1. Предмет информатики

Информатика – наука об автоматической обработке информации с помощью ЭВМ.

Информатика – наука о существовании преимущественно автоматических средств целесообразной обработки информации, рассматриваемой как представление знаний и сообщений в технической, экономической и социальной областях.

Предметом информатики является структура, свойства и закономерности информации, процесс ее сбора, переработки, хранения, поиска, распространения и использования.

Методы работы над информацией:

- Аппаратное обеспечение средств вычислительной техники
- Программное обеспечение средств вычислительной техники
- Взаимодействие аппаратного и программного обеспечения
- Взаимодействие человека с аппаратным и программным обеспечением

2. Информация и сообщения. Интерпретация сообщений

Информация и сообщения – основные неопределяемые понятия информатики. Использование этих понятий можно разъяснить на примерах.

Информация может существовать и передается только в форме некоторого сообщения. Сообщение является материальным носителем информации.

Соответствие между информацией и несущим ее сообщением не является взаимно-однозначным:

1. одна и та же информация может передаваться с помощью различных сообщений (например, лекция может быть прочитана на разных языках)
2. одно и то же сообщение может передавать различную информацию (например, продавец мясного отдела гастронома кричит кассирше: “Перестань выбивать мозги!”), в таких случаях все зависит от **интерпретации сообщений**.

Интерпретация сообщений: Информация i, которая передается сообщением n, устанавливается с помощью правила интерпретации, которое представляет собой отображение рассматриваемого множества сообщений в множество сведений:

$$\varphi : N \to I$$

Множество сообщений N представляет собой **язык** (каждое сообщение n является текстом языка N).

Конечные языки – языки, содержащие конечное количество текстов. Для задания конечного языка перечисляют все его тексты.

Бесконечные языки – языки, содержащие бесконечное множество текстов. Для задания бесконечного языка вводят грамматику языка.

3. Знаки и символы

Каждое сообщение передается на каком-либо материальном **носителе сообщения**. Сообщения, представленные на долговременных носителях, называются **письменными**.

Письменные языковые сообщения представляют собой последовательности знаков. Знаки бывают **атомарные** и **составные**.

- **атомарным знаком** или **буквой** называется элемент какого-либо упорядоченного конечного непустого множества графически (или еще как-либо) отличимых друг от друга литер (предметов, сущностей, объектов, членов), называемого **алфавитом**.
- **составным знаком** или **словом** называется конечная последовательность знаков (атомарных или составных).

Если в состав последовательности входят составные знаки, то необходимо иметь **знак-разделитель** – специальный знак, который ставится между составными знаками.

С каждым знаком связывается его **смысл** или **семантика**. Знак вместе с сопоставленной ему семантикой называется **символом**.

4. Кодирование

Код – правило, описывающее сюръективное отображение $C : A \to A'$ набора знаков A на набор знаков A'.

При кодировании сообщения n получается новое сообщение $C(n)$, которое содержит ту же информацию, что и n.

Если до кодирования для интерпретации использовалось отображение φ_0, то после кодирования сообщения будут интерпретироваться с помощью отображения $\varphi_1 = C^{-1} \circ \varphi_0$, где C^{-1} – отображение, обратное C (отображение **декодирования**).

Если известно отображения C^{-1} (или C), или известен способ построения одного из этих отображений, то говорят, что известен **ключ** кода C.

Кодирование, при котором каждый образ является отдельным знаком, называется **шифрованием**.

Современная теория чисел дает способ создать систему шифрования, в которой используются 2 ключа – один для кодирования, второй для декодирования (**ассиметричное шифрование**).

Одностороннее шифрование используется для хранения паролей Unix и строго говоря кодированием не является. В этом случае используется неоднозначная (например, периодическая) функция. Для каждого шифрованного пароля получается множество вариантов дешифрованных, что затрудняет проникновение в систему.

Утверждение. Произвольный конечный набор знаков (алфавит) может быть закодирован знаками набора $\omega = (b_0, b_1)$.

Доказательство. Знаки исходного набора могут быть закодированы знаками стандартного алфавита ω_m, где m – число знаков в исходном наборе. Пусть $b_i \in \omega_m$. Сопоставим ему код $b_0 b_1 \dots b_1$ ($i+1$ раз b_1) или $b_i \to b_0 b_1^{i+1}$. Утверждение доказано.

5. Системы счисления

Система счисления – способ числовой интерпретации цифровых сообщений.

Способ кодирования $b_i \to b_0 b_1^{i+1}$ может быть положен в основу **натуральной системы счисления**. В этом случае элементам бесконечного счетного множества $\mathbb{N}$ ставятся в соответствие слова в ω^*.

Для представления нуля в натуральной СС можно слегка изменить способ кодирования ($| \to 0; || \to 1; \dots$). Такую систему счисления называют **кардинальной**.

Позиционная система счисления – это полиноминальный способ числовой интерпретации слов над цифровым алфавитом. Позиционная СС задается одним натуральным числом $p \in \mathbb{N}$ – **основанием системы счисления** – которое однозначно определяет алфавит $A_p = \{0, 1, \dots, p-1\}$ и функцией интерпретации

$$x = \overline{a_n a_{n-1} \dots a_1 a_0} \xrightarrow{\varphi_p} \sum_{i=0}^{n} \varphi(a_i) p^i = \sum_{j=0}^{m} \varphi(b_j) q^j \xleftarrow{\varphi_q} \overline{b_m b_{m-1} \dots b_1 b_0} = x$$

6. Обработка сообщений

Обработка сообщений определяется правилом обработки ν, которое представляет собой отображение $\nu : N \to N'$, где N – множество исходных сообщений, а N' – множество сообщений, получающихся в результате обработки.

Приведённые примеры показывают, что каждая обработка сообщений состоит в выделении в исходном сообщении знаков некоторого уровня, составляющих это сообщение, и замене каждого выделенного знака другим знаком.

Таким образом, любая обработка сообщений может рассматриваться как кодирование в широком смысле. Это соображение лежит в основе всякой машинной обработки **дискретных сообщений**.

Зависимость от времени приводит к понятию **эффективности правила обработки** сообщений, которая измеряется скоростью процесса обработки по сравнению со скоростями других процессов.

7. Обработка информации

Если отображение φ обратимо, т.е. $\exists\varphi^{-1}$ (достаточно, чтобы φ было инъективным отображением), то можно построить отображение σ, определяющее **обработку информации** $\sigma : I \to I'$ в виде $\sigma = \varphi' \circ \nu \circ \varphi^{-1}$ так, что $i' = \varphi'(\nu(\varphi^{-1}(i)))$.

Во всех случаях, когда правило обработки σ является отображением, правило обработки сообщений ν называется **сохраняющим информацию**.

$$\begin{array}{ccccc} & N & \xrightarrow{\varphi} & I & \\ \nu & \downarrow & & \downarrow & \sigma \\ & N' & \xrightarrow{\varphi'} & I' & \end{array}$$

Если ν сохраняет информацию, то диаграмма коммутативна: $\nu \circ \varphi' = \varphi \circ \sigma$. В этом случае отображение σ называется **правилом обработки информации**.

