы-элемента

Назначение: Константы-множества позволяют определить константу, представляющую множество значений определенного перечислимого типа.

```
type
```

```
  TDaysOfWeek = (Monday, Tuesday, Wednesday, Thursday,
                 Friday, Saturday, Sunday);

const
  WorkingDays: set of TDaysOfWeek = [Monday, Tuesday,
               Wednesday, Thursday, Friday];
```

Данные константы позволяют удобно и наглядно определять и использовать структурированные данные и множества в программе. Они обеспечивают статическую проверку типов и могут быть использованы в различных контекстах, например, для инициализации переменных, параметров функций и процедур, а также для сравнения и проверки значений в условных операторах и циклах.

17. Файлы. Логический и физический файл. Способы доступа к элементам файла. Типы файлов. Синтаксис задания. Пример.

Файловый тип – это произвольная последовательность элементов, длина которой заранее не определена, а конкретизируется в процессе выполнения программы. Это определение **логического файла**, т.е. того, который используется в программе (файл с точки зрения программиста). **Физический файл** (набор данных) – это поименованная область памяти на внешнем носителе, в которой хранится некоторая информация (файл с точки зрения пользователя).

Способы доступа:

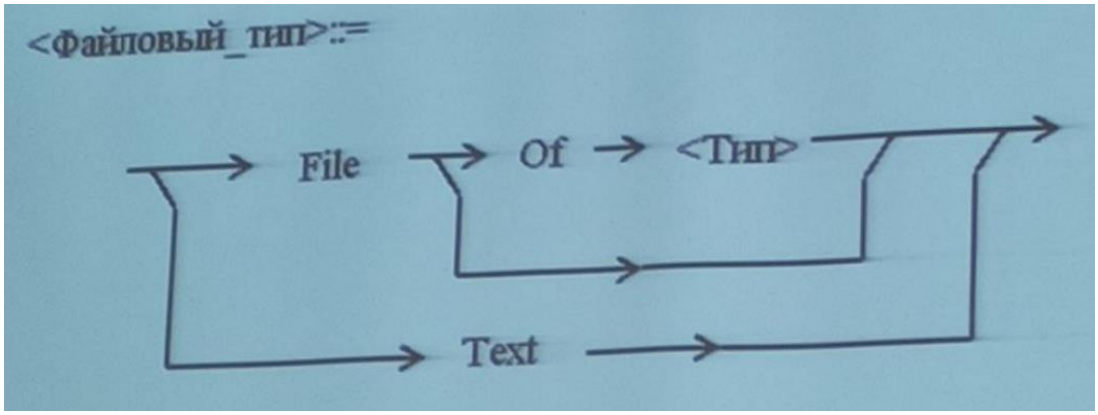
- Последовательный
- Прямой

При **последовательном** доступе по файлу можно двигаться только последовательно, начиная с первого его элемента. В этом случае доступен лишь очередной элемент файла. Чтобы добраться до n-го элемента файла, необходимо начать с первого элемента и пройти через предыдущие n – 1 элементов.

При **прямом** доступе можно обратиться непосредственно к элементу файла с номером n, минуя предварительный просмотр n – 1 элемента файла.

Виды переменных файлового типа:

- "Текстовые файлы"
- "Файлы с типом"
- "Файлы без типа"



Над значениями Файлового типа не определены никакие операции. Работа с файлами осуществляется с помощью так называемых процедур и функций ввода-вывода.

Примеры:

```
fileVar: File; // файл без указания типа данных
intFile: File of Integer;
textFile: Text; //или  textFile: TextFile;
```

С каждым открытым файлом связан т.н. **указатель файла**.

- Другие названия: окно файла, текущая позиция файла.

Указатель файла определяет позицию доступа элемент файла, с которым будет выполняться следующая операция ввода/вывода.

Позиция файла, следующая за его последним элементом, считается **концом файла**.
Конец файла никак не помечается!

18. Процедура Assign/AssignFile. Назначение. Формат. Логические имена устройств ввода-вывода. Пример.

Procedure AssignFile(var F; FileName: String);

- Связывает файловую переменную F с файлом FileName

Любым другим процедурам ввода-вывода предшествует процедура Assign.
Назначение процедуры – организует связь между конкретным физическим файлом на внешнем носителе (конкретным набором данных) и файловой переменной программы (логическим файлом) F.
Имя конкретного набора данных определяется параметром Name. Name – это полное имя физического файла. В общем случае оно имеет вид:
<Диск>:\<Имя_каталога>\…\<Имя_каталога>\<Имя_файла>

<Диск> задается символом от A до Z (символ логического устройства). Если он опущен, то подразумевается логическое устройство, принятое по умолчанию.

\<Имя_каталога>\…\<Имя_каталога>\ - это путь через подкаталоги к фактическому имени файла. Если они опущены, то считается, что файл находится в текущем каталоге.

<Имя_файла> - фактическое имя физического файла. Оно может иметь максимально 8 символов.
Затем может идти уточнение (тип файла) – максимально 3 символа, отделенное от имени точкой. Например, можно определить такие имена файлов:
Rez Rez.pas Rez.exe Rez.txt Rez.dat
Уточнение помогает программисту, пользователю, системе программирования или операционной системе работать с файлами.

Пример полного имени файла: A:\Katalog1\Katalog2\Rez.pas
Максимальная длина полного имени файла – 79 символов.

Процедура Assign всегда предшествует другим процедурам ввода-вывода. Ее нельзя применять к уже открытому файлу.

Вместо имени физического файла в качестве параметра Name в списке фактических параметров процедуры Assign может использоваться любое устройство ввода-вывода (клавиатура, печать, дисплей и т. п.) В этом случае Name – это символическое (логическое) имя устройства ввода-вывода.
Для использования доступны следующие символические имена устройств:

1) **CON** – устройство консоли (при вводе – это клавиатура, при выводе – экран дисплея). Например, процедура Assign (F, ‘CON’); означает ввод в переменную F с клавиатуры или вывод из F на экран.
По умолчанию стандартные текстовые файлы Input и Output связаны с консолью, что соответствует следующему фрагменту программы:
Assign (Input, ‘CON’); Assign (Output, ‘CON’);

2) **LPT1, LPT2, LPT3** – устройства печати. Если подключено одно устройство печати, то используется либо имя LPT1 либо PRN:
Assign (F, ‘PRN’); Assign (Output, ‘PRN’);
С данными логическими устройствами может использоваться только имя выходного файла.

3) **COM1, COM2** – устройства последовательного ввода-вывода, используемые для обмена данными между компьютерами. Вместо COM1 может быть использовано имя ‘AUX’.

4) **NUL** – нулевое устройство. Для него при выводе не осуществляется никаких действий. При попытке чтения возникает ситуация конца файла.

5) **CRT** – устройство текстового ввода-вывода. Аналогично устройству CON, но имеет ряд дополнительных функций управления экраном (например, установка цветов, указание места

на экране для вывода и т.п.). CRT не поддерживается операционной системой.

