«Основы алгоритмизации и программирования (часть 2)»

Вопросы к экзамену

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

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

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

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

5. Параметры open array. Назначение, синтаксис. Механизм организации взаимосвязи с фактическими параметрами при использовании параметров open array. Совместимость с фактическими параметрами. Пример.

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

7. Процедурный тип. Назначение, синтаксис. Условия совместимости с фактическими параметрами подпрограмм. Пример.

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

9. Директивы подпрограмм. Неявная рекурсия. Пример.

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

11. Особенности работы с библиотечными модулями пользователя. Пример.

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

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

14. Оператор присоединения. Назначение. Формат. Полная и сокращённая формы оператора присоединения. Примеры использования.

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

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

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

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

19. Процедура Assign/AssignFile. Назначение. Формат. Абсолютные и относительные пути к файлам. Логические имена устройств ввода-вывода. Пример.

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

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

22. 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.

23. 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.

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

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

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

27. Проверка операций ввода-вывода. Пример.

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

29. Процедуры New и Dispose, GetMem и FreeMem. Назначение. Сравнительная характеристика (достоинства и недостатки). 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 разных мест программы.

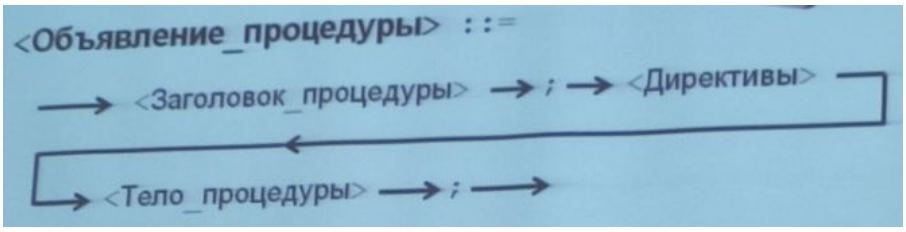
Структура подпрограммы:

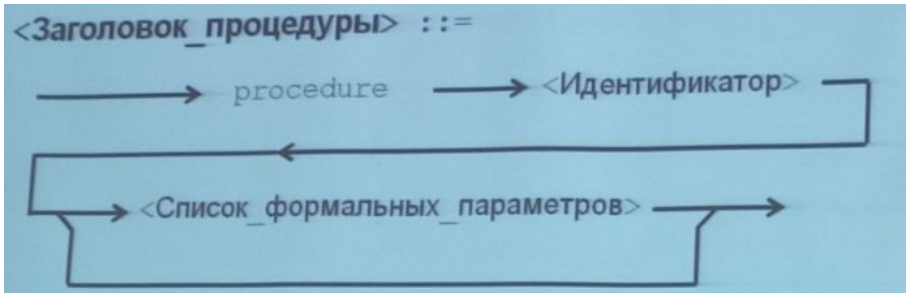
1) заголовок подпрограммы;
2) тело подпрограммы:
   - раздел описаний;
   - раздел операторов.

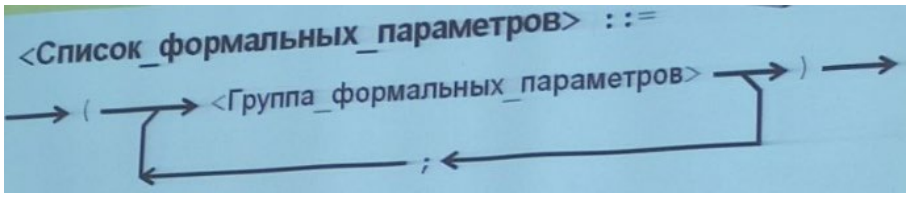
Подпрограммы *делятся на*:

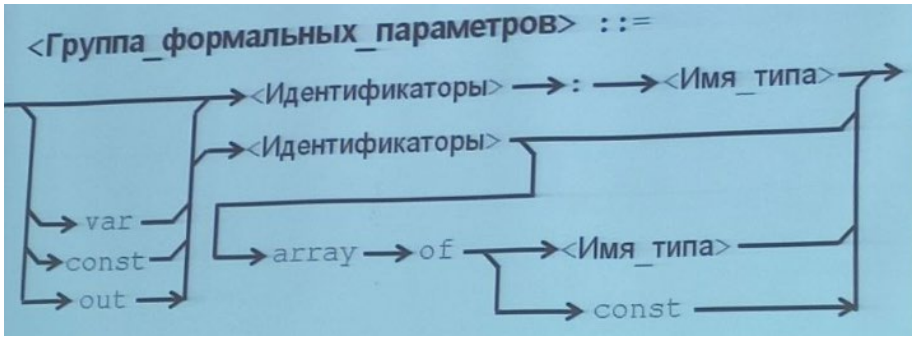
- процедуры;
- функции.

Описание процедур:

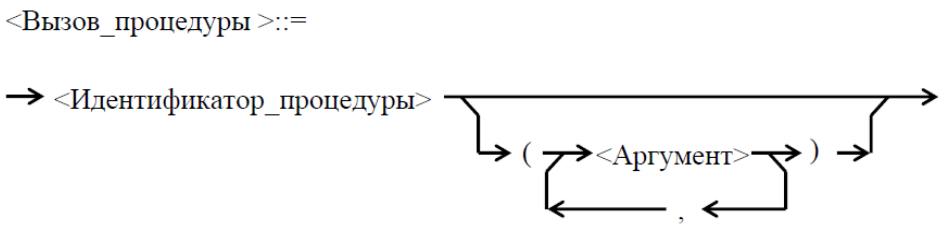








Вызов процедур:



Процедуры без параметров:

- редко используются:
  - большинству подпрограмм нужны исходные данные для работы;
  - передача подпрограмме исходных данных через глобальные параметры – дурной тон:
    - подпрограмму сложно использовать в другой программе;
    - если несколько подпрограмм используют одну и ту же глобальную переменную, логика программы усложняется;
    - минимальная гибкость по сравнению с другими способами передачи параметров.

Пример:

```
var
  X1, X2, Y1, Y2, D: Real; // глобальные переменные
...
procedure CalcDist();
begin
  D := Sqrt(Sqr(X1 - X2) + Sqr(Y1 - Y2));
end;
```

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

Параметры:

- формальные:
  - *формальный параметр* – переменная, которая будет доступна внутри подпрограммы;
- фактические:
  - *фактический параметр* – то, что передаётся при вызове.

Виды параметров:

- в языке Pascal:
  - параметры-значения;
  - параметры-переменные;
  - параметры-константы;
  - параметры без типа;
- в языке Delphi:
  - есть и другие способы передачи параметров.

Для передачи параметров используется программный стек.

Параметры-значения:

- для параметров-значений при вызове создаётся локальная переменная:
  - существует только во время выполнения подпрограммы;
  - при каждом вызове это *новая переменная*;
- в заголовке подпрограммы записывается без модификаторов, указывается только имя и тип данных;
- 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-значение.

Пример:

```
var
  D: Real; // глобальная переменная
...
procedure CalcDist(X1, X2, Y1, Y2: Real); // параметры-значения
begin
  D := Sqrt(Sqr(X1 – X2) + Sqr(Y1 – Y2));
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 подпрограммы записывается с зарезервированным словом **`var`**;
- фактическим параметром может быть *только переменная* того же типа данных, что и формальный параметр;
- в стек помещается адрес переменной-параметра;
- изменение формального параметра внутри подпрограммы приводит к изменению фактического параметра.

Пример:

```
procedure CalcDist(X1, X2, Y1, Y2: Real; var D: Real); // параметры-значения и параметр-
                                                          переменная для возврата значения
begin
  D := Sqrt(Sqr(X1 – X2) + Sqr(Y1 – Y2));
end;
```

Параметры-константы:

- в заголовке подпрограммы записывается с зарезервированным словом **`const`**;
- фактическим параметром может быть выражение того же типа данных, что и формальный параметр;
- компилятор запрещает изменение значения внутри подпрограммы;
- способ передачи значения *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.

Пример:

```
procedure CalcDist(const X1, X2, Y1, Y2: Real; var D: Real); // параметры-константы и
                                                              параметр-переменная для
                                                              возврата значения
begin
  D := Sqrt(Sqr(X1 – X2) + Sqr(Y1 – Y2));
end;
```

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

Параметры без типа:

– для всех параметров-значений тип данных должен быть указан;
– для **var**- и **const**-параметров указание типа необязательно:
    - параметры, у которых тип данных не указан, называются *параметрами без типа* (**untyped**).

Обеспечение совместимости с фактическими параметрами может быть достигнуто одним из *двух способов*:

1. Внутри подпрограммы объявляется локальная переменная нужного типа, налагаемая на переданный фактический параметр. Для описания такой переменной используется директива **absolute**.
   Пример:

```
procedure CopyData(const Source; var Dest; Count: Integer);
var
  S: array[0..MaxInt – 1] of Byte absolute Source;
  D: array[0..MaxInt – 1] of Byte absolute Dest;
  I: Integer;
begin
  for I := 0 to Count – 1 do
    D[I] := S[I];
end;
```

2. Для обеспечения совместимости с фактическими параметрами внутри подпрограммы вводится нужный тип. Данный тип ставится в соответствие параметру без типа с помощью присваивания типа переменной.
   Приведение типов может быть:
   – неявным (автоматическим);
   – явным (явным указанием типа).

Пример:

