Дмитриева Марина Валерьевна
Павлова Марианна Владимировна

ЭЛЕМЕНТЫ ТЕОРИИ МНОЖЕСТВ. СИСТЕМА: ЕЕ СТРУКТУРА И СОСТОЯНИЕ.

Множества и операции над ними.

Множеством называют определенную совокупность объектов. Один из создателей теории множеств Георг Кантор(1845-1918) сказал : ″Множество есть многое, мыслимое нами как единое″. Строгого определения множества нет, так как это понятие, из которого выводятся многие понятия математики, тогда как оно не выводится из других понятий и не определяется. Понятие множества столь же первично как понятие точки или числа. Синонимами слова ″множество″ можно считать такие слова как ″совокупность″, ″коллекция″, ″семейство″, ″собрание″. В дальнейшем понятие множества будет разъяснено на примерах.

Определение: Объекты, из которых составлено множество, называются **элементами** данного множества.

Нет никаких ограничений на природу элементов, составляющих множество. Например: различные предметы меблировки комнаты образуют множество. Множествами являются книги некоторой библиотеки, учащиеся класса, буквы алфавита, автомобили на дорогах города, целые числа от 1 до 1000, атомы серебра в данной монете или всевозможные идеи, которые имело человечество.

Определение: множество, имеющее конечное число элементов, называется **конечным**.

Все рассмотренные ранее примеры служат примерами конечных множеств.

Определение: множество называется **бесконечным**, если оно состоит из бесконечного числа элементов.

Например, множество всех вещественных чисел бесконечно.

Определение: множество, в котором нет ни одного элемента, называют **пустым** множеством.

Например, множество летающих бегемотов пусто.

Для обозначения множества используют заглавные буквы латинского алфавита, а в фигурных скобках через запятую выписывают его элементы. Пустое множество записывают как {} или ∅. Если некоторый элемент принадлежит множеству, то это обозначают знаком ″∈″, а если не принадлежит, то знаком ″∉″.

Например: A={3,5,6}; M={♠,♣,♦,♥}; L={}; 5∈ A; ♦∈ M; 2∉ A; ♣∉ L.

Существует два способа задания конечных множеств. Можно либо дать полный перечень элементов этого множества, либо указать правило для определения того, принадлежит или нет рассматриваемому множеству данный объект. Первый способ называется перечислением множества, а второй - описанием. Для бесконечных множеств возможен лишь второй способ.

Например, перечисление {2,4,6,8} или описание ″Четные числа большие 0 и меньшие 10″; перечисление {вторник, среда, четверг} или описание ″Первый, второй и третий дни после понедельника″.

Определение: множество, состоящее из некоторых элементов другого множества, называется его **подмножеством.** Утверждение, что множество A является подмножеством множества B, записывают так: A⊆B. Считается, что пустое множество является подмножеством любого множества.

Фактическое содержание курса информатики в средней школе существенно зависит от технической оснащенности школы, наличия и качества компьютерных классов. В то же время теоретическая часть курса, способствующая овладению основными концепциями и базовыми компонентами информационных технологий, лишена этой зависимости.

Редакция журнала в этом и последующих выпусках намерена предлагать вниманию учителей и учащихся изложение различных разделов курса, авторами которых являются опытные преподаватели, научные работники и специалисты в области информатики. Вполне вероятно, что предлагаемые варианты окажутся уязвимыми для критики. Редакция будет благодарна всем, кто выскажет критические замечания, предложит свой вариант изложения, предложит другие темы или задаст вопросы, ответы на которые редакция постарается найти у соответствующих специалистов.

Например, множества {4,8} и {6} являются подмножествами множества {2,4,6,8};числа 2,4,6,8-его элементы. Множество { {},{2},{4},{6},{8},{2,4},{2,6},{2,8},{4,6},{4,8},{6,8},{2,4,6},{2,4,8},{2,6,8},{4,6,8},{2,4,6,8} } является множеством всех подмножеств исходного множества. Из последнего примера видно, что множества сами могут быть элементами какого-то множества.