6) **‘’** – использование пустой строки вместо имени Name. В этом случае файловая переменная F связывается с CON (по аналогии с пунктом 1)). Например, Assign (F, ‘’);

19. Файлы с типом. Синтаксис задания. Процедуры открытия, чтения и записи, определенные над файлами с типом. Пример.

Состоят из однотипных компонент, тип которых указан при объявлении.
Примеры объявления файлов с типом.

```
Type
     Zap = Record
          I: Integer;
          R: Real
     End;
 Var
     F1: File Of Real;
     F2: File Of Char;
     F3: File Of String[50];
     F4: File Of Zap;
     F5: File Of Integer;
```

Для работы с типизированными файлами существуют следующие процедуры и функции ввода-вывода:
1) Процедура **Assign** – связывает файловую переменную с внешним файлом.См билет 18

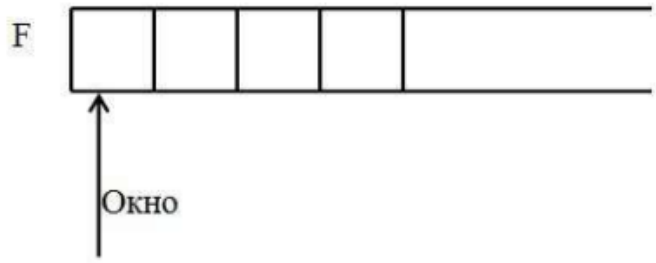
2) Процедура **Rewrite (F)** – создает и открывает новый файл F.

- Если F уже открыт, закрывает и снова открывает его.
- Указатель файла устанавливается на его начало.

Пример:

```
procedure Rewrite (var F: file);
```

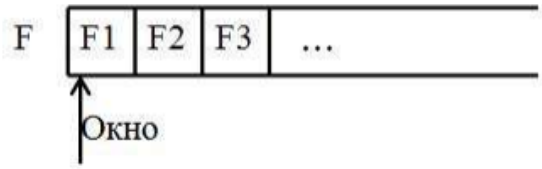
Перед использованием процедуры Rewrite файл F должен быть связан с внешним файлом (набором данных) процедурой Assign.



3) Процедура **Reset (F)**
Открывает существующий файл.

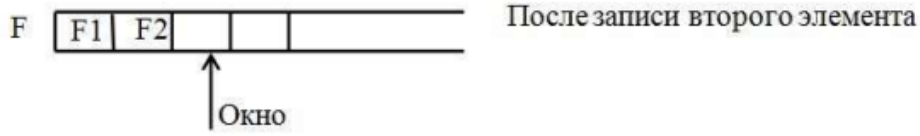
- Если F уже открыт, закрывает и снова открывает его.
- Указатель файла устанавливается на его начало.
- Режим определяется переменной FileMode.

Фактически при этом открывается внешний файл с именем, присвоенным переменной F процедурой Assign. Если файл с данным именем не существует, возникает сообщение об ошибке. Процедура Reset(F) может быть применена к файлу любое количество раз. При выполнении этой процедуры содержимое файла не изменяется.



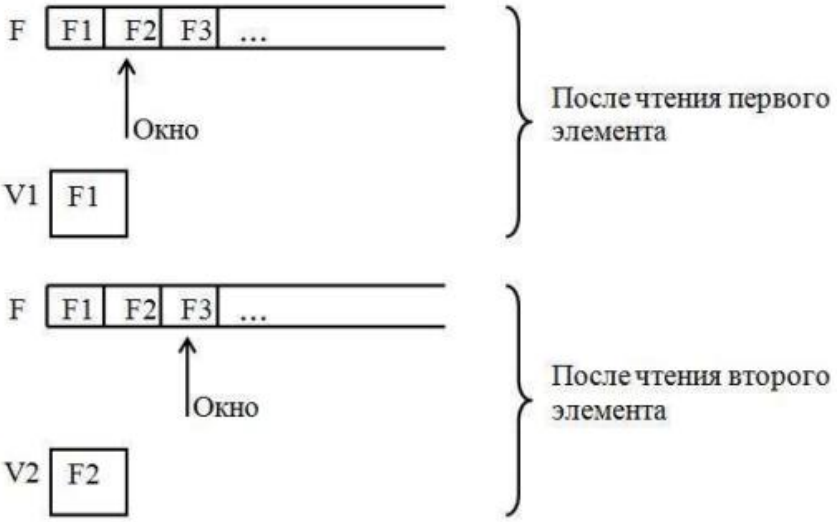
4) Процедура **Write (F, V1 [, V2, … , VN])**
Записывает в файл компоненты из соответствующих переменных.

- F - файловая переменная, открытая с помощью Reset(), Rewrite.
- Для файлов с типом



5) Процедура **Read (F, V1 [, V2, …, VN])**

- Считывает компоненты файла в соответствующие переменные.
  - F - файловая переменная, открытая с помощью Reset(), Rewrite.
  - Для файлов с типом



Файлы с типом всегда допускают как чтение, так и запись, независимо от того, были они открыты с помощью процедуры Reset или Rewrite.

Пример

20. Организация прямого доступа к элементам файлов с типом. Встроенные функции, определенные над файлами с типом. Закрытие файлов с типом. Примеры.

После Reset или Rewrite (см номер 19) типизированный файл доступен как для чтения, так и для записи

6) Функция **Eof (F)**

function Eof(var F): Boolean;

- Проверяет, является ли текущей позицией конец файла.
    - F - файловая переменная, открытая с помощью Reset(), Rewrite.

Если достигнут конец файла F (окно указывает на маркер конца файла – позицию, следующую за последней компонентой файла), или если файл пустой, то значение функции Eof равно True. В противном случае функция Eof возвращает значение False.

Если значение функции Eof равно True, то использование процедуры Read недопустимо.

Если в заголовке функции Eof опущено имя файла, то предполагается файл Input.

7) Функция *FilePos*

function FilePos(var F): Integer;

- Возвращает текущее положение в файле F;
- Если окно установлено на начало файла, то функция возвращает значение 0;
- не может использоваться для текстовых файлов.

8) Функция *FileSize*

function FileSize(var F): Integer;

- Возвращает размер файла F в компонентах;
- Не может быть использована для текстовых файлов.

9) Процедура закрытия *CloseFile*

procedure CloseFile(var F);

- Закрывает файл.

10) Seek

Procedure Seek(var F; N: LongInt)

- Перемещает указатель файла F в позицию N.

