**1)Алгоритм, Алгоритмизация. Пример алгоритма. Свойства алгоритма.  
Алгоритм** - конечная последовательность инструкций, исполнителю в результате выполнения которой обеспечивается получение из входных данных требуемых выходных результатов.  
**Алгоритмизация** — процесс составления алгоритмов для решения поставленных прикладных задач.  
**Свойства алгоритмов**:

1. Дискретность. Алгоритм должен представлять процесс решения задачи как последовательное выполнение простых шагов. Для выполнения каждого шага требуется конечный отрезок времени;

2. Определённость. Каждое правило должно быть чётким и однозначным;

3. К результативности алгоритм должен приводить к решению за конечное число шагов;

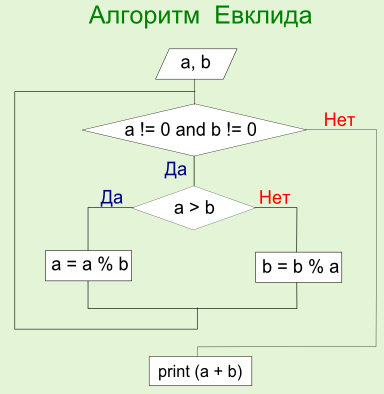
4. Массовость. Алгоритм решения задачи разрабатывается в общем виде, то есть он должен быть применим к некоторому классу задач отличающихся исходными данными.

5. Правильность. Алгоритм правильный, если его выполнение даёт правильные результаты решения поставленной задачи.  
**Пример алгоритма**: Алгоритм Евклида для нахождения НОД( наибольший общий делитель)  
while(a > 0 && b > 0)  
 if(a > b)  
 a %= b;  
 else  
 b %= a;  
 return a + b;   
  
1) Удостоверяемся, что оба числа больше нуля  
2) Большее число делим с остатком на меньшее   
3) Если делится без остатка, то меньшее число и есть НОД (следует выйти из цикла).  
4) Большее число заменяем на остаток от деления  
Пример:  
Найти НОД для 30 и 18.  
30 / 18 = 1 (остаток 12)  
18 / 12 = 1 (остаток 6)  
12 / 6 = 2 (остаток 0)  
Конец: НОД – это делитель 6.  
НОД (30, 18) = 6  
  
**2. Основные шаги для описания алгоритма. Приведите пример**

**алгоритма.**Существуют следующие способы представления алгоритмов:  
1)словесное описание;  
2)описание алгоритма с помощью математических формул;  
3)графическое представление алгоритма в виде блок-схемы;  
4)представление алгоритма с помощью псевдокода;  
5)комбинированный способ описания алгоритма с использованием, например, словесного и графического способов или словесного и с помощью математических формул и т.д. **Пример алгоритма**: Алгоритм Евклида для нахождения НОД( наибольший общий делитель)  
while(a > 0 && b > 0)  
 if(a > b)  
 a %= b;  
 else  
 b %= a;  
 return a + b;   
  
1) Удостоверяемся, что оба числа больше нуля  
2) Большее число делим с остатком на меньшее   
3) Если делится без остатка, то меньшее число и есть НОД (следует выйти из цикла).  
4) Большее число заменяем на остаток от деления  
Пример:  
Найти НОД для 30 и 18.  
30 / 18 = 1 (остаток 12)  
18 / 12 = 1 (остаток 6)  
12 / 6 = 2 (остаток 0)  
Конец: НОД – это делитель 6.  
НОД (30, 18) = 6  
  
**3. Средства описания алгоритма: Операторная форма. Привести пример.**Алгоритм моделирует решение задачи в виде точно определенной последовательности действий для некоторого исполнителя по преобразованию исходных данных в результирующие.  
Для его **описания** используется алгоритмический язык – набор символов и правил образования и истолкования конструкций из этих символов для записи алгоритмов.  
  
Основными изобразительными **средствами алгоритмов** являются следующие способы их записи:  
  
словесный;  
  
формульно-словесный;  
  
графический;  
  
псевдокод;  
  
табличный;  
  
операторный.

**Операторный способ**, который позволяет расчленить сложную задачу на отдельные функциональные части, из которых составляется общая операторная схема решения задачи. При этом отдельные операторы могут быть как стандартизированными, так и произвольными. В последнем случае поясняется содержание операторов. Для удобства записи логических схем операторы, образующие схему, располагаются в одну строку.