У любого множества есть обязательно хотя бы два подмножества: пустое множество и само множество. Эти два подмножества называются несобственными подмножествами. Любое подмножество, отличное от несобственного, называется собственным подмножеством данного множества.

Нетрудно видеть, что у пустого множества нет собственных подмножеств, а оба несобственных подмножества равны между собой. У любого одноэлементного множества также нет собственных подмножеств, но его несобственные подмножества различны. У любого двухэлементного множества есть уже два собственных подмножества. С ростом количества элементов в множестве количество собственных подмножеств растет.

Например, если F={s, t}, то собственными подмножествами множества F будут являться множества {s} и {t}.

Определение: множества, состоящие из одних и тех же элементов, называются **равными**. При этом порядок перечисления элементов множества значения не имеет.

Например: равны множество равносторонних треугольников и множество равноугольных треугольников; равны множества {7,4,1},{1,4,7} и {7,1,4}.

Определение: **мощностью** конечного множества (размером конечного множества) - называют количество его элементов. Мощность пустого множества равна нулю.

Например: мощность множества углов любой трапеции равна четырем, а мощность множества четных чисел, принадлежащих отрезку [1, 10] равна 5.

Существуют несколько основных операций над множествами. С их помощью можно строить новые множества.

Определение: **объединением** множеств A и B называется множество, содержащее все элементы из A и все элементы из B.

Объединением любого множества с пустым является само это множество.

Объединение множеств обозначается знаком $\cup$.

Например, пусть A={5,7,9}, B={6,1}. Тогда C=A$\cup$B={1,5,6,7,9}.

Определение: **пересечением** множеств A и B называется множество, состоящее из тех элементов, которые принадлежат обоим множествам.

Если в пересечении множеств элементов нет, то говорят, что они не пересекаются или что их пересечение - пустое множество. Пересечение любого множества с пустым является пустым множеством. Пересечение множеств обозначается знаком $\cap$.

Например, пусть A={4,5,7,9,10}, B={3,7,9}. Тогда C=A$\cap$B={7,9}.

Определение: **разностью** множеств A и B называется множество тех элементов из A, которые не принадлежат множеству B.

Если из любого множества A вычесть пустое множество, то результатом вычитания будет само множество A. Разность множеств обозначается знаком $\setminus$.

Например, пусть A={1,3,5,6,7}, B={6}. Тогда C=A \ B={1,3,5,7}.

Если G={гусь, кот, пёс, утка, курица, индюк}, H={кот, пёс}, то R=G\H= {гусь, утка, курица, индюк}, а T=H \ $\varnothing$=H.

Отношения и их свойства.

Мы рассмотрели множества, состоящие из каких-то элементов, но ничего не говорили о свойствах этих элементов и их отношениях друг с другом. Рассмотрим теперь некоторое конечное множество элементов: A={a_1,a_2,...a_N}, которое будем называть предметной областью.

Определение: функция называется **логической,** если область её значений состоит из двух логических констант **истина** и **ложь,** или другими словами, если все её значения принадлежат множеству {**истина, ложь**}

Определение: **предикатом** называется логическая функция, определенная на предметной области.

Предикаты делятся на два вида: понятия и отношения.

Определение: **понятие** - свойство одного объекта предметной области. Например, если предметная область - некоторый отрезок натурального ряда, то предикат четности - это свойство какого-то элемента (числа) из этого отрезка быть или не быть четным. Понятия называются одноместными предикатами.

Определение: **отношение** - свойство пар, троек, четверок и т.д. объектов данной предметной области.

Например, свойство одного элемента лежать между двумя другими является отношением (трехместным предикатом).

Напомним, что все элементы, про которые мы выясняем, находятся ли они в данном отношении, должны принадлежать рассматриваемой предметной области. Чаще всего рассматривают двуместные отношения и используют следующие обозначения: $<$ - меньше, $\leq$ - меньше или равно, $=$ - равно, $>$ - больше, $\geq$ - больше или равно, $\neq$ - не равно.