Пример:

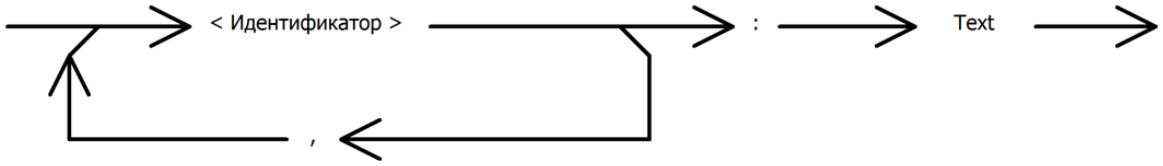
```
var
  F: File of Integer;
  Value, I: Integer;
  Size: Integer;

begin
  AssignFile(F, 'data.dat');
  Rewrite(F);

  for I := 1 to 5 do
  begin
    ReadLn(Value);
    Write(F, Value);
  end;

  Reset(F);

  while not EOF(F) do
  begin
    Read(F, Value);
    WriteLn(Value);
  end;

  Size := FileSize(F);
  WriteLn(Size);

  CloseFile(F);
end.
```

21. Текстовые файлы. Синтаксис задания. Процедуры и функции, обеспечивающие чтение из текстовых файлов, и их особенности по сравнению с файлами с типом. Допустимые типы вводимых переменных. Пример.

Текстовый файл считается последовательностью символов, интерпретируемый как текст.

- НЕ эквивалентен file of Char.

< Задание_переменных_текстового_файла > ::=



Reset

procedure Reset (var F: text);

- Открывает существующий файл.
  - Если F уже открыт, закрывает и снова открывает его.
  - Указатель файла устанавливается на его начало.
  - Для текстовых файлов - в режиме только для чтения (Read-only)

Read

procedure Read(F, V1, …, Vn);

- Считывает компоненты файла в соответствующие переменные.
  - F - файловая переменная, открытая с помощью Reset().
  - Для текстовых файлов преобразует текстовые значения к типу переменной.

В текстовый файл можно записывать данные строкового, символьного, вещественного, целочисленного и логического типа. Delphi автоматически преобразует значения в строковый формат для сохранения их в файле.

```
var
  File: Text;
  Number: Integer;
begin
  AssignFile(File, 'myfile.txt');
  Reset(File);
```

```
  Read(File, Number);

  CloseFile(File);
end.
```

22. Процедуры и функции, обеспечивающие запись в текстовые файлы, и их особенности по сравнению с файлами с типом. Допустимые типы выводимых переменных. Размещение информации в строке по умолчанию. Управление размещением информации по позициям строки. Пример.

В текстовый файл можно записывать данные строкового, символьного, вещественного, целочисленного и логического типа. Delphi автоматически преобразует значения в строковый формат для сохранения их в файле.

Процедура Append (F)

Процедура Append (F) – открывает существующий текстовый файл для добавления. Предварительно файл F должен быть связан с внешним файлом процедурой Assign.

Если не существует внешнего файла с указанным именем, то в результате выполнения процедуры Append возникает сообщение об ошибке ввода-вывода.

После выполнения процедуры Append (F) файл F становится доступным только для записи, а значение функции Eof (F) всегда будет равно True.

Процедура Append определена только для текстовых файлов.

Процедура Rewrite (F)

Отличие для текстовых файлов: файл F открывается только для записи (если текстовый файл F был открыт процедурой Rewrite, то из него читать нельзя, в него можно только писать).

После вызова процедуры Rewrite (F) значение функции Eof (F) всегда равно True.

Процедура Write ([F,] E1 [, E2, …, EN])

Отличия для текстовых файлов:

- файл должен быть открыт процедурой Rewrite или Append;

- если первый параметр (F) опущен, то подразумевается стандартный выходной текстовый файл Output. Например, Write (A, B);

- при выполнении процедуры Write осуществляется преобразование выводимого значения из типа выражения E к символьному типу. Возможны следующие типы E: символьный, арифметические (целочисленные и вещественные), строковый, массив символов, логический и их диапазоны.

· Для данных типа Char и String выводится непосредственно их значение.· Если E представляет собой арифметическое значение, то перед выводом в текстовый файл оно предварительно преобразуется из внутреннего представления в десятичную систему счисления, а затем представляется в коде ASCII (по байту на десятичную цифру).

· Если Ei имеет тип Boolean, то выводится строка True или False.

- если не указан формат вывода, то под вывод всех типов Ei отводится столько символов, сколько минимально необходимо. Для вывода вещественных типов отводится 24 позиции (тип Extended) или 17 позиций (тип Real), причем вещественное число будет выведено в виде мантиссы и порядка;

Процедура Writeln [([F][,][E1, E2, …, EN])]

Процедура Writeln [([F][,][E1, E2, …, EN])] – выполняет процедуру Write, и затем записывает маркер конца строки в файл F.

Если отсутствует список выражений Ei, то записывается только маркер конца строки.

При записи переменных в текстовый файл, они размещаются последовательно без определенных разделителей или фиксированных позиций. Это означает, что каждое значение будет следовать непосредственно за предыдущим без разделителей.

Однако, если требуется управлять размещением информации по позициям строки, то можно использовать форматирующие функции, такие как WriteLn

```
var
  F: TextFile;
  Age: Integer;
  Name: string;

begin
  AssignFile(F, 'data.txt');
  Rewrite(F);

  Age := 25;
  Name := 'John Doe';

  Write(F, 'Имя: ');
  WriteLn(F, Name);
  Write(F, 'Возраст: ');
  WriteLn(F, Age);

  CloseFile(F);
end.
```

23. Процедуры, управляющие работой буфера ввода-вывода для текстовых файлов. Пример.

Процедура SetTextBuf (F, Buf [, Size])

Процедура SetTextBuf (F, Buf [, Size]) – определяет буфер для текстового файла.

Процедуру следует вызывать после процедуры Assign, но до других процедур ввода-вывода. Здесь F – имя текстового файла, Buf – любая переменная (в качестве формального параметра используется параметр-переменная без типа), Size – необязательное выражение типа Word.

Обмен информацией между программой и внешним набором данных осущетвляется через буфер ввода-вывода. Это участок оперативной памяти. Размер стандартного буфера ввода-вывода, принятый по умолчанию, – 128 байт. Каждому открытому файлу назначается свой буфер.

Процедуры Write и Writeln записывают очередные элементы файла последовательно в буфер. После того, как буфер будет полностью заполнен, произойдет физическая запись содержимого буфера во внешний файл. После этого буфер освобождается для приема следующей порции информации.

Аналогично при чтении. Из внешнего файла одновременно считывается количество элементов, помещающееся в буфер. Процедуры Read и Readln читают элементы последовательно из буфера.

Использование буфера ввода-вывода позволяет существенно повысить скорость обмена информацией с внешними файлами (например, за счет уменьшения времени перемещения магнитных головок в дисководах).

Операции обмена данными через буфер ввода-вывода осуществляет специальный обработчик файлов (для каждого файла имеется свой обработчик файлов, он назначается при открытии файла).

Для большинства прикладных программ размер стандартного буфера ввода-вывода (128 байт) оказывается достаточным. Однако, если в программе имеется большое количество операций ввода-вывода, то более эффективным оказывается использование буфера большего размера, так как это позволяет сократить время обращения к внешним наборам данных. Процедура SetTextBuf назначает текстовому файлу F свой буфер ввода-вывода, определяемый параметром Buf. Размер буфера в байтах определяется параметром Size.