```
procedure CopyData(const Source; var Dest; Count: Integer);
type
  TBytes = array[0..MaxInt – 1] of Byte;
var
  I: Integer;
begin
  for I := 0 to Count – 1 do
    TBytes(Dest)[I] := TBytes(Source)[I]; // явное приведение типов
end;
```

5. Параметры open array. Назначение, синтаксис. Механизм организации взаимосвязи с фактическими параметрами при использовании параметров open array. Совместимость с фактическими параметрами. Пример.

В классическом Pascal в заголовке подпрограммы разрешено указывать только идентификаторы типов (по синтаксису).

В Delphi введён специальный синтаксис для передачи массивов произвольного размера.

Объявление open array параметров:

– вместо имени типа записывается конструкция **`array of <Тип>`**:
Пример:

```
procedure ZeroItems(var A: array of Integer);
begin
...
end;
```

! это **НЕ параметр** типа «динамический массив» **!**

При вызове подпрограммы с параметром(-ами) типа open array:

– для передачи в стек помещается 2 величины:
- адрес массива;
- количество элементов в нём;
- если используется **open array constructor**, то массив формируется в стеке «на лету»:

```
procedure Sum(const A: array of Integer; var S: Integer);
var
  I: Integer;
begin
  S := 0;
  for I := Low(A) to High(A) do
    S := S + A[I];
end;
...
  Sum([1, 10, X + 42], Res);
```

– фактическим параметром может быть:
- статический массив с тем же базовым типом;

- динамический массив с тем же базовым типом;

– формальный параметр ведёт себя как обычный *статический массив*, но с некоторыми ограничениями:
  - нумерация всегда начинается с 0;
  - работать можно только с отдельными элементами, но не с полной переменной;
  - в другие подпрограммы могут быть переданы только как:
    - open array parameters;
    - параметры без типа.

Пример:

```
procedure ZeroItems(var A: array of Integer);
var
  I: Integer;
begin
  for I := Low(A) to High(A) do
    A[I] := 0;
end;
```

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

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

Структура подпрограммы:

1) заголовок подпрограммы;
2) тело подпрограммы:
   – раздел описаний;
   – раздел операторов.

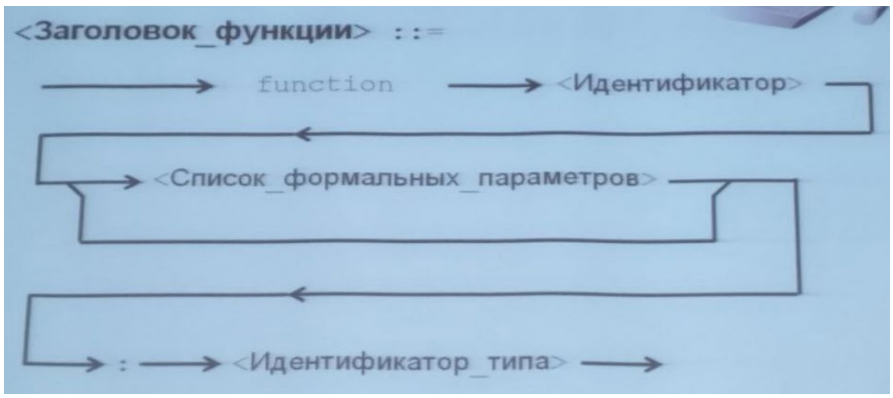
Подпрограммы *делятся на*:

– процедуры;
– функции.

Функция – процедура, возвращающая значение.

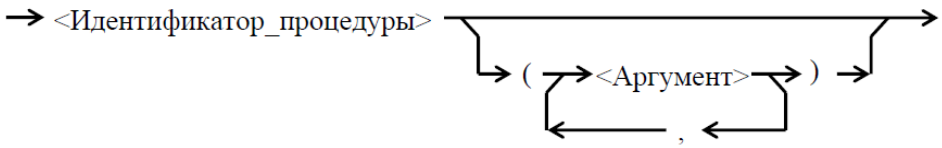
Функции аналогичны процедурам, за исключением некоторых отличий.

Объявление функций:



Вызов функций:

<Вызов_процедуры >::=



!!! ЗАМЕНИТЬ ПРОЦЕДУРУ НА ФУНКЦИЮ !!!

Функции могут использовать те же механизмы передачи параметров и результатов, что и процедуры.

Кроме того, каждая функция вычисляет так называемое **возвращаемое значение функции:**

- тип возвращаемого значения:
  - может быть:
    - стандартным типом;
    - предварительно описанным типом;
  - не может быть:
    - заданием типа;
- возвращаемое значение:
  - 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 то же функции;
  - если возвращаемое значение функции не было присвоено, оно считается *неопределённым*.

Функции, в отличие от процедур, могут использоваться в составе выражений: сам вызов функции является выражением, его тип – тип возвращаемого значения функции.

7. Процедурный тип. Назначение, синтаксис. Условия совместимости с фактическими параметрами подпрограмм. 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.

Пример:

```
type
  TSomeProc = procedure(X: Integer; var Y: Real);
  TSomeFunc = function(S: String): Integer;
...
var
  MyFunc: TSomeFunc;
  MyProc: TSomeProc;
  L: Integer;
...
  MyFunc := MyCoolLength;
  L := MyFunc('Parameter');
```

! скобки при использовании подпрограммы – *однозначный* вызов:

1)

```
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
```

2)

```
if F = G then // переменные процедурного типа в выражении – вызов соответствующей функции
  WriteLn('Equal')
else
  WriteLn('Not equal');

if @F = @G then // сравнение адресов функций
  WriteLn('Equal')
else
  WriteLn('Not equal');
```

Ограничения процедурных типов:

- нельзя присваивать адреса стандартных процедур/функций;
- нельзя присваивать адреса процедур/функций, объявленных с директивой **`inline`**;
- процедура/функция должна точно соответствовать процедурному типу.

Применение процедурного типа:

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

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

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

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

Классический пример – вычисление факториала:

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

Реализация на Delphi:

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

При каждом рекурсивном вызове (активации рекурсивной подпрограммы) создаётся 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:

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

Виды рекурсии:

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

Преимущество рекурсии: как правило, запись такой программы короче и нагляднее.

Недостаток рекурсии: как правило, такая программа выполняется медленнее, чем эквивалентная ей нерекурсивная.

9. Директивы подпрограмм. Неявная рекурсия. Пример.

Директива `inline`.

В некоторых реализациях Pascal:

- позволяла вручную задать машинный код для подпрограммы;
- синтаксис был аналогичен вызову процедуры.

В ранних версиях Delphi поддержка удалена, но начиная с Delphi 2007 директива введена с синтаксисом директивы для подпрограмм.

- 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*.

Преимущество: быстрее, нет затрат на вызов и возвращение значения.

Недостаток: имеет смысл только для маленьких, часто вызываемых подпрограмм.

Пример:

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

Директива `forward`.

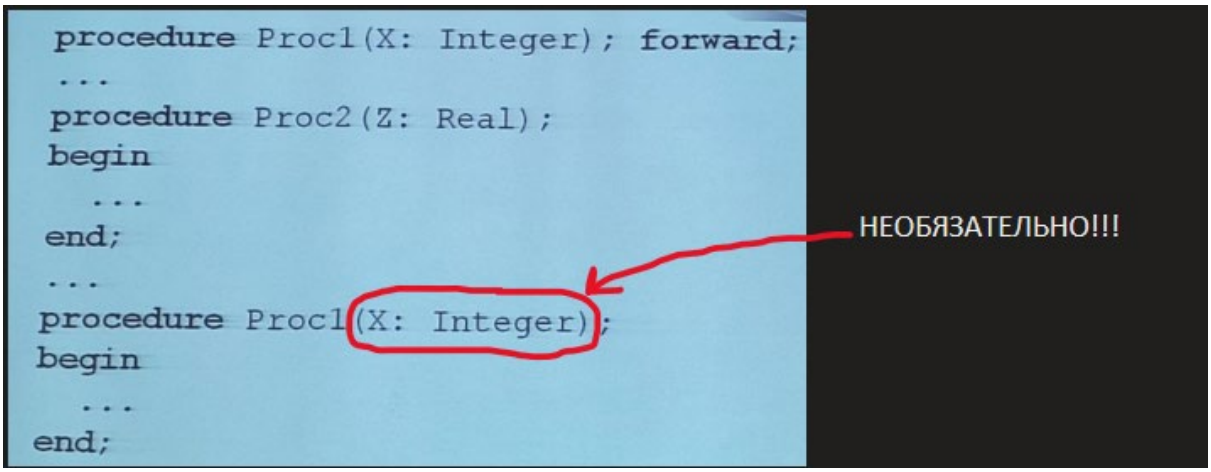
Описание подпрограммы:

- опережающее:
  - заголовок подпрограммы;

- директива **forward**;

– определяющее:
  - заголовок подпрограммы;
  - блок (тело подпрограммы).

Пример:



Неявная рекурсия.

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

Взаимная рекурсия:

– при организации взаимной рекурсии возникает проблема: вызов подпрограмм возможен только после ее описания;
– проблема решается с помощью опережающих описаний (директива **forward**).

Пример:

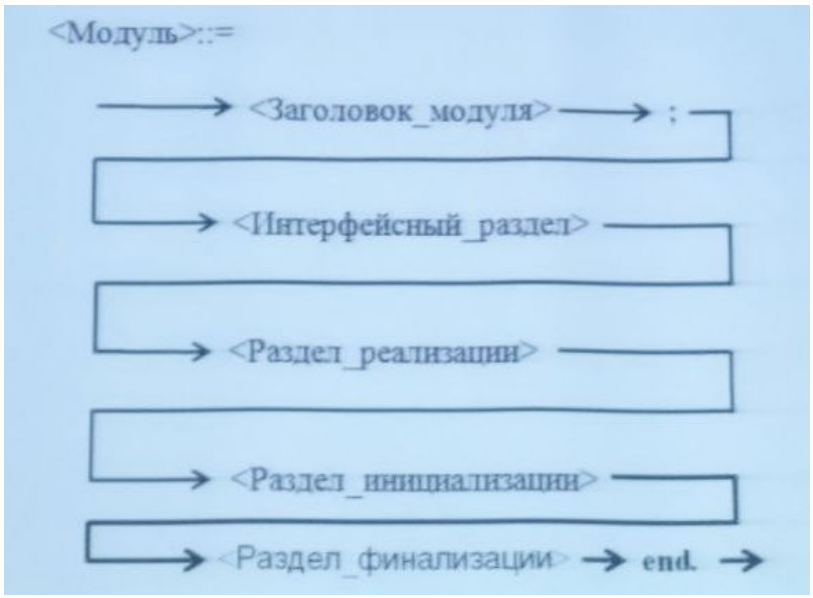
```
procedure ProcedureB(ACounter: Integer); forward;

