Вопросы к государственному экзамену
для направления 010400.62 Прикладная математика и информатика
в 2014-2015 учебном году.

Дисциплины специализации кафедры математического моделирования

1. Комбинаторные правила и структуры.
2. Дизъюнктивные нормальные формы. Минимизация ДНФ.
3. Пути и циклы в графах.
4. Устойчивость графов. Хроматические графы.
5. Продукционные базы знаний.
6. Базы знаний семантических сетей.
7. Логические программы.
8. Организация учета затрат на производство в программе «1С: Бухгалтерия 3.0».
9. Учет расчетов с персоналом по оплате труда в программе «1С: Бухгалтерия 3.0»: учет кадров, начисление и выплата заработной платы.
10. Источники вычислительных погрешностей. Понятие машинного эпсилон. Вычисление машинного эпсилон.
11. Машинное представление целых и вещественных чисел. Нормализованное представление вещественного числа. Выполнение арифметических операций с нормализованными вещественными числами, ошибки округления.
12. Моделирование распространения загрязнений. Постановка задач переноса и диффузии примесей.
13. Метод потоковых диаграмм Форрестера в моделировании сложных систем. Уравнения уровней и темпов.
14. Тестирование методами чёрного и белого ящика.
15. XML и документирование ПО в DocBook.
16. Прототипирование. Интерактивные прототипы.
17. Пролог. Управление выполнением программы.
18. Пролог. Рекурсия и отсечение.
19. Создание моделей бизнеса в стандартах IDEF.
20. Организационные структуры и бизнес-процессы.
21. UML. Диаграммы классов и последовательностей.
22. Объектная модель PHP.
23. Структуры данных JavaScript.
24. Основные понятия и функции ГИС.
25. Организация данных в ГИС. Координатные, векторные и растровые модели.

№1. КОМБИНАТОРНЫЕ ПРАВИЛА И СТРУКТУРЫ. наверх

1.Правило птичьих гнезд

Если имеются $n + 1$ птицы, которых необходимо разместить в n гнездах, то при любом способе размещения хотя бы в одном гнезде окажется не менее двух птиц.

Для обоснования справедливости приведенного правила воспользуемся методом рассуждений от противного. Предположим, что данное правило неверно. Пусть после распределения птиц в каждом гнезде оказалось не более чем по одной птице. Перенумеруем гнёзда, в которые попало по птице натуральными числами $1, \dots, k$. Поскольку в гнёзда попало всего k птиц и $k \le n$, то в гнёздах оказались размещены не все птицы. Из получаемого противоречия следует, что предположение о противном неверно, а, значит, верно правило птичьих гнёзд.

2. Правило умножения.

Пусть необходимо строить все возможные n-элементные последовательности $a_1, \dots, a_n$, *для которых выполнены условия:*

a) *первый элемент таких последовательностей может быть выбран m_1 способами*;

b) *если $i < n$, то для каждого способа выбора значений первых i элементов последовательности значение $i+1$-го элемента таких последовательностей может быть выбрано m_{i+1} способами.*

Тогда число различных последовательностей $a_1, \dots, a_n$ *равно:*

$m_1 m_2 \dots m_n$.

Обосновать справедливость правила умножения можно, например, математической индукцией по значению n.

Базис индукции

Для $n = 1$ правило умножения является справедливым, так как существует ровно m_1 одноэлементных последовательностей, первый элемент которых можно выбрать m_1 разными способами.

Индуктивное предположение

Пусть для $n = k$ количество различных последовательностей, удовлетворяющих условиям правила умножения, равно $m_1 m_2 \dots m_k$.

Индуктивный переход

Пусть $n = k + 1$. По индуктивному предположению существует ровно $m_1\ m_2\ \dots\ m_k$ различных последовательностей длины k, удовлетворяющих условиям правила умножения, каждая из которых при добавлении еще одного элемента преобразуется в m_{k+1} различных последовательностей длины $k+1$, также удовлетворяющих условиям этого правила. Поэтому общее число последовательностей длины $k+1$ в m_{k+1} раз больше числа различных последовательностей, имеющих длину k. То есть всего таких последовательностей ровно $m_1 m_2 \dots m_k m_{k+1}$.

3. Правило сложения

Пусть заданы непересекающиеся конечные множества $A_1, \dots, A_k$. Тогда мощность объединения этих множеств может быть определена по формуле:

$$\left|\bigcup_{i=1,\dots,k} Ai\right| = \sum_{i=1,\dots,k} |Ai|.$$

Для обоснования справедливости правила сложения заметим, что в значении левой части записи правила каждый элемент объединения непересекающихся множеств $A_1, \dots, A_k$ учтен ровно один раз. Значение в правой части правила учитывает все элементы каждого из множеств $A_1, \dots, A_k$. Поскольку последние множества непересекающиеся, то всякий элемент их объединения учитывается в правом значении также ровно один раз. Это означает справедливость правила сложения.

Правило умножения - основное для определения количества комбинаторных объектов. К нему сводятся различные вспомогательные комбинаторные соотношения и задачи, преобразуемые в семейства задач, решаемых с помощью этого правила.

Размещение:

m-размещением n-элементного множества называется всякая последовательность, состоящая из m-элементов этого множества.

Размещение, в котором все элементы разные называется размещением без повторений.

Размещение в котором допускается повторное вхождение элемента называется разм. с повторением.

m < = n – условие для размещения без повторений. A_n^m – без повторений , $\bar{A}_n^m$ – с повторениями.

A_n^m =n(n-1)…(n-m+1)=n!/(n-m)!

$\bar{A}_n^m$ = n*n…*n(m штук) = n^m.

Сочетание:

Сочетанием из n по m называется всякая совокупность состоящая из m-элементов некоторого n-элементного множества.
Сочетание называется сочетанием без повторений если все его элементы разные.
Сочетание в котором допускается повторное вхождение одинаковых элементов называется сочетанием с повторениями. C_n^m – без повторений, $\bar{C}_n^m$ – с повторениями.
А) C_n^m. Пусть D - множество из n элементов. Рассмотрим все размещения без повторений из n по m, составленные из элементов этого множества. Число таких размещений равно A_n^m. Компоненты каждого такого размещения определяют некоторое сочетание без повторений из n по m, составленное из элементов этого размещения. При этом размещения, отличающиеся лишь порядком вхождения значений, образуют одно и то же сочетание. Поскольку размещения, соответствующие одному и тому же сочетанию являются перестановками элементов этого сочетания, то всякое сочетание порождается m! различными размещениями. Поэтому C_n^m = A_n^m /m!.
Б) $\bar{C}_n^m$. Пусть D = {a1, ... , an} - некоторое множество. Сочетания с повторениями, содержащие по m элементов этого множества, будем представлять двоичными последовательностями длины n+m-1, составленными из m нулей и n-1 единиц. Двоичная последовательность, сопоставляемая отдельному сочетанию, состоит из n групп последовательно идущих нулей разделенных n-1 единицами. В i-й группе нулей, отсчитываемой слева направо, столько нулей, сколько экземпляров элемента ai входит в выбранное сочетание. Если некоторый элемент не входит в сочетание без повторений ни одного раза, то соответствующая ему группа нулей окажется пустой.

{Определенное выше соотношение между сочетаниями с повторениями из n по m и двоичными последовательностями длины n + m - 1, содержащими по m нулей, является биективным.}

Поэтому, число сочетаний с повторениями из n по m равно числу рассматриваемых двоичных последовательностей. Таких последовательностей ровно столько, сколько имеется различных способов выбора из n+m-1 позиций в двоичных последовательностях, таких m позиций, в которые записываются нули. Число способов выбора различных m позиций, если имеется n+m-1 разных позиций, равно

$$\bar{C}_n^m = C_{n+m-1}^m$$

№2. ДИЗЪЮНКТИВНЫЕ НОРМАЛЬНЫЕ ФОРМЫ (ДНФ). МИНИМИЗАЦИЯ ДНФ. наверх

Определение: Формула $k = x_{i_1}^{\ 1}$ ¿ $\ldots \wedge x_{i_r}^{\ r}$, где $1 \le i_1 < i_2 < \ldots < i_r \le n$, называется *элементарной конъюнкцией ранга r*.

Определение: *ДНФ* называется всякая формула вида D= $K_1 \vee K_2 \ldots . \vee K_r$, где K1…Kr – это произвольные элементарные конъюнкции.

Определение: *Сложностью ДНФ* D называется число вхождений в D функций & и ∨. Для обозначения сложности ДНФ используют выражение L(D).

Определение: ДНФ D, представляющая функцию f, называется *минимальной ДНФ* для этой функции, если L(D)=min(L(D*)), где минимум берётся по всем ДНФ D*, представляющим функцию f.

Теорема: Пусть f(x1…xn) ¿ P2 и 1<=m<=n.

Тогда справедливо следующее тождество:

$$f\left(x_1, \ldots, x_n\right) = \underset{(\sigma 1, \ldots, \sigma m)}{¿} x_1^{\sigma 1} \wedge \ldots \wedge x_m^{\sigma m} \wedge f\left(\sigma 1, \ldots, \sigma m, x_{m+1}, \ldots, x_n\right) \qquad (1)$$

Док-во:

Покажем, что для каждого набора $\left(\sigma_1, \ldots, \sigma_n\right)$ значения левой и правой частей совпадают.

Значение, принимаемое функцией слева, равно $f\left(\gamma_1, \ldots, \gamma_n\right)$.

Рассмотрим правую часть. Подставив в него выбранные значения переменных, получим запись:

$$\underset{(\sigma 1, \ldots, \sigma m)}{¿} \gamma_1^{\sigma 1} \wedge \ldots \wedge \gamma_m^{\sigma m} \wedge f\left(\sigma 1, \ldots, \sigma m, \gamma_{m+1}, \ldots, \gamma_n\right)$$

Учитывая, что $\gamma_1^{\sigma 1} \wedge \ldots \wedge \gamma_m^{\sigma m} = 1$ тогда и только тогда, когда $\forall i = 1, \ldots, m\left(\gamma_i = \sigma_i\right)$, из полученного выражения можно удалить все нулевые элементы дизъюнкции и получить выражение: $\gamma_1^{\gamma 1} \wedge \ldots \wedge \gamma_m^{\gamma m} \wedge f\left(\gamma 1, \ldots ., \gamma n\right)$. Так как $\gamma_1^{\gamma 1} \wedge \ldots \wedge \gamma_m^{\gamma m} = 1$, то последнее выражение равно: $f\left(\gamma 1, \ldots ., \gamma n\right)$. То есть значения, принимаемые левой и правой частями равенства (1) на наборе значений переменных $\left(\sigma_1, \ldots, \sigma_n\right)$, равны.

Ч.Т.Д.

Теорема: Для каждой б.ф. f, отличной от тождественного нуля, существует минимальная ДНФ.

Док-во:

1. Заметим что всякая f, отличная от тождественного нуля , представляется хотя бы одной ДНФ(например это может быть СДНФ).

2. Если задана б.ф. f(x1..xn), то нетрудно проверить, что существует ровно 3^(n) – 1 конъюнкций разных рангов, составленных только на основе переменных функции f.

Действительно, каждая из переменных f может либо не входить в элементарную конъюнкцию, либо входить в неё без отрицания, либо входить в такую конъюнкцию с отрицанием. Всего таких вариантов для n переменных ровно 3^(n). Из них невозможен случай, когда ни одна переменная не входит в элементарную конъюнкцию.
Из 3^(n) - 1 различных конъюнкций с точностью до порядка вхождения элеентарных конъюнкций можно составить ровно $2^{3^n - 1} - 1$ разных ДНФ.
Действительно, каждая ДНФ задаётся некоторым непустым подмножеством множества всех конъюнкций.

3. Последовательно просматривая все $2^{3^n - 1} - 1$ ДНФ, составленные из переменных функции f, можно найти такую из них, которая представляет f и имеет минимальную сложность среди всех ДНФ, представляющих f.

Ч.Т.Д.

№3. ПУТИ И ЦИКЛЫ В ГРАФАХ. наверх

Графом называется всякая пара G = (V,U), где V – мн-во вершин, а U – мн-во ребер.

Последовательность W = a1, ... , an - вершин графа G=(V,U) образует путь в G, если $\forall i=1,..,n\left(\left(a_i,a_{i+1}\right)\in U\right)$

Прохождение пути в графе - это последовательное перемещение по вершинам графа, по ребрам, соединяющим такие вершины. О таких ребрах говорят, что они принадлежат пути и что путь проходит через эти ребра.

Если W это путь в графе G, то запись E(W) используется для обозначения множества ребер, принадлежащих пути. Вершины a1 и an называются началом и концом пути. При этом говорят, что W ведет из a1 в an. Длиной пути W называется число ребер, проходимых при движении по W из a1 в an. Т.е., если W содержит m вершин, то длина W равна m – 1.

Путь W называется *элементарным*, если все вершины в нем разные.

Путь W называется *простым*, если все ребра из E(W) проходятся ровно по одному разу.

Путь W называется *циклом*, если его начало и конец совпадают.

Цикл называется элементарным (простым) циклом, если в нем все вершины за исключением первой и последней (все ребра) разные.

[..Если некоторый путь W в графе G не является элементарным, то он может быть преобразован в элементарный путь с теми же началом и концом. Для этого необходимо последовательно выбирать пары одинаковых вершин, одна из это внутренняя вершина пути. Для каждой такой пары вершин ai и aj из которых W удаляется часть, состоящая из вершин ai+1, ... , aj, т.е. путь W = a1, ... ,ai, ...,aj, ... ,an преобразуется в путь W1 = a1, ... , ai, aj+1 , ... , an. Для получения элементарного пути приведенное преобразование повторяется до тех пор, пока не будет получен требуемый элементарный путь. Данный процесс заканчивается за конечное число шагов, поскольку при всяком применении операции удаления части исходного пути длина нового пути оказывается меньше длины исходного пути …]

Граф G называется *связным графом*, если для любых двух вершин G существует путь, соединяющий эти вершины. Компонентами связности графа G называются максимальные связные подграфы этого графа.

Пусть G = (V, U) ¿ U это некоторый граф. Ребро u называется *циклическим ребром*, если в G имеется элементарный цикл ненулевой длины, проходящий через концы ребра u.

Неориентированный связный граф, не имеющий петель, называется *деревом*, если он не содержит циклических ребер.

Теорема. Неориентированный связный граф без петель G = (V, U) является деревом тогда и только тогда, когда |V| = |U| + 1.

Цикл в графе G называется циклом Эйлера, если он проходит через все ребра графа и каждое ребро проходится один раз.

Граф G называется четным графом, если степень каждой его вершины четная, т.е. в четном графе из каждой вершины в другие вершины графа ведет четное число ребер.

Теорема Эйлера. Граф G имеет цикл Эйлера тогда и только тогда, когда он является четным графом.

Цикл в графе называется циклом Гамильтона, если он содержит все вершины этого графа по одному разу.

Теорема. Если $\forall a_i,a_j\in V\left(d\left(a_i\right)+d\left(a_j\right)\geq n\right)$, то граф G имеет цикл Гамильтона.