Если параметр Size опущен, то по умолчанию размер буфера принимается равным Sizeof (Buf), то есть вся область памяти, занимаемая параметром Buf, используется как буфер.

Если параметр Size не опущен, он не должен превышать размеры переменной Buf. Процедуру SetTextBuf нельзя применять к открытому файлу (она должна применяться после процедуры Assign и до процедур Reset, Rewrite или Append).

Процедура SetTextBuf определена только для текстовых файлов.

```
Var
  F: Text;
  C: Char;
  Buf: Array [1..10240] Of Char; {Буфер размером 10
                                  килобайт}
Begin
  Assign (F, 'A:\MET\Metod.txt');
  SetTextBuf (F, Buf); {Назначение буфера
    ввода-вывода Buf текстовому файлу F. Размер
    буфера равен размеру переменной Buf – 10240 байт}
  Reset (F);
While Not Eof (F) Do {Цикл чтения из файла F и записи
                                  в файл Output}
Begin
  Read (F, C);
  Writeln (C);
End;
```

Процедура Flush (F)

Процедура Flush (F) – очищает буфер текстового файла, открытого для вывода

процедурой Rewrite или Append. По данной процедуре информация из буфера, независимо от степени его заполнения (заполнен он полностью или частично), записывается во внешний файл.

Данная процедура используется редко в прикладных программах – для очень важных результатов, если нужно подтверждение о физической записи во внешний файл. Процедура определена только для текстовых файлов.

24. Сравнительная характеристика внутренней структуры представления информации в текстовом файле и файле с типом. Достоинства и недостатки использования текстового файла и файла с типом.

Очевидно, что одна и та же информация может быть записана как в текстовый файл, так и в файл с типом.

Рассмотрим внутреннюю структуру представления информации в этих файлах.

I. Представление числовой информации

   а) **В текстовом файле**

Пусть в текстовом файле записана последовательность чисел

8192, 2048, …

Внутреннее представление данной последовательности в текстовом файле представляет рисунок 5.7.

Как видно из данного рисунка, числа представлены в коде обмена информацией ASCII (в распакованном формате в коде

8421). Под каждую цифру десятичного кода числа отводится один байт. Числа отделяются друг от друга хотя бы одним пробелом.

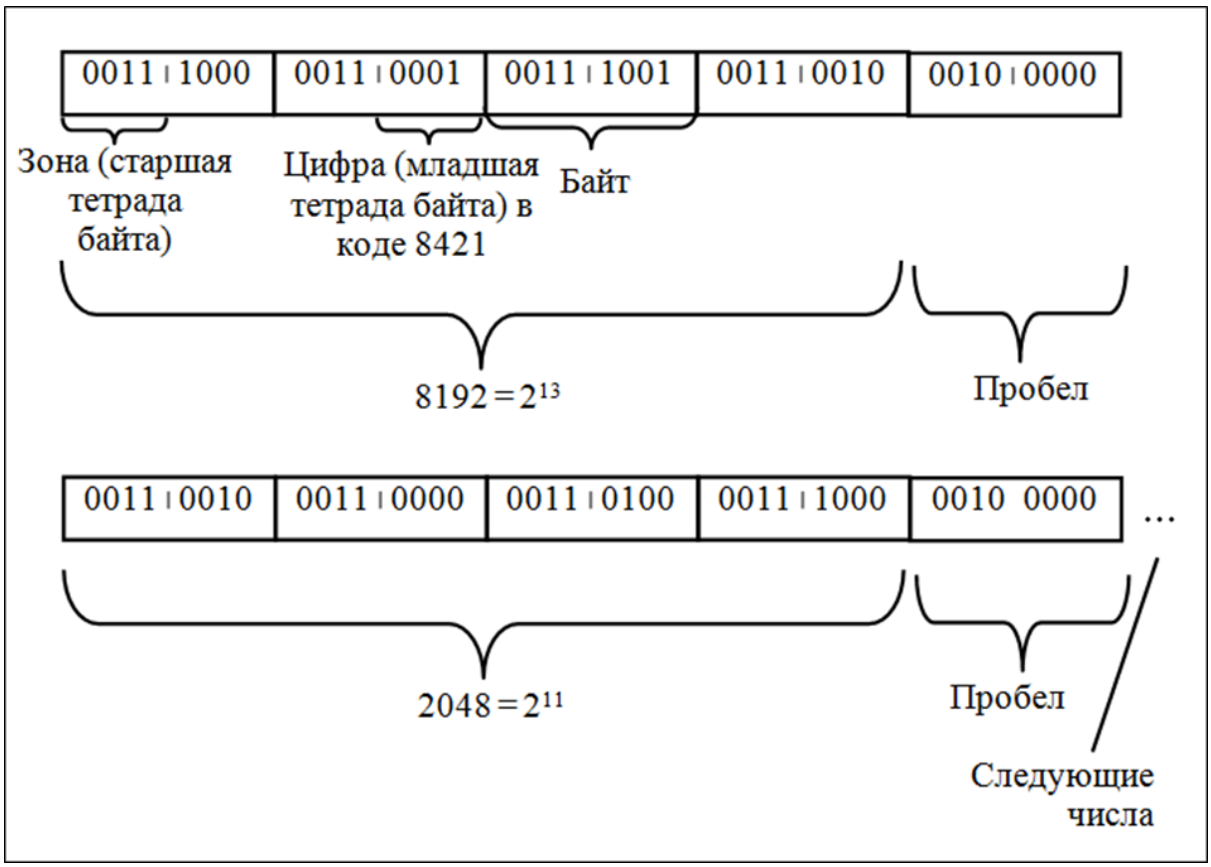


Рисунок 5.7 – Представление числовой информации в текстовом файле

Таким образом, для хранения последовательности из двух первых чисел 8192, 2048 необходимо не менее 10 байтов.

б) **В типизованном файле**

Та же последовательность чисел

8192, 2048, …

в файле типа File Of Integer имеет внутреннее представление, которое иллюстрирует рисунок 5.8.

Как видно из данного рисунка, в файле типа File Of Integer числа представлены в двоичном коде в формате Integer. Под каждое число отводится два байта. Разделители между числами отсутствуют.

Таким образом, для хранения последовательности из двух первых чисел 8192, 2048 необходимо 4 байта.