Если σ – обратимое отображение, т.е. информация при обработке не теряется, то соответствующая обработка называется **перешифровкой**.

Пусть ν – обратимая перешифровка. Тогда обработка обратимой перешифровкой называется **перекодировкой**.

Перешифровка ν, не являющаяся обратимой, называется **сжимающей**. В результате необратимой перешифровки сообщений их кол-во уменьшается, а информация может либо сохраняться, либо теряться.

8. Автоматизация обработки информации

Для автоматизации обработки сообщений необходимо иметь физическое устройство, работа которого "моделировала" бы выполнение отображения ν. Для этого необходимо располагать **тремя физическими представлениями**:

1. Физическим представлением для множества исходных сообщений N – его мы будем обозначать буквой D и называть **множеством исходных данных**.
2. Физическим представлением для множества результирующих сообщений N' – его мы будем обозначать буквой D' и называть **множеством результирующих данных** (результатов).
3. Физическим представлением правила обработки ν, которое должно быть преобразованием, или последовательностью преобразований, оперирующих над D и дающих в результате элемент D', – его мы будем обозначать буквой P; преобразование P должно быть физически реализуемым, т.е. выполняться на некотором физическом устройстве.

Таким образом, для того, чтобы выполнить автоматическую обработку сообщений, необходимо автоматизировать выполнение трех отображений:

1. Отображения кодирования C, которое позволяет представить (абстрактное) сообщение из N с помощью элемента множества данных D, т.е., с помощью конкретного состояния некоторого физического устройства.
2. Отображения P, которое переводит элемент D в элемент D' – конкретное состояние ещё одного физического устройства.
3. Отображения декодирования Q, которое позволяет проинтерпретировать результат – конкретное состояние физического устройства, представляющее элемент из D', переведя его в соответствующее сообщение из множества N'

Таким образом, автоматическая обработка информации может быть представлена диаграммой:

$$\begin{array}{ccccccc} & D & \xleftarrow{C} & N & \xleftarrow{\varphi} & I & \\ P & \downarrow & & \downarrow \nu & & \downarrow & \sigma \\ & D' & \xrightarrow{Q} & N' & \xrightarrow{\varphi'} & I' & \end{array}$$

Из диаграммы следует, что $\nu = C \circ P \circ Q$, причем все 3 отображения C, P и Q должны выполняться автоматом.

9. Конструктивное описание процесса обработки дискретных сообщений

Чтобы отображение P могло служить основой для автоматизации обработки сообщений, оно должно задавать некоторый **способ построения** сообщения $d' = P(d)$, исходя из сообщения $d \in D$. Если D – конечное множество, то отображение P может быть задано таблицей, в которой перечисляются все сообщения $d \in D$ и соответствующие им сообщения $P(d)$. Если же D бесконечно или так велико, что задание P с помощью таблицы оказывается непрактичным, то нужно представить P в виде последовательности элементарных **шагов обработки** (**тактов**), каждый из которых состоит в выполнении одного или нескольких достаточно простых отображений, называемых **операциями**: $P = P_1 \circ P_2 \circ \cdots \circ P_k$. Подобная обработка сообщений называется **эффективной процедурой** или **алгоритмом**. Алгоритм – это точно заданная последовательность правил, указывающая, каким образом можно за конечное число шагов получить выходное сообщение определенного вида, используя заданное исходное сообщение.

10. Свойства алгоритмов

1. **Массовость**, т.е. потенциальная бесконечность исходных сообщений, предназначенных для обработки алгоритмом.
2. **Детерминированность** процесса применения алгоритма, заключающегося в последовательном выполнении дискретных действий (шагов), однозначно определяемых результатами каждого предыдущего шага.
3. **Элементарность каждого шага** обработки, который должен быть достаточно легко выполним.
4. **Результативность** алгоритма, т.е. точное описание того, что считается результатом применения алгоритма после выполнения всех требуемых шагов.
5. **Сложность** алгоритма определяется количеством простых операций, которые необходимо выполнить для обработки сообщения. Это количество прямо зависит от длины входного сообщения n и обычно некоторой функцией $f(n)$. Таким образом, сложность ограничивает применимость алгоритма.

11. Сложность алгоритмов

Сложность алгоритма определяет, как долго придётся ждать, пока будет получен результат его работы. Одновременно сложность определяет размер задачи, которая может быть решена данным алгоритмом за приемлемое время.

Кроме обычной временной сложности алгоритма часто рассматривают и пространственную сложность – объем памяти, необходимый алгоритму для переработки входного сообщения длины n.

Для обозначения класса сложности используется O-нотация. У всякого алгоритма, как у функции, есть главная часть, которая занимает больше всего времени для выполнения. Анализируя количество выполнений главной части, делают вывод о поведении алгоритма в зависимости от длины исходного сообщения.

Все алгоритмы можно разбить на классы по их сложности:

Класс сложности	**Описание**
$O(1)$	Алгоритмы с постоянным временем выполнения, например, доступ к элементу массива. При увеличении размера задачи в n раз время выполнения не меняется.
$O(\log n)$	Алгоритмы с логарифмическим временем выполнения, например, бинарный поиск. Время выполнения удваивается при увеличении размера задачи в n раз.
$O(n)$	Алгоритмы с линейным временем выполнения, например, последовательный поиск или схема Горнера. При увеличении размера задачи в n раз время выполнения также увеличивается в n раз.
$O(n \log n)$	Алгоритмы с линеарифмическим временем выполнения, например, быстрая сортировка. Время выполнения увеличивается немного больше, чем в n раз при увеличении размера задачи в n раз.

$O(n^2)$	Алгоритмы с квадратичным временем выполнения, например, простые алгоритмы сортировки. Время выполнения увеличивается в 4 раза при увеличении размера задачи в 2 раза.
$O(n^3)$	Алгоритмы с кубическим временем выполнения, например, умножение матриц. Время выполнения увеличивается в 8 раз при увеличении размера задачи в 2 раза.
$O(2^n)$	Алгоритмы с экспоненциальным временем выполнения, например задача о составлении расписания. Время выполнения увеличивается в 2^n раз при увеличении размера задачи в 2 раза.

12. Семиотические модели интерпретации дискретных сообщений

Назовём k-местным отношением на множестве m совокупность (множество) r^k упорядоченных наборов из k элементов множества M вида $\langle x_1, x_2, \ldots, x_k \rangle$ (иначе говоря, отношение r – это подмножество декартова произведения $r^k \subseteq M^k$); верхний индекс у r указывает число мест отношения.

Моделью (или реляционной системой) называется некоторое множество $\mathcal{M}$ с заданным на нем набором отношений $\{r_1^{k_1}, r_2^{k_2}, \ldots, r_n^{k_n}\}$. Иначе говоря, модель $\mathcal{M}$ – это упорядоченная пара объектов $\mathcal{M} = \langle M, \{r_1^{k_1}, r_2^{k_2}, \ldots, r_n^{k_n}\}\rangle$, где $r_i^{k_i}$ – отношения на множестве M.