**Операторный способ записи алгоритма** более компактен, чем с помощью блок-схем, но менее нагляден, поэтому используется реже. **Пример алгоритма**: Алгоритм Евклида для нахождения НОД( наибольший общий делитель)  
while(a > 0 && b > 0)  
 if(a > b)  
 a %= b;  
 else  
 b %= a;  
 return a + b;   
  
1) Удостоверяемся, что оба числа больше нуля  
2) Большее число делим с остатком на меньшее   
3) Если делится без остатка, то меньшее число и есть НОД (следует выйти из цикла).  
4) Большее число заменяем на остаток от деления  
Пример:  
Найти НОД для 30 и 18.  
30 / 18 = 1 (остаток 12)  
18 / 12 = 1 (остаток 6)  
12 / 6 = 2 (остаток 0)  
Конец: НОД – это делитель 6.  
НОД (30, 18) = 6  
  
**4. Средства описания алгоритмов: схема алгоритма. Привести пример.**Алгоритм моделирует решение задачи в виде точно определенной последовательности действий для некоторого исполнителя по преобразованию исходных данных в результирующие.  
Для его **описания** используется алгоритмический язык – набор символов и правил образования и истолкования конструкций из этих символов для записи алгоритмов.  
  
Основными изобразительными **средствами алгоритмов** являются следующие способы их записи:  
  
словесный;  
  
формульно-словесный;  
  
графический;  
  
псевдокод;  
  
табличный;  
  
операторный.  
**Графический способ** получил наибольшее распространение. В этом способе алгоритм представляется в виде схемы, состоящей из элементов символов, отражающих отдельные операции технологического процесса обработки данных и отдельные вычислительные операции.

Каждый этап процесса переработки данных представляется в виде геометрических фигур (блоков), имеющих определенную конфигурацию в зависимости от характера выполняемых операций  
**Пример:  
Пример алгоритма**: Алгоритм Евклида для нахождения НОД( наибольший общий делитель)  
while(a > 0 && b > 0)  
 if(a > b)  
 a %= b;  
 else  
 b %= a;  
 return a + b;   
  
1) Удостоверяемся, что оба числа больше нуля  
2) Большее число делим с остатком на меньшее   
3) Если делится без остатка, то меньшее число и есть НОД (следует выйти из цикла).  
4) Большее число заменяем на остаток от деления  
Пример:  
Найти НОД для 30 и 18.  
30 / 18 = 1 (остаток 12)  
18 / 12 = 1 (остаток 6)  
12 / 6 = 2 (остаток 0)  
Конец: НОД – это делитель 6.  
НОД (30, 18) = 6

**5.** **Средства описания алгоритмов: граф схема алгоритма. Привести пример.**Алгоритм моделирует решение задачи в виде точно определенной последовательности действий для некоторого исполнителя по преобразованию исходных данных в результирующие.  
Для его **описания** используется алгоритмический язык – набор символов и правил образования и истолкования конструкций из этих символов для записи алгоритмов.  
  
Основными изобразительными **средствами алгоритмов** являются следующие способы их записи:  
  
словесный;  
  
формульно-словесный;  
  
графический;  
  
псевдокод;  
  
табличный;  
  
операторный.  
  
**Граф-схема алгоритма**  **6. Численные методы реализации алгоритмов.  
7. Описание абстрактной машины Поста, система команд, пример**

**программы.  
Машина Поста** – это абстрактная (несуществующая реально) вычислительная машина, созданная для уточнения (формализации) понятия алгоритма. Представляет собой универсальный исполнитель, позволяющий вводить начальные данные и читать результат выполнения программы.  
**Машина Поста** состоит из …

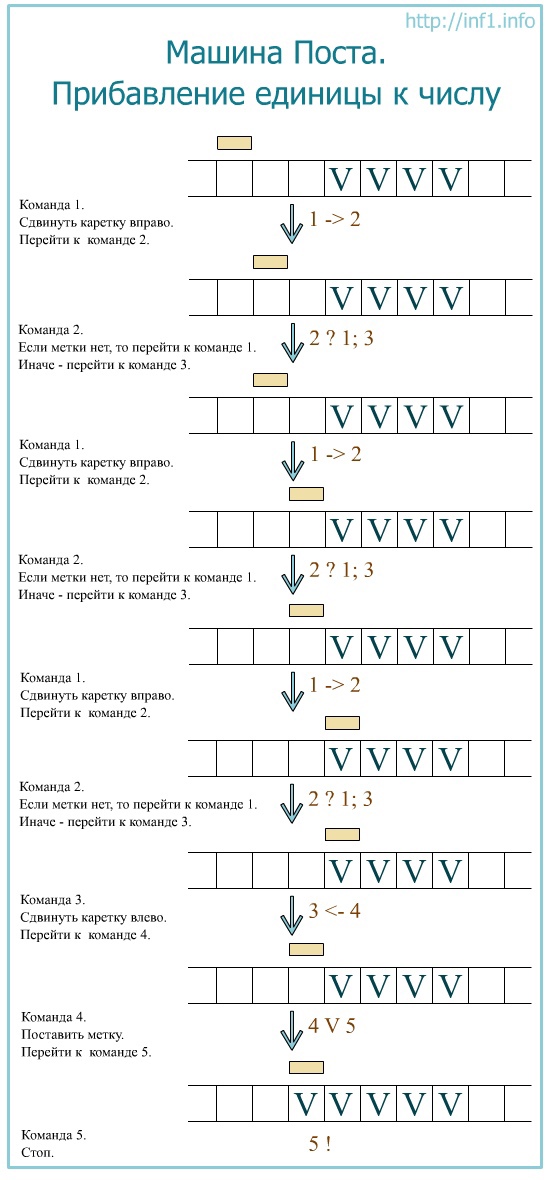
**1.бесконечной ленты**, поделенной на одинаковые ячейки (секции). Ячейка может быть пустой (0 или пустота) или содержать метку (1 или любой другой знак),