Если в качестве предметной области рассмотреть A - отрезок натурального ряда от 3 до 100, то отношение $x < y$, где $x, y \in A$, имеет значение **истина**, например, при x=6, y=123. Отношение $x = y$ имеет значение **ложь**, например, при x=50, y=49, а отношение $x \neq y$ имеет значение **истина** при тех же значениях переменных.

Нетрудно привести примеры и других не столь известных отношений. Так отношением будет являться

логическая функция, которая задана на множестве натуральных чисел, определяющая, будет ли одно натуральное число делиться на другое натуральное число без остатка. Отношением также является функция, выясняющая, какая из двух лампочек горит ярче. Кроме того, примерами отношений служат предикаты, определяющие состоят ли два человека в браке, является ли один человек родителем другого и т. п.

Рассмотрим свойства двуместных отношений.

Будем обозначать ″Θ″ - знак отношения. Тогда отношение f Θ g может обладать следующими свойствами:

— отношение называется **рефлексивным**, если f Θ f для любого f ∈ A;

Например, рефлексивность имеет место для отношений = , <=, >= , т.к. 5=5, 7≤7, 9≥9.

— отношение называется **транзитивным**, если из того, что f Θ g и g Θ h следует, что f Θ h для любых f, g, h∈ A .

Например, транзитивностью обладают отношения =, <=, >=, <, >, так как из того, что 5 < 7 и 7 < 10 следует, что 5 < 10.

— отношение называется **симметричным**, если из того, что f Θ g следует, что g Θ f для любых f и g ∈ A.

Например, пусть предметная область - отрезок целых чисел от 2 до 10000. Рассмотрим отношения:

а) свойство x и y иметь общий делитель,

б) свойство x и y быть равными.

Очевидно, что оба эти отношения будут симметричны.

— отношение называется **антисимметричным**, если из того, что f Θ g и g Θ f следует, что f = g для любых f, g ∈ A .

Например, антисимметричными являются отношения ≤ и ≥, поскольку, если x≤y и y≤x, то y=x для любых x,y∈ A. Отношение, которое определяет делится ли одно число без остатка на другое, антисимметрично, так как если x делится на y, а y делится на x, то x=y.

Заметим, что симметричность и антисимметричность не являются взаимоисключающими свойствами. Например, пусть множество A - множество людей. Определим отношение β так, что для любых x, y ∈ A отношение x β y истинно тогда и только тогда, когда x брат y. В семье, где два брата p и q и сестра r, то есть на предметной области A={p, q, r} отношение β не является симметричным, так как p β r (p является братом r), но неверно, что r β p (r не является братом p). Это отношение не является и антисимметричным, так как p β q (p брат q) и q β p (q брат p), хотя p и q различны.

Использование множеств при решении задач на компьютере.

Для того, чтобы продемонстрировать, как используются отношения, множества и операции над ними в программировании, мы выбрали язык Паскаль, так как именно на нем можно описывать и обрабатывать данные типа множество. Множество может строиться из объектов перечислимого типа или типа диапазона. Тип множество - составной тип, и его описание выглядит следующим образом:

Type множество=set of тип-элемента,

где ″множество″- идентификатор определяемого типа, а ″тип-элемента″ - описатель типа элементов множества. Обычно каждая система программирования на Паскале накладывает дополнительные ограничения на возможные типы элементов.

