18.12.20г\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

19ИСиП 1Д \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Основы алгоритмизации и программирования

ТЕМА: П\З **Работа с данными типа множество**

СОЗДАНИЯ ПРОГРАММЫ

Изначально создание программы можно разделить на следующие этапы:

1. Разработка алгоритма работы программы.
2. Дизайн пользовательского интерфейса. Размеры и свойства окна программы, расположение элементов управление внутри окна (окошки, кнопочки, менюшки и другие) и так далее.
3. Написание логики программы. Это именно та часть, которая реагирует на события от нажатия кнопок, события операционной системы и выводит результат своей работы на экран.

**Множественный тип**

Еще одним сложным стандартным типом данных, определенным в языке Паскаль, является множественный тип. Значением множественного типа данных является множество, состоящее из одно­типных элементов. Тип элемента множества называется базовым типом. Базовым типом может быть скалярный или ограниченный тип. Таким образом, множество значений множест­венного типа - это множество всех подмножеств базового типа, включая и пустое множество. Если базовый тип содержит N элементов, соответствующий множественный тип будет содержать 2Nэлементов.

Характерное отличие множественного типа - определение на нем наиболее рас­про­­страненных теоретико-множественных операций и отношений. Это делает мно­жес­т­венный тип похо­жим на простые типы данных. Множественные типы описываются в разделе типов следующим образом :

**Type < имя типа > = Set of < базовый тип>**

Например,

а) Type Beta = Set of 100..200;

б) Type Glas = Set of char ; {Vowel}

в) Type Color = (red, orange, yellow, green, light\_blue, blue, violet);

Paint = Set of Color;

г) Type TwoDigNum = Set of 10..99;

Var A, B: Beta;

llet, flet: Glas;

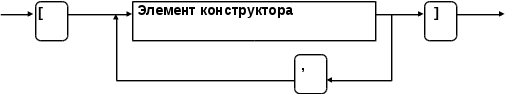
last, first: Paint;

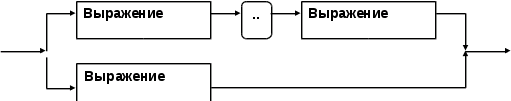
Sinit: TwoDigNum;

**Конструктор множества**

Множества строятся из своих элементов с помощью конструктора множества. Кон­структор представляет собой перечисление через запятую элементов множества или отрезков базо­вого типа, заключенное в квадратные скобки [ , ]. Пустое множество обозначается через [].

**конструктор**





**Элемент**

**конструктора**

Например:

[ ] - пустое множество

[2, 5 ..7] - множество {2, 5, 6, 7}

['A'..'Z', 'O'..'9'] - множество, состоящее из всех прописных латинских букв и цифр

[i + j .. i + 2\*j] - множество, состоящее из всех целых чисел между i + j и i + 2j

Отметим, что если в выражении [v1..v2] v1 > v2, множество [v1 .. v2] - пустое.

**Операции и отношения**

К операндам - однотипным множествам А и В применимы следующие операции:

А + В - объединение А ∪ В

А \* В - пересечение А ∩ В

А - В - разность А \ В

Между А и В определены также отношения порядка и равенства

А = В, А <> В, А < В, А <= В, А > В, А >= В;

Отношения порядка интерпретируются как теоретико-множественные включения.

Если А - множество и х - элемент базового типа, то определено отношение принад­леж­ности **х in A** - x принадлежит A ( x ∈ A ).

Каждое из отношений, описанных выше, по-существу является операцией, результат которой имеет тип Boolean. Таким образом, если Init - переменная типа Boolean, возможно присваивание Init := A < B. Возможны такие сравнения

( А = В ) = ( С = D ).

Наличие операций над множествами позволяет применять в программах операторы присваивания, в левой части которых стоит переменная типа множества, а в правой - выражение того же типа. Например :

А := А \* [1 .. 10] + B ; B := (А + B)\*['A' .. 'Z'] ;

**Примеры**

При реализации языка размеры множеств всегда ограничены константой, зависящей от реализации. Обычно эта константа кратна длине машинного слова. Это происходит потому, что мно­жества реализованы в виде логических (двоичных) векторов следующим образом: каждой координате двоичного вектора однозначно соответствует один из элементов базового типа. Если эле­мент a принадлежит представляемому множеству A, то значение координаты вектора, со­ответствующее a, равно 1. В противном случае значение соответствующей координаты равно 0.