## 2.головки (каретки), способной передвигаться по ленте на одну ячейку в ту или иную сторону, а также способной проверять наличие метки, стирать и записывать метку. Текущее ****состояние машины Поста**** описывается состоянием ленты и положением каретки. ****Состояние ленты**** – информация о том, какие секции пусты, а какие отмечены. ****Шаг**** – это движение каретки на одну ячейку влево или вправо. Состояние ленты может изменяться в процессе выполнения программы. Всего для машины Поста существует шесть типов команд:

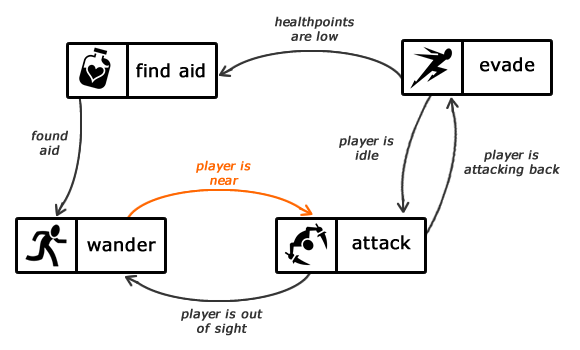
* **V j** - поставить метку, перейти к j-й строке программы.
* **X j** - стереть метку, перейти к j-й строке программы.
* **<- j** - сдвинуться влево, перейти к j-й строке программы.
* **-> j** - сдвинуться вправо, перейти к j-й строке программы.
* **? j1; j2** - если в ячейке нет метки, то перейти к j1-й строке программы, иначе перейти к j2-й строке программы.

## ! – конец программы (стоп). Пример работы машины Поста:

**Задача:** увеличить число 3 на единицу (изменить значение в памяти с 3 на 4).  
Целое положительное число на ленте машины Поста представимо идущими подряд метками, которых на одну больше, чем кодируемое число. Это связано с тем, что одна метка обозначает ноль, а уже две – единицу, и т.д.  
Допустим, точно известно, что каретка стоит где-то слева от меток и обозревает пустую ячейку. Тогда программа увеличения числа на единицу может выглядеть так:  
**1 -> 2  
2 ? 1;3  
3 <- 4  
4 V 5  
5 !**