Для представления множеств в программе служит специальная конструкция, позволяющая задать набор элементов, представляющих какое-либо множество. Конструкция представляет собой набор выражений, перечисленных через запятую, заключенный в квадратные скобки. Перечисленные выражения и задают значения элементов множества, например,

['A', 'D', 'N'] [5, i+1, j-7, k, 43] ['a', 's', 'f', 'j', 'l'] [x] [3,8,4,2,6,1,5,7]

Пустое множество задается так: []. Для задания в качестве элементов множества последовательности из нескольких подряд идущих значений можно указать в изображении множества первое и последнее из включаемых значений, соединив их символом ″..″, например:

[23..67] [2..5, 8..34] ['0'..'9'] ['A'..'Z', 'a'..'z'] ['a'..'я'] ['Ч','Щ'..'Э','Я']

Для работы с множествами определены операции, позволяющие сравнивать множества между собой, определять принадлежность элемента множеству, выполнять обычные теоретико-множественные операции нахождения объединения, пересечения и разности множеств. Приведем таблицу допустимых операций над множествами, пояснив каждую операцию эквивалентной записью, принятой в теории множеств. В таблице M, M1 и M2 означают множества с элементами одного и того же типа, x - значение элемента множества M. Операция принадлежности IN относится к группе операций отношения так же, как и операции отношения со знаками ″>″, ″<″ , ″=″ , ″<>″, ″<=″, ″>=″. Пользуясь этими операциями, можно выразить такие элементарные действия, как добавление в множество или удаление из него одного или нескольких элементов и другие. Например, результатом исполнения присваивания M:= M+[x] будет добавление элемента со значением x в

множество M. Аналогично, в результате выполнения оператора присваивания M:=M-[x] элемент x будет удален из множества M.

В языке Паскаль	В теории множеств
x IN M	x ∈ M
M1=M2	M1≡M2
M1<>M2	M1≠M2
M1<=M2	M1⊆M2
M1>=M2	M1⊇M2
M1+M2	M1∪M2
M1 * M2	M1∩M2
M1-M2	M1\ M2

Пользуясь аппаратом теории множеств, приведем решение нескольких задач.

Пример 1. В последовательности натуральных чисел i (i<256) определить число различных чисел и вывести их в порядке убывания. Признаком конца последовательности служит число 0.

Будем просматривать последовательность и формировать множество чисел Z, входящих в эту последовательность. Затем выведем все элементы построенного множества.

```
program p1;
  type set_byte=set of byte;
  var Z: set_byte; m:byte; k: integer;
begin writeln('Введите последовательность чисел, 0- признак конца:');
  read(m); Z:=[];
  while m <>0 do   begin Z := Z+[m]; read(m) end;
   k:=0; writeln('Числа в порядке убывания:');
   for m:= 255 downto 1 do
      if m in Z then begin write(m,' '); inc(k) end;     writeln;
   writeln('Количество различных чисел в последовательности : ',k)
end.
```

Пример 2 . Найти все простые числа, меньшие заданного N.

Можно предложить алгоритм, в котором все числа перебираются по очереди и для каждого из чисел производится проверка, является ли оно простым. Однако уже в Древней Греции был известен более эффективный алгоритм, получивший название ″решета Эратосфена″. Согласно этому алгоритму, сначала рассматривается множество всех чисел от 2 до N, а затем из него последовательно удаляются элементы, не являющиеся простыми числами. Для этого на каждом шаге алгоритма выбирается наименьший элемент множества и вычеркиваются из множества все элементы кратные выбранному. Сам элемент является простым числом и в дальнейшем не рассматривается.

Будем считать, что в программе определен тип множества resheto и переменная типа множества simple , то есть выполнены описания:

```
const N = 255;
type resheto = set of 2..N;
var simple: resheto; {Формируемое множество}
```

После выполнения оператора присваивания simple :=[2..N] переменная simple содержит множество целых значений из диапазона 2..N. Выбираем из этого множества наименьшее простое число K. Очевидно, что первым выбранным числом будет число 2. Далее исключаем из этого множества все числа кратные двум. Во время работы цикла в множестве simple содержатся все простые числа, не превосходящие K, а среди элементов, больших K, содержатся все числа, кроме кратных этим простым числам. Когда K пробежит значения до SQRT(N+1), в множестве simple останутся лишь простые числа. Следующая программа реализует описанный алгоритм:

```
program  simpleres;
  const N = 255;
  type resheto = set of 2..N;
  var simple: resheto; i,K,l: integer;
  begin writeln;    writeln ('Простые числа меньшие  ',N, ': ');
           simple :=[2..N];    L := trunc(sqrt(N+1));    K := 1;
         while K <= L do
            begin repeat  K := K+1 until K in simple ;  write(K:3,' ');
                  for i :=  2 to N div K do  simple := simple - [K*i]
            end;
       for K := L+1 to N do  if K in  simple then  write(K:3,' ')
end.
```

