14. Множества (Sets).

Множество (Set) — это очередной структурированный тип данных, предназначенный для представления группы однотипных элементов. Множество является просто набором с ограниченным диапазоном значений. Базовым для множества может служить только скалярный тип, максимальный диапазон которого ограничивается 256-ю значениями. Эти типы: Byte, ShortInt, Char, Boolean.

Общий формат объявления множественного типа имеет вид:

Type Имя — типа = Set of Базовый — тип;

В реальной жизни множества используются для классификации, т.е. для объединения элементов в различные группы по какому-либо признаку. Рассмотрим следующее множество целых чисел:

1,2,3,4,5,6,7,8,9, 10

На Турбо-Паскале такой тип можно определить следующим образом:

```
Type littleNumbers = Set of 1 ..10;
Var numberL : littleNumbers;
```

Строка «Type littleNumbers = Set of 1..10;» объявляет новый тип *littleNumbers,* который может представлять целые числа с минимальным допустимым значением — 1 и с максимальным — 10. В данном примере базовым типом для элементов множества служит тип-диапазон 1..10.

Переменная *numberL* является множеством, которое может содержать произвольное количество (максимум 256) элементов базового типа. При этом, элементы, содержащиеся в множестве, заключаются в квадратные скобки, например:

```
numberL := [1,3,5,7,9];
```

Рассмотрим другие примеры задания множественного типа с различными допустимыми базовыми типами:

```
Var sez :Set of(Winter,Spring,Summer,Autumn);
         bset: Set of Byte;
         cset: Set of Char;
         cc: Set of'a'..'z';
         Iset: Set of Boolean;
         logical: Set of False..True;
         dset: Set of 1..200;
```

14.1. Операции, определенные на множества.

Основными операциями, операторами, определенными для множества, являются:

:=	–	Присвоение значения множеству;
In	–	Проверка, содержится ли какая-либо величина в данном множестве — операция включения;
+	–	Объединение множеств;
-	–	Разность множеств;
*	–	Пересечение множеств.

14.1.1. Присвоение значения множеству.

Инициация множества осуществляется оператором присвоения. Например, инициация пустого множества из вышеприведенного примера будет иметь вид:

```
numberL := [ ];
```

С другой стороны, оператор:

```
numberL := [2..6];
```

присваивает множественной переменной *numberL* подмножество (целые числа от 2 до 6) из заданного диапазона, определенного базовым типом. Заметим здесь, что если переменной присвоить зна-

чение, которое выходит за область определения типа, компилятор будет генерировать логическую ошибку. Так, например, оператор:

numberL := [6..32];

нелегален, так как присваиваемые значения не входят в заданный для типа *littleNumbers* диапазон.

14.1.2. Операция включения.

Рассмотрим такой пример: пусть задаются целые числа, и нам надо определить, содержится ли данное число в множестве *littleNumbers* или нет. Пусть числа считываются с клавиатуры, а признак окончания чтения — ввод числа 0. С этой целью нам понадобится использовать *оператор включения* «In»:

```
Program testNumber;
Type littleNumbers = Set of 1.. 10;
Var numberL: littleNumbers;
    value:Integer;
Begin
numberL:=[1,,6];
Repeat
    Writeln(' Введите целое число  (0 — для конца)');
    Readln(value);
    If value <> 0 Then
    begin
            { Проверка на принадлежность к множеству}
            If value IN numberL
            Then Writeln(' Число из интервала [1,6]')
            Else Writeln(' Число не из интервала [1,6]')
   End
Until value = 0
End.
```

14.1.3. Объединение множеств.

Иногда требуется объединить несколько множеств в одно. Рассмотрим пример, в котором показывается, как объединить два множества, получая третье, используя оператор сцепления «+»:

```
Program testNumber;
Type numbers = Set of 1 ..50;
Var numberLittle, numberBig, numberUnion:numbers;
    value:Integer;
Begin
    numberLittle:= [1..6];
    numberBig:= [40..50];
    { Объединение трех множеств }
    numberUnion:= numberLittle + numberBig + [5..20];
    Repeat
        Writeln(' Введите целое число  (0 — для конца)');
        Readln(value);
        If value <> 0 Then
        begin
                If value IN numberUnion
                Then Writeln(' Число содержится в множестве')
                Else Writeln(' Число не содержится в множестве')
    End
    Until value = 0
End.
```

Здесь строка

```
Var numberLittle, numberBig, numberUnion:numbers;
```

объявляет три переменных типа *numbers*. В строках

```
numberLittle:= [1..6];
```

и

```
numberBig:= [40..50];
```

определяются значения двух из них с помощью оператора присвоения. Строка же

```
numberUnion:= numberLittle + numberBig + [5..20];
```

определяет новое множество, которое представляет объединение трех множеств *numberLittle, numberBig* и [5..20]. При этом, если объединяемые множества содержат одни и те же элементы (как, например, 5 и 6 в множествах *numberLittle* и [5..20]), то дубликаты будут игнорироваться. В данном примере множество *numberUnion* будет содержать следующие элементы:

1,2,3,4,5,6,7,8,9,10,11,12,13,14,17,18,19,20,40,41,42,43,44,45,46,47,48,49,50.