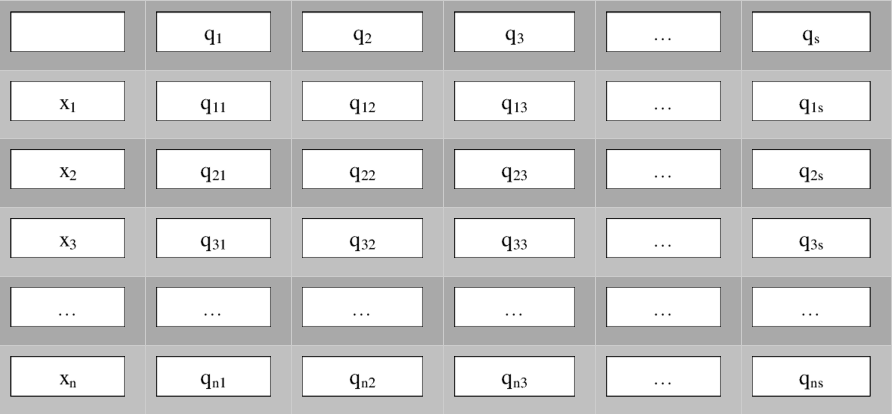
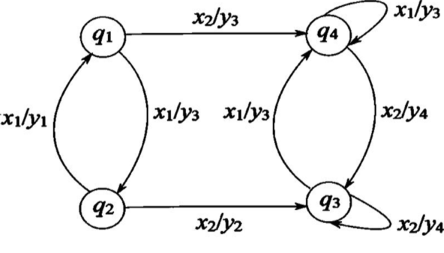
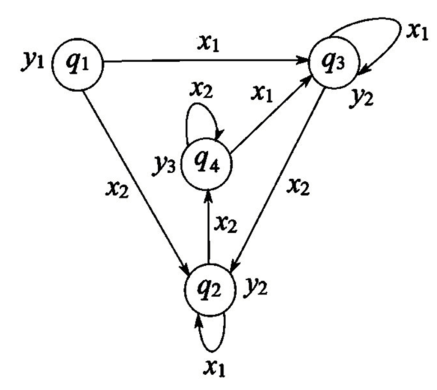
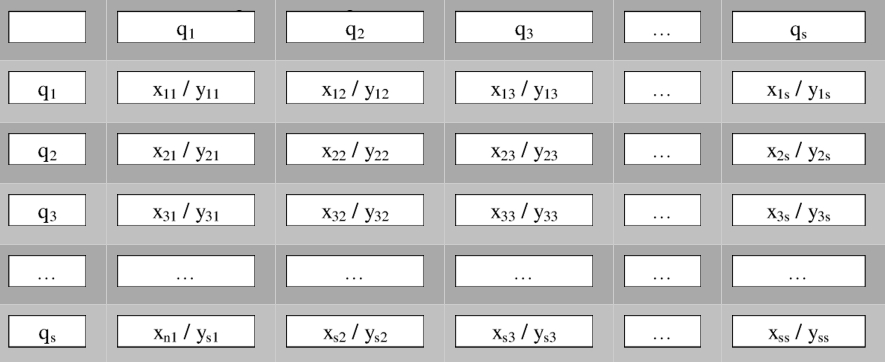
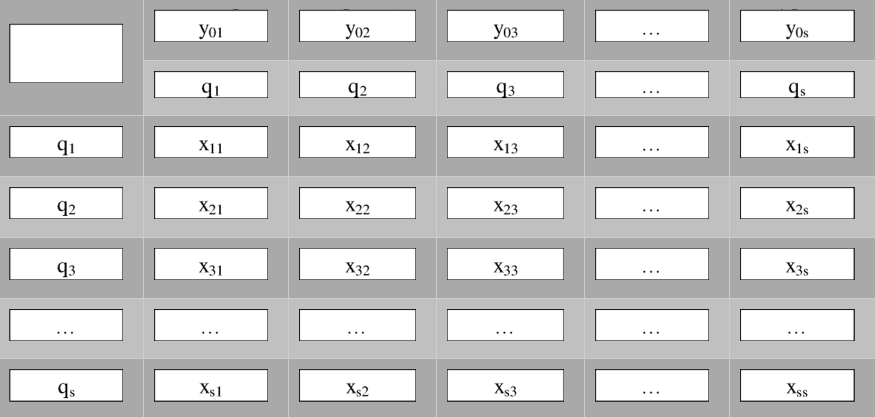


**8. Дайте определения конечного автомата и автономного автомата.  
Конечный автомат** — это некоторая абстрактная модель, содержащая конечное число состояний чего-либо. Используется для представления и управления потоком выполнения каких-либо команд. Конечный автомат идеально подходит для реализации искусственного интеллекта в играх, получая аккуратное решение без написания громоздкого и сложного кода.  
**Конечный автомат** (или попросту FSM — Finite-state machine) это модель вычислений, основанная на гипотетической машине состояний. В один момент времени только одно состояние может быть активным. Следовательно, для выполнения каких-либо действий машина должна менять свое состояние.

Конечные автоматы обычно используются для организации и представления потока выполнения чего-либо. Это особенно полезно при реализации ИИ в играх. Например, для написания «мозга» врага: каждое состояние представляет собой какое-то действие (напасть, уклониться и т. д.).  


**Автономный автомат -**

**9. Дайте определение конечного автомата с выходным преобразователем.  
Конечный автомат** — это некоторая абстрактная модель, содержащая конечное число состояний чего-либо. Используется для представления и управления потоком выполнения каких-либо команд. Конечный автомат идеально подходит для реализации искусственного интеллекта в играх, получая аккуратное решение без написания громоздкого и сложного кода.  
**Конечный автомат** (или попросту FSM — Finite-state machine) это модель вычислений, основанная на гипотетической машине состояний. В один момент времени только одно состояние может быть активным. Следовательно, для выполнения каких-либо действий машина должна менять свое состояние.