В программах обработки текстов иногда удобно использовать множества символов. Тип такого множества можно описать так:

```
Type  set_sym= set of char
```

Использование множеств часто оказывается эффективнее других конструкций даже в тех программах, которые впрямую не связаны с обработкой множеств, например, вместо выражения

```
(C >='0') and (C<='9') or (C >='A') and (C<='Z')
```

проще и понятнее написать C IN ['0'..'9', 'A'..'Z'], причем последняя конструкция будет, как правило, и более эффективной.

Пример 3. Определить какие из символов заданного текста являются уникальными, то есть встречаются в тексте лишь один раз.

Для решения задачи удобно использовать множества символов. При анализе текста будем формировать два множества, в множество M включим все символы, которые встречались в тексте хотя бы раз, в множество MM включим символы, которые встречались в тексте более одного раза. Первоначально оба множества пусты. После просмотра текста нужное нам множество символов получается как разность множеств M и MM. Для того, чтобы вывести результаты, необходимо перебирать все символы и для каждого символа проверять, принадлежит ли он множеству. Приведем программу, реализующую этот алгоритм:

```
program mn3;
type set_let = set of char;
var M1,M,MM : set_let;   s: string; c : char; i:byte;
begin
   writeln('Введите исходный текст: ');     readln(s);
   M:=[]; MM:=[];
   for i := 1 to length(s)
   do begin c:= s[i];
              if c in M
              then {символ уже ранее встречался}
                     MM:= MM+[c]
              else {символ встретился впервые}
                     M:= M+[c]
      end;
   M1:= M-MM;
   Writeln ('Множество уникальных символов: ');
   if M1 = [] then writeln ('пусто')
   else
   for c := chr(0) to chr (255) do if c in M1  then write(c,' ');
   Writeln;
end.
```

Пример 4. Написать программу, которая проверяет, состоят ли два заданных текста из одинаковых символов.

В программе опишем процедуру, которая по заданной строке s формирует множество различных символов m, из которых строка состоит:

```
procedure settex (var s : string; var m : set_let);
    var i : byte;
 begin   m:=[];   for i := 1 to length (s) do     m := m + [s[i]]  end.
```

Теперь для решения задачи нам достаточно по каждому из текстов построить соответствующее множество и затем эти два множества сравнить. Полностью программа может выглядеть так:

```
Program pm4;
  type set_let = set of char;
  var m1,m2 : set_let;  s: string; c : char;
  procedure settex (var s : string; var m : set_let);
      var i : byte;
  begin   m:=[];   for i := 1 to length (s) do     m := m + [s[i]]  end;
begin
   writeln('Введите первый текст:'); readln(s); settex(s,m1);
   writeln('Введите второй текст:'); readln(s); settex(s,m2);
   if m1 = m2 then writeln('Тексты состоят из одинаковых символов')
   else writeln('Тексты состоят из различных символов');
end.
```

Используя множества, легко решать следующие задачи анализа текста: найти все символы, которые встречаются в двух заданных текстах; определить символы, которые встречаются в первом тексте, но не встречаются во втором; определить, сколько различных символов в тексте и т.п.

Рассмотрим еще одну задачу, в которой требуется проанализировать текст. Предположим, что нас интересуют в тексте только буквы.