Для того, чтобы указать, что же моделируется, введем понятия **теории** и **сигнатуры**.

Сигнатура – это перечень знаков (имён) отношений.

Теория – это упорядоченная пара $T = (\Omega, A)$, где Ω – сигнатура, а A – множество **аксиом**, высказываний о свойствах сигнатуры Ω (обычно в качестве аксиом используются формулы специального вида).

Модель $\mathcal{M}$ называется моделью теории T, если $\mathcal{M}$ и T имеют одинаковую сигнатуру и если после интерпретации φ каждого имени отношения теории, как одноименного отношения в модели, каждая аксиома становится истинным высказыванием.

Из сказанного следует, что любая задача обработки информации, которую мы хотим автоматизировать, должна быть поставлена на соответствующей модели. Прежде, чем разрабатывать алгоритм решения задачи обработки сообщений, необходимо **формализовать** задачу, т.е. четко описать модель, на которой ставится задача, и попытаться сформулировать теорию для этой модели.

13. Необходимость формального определения алгоритма

Основным недостатком неформального определения алгоритма является его расплывчатость.

Не все математические задачи алгоритмически разрешимы, а доказательство алгоритмической неразрешимости неизбежно содержит высказывание о всех мыслимых алгоритмах. Такие высказывания невозможны без строгого формального определения алгоритма.

Формализация реализуется с помощью построения **алгоритмических моделей**, среди которых выделяется 3 основных типа:

1. **Рекурсивные функции** (понятие алгоритма связывается с вычислениями и числовыми функциями).
2. **Машины Тьюринга** (алгоритм представляется как описание процесса работы некоторой машины, способной выполнять лишь небольшое число весьма простых операций).
3. **Нормальные алгоритмы Маркова** (алгоритмы описываются как преобразования слов в произвольных алфавитах).

Тезис Тьюринга-Чёрча. Всякий интуитивный алгоритм может быть выражен средствами одной из алгоритмических моделей.

14. Машины Тьюринга

Машина Тьюринга состоит из ограниченной с одного конца бесконечной **ленты** и комбинированной читающей и пишущей **головки**, которая может перемещаться вдоль ленты от ячейки к ячейке.

В каждой ячейке ленты может быть записан один знак алфавита A, называемого **рабочим алфавитом** МТ, либо пробел (λ).

Головка МТ в каждый момент времени располагается над одной из ячеек ленты, называемой **рабочей ячейкой**, и воспринимает знак, записанный в этой ячейке. При этом головка находится в одном из конечного множества $Q = \{q_0, q_1, \ldots, q_s\}$ дискретных **состояний**, среди которых выделено одно **начальное состояние** q_0. В зависимости от состояния, в котором находится головка и от буквы, записанной в рабочей ячейке, МТ выполняет одну из **команд**, составляющих ее программу.

Перед началом работы МТ на её ленту записывается исходное сообщение так, что в каждую ячейку ленты записывается одна буква сообщения, либо пробел. Любое сообщение занимает конечное число ячеек ленты. При этом ячейки, расположенные справа от последней буквы исходного сообщения, заполняются пробелами и считаются пустыми.

Каждая команда МТ описывается упорядоченной четверкой символов (q, a, v, q'), где:

- $q \in Q$ – символ текущего состояния головки;
- $a \in A \cup \{\lambda\}$ – буква, записанная в рабочей ячейке;
- $v \in \{l, r\} \cup A \cup \lambda$ – символ выполняемого действия, ($v = l$ означает сдвиг головки влево, $v = r$ – сдвиг головки вправо, а $v \in A \cup \{\lambda\}$ – запись соответствующего знака в рабочую ячейку).
- $q' \in Q$ – символ состояния, в которое команда переводит головку.

Ситуацией (или **состоянием ленты**) называется упорядоченная пара объектов $S = (z, k)$, где S – имя ситуации, z – сообщение, записанное на ленте, k – неотрицательное целое число, равное расстоянию (в ячейках) от края ленты до рабочей ячейки.

Конфигурацией называется упорядоченная пара объектов $C = (S, q)$, где S – текущая ситуация, q – текущее состояние головки.

Ситуацию (конфигурацию), соответствующую начальному состоянию МТ, будем называть **начальной ситуацией** (конфигурацией).

15. Нормальные алгоритмы Маркова

Нормальные алгоритмы Маркова по существу являются **детерминистическими** текстовыми заменами, которые для каждого входного слова однозначно задают вычисления и тем самым в случае их завершения порождают определенный результат.

Марковская стратегия правил заключается в следующем:

1. Если применимо несколько правил, то берется правило, которое встречается в алгоритме первым.
2. Если правило применимо в нескольких местах обрабатываемого слова, то выбирается самое левое из этих мест.

Существует два простых достаточных признака применимости НАМ ко всем входным словам:

1. Левые части всех продукций непустые, а в правых частях нет букв, входящих в левые части.
2. В каждом правиле правая часть короче левой.

16. Диаграммы машин Тьюринга

Диаграммы Тьюринга представляют одни МТ через другие, более простые МТ визуально-топологическим способом, причём, как будет показано далее, этот способ не менее строг и полон, чем обычные МТ.

Рассматриваемая МТ будет описана через **элементарные** МТ, т.е. такие, которые уже нельзя описать через более простые МТ, так как каждая из них выполняет всего одно элементарное действие и останавливается.

Элементарные МТ над алфавитом $A_p = (a_1, a_2, \ldots, a_p)$ определяются следующими программами.

1. Элементарная МТ l (сдвиг головки на одну ячейку влево).
2. Элементарная МТ r (сдвиг головки на одну ячейку вправо).
3. Элементарные МТ λ $(a_1, \ldots, a_p)$ (запись соответствующего знака на ленту).

Комбинируя элементарные МТ, можно получить более сложные МТ. Например, применяя последовательно элементарную МТ l (или r) до тех пор, пока в рабочей ячейке не встретится знак λ, получим МТ, сдвигающую головку до левого (правого) конца слова. Построенные МТ будем называть L и R.

Диаграмма Тьюринга рисуется на листе бумаги слева направо и сверху вниз. Состояния будем обозначать точками ($\bullet$). Каждому символу МТ соответствуют 2 состояния: начальное и конечное, поэтому символ должен быть изображен на диаграмме между двумя точками. Если после действия, определяемого машиной M_1, должно быть выполнено действие, определяемое машиной M_2, то правая точка машины M_1 (заключительное состояние), соединяется с левой точкой машины M_2 (начальное состояние), причем над стрелкой надписываются те знаки рабочего алфавита A_p, которые вместе с заключительным состоянием машины M_1 определяют переход к действию, которая задается машиной M_2.