№4. УСТОЙЧИВОСТЬ ГРАФОВ. ХРОМАТИЧЕСКИЕ ГРАФЫ. наверх

При изучении и использовании моделей, систем, основанных на графах, оказываются полезными множества вершин, обладающие специальными свойствами.

ОПРЕДЕЛЕНИЕ
Множество **E** вершин графа **G** = (**V**, **U**) называется <u>внутренне устойчивым</u>, если для любых двух вершин **a** и **b** из **E** справедливо соотношение (**a**, **b**) $\notin$ **U**.

ОПРЕДЕЛЕНИЕ
Множество I вершин графа G = (V, U) называется <u>внешне устойчивым</u>, если для любой вершины a $\in$ V \ I найдется такая вершина b $\in$ I, что (a, b) $\in$ U.

То есть всякое множество таких вершин графа, никакие две из которых не являются соседними, внутренне устойчивое.

Произвольное множество вершин графа считается внешне устойчивым, если всякая из остальных вершин графа является соседней хотя бы для одной вершины из этого множества.

В качестве примера рассмотрим граф, изображенный на рисунке:

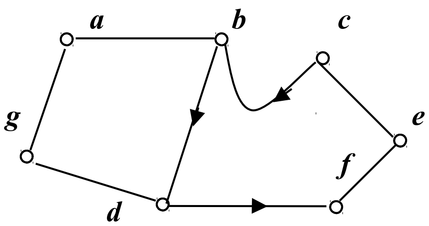


Рис. ***5.26***

В этом графе множества:

I *=* ***{a, d, c}*** *- внутренне устойчивое;*

E *=* ***{d, e, c, f, g}*** *- внешне устойчивое;*

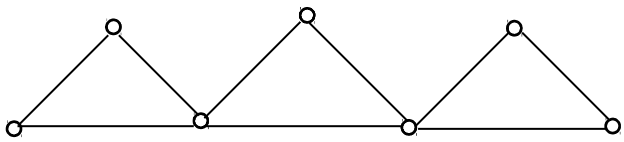
K *=* ***{g, b, e}*** *является одновременно как внутренне, так и внешне устойчивым.*

Нетрудно видеть, что удаление элементов внутренне устойчивого множества вершин графа приводит к внутренне устойчивому множеству вершин. Добавление новых вершин к внешне устойчивому множеству вершин графа приводит к внешне устойчивому множеству вершин.

Будем рассматривать неориентированные графы без петель. Граф G называется p-раскрашиваемым, если все его вершины можно раскрасить с использованием p цветов так, чтобы никакие две соседние вершины G не были окрашены одинаково.

Иначе говоря, <u>p-раскраска графа</u> G = (V, U) — это такое разбиение множества V на p непустых подмножеств $V_1, \dots, V_p$, что всякое ребро из U соединяет вершины из разных множеств разбиения.

Например, граф, изображенный на рисунке, является 3-раскрашиваемым и 4-раскрашиваемым, но не является 2-раскрашиваемым.



ОПРЕДЕЛЕНИЕ

<u>P – раскраской графа</u> G = (V, U) называется разбиение V на множества $V_1, \dots, V_p$, для которого выполнено условие:

$\forall (a, b) \in U \ (a \in V_i \ \& \ b \in V_j \rightarrow i \neq j)$.

ОПРЕДЕЛЕНИЕ

<u>Хроматическим числом графа</u> G называется такое минимальное число k, для которого существует k-раскраска этого графа.

Хроматическое число определено для всякого неориентированного графа G, не имеющего петель, и обозначается как $\chi(G)$.

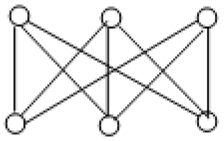
Построим описание множества графов, хроматическое число которых равно 2. Они называются <u>2-хроматическими графами.</u>

ТЕОРЕМА (Критерий 2-хроматичности графов).
Если граф G - неориентированный граф без петель, имеющий хотя бы одно ребро, то χ (G) = 2 тогда и только тогда, когда G не имеет элементарных циклов нечетной длины. (Док-во не приведено, большое).

Графы с хроматическим числом 2 называются *двудольными*.

По определению граф G = (V, U) является *двудольным*, если множество его вершин может быть разбито на два класса V_1 и V_2 так, что всякое ребро в U соединяет вершины из разных классов.

Примером двудольного графа является граф $K_{3,3}$.



Двудольными графами можно представлять словари соответствия слов в двух языках. Вершины таких графов соответствуют словам, а ребра соединяют пары слов разных языков, имеющих общий смысл.

№5. ПРОДУКЦИОННЫЕ БАЗЫ ЗНАНИЙ. наверх

Рассмотрим модель знаний, знания которой оперируют с одиночными фактами. Всякое высказывание называется «атом». Для обозначения атомов будем использовать символы латинского алфавита.

Основной способ представления знаний с помощью атомов являются правила или продукции вида $\frac{A1 \ldots An}{An+1}$,

где A1,…,An – конкретные атомы, An+1 – заключение. Создана продукция представления правил, означающая, что если истины атомы A1…An, то истинным является атом An+1.

Например, $\frac{усы\ лапы\ хвост\ полосы}{тигр}$.

Всякое правило устанавливает причинно-следственную зависимость между истинностью посылок A1,…,An и истинностью заключения An+1. Такая зависимость представляет утверждение: если атомы A1…An истинны, то истинно и высказывание, представленное атомом An+1. Продукции, в которых n=0 называются аксиомами, среди множества знаний, представленных атомарными продукциями, аксиомы представляют атомы, истинные безусловно и обычно являются начальными данными.

В атомарно-продукционной модели знания представляются продукциями. Атомарная продукция (A,P), где A – множество различных атомов, использованных в записях продукций, в том числе в представлении нач. данных. P- мн-во продукций, составленных с использованием мн-ва атомов A.

Постановка задач в атомарно-продукционных системах.

1. Частная задача.

A-? , где A некоторый конкретный атом. Означает вопрос проверки истинности атома A на основе заданной системы продукций.

2. Общая задача.

X-? , где X символ неизвестной. Решение этой задачи предполагает построение всех атомов, истинность которых может быть установлена на основе атомарных продукций системы.

Для решения задачи можно использовать механизм прямого вывода, со следующими изменениями: на каждом шаге просматривается список L2. В списке отыскивается первая по порядку продукция, все посылки которой являются следствиями элементов списка L1, заключение может быть явно вычислено, значение заключения не является следствием предикатов в L1.

№6 БАЗЫ ЗНАНИЙ СЕМАНТИЧЕСКИХ СЕТЕЙ. наверх

Семантическая сеть – иерархическая сеть представленная виде графа с нагруженными вершинами и рёбрами. Используется как подход представления знаний в БЗ.

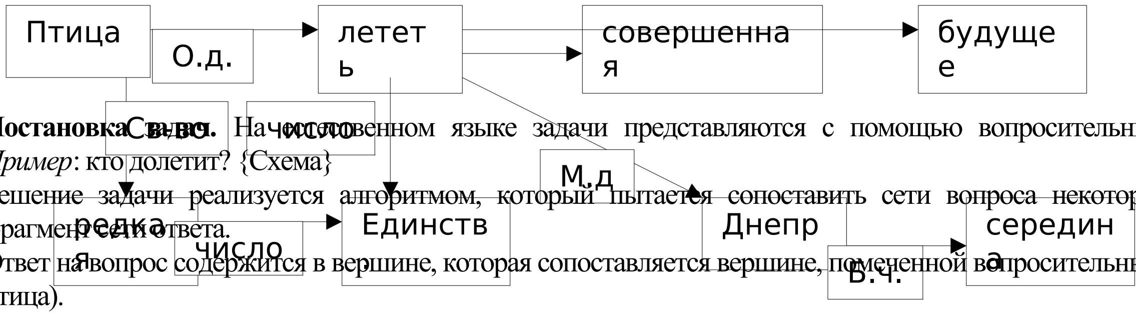
Иерархии в отношениях.
Отношение *«являться»* представляет принадлежность отдельных объектов или свойств некоторых классов более широким классам таких объектов или свойств. Например, танкер являться корабль.
Отношение *«быть частью»* устанавливает вхождение одних объектов или понятий в другие объекты в качестве их составных частей.

Элементарные задачи: А являться Б? А быть частью Б?
<u>В первой задаче</u> если А Б конкретные значения, то нужно проверить существование пути вверх от А к Б. Если А Б неизвестные, то необходимо перебрать различные варианты значений А и Б.
<u>Для второй задачи</u> если А Б конкретные значения, то для решения задачи достаточно проверить, что их А можно переместиться в Б сначала поднимаясь вверх по отношению быть частью, а затем вниз по отношению являться. Если А Б неизвестные, то задача может быть решена перебором разл. конкретных значений, которые могут принимать значения А, Б.

Семантические сети для предложений естественного языка

Формальное представление смысла заключённого в предложениях ЕЯ принято конструировать в виде ориентированных графов, к вершинам которых приписаны понятия из предложения или параметры языка, а связи между вершинами указывают на семантические зависимости между понятиями.

Пример: Редкая птица долетит до середины Днепра.



Постановка задачи. На естественном языке задачи представляются с помощью вопросительных предложений. *Пример*: кто долетит? {Схема}
Решение задачи реализуется алгоритмом, который пытается сопоставить сети вопроса некоторый близкий ему фрагмент сети ответа.
Ответ на вопрос содержится в вершине, которая сопоставляется вершине, помеченной вопросительным словом. (кто? – птица).

Этапы построения СС в предложении естественного языка:
1) лексический анализ (выделение отдельных слов или лексем)
2) морфологический анализ (определение нейтральной формы слов и их грамматическая характеристика
3) синтаксический анализ (определяется структура предложения, в котором выделяются главные, второстепенные члены предложения, главные и подчиненные предложения)
4) семантический анализ (все виды связи между парами слов) осуществляется на основе правил, учитывающих части речи, которые принимают слова, взаимное сорасположение в предложении, согласованность грамматических форм слов. После этого этапа СС одного предложения естественного языка может быть построена полностью.

№7. ЛОГИЧЕСКИЕ ПРОГРАММЫ. наверх

1. Константы.
К или индивиды – это конкретные объекты, которые не изменяются принимая единственное возможное значение. Для представления К можно использовать запись их значений, если в системе предусмотрено соответствие типа значений, или специальные символьные записи, являются именами К.

2. Переменная.
Это символы, которые обозначают элементы из некоторых множеств. При этом П может соответствовать любое значение из множеств. П записываются с помощью имен и начинаются с прописной буквы. Среди П выделяется специальная переменная, она называется анонимной и записывается как _ и используется в записях предикатов в составе логических программ для обозначения переменных таких предикатов, значение которых не существенно в рамках решения задачи.

3. Функциональные выражения или термы.
Составляются из К, символьных П, функциональных символов и разделителей по следующим правилам:
- всякая К есть терм.
- всякая П есть терм.
- если f – n-местный функциональный символ и t1,…,tn – термы, принимающие значения из множества значений 1,2,…,n функции f, то запись f(t1,…,tn) является термом. Такой терм принимает значение во множестве значений функции f.

4. Предикаты.
Представляют собой логические функции P(x1,…,xk), где x1,…,xk – П, принимающие значения на заданных множествах. В составе программы Пр используются в виде P(t1,…,tn), где t1,…,tn – термы, типы значений которых совпадают с типами значений x1,…,xk. Количество П Пр называется его размерностью или арностью.

5. Предложение.
Представляет собой основные структурные единицы логических программ. Они бывают 2 видов:
- факты
-следование
Содержание факта принимается как истинное утверждение. Поэтому в качестве теров в фактах стоят константы.
Предложение следования имеет вид: A<=B1,…,Bn. Здесь A,B1,…,Bn – предикаты.
Отдельные предложения в тексте программы, не связаны между собой, т.к. порядок следствия предложений не важен в логической программе.

6. Постановка задачи.
Задача в логическом программировании становится в виде предиката P(t1, …, tn), где t1, …, tn – термы.
Задача предполагает нахождения решения, в котором будут указаны значения переменных в постановке задачи, в которой предикат является следствием логической программы.

7. Механизм вывода.
МВ – способ организации решения задачи на основе логической программы.
Известными правилами вывода являются правило следования:

$$\frac{A, A \rightarrow B}{B}$$

и правило резолюций:

$$\frac{A, A \leftarrow B}{\bar{B}}.$$

Данная схема реализует метод рассуждения от противного и является основным правилом для решения задач с помощью логических программ.

№8. ОРГАНИЗАЦИЯ УЧЕТА ЗАТРАТ НА ПРОИЗВОДСТВО В ПРОГРАММЕ «1С: БУХГАЛТЕРИЯ 3.0». наверх

В прикладном решении «1С:Бухгалтерия 8 (ред.3.0)» доработан механизм учета затрат на производство. Теперь не обязательно отражать сумму выручку по номенклатурным группам, на которых были отражены затраты на производство. Это значительно упрощает процедуру ведения учета на предприятиях, которые в определенные месяцы по каким-либо причинам не имели выручки для распределения сумм образовавшихся затрат.

Теперь пользователь сам может выбрать один из вариантов учета затрат на производство, который выбирается в учетной политике по организации на закладке «Затраты на производство»

Из предложенного списка возможен выбор следующих вариантов:

1. **Выпуск продукции** – данный флаг необходимо установить тем предприятиям, которые выпускают продукцию и формируют себестоимость с помощью аккумуляции затрат на 20 счете. Данная галка включает полностью весь функционал «1С: Бухгалтерии 8», который был реализован и заложен в предыдущую редакцию данного прикладного решения.

2. **Выполнение работ, оказание услуг заказчикам** – данный флаг необходимо установить, если предполагается использование счета 20, как счета, на котором будут отражаться затраты по оказываемым услугам покупателям (заказчикам). При использовании данного варианта определения и списания затрат на производство добавляется возможность самостоятельно определить, каким образом будет происходить списание затрат:

    1. *Без учета выручки* – этот вариант как раз для тех организаций, которые не ведут учет затрат по номенклатурным группам и встречается такая ситуация, когда за определенные месяцы по определенным номенклатурным позициям отсутствует выручка. При установленном варианте программа спишет затраты с 20 счета вне зависимости от того, была ли на счете 90.01.1 сумма по соответствующей номенклатурной группе

    2. *С учетом выручки* – этот вариант дублирует возможности предыдущей редакции прикладного решения и выполнит списание затрат только при наличии выручки по счету 90.01.1 по соответствующей номенклатурной группе. Важно помнить, то выручка по счету 90.01.1 должна быть отражена документом «Реализация товаров и услуг». Если по итогам закрытия месяца необходимо отразить незавершенное производство на счете 20.01, то необходимо провести документ «Инвентаризация НЗП», в котором указать конкретные номенклатурные группы, которые не должны закрыться на счет себестоимости 90.02.

    3. *С учетом выручки только по производственным услугам* – данный вариант направлен на предприятия, которые оказывают услуги производственного характера и такого рода операции отражают документом «Акт оказания производственных услуг». При таком варианте будет учитываться только та сумма выручки, которая была проведена с помощью вышеуказанного документа. Если будет проведен документ «Реализация товаров и услуг», то данная выручка для расчета списания затрат будет проигнорирована.

