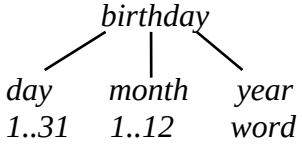
19. Записи.

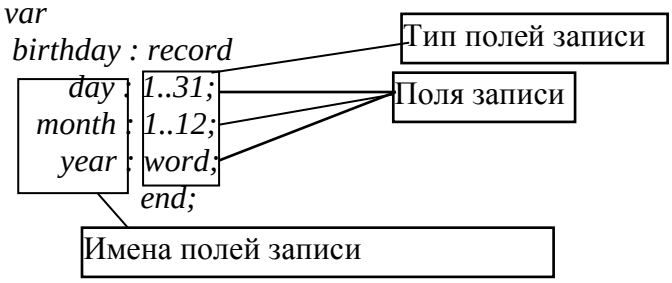
19.1 Понятие записи. Объявление записи в программе.

Тип записи используется для хранения под *одним* именем *разнотипных* (в отличие от массива) элементов. Эти элементы называются обычно ***полями* записи.**

Допустим необходимо представить в виде записи информацию о дне рождения: день, месяц и год. Соответствующая иерархическая структура будет иметь вид:



Описание такой записи на Паскале:




```
Var
  r:record
       описание 1-го поля;
       описание 2-го поля;
       описание 3-го поля;
        ......
       описание N-го поля;
     end;
```

Замечание:

Тип записи, как и тип массива, можно описывать прямо в секции описания переменных. *Но* такую запись нельзя будет передать в подпрограмму (по адресу - как параметр-переменную).

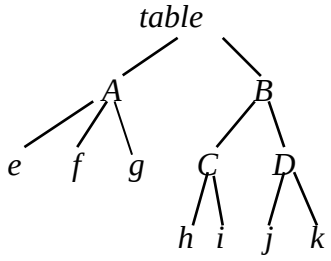
Начинается описание записи со слова *record*, заканчивается словом *end*. Порядок описания полей соответствует порядку их размещения в памяти, т.е. в памяти эти поля будут расположены таким образом, что *адреса будут расти при движении по записи сверху вниз*. Записи часто бывают вложенными, когда одна запись вкладывается в другую и представляет собой достаточно сложную иерархическую структуру. Вложенные записи часто используются для **представления таблиц** (структура данных).

1

Пусть таблица *table* имеет следующую структуру:

A			B			
			C		D	
e	f	g	h	i	j	k

Такой структуре записи соответствует следующая иерархическая структура данных:



Соответствующее описание записи:

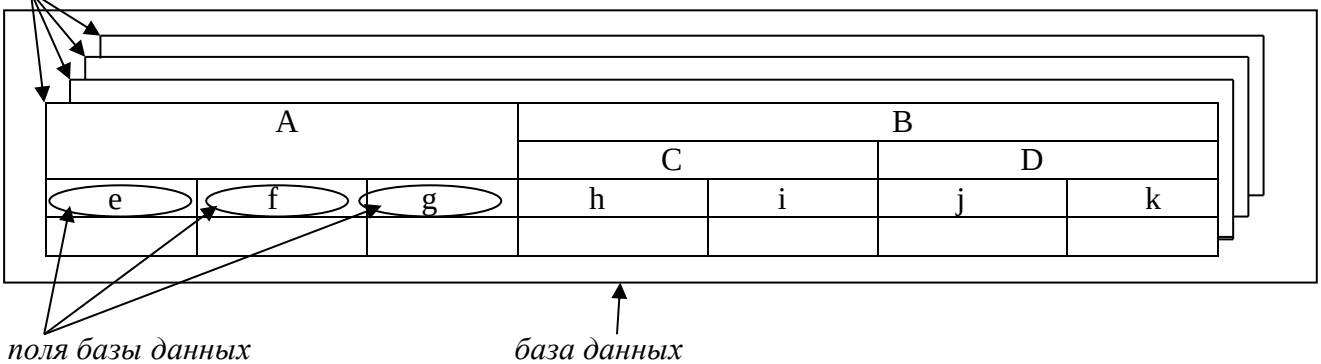
```
var
 table : record
          A : record
                e : byte;
                f : word;
                g : real;
              end {A};
          B : record
                C : record
                      h : byte;
                      i : char;
                    end {C};
                D : record
                      j : real;
                      k : byte;
                    end{D};
              end {B};
        end {table};
```