14.1.4. Разность множеств.

Разность двух множеств по определению содержит те элементы первого множества, которые не содержатся во втором. Например, рассмотрим программу:

```
Program testNumber;
Const nMax=50;
Type numbers = Set of 1.. nMax;
    littleNumbers = Set of 1..10;
Var numberLittle, numberBig, numberSub:littleNumbers;
    i:l..nMax;
Begin
numberLittle:=[1..30];
numberBig:=[21..50];
{Разность множеств}
numberSub:=numberLittle - numberBig;
{Вывод элементов множества на экран }
Writeln('Разность: [1..30] - [21..50] = ');
Write(' {');
For i:=l to nMax Do
    If i IN numberSub Then Write(i:3);
Writeln('}');
End.
```

Результирующее множество будет состоять из элементов:

1,2,3,4,5,6,7,8,9,10,11,12,13,14,17,18,19,20.

Обратите внимание на фрагмент вывода элементов множества на экран:

```
Write(' {');
For i:=l to nMax Do
If i IN numberSub Then Write(i:3);
Writeln('}');
```

Здесь используется цикл с перебором всех дозволенных элементов из диапазона определения множества, и если элемент входит в интересующее нас в данный момент множество, он выводится на экран. Это неудобство связано с тем фактором, что прямой вывод множества на экран в Турбо-Паскале недопустим.

14.1.5. Пересечение множеств.

Множество, полученное при пересечении двух множеств, содержит только общие элементы обоих множеств. Например,

```
Program testNumber;
Const nMax=50;
Type numbers = Set of 1 ..nMax;
    littleNumbers = Set of 1..10;
Var numberLittle, numberBig, numberSub: littleNumbers;
Begin
     numberLittle:= [ 1..30];
     numberBig:=[21..50];
     {Пересечение множеств}
     numberSub:= numberLittle * numberBig * [45..50];
     {Вывод элементов множества на экран }
     Writeln('Пересечением [1..30] и [21..50] будет:');
     Write(' {');
     For i:=l to nMax Do If i IN numberSub Then Write(i:3);
     Writeln(' }');
     End.
```

Результирующее множество будет состоять из элементов:

21,22,23,24.25,26,27,28.29,30,45,46,47,48.49,50.

14.2. Несколько примеров использования множеств.

Программа-пример 1.

Пусть пользователь вводит набор «интересных» символов с клавиатуры (признак конца ввода — набор нуля), а затем текстовую строку. Программа вычисляет количество выбранных символов в заданной строке.

Алгоритм:

Инициация:
    СпецМножество =Ø
    Счетчик = 0;
Повторять
    { Считать СпецСимвол;
    Если (*СпецСимвол* отличен от 0)
    Тогда Занести *СпецСимвол* в СпецМножество;
    Пока—не (СпецСимвол = 0);
    }
Считать *Строку,*
Для i = от 1 доДлины*(Строки) Делать*
    Если i-ый элемент *Строки* содержится в *СпецМножестве*
    Тогда Увеличить *Счетчик,*
Вывести на экран значение *Счетчика.*

Программа:

```
Program calcSymbols;
{ Подсчитывает количество символов в строке }
Var st: String;
        i, symNumber: Integer;
```

```
        ch:Char;
        symbolSetSet of Char;

Begin
    symbolSet:= [];
    Write (' Введите «интересные» символы:');
        Writeln(' ':10,' Признак конца ввода — символ 0');
{ Формирование множества «интересных» символов }
    Repeat
        ch:=ReadKey;
        symbolSet:= symbolSet + [ch],
Until ch='0';
Writeln(' Введите строку.');
Readln(st);
symNumber:=0;
For i:=l to Length(st) Do
    If st[i] in symbolSet Then Inc(symNumber);
Writeln(' В данной строке всего', symNumber,
                    '«интересных» символов');
End.
```

Программа-пример 2.

Программа считывает целое число и проверяет, является ли оно простым. Для тестировки используется логическая функция, которая принимает участие в формировании условного оператора.

```
Program testPrimes;
Uses Grt;
{ Проверяет, является ли заданное число простым }
Var n:Byte;
    c:Char;

Function prime (i: Byte): Boolean;
Var j: Byte;
    primesSet: Set of Byte;
Begin
    primesSet := [];
```

```
        For j := 2 to i Div 2 Do
        If ((i mod j) = 0) Then primesSet := primesSet + [j];
    If (primesSet = []) And (Abs(n) >= 2) Then prime:=True
                                          Else prime := False;
End;

BEGIN
  Repeat
    Writeln;
    Write(' Enter integer number: ');
    Writeln(' ': 10, '(0 - for the end)');
    Read(n);
    If prime(n) Then Writeln(n,' is a prime number!')
                Else Writeln(n,' is not a prime number!');
    Until n = 0;
    Writeln;
    Writeln('Good-bye!');
    c:=ReadKey
END.
```