№9. УЧЕТ РАСЧЕТОВ С ПЕРСОНАЛОМ ПО ОПЛАТЕ ТРУДА В ПРОГРАММЕ «1С: БУХГАЛТЕРИЯ 3.0»: УЧЕТ КАДРОВ, НАЧИСЛЕНИЕ И ВЫПЛАТА ЗАРАБОТНОЙ ПЛАТЫ. наверх

Кадровый учет. Рассматриваются справочники: «Физические лица» и «Сотрудники». В этих справочниках указываются сведения о работниках организации.

В справочнике «Сотрудники» фиксируются заключенные трудовые договора с работником. В этом справочнике работник может быть представлен неоднократно, в зависимости от количества заключенных трудовых договоров.

В справочнике «Физические лица» работник будет представлен в одном экземпляре. Сведения о начисленных страховых взносах и об удержанном исчисленном НДФЛ обобщаются все по одному физическому лицу. Когда будем предоставлять отчетность в Фонды или Налоговую, предоставим данные в разрезе одного физического лица.

Если у работника есть дети, то необходимо предоставить вычет. Вычеты предоставляются следующим образом: сначала начисляется зарплата. Затем отнимается сумма вычета и определяется налоговая база по НДФЛ, то есть с предоставлением вычета налоговая база уменьшается. Также здесь указывается статус налогоплательщика: резидент, нерезидент и дата, с которой установлен данный статус. В течение года данный статус может меняться в зависимости от того, какой статус приобрел работник. Но окончательный статус плательщика определяется по итогам года.

Работник является *резидентом*, если он находится на территории Российской Федерации свыше 185 календарных дней. Также из данной формы можно перейти к доходам с прежнего места работы. Можно указать доходы с предыдущего места работы. Эти сведения необходимы для того, чтобы предоставлять стандартный вычет на ребенка.

В карточке сотрудника по гиперссылке «Учет затрат» для каждого работника можно указать свой способ начисления зарплаты. Если на работника распространяется общий способ начисления зарплаты, то эти данные не указываются. Общий способ отражения зарплаты уже был указан в настройках учета зарплаты. По гиперссылке «Страхование» указывается статус застрахованного лица. Если есть справка об инвалидности, заполняются сведения об инвалидности. Эти сведения отразятся при начислении страховых взносов. Эта информация также будет отражаться в отчетности, которую будем предоставлять в органы. Рассмотрим процедуру начисление зарплаты.

Налоговая база представляет собой начисленную зарплату минус вычеты. Заработная плата может выплачиваться по кассе, может выплачиваться через Банк. При этом формируется соответствующая ведомость. Можно выдать «Расходный кассовый ордер» на каждого сотрудника, указанного в платежной ведомости. А можно сформировать один «Расходный кассовый ордер» для сотрудников, указанных в этой ведомости. Но в этом случае, необходимо, чтобы сотрудники расписались в платежной ведомости, что они денежные средства получили.

Перечисление заработной платы по ведомостям определился автоматически. Документ проведем и проанализируем сформированные им проводки. Происходит погашение кредиторской задолженности организации и происходит уменьшение денежных средств на расчетном счете. Если в организации нет зарплатного проекта, и Вы планируете перечислять денежные средства на карточки сотрудникам, то необходимо будет сформировать «Ведомость в банк» индивидуально для каждого работника. На основании ведомости нужно будет сформировать платежное поручение. В качестве получателя следует указать либо соответствующий банк и в назначении платежа указать лицевой счет сотрудника, либо указать самого сотрудника и номер карты. Факт списания денежных средств с расчетного счета вы отразите на основании платежных поручений. Но если использовать обмен с системой Клиент-Банк, то документы «Списание с расчетного счета» сформируется автоматически. Вам необходимо будет уточнить в них вид операции и прикрепить соответствующую платежную ведомость.

№10. ИСТОЧНИКИ ВЫЧИСЛИТЕЛЬНЫХ ПОГРЕШНОСТЕЙ. ПОНЯТИЕ МАШИННОГО ЭПСИЛОН. ВЫЧИСЛЕНИЕ МАШИННОГО ЭПСИЛОН. наверх

В большинстве компьютеров вещественные числа представляются в форме с плавающей точкой. В ПК можно представить лишь конечный набор вещественных чисел, а для всех остальных возможно лишь приближенное представление с некоторой ошибкой – ошибкой округления.

Машинное эпсилон $\tilde{E}$ - это такое минимальное положительное число, представленное в машине, что

$1 + \tilde{E} > 1$,

где «+» - машинное сложение.
<u>Например</u>, $1 + 10^{-9} = 1$.

Альтернативная формулировка: **наибольшее положительное число Ẽ, такое что, $1 \oplus \varepsilon_m = 1$, где $\oplus$ - машинное сложение.**
1)При представлении в ПК допускаются погрешности (нет π, e, $\sqrt{2}$ ..).
При переводе десятичной дроби 0,1 в двоичную систему получается бесконечная периодическая дробь 0,0001001100….
2) $0 < x0 \le |X| \le X_\infty$ - все числа, меньше 0 будут заменены на 0, а больше X_∞ - на X_∞
3) Арифметические операции над числами с плавающей точкой не могут быть реализованы точно, то есть идет округление.

Погрешности арифметических операций

При вычислениях с плавающей точкой операция округления может потребоваться после выполнения любой из арифметических операций. Так умножение или деление двух чисел сводится к умножению или делению мантисс. Так как в общем случае количество разрядов мантисс произведений и частных больше допустимой разрядности мантиссы, то требуется округление мантиссы результатов. При сложении или вычитании чисел с плавающей точкой операнды должны быть предварительно приведены к одному порядку, что осуществляется сдвигом вправо мантиссы числа, имеющего меньший порядок, и увеличением в соответствующее число раз порядка этого числа. Сдвиг мантиссы вправо может привести к потере младших разрядов мантиссы, т.е. появляется погрешность округления.
Для машинной арифметики не выполнены законы ассоциативности и коммутативности:

1 + 10E-8 + … + 10E-8 = 1
10E-8 + 10E-8 + …+ 1 = 1,00…001 - разница очевидна.
Еще:
10E25*10E20*10E-18*10E-22 – будет переполнение
10E-18*10E-22*10E20*10E25 = 0 – а тут обнуление

Пример вычисления машинного эпсилона на языке Си.

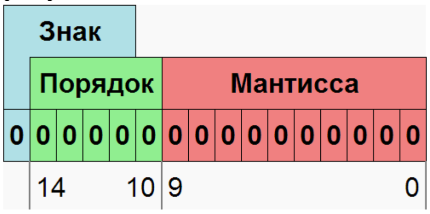
```
float macheps(void)
{
        float e = 1.0f;

        while (1.0f + e / 2.0f > 1.0f)
                e /= 2.0f;
        return e;
}
```

№11. МАШИННОЕ ПРЕДСТАВЛЕНИЕ ЦЕЛЫХ И ВЕЩЕСТВЕННЫХ ЧИСЛЕ. НОРМАЛИЗОВАННОЕ ПРЕДСТАВЛЕНИЕ ВЕЩЕСТВЕННОГО ЧИСЛА. ВЫПОЛНЕНИЕ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ С НОРМАЛИЗОВАННЫМИ ВЕЩЕСТВЕННЫМИ ЧИСЛАМИ, ОШИБКИ ОКРУГЛЕНИЯ. наверх

Вещественные числа обычно представляются в виде чисел с плавающей запятой. *Числа с плавающей запятой* — один из возможных способов представления действительных чисел, который является компромиссом между точностью и диапазоном принимаемых значений, его можно считать аналогом экспоненциальной записи чисел, но только в памяти компьютера.

Число с плавающей запятой состоит из набора отдельных двоичных разрядов, условно разделенных на так называемые *знак* (англ. sign), *порядок* (англ. exponent) и *мантиссу* (англ. mantis). В наиболее распространённом формате (стандарт IEEE 754) число с плавающей запятой представляется в виде набора битов, часть из которых кодирует собой мантиссу числа, другая часть — показатель степени, и ещё один бит используется для указания знака числа (0 — если число положительное, 1 — если число отрицательное). При этом порядок записывается как целое число в коде со сдвигом, а мантисса — в нормализованном виде, своей дробной частью в двоичной системе счисления. Вот пример такого числа из 16 двоичных разрядов:



Знак — один бит, указывающий знак всего числа с плавающей точкой. *Порядок* и *мантисса* — целые числа, которые вместе со знаком дают представление числа с плавающей запятой в следующем виде:

$(-1)^S \times M \times B^E$, где S — знак, B — основание, E — порядок, а M — мантисса. Десятичное число, записываемое как ReE, где R — число в полуинтервале $[1;10)$, E — степень, в которой стоит множитель 10; в нормализированной форме модуль R будет являться мантиссой, а E — порядком, а S будет равно 1 тогда и только тогда, когда R принимает отрицательное значение. Например, в числе $-2435e9$. При этом лишь некоторые из вещественных чисел могут быть представлены в памяти компьютера точным значением, в то время как остальные числа представляются приближёнными значениями.

Нормализованная форма записи числа. В ней мантисса десятичного числа принимает значения от 1 (включительно) до 10 (не включительно), а мантисса двоичного числа принимает значения от 1 (включительно) до 2 (не включительно). То есть в мантиссе слева от запятой до применения порядка находится ровно один знак. В такой форме любое число (кроме 0) записывается единственным образом. Ноль же представить таким образом невозможно, поэтому стандарт предусматривает специальную последовательность битов для задания числа 0 и ∞.

Действия с числами с плавающей запятой

Умножение и деление

Самыми простыми для восприятия арифметическими операциями над числами с плавающей запятой являются умножение и деление. Для того, чтобы умножить два вещественных числа в нормализованной форме необходимо перемножить их мантиссы, сложить порядки, округлить и нормализовать полученное число.

Соответственно, чтобы произвести деление нужно разделить мантиссу делимого на мантиссу делителя и вычесть из порядка делимого порядок делителя. Затем точно так же округлить мантиссу результата и привести его к нормализованной форме.

Сложение и вычитание

Идея метода сложения и вычитания чисел с плавающей точкой заключается в приведении их к одному порядку. Сначала выбирается оптимальный порядок, затем мантиссы обоих чисел представляются в соответствии с новым порядком, затем над ними производится сложение/вычитание, мантисса результата округляется и, если нужно, результат приводится к нормализированной форме. Пример:

Ошибки округления

Вещественные числа, разрядность мантиссы которых превышает число разрядов, выделенных под мантиссу в ячейке памяти, представляются в компьютере приближенно (с "обрезанной" мантиссой). Например, рациональное десятичное число 0,1 в компьютере будет представлено приближенно (округленно), поскольку в двоичной системе счисления его мантисса имеет бесконечное число цифр. Следствием такой приближенности является погрешность машинных вычислений с вещественными числами.

Вычисления с вещественными числами компьютер выполняет приближенно. Погрешность таких вычислений называют погрешностью машинных округлений.

№12. МОДЕЛИРОВАНИЕ РАСПРОСТРАНЕНИЯ ЗАГРЯЗНЕНИЙ. ПОСТАНОВКА ЗАДАЧ ПЕРЕНОСА И ДИФФУЗИИ ПРИМЕСЕЙ. наверх

Рассмотрим сначала <u>уравнения, описывающие движение самой среды распространения загрязняющей примеси.</u> Введем в рассмотрение *вектор скорости движения воздушных масс*
$\mathbf{a}=\left(u(x,y,z,t),v(x,y,z,t),w(x,y,z,t)\right)$ в точке с координатами (x, y, z) в момент времени t.
Давление и плотность p(x,y,z,t), ρ(x,y,z,t) также являются величинами, характеризующими движение жидкости; F(x,y,z,t)=(Fx(x,y,z,t),Fy(x,y,z,t),Fz(x,y,z,t)) – плотность внешних сил (если они имеются), рассчитанная на единицу массы.
В проекциях на оси прямоугольной декартовой системы координат *уравнения, описывающие движение среды*, имеют вид

$$\frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}+w\frac{\partial v}{\partial z}=F_y-\frac{1}{\rho}\frac{\partial P}{\partial y}+\nu_0\Delta v;$$

$$\frac{\partial w}{\partial t}+u\frac{\partial w}{\partial x}+v\frac{\partial w}{\partial y}+w\frac{\partial w}{\partial z}=F_z-\frac{1}{\rho}\frac{\partial P}{\partial z}+\nu_0\Delta w. \qquad (1)$$

Эти уравнения носят название **уравнений Навье–Стокса**. Здесь ν0=μ0⁄ρ – кинематический, μ0 – динамический коэффициент вязкости.
Чтобы замкнуть эту систему уравнений, где в общем случае неизвестными являются u, v, w, P, ρ, к ним необходимо добавить уравнение неразрывности

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}=0$$

или $div(\mathbf{a})=0$.

С количественной стороны изменение содержания примеси в среде во времени и пространстве описывается уравнение переноса. Пусть φ(x,y,z,t) – интенсивность (концентрация) загрязняющей примеси, мигрирующей вместе с потоком воздуха в атмосфере; $-\infty<(x,y,z)<\infty\ ,\ 0<z<H$.
Если общее количество примеси не изменяется в процессе переноса

$$dt$$

$$\frac{\partial\varphi}{\partial t}+\frac{\partial\varphi}{\partial x}\frac{dx}{dt}+\frac{\partial\varphi}{\partial y}\frac{dy}{dt}+\frac{\partial\varphi}{\partial z}\frac{dz}{dt}=0.$$

Или иначе

$$\frac{\partial\varphi}{\partial t}+\frac{\partial\varphi}{\partial x}u+\frac{\partial\varphi}{\partial y}v+\frac{\partial\varphi}{\partial z}w=0.$$

С учетом выполнения уравнения неразрывности это уравнение записывают также

$$\frac{\partial\varphi}{\partial t}+\frac{\partial(u\varphi)}{\partial x}+\frac{\partial(v\varphi)}{\partial y}+\frac{\partial(w\varphi)}{\partial z}=0.$$

Чтобы описать факт поглощения аэрозольной субстанции, связанного с его распадом уравнение переноса нагружают еще одним членом. (Если при распространении часть примеси входит в реакцию с внешней средой или распадается, этот факт можно интерпретировать, как поглощение субстанции). В этом случае уравнение запишется

$$\frac{\partial\varphi}{\partial t}+\frac{\partial(u\varphi)}{\partial x}+\frac{\partial(v\varphi)}{\partial y}+\frac{\partial(w\varphi)}{\partial z}+\sigma\varphi=0.$$

$$\frac{\partial\varphi}{\partial t}+\sigma\varphi=0.$$

Решением этого уравнения будет $\varphi=\varphi_0 e^{-\sigma t}$, где φ_0 определяется из начального заданного условия $\varphi\big|_{t=0}=\varphi_0(x,y,z)$

Отсюда следует, что σ есть величина обратная интервалу времени, за который интенсивность примеси уменьшается по сравнению с начальной в е раз.

