Занятие 19. Записи

Как уже отмечалось в занятии 6, структурированные типы данных могут содержать несколько значений в одной переменной. Кроме массивов и множеств, к структурированным типам данных языка Паскаль относятся ещё и записи[1].

В отличие от массивов и множеств, записи позволяют в одной переменной хранить значения разных типов. Каждому из хранимых значений соответствует своя часть записи, называемая полем.

Определение типа данных «запись»

```
Type ИмяТипаЗаписи = Record
          Список_полей1 : тип1;
          Список_полей2 : тип2;
          …
          Список_полейN : типN;
     End
```

Перед End, которым заканчивается определение типа данных, точку с запятой ставить не обязательно. Каждый список полей разделяется запятыми. Типы полей `тип1` … `типN` не обязаны быть простыми. Среди них могут оказаться и массивы, и строки, и множества, и другие типы данных.

Примеры определения типов-записей

```
Type TDate = record
    Year : Word;
    Day, mon : Byte
End;

TPerson = record
    Name : string;
    bd : TDate
End;
```

Запись TDate состоит из трёх полей: year (год) типа Word и двух полей day (день) и mon (месяц) типа Byte. Запись TPerson имеет поля Name типа string и bd (дата рождения) типа TDate. Заметим, что тип TDate определён раньше типа TPerson, который его использует.

Примеры описания переменных типа запись

```
Var d,dt : TDate;
       p : array[1..100] of TPerson;
```

Здесь описаны переменные `d` и `dt` типа TDate и массив `p` из 100 записей типа TPerson. Заметим также, что память для хранения данных выделяется не при определении типа, а при описании переменной этого типа.

Операции над записями

Для записей одного и того же типа определена операция присваивания, которая сводится к копированию значений всех полей одной записи в соответствующие поля другой. Например, пусть в переменной `d` из примера выше хранится дата 11 января 2011 года. Тогда после выполнения оператора

[1] Этим, однако, не исчерпывается весь список структурированных типов данных языка Паскаль

```
dt:=d
```

значение переменной `dt` тоже будет соответствовать 11 января 2011 года, т.е. у обеих записей значение поля `year` будет равно 2011, поля `day` — 11, а поля `mon` — 1.

Для доступа к какому-либо полю записи надо написать

```
Имя_переменной_типа_запись.имя_поля
```

Например,

```
d.year:=2011
```

Это значит, что полю `year` записи `d` будет присвоено значение 2011. Обратите внимание, что перед точкой пишется не имя типа, а имя *переменной* этого типа, т.е. имя конкретного экземпляра записи.

Или ещё примеры

```
p[1].name:= 'Иванов';
p[1].bd.year:=1990;
```

Здесь полю `name` первого элемента массива записей `p` присваивается значение `'Иванов'` и указывается год его рождения 1990. Но поле `bd` само является записью, поэтому последний пример надо понимать как полю `year` поля `bd` записи `p[1]` присвоить значение 1990.

Доступ к полю записи можно получить также при помощи *оператора присоединения with.*

```
With имя_переменной_типа_запись Do
Имя_поля
```

С помощью оператора присоединения последний пример можно переписать так

```
With p[1] Do
Begin
   name:= 'Иванов';
   bd.year:=1990
End
```

Здесь поля `name` и `bd.year` относятся к записи `p[1]`.

Для иллюстрации рассмотрим короткую программу

```
Program Test;
Type TDate =  record
                 year : Word;
                 mon, day : Byte
              end;

Var d, dt : TDate;
Begin
  d.year:=2011;
  d.day:=11;
  d.mon:=1;
  dt:=d;
  with dt do
    WriteLn(day,'.',mon,'.',year)
End.
```

Отслеживание значений переменных

Наберите текст этой программы в интегрированной среде разработки (IDE) FreePascal. В эту среду включена возможность работы с отладчиком, при помощи которого можно проследить за изменением значений переменных по ходу исполнения программы. Сейчас мы собираемся сделать именно это.