То, что написано выше позволяет сохранить только одну строку в сложной таблице. Если необходимо представлять таблицу размером, больше чем одна строка, то возможно два варианта:

1). Можно рассматривать *сложную* таблицу как бы состоящую из большого количества однострочных карточек: Такой структуре данных будет соответствовать (как структура хранения) *массив записей,* где каждая запись позволяет описать одну строку таблицы:

```
var
 table : array [1..10] of record
            A: record
                 e : byte;
                 f : word;
                 g : real;
               end {A};
            B : record
                 C : record
                       h : byte;
                       i : char;
                     end {C};
                 D : record
                       j : real;
                       k : byte;
                     end {D};
               end {B};
end {table};
```

10 записей в массиве записей (в базе данных из записей)



поля базы данных *база данных*

2). Массивом будет являться каждое из полей записи, расположенных внизу иерархии:

```
var
    table :  record
                A: record
                     e : array[1..10] of byte;
                     f : array[1..10] of word;
                     g : array[1..10] of eal;
                   end;
                B : record
                     C : record
                           h : array[1..10] of byte;
                           i : array[1..10] of char;
                         end;
                     D : record
                           j : array[1..10] of real;
                           k : array[1..10] of byte;
                         end;
                   end;
end;
```

A			B			
			C		D	
e	f	g	h	i	j	k

10 строк

в каждой строке колонки (поля) логически связаны друг с другом (образуют кортеж), т.к. обычно относятся к одной сущности (как ведомости на зарплату).

Первый способ представить таблицу отличается от второго, например, удобством сортировки записей в целом: если необходимо сортировать строки в таблице, то по 2-му способу надо будет переупорядочивать для каждой колонки все элементы массивов, а по 1-му способу – будут «сдвигаться» записи (строки) целиком, что удобнее.

19.2 Доступ к полям записи.

Доступ к полям записи производится с использованием т.н. *составных имен*, в записи которых всегда присутствует точка. Применительно к записи составное имя образуется из имени переменной типа запись, справа от которого через точку записывается в простейшем случае (в случае простой записи - без вложенных записей) имя интересующего нас поля (перед которым - слева) ставятся имена вышестоящих по иерархии полей типа запись) .

```
Var
 A: record
        b: byte;
        C : record
            d: char;
            e : byte;
         end;
end;
```

3

В приведенной записи доступ к полям выглядит следующим образом:

```
A. b :=          ;
A. C. d :=          ;
A. C. e :=          ;
```

Имя поля

Путь к полю записи

В записи три поля. Доступ к полю *b* записи достаточно прост. Доступ к полям *e* и *d* более сложен, потому что слева записывается путь, который нужно пройти (до нужного поля), двигаясь по именам записей от начала самого описания самой внешней записи.

В случае массива записей доступ для поля *h* будет выглядеть:

table[5]. B. C. h:= .... ;

Для случая, где массивы являются полями записи доступ к полю *h* выглядит следующим образом:

table. B. C. h[5]:= .... ;

19.3 Оператор *with*

Формат его записи:

with префикс *do*

оператор;

один оператор!

Если описание пути к полю записи длинное (путь к полям записи долго писать) и необходимо выполнить несколько единообразных действий над несколькими полями этой записи, то оператор *with* избавляет от необходимости каждый раз для каждого действия записывать один и тот же полный путь к интересующей нас записи.

Для последнего случая, если надо обработать несколько элементов (полей массива), то обработку традиционны образом можно было бы описать следующим образом:

```
table.B.C.h[1]:=        ;
table.B.C.h[2]:=        ;
table.B.C.h[3]:=        ;
...
table.B.C.h[9]:=        ;
```

С использование оператора WITH та же последовательность действий будет такой:

```
with table.B.C do
  begin
    h[1]:= ...;
    h[2]:= ...;
   ...
  end;
```

Путь к полям записи

В качестве префикса (после *with*) необходимо указывать ту часть пути, которую не надо (не хочется) многократно переписывать. В этом случае после *do* в описании действий над полями можно префикс перед именами полей автоматически опускать.