***Пример* 5.** В заданном тексте найти буквы, причем отдельно сформировать буквы латинского и русского алфавитов.

При анализе текста сформируем множество различных символов, входящих в текст, а затем с помощью операции пересечения множеств выделим соответствующие множества. Пусть в программе переменная mlat- множество букв латинского алфавита, mrus- русского, m-множество различных символов текста. Для того, чтобы определить множество букв, встречающихся в тексте, достаточно сформировать множество m*(mrus+mlat). Приведем программу, решающую задачу:

```
program mn5;
type set_let = set of char;
var mlat, mrus, m, rus, lat, lit : set_let;
      s: string; c : char;    i: word;
procedure settex (var s : string; var m : set_let);
 var i : byte;
begin    m:=[];    for i := 1 to length (s) do m := m + [s[i]]   end;
begin
mlat := ['A'..'Z','a'..'z'];      mrus := ['А'..'Я','а'..'п','р'.. 'я'];
writeln;      writeln('Введите текст:');      readln(s);    settex(s,m);
writeln('Различные символы, входящие в текст:');
for c := chr(0) to chr (255)     do if c in m    then write(c);    writeln;
writeln('Буквы русского алфавита, входящие в текст:');
rus:=m*mrus;
for c := chr(0) to chr (255)  do if c in rus  then write(c); writeln;
writeln('Буквы латинского алфавита в заданном тексте:');
lat:=m*mlat;
for c := chr(0) to chr (255)  do if c in lat    then write(c); writeln;
writeln('Текст содержит буквы:');
lit:=m*(mrus+mlat);
for c := chr(0) to chr (255)  do if c in lit then write(c);
writeln;
end.
```

Пример 6. Задан текстовый файл, содержащий русские и латинские буквы, а также, возможно, другие символы. Переписать его в другой файл, перекодировав русские буквы в латинские или сочетания латинских букв следующим образом:

Таблица перекодировки

а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
a	b	v	g	d	e	sz	z	i	j	k	l	m	n	o	p
р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
r	s	t	u	f	h	c	ch	sh	sch	-	y	-	e	ju	ja

Строковая структура текстового файла, а также все символы, не являющиеся русскими буквами, должны сохраниться.

Для решения этой задачи составим таблицу перекодировки Tabcod, в которой сопоставим каждой русской букве ее латинский эквивалент. Особым образом должны обрабатываться буквы 'Ж','Ч','Ш','Щ','Ь','Ъ','Ю','Я', поэтому составим из этих букв множество, которое надо обрабатывать особым образом. Соответствие задается в программе с помощью двух строк, содержащих кодируемые буквы (rusalf) и значения кодов (latalf). Построение перекодировочной таблицы производится процедурой InitTab. После этого обработка файла осуществляется в соответствии с построенной таблицей InitTab.

Программу, решающую задачу, можно описать следующим образом:

```
program pcodfile;
  var Tabcod: array [char] of char;  f, g : text;
{Процедура InitTab формирует таблицу TabCode, в которой каждой }
{ букве русского алфавита сопоставляется одна буква латинского }
 procedure InitTab;
   var rusalf, latalf : string; i: byte; c: char;
 begin
     rusalf:='абвгдезийклмнопрстуфхйыэ';
     latalf:='abvgdezijklmnoprstufhcye';
    for c := chr(0) to chr(255) do  Tabcod[c] := c;
    for i :=1 to length(latalf) do     Tabcod[rusalf[i]] := latalf[i]
 end ;
 procedure CodeFile(var f,g : text);
   var lr: set of char; c: char;
 begin lr:= ['ж','ч','ш','щ','ь','ъ','ю','я']; {множество особых символов }
    while not eof(f) do
        begin  while not eoln(f) do
                begin read(f,c);
                        if c in lr
                        then case c of
                                'ж': write(g,'sz');
                                'ч': write(g,'ch');
                                'ш': write(g,'sh');
                                'щ': write(g,'sch');
                                'ь','ъ':;
                                'ю': write(g,'ju');
                                'я': write(g,'ja');
                             end
                          else write (g, Tabcod[c])
                end;
               readln(f);  writeln(g)
        end
 end;
  begin InitTab; Assign(f, 'fcod.pas');  Assign(g, 'gcod.pas');
          Reset(f); Rewrite(g);
          writeln ('Начали обработку файлов');
          CodeFile(f,g);
          writeln ('Кончили обработку файлов');
          Close (f);  Close (g)
  end.
```

Понятие системы, ее структура и функция. Виды систем.