Мы хотим проследить за изменением значений переменных d и dt. Сообщим об этом отладчику. Для этого выберем в меню «Debug» пункт «Add Watch» или просто нажмём Ctrl-F7. В появившемся окне введём d и нажмём кнопку «OK». Повторим те же действия для переменной dt. Если в нижней части экрана мы не видим окна «Watches», то нажмём кнопку F5 или попробуем мышкой уменьшить размер окна с текстом программы, перетаскивая его правый нижний угол, выделенный зелёным цветом. (В Delphi кнопка Ctrl-F7 тоже работает, только окно «Watches» расположено слева.) Теперь начнём пошаговое исполнение программы. Для этого в меню «Run» есть две возможности: «Trace into» (или F7) и «Step over» (или F8). Различия в них проявляются при вызове подпрограмм. F8 заставляет всю подпрограмму выполниться до конца, не отслеживая ход её выполнения, как будто вся подпрограмма является одной неделимой командой. F7 подробно прослеживает ход выполнения подпрограммы.

Поскольку в нашей программе подпрограмм нет, нам сейчас всё равно, что нажимать: F7 или F8. Нажмём, например, F8. В окне «Watches» отобразятся текущие значения всех полей записей d и dt. Нажимая F8 ещё и ещё, мы сможем отслеживать значения переменных в окне «Watch». Подсвеченная строка показывает нам ту команду, которая будет выполнена на следующем шаге, например при следующем нажатии F7 или F8. Когда будет выполнена команда dt:=d, мы увидим, что значения всех полей записи d скопировались в запись dt. Продолжаем нажимать F8.

После окончания выполнения программы появится окно «Information» с кодом завершения программы (exitcode). Exitcode=0 свидетельствует о её нормальном (не аварийном) завершении[2]. Добавлять переменные в окно отладчика, как и удалять из него, можно в любой момент по ходу отладки программы. Для удаления отображения значения переменной в окне отладчика просто щелкаем мышкой по соответствующей строке и наживаем клавишу Delete. Прервать отладку тоже можно в любой момент, нажав Ctrl-F2. Если надо скрыть окно отладчика, ставим курсор в окно с текстом программы и нажимаем F5. Если надо с текущего места выполнить программу до конца, нажимаем Ctrl-F9 в FreePascal или F9 в Delphi.

Можно сделать так, чтобы при запуске программы в автоматическом режиме её выполнение останавливалось на некоторой строке. Для этого надо определить точку останова breakpoint. Делается это нажатием Ctrl-F8, когда курсор находится на нужной строке. После того, как выполнение программы было остановлено на требуемой строке, можно использовать F7, F8 или продолжить выполнение в автоматическом режиме (Ctrl-F9 в FreePascal). Точек останова в программе может быть несколько. Удаляют breakpoint так же, как и ставят.

Мы уделили внимание описанию простейших возможностей отладчика не только для того, чтобы проследить за копированием переменной d в переменную dt, а ещё и для того, чтобы при необходимости можно было воспользоваться им для поиска ошибок в программе.

Итак, при использовании записей обычно приходится обрабатывать каждое поле записи отдельным оператором, однако для копирования записей можно использовать оператор присваивания.

Мы рассмотрели не все возможности записей в FreePascal и Delphi, однако нам этого пока будет достаточно. Перейдём к задачам.

---

[2] Exitcode=0 ещё не значит, что программа выдаёт правильные результаты. Ведь могут быть ошибки в алгоритме.

Задача 1

Два треугольника заданы координатами своих вершин на плоскости. Периметр какого из них больше? Координаты вершин — вещественные числа.

Решение.

Периметр многоугольника — это сумма длин всех его сторон.

Длины сторон треугольника можно найти по теореме Пифагора $l = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$, где (x_1, y_1) и (x_2, y_2) — координаты вершин треугольника.