Если в рассматриваемой области есть источник загрязняющего вещества, описываемый функцией f(x,y,z,t), уравнение переноса примет вид

$$\frac{\partial\varphi}{\partial t}+\frac{\partial\varphi}{\partial x}u+\frac{\partial\varphi}{\partial y}v+\frac{\partial\varphi}{\partial z}w+\sigma\varphi=f(t,x,y,z)$$

Такая простейшая модель, однако, не описывает ряд особенностей переноса примеси от источника, описываемого функцией f(x,y,z,t). На самом деле примесь как бы расплывается в атмосфере, образуя довольно сложное распределение аэрозолей в значительной окрестности от выброса, поскольку даже в безветренную погоду атмосфера является турбулентной средой. Чтобы учесть процесс диффузии загрязняющей примеси, в уравнение переноса вносят дополнительные члены

$$\frac{\partial\varphi}{\partial t}+\frac{\partial(u\varphi)}{\partial x}+\frac{\partial(v\varphi)}{\partial y}+\frac{\partial(w\varphi)}{\partial z}+\sigma\varphi-\mu\left(\frac{\partial^2\varphi}{\partial x^2}+\frac{\partial^2\varphi}{\partial y^2}\right)-\nu\frac{\partial^2\varphi}{\partial z^2}=f(t,x,y,z) \quad (2)$$

Вошедшие сюда множители μ, ν, носят названия коэффициентов горизонтальной и вертикальной диффузии соответственно. (При μ=ν диффузия происходит одинаково во всех направлениях).
На практике обычно не рассматривается полная постановка задачи, когда скорости движения среды определяются из уравнений (1). На небольших временных интервалах компоненты вектора скорости u, v, w можно считать постоянными, тогда уравнение переноса примет вид

$$\frac{\partial\varphi}{\partial t}+u\frac{\partial\varphi}{\partial x}+v\frac{\partial\varphi}{\partial y}+w\frac{\partial\varphi}{\partial z}+\sigma\varphi-\mu\left(\frac{\partial^2\varphi}{\partial x^2}+\frac{\partial^2\varphi}{\partial y^2}\right)-\nu\frac{\partial^2\varphi}{\partial z^2}=f(t,x,y,z) \quad (3)$$

Отметим, что горизонтальные составляющие скорости перемещения примеси u, v совпадают с составляющими скорости ветра. Что касается вертикальной составляющей, то для газообразных примесей, а также мелких (легких) жидких и твердых примесей, радиус частиц которых меньше 1 мкм, она практически равна скорости движения воздуха. Однако, в случае крупных (тяжелых) примесей (радиус частиц больше 1 мкм) под составляющей w в уравнении (3) следует понимать алгебраическую сумму вертикальной скорости движения воздуха и средневзвешенной (по массе) скорости падения частиц примеси под влиянием силы тяжести

$$\frac{\partial\varphi}{\partial t}+u\frac{\partial\varphi}{\partial x}+v\frac{\partial\varphi}{\partial y}+(w-w_g)\frac{\partial\varphi}{\partial z}+\sigma\varphi-\mu\left(\frac{\partial^2\varphi}{\partial x^2}+\frac{\partial^2\varphi}{\partial y^2}\right)-\nu\frac{\partial^2\varphi}{\partial z^2}=f(t,x,y,z)$$

При этом под w_g понимается абсолютная величина скорости падания частиц, которая всегда направлена по вертикали вниз. Вертикальная же скорость положительна при восходящем движении воздуха и отрицательна при нисходящем.

Постановка задач для уравнения переноса и диффузии.

В качестве области распространения примеси можно рассматривать как ограниченную (усеченный цилиндр), так и полуограниченную. В данном случае в качестве таковой рассматривается слой (атмосферы или водного пространства), в котором все коэффициенты уравнения считаются постоянными, либо в общем случае – пакет слоев.
Толщина слоя атмосферы, в котором происходит рассеяние технологической примеси, может быть значительной, особенно в случае мощных источников с высотой труб 200–300 м. В таком слое изменения скорости ветра и составляющих коэффициента турбулентного обмена имеет сложный характер.
Распространение примесей в атмосфере существенным образом определяется ее ветровыми и турбулентными характеристиками, поэтому можно рассматривать стратифицированный по высоте слой атмосферы, который можно условно разбивается на несколько подслоев, в каждом из которых компоненты вектора скорости и коэффициенты турбулентного обмена считаются постоянными. Такой подход позволяет использовать в каждом слое уравнение с постоянными коэффициентами. Как правило, рассматриваются плоскопараллельные границы слоев, плоскость XOY при этом обычно совмещается с поверхностью земли.

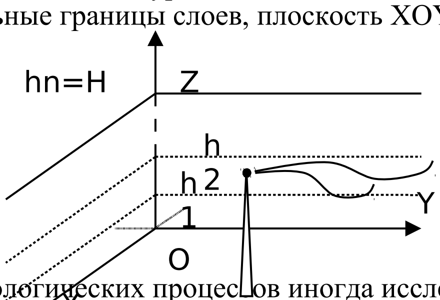


Рис.1.

На фоне метеорологических процессов иногда исследуется стационарный процесс переноса загрязняющих примесей, поступающих в атмосферу от источника постоянной мощности, тогда для слоистой атмосферы перенос однокомпонентной загрязняющей примеси будет описываться системой дифференциальных уравнений

$$u_n \frac{\partial\varphi_n}{\partial x} + v_n \frac{\partial\varphi_n}{\partial y} + (w_n - w_g)\frac{\partial\varphi_n}{\partial z} - \nu_n \frac{\partial^2\varphi_n}{\partial z^2} - \mu_n\left(\frac{\partial^2\varphi_n}{\partial z^2} + \frac{\partial^2\varphi_n}{\partial y^2}\right) + \sigma\varphi_n = f(x,y,z),$$

$$n = 1, \ldots, N.$$

Для однородной атмосферы перенос **и диффузия однокомпонентной загрязняющей примеси** будет описываться уравнением

$$u\frac{\partial\varphi}{\partial x} + v\frac{\partial\varphi}{\partial y} + (w - w_g)\frac{\partial\varphi}{\partial z} - \nu\frac{\partial^2\varphi}{\partial z^2} - \mu\left(\frac{\partial^2\varphi}{\partial z^2} + \frac{\partial^2\varphi}{\partial y^2}\right) + \sigma\varphi = f(x,y,z),$$

где (u,v,w) – известный вектор скорости воздушного потока; w_g – величина скорости гравитационного оседания (для легких примесей w_g=0); σ – коэффициент разложения загрязняющего вещества; μ – коэффициент горизонтальной диффузии; ν –коэффициент вертикальной диффузии; f(x,y,z)– функция, описывающая точечный источник загрязняющей примеси.
Граничные условия задаются в предположении, что примесь взаимодействует с поверхностью земли. Учитывается также возможность существования поверхностных источников выброса.
При постановке задач переноса и диффузии обычно используются следующие граничные условия:
На верхней границе может быть задана концентрация загрязняющего вещества (ЗВ)

$$\varphi(x,y,z)\Big|_{z=H} = f_1(x,y),$$

в случае, если рассматривается значительный по высоте слой атмосферы, концентрация примеси на верхней границе может быть принята равной нулю

$$\varphi(x,y,z)\Big|_{z=H} = 0.$$

В предположении отсутствия вертикальных воздушных потоков на больших высотах, может быть поставлено условие

$$\frac{\partial\varphi(x,y,z)}{\partial z}\Big|_{z=H} = 0.$$

На нижней границе в общем случае могут быть заданы условия вида

$$\left(\gamma\frac{\partial\varphi}{\partial z} - \lambda_i\varphi\right)\Big|_{z=0} = g_i(x,y) \qquad x,y \in \Omega_i \quad , \quad i = 1..N,$$

где вся подстилающая поверхность $\Omega : -\infty < x,y < \infty$, неоднородная по своим свойствам, представлена в виде объединения N областей $\Omega = \bigcup_{i=1}^{N}\Omega_i$, имеющих различные возможности осаждения и отражения примесей. Здесь g_i – мощность некоторого площадного источника (в общем случае зависит от координат (x,y); λ_i - имеет размерность скорости и описывает взаимодействие примеси с подстилающей поверхностью. Нетрудно видеть, что в случае $g_i(x,y)=0$ значения $0 < \lambda_i < \infty$ – промежуточные ситуации частичного отражения и частичного поглощения примеси, в то время как граничные значения соответствуют полному отражению и полному поглощению.
Если рассматривается сосредоточенный источник, расположенного в точке с координатами (x_0, y_0, z_0) и моделируемого δ-функцией Дирака, то рассматриваемая модель распространения загрязняющих примесей в пограничном слое атмосферы $0 < z < H$ основана на полуэмпирическом уравнении турбулентной диффузии для однокомпонентной примеси.
Для установившегося во времени процесса, перенос однокомпонентной загрязняющей примеси будет описываться системой дифференциальных уравнений

$$u_n\frac{\partial\varphi_n}{\partial x} + v_n\frac{\partial\varphi_n}{\partial y} + (w_n - w_g)\frac{\partial\varphi_n}{\partial z} - \nu_n\frac{\partial^2\varphi_n}{\partial z^2} - \mu_n\left(\frac{\partial^2\varphi_n}{\partial z^2} + \frac{\partial^2\varphi_n}{\partial y^2}\right) + \sigma\varphi_n =$$

$$= \delta_n M\delta(x - x_0, y - y_0, z - z_0), \quad n = 1,\ldots,N. \qquad (4)$$

где $\delta_n = \begin{cases} 0, n \neq i, \\ 1, n = i. \end{cases}$ i – номер слоя, содержащий источник.

Здесь: $\varphi_n(x,y,z)$ – функция концентрации загрязняющего вещества в n- том слое; un, vn, wn – компоненты вектора скорости в направлениях x, y, z для n-того слоя; μn, νn – коэффициенты диффузии в

горизонтальном и вертикальном направлениях для n-того слоя; M – постоянная, характеризующая мощность источника; n – номер слоя.
В декартовой системе координат, в которой плоскость XOY совмещена с дневной поверхностью Земли, а ось OZ направлена вертикально вверх, рассматривается следующая графическая постановка задачи

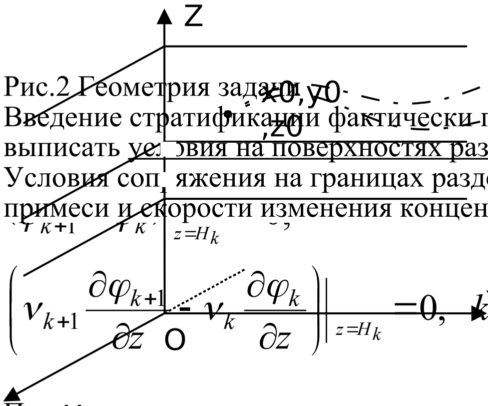


Рис.2 Геометрия задачи
Введение стратификации фактически предполагает введение поверхности разрыва. Поэтому необходимо выписать условия на поверхностях разрыва, которые далее будем называть условиями сопряжения.
Условия сопряжения на границах раздела подслоев выражаются в равенстве значений концентрации примеси и скорости изменения концентрации на границах соседних слоев.

$$\left.\left(\varphi_{k+1} - \varphi_k\right)\right|_{z=H_k} = 0,$$

$$\left.\left(\nu_{k+1}\frac{\partial\varphi_{k+1}}{\partial z} - \nu_k\frac{\partial\varphi_k}{\partial z}\right)\right|_{z=H_k} = 0, \quad k = 1,\dots,N-1. \qquad (5)$$

Постановка задачи является достаточно общей. Варьируя различные физические данные, можно получить конкретные задачи для случаев оседания тяжелой примеси от точечного источника, распространения примеси от площадного источника, некоторые виды задач для легкой примеси.
Рассматриваемая задача описывает частный процесс переноса загрязнителей с неизменными по времени входными данными, однако набор частных решений, соответствующий различным стационарным характеристикам, может быть использован при описании более сложных физических ситуаций.

№13. МЕТОД ПОТОКОВЫХ ДИАГРАММ ФОРРЕСТЕРА В МОДЕЛИРОВАНИИ СЛОЖНЫХ СИСТЕМ. УРАВНЕНИЯ УРОВНЕЙ И ТЕМПОВ. наверх

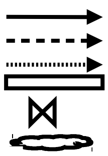
Модель Форрестера не является жестко привязанной к протекающим процессам, она осредняет и сглаживает дискретные явления, сопровождающие реальные процессы
В качестве примера моделируемой системы рассмотрим производственно-сбытовой комплекс. Представим его в виде совокупности *резервуаров*, каждый из которых имеет некоторое наполнение (товарами, оборудованием, заказами, людскими ресурсами и т.д.) Каждый резервуар имеет свой *уровень наполнения* (количественная характеристика – число штук, объем и т.д.). Резервуары соединены друг с другом *потоками*, реализующими перетекание наполнения из одного резервуара в другой. Если резервуары характеризуются некоторым уровнем (количеством), то потоки – *темпом* (изменением количества во времени), т.е.
Резервуар – уровень – количество;
Поток – темп – количество/время.
Наряду с потоками товаров, рабочей силы, оборудования и т.п. в модели обязательно вводятся *информационные потоки*. Потоки не пересекаются между собой, исключение составляют лишь потоки информации. Информационные потоки в отличие от остальных выходят из резервуаров, а входят как в резервуары, так и в потоки (выходя из уровней информационные потоки не изменяют их значения).
Весь процесс функционирования системы рассматривается, как набор *«снимков»* в моменты времени, отстоящие друг от друга на некоторый временной интервал ***Dt***. При этом все протекающие нелинейные процессы должны быть адекватно аппроксимированы ломаными. На малых временных интервалах функцию, описывающую процесс, можно аппроксимировать линейно

$$f(t+Dt)=f(t)+f'(t)\,Dt.$$

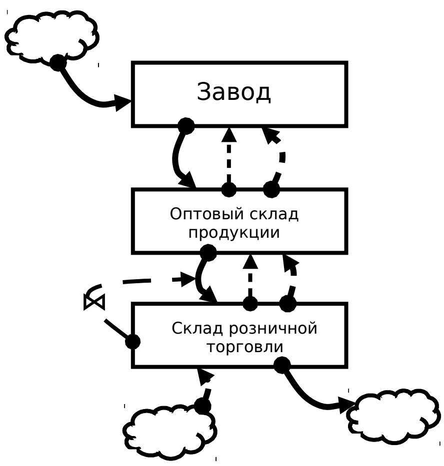
Или в интегральном виде

$$f(t+Dt)-f(t)=\int_0 f'(t+\xi)\,d\xi.$$

Промежуток времени *Dt* выбирается таким образом, чтобы из поля зрения не ускользали существенные моменты функционирования системы. Временной интервал, очевидно, должен позволять улавливать изменения всех уровней.
Введем графические обозначения для интерпретации модели

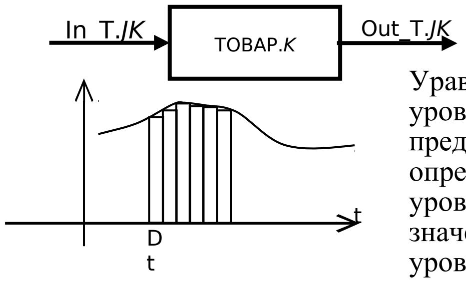


товарные потоки
информационные потоки
потоки заказов
резервуары
вентили
источники и стоки



В моделях Форрестера все системы являются замкнутыми, однако модель может быть разбита на блоки, тогда в случае рассмотрения отдельных блоков, внешние блоки будем интерпретировать как некие источники и стоки, например (см. рис.).
Вентили на потоковых диаграммах характеризуют управленческие решения. Таким образом, информационная система регулирует значения темпов.
Для описания процесса рассматриваются три момента времени (три точки временного ряда): *J*, *K* (*K=J+Dt*), *L* (*L=K+ Dt*).