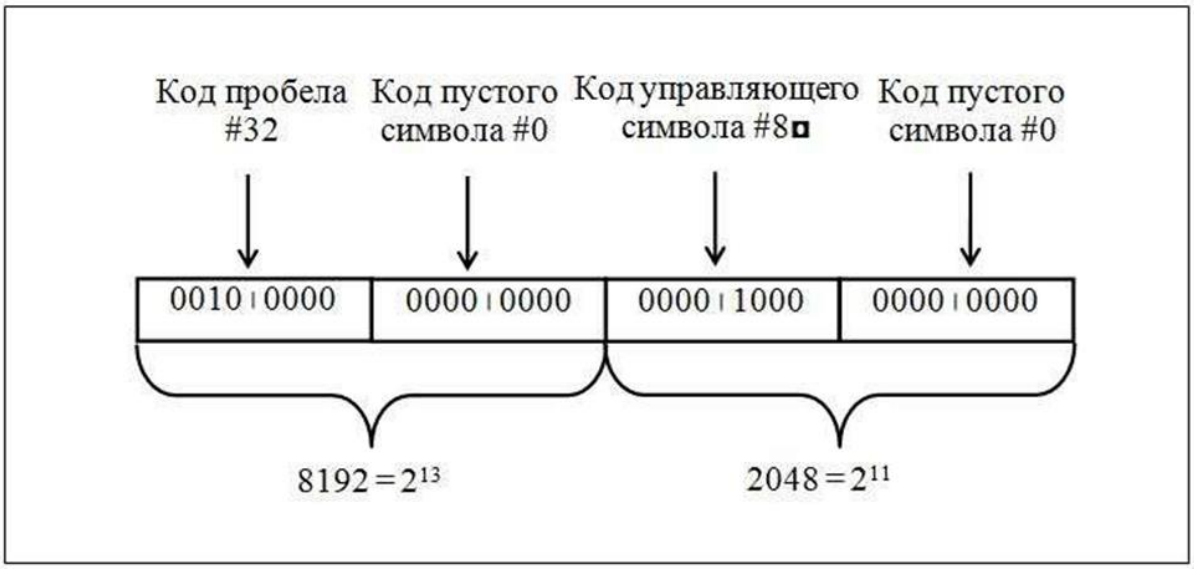


Рисунок 5.8 – Представление числовой информации в файле типа File Of Integer

При попытке вывода данных байтов файла на экран выведется пробел (не изображается), пустой символ (не изображается), символ кода ASCII с номером 8 (изображается в виде

◘), пустой символ.

Таким образом, при работе с числовой информацией, если ее не нужно выводить на экран или печать (это осуществляют лишь текстовые файлы), эффективнее использовать файлы с типом:

1) работа с ними осуществляется быстрее за счет следующих факторов:

а) отсутствует преобразование информации, она в файле представлена так же, как в памяти;

б) при работе с файлами с типом возможен режим прямого доступа;

в) меньше операций физического ввода-вывода за счет меньшего размера файла;

2) они занимают меньше места (в текстовом файле каждая цифра числа занимает байт, разделители – не менее одного пробела между числами, маркер конца строки – два управляющих символа #13#10; в файле типа File Of Integer все число занимает два байта, разделители между числами не нужны).

II. Представление текстовой информации.

а) файле.В текстовом

Пусть в текстовом файле записан текст «ВАШ ОТВЕТ НЕВЕРЕН». Пусть данный текст разбит на строки, которые изображает рисунок 5.9.

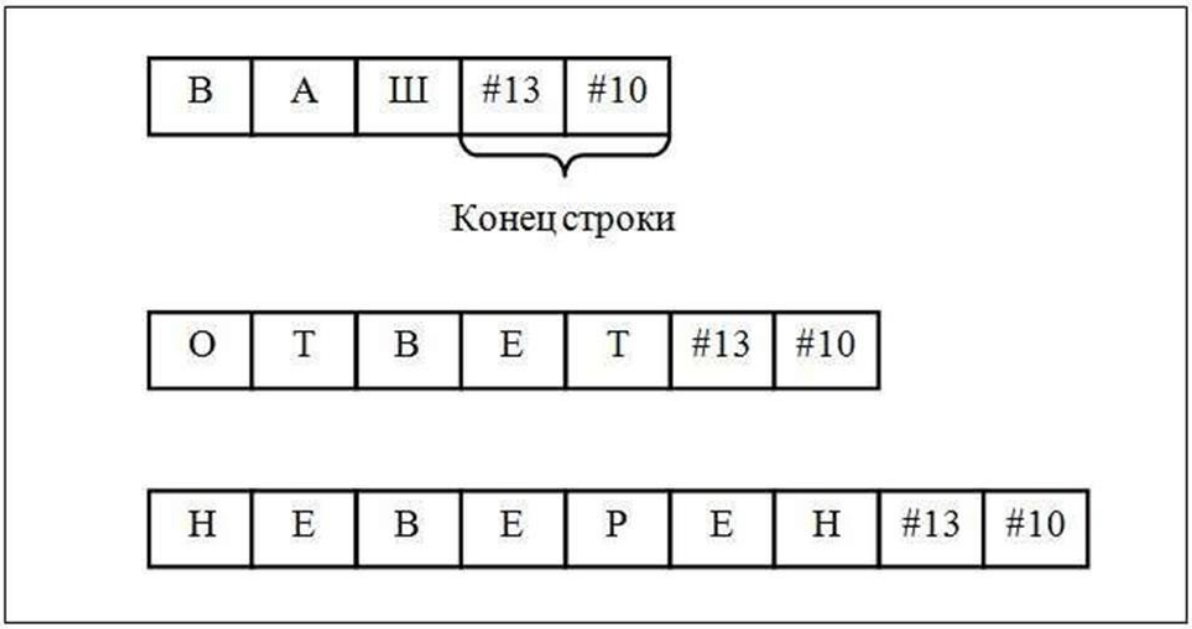


Рисунок 5.9 – Представление текстовой информации в текстовом файле

Каждая строка представляет собой одно слово, имеет текущую длину и заканчивается маркером конца строки. Каждый символ текста представлен в коде ASCII и занимает один байт.
Таким образом, для представления данной информации в текстовом файле необходим 21 байт.

б) В файле of string [7].

Тот же текст в файле типа File Of String[7] имеет внутреннее представление, которое представляет рисунок 5.10.