17. Моделирование Машин Тьюринга

Рассмотрим две МТ $T = (A, Q, P, q_0)$ и $T' = (A', Q', P', q'_0)$. $T' \simeq T \Leftrightarrow$ машина T' моделирует машину T, если выполняются следующие условия:

1. Указан способ кодирования знаков (букв) алфавита A знаками (буквами или словами) алфавита $A'C : A \to A'$.
2. Каждому состоянию $q \in Q$ машины T поставлено в соответствие некоторое состояние $q' \in Q'$ машины T', т.е. определено отображение $Q \to Q'$.
3. Если C_0 – начальная конфигурация машины T, то ее образ C'_0 – начальная конфигурация машины T'.
4. Если машина T из начальной конфигурации C_0 после конечного числа тактов останавливается в конфигурации C_k, то машина T' из начальной конфигурации C'_0, являющейся образом C_0, после конечного числа тактов также останавливается в конфигурации C'_k, являющейся образом C_k.
5. Если из начальной конфигурации C_0 машина T пробегает (конечную или бесконечную) последовательность конфигураций $C_0, C_1, \dots, C_k, \dots$, то каждая последовательность конфигураций, пробегаемая машиной T' из начальной конфигурации C'_0, содержит в качестве подпоследовательности последовательность конфигураций $C'_0, C'_1, \dots, C'_k, \dots$, где C'_i $(i = 0, 1, \dots, k, \dots)$ – образы конфигураций C_i машины T.

18. Эквивалентность программ и диаграмм МТ

Теорема. Каждой программе P, задающей МТ $T = (A, Q, P, q_0)$ можно эффективно составить диаграмму D, образованную символами элементарных МТ, так, чтобы МТ, определяемая этой диаграммой, моделировала бы машину T.

Построение диаграммы D. Каждой строке $(q, a, v, q') \in P$ поставим в соответствие на диаграмме символы $\cdot v \cdot$ Для $v = s$ ничего ставить не надо, ведь пустые действия на диаграмме не отображаются. В этом случае окончанием работы машины, определяемой диаграммой, будет правая точка предыдущего элемента диаграммы.

19. Эквивалентность диаграмм и программ МТ

Теорема. Каждой диаграмме Тьюринга D может быть эффективным образом сопоставлена программа P так, что МТ, описываемая этой программой, моделирует МТ, описываемую диаграммой D.

Построение программы P. Заменим на диаграмме D все символы неэлементарных МТ, продолжая замену до тех пор, пока на диаграмме не останутся только обозначения элементарных МТ. В полученной диаграмме пронумеруем одинаковые символы элементарных МТ, снабдив их числовыми индексами. Пронумеруем также все точки, соответствующие выходным состояниям. Каждому символу элементарной МТ сопоставим программу этой машины, пометив каждое состояние этой программы дополнительными индексами j и v, где v совпадает с названием соответствующей элементарной МТ, а j – номер, присвоенный этой элементарной МТ.

20. Первая теорема Шеннона

Для любой машины Тьюринга $T = (A_p, Q, P, q_0)$ с множеством состояний $Q = \{q_0, q_1, \ldots, q_s\}$ можно эффективным образом построить машину Тьюринга $T' = (A_r, \{\alpha, \beta\}, P', q_0')$, моделирующую машину T и имеющую всего 2 состояния: α и β. Рабочий алфавит A_r машины T' содержит $r = 3(p+1)(s+1)+p$ букв.

21. Вторая теорема Шеннона

Для любой машины Тьюринга $T = (A_p, Q, P, q_0)$ можно эффективным образом построить машину Тьюринга $T'' = (A_1, Q'', P'', q_0'')$, моделирующую машину T и имеющую однобуквенный рабочий алфавит $A_1 = \{a_1\} = \{|\}$.

22. Вычислимые функции

Пусть A_s и A_t – два алфавита и пусть $L \subseteq A_t^*$. **Функцией** из множества L в множество A_t^* называется правило, которое каждому слову $u \in L$ ставит в соответствие единственное слово $w \in A_t^*$, называемое значением функции f на слове u (это значение мы будем обозначать $w = f(u)$.

Функция $f : (A_s^*)^n \to A_t^*$ называется **вычислимой по Тьюрингу** (ВТ-функций), если существует МТ с рабочим алфавитом A_p, содержащим алфавиты A_s и A_t (в силу принятого соглашения об алфавитах; для этого достаточно, чтобы $p = \max(s, t)$) такая, что функция $f_T : (A_s^*)^n \to A_t^*$, определяемая для этой МТ, совпадает с f ($f_T \equiv f$)). О любой такой МТ говорят, что она вычисляет f.

23. Нормированные вычисления. Теорема о нормированной вычислимости

Функция $f : (A_s^*)^n \to A_t^*$ называется **нормированно вычислимой по Тьюрингу**, если существует МТ T_f, которая вычисляет f и при этом удовлетворяет следующим требованиям:

1. Если $X = (u_1, u_2, \ldots, u_n) \notin Def(f)$, то машина T_f после применения к X никогда не останавливается.
2. Если $X = (u_1, u_2, \ldots, u_n) \in Def(f)$, то T_f вычисляет значение функции.

Всякая ВТ-функция является НВТ-функцией, причем соответствующая МТ не использует никаких вспомогательных букв.

Для любой МТ можно эффективным образом построить машину T_N, которая имеет ленту, не ограниченную только с одного конца, и которая моделирует машину T.

24. Теорема о композиции

Пусть n-местная функция $f : (A_s^*)^n \to A_t^*$ определяется равенством:

$$f(u_1, u_2, \ldots, u_n) = h(g_1(u_1, u_2, \ldots, u_n), g_2(u_1, u_2, \ldots, u_n), \ldots, g_m(u_1, u_2, \ldots, u_n))$$

, где $g_i : (A_s^*)^n \to A_r^*$ $(i = 1, 2, \ldots, m)$ и $h : (A_r^*)^m \to A_t^*$ – ВТ-функции. Тогда f является ВТ-функцией, причем МТ, вычисляющая функцию f, может быть эффективно построена из МТ, вычисляющих функции g_i $(i = 1, 2, \ldots, m)$ и h.

25. Теорема о ветвлении

Теорема о ветвлении. Пусть n-местная функция $f : (A_s^*)^n \to A_t^*$ определяется равенством:

$$f(u_1, u_2, \ldots, u_n) = \begin{cases} g_1(u_1, u_2, \ldots, u_n), & \text{если } p_1(u_1, u_2, \ldots, u_n) = \text{true}; \\ g_2(u_1, u_2, \ldots, u_n), & \text{если } p_1(u_1, u_2, \ldots, u_n) = \text{true}; \\ \vdots & \\ g_m(u_1, u_2, \ldots, u_n), & \text{если } p_m(u_1, u_2, \ldots, u_n) = \text{true}; \end{cases}$$

Где $g_i : (A_s^*)^n \to A_t^*$ $(i = 1, 2, \ldots, m)$ – ВТ-функции, а $p_i : (A_s^*)^n \to \{\text{true}, \text{false}\}$ $(i = 1, 2, \ldots, m)$ – ВТ-предикаты, причем МТ, вычисляющая функцию f, может быть эффективно построена из МТ, вычисляющих g_i и предикаты p_i $(i = 1, 2, \ldots, m)$.