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;
```

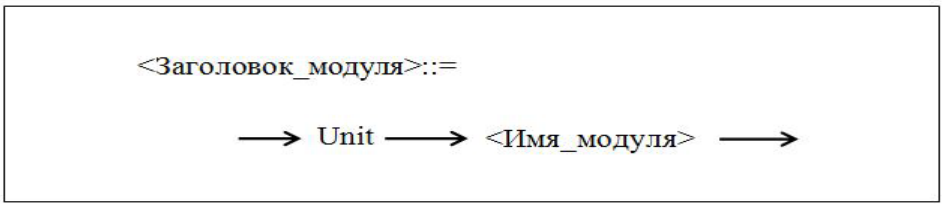
10. Библиотечные модули пользователя. Назначение модуля. Структура модуля. Синтаксис и назначение разделов модуля. Пример.

Файлы в Delphi:

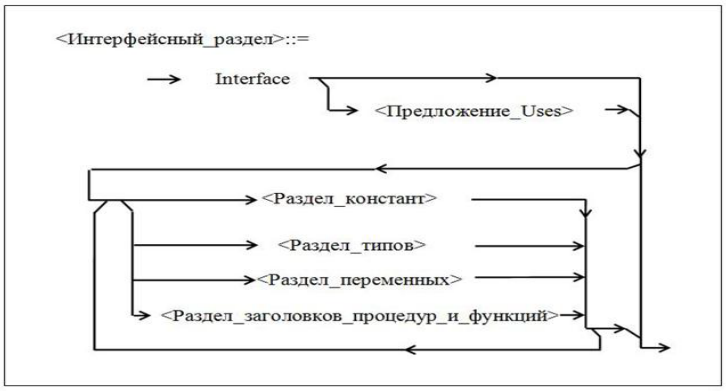
- файл проекта (.dpr) - программный модуль;
- исполняемый файл (.exe) — результат компиляции проекта;
- вспомогательные модули (.pas);
- скомпилированные модули (.dcu).

Вспомогательный модуль:





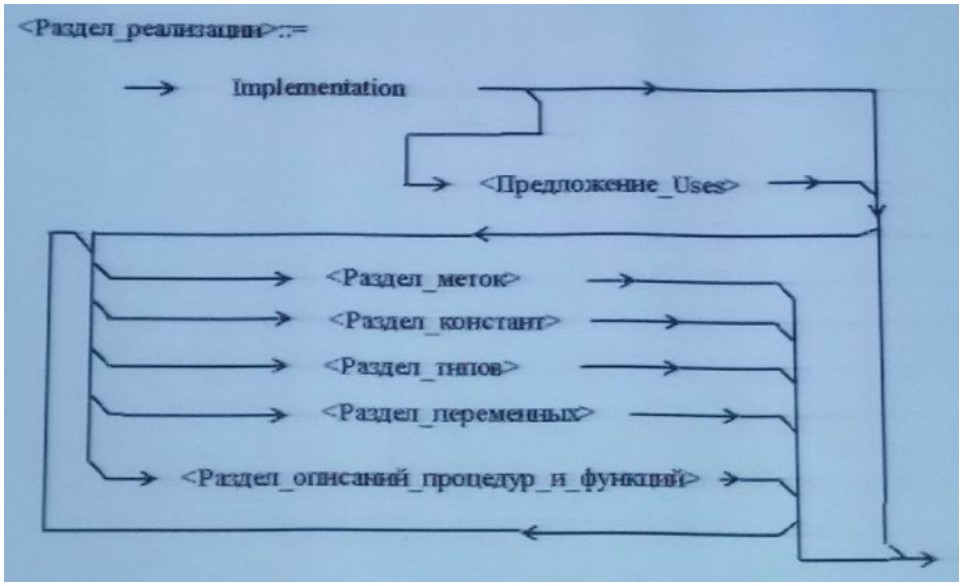
Секция `interface` (интерфейсный раздел):



- описываются внешние элементы – глобальные типы, константы, переменные, процедуры и функции – которые доступны основной программе или другому модулю;

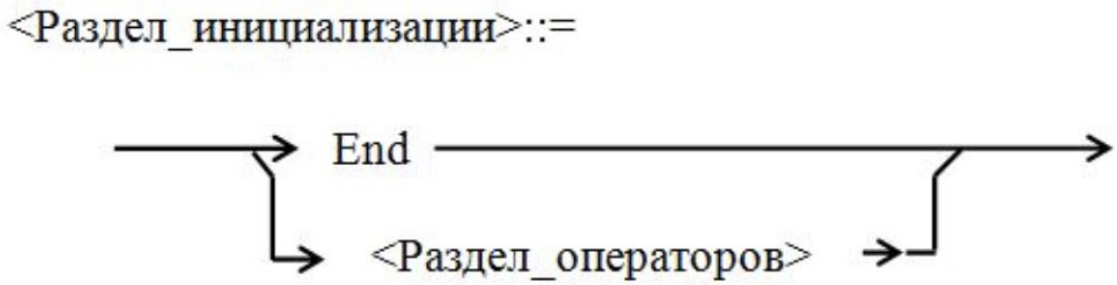
– описания могут следовать в любом порядке (как и в программном модуле);
– если при объявлении типов, данных или подпрограмм используются элементы, введённые в других модулях, то эти модули должны быть перечислены в директиве `uses` сразу после слова `interface`;
– в директиве `uses` рекомендуется указывать только те модули, которые необходимы именно в этой секции.

Секция `implementation` (раздел реализации):



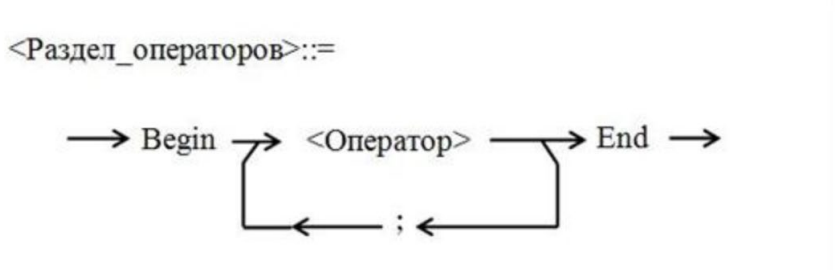
– описываются локальные элементы (типы, константы, переменные, метки), содержатся тела процедур и функций, заголовки которых описаны в секции `interface`;
– все элементы, описанные в интерфейсной секции, доступны в секции реализации;
– всё, что описано в секции реализации, недоступно программе или другим модулям, это *скрытая* часть модуля;
– описанные 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 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`.

Секции инициализации:



- содержатся операторы, которые выполняются до операторов из тела программы (т.е. до главного программного модуля, обычно это установки некоторых стартовых значений);
- если программный модуль использует несколько модулей `unit`, то их разделы инициализации будут выполняться в порядке их перечисления в предложении `uses` программы до операторов тела программы;
- если операторная часть в секции инициализации отсутствует, то секция состоит из слова `end`;
- все разделы модуля являются необязательными, но служебные слова, начинающие разделы, должны присутствовать обязательно.

Раздел операторов:



Структура пустого модуля. Модуль содержит только служебные слова, которые должны присутствовать в `unit` обязательно:

```
unit Hollow;
  interface
  implementation
  {
  initialization
                }— необязательно
  finalization
  }
End.
```

11. Особенности работы с библиотечными модулями пользователя. Пример.

Особенности работы с модулями:

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:
   - в случае существования одинаковых идентификаторов в разных модулях, чтобы обеспечить доступ к содержимому нужного модуля, используются составные имена, в первой части которых указывается имя модуля:

     ```
     U1.Ned, U1.D, U1.SetNed(X)
     ```

   - если вводят одинаковые идентификаторы, то необходимо учитывать порядок их подключения или гарантировать корректность обращения, указывая явно принадлежность библиотечных подпрограмм, данных или типов к конкретному модулю.
3. решение проблемы закольцованности (циклических ссылок модулей друг на друга) зависит от того, в каком разделе возникла закольцованность:
   - если оба модуля подключают друг друга в разделе реализации:

     ```
     unit U1;
       interface
       implementation
         uses U2;
         …

     unit U2;
       interface
       implementation
         uses U1;
         …
     ```

     то закольцованность автоматически разрешается компилятором, поскольку Delphi для обоих модулей может выполнять полную компиляцию интерфейсных секций (а описания в интерфейсных секциях аналогичны опережающему описанию процедур и функций с помощью директивы `forward`);
   - если хотя бы один из модулей подключает другой в разделе `interface`, то проблема закольцованности может быть решена только программным путём. В этом случае обычно используется третий модуль:
     - в него помещаются все типы, переменные или подпрограммы, которые ссылаются друг на друга в первых двух модулях;

- затем они удаляются из первых двух модулей, и к данным модулям подключается третий модуль, независящий от своего предложения `uses`.

Достоинства использования модулей Unit:

- 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;
- модули компилируются независимо друг от друга, при подключении модуля к другой программе он не компилируется заново: сокращается время компиляции больших программ;
- наличие модулей позволяет создавать большие программы с суммарным размером исполнимого кода большим, чем 64К.

Директива `{$I}`:

- позволяет подставить в текст модуля содержимое внешнего текстового файла:

```
…
{$I SomeFile.inc}
…
```

- включаемый файл должен удовлетворять условиям:
  - при его включении на место директивы `{$I …}` он должен вписаться в структуру и смысл программы без ошибок;
  - он должен содержать законченный смысловой фрагмент, то есть блок от `begin` до `end` должен храниться целиком в одном файле;
  - включаемый файл не может быть указан в середине раздела операторов;
- включаемые файлы сами могут содержать директивы `{$I …}`: максимальный уровень такой вложенности равен восьми;
- недостатки такого подключения к программе внешнего файла по сравнению с использованием библиотечных модулей:
  - подключаемые файлы каждый раз компилируются заново, что увеличивает время компиляции;
  - размер программы не может превышать 64К.

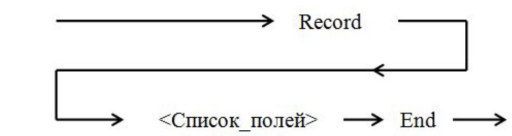
12. Записи. Синтаксис задания. Записи без вариантной части. Операции над записями и над полями. Пример.

Запись – структура данных, состоящая из упорядоченных разнородных компонентов. Компоненты записи – поля.

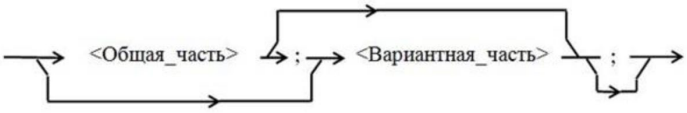
Другие названия:

- комбинированный тип;
- тип record;
- структура (в С-подобных языках).

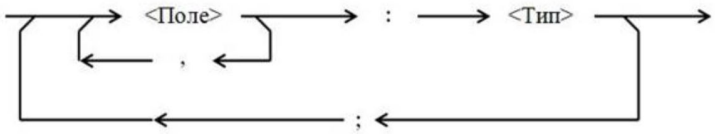
<Тип_Record>::=



<Список_полей>::=



<Общая_часть>::=



Пример:

```
type
  TComplex = record
    Re: Real;
    Im: Real;
   end;
```

что эквивалентно:

```
type
  TComplex = record
    Re, Im: Real;
   end;
```

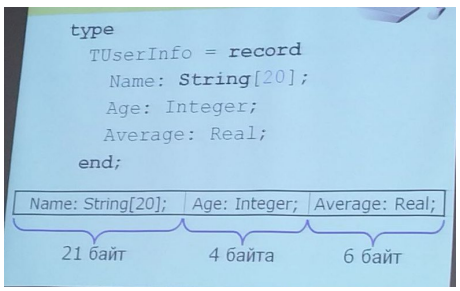
В разделе `var` необходимо ввести переменные типа `TComplex`:

```
var
    X, Y: TComplex;
```

Тип поля записи может быть определен двумя способами:

- непосредственно задан в описании записи;
- описан предварительно: указывается имя типа.

Представление записи в памяти:



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

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

Обращение к значению поля осуществляется с помощью идентификатора полной переменной, идентификаторов всех полей (с учетом их иерархии), в состав которых входит поле, и имени данного поля, разделенных точкой. Такое имя называется **составным именем**:

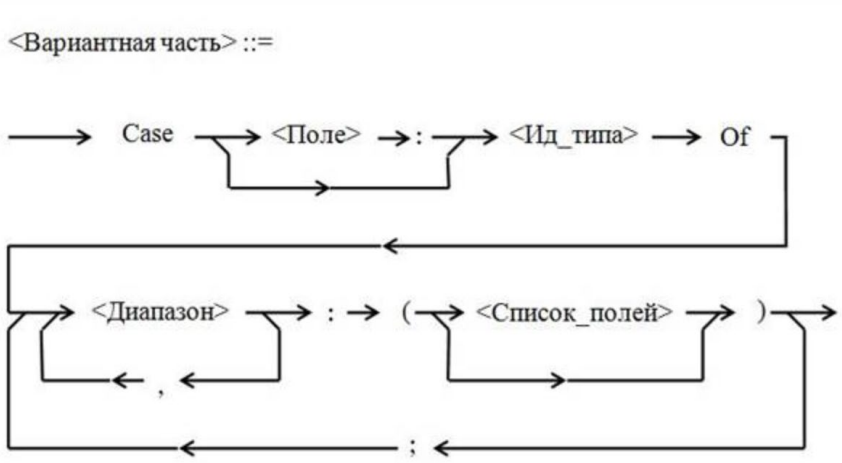
```
X.Re := 5;
```

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.

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



Пример:

```
type
  TEmployee = record
    FirstName, LasName: String[40];
    BirthDate: TDate;
    case Salaried: Boolean of
      True: (AnnualSalary: Currency);
      False: (HourlyWage: Currency);
  end;
```

Свойства вариантной части:

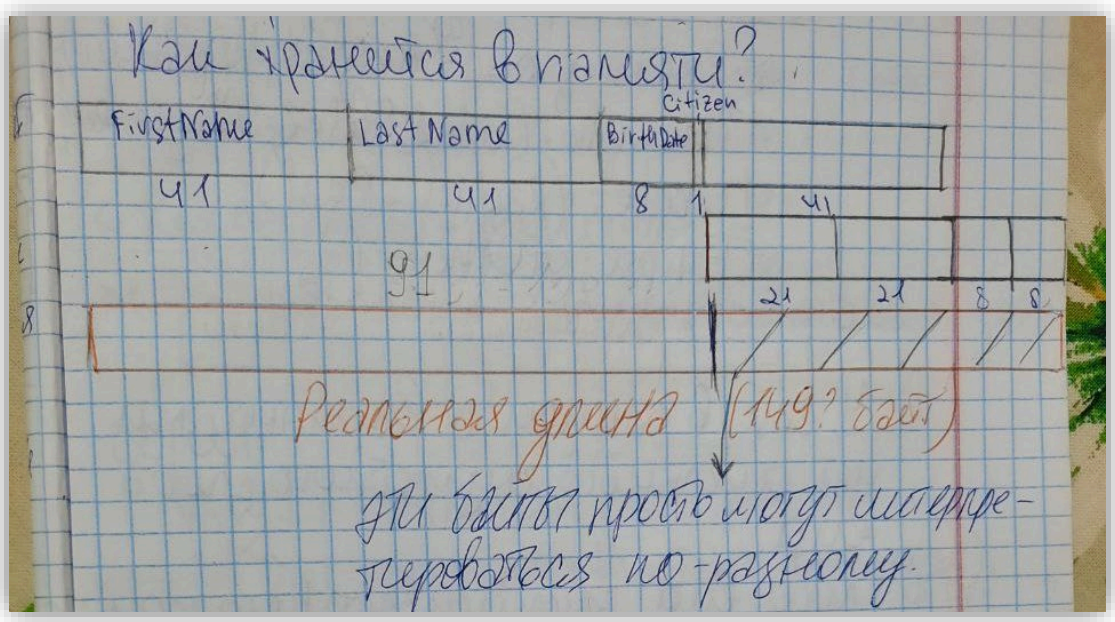
- поля каждого варианта занимают одну и ту же область памяти:
  - размер этой области памяти определяется по наибольшему варианту;
- поле, объявленное в `case`, – поле признака:
  - поле признака относится к общей части записи;

– никаких дополнительных проверок в зависимости от значения поля признака не производится;

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

```
type
  TPerson = record
      FirstName, Name: String[40];
      BirthDate: TDate;
      case Citizen Boolean of
          True: (Birthplace: String[40]);
          False: (Country: String[20];
                  EntryPort: String[20];
                  EntryDate, ExitDate: TDate);
  end;
```

Представление в памяти:



Особенности использования вариантной части:

– запись может не иметь общей части:
  - в этом случае вместо поля признака указывается только имя любого перенумерованного типа (`case Integer of` ...);

– вариантная часть может быть пустой;

– 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.

14. Оператор присоединения. Назначение. Формат. Полная и сокращённая формы оператора присоединения. Примеры использования.

Назначение:

- облегчает доступ к полям записей типа record 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`, следовательно, любые модификации переменных, влияющие на вычисленное значение адреса, до завершения оператора `with` не отражаются на значении вычисленного ранее адреса.

Пример:

```
type
  TMonth = (Jan, Feb, ..., Dec);
  TBirthDate = record
    Day: Integer;
    Month: TMonth;
    Year: Integer;
  end;
...
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:

```
unit MyUnit
…
var
  X: TBirthDate;
  Year: Integer;
…
  with X do
    MyUnit.Year := 2018; // Чтобы обратиться к глобальной переменной Year нужно писать так
                         // Однако, не должно существовать X.MyUnit.Year
```

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

Множественный тип (set) соответствует понятию множества в математике - неупорядоченный набор значений:

- максимальное количество элементов – 256;
- все элементы должны быть одного и того же типа – базового типа множества.