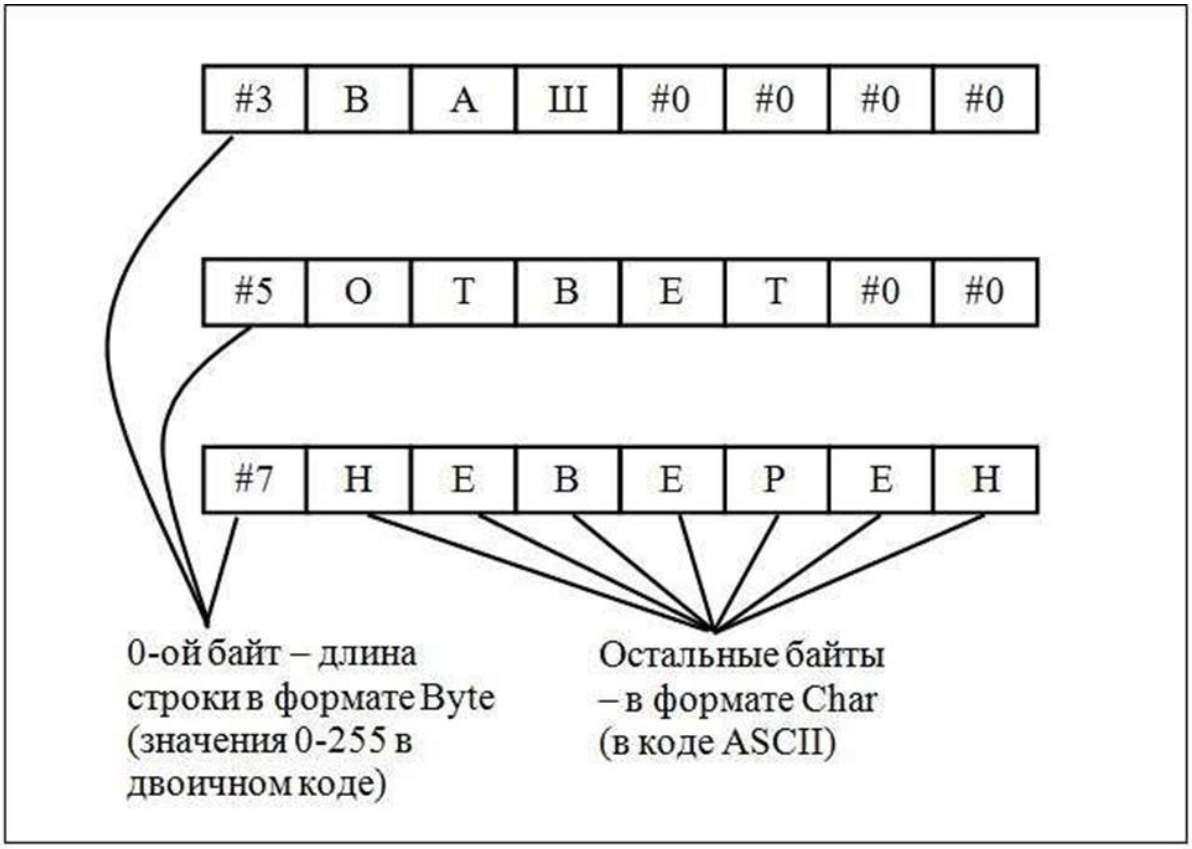


Рисунок 5.10 – Представление текстовой информации в файле типа File Of String [7]

В файле с типом строки имеют постоянную (максимальную) длину. Пустой символ (символ #0 в кодовой таблице) дополняет строки текущей длины до максимальной длины.

Следовательно, для представления данной информации в файле типа File Of String [7] необходимо 24 байта.

Сравнение файлов типа Text и File Of String показывает, что:

а) меньше места в общем случае занимает текстовый файл (так как в нем используются строки текущей длины);

б) большую скорость работы обеспечивает File Of String, так как для типизированных файлов имеется возможность работы в

режиме прямого доступа, работы одновременно в режиме записи-чтения, не нужно отслеживать управляющие символы #13#10 (маркер конца строки).

25. Файлы без типа. Синтаксис задания. Назначение. Факторы повышения скорости обмена информацией. Процедуры и функции, определенные над файлами без типа. Пример.

Файл без типа состоит из компонентов одинакового размера. Структура этих компонентов неизвестна или не имеет значения.

Такое представление позволяет стереть все различия между файлами любых типов. Любой файл, независимо от того, как он был подготовлен (текстовый файл или файл с типом), можно открыть и начать работу с ним, как с файлом без типа.

Объявление файла без типа:

```
Var

  F: File;
```

Назначение файлов без типа – максимально повысить скорость обмена информацией с внешними наборами данных. **Скорость обмена повышается за счет следующих факторов:**

1) В файлах без типа отсутствует преобразование типа компонентов;
2) Не выполняется поиск управляющих символов (типа конец строки);
3) В файлах без типа, как и в файлах с типом, возможна организация метода прямого доступа. Поэтому в них возможно одновременное использование операций чтения и записи

независимо от того, какой процедурой (Reset или Rewrite) они были открыты.

4) Обмен информацией с внешними наборами данных может быть осуществлен большими блоками.

Последний фактор является основным с точки зрения повышения скорости обмена.

Для файлов без типа определены те же процедуры и функции, что и для файлов с типом, за исключением процедур Read и Write. Определены процедуры Assign, Rewrite, Reset, Seek, Close, функции Eof, Filesize, Filepos.

1) Процедуры Reset и Rewrite

Процедуры Reset и Rewrite имеют следующие особенности.

При вызове процедур Reset и Rewrite может быть использовано два параметра. Формат вызова данных процедур имеет вид:

```
Rewrite (F [,Recsize]);

Reset (F [,Recsize]);
```

Recsize – это необязательное выражение типа Word, определяющее размер записи (в байтах), используемый при передаче данных.

Например,

```
Rewrite (F, 1);

Reset (F, 1);
```

В данном случае второй параметр определяет длину записи в 1 байт.
Если параметр Recsize опущен, то подразумевается длина записи, равная 128 байт. Это физически минимально возможный объем информации для обмена.

2) Процедура Blockread

Вместо процедур Read и Write в файлах без типа используются процедуры Blockread и Blockwrite.

Процедура Blockread имеет следующий формат вызова:

```
Blockread (F, Buf, Count [, Result]);
```

Здесь F – имя файловой переменной без типа, Buf –переменная любого типа (в качестве формального параметра используется параметр-переменная без типа), Count – выражение типа Word, определяющее количество считываемых записей.

Процедура Blockread считывает блок информации длиной в Count или меньше записей в область памяти, занимаемую переменной Buf (начиная с ее первого байта). Действительное число считанных полных записей (оно не может превышать значение Count) заносится в параметр Result (если он есть).

Если Result меньше Count, то это значит, что конец файла достигнут до полного окончания передачи. В этом случае, если параметр Result отсутствует, возникает сообщение об ошибке ввода-вывода (поэтому лучше параметр Result использовать).

В результате выполнения процедуры Blockread “окно” файла (текущая позиция файла) передвинется на число записей, равное значению Result.

Объем блока информации, считываемый процедурой Blockread в переменную Buf, занимает Result* Recsize байтов. Здесь Recsize – размер записи, определенный при открытии файла. Размер блока информации не должен

превышать 64К байта (это размер сегмента данных). В противном случае возникнет сообщение об ошибке ввода-вывода.

3) Процедура Blockwrite

Процедура Blockwrite имеет следующий формат вызова:

```
Blockwrite (F, Buf, Count [, Result])
```

Назначение параметров – то же, что и в предыдущей процедуре.

Процедура записывает одну или несколько записей (в соответствии с Count) из области памяти, занимаемой переменной Buf (начиная с ее первого байта), в файл F.

Параметр Result возвращает количество полных записанных записей. Если Result меньше Count, то это значит, что диск переполнился до завершения пересылки данных. В этом случае, если параметр Result отсутствует, возникает сообщение об ошибке ввода-вывода.

В остальном описание процедуры Blockwrite аналогично

процедуре Blockread.