Для хранения координат точек используем запись

```
TPoint = record
           x,y : Double
         end
```

В отличие от целых, ***вещественные числа хранятся в памяти компьютера приближённо***. Поэтому проверка таких величин на точное равенство лишена смысла. Вместо этого будем считать их совпадающими, если значения периметров отличаются по модулю не больше, чем на некоторое малое число $\varepsilon > 0$.[3] Пусть $\varepsilon = 10^{-9}$.

```
{$mode delphi}
Program Sample;
Const EPS = 1e-9; //Максимальная допустимая погрешность

Type TPoint = record
                x,y : double
              end;

//Вычисление расстояния между точками A и B
Function len(const p1, p2 : TPoint) : double;
Begin
  Result := sqr(p2.x-p1.x)+sqr(p2.y-p1.y)
End;

//Вычисление периметра треугольника ABC
Function Perim(const A, B, C : TPoint) : double;
Begin
  Result := len(A,B) + len(B,C) + len(C,A)
End;

//Основной алгоритм
Var A,B,C : TPoint;
    per : array[1..2] of Double; //Периметры треугольников
    i   : Byte;
Begin
  For i:=1 To 2 Do
  Begin
    WriteLn('Координаты вершин ',i,'-го треугольника x,y');
    Read(A.x,A.y,B.x,B.y,C.x,C.y);
    per[i]:=Perim(A,B,C);
  End;

  If abs(per[1]-per[2])<=EPS Then
    WriteLn('Периметры треугольников равны')
  Else If per[1]<per[2] Then
         WriteLn('Периметр первого треугольника меньше')
       Else
         WriteLn('Периметр первого треугольника больше')
```

[3] ε читается «эпсилон»

```
End.
```

Функция abs(x) вычисляет модуль числа x. В этой программе функция abs применяется для вычисления модуля разности периметров треугольников и последующего сравнения с величиной допустимой погрешности. Если такой способ сравнения кому-то кажется излишним, попробуйте запустить эту программу и ввести следующие данные.

Координаты вершин первого треугольника (0.1,100.11), (20,20), (70,90)
Координаты вершин второго треугольника (0.5,100.51), (20.4,20.4), (70.4,90.4)

Второй треугольник смещён относительно первого на 0.4 по оси *x* и на столько же по оси *y*, т.е. получен из первого при помощи параллельного переноса. Значит, эти треугольники равны. Поэтому равны и их периметры. Это и покажет наша программа.

Теперь попробуем задать значение EPS=0. Этим мы переходим к простому, казалось бы «точному», сравнению вещественных чисел. При запуске программы с теми же исходными данными она даёт уже другой результат: периметр первого треугольника меньше. Это противоречие подтверждает, что вычисления на компьютере с вещественными числами не являются точными. Имеет смысл проверять только приближённое равенство вещественных чисел.

При передаче параметров в подпрограммы в программе Sample используются параметры-константы. Можно было передавать их и любым другим способом, например как параметры-значения. Но параметры-константы передаются по адресу. А адрес любой переменной занимает 4 байта памяти в отличие от записи TPoint, размер которой не меньше суммарного размера всех входящих в неё полей: двух величин типа Double по 8 байт каждая. Поэтому наш подход будет эффективнее. Программа должна работать немного быстрее.

Задача 2. Кёрлинг[4]

Имя входного файла:	curling.in
Имя выходного файла:	curling.out
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Как известно, с 1998 года кёрлинг признан олимпийским видом спорта. Во время просмотра Зимних Олимпийских игр в Ванкувере премьер-министр Флатландии очень заинтересовался этим видом спорта и решил, что на олимпиаде 2014 года его страна обязательно должна участвовать в соревнованиях по кёрлингу, причем, как в женских, так и в мужских.

Позже выяснилось, что в стране пока что нет ни одной площадки для этой игры. Премьер-министр решил подойти к этому вопросу серьезно и приказал построить как минимум по одной площадке в каждом крупном городе. Первой начала строиться площадка в самом центре Сен-Флатбурга (столицы Флатландии). Она уже почти построена, для показа текущего счета уже куплено самое большое табло, но для него пока не написано программное обеспечение. Вам поручено написать программу, которая будет считать текущий счет партии в кёрлинг.