Под системой понимают совокупность взаимосвязанных и взаимодействующих объектов, подчиненных определенной единой цели.

Это понятие основывается на понятиях **множество** и **отношение,** но гораздо сложнее последних. Если мы рассмотрим набор деталей, из которых можно собрать компьютер, то пока они лежат на столе отдельно друг от друга, их можно рассматривать как конечное множество деталей. Если мы будем сравнивать эти детали между собой по какому-то признаку, например, по весу или стоимости, то это уже можно рассматривать

как определенное отношение на заданной предметной области. Если же из этих деталей собрать действующий компьютер, то это уже будет система. Каждая деталь, взаимодействуя с другими, внесёт свой вклад в работу системы. Система сама может являться частью другой более сложной системы.

Например, уже рассмотренный нами компьютер будет входить как часть в систему, представляющую собой рабочее место программиста. Кроме него, в рабочее место могут входить, например, принтер, модем, графопостроитель, звуковая карта с колонками и другие системы.

Система считается описанной, если определены структура системы и функция системы.

Структура и функционирование системы определяются поставленными перед ней целями. Целью для компьютера является автоматизация выполнения алгоритмов при помощи программ, а для рабочего места программиста – например, возможность произвести распечатку и пересылку результатов, полученных при программировании, или при наборе текста в некотором редакторе, или при работе с какой-то базой данных.

Структура системы – это множество элементов, из которых состоит система, и взаимосвязи (отношения) между ними.

Структура позволяет в наглядной форме показать, из каких элементов состоит система и как эти элементы связаны друг с другом. Для представления структуры системы используют различные схемы, подобно тому, как для представления структуры программы используют блок-схемы. Для описания систем на строгом математическом языке используется понятие графа. Основателем теории графов является Эйлер (1707 - 1782). Сейчас теория графов используется практически во всех областях науки и позволяет наглядно, ясно и вместе с тем строго описывать многие абстрактные структуры.

Определение: **графом** G называется пара множеств (V, E), где V – непустое конечное множество элементов, называемых вершинами графа, а E – конечное множество пар (упорядоченных или неупорядоченных) элементов из множества V, называемых ребрами или дугами графа.

В терминах теории графов элементы системы представляют собой вершины графа, а связи между ними – дуги. Для наглядного изображения системы посредством графа нужно точками на плоскости изобразить её элементы, а линиями, связывающими между собой некоторые из этих точек – связи между ними. При этом расположение точек на плоскости значения не имеет. Дуги также могут быть отрезками или частями каких-то кривых, главное, чтобы они однозначно и правильно отражали, какие элементы с какими взаимодействуют.

Функция системы представляет собой описание всех допустимых процессов, которые могут иметь место в данной системе. Функции описывают в математической форме в виде уравнений, систем уравнений или неравенств. Иногда такое описание из-за сложности процессов невозможно в полном объёме. В этом случае некоторыми деталями информации (частью параметров, задающих систему) приходится пренебрегать. Необходимо следить, чтобы отброшенный параметр не был существенным с точки зрения целей, ради которых система строится.

Любая система обладает важным свойством: целое, образованное из множества взаимосвязанных и взаимодействующих элементов, обладает теми качествами, которые не присущи отдельным частям этого целого. Поэтому систему нельзя рассматривать как простую сумму составляющих её элементов.