Программа быстрого копирования файла F1 в файл F2.

```
Program CopyFile;
Var
    F1, F2: File;
    Buf: Array [1..2048] Of Char;
    Numr, Numw: Word;
Begin
    Assign (F1, 'DATA1');
```

```
    Assign (F2, 'DATA2');
    Reset (F1, 1); {Размер записи при передаче данных
равен одному байту}
    Rewrite (F2, 1);
    Repeat
        Blockread (F1, Buf, Sizeof (Buf), Numr);
        Blockwrite (F2, Buf, Numr, Numw);
    Until (Numr=0) Or (Numr<>Numw); {Закончился файл
F1 или переполнился диск при
создании файла F2}
    Close (F1);
    Close (F2);
End.
```

26. Проверка операций ввода-вывода. Пример.

По умолчанию при выполнении операций ввода-вывода осуществляется их стандартная проверка системными средствами. Для управления проверкой служит опция компилятора {$I} – проверка ввода-вывода, которая по умолчанию включена ({$I+}). При этом, если произошла ошибка ввода-вывода, то выполнение программы прерывается и выдается сообщение о типе ошибки.

Если нежелательно, чтобы выполнение программы прерывалось по ошибке ввода-вывода, то есть если мы сами хотим обрабатывать данные ошибки, необхотимо отключить стандартную проверку ввода-вывода. С этой целью в тексте программы перед теми операциями ввода-вывода, которые программист желает контролировать сам, необходимо отключить стандартную проверку ввода-вывода опцией {$I-}, а после этих операций снова включить ее опцией {$I+}.

Для контроля операций ввода-вывода в состоянии {$I-} служит специальная функция IOResult (без параметров).

Функция IOResult возвращает целочисленное значение типа Word, которое является состоянием последней выполненной операции ввода-вывода. Если ошибки ввода-вывода не было, то

функция возвращает значение ноль, в противном случае – код ошибки. Если опция {$I} находится в состоянии {$I-} и при некоторой операции ввода-вывода произошла ошибка, то последующие операции ввода-вывода, используемые до вызова функции IOResult, будут игнорироваться.

Таким образом, нежелательно использовать один вызов функции IOResult на несколько операций ввода-вывода, так как неясно, в какой из них произошла ошибка. Вызов функции IOResult очищает свой внутренний флаг ошибки, (возвращаемое значение устанавливается в ноль). Поэтому нельзя повторно считать одно и то же значение IOResult

```
Var
  F: File Of Char;
Begin
  Assign (F, 'PRIMER');
  {$I-}
  Reset(F);
  {$I+}
  If IOResult = 0 Then
    Writeln ('Размер файла:', Filesize (F), 'байт')
  Else
    Writeln ('Файл не обнаружен');
End.
```

27. Ссылочный тип. Назначение. Синтаксис задания. Представление в памяти. Виды указателей. Операции над указателями. Пример.

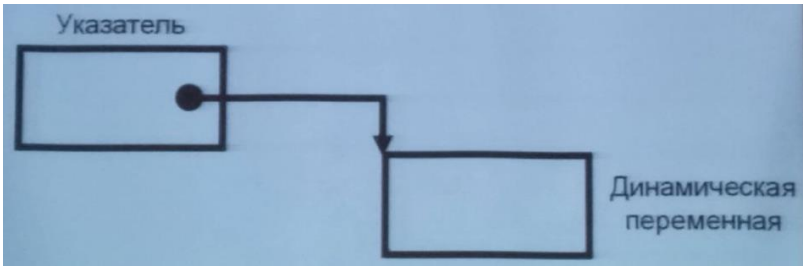
Динамическая переменная - переменная, память которой выделяется во время выполнения программы.

Все идентификаторы имеют смысл только в исходном коде.

- В скомпилированной программе есть только обращения по адресам, но не имена переменных.

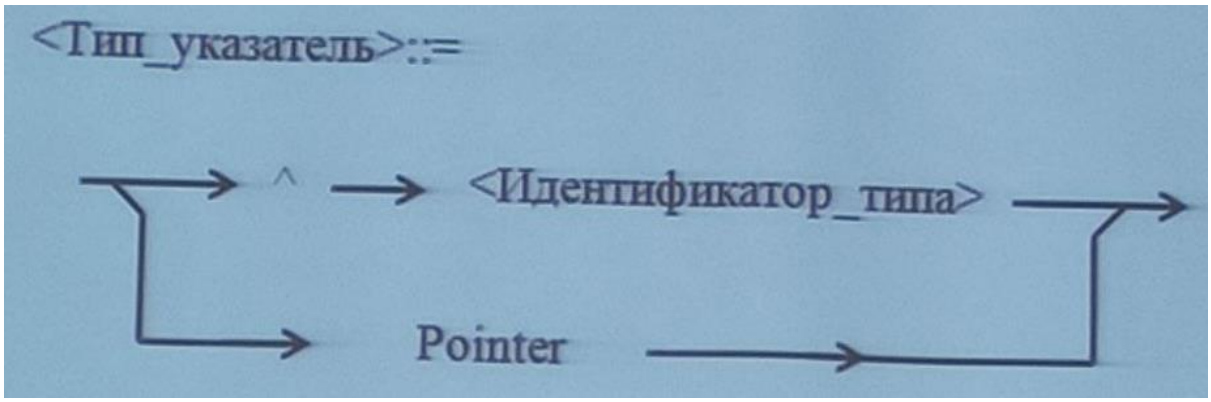
Динамические переменные не объявляются в программе.

- Единственный способ обратиться к ним по адресу.



Ссылочный тип (тип указатель):

- Значение - адреса в памяти.



Указатели:

- Типизированный;

- Не типизированный.

Ссылочные типы можно объявлять до объявления базовых типов.

```
PListItem = ^TListItem;
TListItem = record
      Data: Integer;
      Next: PListItem;
end;
```

Нулевой указатель (пустая ссылка)

Когда переменная-указатель не связана ни с какой переменной, используют специальное значение nil.

Оно может быть присвоено переменной-указателю любого типа.

Операции

1. Для получения адреса используется унарная операция @. Вместо неё можно использовать функцию Addr().
1. (2). Для доступа к динамической переменной через указатель используется унарная операция разыменования указателя ^.

Типам-указателям принято давать имена, начинающиеся с буквы P.

28. Процедуры New и Dispose. Назначение. Достоинства и недостатки их использования. Пример.

Динамическое выделение памяти сводится к 2 операциям:

- выделение блока памяти заданного размера;
- освобождение блока памяти.

Дополнительно могут быть доступны другие операции:

- изменение размера блока;
- получение размера блока;
- и др.

Стандартная библиотека Pascal/Delphi предоставляет 2 способа работы с динамической памятью:

- New/Dispose
- GetMem/FreeMem

Procedure New(var P: Pointer);
Procedure Dispose(var P: Pointer);

New() выделяет блок памяти и помещает его адрес в P.

- Размер блока равен размеру типа в объявлении переменной P.