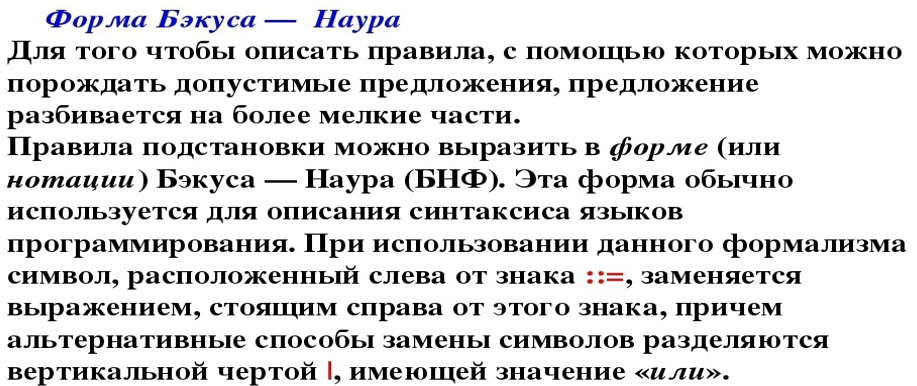
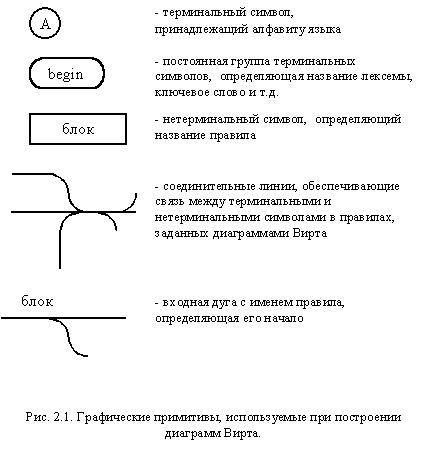
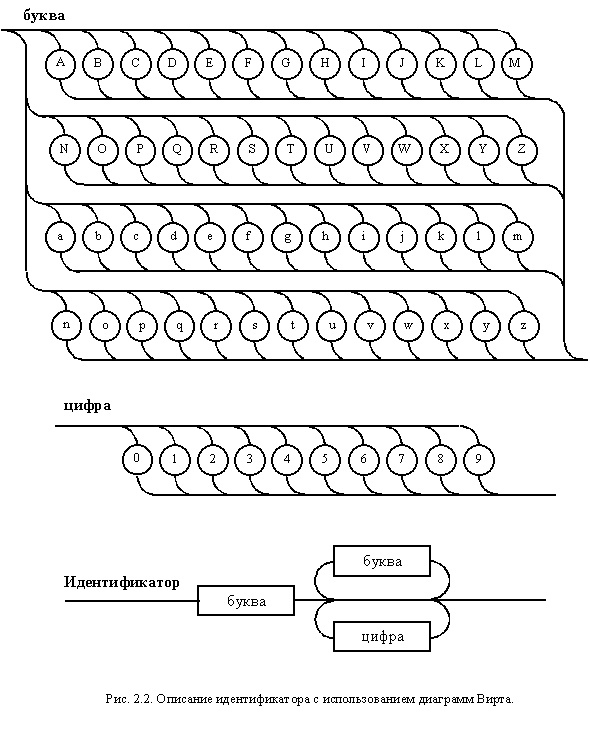
Конечные автоматы обычно используются для организации и представления потока выполнения чего-либо. Это особенно полезно при реализации ИИ в играх. Например, для написания «мозга» врага: каждое состояние представляет собой какое-то действие (напасть, уклониться и т. д.).  
  
**Конечный автомат с выходным преобразователем.**  
Для того, чтобы наделить модель абстрактного конечного автомата способностью не только давать ответ типа ДА/НЕТ, но и выполнять какие-то преобразования, в модель добавляют конечный алфавит выходных символов Z и функцию выхода ϕ . Если функция выхода имеет вид ϕ : X ×Y → Z , то вычислитель называется автоматом Мили, а если ϕ :Y → Z – автоматом Мура. Таким образом, рассматривается шестерка 0 X ,Y,Z,δ ,ϕ, y.  
  
(  
*ваще здесь не обязательно, но в сле*д *билетах пригодится:***Автомат Мура** (абстрактный автомат второго рода) в теории вычислений — конечный автомат, выходное значение сигнала в котором зависит лишь от текущего состояния данного автомата, и не зависит напрямую, в отличие от автомата Мили, от входных значений.  
**Автомат Мили** (англ. Mealy machine) — конечный автомат, выходная последовательность которого (в отличие от автомата Мура) зависит от состояния автомата и входных сигналов. Это означает, что в графе состояний каждому ребру соответствует некоторое значение (выходной символ). В вершины графа автомата Мили записываются выходящие сигналы, а дугам графа приписывают условие перехода из одного состояния в другое, а также входящие сигналы.   
)  
  
**10. Способ задания конечного автомата с помощью основной таблицы.**В основе табличного способа лежит построение таблиц, строки которых помечены символами входного алфавита, а столбцы - состояниями.  
Различают таблицы переходов и таблицы выходов.  
В таблице **переходов** на пересечении строки и столбца указывается соответствующее состояние, в которое переходит автомат в соответствии с функцией переходов 5   
Общий вид таблицы переходов для автомата Мили 5 :   
**Q X х-^ Q, qij = 5(qj, Xj)**В таблице **выходов** на пересечении строк и столбцов указывается выходной сигнал, получаемый как результат функции выходов X   
Общий вид таблицы выходов для автомата Мили:  
**  
  
  
11. Способ задания конечного автомата с помощью граф-схемы.  
  