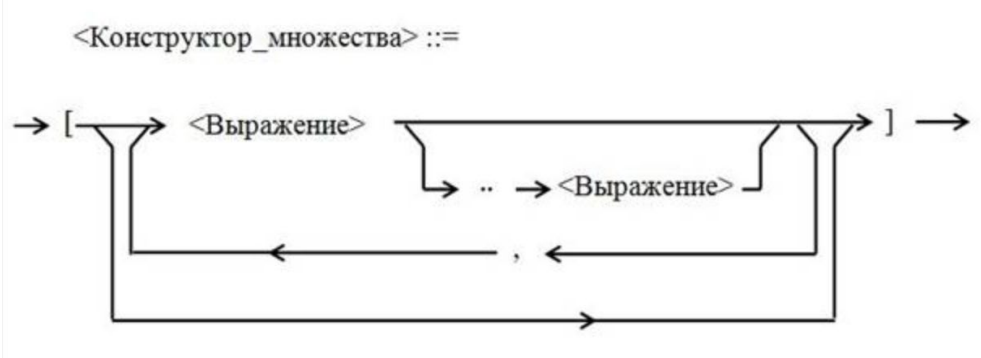
Базовый тип – любой скалярный тип, кроме вещественных, а для остальных типов есть ограничения:

- 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 включён во множество;
  - если бит содержит 0,элемент не входит во множество.

Конструктор множества:



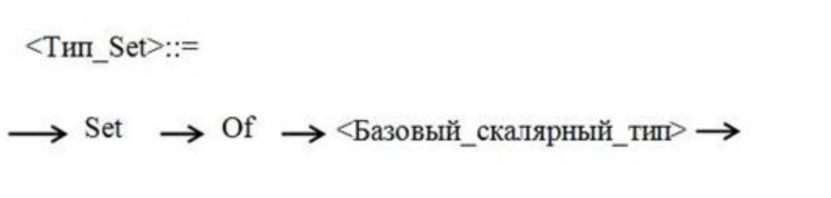
Конструктор множества – это список разделенных запятыми выражений или диапазонов, заключенный в «[ ]».

Конструктор множества – это фактически *множественная константа*.

Порядок перечисления не важен:

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

Задание множественного типа:



Пример:

set of 1..3:

- []
- [1]
- [2]
- [3]
- [1, 2]
- [1, 3]
- [2, 3]
- [1, 2, 3]

Set Of Boolean:

- []
- [False]
- [True]
- [False, True]

Задание типа `set` аналогично представлению *множества с помощью массива*:

```
array[<Базовый_скалярный_тип>] of Boolean.
```

Обработка таких массивов неэффективна, поэтому используется тип Set.

Set Of Char:

– в старых версиях Delphi:
  - `Char = AnsiChar`
  - `set of Char = set of AnsiChar`

– в Unicode-версиях Delphi:
  - `Char = WideChar`
  - `set of Char = set of AnsiChar`

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

Множественное выражение – выражение, значением которого является множество. Частным случаем множественных выражений являются множественные переменные и конструкторы множества.

Операции над множествами:

Операция	Описание	Тип результата
=	равно	Boolean
<>	не равно	
<=	включение множества	
>=	противоположно <=	
in	вхождение элемента в множество	
+	объединение множеств	set
*	пересечение множеств	
-	разность множеств	
not	дополнение множества	
xor	исключающее объединение множеств	

Операция `in`:

- операция проверки вхождения элемента в множеств
  - левый операнд должен принадлежать базовому типу;
  - правый операнд – множественному типу, построенному на основе этого базового типа.

Встроенные функции и процедуры, определённые над множественным типом:

- Над множественными переменными определена одна встроенная функция – SizeOf(x):
  - указывает количество байт для представления значения ч множественной переменной;
  - возвращаемый параметр – Integer.

В языке Delphi нет прямого способа ввода и вывода множества в целом:

- множество представляет собой набор элементов;
- для ввода и вывода нужно работать с отдельными элементами множества.

Пример:

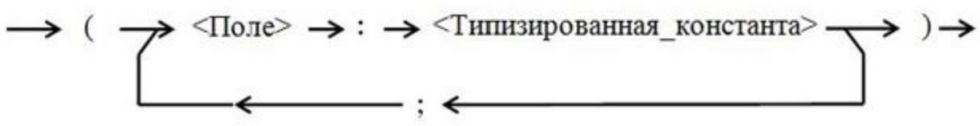
```
var
  A: set of 1..5;
  X, I: Integer;
begin
  for I := 1 to 5 do
  begin
    ReadLn(X);
    A := A + X;
  end;
```

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

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

Константа-запись:

<Константа_запись> ::=



- задаётся в разделе констант.

Назначение: может использоваться в качестве инициированной переменной типа запись (переменной, которой при запуске программы присваивается начальное значение).

Пример:

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

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

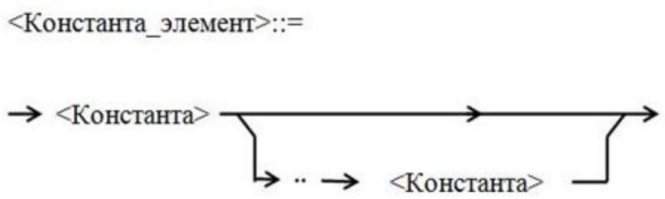
Особенности константы-записи:

- поля должны задаваться в том же порядке, что и в объявлении типа;
- если есть вариантная часть:
  - задаваться должны поля только одного из вариантов;
  - если есть поля признака, следует задавать поля выбранного варианта (в соответствии со значением поля признака);
  - использование компонент файлового типа в структурных константах-записях запрещено.

Константа-множество:

<Константа_множество>::=

- задаётся в разделе констант;
- <Константа_элемент> представляет собой значения или диапазоны значений базового типа множества.



Назначение: может использоваться как инициированная переменная типа множество.

Пример:

```
const
  Dig: set of 0..9 = [1, 3, 5];
  Dig1: set of 0..9 = [];
  Ch: set of ‘A’..‘Z’ = [‘A’..‘E’, ‘I’, ‘P’, ‘T’];
```

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

Физический файл (набор данных) – это поименованная область памяти на внешнем носителе, в которой хранится некоторая информация (файл с точки зрения пользователя).

Логический файл – представление физического файла в программе.

Способы доступа:

- последовательный:
  - по файлу можно двигаться только последовательно, начиная с первого его элемента;
  - доступен лишь очередной элемент файла;
  - чтобы добраться до n-го элемента файла, необходимо начать с первого элемента и пройти через предыдущие n – 1 элементов;
- прямой:
  - можно обратиться непосредственно к элементу файла с номером n, минуя предварительный просмотр n – 1 элементов файла.

Виды переменных файлового типа:

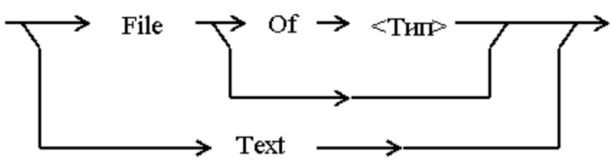
- «текстовый файлы»;
- «файлы с типом»;
  〉– бинарные файлы
- «файлы без типа».

! У физических файлов *нет типа*, они хранятся одинаково.

Файловый тип – произвольная последовательность элементов, длина которой заранее не определена, а конкретизируется в процессе выполнения программы.

Файловый тип:

<Файловый_тип>::=



- над значениями файлового типа не определены никакие операции;
- работа с файлами осуществляется с помощью так называемых процедур и функций ввода-вывода;
- в Delphi на данные момент представлен в виде packed record;
- устройство файлового типа не зависит от компилятора.

Текущая позиция в файле:

- с каждым открытым файлом связан т.н. **указатель файла**:
  - другие названия: окно файла, текущая позиция 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.

19. Процедура Assign/AssignFile. Назначение. Формат. Абсолютные и относительные пути к файлам. Логические имена устройств ввода-вывода. Пример.

Формат:

```
procedure Assign(var F; Name: String);
procedure AssignFile(var F; Name: String);
```

Параметры:

- F – имя файловой переменной;
- Name – выражение строкового типа, задающее полное имя физического файла.

Назначение:

- организует связь между конкретным физическим файлом на внешнем носителе (конкретным набором данных) и файловой переменной программы F;
- всегда предшествует другим процедурам ввода-вывода;
- нельзя применять к уже открытому файлу.

Параметр **Name** в общем случае оно имеет вид:

<Диск>:\<Имя_каталога>\…\<Имя_каталога>\<Имя_файла>

- <Диск> задаётся 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.

Затем может идти уточнение (**тип файла**), отделённое от имени точкой.

Пример **полного (абсолютного) имени** файла:

A:\Katalog1\Katalog2\Rez.pas

Относительным называется путь относительно текущего каталога.

Пример **относительных** имён файлов:

- Rez
- Rez.pas
- Rez.exe
- Rez.txt
- Rez.dat

Вместо имени физического файла в качестве параметра Name может использоваться любое устройство ввода-вывода. В этом случае Name – это символическое (логическое) имя устройства ввода-вывода.