26. Теорема о цикле

Пусть $g : (A_s^*)^n \to A_s^*$ – ВТ-функция, вычисляемая машиной T_g, а $p : (A_s^*)^n \to \{\text{true}, \text{false}\}$ – ВТ-предикат, вычисляемый машиной T_p. Тогда функция $f : (A_s^*)^n \to A_s^*$, определяемая вычислительным процессом

$$g = g_0, \quad g_0 \in A_s^*$$
$$f(u_1, u_2, \ldots, u_n) = g(u_1, u_2, \ldots, u_{j-1}, g, u_{j+1}, \ldots, u_n)$$

27. Обобщённая теорема о цикле

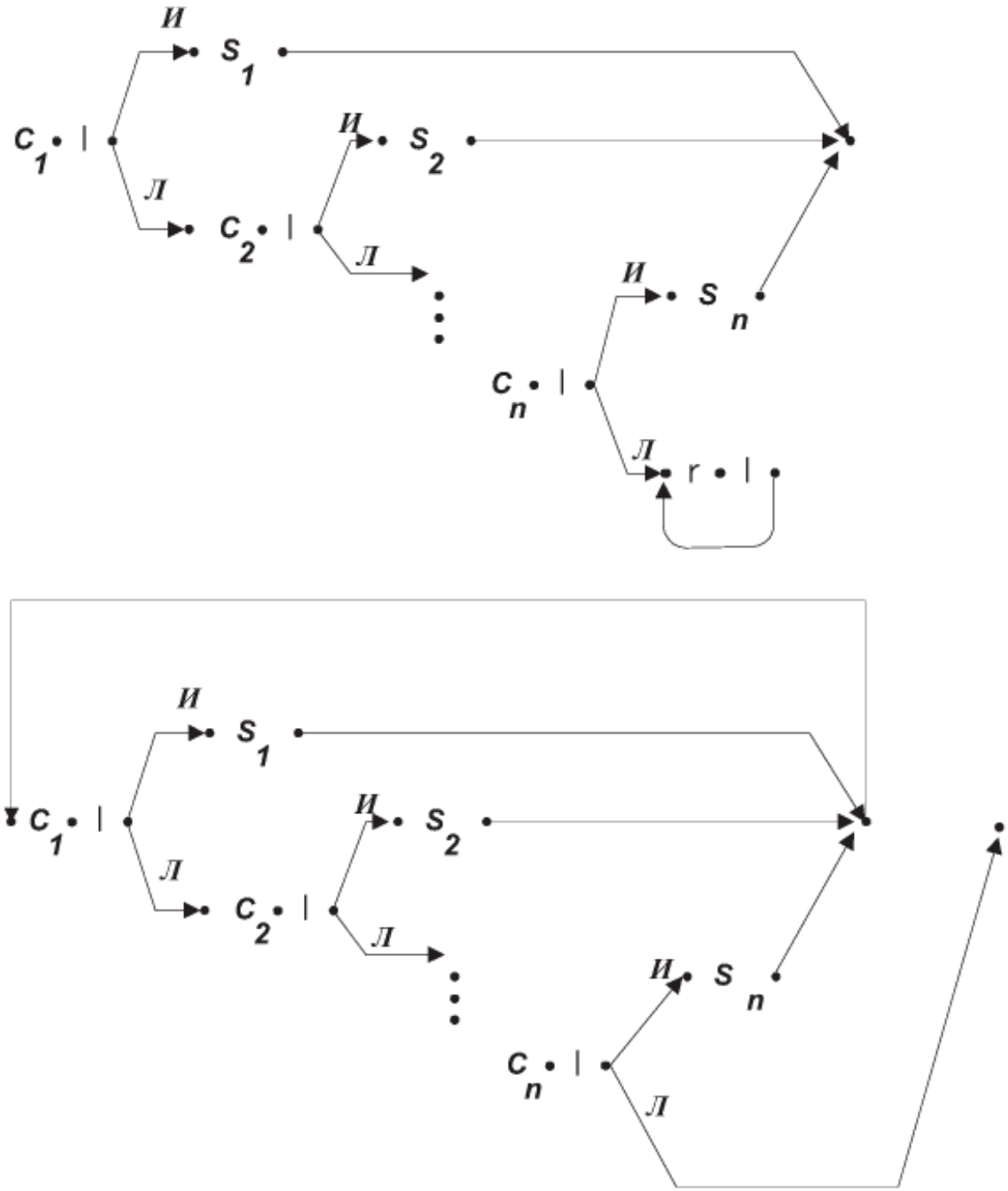
Пусть $g_i : (A_s^*)^n \to A_s^*$ $(i = 1, 2, \ldots, m)$ – ВТ-функции, вычисляемые МТ T_g, а $p_i : (A_s^*)^n \to \{\text{true}, \text{false}\}$ $(i = 1, 2, \ldots, m)$ – ВТ-предикаты, вычисляемые МТ T_{p_i} $(i = 1, 2, \ldots, m)$ соответственно. Тогда функция, определяемая соотношениями:

$$g_i = g_{0_i}, \quad g_{0_i} \in A_s^*, i = 1, 2, \ldots, m$$
$$f(u_1, u_2, \ldots, u_n) = \begin{cases} g_1(u_1, \ldots, u_{j_1-1}, g_1, u_{j_1+1}, \ldots, u_n), & \text{если } p_1(u_1, \ldots, g_1, \ldots, u_n) = \text{true}; \\ g_2(u_1, \ldots, u_{j_2-1}, g_2, u_{j_2+1}, \ldots, u_n), & \text{если } p_2(u_1, \ldots, g_2, \ldots, u_n) = \text{true}; \\ \vdots & \\ g_m(u_1, \ldots, u_{j_m-1}, g_m, u_{j_1+1}, \ldots, u_n), & \text{если } p_m(u_1, \ldots, g_m, \ldots, u_n) = \text{true}; \end{cases}$$
$$\text{пока} \bigvee_{j=1}^{m} p_j = \text{true}$$

28. Схемы машин Тьюринга. Нисходящая разработка

Понятие схемы машины Тьюринга определим следующим образом, отражающим нисходящий процесс конструирования:

1. Символ $\bullet$ составляет схему.
2. Если $S_1, S_2, \ldots, S_n$ – схемы, то их последовательная композиция $S_1 S_2 \ldots S_n$ является схемой.
3. Если $C_1, C_2, \ldots, C_n$ – схемы МТ, вычисляющих предикаты $P_1, P_2, \ldots, P_n$ соответственно, и $S_1, S_2, \ldots, S_n$ – схемы, то и следующие конструкции являются схемами:



4. Символы элементарных МТ составляют схемы.
5. Никакие другие объекты схемами не являются.

29. Теорема Бойма-Якопини-Миллса

Для любой машины Тьюринга T можно эффективно построить машину Тьюринга S, которая является структурной (т.е. диаграмма которой является схемой) и которая моделирует машину T.