Графический способ** задания автомата использует аппарат **графов**. Граф автомата представляет собой ориентированный граф, вершины которого соответствуют состояниям, а дуги - переходам между ними.  
  
Для графа автомата должно выполняться условие однозначности: из одной и той же вершины не должно выходить двух дуг с одинаковыми входными сигналами.  
  
Если вершина графа имеет только исходящие дуги, но не имеет входящих дуг, то соответствующее состояние автомата называется преходящим.  
  
Если вершина графа не имеет исходящих дуг, но имеет хотя бы одну входящую дугу, то соответствующее состояние автомата называется тупиковым.  
  
Если вершина графа не имеет ни входящих, ни исходящих дуг, то соответствующее состояние называется изолированным.  
  
Ниже в качестве примеров приведены графические и табличные представления автоматов Мили и Мура.  
  
Мили:  
  
  
Мура:  
  
  
**12. Способ задания конечного автомата с помощью с помощью матриц.**  
Еще одной возможностью табличного задания автомата является построение **матрицы** переходов и выходов.  
  
Для автомата Мили она строится следующим образом. Строки и столбцы отмечаются состояниями, а на их пересечениях размещаются соответствующие им входные и выходные сигналы. Если под действием входного сигнала ху происходит переход из состояния qs в состояние qj с выдачей выходного сигнала уу, то в ячейку на пересечении строки qs и столбца qj записывается ху / уу   
****  
  
Для матрицы переходов и выходов автомата Мура строки и столбцы также отмечаются состояниями, однако столбцы дополнительно помечаются соответствующими выходными сигналами, а на пересечении строк и столбцов указываются только входные сигналы   
  
  
Общий вид матрицы переходов и выходов для автомата Мура  
****  
  
**13. Метасинтаксический язык Бэкуса.  
  
Форма Бэкуса — Наура** (сокр. БНФ, Бэкуса — Наура форма) — формальная система описания синтаксиса, в которой одни синтаксические категории последовательно определяются через другие категории. БНФ используется для описания контекстно-свободных формальных грамматик. Существует расширенная форма Бэкуса — Наура, отличающаяся лишь более ёмкими конструкциями.  
  
БНФ-конструкция определяет конечное число символов ([нетерминалов](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D1%82%D0%B5%D1%80%D0%BC%D0%B8%D0%BD%D0%B0%D0%BB" \o "Нетерминал)). Кроме того, она определяет правила замены символа на какую-то последовательность букв (терминалов) и символов. Процесс получения цепочки букв можно определить поэтапно: изначально имеется один символ (символы обычно заключаются в угловые скобки, а их название не несёт никакой информации). Затем этот символ заменяется на некоторую последовательность букв и символов, согласно одному из правил. Затем процесс повторяется (на каждом шаге один из символов заменяется на последовательность, согласно правилу). В конце концов, получается цепочка, состоящая из букв и не содержащая символов. Это означает, что полученная цепочка может быть выведена из начального символа.

БНФ-конструкция состоит из нескольких предложений вида

<определяемый символ> ::= <посл.1> | <посл.2> | . . . | <посл.n>

, описывающих правила. Такое правило означает, что символ <определяемый символ> может заменяться на одну из последовательностей <посл.n>. Знак определения обычно выглядит как ::= или →, но возможны и другие варианты.

Некоторые специальные символы, как например <пусто>, означают какую-то последовательность (в данном случае — пустую).

* **14. Нотации Бэкуса-Наура.**Расширенная форма Бэкуса — Наура (расширенная Бэкус — Наурова форма (РБНФ)) (англ. Extended Backus–Naur Form (EBNF)) — формальная система определения синтаксиса, в которой одни синтаксические категории последовательно определяются через другие. Используется для описания контекстно-свободных формальных грамматик. Предложена Никлаусом Виртом. Является расширенной переработкой форм Бэкуса — Наура, отличается от БНФ более «ёмкими» конструкциями, позволяющими при той же выразительной способности упростить и сократить в объёме описание.   
  ****  
  **15. Диаграммы Вирта.**  
  Терминальные символы и их постоянные группы располагаются в окружностях или прямоугольниках со скругленным вертикальными сторонами;  
    
  нетерминальные символы заносятся внутрь прямоугольников;  
    
  каждый графический элемент, соответствующий терминалу или нетерминалу, имеет по одному входу и выходу, которые обычно рисуются на противоположных сторонах;  
    
  каждому правилу соответствует своя графическая диаграмма, на которой терминалы и нетерминалы соединяются посредством дуг;  
    
  альтернативы в правилах задаются ветвлением дуг, а итерации – их слиянием;  
    
  должна быть одна входная дуга (располагается обычно слева и сверху), задающая начало правила и помеченная именем определяемого нетерминала, и одна выходная, задающая его конец (обычно располагается справа и снизу).  
    