Для использования доступны следующие символические имена устройств:

1) **CON** – устройство консоли (при вводе – клавиатура, при выводе – экран дисплея).

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. 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)).

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

Файл с типом:

- `file of <Тип>`;
- файл считается состоящим из элементов, каждый из которых имеет тип `<Тип>`.

Пример:

```
type
  Rec = record
    I: Integer;
    R: Real;
  end;
var
  F1: file of Real;
  F2: file of Char;
  F3: file of String[50];
  F4: file of Rec;
  F5: file of Integer;
```

Подпрограммы для открытия, чтения и записи, определённые над файлами с типом:

Подпрограмма	Параметры	Описание
`procedure AssignFile(var F; FileName: String);`	**F** – файловая переменная; **FileName** – путь к файлу	Связывает файловую переменную F с файлом FileName.
`procedure Reset(var F: file);`	**F** – файловая переменная, предварительно связанная с файлом процедурой Assign	Открывает существующий файл. Если F уже открыт, закрывает и снова открывает его. Указатель файла устанавливается на его начало. Режим определяется переменной FileMode.
`procedure Rewrite(var F: file);`	**F** – аналогично Reset	Создает и открывает новый файл F. Если F уже открыт, закрывает и снова открывает его. Указатель файла устанавливается на его начало.
`procedure Read(F, V1, ..., Vn)`	**F** – файловая переменная, предварительно открытая процедурой Reset или Rewrite;	Считывает компоненты файла в соответствующие переменные.

	V1, ..., Vn – переменные типа файла	
`procedure Write(F, V1, ..., Vn)`	**F** – аналогично Read; **V1, ..., Vn** – переменные типа файла	Записывает в файл компоненты из соответствующих переменных.

После открытия файла с типом доступны процедуры и чтения, и записи вне зависимости от способа открытия.

Пример:

```
var
  F: file of Integer;
  Value, I: Integer;
begin
  AssignFile(F, 'data.dat');
  Rewrite(F);
  for I := 1 to 5 do
  begin
    ReadLn(Value);
    Write(F, Value);
  end;
  CloseFile(F);
end;
```

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

Прямой доступ – способ доступа к элементам файла, при котором можно обратиться непосредственно к элементу файла с номером n, минуя предварительный просмотр n – 1 элементов файла.

Подпрограммы, определённые над файлами с типом:

Подпрограмма	Параметры	Описание
`function Eof(var F): Boolean;`	**F** – файловая переменная, предварительно открытая процедурой Reset или Rewrite	Проверяет, является ли текущей позицией конец файла.
`function FilePos(var F): Integer;`	**F** – файловая переменная, предварительно связанная с файлом процедурой Assign	Возвращает текущее положение в файле F в байтах. Если окно установлено на начало файла, то функция возвращает значение 0.

`function FileSize(var F): Integer;`	**F** – аналогично FilePos	Возвращает размер файла F в компонентах.
`procedure Seek(var F; N: LongInt)`	**F** – аналогично FilePos; **N** – номер компонента	Перемещает указатель файла F в позицию N.
`procedure CloseFile(var F);`	**F** – аналогично Eof	Закрывает файл.

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.
```

22. 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:

- `Text/TextFile`;
- считается последовательностью символов, интерпретируемый как текст:
  - **!** НЕ эквивалентен file of Char;
- в конце каждой строки помещается специальный управляющий символ – *маркер конца строки*.

Подпрограммы, обеспечивающие чтение из тестовых файлов:

Подпрограмма	Параметры	Описание
`procedure AssignFile(var F; FileName: String);`	**F** – файловая переменная; **FileName** – путь к файлу	Связывает файловую переменную F с файлом FileName.

`procedure Reset(var F: file);`	**F** – файловая переменная, предварительно связанная с файлом процедурой Assign	Создает и открывает новый файл F. Если F уже открыт, закрывает и снова открывает его. Указатель файла устанавливается на его начало.
`procedure Read(F, V1, ..., Vn)`	**F** – файловая переменная, предварительно открытая процедурой Reset; **V1, ..., Vn** – переменные	Считывает компоненты файла в соответствующие переменные.
`procedure ReadLn(F, V1, ..., Vn)`	**F –** аналогично Read; **V1, ..., Vn** – переменные	Выполняет процедуру Read, а затем переходит в начало следующей строки файла.

Особенности процедур для чтения данных из текстового файла в отличие файлов с типом:

- процедура Reset открывает файл в режиме только для чтения (read-only);
- процедура ReadLn определена только для текстовых файлов;
- первый параметр в процедурах Read/ReadLn может быть опущен, в этом случае подразумевается стандартный входной текстовый файл Input;
- при выполнении процедур Read/ReadLn осуществляется преобразование очередного элемента файла из текстового представления к типу переменной.

Допустимые типы данных:

- символьный;
- целочисленные;
- вещественные;
- строковый тип;
- тип массива символов;
- тип диапазона данных типов.

Пример:

```
var
  F: TextFile;
  Number: Integer;
begin
  AssignFile(F, 'myfile.txt');
  Reset(F);
  Read(F, Number);
  CloseFile(F);
end.
```

23. 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.

Подпрограммы, обеспечивающие запись в текстовые файлы и иное взаимодействие с ними:

Подпрограмма	Параметры	Описание
`procedure Rewrite(var F: file);`	**F** – файловая переменная, предварительно связанная с файлом процедурой Assign	Создает и открывает новый файл F. Если F уже открыт, закрывает и снова открывает его. Указатель файла устанавливается на его начало.
`procedure Append(var F: TextFile);`	**F** – аналогично Rewrite	Открывает существующий файл для добавления. Если F уже открыт, закрывает и снова открывает его. Указатель файла устанавливается на его начало.
`procedure Write(F, V1, ..., Vn)`	**F** – файловая переменная, предварительно открытая процедурой Rewrite/Append; **V1, ..., Vn** – переменные	Записывает в файл компоненты из соответствующих переменных.
`procedure WriteLn(F, V1, ..., Vn)`	**F** – аналогично Write; **V1, ..., Vn** – переменные	Выполняет процедуру Write и записывает маркер конца строки в файл F.
`function Eof(var F): Boolean;`	**F** – файловая переменная, предварительно открытая процедурой Reset/Rewrite/Append	Проверяет, является ли текущей позицией конец файла.
`function EoLn(var F): Boolean;`	**F** – аналогично Eof	Проверяет, является ли текущей позицией маркер конца строки.
`procedure SeekEof(var F: TextFile);`	**F** – аналогично Eof	Устанавливает файл F в состояние «конец файла».
`procedure SeekEoLn(var F: TextFile);`	**F** – аналогично Eof	Устанавливает файл F в состояние «конец строки».

`procedure CloseFile(var F);`	**F** – аналогично Eof	Закрывает файл. При закрытии файла очищается буфер ввода-вывода данного файла.

Особенности процедур для работы с данными из текстового файла в отличие файлов с типом:

- процедуры Append, WriteLn, EoLn, SeekEof, SeekEoLn определены только для текстовых файлов;
- процедуры Rewrite и Append открывают файл в режиме только для записи (write-only);
- в процедурах Write/WriteLn первый параметр может быть опущен, в этом случае подразумевается стандартный входной текстовый файл Input;
- процедуры Write/WriteLn записывают в файл текстовое представление значения переменной.
- при записи переменных в текстовый файл с помощью процедуры Write они размещаются последовательно без определенных разделителей или фиксированных позиций:
    - каждое значение будет следовать непосредственно за предыдущим без разделителей.

Допустимые типы данных:

- символьный;
- арифметические (целочисленные и вещественные);
- строковый;
- массив символов;
- логический;
- их диапазоны.

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

Буфер ввода-вывода:

- участок оперативной памяти;
- через него осуществляется обмен информацией между программой и внешним набором данных;
- стандартный размер – 128 байт (может быть переопределён);
- каждому открытому файлу назначается свой буфер;
- операции обмена данными через буфер ввода-вывода осуществляет специальный обработчик файлов (для каждого файла имеется свой обработчик файлов, он назначается при открытии файла);
- позволяет существенно повысить скорость передачи данных.

Буферизация ввода-вывода осуществляется с помощью процедуры **SetTextBuf()** и функции **Flush()**.

SetTextBuf:

- `procedure SetTextBuf(F, Buf [, Size]);`
- параметры:
  - F – имя файловой переменной для текстового файла;
  - Buf – любая переменная (в качестве формального параметра используется параметр-переменная без типа);
  - Size – необязательная переменная типа Word;
- назначение: определяет буфер для текстового файла;
- ограничения: нельзя применять к открытому файлу (процедура должна применяться после процедуры Assign/AssignFile и перед процедурами Reset/Rewrite/Append).

Flush():

- `function Flush(F): Integer;`
- параметры:
  - F – имя файловой переменной для текстового файла;
- назначение: очищает буфер текстового файла, открытого для вывода процедурой Rewrite или Append;
- возвращаемое значение:
  - 0, если очистка была завершена успешно;
  - код ошибки в противном случае.

Пример:

```
var
  F: Text;
  C: Char;
  Buf: array[1..10240] of Char;
begin
  AssignFile(F, 'A:\MET\Metod.txt');
  SetTextBuf(F, Buf);
  Reset(F);
  while not Eof(F) do
  begin
    Read(F, C);
    WriteLn(C);
  end;
  Flush(F);
  CloseFile(F);
end.
```

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

Внутренняя структура представления данных в текстовых файлах и файлах с типом:

1. **Представление числовой информации:**
   - в текстовом файле:
     - пусть записана последовательность чисел 8192, 2048, …;

- под каждую десятичную цифру отводится один байт, содержащий код символа в кодировке ASCII;
- между числами есть разделители в виде запятой и пробела, каждый из символов также занимает один байт.

– в файле с типом (file of Integer):
- пусть записана последовательность чисел 8192, 2048, …;
- числа представлены в двоичном коде в формате Integer, под каждое число отводится четыре байта;
- разделители между числами отсутствуют.

Если нет необходимости выводить числовую информацию на экран или печать, эффективнее использовать *файлы с типом*:
- отсутствует преобразование информации, она в файле представлена так же, как в памяти;
- при работе с файлами с типом возможен режим прямого доступа;
- меньше операций физического ввода-вывода за счет меньшего размера файла;
- они занимают меньше места.

2. Представление текстовой информации:

– в текстовом файле:
- пусть записан текст «ВАШ ОТВЕТ НЕВЕРЕН»;
- 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.

– в файле с типом (file of String[7]):
- пусть записан текст «ВАШ ОТВЕТ НЕВЕРЕН»;
- строки имеют постоянную (максимальную) длину, пустой символ (символ #0 в кодовой таблице) дополняет строки текущей длины до максимальной длины.

Сравнение файлов типа TextFile и file of String[7] показывает, что:
- меньше места в общем случае занимает текстовый файл (так как в нем используются строки текущей длины);
- большую скорость работы обеспечивает file of String[7], так как для типизованных файлов имеется возможность работы в режиме прямого доступа, работы одновременно в режиме записи-чтения, не нужно отслеживать управляющие символы #13#10 (маркер конца строки).

Таким образом, для хранения текстовой информации выбирать файл типа TextFile или file of String следует в каждом конкретном случае, исходя из особенностей задачи.

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

Файлы без типа:

- `file`;
- файл считается состоящим из элементов, размер которых определяется при открытии файла.

Назначение файлов без типа – максимально повысить скорость обмена информацией с внешними наборами данных.

```
procedure Reset(var F: file; RecSize: Word);
```

Скорость обмена повышается за счет следующих **факторов**:

- в файлах без типа отсутствует преобразование типа компонентов;
- не выполняется поиск управляющих символов (типа конец строки);
- 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 они были открыты;
- обмен информацией с внешними наборами данных может быть осуществлен большими блоками.

Последний фактор является основным с точки зрения повышения скорости обмена.

Подпрограммы для работы с файлами без типа:

Подпрограмма	Параметры	Описание
`procedure AssignFile(var F; FileName: String);`	**F** – файловая переменная; **FileName** – путь к файлу	Связывает файловую переменную F с файлом FileName.
`procedure Reset(var F: file);` `procedure Reset(var F: file; RecSize: Word);`	**F** – файловая переменная, предварительно связанная с файлом процедурой Assign; **RecSize** – необязательное выражение типа Word, определяющее размер записи (в байтах), используемый при передаче данных	Открывает существующий файл. Если F уже открыт, закрывает и снова открывает его. Указатель файла устанавливается на его начало. Режим определяется переменной FileMode.
`procedure Rewrite(var F: file);`	**F** – аналогично Reset; **RecSize** – аналогично Reset	Создает и открывает новый файл F. Если F уже открыт, закрывает и снова открывает его.

procedure Rewrite(**var** F: **file**; RecSize: Word);		Указатель файла устанавливается на его начало.
procedure BlockRead(**var** F: file, **var** Buf, Count: Integer[; **var** Done: Integer])	**F** – файловая переменная, предварительно открытая процедурой Reset или Rewrite; **Buf** – переменная любого типа; **Count** – выражение типа Word, определяющее количество считываемых компонентов; **Done** – количество полных считанных компонентов	Считывает не более Count компонентов в переменную Buf из файла F. Если Done меньше Count, то это значит, что конец файла достигнут до полного окончания передачи. В этом случае, если параметр Done отсутствует, возникает сообщение об ошибке ввода-вывода.
procedure BlockWrite(**var** F: file, **const** Buf, Count: Integer[; Done: Integer])	**F, Buf** – аналогично Read; **Count** – выражение типа Word, определяющее количество записываемых компонентов; **Done** – количество полных записанных компонентов	Записывает не более Count компонентов из переменной Buf в файл F. Если Done меньше Count, то это значит, что диск переполнился до завершения пересылки данных. В этом случае, если параметр Done отсутствует, возникает сообщение об ошибке ввода-вывода.
function Eof(**var** F): Boolean;	**F** – файловая переменная, предварительно открытая процедурой Reset или Rewrite	Проверяет, является ли текущей позицией конец файла.
function FilePos(**var** F): Integer;	**F** – файловая переменная, предварительно связанная с файлом процедурой Assign	Возвращает текущее положение в файле F в байтах. Если окно установлено на начало файла, то функция возвращает значение 0.
function FileSize(**var** F): Integer;	**F** – аналогично FilePos	Возвращает размер файла F в компонентах.
procedure CloseFile(**var** F);	**F** – аналогично Eof	Закрывает файл.

		При закрытии файла очищается буфер ввода-вывода данного файла.

Пример:

```
program CopyFile;
var
  F1, F2: File;
  Buf: array[1..2048] of Char;
  NumR, NumW: Word;
begin
  AssignFile(F1, 'DATA1');
  AssignFile(F2, 'DATA2');
  Reset(F1, 1);
  Rewrite(F2, 1);
  repeat
    Blockread(F1, Buf, SizeOf(Buf), NumR);
    Blockwrite(F2, Buf, NumR, NumW);
  until (NumR = 0) or (NumR <> NumW);
  CloseFile(F1);
  CloseFile(F2);
end.
```

27. Проверка операций ввода-вывода. Пример.

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

Если нежелательно, чтобы выполнение программы прерывалось по ошибке ввода-вывода, то есть если мы сами хотим обрабатывать данные ошибки, необходимо 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.

Пример:

```
var
  F: file of Char;
begin
```

```
  AssignFile(F, 'DATA');
  {$I-}
  Reset(F);
  {$I+}
  if IOResult = 0 then
    writeln('Размер файла: ', FileSize(F), 'байт.')
  else
    writeln('Файл не обнаружен.');
end.
```

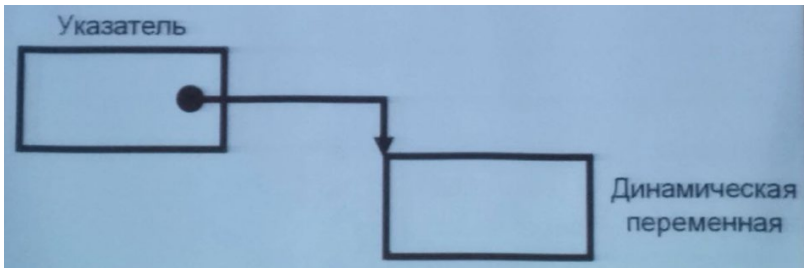
28. Ссылочный тип. Назначение. Синтаксис задания. Представление в памяти. Виды указателей. Операции над указателями. Пример.

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

Все идентификаторы имеют смысл *только в исходном коде*.

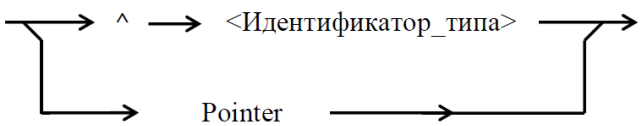
! В скомпилированной программе есть только обращения по адресам, но не имена переменных **!**

Динамические переменные не объявляются в программе: единственный способ обратиться к ним – по адресу.



Ссылочный тип (тип указатель): значения – адреса в памяти.

<Тип_указатель>::=



Пример:

```
var
  P: ^TSomeType; // P содержит ссылку на данные типа TSomeType
```

Указатели:

– типизированные;

– нетипизированные.

Внутреннее представление переменной указателя одинаково вне зависимости от того, типизированный указатель или нет.

Ссылочные типы можно объявлять до объявления базовых типов.

Пример:

```
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 над переменными типа указатель:

- для получения адреса используется унарная операция **@** (вместо неё можно использовать функцию **Addr()**);
- для доступа к динамической переменной через указатель используется унарная операция разыменования указателя ^.

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

29. Процедуры New и Dispose, GetMem и FreeMem. Назначение. Сравнительная характеристика (достоинства и недостатки). 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.

GetMem/FreeMem:

- GetMem:
  - `procedure GetMem(var P: Pointer; Size: Integer);`
  - выделяет блок размером Size байт и помещает его адрес в переменную P;
- FreeMem:
  - `procedure FreeMem(var P: Pointer[; Size: Integer]);`
  - освобождает блок с адресом P:
    - ! необязательный параметр Size должен иметь то же значение, что и при вызове GetMem().

New/Dispose:

- P должен быть *типизированным*;

– New:
  - **`procedure New(var P: Pointer);`**
  - выделяет блок памяти и помещает его адрес в переменную P:
    - размер блока равен размеру P;
– Dispose:
  - **`procedure Dispose(var P: Pointer);`**
  - освобождает блок с адресом P.

В общем случае процедуры GetMem/FreeMem дают большую свободу действий по сравнению с процедурами New/Dispose, так как позволяют работать без привязки к конкретному типу переменных.