В каждый момент времени система находится в каком-то состоянии, называемом текущим. Текущее состояние системы определяется набором значений всех составляющих её параметров. Система называется **детерминированной,** если в любой момент времени можно однозначно определить её состояние в следующий момент времени. Можно представить себе такие системы, в которых независимо от того, насколько полно определены состояния и насколько точно заданы значения параметров, невозможно точно определить последующие состояния. Такие системы называются **недетерминированными.**

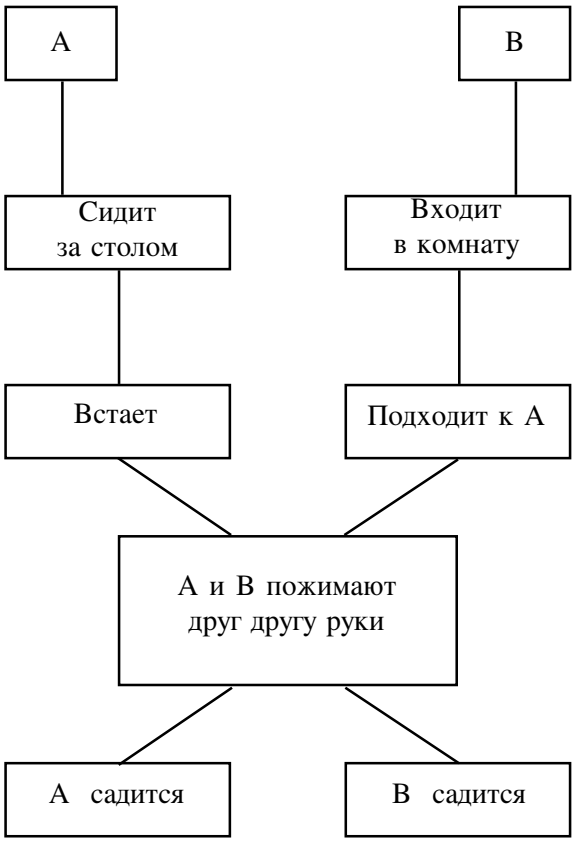
По способу взаимодействия между собой отдельных частей (компонент) системы делятся на несколько типов.

Если компоненты системы могут работать друг за другом строго поочередно, то говорят о **последовательной** системе. Если активность компонент может иметь место одновременно, то говорят о **параллельно** работающей системе. Если такие системы построены из отдельных, удаленных друг от друга в пространстве компонент, то говорят о **распределенных** системах.

Архитектура ЭВМ может быть описана как распределенная система. Другими примерами систем могут быть крупные производственные, транспортные и информационные системы, сложные электронные устройства, нервная система, солнечная система, отопительная система.

Автоматизированные системы представляют собой совокупность управляемого объекта и автоматических управляющих устройств, где часть функций управления выполняет человек (оператор). В ней автоматические устройства осуществляют сбор информации с объекта управления, ее передачу, преобразование и обработку, формирование управляющих команд и их выполнение на управляемом объекте, то есть те функции, которые легче всего поддаются формализации. Человек-оператор определяет цели управления и корректирует систему при изменении условий.

Рассмотрим пример простой распределенной системы:



Здесь самый существенный параметр - время, если его не учесть, то получится система, лишенная смысла.

Литература.

1. Ю.А.Шиханович. Введение в современную математику. М.: Наука, 1965.
2. Д.Кук, Д.Бейз. Компьютерная математика. М.: Наука, 1990.
3. Ян Стюарт. Концепции современной математики. Минск: Вышейшая школа, 1980.
4. Дж.Кемени, Дж.Снелл, Дж.Томпсон. Введение в конечную математику, Иностранная литература, 1963.
5. М.В.Дмитриева, А.А.Кубенский. Элементы современного программирования, Изд-во Санкт-Петербургского Университета,1991.

Дмитриева Марина Валерьевна, доцент кафедры информатики Санкт-Петербургского государственного университета.

Павлова Марианна Владимировна, старший преподаватель кафедры информатики СПбГУ, преподаватель школы-лицея N 419.

НАШИ АВТОРЫ