    
    
  Описание идентификатора с использованием **диаграмм Вирта** **16. Подходы к построению алгоритмических языков: Основные этапы.  
    
    
    
    
    
  17. Свойства хорошего языка программирования.**Понятность (удобочитаемость) конструкций языка– это свойство, обеспечивающее легкость восприятия программ человеком [15]. Это свойство языка зависит от целого ряда факторов, начиная с выбора ключевых слов и заканчивая возможностью построения модульных программ. Понятность конструкций языка также зависит от выбора такой нотации, которая позволяла бы при чтении текста программы легко выделять основные понятия каждой конкретной части программы, не обращаясь ни к какой другой документации на программу. Реализация требования понятности во многом зависит от программиста.  
    
  Надежность– степень автоматического обнаружения ошибок, которое может быть выполнено транслятором или операционной средой, в которой работает программа. Надежный язык позволяет выявить большинство ошибок во время трансляции программы, а не во время ее выполнения. Принципиальным средством достижения высокой надежности языка, поддерживаемым на этапе трансляции, является система типизации данных.  
    
  Гибкость– сколько возможностей язык предоставляет программисту для выражения всех операций, которые требуются в программе, не прибегая к вставкам ассемблерного кода или другим ухищрениям. Требование гибкости конфликтует с требованием надежности, поэтому необходимо понимать, на какие компромиссы приходится идти при решении каждой конкретной задачи.  
    
    
  Простота – легкость понимания семантики языковых конструкций и запоминания их синтаксиса. Простой язык предоставляет ясный, простой и единообразный набор понятий, которые могут быть использованы в качестве базовых при разработке алгоритмов. При этом желательно иметь минимальное количество различных понятий с как можно более простыми и систематизированными правилами их комбинирования – язык должен обладать свойствомконцептуальной целостности. Концептуальная целостность включает в себя три взаимосвязанных аспекта: экономию, ортогональность и единообразие понятий. Экономия – использование минимального числа понятий. Ортогональность – между понятиями нет взаимного влияния (любые языковые конструкции можно комбинировать по определенным правилам). Единообразие понятий – согласованный, единый подход к описанию и использованию всех понятий.  
    
  Естественность – язык должен содержать такие структуры данных, управляющие структуры и операции, а также иметь такой синтаксис, которые позволяли бы отражать в программе логические структуры, лежащие в основе реализуемого алгоритма.  
    
  Мобильность – возможность переносить программы с одной платформы на другую с относительной легкостью. На мобильность в значительной степени влияет уровень стандартизации языка. Для языков, имеющих стандартное определение (Ada,Pascal,C) все реализации языка должны основываться на этом стандарте.  
    
  Стоимость – суммарная стоимость использования языка программирования складывается из нескольких составляющих: стоимости обучения языку, стоимости создания программы, стоимости трансляции программы, стоимости выполнения программы, стоимости сопровождения программы.
* **18. Синтаксические элементы языка.**1)наборы символов. Определение набора символов -это одна из первоочередных задач при разработке синтаксиса языка
* 2)Идентификатор различия правил составления идентификаторов в языках сводится к включению в допустимый набор букв и цифр спецификация символов длина\читаемость\стандарты языка
* 3) символы операций
* 4) ключевые лова зарезервированные слова, которые используются в качестве фиксированной части синтаксиса какого-либо оператора
* 5) комментарии являются важной частью документирования программы
* 6) разделители и скобки разделители -это синтаксический элемент функция которого обозначать начало и конец некоторой синтаксической конструкции
* 7)операторы важные синтаксические компоненты от них зависит удобство записи и чтения программы
* 8) выражения — это функции которые обрабатывают какие-либо данные в программе и возвращают некоторые значения

**19. Организация выполнения программ.  
  
20. Использование формальных языков для поиска.**

**21. Дайте определение понятиям: Базы данных и Банки данных.   
  
База данных (БД)** – это упорядоченное хранение информационных ресурсов в виде объединённых структурированных данных, обеспечивающих быстрый доступ и удобное рациональное взаимодействие между данными.  
  
  
  
**Банк данных (БнД)** – это автоматизированная система, обеспечивающая хранение, накопление, поиск и выдачу информации в совокупности программных и технических средств. Основными элементами банка данных являются база данных и программно-информационные продукты, называемые системой управления базой данных (СУБД).