В кёрлинг играют две команды. Каждая игра состоит из десяти партий, которые называют *эндами*. Если десять эндов заканчиваются вничью, то назначается дополнительный одиннадцатый экстра-энд. В каждом энде команды по очереди «играют» камни. В течение энда некоторые камни могут выйти из игры, в этом случае их убирают с площадки.

После того, как каждой командой сыграны все восемь камней, производится подсчет очков в энде. Команда, чей камень оказался ближе всего к центру дома, считается выигравшей энд. Она получает по одному очку за каждый камень, оказавшийся строго ближе к центру, чем любой из камней противника. Текущий счет определяется как сумма очков по каждому из завершенных эндов.

[4] Цикл интернет-олимпиад для школьников, сезон 2009–2010. Пятая индивидуальная олимпиада. 27 февраля 2010 года. (Проводятся Санкт-Петербургским государственным университетом ИТМО)

В игре по керлингу прошло несколько эндов. Для каждого из них задано конечное расположение камней, оставшихся в игре. Необходимо определить текущий счет.

Формат входного файла

Первая строка входного файла содержит единственное число n ($1 \le n \le 11$) — количество завершенных эндов. Далее следуют описания каждого из эндов в следующем формате. Первая строка описания содержит два числа m и k ($1 \le m, k \le 8$) — количество оставшихся в игре камней, принадлежащих соответственно первой и второй командам. Следующие m строк содержат по два целых числа, не превосходящих 1000 по модулю — координаты соответствующего камня первой команды. Следующие k строк также содержат по два целых числа, не превосходящих 1000 по модулю — координаты соответствующего камня второй команды. В одном энде никакие два камня не расположены в одной точке.

Центр дома находится в начале координат — точке с координатами (0, 0).

Формат выходного файла

В выходной файл выведите текущий счет в партии в формате a:b, где a — текущее количество очков у первой команды, b — текущее количество очков у второй команды.

Примеры

curling.in	curling.out
2 2 2 1 1 2 2 0 1 -1 -1 2 2 0 0 1 0 -1 -1 100 100	2:1
1 1 1 10 0 0 10	0:0

Решение

Для определения победителя вовсе не нужно знать точное расположение камней на площадке после окончания каждого из эндов. Чтобы узнать, сколько очков в данном энде получил победитель энда, достаточно располагать информацией об удалении каждого камня от центра дома после окончания этого энда и о принадлежности камней той или другой команде.

Будем пока считать, что энд *не* завершился вничью.

1. Упорядочим камни по возрастанию расстояния от центра дома (далее просто «расстояния»). Тогда первые камни будут принадлежать победителю энда. Например, камни могут быть расположены на таких расстояниях.

Таблица 1

2	10	15	15	15	15	30	40	50	90

Здесь камни команды-победителя помечены подчёркиванием прямой линией и жёлтым фоном. Камни соперника помечены подчёркиванием волнистой линией.

2. Кто победил, ясно по метке первого камня.

3. Начнём с первого камня: j:=1. Пока соседний справа камень (с номером j+1) принадлежит победителю, будем передвигаться на этот соседний камень: j:=j+1. Такой циклический процесс непременно закончится, ибо у каждой команды, в том числе и проигравшей, есть хотя бы по одному камню. Теперь j не больше[5] количества камней, находящихся на расстоянии, меньшем или равном расстоянию до первого камня проигравшей команды. (В нашем примере j=4.)
4. Двигаемся влево (j:=j-1) до тех пор, пока расстояние до камня номер j равно расстоянию до первого камня проигравшей команды. (В нашем примере дойдём до j=2.) Таким образом найдём последний камень, который даёт очки победителю энда. Победитель получит ровно j очков.
   Нет ли опасности выхода за пределы массива камней влево? Если игра не закончится вничью, то нет. Слева всегда найдётся камень с меньшим расстоянием. А вот если игра завершается вничью, то выход за пределы массива обязательно произойдёт. Чтобы учесть и этот «ничейный» случай, можно просто поместить слева, в позиции номер ноль, барьерный камень с фиктивным отрицательным расстоянием. Тогда в случае ничьей после выхода из цикла окажется j=0.
5. Победителю энда добавляем j очков. (Даже если энд завершился вничью, то можно владельцу первого камня добавить j очков. Ведь в этом случае j=0.)