Каждому уровню присваивается имя, например, ТОВАР, уровень в момент времени J обозначается ТОВАР.J. Каждый темп также имеет имя, например, Т и на временном интервале от J до K обозначается T.JK.



Уравнения модели Форрестера составляются таким образом, что каждый уровень в последующий момент времени может быть определен через предыдущие значения уровней и темпов. Каждый темп, в свою очередь, определяется с помощью предыдущих темпов и уровней. Т.е., значения уровней в момент *J* позволяют определить темпы между *J* и *K*. Затем по значениям темпов между моментами *J* и *K* и уровней в момент *J* определяются уровни в момент *K* и так далее. Таким образом, системы уравнений модели позволяют последовательно определять значения уровней и темпов в различные моменты времени.
Чтобы различить уровни и темпы, следует мысленно остановить функционирующую систему, в этой ситуации темпы исчезают, а уровни остаются.
Уравнение уровня ТОВ в момент *K* запишется в виде
$ТОВ.K=ТОВ.J+Dt(In\_T.JK–Out\_T.JK)$.

Все экономические производства обладают свойством запаздывания. Форрестер приблизил модели к жизненным реалиям, введя в них соответствующий элемент. Время ***запаздывания*** (задержки) обычно обозначаются *Del*. В связи с тем, что запаздывания являются неотъемлемой частью производства, они являются одной из составляющих, регулирующих вентили потоков (управленческие решения) и располагаются на темпах. Форрестером введено также понятие порядка запаздывания, позволяющее моделировать различные ситуации.
Оригинальность идеи Форрестера состоит в том, что в потоковых диаграммах он моделировал запаздывания теми же элементами, что и объекты деятельности системы. В результате модель не требовала введения новых подмоделей или дополнительных параметров.
Как было сказано выше, уравнения уровней составляются просто, если известны темпы в предшествующие моменты времени. Правильное же описание темпов является сложной задачей, так как темпы зависят от принимаемых решений. Простейшее уравнение темпа получается, когда встает вопрос о времени, необходимом для опустошения некоторого уровня, например, ТОВ.*K*

$$Out\_.KL=\frac{ТОВ.K}{Del}.$$

Тогда, принимая во внимание приведенное выше уравнение для уровня, можно записать

$$Out\_.KL=\frac{ТОВ.J+Dt(In\_.JK-Out\_.JK)}{Del}=\frac{Out\_.JK}{Del}\cdot Del+\frac{Dt(In\_.JK-Out\_.JK)}{Del}=$$

$$=Out\_.JK+\frac{Dt(In\_.JK-Out\_.JK)}{Del}.$$

№14. ТЕСТИРОВАНИЕ МЕТОДАМИ ЧЁРНОГО И БЕЛОГО ЯЩИКА. наверх

МЕТОДЫ СТРАТЕГИИ БЕЛОГО ЯЩИКА.

Стратегия белого ящика (структурный подход) заключается в проверке каждого пути, каждой ветви алгоритма. При этом внешняя спецификация во внимание не принимается.
Тестирование по принципу белого ящика характеризуется степенью, какой тесты выполняют или покрывают логику (исходный текст программы).

Метод покрытия операторов
Целью этого метода тестирования является выполнение каждого оператора программы хотя бы один раз.
Метод покрытия решений (покрытия переходов)
Согласно данному методу каждое направление перехода должно быть реализовано по крайней мере один раз.
Покрытие решений обычно удовлетворяет критерию покрытия операторов. Поскольку каждый оператор лежит на некотором пути, исходящем либо из оператора перехода, либо из точки входа программы, при выполнении каждого направления перехода каждый оператор должен быть выполнен.
Метод покрытия условий
Лучшие результаты по сравнению с предыдущими методами может дать метод покрытия условий. В этом случае записывается число тестов, достаточное для того, чтобы все возможные результаты каждого условия в решении выполнялись по крайней мере один раз.
Критерий решений (условий)
Критерий покрытия решений/условий требует такого достаточного набора тестов, чтобы все возможные результаты каждого условия в решении выполнялись по крайней мере один раз, все результаты каждого решения выполнялись по крайней мере один раз и, кроме того, каждой точке входа передавалось управление по крайней мере один раз.
Недостатком критерия является невозможность его применения для выполнения всех результатов всех условий.
Метод комбинаторного покрытия условий
Критерием, который отчасти более чувствителен к ошибкам в логических условиях, является комбинаторное покрытие условий. Он требует создания такого числа тестов, чтобы все возможные комбинации результатов условия в каждом решении выполнялись по крайней мере один раз. Набор тестов, удовлетворяющих критерию комбинаторного покрытия условий, удовлетворяет также и критериям покрытия решений, покрытия условий и покрытия решений/условий.

ПРИНЦИПЫ ТЕСТИРОВАНИЯ ЧЕРНОГО ЯЩИКА.

В этом методе программа рассматривается как чёрный ящик. Целью тестирования ставится выяснение обстоятельств, в которых поведение программы не соответствует спецификации.
Для обнаружения всех ошибок в программе необходимо выполнить *исчерпывающее тестирование*, то есть тестирование на всевозможных наборах данных.
Для большинства программ такое невозможно, поэтому применяют *разумное тестирование*, при котором тестирование программы ограничивается небольшим подмножеством всевозможных наборов данных. При этом необходимо выбирать наиболее подходящие подмножества, подмножества с наивысшей вероятностью обнаружения ошибок.

Свойства правильно выбранного теста

1. Уменьшает более, чем на одно число других тестов, которые должны быть разработаны для разумного тестирования.
2. Покрывает значительную часть других возможных тестов, что в некоторой степени свидетельствует о наличии или отсутствии ошибки до и после ограниченного множества тестов.

Приёмы тестирования чёрного ящика

1. Эквивалентное разбиение.

Основу метода составляют два положения:

1) исходные данные необходимо разбить на конечное число классов эквивалентности. В одном классе эквивалентности содержатся такие тесты, что, если один тест из класса эквивалентности обнаруживает некоторую ошибку, то и любой другой тест из этого класса эквивалентности должен обнаруживать эту же ошибку.

2) каждый тест должен включать, по возможности, максимальное количество классов эквивалентности, чтобы минимизировать общее число тестов.

2. Анализ граничных значений.

Отличается от эквивалентного разбиения следующим:

1) выбор любого элемента в классе эквивалентности в качестве представительного осуществляется таким образом, чтобы проверить тестом каждую границу этого класса.

2) при разработке тестов рассматриваются не только входные значения (пространство входов), но и выходные (пространство выходов).

Метод требует определённой степени творчества и специализации в рассматриваемой задаче.

3. Граничные условия

Рассматриваются ситуации, возникающие на высших и нижних границах входных классов эквивалентности.

4. Анализ причинно-следственных связей.

Этапы построения теста:

1. Спецификация разбивается на рабочие участки.
2. В спецификации определяются множество причин и следствий. Под *причиной* понимается отдельное входное условие или класс эквивалентности. *Следствие* представляет собой выходное условие или преобразование системы. Здесь каждой причине и следствию присваивается номер.
3. На основе анализа семантического (смыслового) содержания спецификации строится *таблица истинности*, в которой последовательно перебираются всевозможные комбинации причин и определяются следствия для каждой комбинации причин.
5. Предположение об ошибке.

Тестировщик с большим опытом выискивает ошибки без всяких методов, но при этом он подсознательно использует **метод предположения об ошибке**. Данный метод в значительной степени основан на интуиции.

Основная идея метода состоит в том, чтобы составить список, который перечисляет возможные ошибки и ситуации, в которых эти ошибки могли проявиться. Потом на основе списка составляются тесты.

№15. XML И ДОКУМЕНТИРОВАНИЕ ПО В DOCBOOK. наверх

Не вдаваясь в подробности, скажем, что *XML* — это язык описания языков разметки с общим синтаксисом и разными словарями. Свобода словаря позволяет создавать конкретные специализированные XML-языки, а общность синтаксиса — обрабатывать произвольные XML-данные одними и теми же программами. Для ввода данных вы сможете использовать любой XML-редактор, а для публикации - любой XSLT-процессор. Одновременно у вас есть возможность автоматически загружать данные в базу данных и находить там нужные сведения по существенным для предметной области признакам.

Основой технологической платформы DocBook/XML служит одноименный проблемно-ориентированный *язык разметки*. Он предназначен для записи текста технической документации на программы, алгоритмические языки, компьютерное оборудование и другие решения в области информационных технологий, чем принципиально отличается от большинства форматов хранения текстовых данных (но не XML-языков).

В языке DocBook/XML предусмотрены средства описания фрагментов, свойственных технической документации. Специальными элементами полагается выделять:

- названия элементов интерфейса,
- обозначения клавиш,
- имена переменных,
- термины,
- различные врезки (замечания, подсказки, предупреждения),
- листинги,
- описания выполняемых пользователем процедур.

Разметка, задаваемая языком DocBook/XML, носит преимущественно функциональный характер: автору предписано указывать роль, которую тот или иной фрагмент играет в тексте, а не способ его внешнего оформления. Такой подход сковывает автора, зато позволяет добиться заведомой независимости содержания и оформления выходного документа и унифицировать некоторые важные качества стиля изложения при работе нескольких авторов в одном проекте.

Преобразованием DocBook-документа в форматы, доступные для печатного или просто визуального представления (в том числе PDF, HTML, man-страницы) занимаются различные утилиты, обычно осуществляющие такое преобразование на основе настраиваемых шаблонов, или «таблиц стилей» (DSSSL или XSL), то есть происходит настоящая изоляция структуры документа от визуального представления.

В отличие от HTML-документа, DocBook-документ не рассматривается как конечный формат, поэтому, например, один документ в этом формате после преобразования может выглядеть и как один большой документ со сложной структурой, и как набор небольших простых документов-глав.

DocBook разрабатывался для создания технической документации, но может использоваться и в других целях (для создания сайтов, с преобразованием в HTML; для создания презентаций).

№16. ПРОТОТИПИРОВАНИЕ. ИНТЕРАКТИВНЫЕ ПРОТОТИПЫ. наверх

Прототипирование программного обеспечения — этап разработки программного обеспечения (ПО), процесс создания прототипа программы — макета (черновой, пробной версии) программы, обычно — с целью проверки пригодности предлагаемых для применения концепций, архитектурных и/или технологических решений, а также для представления программы заказчику на ранних стадиях процесса разработки.

Прототип позволяет также получить обратную связь от будущих пользователей, причем, именно тогда, когда это наиболее необходимо: в начале проекта еще есть возможность исправить ошибки проектирования практически без потерь.

Преимущества применения прототипирования:

- уменьшение времени, стоимости, рисков: прототипирование улучшает качество спецификаций; чем позднее проводятся изменения в спецификации, тем они дороже, поэтому, уточнение «чего же пользователи/заказчики хотят на самом деле» на ранних стадиях разработки — снижает общую стоимость.
- вовлечение пользователя в процесс разработки: прототипирование вовлекает будущих пользователей в процесс разработки, и позволяет им видеть то, как именно будет выглядеть будущая программа, что позволяет избавиться от возможных расхождений в представлении о программе между разработчиками и пользователями.

Недостатки:

- недостаточный анализ: концентрация усилий на ограниченном прототипе может отвлекать разработчиков от надлежащего анализа требований на полную систему.
- смешение прототипа и готовой системы в представлении пользователей: пользователи могут подумать, что прототип, который предполагается «выбросить», и есть основа будущей системы. Исходя из этого предположения, пользователи могут требовать от прототипа более точного поведения, могут разочароваться в возможностях разработчиков.
- чрезмерное время на создание прототипа: ключевое свойство прототипа — то, что он создается за короткое время. Если разработчики не принимают это во внимание, то они тратят время на создание слишком сложного прототипа, и теряют преимущества от применения прототипирования вообще.

Интерактивный прототип — это действующая модель пользовательского интерфейса. Он имитирует работу системы, так что ее можно оценить в действии еще до того, как начата разработка. Хотя прототип не сохраняет данные и не работает с базой данных, в остальном он может быть максимально приближен к будущему продукту. Обычно это соединенные между собой HTML-страницы имитацией реакций системы через JavaScript. Хотя в некоторых случаях он может быть сделан на Flash или других технологиях.

Назначение:

- Демонстрация инвесторам. Действующую модель продукта можно показать заказчику еще до того, как вложены время и деньги в разработку.
- Доработка концепции. При работе над сложными и инновационными проектами постоянно идут эксперименты над концепцией и, соответственно, интерфейсом. В прототип легко вносить правки, а значит и сравнивать альтернативные решения.
- Раннее юзабилити-тестирование. Прототип позволяет проверить удобство и эффективность системы, показав ее потенциальным пользователям.
- Часть технического задания для разработчиков. Команде разработки проще понять как должна работать система, поработав с ее действующей моделью.

№17. ПРОЛОГ. УПРАВЛЕНИЕ ВЫПОЛНЕНИЕМ ПРОГРАММЫ. наверх

Суть механизма **бэктрекинга** такова: в том месте программы, где возможен выбор нескольких вариантов, Пролог сохраняет в специальный стек *точку возврата* для последующего возвращения в эту позицию. Точка возврата содержит информацию, необходимую для возобновления процедуры при откате. Выбирается один из возможных вариантов, после чего продолжается выполнение программы.

Во всех точках программы, где существуют альтернативы, в стек заносятся *указатели*. Если впоследствии окажется, что выбранный вариант не приводит к успеху, то осуществляется откат к последней из имеющихся в стеке точек программы, где был выбран один из альтернативных вариантов. Выбирается очередной вариант, программа продолжает свою работу. Если все варианты в точке уже были использованы, то регистрируется неудачное завершение и осуществляется переход на предыдущую точку возврата, если такая есть. При откате все связанные переменные, которые были означены после этой точки, опять освобождаются.

При объяснении сущности **бэктрекинга** часто приводят пример с поиском пути в лабиринте. Один из возможных способов найти выход из лабиринта — это на каждой развилке поворачивать в одну и ту же сторону, до тех пор, пока не попадешь в тупик. После попадания в тупик нужно вернуться до ближайшей развилки. На ней нужно выбрать другое направление. После этого нужно опять на всех развилках выбирать поворот в том же направлении, что и в самом начале. Продолжаем этот алгоритм до тех пор, пока не выберемся из лабиринта.

Выбор среди альтернатив

Несколько утверждений в определении предиката рассматриваются как *альтернативные варианты*. Поэтому для реализации ветвления создаётся новое отношение, содержащее в описании столько утверждений, сколько ветвей. В теле каждого утверждения должны быть описаны условия, при которых выполняется эта ветвь. Условия должны быть взаимоисключающими.