30. Универсальная Машина Тьюринга

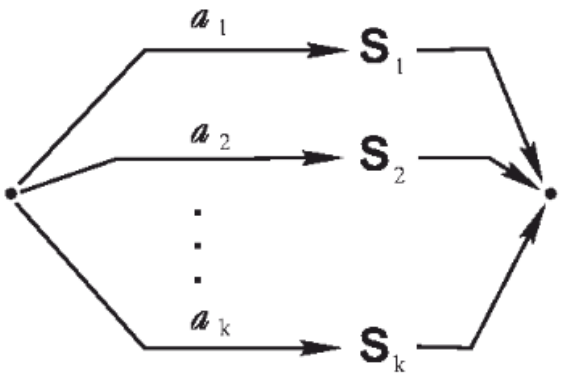
Машина Тьюринга, способная выполнить любой алгоритм, сымитировав, сымитировав работу соответствующей МТ, называется **универсальной**.

До начала работы универсальной МТ на её ленту записывается последовательность четверок, представляющая программу машины, работу которой необходимо выполнить, и начальные данные. Универсальная МТ просматривает программу, записанную на ее ленте, и выполняет команду за командой этой программы, получая на ленте вслед за аргументами результат вычислений. Текст программы (в виде последовательности четверок) помещается на ленте слева от аргументов и, следовательно, при нормированных вычислениях сохраняется неизменным в процессе выполнения программы.

31. Линейная запись схем машин Тьюринга

Композиция состоит в последовательном выполнении нескольких действий, представленных в схеме символами соответствующих МТ. Эта часть записи схемы и так линейна, так как в случае композиции символы указанных МТ просто выписываются один за другим.

Ветвление изображается фрагментом схемы, который имеет вид:

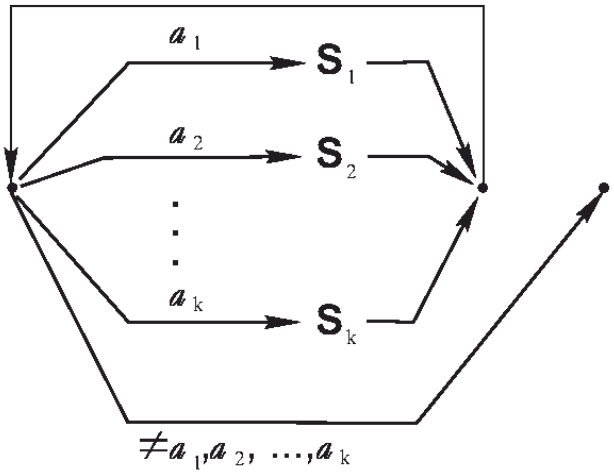


В линейной записи рассматриваемый фрагмент схемы будем представлять в виде

$$\textbf{IF} \quad a_1?S_1 \quad \square \quad a_2?S_2 \quad \square \quad \ldots \quad a_k?S_k \quad \textbf{FI}$$

IF и **FI** соответственно обозначают начало и конец ветвления (состояния q и q' соответственно). Знак "?" отделяет букву, написанную над стрелкой, от фрагмента схемы, который описывает действия в соответствующей ветви. Символ "□" отделяет одну ветвь от другой.

Цикл изображается следующим фрагментом схемы:



В линейной записи описание цикла имеет вид:

$$\textbf{DO} \quad a_1?S_1 \quad \square \quad a_2?S_2 \quad \square \quad \ldots \quad a_k?S_k \quad \textbf{OD}$$

DO и **OD** обозначают соответственно начало и конец цикла, а знаки "?" и "□" имеют тот же смысл, что в случае ветвления.

32. Критика моделей вычислений Тьюринга

Примеры программ на языке ОСТ показывают, что этот язык вряд ли можно считать удобным и эффективным средством описания алгоритмов.

1. Описания большинства программ МТ должны быть наряду с описанием основной программы, что явно увеличивает текст программы.
2. Необходимость многочисленных копирований. Можно заметить, что копирования составляют около половины всех действий, выполняемых при работе соответствующих МТ. Частые копирования не только резко увеличивают время выполнения программы, но и вызывают трудности при ее составлении; каждое слово присутствует на ленте в нескольких экземплярах, что усложняет проблему поиска нужных слов при составлении алгоритма.
3. Необходимость при составлении программ выписывать ситуации на ленте, так как если этого не делать то можно легко запутаться в расположении данных на ленте что приведет к ошибкам при составлении алгоритма.
4. Согласно выбранным способам нормированных вычислений, каждое неэлементарное действие выполняется над крайними правыми словами ленты, так что перед выполнением каждого неэлементарного действия необходимо переместить головку к крайнему правому слову, над которым оно должно быть выполнено. Необходимость постоянного слежения за ситуациями на ленте при составлении программы возникает в связи с тем, что положение слов на ленте определяется относительно текущего положения головки, которое все время меняется.

33. Алгоритмическая модель фон Неймана. Адреса и имена.

Для разработки более эффективных и удобных средств описания алгоритмов в середине 40-х годов была построена модель Джона фон Неймана.

Номер, сопоставленный некоторому слову, назовем **адресом** этого слова. Таким образом, каждому значению (слову), расположенному на ленте, ставится в соответствие его адрес, который не меняется в процессе выполнения программы. Приписанные словам адреса нигде в явном виде не записываются. Чтобы получить адрес конкретного слова, надо “проехать со счётчиком в руках” от первого слова к заданному. То есть понятие адреса вторично; оно задано конструктивным образом, т.к. адрес каждого слова может быть сгенерирован по известному алгоритму.

Если в процессе выполнения программы значения слов меняются, то будем их называть переменными, а их адреса – адресами переменных; если значения не меняются, то соответственно константами и адресами констант (подробнее с.116-117 методички, там много кода на неизвестном мне языке)

Текст программы становится еще более понятным, если вместо адресов переменных и констант использовать **имена**.

Именем называется слово над произвольным алфавитом, сопоставленное адресу переменной/константы. Сопоставление может быть осуществлено с помощью **таблицы имен**, т.е. последовательности слов, в которой перечислены все используемые имена, причем вслед за именем помещён соответствующий адрес.

34. Специализированные процессоры для обработки сообщений

Универсальная МТ, D-лента которой содержит программы, реализующие арифметические действия, имеет качественно новый уровень по сравнению с универсальной МТ с пустой D-лентой: алгоритмы, связанные с выполнением арифметических действий над целыми числами, описываются на ней гораздо более простыми программами. Назовем эту МТ **арифметическим процессором**.

Таким образом, предварительно разработав и записав на D-ленте набор внешних МТ, реализующих действия, используемые при составлении некоторого класса Тьюринговских программ, мы получаем **процессор** для этого класса программ, существенно упрощающий процесс разработки.

Действия, реализуемые МТ, записанными на D-ленту, называются **элементарными действиями процессора** или **операциями**.

Все внешние МТ, реализующие операции, имеют одинаковый рабочий алфавит, называемый **рабочим алфавитом процессора**

Каждая операция применяется к одному или нескольким словам над рабочим алфавитом процессора, называемых **операндами**.

Количество операндов определяет **местность** операции.

Операция выполняется нормированно: результат помещается за операндами.