Определять расстояние до камня с целочисленными координатами (x, y) приходится по теореме Пифагора: $r = \sqrt{x^2 + y^2}$. Расстояние получится вещественным числом. А их сравнивать на компьютере, как мы уже видели, затруднительно. Поэтому вместо расстояний будем хранить и сравнивать их квадраты. Они-то будут целыми числами.

Какой алгоритм сортировки предпочесть? Может быть, следует применить алгоритм быстрой сортировки? Нет, не надо. У нас будет маленький массив, не больше 16 элементов. При таких размерах массивов лучше работают прямые методы сортировки. Остановимся на сортировке прямым включением с барьерным элементом[6]. (Теперь размер массива будет максимум 17 элементов.)

```
{$mode delphi}
Program curling;
Const MAXM=8; //Количество камней у одной команды в начале энда

Type TStone = record  //Описание камня
                r : Integer; //Квадрат расстояния камня от центра дома
                team : Byte; //Номер команды, которой принадлежит этот камень
              end;
     TLocations = array[0..2*MAXM] Of TStone;

//Ввод расположения камней после окончания энда
//n - общее количество камней на площадке, обеих команд вместе
Procedure InputLocations(var f : Text; out n : Byte; out p : TLocations);
Var mk : array[1..2] Of Byte; //Количество камней у первой и
                              //второй команды соответственно
    team  : Byte; //Номер команды
    i,j   : Byte;
    x,y   : Integer;
Begin
  Read(f,mk[1],mk[2]);
  n:=mk[1]+mk[2];

  j:=0; //Номер камня при сплошной нумерации, вне зависимости от команды
  For team:=1 To 2 Do
  For i:=1 To mk[team] Do
  Begin
```

[5] Почему «не больше?» Смотрите таблицу 1. Могут найтись камни с одинаковым расстоянием. Кроме того, после сортировки камни с одинаковым расстоянием следуют в произвольном порядке, вне зависимости от принадлежности той или иной команде.

[6] См. занятие 7

```
    Read(f,x,y);
    j:=j+1;
    p[j].r:=x*x+y*y;
    p[j].team:=team
  End
End;

//Сортировка камней по возрастанию расстояния от центра дома (прямым включением)
//n - количество элементов массива a
Procedure Sort(var a : TLocations; n : Byte);
Var i,j : Byte;
    x   : Integer;
Begin
  For i:=2 To n Do
  Begin
    x := a[i].r;
    a[0].r := x;
    a[0].team := a[i].team;
    j := i;
    While x < a[j-1].r Do
    Begin
      a[j]:=a[j-1];
      j:=j-1;
    End;
    a[j].r:=x;
    a[j].team:=a[0].team
  End
End;


//Основной алгоритм
Var p : TLocations;
    f : Text;
    n : Byte;  //Количество завершённых эндов
    count  : array[1..2] Of Integer = (0,0); //Счёт
    i,j,sz : Byte;
    winner : Byte; //Команда-победитель
    r      : Integer;
Begin
  Assign(f,'curling.in');
  Reset(f);
  Read(f,n);

  For i:=1 To n Do
  Begin
    InputLocations(f,sz,p);
    Sort(p,sz);
    winner:=p[1].team;
    j:=1;
    //Находим последний подряд идущий в массиве p камень победившей команды,
    //ближе которого к центру дома нет камней соперника
    While p[j+1].team=winner Do
      j:=j+1;
    //Если на таком же расстоянии от центра дома есть камень соперника,
    //не учитываем эти свои камни
    r:=p[j+1].r;
    p[0].r:=-1; //Установка барьера для случая ничейного исхода
    While p[j].r=r Do
      j:=j-1;
    //Корректируем счёт
    count[winner]:=count[winner]+j
  End;
  Close(f);
```

```
  Assign(f,'curling.out');
  Rewrite(f);
  Write(f,count[1],':',count[2]);
  Close(f)
End.
```