Например, если множество A описано как Set of 0..15, то его представляет 16-ти мерный двоичный вектор, координаты которого перенумерованы от 0 до 15, и i-той координате соот­ветствует элемент i базового типа.

Базовый тип : 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Двоичный вектор : 0 0 1 1 0 1 0 1 0 0 0 1 0 1 0 0

Представленное множество : A = [2, 3, 5, 7, 11, 13]

Такой способ реализации позволяет быстро выполнять операции над множествами и проверки теоретико-множественных отношений. Поэтому, например, вместо

For X := 'A' to 'Z' do

If (X ='A') or (X ='E') or (X ='I') or (X ='O') or (X='U')

then Statement1

else Statement2

лучше написать

For X := 'A' to 'Z' do

If X in ['A','E','I','O','U']

then Statement1

else Statement2

Последняя форма записи не только лучше читается, но и гораздо быстрее вычисляется.

В системе Turbo-Паскаль максимальное количество элементов в множестве равно 256. Таким образом, в качестве базового типа можно выбрать, например, Char или отрезок 0..255. В заклю­че­ние раздела приведем пример программы, использующей множественные типы данных.

Пример. Построить множество всех простых чисел из отрезка 2..n (n ≤ 255).

Метод, с помощью которого мы это сделаем, известен как "Решето Эратосфена". Суть этого метода в следующем: Пусть Prime - строимое множество простых чисел и Grating - множество, называемое решетом. Алгоритм начинает работу с Prime = []; Grating = [2..n].

Шаг основного цикла:

а. Наименьший элемент Grating поместить в Prime;

б. Удалить из Grating все числа, кратные этому элементу;

Алгоритм заканчивает работу при Grating = []

Program EratosfenGrating;

Const n = 255;

Var Grating, Prime: set of 2 .. n ;

i, Min : integer ;

Begin

Grating := [2 .. n] ; Prime := [] ; Min := 2; {инициализация}

While Grating <> [] do begin {основной цикл}

While not(Min in Grating) do {поиск наименьшего элемента в решете}

Min := Min + 1;

Prime := Prime + [Min] ; {пополнение множества простых чисел}

For i := 1 to n div Min do {исключение кратных из решета}

Grating := Grating - [i\*Min];

end;

Writeln('Primes: '); {вывод множества простых чисел}

For i := 1 to n do

If i in Prime then write(i, ', ')

End.

Отметим, что доступ к элементу множества в языке не предусмотрен. В этом - еще одно качественное отличие множественного типа от других сложных типов данных. Поэтому, напри­мер, для вывода множества Prime приходится перебирать все элементы базового типа и каждый из них проверять на принадлежность Prime.

**Предварительное распределение памяти и контроль типов.**

Любая встречающаяся в каком-либо операторе из раздела операторов переменная должна быть описана в разделе переменных. Описание переменной связывает с новой переменной ее имя и тип. Информация, содержащаяся в разделе переменных, используется компилятором для:

1.Распределения памяти. Распределение (резервирование) памяти для переменных, описанных в разделе переменных, производит компилятор на этапе генерации кода. Для каждой переменной в ОЗУ отводится определенное место. Размер этой части памяти определяется типом переменной.

2.Правильной интерпретации действий над данными. Например, сложение целых чисел интерпретируется не так, как сложение вещественных чисел или строк.

3.Контроля правильности применения переменных. Ошибка, допущенная при написании переменной в разделе операторов, приведет к сообщению об синтаксической ошибке, так как эта переменная не описана в разделе переменных.

**Ввод/вывод и внешние вычислительные структуры.**

Программа, написанная на языке программирования, должна каким-то образом обмениваться дан­ными с внешними устройствами (получать данные с клавиатуры, магнитного диска, выводить дан­ные на экран, принтер и т.д.) Многообразие типов внешних устройств, не говоря уже разнообразии фирм-производителей ВУ, ставит перед теоретиками программирования и системными программистами проблему разработки соответствующих абстракций, в рамках которых взаимодействие программы и ВУ может быть описано. Абстрактные вычислительные структуры, описывающие ввод/вывод называют внешними.

Проблема исследования и реализации внешних вычислительных структур в значительной степени повлияла на широкое распространение ОО подхода к проектированию ВС. В современных системах программирования ВУ во всем своем многообразии описываются как виртуальные объекты, взаимодействующие друг с другом в соответствии со своими протоколами. Но сначала были файлы.

Преподователь \_\_\_\_\_\_\_\_\_\_\_\_\_\_Дузаев И.К.