Предикат fail

В Prolog есть встроенный предикат fail, не имеющий аргументов и всегда принимающий значение “ложь”. Предикат находит своё применение в моментах, когда в процессе выполнения необходимо <u>сгенерировать неудачу</u>. Например, создать процесс возврата для получения ответов на запрос.
Стоит отметить <u>общие принципы построения</u> повторяющегося процесса при организации возврата с помощью fail :

- Первое правило в определении рабочее и включает предикат fail для инициирования возврата ;
- Последнее в описании утверждение всегда истинно и обеспечивает успешное завершение возвратов.

Такой способ организации повторяющегося процесса возможен только, если в теле рабочего правила есть хотя бы один предикат, для которого в базе имеется несколько вариантов согласований

Предикат not

Встроенный предикат **not** имеет один аргумент. Этим аргументом является отношение, значение истинности которого (после обработки этого отношения) <u>заменяется на противоположное</u>.

Предикат repeat

Встроенный предикат **repeat** обеспечивает дополнительную возможность для <u>порождения множественных решений в процессе возврата</u>. Хотя он и является встроенным, его поведение полностью соответствует следующему определению:

```
repeat.
repeat :- repeat.
```

Что произойдет, если поместить предикат **repeat** как целевое утверждение в одно из наших правил? Во-первых, это целевое утверждение согласуется с базой данных, так как имеется соответствующий факт – первое утверждение определения предиката **repeat**. Во-вторых, если в процессе возврата вновь будет достигнуто это место, то Prolog будет иметь возможность испробовать альтернативу – правило (второе утверждение). При использовании правила порождается другое целевое утверждение **repeat**. Так как оно сопоставляется с фактом для предиката **repeat**, то это целевое утверждение вновь согласуется с базой данных. Если процесс возврата снова дойдет до этого места, то Prolog вновь использует правило там, где он ранее использовал факт. Чтобы доказать согласованность вновь порожденного целевого утверждения, он снова выберет факт. И так далее. В действительности в процессе возврата целевое утверждение **repeat** может быть согласовано бесконечное число раз. Обратим внимание на существенность порядка, в котором записаны утверждения для предиката **repeat**.

Для чего нужно порождать целевые утверждения, которые всегда будут согласовываться в процессе возврата? Они нужны для того, чтобы создавать правила, в которых имеется возможность выбора вариантов, из правил, которые такой возможности не содержат. И тем самым мы можем заставить их порождать каждый раз различные значения.

№18. ПРОЛОГ. РЕКУРСИЯ И ОТСЕЧЕНИЕ. наверх

Рекурсия определяется как способ описания объектов, данных процессов или функций через самих себя. Процесс на очередном шаге определяется через тот же самый процесс на предыдущем шаге. Чтобы перейти к рекуррентному определению в программе, необходимо для описания процесса определить предикат, который обращался бы к самому себе, т.е. к описанию того же процесса, при других значениях аргументов. При этом допускается, что уже имеется процесс, который правильно выполняется и позволяет получить состояние на предшествующем шаге.

Пример: вычислить факториал F = n!

```
predicates
fact (integer, integer)
clauses
fact (0, 1) :- !.
fact (K, F) :- K1=K-1, fact (K1, F1), F=F1*k.
```

Рекурсивное определение в программе делится на две части : одна соответствует рекуррентным формулам и состоит из правил, рекурсивно вызывающих самих себя, другая часть содержит утверждения, описывающие ситуации, в которых рекурсивное обращение предиката к самому себе прекращается.

В качестве правила остановки выбирается одно из граничных условий, обычно являющемся первым утверждением в определении.

При выполнении происходит следующее:

- Оценивается 1-ое утверждение в рекурсивном определении;
- Если 1-ое утверждение не выполняется, осуществляется переход к следующему в определении утверждению, и оно оценивается;
- После прохождения первого уровня рекурсии выполнение возвращается к 1-му утверждению в определении, повторно оценивая его истинность;
- Если оценивание заканчивается неудачей, выполнение переходит ко 2-му в определении утверждению и входит во 2-ой уровень рекурсии. Процесс выполняется до тех пор, пока 1-ое утверждение не выполнится и, таким образом, остановит рекурсию.

Различают два стиля в определении рекурсивных определений:

1. *Нисходящая рекурсия.*

Описание процесса начинается с конца, с момента $K = N$, и номер шага постепенно уменьшается на 1. В качестве правила остановки выбираются начальные условия.

2. *Восходящая рекурсия.*

Описание процесса начинается с начала, с момента $K = 0$. Номер шага постепенно возрастает. В качестве правила остановки выбирается процесс достижения условий окончания процесса $K = N$. Начальные условия задаются в качестве значений аргументов целевого утверждения-запроса. Параметры, характеризующие состояние процесса, вычисляются на каждой стадии рекурсии в процессе выполнения прямого хода.

Системный предикат **отсечение** *cut* предназначен для управления процессом возврата, позволяющий сократить пространство поиска решений при возврате. Синтаксически использование отсечения в правиле выглядит как вхождение целевого утверждения с предикатом *cut*, не имеющим аргументов (допускается использование *“!”* вместо *cut*). При прямом ходе доказательства некоторой цепи отсечение не оказывает никакого воздействия на процесс выполнения. Оно работает только при инициализации процесса возврата вследствие ложности одного из предикатов, расположенных после отсечения в теле правила.

Действие отсечения сводится к следующим моментам:

- Отсечение выбрасывает из рассмотрения все утверждения, расположенные после предложения, в котором находится отсечение;
- Отсечение отбрасывает все альтернативные решения конъюнкции целей, стоящих перед отсечением, приводит к не более чем 1-му решению;
- Отсечение не влияет на цели, расположенные правее него. В случае возврата они могут порождать более 1-го решения;

- Отсечение при возврате сокращает пространство поиска вариантов доказательства;
- Отсечение применяется для устранения бесконечных циклов, при программировании взаимоисключающих утверждений и при необходимости неудачного завершения доказательства цели.

№19. СОЗДАНИЕ МОДЕЛЕЙ БИЗНЕСА В СТАНДАРТАХ IDEF. наверх

Бизнес-процесс – это логичный, последовательный, взаимосвязанный набор мероприятий, который потребляет ресурсы, создаёт ценность и выдаёт результат.
Моделирование бизнес-процессов – это эффективное средство поиска путей оптимизации деятельности компании, позволяющее определить, как компания работает в целом и как организована деятельность на каждом рабочем месте.
Под методологией создания модели бизнес-процесса понимается совокупность способов, при помощи которых объекты реального мира и связи между ними представляются в виде модели. Для каждого объекта и связей характерны ряд параметров, или атрибутов, отражающих определённые характеристики реального объекта (номер объекта, название стоимость и др.).
Основу многих современных методологий моделирования бизнес-процессов составили методология SADT (метод структурного анализа и проектирования), семейство стандартов IDEF и алгоритмические языки.

IDEF - методологии создавались в рамках предложенной ВВС США программы компьютеризации промышленности. Принципиальным требованием при разработке рассматриваемого семейства методологий была возможность эффективного обмена информацией между всеми специалистами. После опубликования стандарта он был успешно применен в самых различных областях бизнеса, показав себя эффективным средством анализа, конструирования и отображения бизнес-процессов.

В основе **IDEF0** методологии лежит понятие блока, который отображает некоторую *бизнес-функцию*. Четыре стороны блока имеют разную роль: левая сторона имеет значение "входа", правая - "выхода", верхняя - "управления", нижняя - "механизма".

Взаимодействие между функциями в IDEF0 представляется в виде дуги, которая отображает поток данных или материалов, поступающий с выхода одной функции на вход другой. В зависимости от того, с какой стороной блока связан поток, его называют соответственно "входным", "выходным", "управляющим".

В IDEF0 реализованы три базовых принципа моделирования процессов:

1. **Принцип функциональной декомпозиции**

   Представляет собой способ моделирования типовой ситуации, когда любое действие, операция, функция могут быть разбиты (декомпозированы) на более простые действия, операции, функции.
2. **Принцип ограничения сложности** (2< количество блоков < 6);

   Существенным является условие их разборчивости и удобочитаемости. Количество блоков на диаграмме должно быть не менее двух и не более шести.
3. **Принцип контекста** (контекстная диаграмма);

Моделирование делового процесса начинается с построения контекстной диаграммы. На этой диаграмме отображается только один блок - главная бизнес-функция моделируемой системы. Она определяет, значение и границы моделируемой бизнес-системы .

Методология функционального моделирования IDEF0 является достаточно простым инструментом, который позволяет разработчикам корпоративных информационных систем изучить сферу деятельности заказчика и решать задачи по повышению эффективности этой деятельности.

IDEF3 - методология моделирования, использующая графическое описание информационных потоков, взаимоотношений между процессами обработки информации и объектов, являющихся частью этих процессов. IDEF3 дает возможность аналитикам описать ситуацию, когда процессы выполняются в определенной последовательности, а также описать объекты, участвующие совместно в одном процессе.

Объекты IDEF3-диаграммы.
Работа изображается прямоугольником с прямыми углами (рис. 1) и имеет имя, обозначающее процесс действия, и номер. Все стороны работы равнозначны. В каждую работу может входить и выходить ровно по одной стрелке.

Рисунок 1. Работа IDEF3

Связи показывают взаимоотношения работ. Все связи в IDEF3 однонаправлены и могут быть направлены куда угодно, но обычно диаграммы IDEF3 стараются построить так, чтобы связи были направлены слева направо. В IDEF3 возможны *три вида связей*:

Изображение стрелки	**Название**	**Описание**
	Старшая (Precedence) стрелка	сплошная линия, связывающая единицы работ (UOW). Рисуется слева направо или сверху вниз. Показывает, что работа-источник должна закончиться прежде, чем работа-цель начнется
	Потоки объектов (Object Flow)	стрелка с двумя наконечниками, применяется для описания того факта, что объект используется в двух или более единицах работы, например когда объект порождается в одной работе и используется в другой
	Стрелка отношения (Relational Link)	пунктирная линия, использующаяся для изображения связей между единицами работ (UOW), а также между единицами работ и объектами ссылок. Значение задается аналитиком отдельно для каждого случая

Перекрестки используются для отображения логики взаимодействия стрелок при слиянии и разветвлении или для отображения множества событий, которые могут или должны быть завершены перед началом следующей работы. Различают перекрестки для слияния и разветвления стрелок. Перекресток не может использоваться одновременно для слияния и для разветвления.

Типы перекрестков:

Обозначение	**Наименование**	**Смысл в случае слияния стрелок**	**Смысл в случае разветвления стрелок**
&	Асинхронное «И» (Asynchronous AND)	Все предшествующие процессы должны быть завершены	Все следующие процессы должны быть запущены
&	Синхронное «И» (Synchronous AND)	Все предшествующие процессы завершены одновременно	Все следующие процессы запускаются одновременно
O	Асинхронное «ИЛИ» (Asynchronous OR)	Один или несколько предшествующих процессов должны быть завершены	Один или несколько следующих процессов должны быть запущены
O	Синхронное «ИЛИ» (Synchronous OR)	Один или несколько предшествующих процессов завершены одновременно	Один или несколько следующих процессов запускаются одновременно
X	Исключающее «ИЛИ» XOR (Exclusive OR)	Только один предшествующий процесс завершен	Только один следующий процесс запускается

Объект ссылки в IDEF3 выражает некую идею, концепцию или данные, которые нельзя связать со стрелкой, перекрестком или работой. Они используются в модели для привлечения внимания читателя к каким-либо важным аспектам модели. При внесении объектов ссылок помимо имени следует указывать тип объекта ссылки (рис. 2).

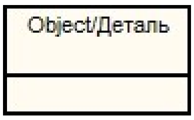


Рисунок 2. Объект ссылки

№20. ОРГАНИЗАЦИОННЫЕ СТРУКТУРЫ И БИЗНЕС-ПРОЦЕССЫ. наверх

Организационная структура — документ, схематически отражающий состав и иерархию подразделений предприятия. Организационная структура устанавливается исходя из целей деятельности и необходимых для достижения этих целей подразделений, выполняющих функции, составляющие бизнес-процессы организации.

Организационная структура определяет распределение ответственности и полномочий внутри организации. Как правило, она отображается в виде органиграммы — графической схемы, элементами которой являются иерархически упорядоченные организационные единицы (подразделения, должностные позиции).

Типологии организационных структур:

- Иерархическая;
- Линейная;
- Линейно-штабная;
- Функциональная;
- Упрощённая матричная;
- Сбалансированная матричная;
- Усиленная матричная;
- Проектная;
- Процессная;
- Дивизиональная.

Критерием типологии организационных структур является распределение ответственности (способ группирования ответственности). Нередко организационную структуру подстраивают под процесс производства продуктов или услуг в зависимости от типа производства и вида производства.

Основные параметры проектирования организационной структуры

Параметры индивидуальной деятельности:

- уровень специализации деятельности;
- уровень формализации поведения;
- параметры обучения;
- параметры восприятия организационной культуры.

Структурные параметры:

- параметры группировки ресурсов по подразделениям;
- размеры подразделений.

Параметры поперечных связей:

- параметры системы планирования и контроля;
- параметры механизмов связи.

Параметры системы принятия решений:

- уровень вертикальной децентрализации;
- уровень горизонтальной децентрализации.

Бизнес-процесс — это совокупность взаимосвязанных мероприятий или задач, направленных на создание определённого продукта или услуги для потребителей. В качестве графического описания деятельности применяются блок-схемы бизнес-процессов.

Существуют три вида бизнес-процессов:

- *Управляющие* — бизнес-процессы, которые управляют функционированием системы. Примером управляющего процесса может служить Корпоративное управление и Стратегический менеджмент.
- *Операционные* — бизнес-процессы, которые составляют основной бизнес компании и создают основной поток доходов. Примерами операционных бизнес-процессов являются: Снабжение, Производство, Маркетинг и Продажи.
- *Поддерживающие* — бизнес-процессы, которые обслуживают основной бизнес. Например, Бухгалтерский учет, Подбор персонала, Техническая поддержка, АХО.

Бизнес-процесс начинается со спроса потребителя и заканчивается его удовлетворением. Процессно-ориентированные организации стараются устранять барьеры и задержки, возникающие на стыке двух различных подразделений организации при выполнении одного бизнес-процесса.

Бизнес-процесс может быть декомпозирован на несколько подпроцессов, процедур и функций, которые имеют собственные атрибуты, однако также направлены на достижение цели основного бизнес-процесса. Такой анализ бизнес-процессов обычно включает в себя составление карты бизнес-процесса и его подпроцессов, разнесенных между определенными уровнями активности.

Бизнес-процессы должны быть построены таким образом, чтобы создавать стоимость и ценность для потребителей и исключать любые необязательные или вовсе лишние активности. На выходе правильно построенных бизнес-процессов увеличиваются ценность для потребителя и рентабельность (меньшая себестоимость производства товара или услуги).

Бизнес-процессы могут подвергаться различному анализу в зависимости от целей моделирования. Анализ бизнес-процессов может применяться при бизнес-моделировании, функционально-стоимостном анализе, формировании организационной структуры, реинжиниринге бизнес-процессов, автоматизации технологических процессов.