Процессор наряду с D-лентой, на которой находятся программы его операций, наделяется R-лентой (так мы назовем часть ленты, расположенную левее ячейки с символом $\square$, на которую помещаются операнды перед выполнением операции), и S-лентой, на которой записываются программа и ее данные.

35. Построение универсального процессора фон Неймана

Для построения процессора необходимо:

- Выбрать и зафиксировать рабочий алфавит процессора; обычно в качестве рабочего алфавита будем брать $A_p = \{0, 1, 2, \ldots, p-1\}$ первых p неотрицательных чисел (начиная с 0).
- Зафиксировать конечное или бесконечное множество допустимых слов над алфавитом A_p; если множество допустимых слов конечно и включает лишь слова фиксированной длины k, то удобно добавить к множеству допустимых слов ещё одно слово, называемое "переполнением", которое означает, что в результате выполнения операции получилось слово, имеющее длину, большую, чем k.
- Составить и записать на D-ленту программы МТ, выполняющие операции процессора.
- Сообщить обозначения операций универсальной МТ, на основе которой строится процессор.

В число операций процессора могут входить **отношения**, т.е. такие операции, которые по одному или нескольким допустимым словам (аргументам отношения) вычисляют логические значения **ИСТИНА** и **ЛОЖЬ**; количество аргументов каждого отношения постоянно и называется **местностью**.

Отношения связывают числовые операнды, имеют истинностные значения и не являются операциями, поскольку размыкают множества допустимых операндов.

36. Машина фон Неймана. Принцип реализации

Машиной фон Неймана называется аппаратная реализация процессора фон Неймана. Машина фон Неймана состоит из управляющего устройства, устройства памяти (аппаратная реализация S-ленты) и одного из нескольких регистров (аппаратная реализация R-лент).

Принципы реализации:

- Устройство памяти, в отличии от S-ленты содержит лишь количество ячеек ограниченного размера, так что не любая программа, вычислимая по Тьюрингу (и ему подобным) реализуема на машине фон Неймана; данные, записанные в ячейки устройства памяти, недоступны для непосредственного восприятия человеком.
- В состав машины должны входить устройства для записи сообщений в память (**устройства ввода**) и вывода данных из памяти на носитель, доступный для восприятия человеком (**устройства вывода**).
- Язык описания программ для машины фон Неймана должен содержать средства управления устройствами ввода и вывода, преобразующими сообщения в данные и наоборот.

В результате аппаратной реализации процессора фон Неймана мы перешли от сообщений к данным, потеряв абсолютную вычислимость и лишившись возможности наблюдать и участвовать в процессе обработки данных в ЭВМ.

37. Структура программ для машины фон Неймана

Для обеспечения работы машины Фон Неймана необходимо поместить в ее память следующие объекты: программу, таблицу имен и все данные, которые должна обработать программа. Каждый объект размещается в одной или нескольких ячейках памяти, и, следовательно, связывается с определенными адресами памяти. Память машины фон Неймана сконструирована как хранилище произвольных слов без какой-либо интерпретации. Это означает, что ответственность за правильное использование слов лежит на программисте.

Для упрощения процесса составления таблицы имен имеющуюся в каждой программе обработки данных информацию о свойствах объекта будем в явном виде помещать в начало программы. Эта информация, в частности, позволяет назначить процессор для обработки поименованного объекта программы.

38. Нотация программ Э.Дейкстры. Обобщенная инструкция присваивания и композиции

Обобщенная инструкция присваивания. В любом языке программирования фон Неймановского типа основной элементарной инструкцией обработки данных является **инструкция присваивания**.

Инструкция присваивания имеет вид $X := A$.

Обобщением инструкции присваивания является инструкция, задающая одновременную замену нескольких объектов. Левая часть – список именнованых объектов. Правая часть инструкции присваивания должна содержать столько же выражений, сколько имен объектов в списке левой части инструкции. Инструкция одновременного присваивания имеет вид:

$$X_1, X_2, \ldots, X_n := A_1, A_2, \ldots, A_n;$$

При выполнении этой инструкции i-я переменная ($i = 1, 2, \ldots, N$) ее левой части получает одно и то же значение – значение этого выражения.

Обобщённая инструкция композиции. Для рациональной организации вычислений возможно разбивать процесс вычисления выражения на ряд последовательных процессов вычисления подвыражений, полученные значения которых можно было бы использовать в качестве аргументов при последующих вычислениях.

Составную инструкцию, состоящую из последовательности инструкций S_i, разделенных знаком ";", будем называть **композицией**:

$$S_1; S_2; S_3; \ldots; S_N$$

39. Обобщенная инструкция ветвления

Инструкция ветвления определяется следующим образом:

```
IF <предохранитель 1> <охраняемая инструкция 1>
    □ <предохранитель 2> <охраняемая инструкция 2>
    ...
    □ <предохранитель m> <охраняемая инструкция m>
FI
```

Символы **IF** и **FI** играют роль открывающей и закрывающей скобок ветвления, символ "□" играет роль разделителя охраняемых инструкций. По инструкции ветвления из списка охраняемых функций, заключенных в скобках **IF-FI**, должна быть выбрана одна и только одна инструкция для выполнения, и причем та, предохранитель которой принимает значение **ИСТИНА**.

Правила выполнения инструкции ветвления:

1. Одновременно и независимо вычисляются все предохранители.
2. Среди вычисленных предохранителей инструкции ветвления должен быть хотя бы один, принимающий значение **ИСТИНА**. Если таких нет, то происходит **ОТКАЗ** – выполнение программы прекращается аварийно. Это воспринимается как грубая ошибка периода выполнения, препятствующая дальнейшему хоть сколько-нибудь осмысленному выполнению программы.
3. Допускается, чтобы среди предохранителей инструкции было более одного предохранителя, принимающего значение **И**.

Не предполагается, что охраняемые инструкции каким-то образом упорядочены. Если среди предохранителей инструкции окажется два предохранителя, принимающих значение **И**, то выбор инструкции для выполнения осуществляется недетерминированным образом и не определяется порядком записи охраняемых инструкций.

40. Обобщенная инструкция цикла

Инструкция цикла определяется следующим образом:

```
DO <предохранитель 1> <охраняемая инструкция 1>
    □ <предохранитель 2> <охраняемая инструкция 2>
    ...
    □ <предохранитель m><охраняемая инструкция m>
OD
```

Символы **DO** и **OD** играют роль открывающей и закрывающей скобок ветвления, символ "□" играет роль разделителя охраняемых инструкций.

Правила выполнения инструкций **DO-OD**:

1. Если среди предохранителей инструкции один и только один принимает значение **И**, то соответствующая охраняемая инструкция выполняется, и после этого вновь осуществляется одновременная и независимая проверка всех предохранителей инструкций цикла.
2. Количество предохранителей, принимающих значение **И**, может быть и более одного. В этом случае не предполагается, что охраняемые инструкции упорядочены и выбор инструкции для выполнения очередного повторения цикла осуществляется вне связи с порядком их написания в инструкции, т.е. недетерминированным образом.
3. Выполнение каждой охраняемой инструкции должно приводить к изменению аргументов предохранителей. В противном случае выполнение инструкции может никогда не закончиться.