про достоинства/недостатки лень пиздец

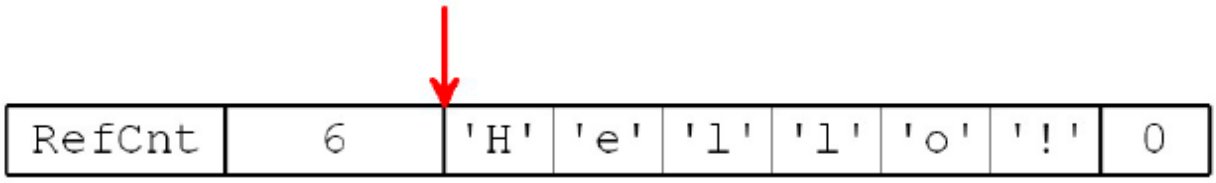
Пример:

```
var
  P: ^Integer;
begin
  New(P);
  ...
  Dispose(P);
end;
```

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

Delphi-строки:

- в Delphi по умолчанию используется *особый вид строк*:



  - *Счётчик ссылок* (RefCnt):
    – подсчёт ссылок широко используется для решения задачи автоматического управления памятью;
    – значение счётчика ссылок – количество переменных, *ссылающихся* на объект (строку, массив):
      - когда счётчик ссылок становится равным 0, память, занятая объектом (строкой, массивом), освобождается;
  - *Фактическая длина строки*;
- тип `String` может соответствовать *различным видам* строк:
  - `String[N]/ShortString` – всегда Pascal-строка;
  - `String` – Pascal- или Delphi-строка в зависимости от *настроек компилятора;*
- переменная типа Delphi-строка на самом деле является *указателем*:
  – присваивание **`nil`** позволяет удалить ссылку на строку;

  - значение **`nil`** эквивалентно пустой строке;
- у *строковых констант* счётчик ссылок равен –1 и никогда не изменяется;
- при попытке изменения строки:
  - если счётчик ссылок равен 1, изменение происходит «по месту»;
  - иначе создаётся копи строки и в дальнейшем работа ведётся с этой копией:
    - при этом счётчик ссылок уменьшается на 1 (кроме случая, когда он равен –1);
- Delphi-строки – *управляемый тип данных*:
  - инициализируются компилятором в значение **`nil`** (в том числе локальные переменные);
  - управление занятой памятью и размерами осуществляется *компилятором*.

честно спизжено из моих же билетов 1-го сема, не судите женщину за ~~член~~ Delphi-строки

31. Директива absolute. Принцип работы. Пример.

Директива `absolute`:

- позволяет создать переменную, которая будет располагаться в той же области памяти, что и какая-либо другая переменная;
- синтаксис задания:

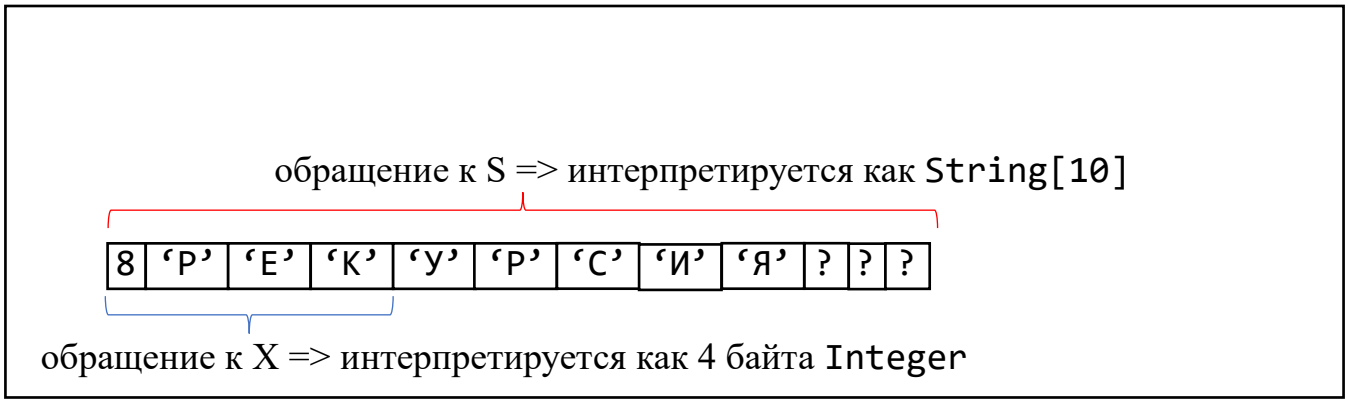
```
var
  S: String[10];
  X: Integer absolute S;
```

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.

Представление в памяти:

RAM



суки бляди заебалась равнять

Пример:

```
procedure CopyData(const Source; var Dest; Count: Integer);
var
  S: array[0..MaxInt - 1] of Byte absolute Source;
  D: array[0..MaxInt - 1] of Byte absolute Dest;
  I: Integer;
begin
  for I := 0 to Count - 1 do
    D[I] := S[I];
end;
```

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

Packed Record:

- 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;
- наиболее эффективно чтение адресов, кратных размерам читаемых данных.

Выравнивание – способ размещения переменных в памяти, позволяющий минимизировать количество обращений к памяти.

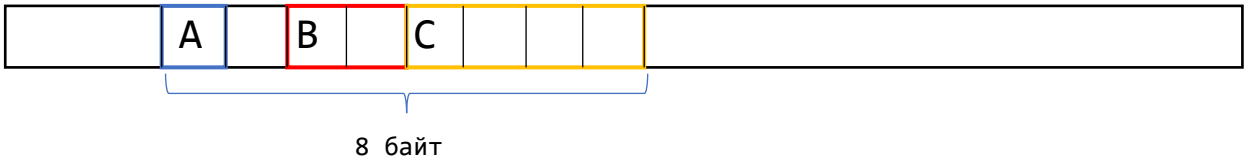
В общем случае переменные следует выравнивать по адресам, кратным их размеру.

По умолчанию компилятор «выравнивает» поля всех записей:

- при этом некоторые байты могут оставаться неиспользуемыми (только для выравнивания);
- директива **{$A}** позволяет управлять выравниванием.

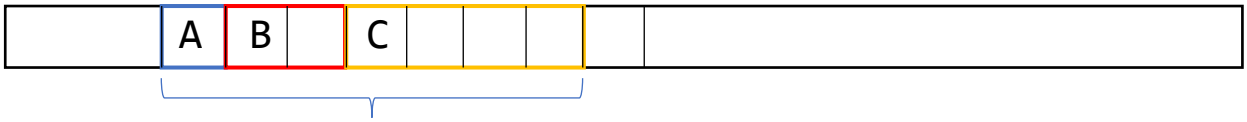
Зарезервированное слово `packed` заставляет компилятор располагать поля записи вплотную друг к другу:

```
type
  TMyRecord = record
    A: Byte;
    B: Word;
    C: LongWord;
  end;

SizeOf(TMyRecord) = 8:
```



8 байт

```
type
  TMyRecord = packed record
    A: Byte;
    B: Word;
    C: LongWord;
  end;

SizeOf(TMyRecord) = 7:
```



7 байт

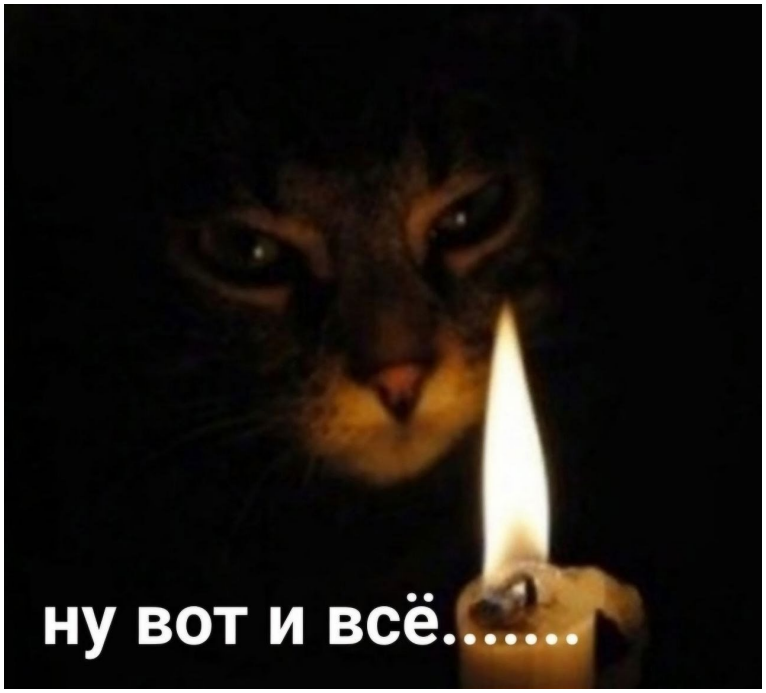
Преимущество `packed`: предсказуемость внутреннего устройства.
Преимущество не-`packed`: быстрота.
Применение `packed`:

- если используются (типизированные) файлы и необходимо, чтобы всё, что в есть в файле, отображалось в программе аналогично, без выравнивания;
- для совместимости разных платформ;
- полезна при работе со сложными форматами.

Если запись неупакованная, то в качестве локальной переменной может содержать неинициализированные байты. Такая запись может быть затем записана в файл «как есть», т.е. с этими «мусорными» байтами.

Зарезервированное слово `packed` также может применяться для массивов (**`packed array`**).

Редакторы: Колбеко Влада, Лутай Варвара.


ну вот и всё.....