Одним из методов анализа текущей деятельности является составление модели бизнес-процесса «как есть». После этого модель бизнес-процесса подвергается критическому анализу или обрабатывается специальным программным обеспечением. По результатам анализа формируется модель бизнес-процесса «как должно быть» и план мероприятий по внедрению необходимых изменений.

Существует множество нотаций, применяемых для моделирования бизнес-процессов, например:

- BPMN — функциональная последовательность работ ;
- EPC — событийная последовательность работ ;
- IDEF0 — логическая последовательность работ.

№21. UML. ДИАГРАММЫ КЛАССОВ И ПОСЛЕДОВАТЕЛЬНОСТЕЙ. наверх

UML (Unified Modeling Language) - унифицированный язык моделирования – это язык графического описания для объектного моделирования в области разработки программного обеспечения. UML является языком широкого профиля, это открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML моделью. UML был создан для определения, визуализации, проектирования и документирования в основном программных систем.

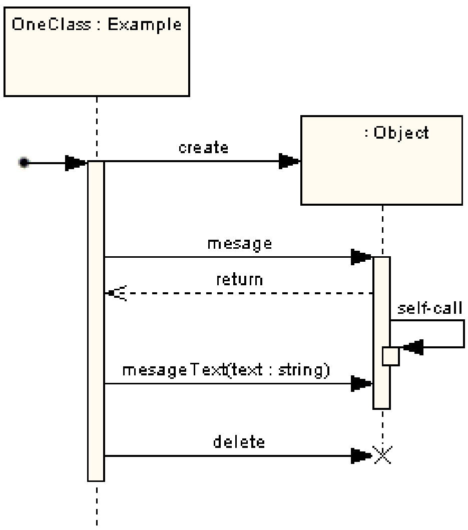
Использование UML не ограничивается моделированием программного обеспечения. Его также используют для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

РАЗНОВИДНОСТИ

Существует 13 официальных диаграмм UML 2.0, каждая из которых представляет собой различное представление разных аспектов системы:

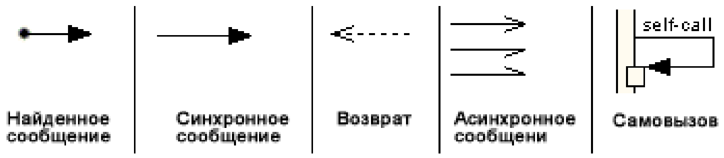
- Диаграмма активности;
- Диаграмма классов;
- Диаграмма связей;
- Диаграмма компонентов;
- Диаграмма составных структур;
- Диаграмма развертывания;
- Диаграмма обзора взаимодействий;
- Диаграмма объектов;
- Диаграмма пакетов;
- Циклограмма;
- Диаграмма машин состояния;
- Диаграмма синхронизации;
- Диаграмма прецедентов.

Диаграммы взаимодейстаия – описывают взаимодействие групп объектов в различных условиях их поведения. Наиболее используемым типом данных диаграмм является диаграмма последовательности.
Диаграмма последовательности – это диаграмма, чаще всего, описывающая один сценарий приложения. На диаграмме изображаются экземпляры объектов и сообщения, которыми они обмениваются в рамках одного прецедента (***use case***).
Пример:

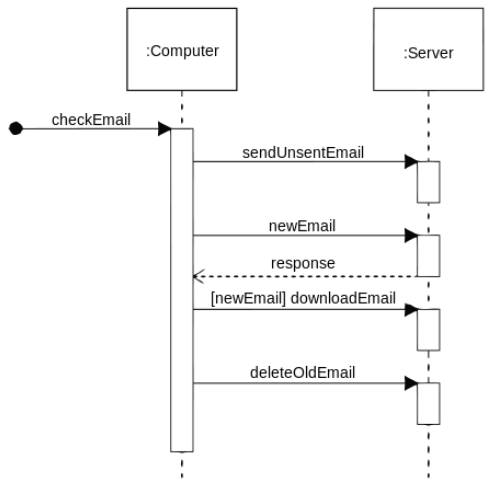


На диаграмме последовательности, каждый участник представлен вместе со своей **линией жизни** (***lifeline***), это вертикальная линия под объектом, вертикально упорядочивающая сообщения на странице.
Все сообщения на диаграмме следует читать сверху вниз. Каждая линия жизни имеет **полосу активности** (прямоугольники**)**, которая показывает интервал активности каждого участника при взаимодействии.
Участники диаграммы именуются сделующим образом: **имя : Класс**, где и имя, и класс являются не обязательными, но если используется класс, то присутствие двоеточия обязательно.

Сообщения можно разделить на 2 вида: **синхронные (*synchronous message*)** – требующие возврата ответа и**асинхронные (*asynchronous message*)** – ответ не требуется и вызывающий объект может продолжать работу. На диаграмме синхронные вызовы обозначаются закрашенными стрелочками. Асинхронные – незакрашенными или половинными стрелочками.
У первого сообщения, в нашей диаграмме, нет участника, пославшего его, поскольку оно приходит от неизвестного источника. Такое сообщение называется **найденным сообщением (*found message*)**.
Обратной пунктирной стрелкой показан возврат ответа на сообщение **message** (т.к. сообщение message является синхронным). Лучше применять изображение возврата только в тех случаях, когда это поможет лучше понять устройство взаимодействия. Во всех остальных случаях, стоит опускать изображения возвратов, т.к. они будут вносить некоторую неразбериху. Просто, при использовании синхронного сообщения, стоит помнить, что у него всегда есть возврат.
Если в сообщении требуется передать параметры, то они указываются в скобках через запятую, с указанием типа.
Итог:



Создание и удаление участников
На первом рисунке создание нового объекта – это сообщение **create**. Создаваемый объект находится всегда ниже объекта из которого он был создан. Если при создании применяется конструктор то имя сообщения не обязательно, но для ясности все же лучше писать **new**.
Удаление участника обозначается крестом. Стрелка сообщения идущая к такому кресту, означает, что один участник, явным образом удаляет другого участника. Если же крестик стоит в конце линии жизни участника, и к нему не идет никаких сообщений – это означает, что участник удаляет сам себя, т.е. срабатывает деконструктор.
Циклы, условия и другие.
Диаграммы последовательностей хороши когда требуется графически представить взаимодействие нескольких объектов, в рамках определенного сценария. Но вот для точного определения поведения объектов, эти диаграммы подходят гораздо хуже.
Разработчики UML все же нашли способ отображать на диаграмме циклы и условные операции. Для этого используются **фреймы взаимодействий (*interaction frames*)**.
Фреймы взаимодействия, это нововведение UML 2, но многие разработчики, при обозначении циклов и условий, продолжают пользоваться приемами из UML 1. В UML 1, для этих целей использовались **маркеры итераций** и**маркеры защиты**.
Маркер итераций (***iteration marker***) обозначается символом *, добавленным к имении сообщения. Для уточнения количества итераций, к маркеру можно добавить текст в квадратных скобках, например, маркер – *** [для всех элементов массива],** показывает, что сообщение, к которому он добавлен, должно быть применено ко всем элементам массива.
Защита (***guard***) – это условное выражение, которое записывается в квадратных скобках и означающее, что сообщение должно быть послано только когда защита принимает истинное значение.
Пример. Взаимодействия при обработке электронной почты:



№22. ОБЪЕКТНАЯ МОДЕЛЬ PHP. наверх

Описание классов в PHP начинаются служебным словом class:

```
class Имя_класса {
// описание членов класса - свойств и методов для их обработки
}
```

Для **объявления объекта** необходимо использовать оператор new. Данные описываются с помощью служебного слова var.

Пример класса на PHP:

```
<?php
// Создаем новый класс Coor:
class Coor {
// данные (свойства):
var $name;
var $addr;

// методы:
 function Name() {
 echo "<h3>John</h3>";
 }
}
// Создаем объект класса Coor:
$object = new Coor;
?>
```

Доступ к классам и объектам в PHP можно получить с помощью оператора ->. Приведем пример:

```
<?php
// Создаем новый класс Coor:
class Coor {
// данные (свойства):
var $name;

// методы:
 function Getname() {
 echo "<h3>John</h3>";
 }
}
// Создаем объект класса Coor:
$object = new Coor;
// Получаем доступ к членам класса:
$object->name = "Alex";
echo $object->name;
// Выводит 'Alex'
// А теперь получим доступ к методу класса (фактически, к функции внутри класса):
$object->Getname();
// Выводит 'John' заглавными буквами
?>
```

Чтобы получить **доступ к членам класса внутри класса** (не только к методам, но и к данным), необходимо использовать указатель **$this**, который всегда относится к текущему объекту.

```
function Setname($name) {
$this->name = $name;  // доступ к данным класса
$this->Getname();     // доступ к методам класса
}
```

Конструкторы. Имя метода-конструктора должно совпадать с именем класса, в котором он содержится. Пример конструктора:

```
<?
class Webpage {
var $bgcolor;
 function Webpage($color) {
 $this->bgcolor = $color;
 }
}

// Вызвать конструктор класса Webpage
$page = new Webpage("brown");
?>
```

Деструкторы. В PHP отсутствует непосредственная поддержка деструкторов. Тем не менее, можно легко имитировать работу деструктора, вызывая функцию PHP **unset()**. Эта функция уничтожает содержимое переменной и возвращает занимаемые ею ресурсы системе. Например, после завершения работы с этим с объектом $Webpage вызывается функция:

```
unset($Webpage);
```

Необходимость в вызове деструкторов возникает лишь при работе с объектами, использующими большой объем ресурсов, поскольку все переменные и объекты автоматически уничтожаются по завершении сценария.

Обращение к элементам классов осуществляется с помощью оператора :: "двойное двоеточие". Используя "::", можно обращаться к методам классов используя имена этих классов.

```
<?php
class A {
     function example() {
         echo "Это первоначальная функция A::example().<br>";
     }
}
class B extends A {
     function example() {
         echo "Это переопределенная функция B::example().<br>";
         A::example();
     }
}
// Не нужно создавать объект класса A.
// Выводит следующее:
// Это первоначальная функция A::example().
A::example();
// Создаем объект класса B.
$b = new B;
// Выводит следующее:
//   Это переопределенная функция B::example().
//   Это первоначальная функция A::example().
$b->example();
?>
```

Работа с объектами классов PHP.

Копирование объектов. В PHP все переменные, в том числе и объекты, всегда рассматриваются как простой набор значений и копируются целиком. Например, если у нас есть объект $a и мы выполняем оператор $b=$a, то все содержимое $a будет скопировано в $b один-в-один.

```
<?php
class A {
// Создаем новый метод:
 function Test() {
```

```
 echo "<h1>Hello!</h1>";
 }
}
// Создаем объект класса A:
$a=new A();
// Копируем объект $a:
$b=$a;
// Теперь работаем с новым объектом $b
$b->Test(); // Выводит 'Hello!'
?>
```

Сравнение объектов. В PHP 4 объекты сравниваются очень просто: по именам. Два объекта равны, если они имеют те же самые свойства и значения, а также являются экземплярами одного и того же класса. Сравнение двух объектов осуществляют, используя оператор эквивалентности (===). Пример:

```
<?php
class A {
// Создаем новый метод:
 function Test() {
 echo "<h1>Hello!</h1>";
 }
}
// Создаем объект класса A:
$a=new A();
// Создаем объект класса A:
$b=new A();
// Выводит 'Объекты равны':
if ($a===$b) echo "<h3>Объекты равны</h2>";
?>
```

PHP позволяет создавать **ссылки на объекты**. Пример:

```
<?php
class A {
// Создаем новый метод:
 function Test() {
 echo "<h1>Hello!</h1>";
 }
}

// Создаем объект класса A:
$a=new A();
// Ссылка на объект класса A:
$b=& new A();
$b->Test();
?>
```

№23. СТРУКТУРЫ ДАННЫХ JAVASCRIPT. наверх

Переменные

Переменные в JavaScript могут быть определены назначением или при помощи оператора var. Переменные могут принимать самые разные значения, при этом тип переменной определяется контекстом.

```
i=10;
var i;
var i=10;
var id = window.open();
var a = new Array();
```

Массивы

Массивы делятся на встроенные (document.links[], document.images[],...) и определяемые пользователем .
Для определения массива пользователя существует специальный конструктор:

```
a = new Array();
b = new Array(10);
c = new Array(10,"Это значение");
```

Пример использования:

```
<SCRIPT>
c = new Array(30,"Это значение");
</SCRIPT>
<FORM><INPUT SIZE=& {c[0];};
value=& {c[1];};
onFocus="this.blur();">
</FORM>
```

Массив может состоять из разнородных элементов. Массивы не могут быть многомерными.
Массивы обладают свойством **length**, которое позволяет получить число элементов массива.
Для работы с массивами применяются методы:
Метод join() позволяет объединить элементы массива в одну строку. Он является обратной *функцией* методу split(), который применяется к *объектам* типа STRING.
Метод reverse() применяется для изменения на противоположный порядка элементов массива внутри массива. Предположим, массив натуральных чисел упорядочен по возрастанию:

```
a = new Array(1,2,3,4,5);
```

Упорядочим его по убыванию:

```
a.reverse();
```

a[0]=5 a[1]=4 a[2]=3 a[3]=2 a[4]=1
Метод sort() позволяет отсортировать элементы массива в соответствии с некоторой *функцией* сортировки, чье имя используется в качестве аргумента метода:

```
a = new Array(1,6,9,9,3,5);
function g(a,b)
{
    if(a > b) return 1;
    if(a < b) return -1;
    if(a==b) return 0;
}
b = a.sort(g);
```

В результате выполнения этого кода получим массив следующего вида:
b[0]=1 b[1]=3 b[2]=5 b[3]=6 b[4]=9 b[5]=9

Функции

В JavaScript *функция* выступает в качестве одного из основных *типов данных*. Одновременно с этим в JavaScript определен *объект* Function.
В общем случае любой объект JavaScript определяется через *функцию*. Для создания объекта используется *конструктор*, который в свою очередь вводится через Function. Таким образом, с функциями в JavaScript связаны следующие ключевые вопросы:

- функция — тип данных;
- функция — объект;
- конструкторы объектов.

Определяют функцию при помощи ключевого слова function:

```
function f_name(arg1,arg2,...)
{
    /* function body */
}
```

Объекты

Сначала рассмотрим пример произвольного, определенного пользователем *объекта*, потом выясним, что же это такое:

```
function Rectangle(a,b,c,d)
{
this.x0 = a;
this.y0 = b;
this.x1 = c;
this.y1 = d;
this.area = new Function( "return Math.abs(this.x0-this.x1)*Math.abs(this.y0-this.y1)");
this.perimeter = new Function( "return (Math.abs(this.x0-this.x1) + Math.abs(this.y0-this.y1))*2");
}
c = new Rectangle(0,0,100,100);
document.write(c.area());
```

Результат исполнения:

10000

Функция rectangle() — это конструктор объекта класса Rectangle, который определяется пользователем. Конструктор позволяет создать реальный объект данного класса. Для того чтобы эти действия функции были выполнены, необходимо передать функции управление. В нашем примере это делается при помощи оператора **new** . Он вызывает функцию и тем самым генерирует реальный объект.