Задания для самостоятельной работы

1. Перепишите программу, используя везде, где это возможно, оператор присоединения.
2. Напишите программу вычитания из простой дроби $\frac{a}{b}$ простой дроби $\frac{c}{d}$. (a, c — целые, а b, d — натуральные числа, по модулю не превышающие 10^4.) Ответ должен быть представлен в виде простой несократимой дроби или, если это возможно, целого числа. Для хранения дробей используйте записи.
   Попробуйте применить отладчик для отслеживания значений переменных.
3. В результате некоторых соревнований каждая команда получила некоторое количество очков. По правилам этих соревнований в турнирной таблице команды располагаются в порядке убывания набранного количества очков. Команды, показавшие одинаковый результат, располагаются в алфавитном порядке. При этом все они занимают одинаковое место. Для любой команды верно утверждение: команда занимает -е место, если ровно $k-1$ команда набрала строго больше очков, чем она. Напишите программу для подведения итогов соревнований.
   Формат входного файла
   В первой строке файла *input.txt* содержится целое число n — количество команд, участвовавших в соревнованиях ($0 < n \leq 100$).
   В каждой из следующих n строк содержатся разделённые ровно одним пробелом название команды, не содержащее пробелов, и целое число m — количество очков, набранное командой ($0 \leq m \leq 100$).
   Формат выходного файла
   В выходной файл *output.txt* выведите n строк турнирной таблицы. Каждая строка содержит место, занятое командой, её название и набранное количество очков, разделённые одним пробелом. Например,
   1 Динамо 70
   1 Спартак 70
   3 Локомотив 69
   4 Буревестник 65
   Указание. Для хранения данных о командах (название, количество очков) используйте записи (пусть соответствующий тип называются, например, Team). Для сравнения записей определите логическую функцию less(Team *a*,*b*), которая возвращает значение *true*, если команда *a* должна быть выше в турнирной таблице, чем *b*, и используйте её при сортировке записей.
4. Петя хочет подсчитать, сколько товаров на складе имеют цену ровно 10 копеек. Для этого он написал такую программу.

```
program Error;
Var a     : real;
    b,i,N : Integer;
Begin
  WriteLn('Количество товаров');
  Read(N);
  WriteLn('Введите цены ',N,' товаров в рублях');
  b:=0;
  For i:=1 To N Do
  Begin
    Read(a);
    If a=0.1 Then
      b:=b+1
  End;
  WriteLn('Товаров стоимостью 10 копеек ', b,' штук')
```

```
End.
```

Как вы думаете, почему эта программа работает неправильно?
А как эту задачи решили бы вы? Напишите свою программу.
(Чтобы понять, почему эта программа не работает для цены 10 копеек, но работает для цены 50 копеек, надо знать особенности представления чисел в памяти компьютера. Мы этим пока заниматься не будем. Но даже тех сведений, которые изложены в тексте этого занятия, достаточно для ответа на вопрос задачи.)

Литература

1. Michaël van Canneyt. Reference guide for Free Pascal, version 2.4.2. — November 2010
2. Borland Help для BDS2006
3. Фаронов В.В. TurboPascal 7.0. Начальный курс. Учебное пособие. — М.: «Нолидж», 1998
4. Условия задач пятой индивидуальной олимпиады цикла интернет-олимпиад для школьников сезона 2009-2010 года (27 февраля 2010 года)
   http://neerc.ifmo.ru/school/io/archive/20100227/problems-individual-20100227.pdf