Dispose() освобождает блок P.

```
var
  p: ^Integer;
begin
  // Выделение памяти для объекта типа Integer
  New(p);

  { КАКАЯ-ТО ЛОГИКА }

  // Освобождение памяти
  Dispose(p);
end;
```

29. Процедуры GetMem и FreeMem. Назначение. Достоинства и недостатки их использования. Пример.

Динамическое выделение памяти сводится к 2 операциям:

- выделение блока памяти заданного размера;
- освобождение блока памяти.

Дополнительно могут быть доступны другие операции:

- изменение размера блока;
- получение размера блока;
- и др.

Стандартная библиотека Pascal/Delphi предоставляет 2 способа работы с динамической памятью:

- New/Dispose
- GetMem/FreeMem

Procedure GetMem(var P: Pointer; Size: Integer);
Procedure FreeMem(var P: Pointer [; Size: Integer]);

GetMem() выделяет блок размером **Size** байт и помещает его адрес в **P**. FreeMem() освобождает блок с адресом **P**.

- Необязательный параметр Size должен иметь то же значение, что и при вызове GetMem(), [] как в РБНФ (необязательный параметр).

```
var
  p: Pointer;
begin
  // Выделение памяти размером 100 байт
  GetMem(p, 100);

  { КАКАЯ-ТА ЛОГИКА }

  // Освобождение памяти
  FreeMem(p);
end;
```

30. Строковый тип в Delphi. Представление в памяти. Автоматическое управление памятью для Delphi-строк.

Динамическая строка - последовательность символов неограниченной длины (теоретическое ограничение - 2гб). В

зависимости от присваиваемого значения строка увеличивается и сокращается динамически.

Delphi-строки - управляемый тип данных. Он совмещает в себе лучшие качества Pascal-(реальная длина строки перед самой строкой в памяти) и C- (наличие символа конца строки - ‘\0’) строк.

Это дает им следующие **преимущества:**

- *быстрота* выполнения операций над ними (не нужно всякий раз высчитывать длину строки);
- *Совместимость* с C-строками (Позволяет передавать Delphi-строки операционной системе, не выполняя дополнительных преобразований);
- *Экономия* памяти (подсчёт ссылок).

4 байта	4 байта	1	2	3	4	5	6	7
refCnt	6	‘H’	‘e’	‘l’	‘l’	‘o’	‘!’	0

Перед самой строкой располагается несколько байтов (8) с дополнительной информацией о данной строке:

- **счетчик ссылок** (refCnt) - количество переменных, ссылающихся на эту строку;
- **длина строки** - реальная длина строки.

Когда счетчик ссылок становится равным нулю - память, занимаемой строкой становится “свободной”

У строковых констант счетчик ссылок равен -1 и никогда не изменяется.

Delphi автоматически управляет памятью строк. Когда переменные выходят из области видимости, Delphi освобождает память, занимаемую этими строками.

Автоматическое управление памятью для строк в Delphi обеспечивает удобство и безопасность программирования, так как снимает с вас ответственность за управление памятью и предотвращает утечки памяти и ошибки освобождения памяти.

Когда вы выполняете операции со строками, такие как конкатенация или изменение подстроки, Delphi автоматически создает новую строку с необходимым размером памяти и освобождает старую строку. Это позволяет вам работать со строками без необходимости заботиться о деталях управления памятью.

31. Директива absolute. Принцип работы. Пример.

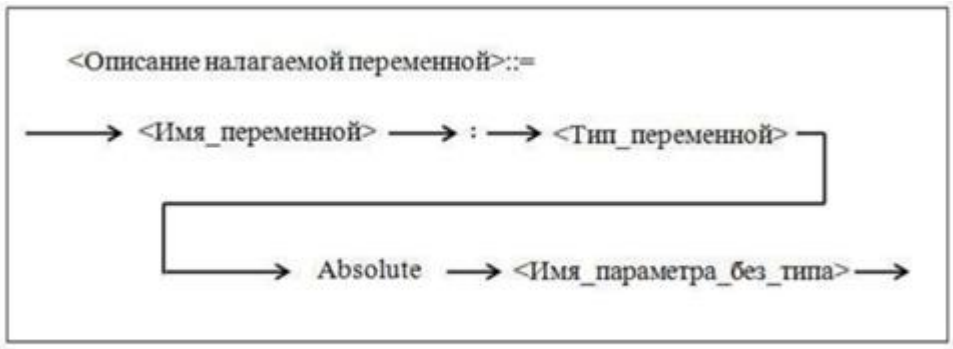


Рисунок 1.6 – Синтаксическая диаграмма описания налагаемой переменной

Позволяет создать переменную, которая будет располагаться в той же области памяти, что и какая-либо другая переменная.

Синтаксис:
```
var
    S: string[10]
    X: Integer absolute S;
```

Переменная - это не область памяти!
Переменная - это:

- идентификатор (имя);
- его связь с областью памяти.

С именем связаны:

- адрес этой области;
- ее размер в байтах.

Обычное объявление переменной заставляет компилятор:

- Выделить область памяти;
- «создать» идентификатор и связать его с этой областью памяти.

Директива absolute позволяет создавать дополнительные имена для имеющихся областей памяти.

```
Var
    X: LongWord absolute $0000:$045C
    V: array = x[0..199, 0..319] of Byte absolute $A000:$0000;
```

Оператор absolute указывает на то, что X / V будет иметь адрес, указанный после ключевого слова absolute. В данном случае адрес указан в шестнадцатеричной форме $0000:$045C / $A000:$0000, что означает, что переменная X / массив V будет располагаться по адресу 0000:045C / $A000:$0000.

Использование абсолютных адресных объявлений делает код зависимым от конкретной аппаратной архитектуры и адресного пространства. Адреса относятся к конкретным областям памяти, которые могут использоваться для доступа к аппаратному оборудованию или видеопамяти. Поэтому код, использующий абсолютные адресные объявления, может не работать на других системах или архитектурах, где адресное пространство отличается.

32. Зарезервированное слово packed. Влияние на представление в памяти записей и массивов.

Процессор:

- обращается к памяти по адресам, кратным разрядности платформы;
- считывает/записывает за одно обращение количество данных, равное разрядности платформы.

Чтение 1 байта по адресу 007F0001:

- чтение 4 байт по адресу 007F0000;
- лишние байты не используются.

Чтение 4 байт по адресу 007F0001:

- чтение 4 байт по адресу 007F0000;
- чтение 4 байт по адресу 007F0004;
- объединение нужных байтов (3+1);
- лишние байты не используются.

Выравнивание - способ размещения переменных В памяти, позволяющий минимизировать количество обращений к памяти.

В общем случае переменные следует выравнивать по адресам, кратным их размеру.

По умолчанию компилятор «выравнивает» поля всех записей.

- При этом некоторые байты могут оставаться неиспользуемыми
  (только для выравнивания).
- Директива ($A) позволяет управлять выравниванием.

Зарезервированное слово packed заставляет компилятор располагать поля записи вплотную друг к другу.

Бывает полезно при работе с файлами сложных форматов.

Зарезервированное слово packed также может применяться для массивов.