Создается четыре переменных: x0, y0, x1, y1 — это свойства объекта Rectangle. К ним можно получить доступ только в контексте объекта данного класса, например:

```
up_left_x = c.x0;
up_left_y = c.y0;
```

Кроме свойств мы определили внутри конструктора два объекта типа Function(), применив встроенные конструкторы языка JavaScript, — area и perimeter. Это методы объекта данного класса. Вызвать эти функции можно только в контексте объекта класса Rectangle:

```
sq = c.area();
length = c.perimeter();
```

Методы объекта

Метод **toString()** осуществляет преобразование объекта в строку символов. Он используется в JavaScript-программах повсеместно, но неявно.

Аналогично ведет себя и метод **valueOf().** Этот метод позволяет получить значение объекта. В большинстве случаев он работает подобно методу toString(), особенно если нужно выводить значение на страницу.

В отличие от двух предыдущих методов, **assign()** позволяет не прочитать, а переназначить свойства и методы объекта. Данный метод используется в контексте присваивания объекту некоторого значения:

```
object = value; <=> object.assign(value);
```

№24. ОСНОВНЫЕ ПОНЯТИЯ И ФУНКЦИИ ГИС. наверх

На базе информационных технологий созданы *географические информационные системы* (ГИС) – особые аппаратно-программные комплексы, обеспечивающие сбор, обработку, отображение и распространение пространственно координированных данных. Ода из основных функций ГИС – создание и использование компьютерных (электронных) карт, атласов и других картографических произведений.

Территориальные уровни ГИС

Вид ГИС	Охват территории	Масштабы
Глобальные	$5*10^8$ км^2	1:1 000 000–1:1 00 000 000
Национальные	10^4–10^7 км^2	1:1 000 000–1:10 000 000
Региональные	10^3–10^5 км^2	1:100 000–1:2 500 000
Муниципальные	10^3 км^2	1:1 000 000–1:1 000 000 000
Локальные (заповедники, национальные парки и др.)	10^2–10^3 км^2	1:1000–1:50 000

ГИС подразделяются и по <u>проблемной ориентации</u> (тематике):

- земельные информационные системы (ЗИС),
- кадастровые (КИС),
- экологические (ЭГИС),
- учебные,
- морские, и многие другие системы.

К <u>обязательным признакам ГИС</u> относятся:

- географическая пространственная привязка данных;
- генерирование новой информации на основе синтеза имеющихся данных;
- отражение пространственно-временных связей объектов;
- обеспечение принятия решений;
- возможность оперативного обновления баз данных за счет вновь поступающей информации.

<u>Управление ГИС</u> осуществляют пользователи, которые разрабатывают и поддерживают систему или просто решают поставленные задачи.

<u>Структуру ГИС</u> обычно представляют как набор информационных слоев. *Слой* – это совокупность однотипных пространственных объектов, относящихся к одной теме или классу объектов в пределах некоторой территории и в системе координат, общих для набора слоев.
Основу любой ГИС составляет *автоматизированная картографическая система* – комплекс приборов и программных средств, обеспечивающих создание и использование карт, которая состоит из ряда подсистем, важнейшие из которых являются подсистемы ввода, обработки и вывода информации.

<u>Функциональные возможности</u> ГИС многообразны, основные из них:

- ввод в компьютер цифровых данных;
- преобразование данных, трансформация картографических проекций, конвертирование данных в различные форматы;
- хранение и управление данными;
- картометрические операции и др.

В ГИС все <u>объекты</u> представлены пространственными объектами, которые подразделяются на <u>четыре типа</u>: точки, линии, области и поверхности. Вместе они могут представлять большинство природных и социальных феноменов, которые мы встречаем каждый день.
Точечные объекты – это такие объекты, каждый из которых расположен только в одной точке пространства, например, <u>деревья, отметки высот</u> и мн. др. О таких объектах говорят, что они дискретные, в том смысле, что каждый из них может занимать в любой момент времени только определенную точку пространства. В целях моделирования считают, что у таких объектов нет пространственной протяженности, длины или ширины, но каждый из них может быть обозначен координатами своего местоположения.
Линейные объекты представляются как одномерные в нашем координатном пространстве. Такими «одномерными» объектами могут быть <u>дороги, реки, границы,</u> любые другие объекты, которые существенно длинны и узки.

Полигоны или *площадные объекты* представляются как двумерные в координатном пространстве, т. е. у них есть длина и ширина. Ими могут быть озера, поля, здания и т. д.

Карты предназначены для того, чтобы представлять не только объекты на ее поверхности, но и форму Земли. Глобус – традиционный способ отображения формы Земли. Картографы разработали набор методов, называемых картографическими проекциями, которые предназначены для изображения с приемлемой точностью сферической Земли на плоском носителе. Процесс проецирования сферической поверхности на плоский носитель выполняется с использованием методов геометрии и тригонометрии, которые воспроизводят физический процесс проецирования света через глобус.

Атрибут – это элементарное данное, описывающее свойство какого-либо элемента модели (объектами понятия). Атрибутами могут быть символы (названия), числа (отражающие статистические характеристики), графические признаки (цвет, рисунок, графическая структура контура и т. п.). Ими также удобно отражать временные параметры.

Обычно атрибуты группируют в виде специальных таблиц, в таблице могут храниться как координаты объектов (координатные данные), так и описательные характеристики-атрибуты. С помощью атрибутов можно упорядочивать и типизировать данные. Таблицы производят строгое ранжирование параметров, определяющих различные признаки объектов, поскольку каждому объекту соответствует строка в таблице, а каждому тематическому признаку отводится свой столбец.

№25. ОРГАНИЗАЦИЯ ДАННЫХ В ГИС. КООРДИНАТНЫЕ, ВЕКТОРНЫЕ И РАСТРОВЫЕ МОДЕЛИ. наверх

Векторная модель данных (*vector data model*) или цифровое представление точечных, линейных и полигональных *пространственных объектов* в виде набора координатных пар, с описанием только геометрии объектов. Векторным моделям соответствует ***векторный формат*** пространственных данных (*vector data format*).

Векторные модели строятся с использованием векторов, в которых каждая точка на карте определяется через ее удаленность от опорной точки и величину угла между направлением на точку из опорной точки и направлением на Север (по часовой стрелке). Векторные изображения занимают значительно меньше памяти ЭВМ при хранении, чем растровые, требуют меньше затрат времени на обработку.

Базовым примитивом векторных моделей ГИС является *точка*. Через понятие «точка» определяются все остальные объекты векторной модели.

Безразмерные типы объектов:

- точка - определяет геометрическое местоположение объекта;
- узел – топологический переход или конечная точка, также может определять местоположение объекта.

Одномерные типы объектов:

- линия – одномерный объект;
- линейный сегмент – прямая линия между двумя точками;
- дуга – геометрическое место точек, которые формируют кривую, определенную математической функцией;
- связь – соединение между двумя узлами;
- направленная связь – связь с одним определенным направлением;
- кольцо – последовательность непересекающихся цепочек, строк, связей или замкнутых дуг.

Двумерные типы объектов:

- область – ограниченный непрерывный объект, который может включать или не включать в себя собственную границу;
- внутренняя область – область, которая не включает собственную границу;
- полигон (контур) – двумерный (площадной) объект, внутренняя область которого образована замкнутой последовательностью сегментов в модели «спагетти».

Векторное изображение можно получить различными способами. Наиболее часто используют *векторизацию* сканированного (растрового) изображения. Векторизация заключается в распознавании на растровом изображении объектов, выделение их, представление каждого объекта в векторном формате. Для автоматической векторизации необходимо иметь изображения высокого качества.

К особенностям векторных моделей можно отнести следующие:

- в векторной модели легко осуществляются некоторые операции с объектами, например, разбивка объекта (речной сети) на участки, замена условных обозначений;
- легко проводятся изменение масштаба, повороты, растягивание и другие операции;
- векторные модели имеют преимущество перед растровыми моделями в точности представления точечных объектов.

В ***растровых моделях*** дискретизация непрерывных последовательностей реального мира осуществляется наиболее простым способом: вся территория представляется последовательностью ячеек (пикселей), образующих регулярную сеть.

Если векторная модель дает представление, «где» находится объект, то растровая модель – «что» расположено в той или иной точке территории.

В качестве атомарной модели используется элементарный участок территории – *пиксель*. Упорядоченная последовательность пикселей образует *растр*, который является *моделью* карты. Каждый элемент растра имеет одно значение плотности или цвета.

Характеристики растровых моделей:

- разрешение;
- значение;
- ориентация;
- зона;

- положение.

Разрешение – минимальный линейный размер наименьшего участка пространства или поверхности, отображаемый одним пикселем. Пиксель чаще всего изображается прямоугольником или квадратом (иногда шестиугольником).

Значение – элемент информации, хранящийся в пикселе. В качестве типов значений могут использоваться разные классы значений: цифровые, буквенные и др. Для отображения значения чаще всего используются заливка.

Ориентация определяется через угол между направлением на Север и положением колонок растра.

Зона – это соседствующие друг с другом ячейки, имеющие одинаковые значения. Зоны могут присутствовать не во всех слоях. Основные характеристики зоны – значение и положение.

Буферная зона – зона, границы которой удалены на известное расстояние от любого объекта на карте.

Положение задается упорядоченной парой координат, которые однозначно отображают положение каждого элемента на карте.

<u>Достоинства растровых моделей:</u>

- растр не требует предварительного ознакомления с предметной областью; данные собираются с равномерно расположенной сети точек, могут легко подвергаться статистической обработке;
- растровые модели просты в обработке, возможна обработка по параллельным алгоритмам, за счет чего обеспечивается высокое быстродействие;
- некоторые задачи, например, создание буферной зоны, проще решаются в растровом виде;
- многие растровые модели позволяют вводить векторные объекты, обратная задача много труднее;
- процессы растеризации проще процессов векторизации алгоритмически.

<u>Основные недостатки растровых моделей:</u>

- требуют больших объемов (по сравнению с векторными моделями) памяти для хранения изображения;
- растровые объекты сложно масштабировать: при увеличении объекта становятся видны отдельные пиксели, контуры изображения теряют гладкость, изображение становится зернистым;
- сложно рассчитать результирующий цвет пикселя, который получается при слиянии нескольких пикселей разных цветов;
- проблемы разбиения сложного изображения на произвольные элементы для их раздельного использования и редактирования.
- растровые цифровые изображения занимают, как правило, большие объемы памяти.

Основные типы ***координатных данных*** в ГИС.

При построении ГИС применяют набор базовых геометрических данных, из которых затем компонуют остальные более сложные данные. В ГИС используются следующие <u>типы атомарных геометрических данных:</u>

- точка (узел, вершина);
- линия незамкнутая;
- контур (линия замкнутая);
- полигон (ареал, район) – группа прилегающих друг к другу замкнутых участков;
- пространственная сеть (развитие типа «полигон»).

На практике из этих атомарных моделей формируются сложные составные модели. В разных ГИС они отличаются, поэтому в качестве примера будем в дальнейшем рассматривать модели ГИС ГеоГраф (GeoGraph).

<u>Основные элементы промышленного пакета ГеоГраф</u> (см рис.):

1. *Точка* – геометрический объект, заданный парой координат X и Y.
2. *Отрезок* – линия, соединяющая две точки.
3. *Вершина* (вертекс) – начальная или конечная точка отрезка.
4. *Дуга* (полилиния) – упорядоченный набор связных отрезков.
5. *Узел* – начальная или конечная вершина дуги.

6. Висячий узел – узел, принадлежащий только одной дуге, у которой начальная и конечная вершины не совпадают.

7. Псевдоузел – узел, принадлежащий двум дугам или одной замкнутой дуге, у которой начальная и конечная вершины совпадают (узел, при прохождении которого нет альтернативы выбора дальнейшего пути).

8. Нормальный узел – узел, принадлежащий трем или более дугам, или узел, принадлежащий двум дугам, одна из которых самозамкнута на этом узле, а вторая примыкает к нему (узел, при прохождении которого есть возможность выбора дальнейшего пути).

9. Замкнутая дуга – дуга, у которой совпадают начальная и конечная вершины (дуга, у которой имеется только один узел).

10. Полигон – область, ограниченная замкнутой дугой или упорядоченным набором связанных дуг, которые образуют замкнутый контур.

11. Покрытие – набор файлов, фиксирующих в виде цифровых записей пространственные объекты и структуру отношений между ними.

12. Пустое покрытие – покрытие, на котором отсутствуют пространственные объекты.

13. Слой – покрытие, рассматриваемое в контексте его содержательной определенности (рельеф, растительность и др.).

Точечные объекты. К простейшим типам точечных объектов относятся не только собственно точки, но и условные точечные знаки.

Линейные объекты. Они используются для описания **сетей** (например, дорожная, транспортная, телефонная, гидрологическая сеть).

Любая сеть состоит из **узлов** (вершин) и обособленных линий и дуг (**звеньев**). Для каждого узла у линейных объектов существует характерист – *валентность*.

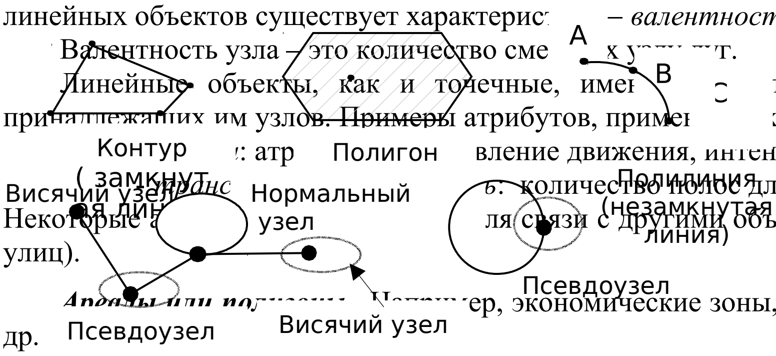
Валентность узла – это количество сме х узлы дуг.

Линейные объекты, как и точечные, име трибуты, причем разные для дуг и для принадлежащих им узлов. Примеры атрибутов, приме для описания дуг:

: атр вление движения, интенсивность движения, протяженность;

транс б: количество полос для движения; время в пути;

Некоторые а ля связи с другими объектами (например, названия пересекающих улиц).



Ареалы или полигоны. Например, экономические зоны, данные о сельскохозяйственных угодьях и др.

Рис. Основные элементы векторных данных

Взаимосвязи между координатными данными. Взаимосвязи могут существовать как между объектами одного типа, так и между объектами разных типов. Между координатными объектами выделяют <u>три типа взаимосвязей</u>:

Первый тип – связи, использующиеся для построения сложных объектов из простых элементов, т.е. взаимосвязи типа «состоит из».

Второй тип – взаимосвязи, которые можно вычислить по координатам объектов. Например, координаты точки пересечения двух линий – взаимосвязь типа «скрещивается», полигон и внутренняя точка – тип «содержится в».

Третий тип – «интеллектуальная связь». Это взаимосвязи, которые нельзя вычислить, их нужно заложить в базу данных при вводе.