**22. Основные требования к Базам данных.**К современным базам данных предъявляются следующие основные требования:  
  
Высокое быстродействие (малое время отклика на запрос). Время отклика - промежуток времени от момента запроса к БД до фактического получения данных. Похожим является термин время доступа - промежуток времени между выдачей команды записи (считывания) и фактическим получением данных. Под доступом понимается операция поиска, чтения данных или записи их. Часто операции записи, удаления и модификации данных называют операциями обновления.  
  
Простота обновления данных. Важнейшими являются эти первые два противоречивых требования: повышение быстродействия требует упрощения структуры БД, что, в свою очередь, затрудняет процедуру обновления данных, увеличивает их избыточность  
  
Независимость данных - возможность изменения логической и физической структуры БД без изменения представлений пользователей. Независимость данных предполагает инвариантность к характеру хранения данных, программному обеспечению и техническим средствам. Она обеспечивает минимальные изменения структуры БД при изменениях стратегии доступа к данным и структуры самих исходных данных. Это достигается, как будет показано далее, «смещением» всех изменений на этапы концептуального и логического проектирования с минимальными изменениями на этапе физического проектирования  
  
Совместное использование данных многими пользователями.  
  
Безопасность данных - защита данных от преднамеренного или непреднамеренного нарушения секретности, искажения или разрушения. Безопасность данных включает их целостность и защиту. Целостность данных - устойчивость хранимых данных к разрушению и уничтожению, связанных с неисправностями технических средств, системными ошибками и ошибочными действиями пользователей. Целостность данных предполагает:  
  
1) отсутствие неточно введенных данных или двух одинаковых записей об одном и том же факте;  
  
2) защиту от ошибок при обновлении БД;  
  
3) невозможность удаления (или каскадное удаление) связанных данных разных таблиц;  
  
4) неискажение данных при работе в многопользовательском режиме и в распределенных базах данных;  
  
5) сохранность данных при сбоях техники (восстановление данных).  
  
Для обеспечения целостности БД накладывают ограничения целостности в виде некоторых условий, которым должны удовлетворять хранимые в базе данные (например, диапазон значений атрибутов и др.)

* **23. Этапы проектирования БД: концептуальная модель.**Традиционно процедуру проектирования базы данных разбивают на три этапа, каждый из которых завершается созданием соответствующей информационной модели.  
  Этап 1-й. Концептуальное проектирование – создание представления (схемы, модели) БД, включающего определение важнейших сущностей (таблиц) и связей между ними, но не зависящего от модели БД (иерархической, сетевой, реляционной и т. д.) и физической реализации (целевой СУБД).  
  Этап 2-й. Логическое проектирование – развитие концептуального представления БД с учетом принимаемой модели (иерархической, сетевой, реляционной и т.д.).  
    
  Этап 3-й. Физическое проектирование – развитие логической модели БД с учетом выбранной целевой СУБД.  
  Цель концептуального проектирования – создание концептуальной модели данных на основе представлений о предметной области каждого отдельного типа пользователей. Концептуальная модель представляет собой описание основных сущностей (таблиц) и связей между ними без учета принятой модели БД и синтаксиса целевой СУБД. Часто на такой модели отображаются только имена сущностей (таблиц) без указания их атрибутов. Представление пользователя включает в себя данные, необходимые конкретному пользователю для принятия решений или выполнения некоторого задания.  
  Ниже рассматривается последовательность шагов при концептуальном проектировании.  
  1. Выделение сущностей.  
  2. Определение атрибутов.  
  3. Определение связей.  
  4. Определение суперклассов и подклассов.
* 24)ER-модель представляет собой формальную конструкцию, которая сама по себе не предписывает никаких графических средств её визуализации. В качестве стандартной графической нотации, с помощью которой можно визуализировать ER-модель, была предложена диаграмма сущность-связь (ER-диаграмма) (англ. entity-relationship diagram, ERD).  
  ER-модель используется при высокоуровневом (концептуальном) проектировании баз данных. С её помощью можно выделить ключевые сущности и обозначить связи, которые могут устанавливаться между этими сущностями.  
    
  Во время проектирования баз данных происходит преобразование ER-модели в конкретную схему базы данных на основе выбранной модели данных (реляционной, объектной, сетевой или др.).

**24. Модель “Сущность-Связь” (ER-диаграмма).**ER-модель представляет собой формальную конструкцию, которая сама по себе не предписывает никаких графических средств её визуализации. В качестве стандартной графической нотации, с помощью которой можно визуализировать ER-модель, была предложена диаграмма сущность-связь (ER-диаграмма) (англ. entity-relationship diagram, ERD).  
ER-модель используется при высокоуровневом (концептуальном) проектировании баз данных. С её помощью можно выделить ключевые сущности и обозначить связи, которые могут устанавливаться между этими сущностями.  
  
Во время проектирования баз данных происходит преобразование ER-модели в конкретную схему базы данных на основе выбранной модели данных (реляционной, объектной, сетевой или др.).