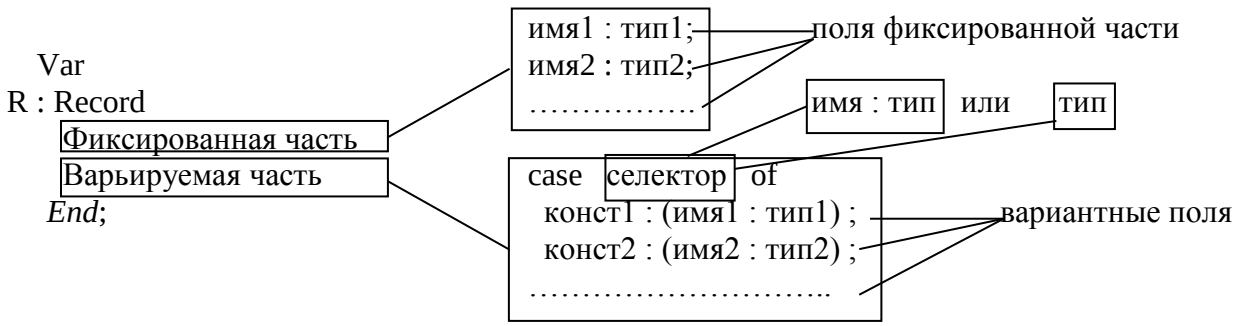
Примечание. После *do* можно писать только один оператор. Если надо написать несколько, то следует использовать операторные скобки (составной оператор).

19.4 Действия над записями

1) Разрешается присваивать (как у массивов) одной записи целиком содержимое другой (но обязательно однотипной) записи (надо отдельно описать тип). 3) Ввод-вывод возможен только по отношению к полям записи. 2) Запись можно передавать как параметр в подпрограммы. Надо только предварительно описать тип такой записи в секции описания типов (для обеспечения эквивалентности типа формального и фактического параметра-записи).

19.5 Записи с вариантами (union в Си)

Основная идея состоит в том, что в рамках одной и той же области памяти, выделенной для записи, привести описания различных вариантов заполнения этой области значениями других типов. В общем случае вариантная запись состоит из двух частей: фиксированной и вариантной (переменной).



Замечание1:Порядок частей – именно такой, как показано: фиксированная часть всегда первая (или единственная)
Замечание2: Фиксированная часть может вообще отсутствовать (в вариантной записи).
Замечание3: Вариантная часть может быть только одна.

Фиксированная часть состоит из отдельных полей и структура их точно такая же, как у ранее рассмотренных записей. Под каждую переменную, описываемую в фиксированной памяти, выделяется отдельная ячейка памяти.

Вариантная часть начинается со слова *case*.

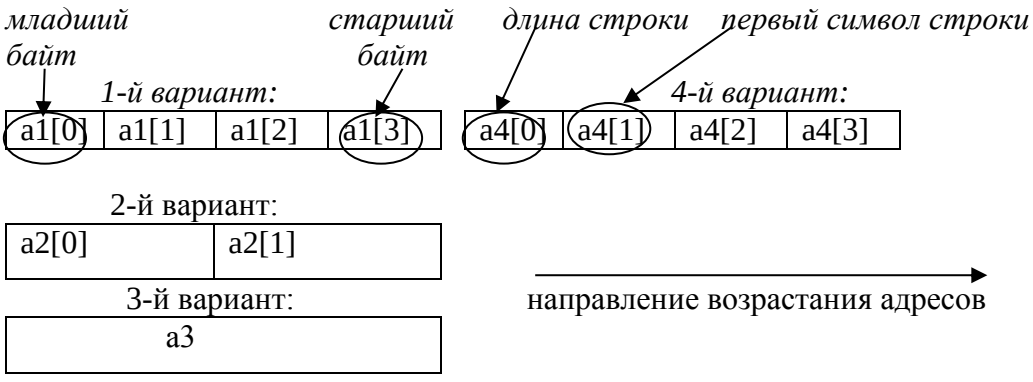
Особенности вариантной части:

- Поле селектора может иметь следующие два вида:
  1). имя : тип
  2). тип
- Каждое вариантное поле предваряется константой, которая отделяется от описания поля двоеточием.
- Каждое описание вариантного поля (после константы и двоеточия) заключается в круглые скобки.
- В вариантной части каждое поле описывается так же как и в фиксированном (имя: тип).
- По поводу типа селектора и типа константы. Тип селектора выбирается с таким расчетом, чтобы число возможных изменений значений этого типа было не меньше, чем число этих вариантных полей.
- В ТР не принято использовать имя (и значение) селектора вообще. Это в старом Паскале для того, чтобы выбрать нужный вариант, надо было присвоить соответствующее значение переменной-селектору.
- Тип селектора может быть любым порядковым кроме longint.
- Если известно, что число возможных значений не больше 2, то обычно используют *boolean* для типа селектора.

- По поводу того как вариантная запись располагается в памяти. Особенность этого состоит в том, что все вариантные поля одной и той же записи разделяют одну и ту же область памяти. Размер этой области выбирается компилятором по размеру наибольшего вариантного поля. (по типу : 1 байт, 2 б, 4б и т.д.).
- В каждый конкретный момент времени доступно только одно поле из вариантной части.
- При мысленном переходе от одного вариантного поля к другому, значение вариантной части не меняется, меняется лишь та точка зрения на ту область памяти, которая отведена под вариантную часть.

Пример:

```
Var
  A : record
        Case byte of
          0 : (a1 : array [0..3] of byte);
          1 : (a2 : array [0..1] of word);
          2 : (a3 : longint);
          3 : (a4 :string[3]);
      end;
```



Здесь изображены четыре точки зрения на одну и ту же область памяти из 4 байтов:
1) Массив из 4-х элементов размером 1 байт.
2) Массив из 2-х элементов размером 2-а байта.
3) Одно значение размером в 4-е байта.
4) Строка из трех символов

Наиболее частый случай использования таких вариантных записей состоит или в неявном преобразовании типов (при переходе от поля к полю) или в прямом (без масок) доступе к разным частям длинного знвчения.

Примеры:
a.a1[0] : = 1; - в младший байт 4-байтной области памяти записывается значение 1.
a.a1[3] : = 1; - в старший байт 4-байтной области памяти записывается значение 1.

Если бы вместо *longint* записать *string[3]*, то в этом случае можно было бы с каждым символом строки работать как с целым числом без всяких преобразований типов.

В Си есть записи (структуры) с битовыми полями, которые позволяют получить доступ к отдельным битам:

```
union
   {
     unsigned char i;  ≡ byte на Паскале
     struct
       {
          i1:1;  //мл. разряд
          i2:6;
          i3:1;  //ст. разряд
       } j; //имя структуры (записи)
   } u;  //имя union'a
Можно писать:
    u.i = 25;   //обратились к байту целиком
    u,j.i1 = 1; //обратились только к мл. биту байта
```

19.6 Типизированные константы - записи

Рассмотрим четыре примера задания записей как типизированных констант. Использование типизированных констант-записей удобно, например, при отладке, чтобы многократно (при каждом запуске программы) не вводить значения полей.

1).

```
const
 v : record;
        a : byte;
        b : char;
    end = (a : 1; b : '2');
```

так же как и для массива все заключается в круглые скобки; только в отличие от массива одно значение от другого выделяется точкой с запятой (а не запятой как для массива).

2).

```
const
 v : record;
        a : byte;
        b : char;
        c : record;
              d : integer;
              e : byte;                !!!
             end;
       end = (a : 1; b : '2'; c:(d : 3; e : 4));
```

3).

```
const
 v : record;
        a : byte;
        b : char;
        c : record;
              d : integer;
              e : byte;
             end;
       case Boolean of
        false : (f : integer);          можно инициализировать лишь одно вариантное поле
        true : (g : byte);
       end = (a : 1; b : '2'; c:(d : 3; e : 4); f : 5);
```

4).

```
const
 v : record;
        a : byte;
        b : char;
        c : record;
              d : integer;
              e : byte;
             end;
       case n : Boolean of
        false : (f : integer);
        true : (g : byte);
       end = (a : 1; b : '2'; c:(d : 3; e : 4); n : false; f : 5);
```

по логике мы должны обеспечить соответствие между значением констант вариантного поля и именем вариантного поля, которому мы хотим присвоить начальное значение