41. Понятие типа данных

Тип данных – это *множество изображений* (слов над некоторым алфавитом), для которых определено правило их интерпретации, позволяющее каждому изображению сопоставить его значение, и *множество атрибутов*, которые позволяют одному или нескольким элементам типа данных сопоставить либо изображения данных того же типа, либо изображения данных другого типа. В частности, к атрибутам типа данных относятся минимальное и максимальное значения, количество значений типа, операции, отношения и функции, определённые над значениями этого типа. Т.е. определение типа имеет теоретико-множественный и алгебраический аспекты. Интерпретация задается с помощью отображения:

$$\varphi : W \to X$$

, где W – множество изображений, а X – множество значений.

42. Тип логический

Значениями логического типа являются **Истина**, **Ложь** и **Неопределенность** (изображаемые словами **И**, **Т**, **True**, либо **Л**, **F**, **False** и $\perp$).

Над значениями логического типа определены:

- Одноместная операция $\neg$ (отрицание, NEG или !)
- Двухместные операции $\wedge$ (конъюнкция) и $\vee$ (дизъюнкция)
- Отношения $\equiv$ (эквиваленция, тождественность) и $\not\equiv$ (нетождественность, исключающее ИЛИ, XOR)

Операции логического типа обладают следующими свойствами:

1. Если хотя бы один операнд имеет значение $\perp$, то и результат имеет значение $\perp$
2. Если оба операнда дизъюнкции (конъюнкции) истинны (ложны), то истинен или ложен результат.
3. Если X истинно, а Y имеет произвольное логическое значение, отличное от $\perp$, то $X \vee Y$ также является истинным.
4. Если X ложно, а Y имеет произвольное логическое значение, отличное от $\perp$, то $X \wedge Y$ также является ложным.
5. Дизъюнкция и конъюнкция коммутативны.
6. $X \vee \neg X$ всегда истинно, если $X \neq \perp$

7. $X \wedge \neg X$ всегда ложно, если $X \neq \bot$
8. $\neg\neg X = X$

43. Тип литерный

Константы литерного типа – самоопределённые термы – заключаются в апострофы или кавычки. Множество значений определяется конкретной кодировкой. Эта кодировка задаёт порядковый номер литеры, связывающей литерный тип с поддиапазоном целого (обычно [0..255]). Одновременно кодовая таблица задаёт и обратное соответствие литер и внутренних кодов. Тип литерный имеет минимальный набор операций и отношений, включающий в себя присваивание и стандартный набор отношений. Литерный тип используется для ввода-вывода и обработки текстовых данных. Часто используются одномерные массивы литер, называемые строками.

В языке Си строкового как такового типа нет. Вместо строк используется массив литер. Признаком конца строки является элемент с кодом 0 (null-terminated string).

44. Тип целый

Значениями машинного целого типа являются все математические целые числа, заключенные между двумя фиксированными значениями **MIN** и **MAX**, являющимися атрибутами конкретного целого типа. Над значениями целого типа определены обычные целочисленные арифметические операции, а именно: сложение (+), вычитание (-), умножение (*), целочисленное деление (/ (для целочисленных операндов!)). Кроме этих операций дополнительно определены одноместная операция изменения знака числа на противоположный ($\neg$) и двухместная операция вычисления остатка от целочисленного деления (mod). Для значений целого типа определены отношения порядка ($>, \geqslant, <, \leqslant$) и отношения равенства ($=, \neq$).

45. Тип вещественный

Множество изображений вещественного типа – это множество слов конечной длины над алфавитом $\{0,1,2,3,4,5,6,7,8,9\} \cup \{+,-,.,\mathbf{E}\}$, причем длина слова, изображающего вещественное число, в языке программирования не фиксирована и определяется с помощью соответствующих базовых атрибутов, которые имеют определенное значение для каждой реализации.

Функция интерпретации определяется следующим образом:

$$\varphi_{\mathrm{REAL}}(s_1 a_0 a_1 \dots a_c . a_{c+1} a_{c+2} \dots a_d \mathbf{E} s_2 a_{d+1} a_{d+2} \dots a_k) =$$
$$= s_1 \left(\sum_{i=0}^{c} a_i \cdot 10^{c-i} + \sum_{i=c+1}^{d} a_i \cdot 10^{c-i} \right) \cdot 10^{s_2 \sum\limits_{i=d+1}^{k} a_i \cdot 10^{k-i}}$$

, где s_1 – знак числа, . – десятичная точка, $\mathbf{E}$ – символ порядка, s_2 – знак порядка, $a_0, \dots, a_n$ – цифры, изображающие целую часть вещественного числа, $a_d, \dots, a_k$ – числа, изображающие дробную часть вещественного числа.

Данное изображение – запись вещественного числа с плавающей точкой. Фактически оно изображается с помощью 2 чисел p и m, каждое из которых содержит конечное (фиксированное для каждой реализации) число цифр, так что вещественное число $X = m * B^p$, причём $minexp \leqslant p \leqslant maxexp; minmant \leqslant m \leqslant maxmant$; m называется мантиссой; p – порядком; число B называется основанием представления с плавающей точкой.

Итак, $\mathrm{INT}_{real}(m,p) = m * B^p$.

46. Согласование типов

Выражения со смешанными типами явно (с помощью функций согласования типов) или неявно (функции преобразования осуществляются языковой средой по умолчанию) приобретают вполне определенный однозначный смысл.

Помимо соответствия литерного и целого типов, можно выделить согласование целого и вещественного типов. Оно может осуществляться как округлением (round()), так и отбрасыванием дробной части (trunc()). Обратное преобразование (из целого в вещественный) производится по умолчанию в случае необходимости приведения смешанного числа к более сложному вещественному типу.

47. Небазовые типы данных (диапазон, перечисление, множество)

47.1. Отрезок (диапазон)

type diap = (0.0 .. 2.0; **real**; **real**);

Здесь в качестве множества значений берется отрезок вещественного типа, операции и отношения также заимствуются из вещественного типа.

47.2. Перечисление

В программе управления уличным движением удобно объявить

type StreetLight = ((Red, Yellow, Green)); ≔ ; **integer**);

(Крч непонятный какой-то псевдокод, но имеется в виду enum из 11-й лабы)

48. Понятие о структурном типе данных

Рассмотрим типы данных со структурными значениями.

Структурные значения – это упорядоченные систематически организованные совокупности других (в том числе и структурных) значений, рассматриваемых как единое целое. Компоненты структурных значений также называются его **полями** и **элементами**.

Переменные и константы структурного типа называются **структурными**.

49. Тип массив

Тип массив – это регулярный структурный тип с индексированным методом доступа. Регулярность структуры означает одинаковый тип всех компонент и использование в качестве индексной структуры декартова произведения отрезков перечислимых типов.

Для массивов в точности одного и того же типа определена операция присваивания и отношение равенства ($A \coloneqq B$ и $A = B$). Кроме того, могут быть тем или иным образом использованы операции над компонентами, например, целые и вещественные компоненты массивов можно умножать или располагать в порядке возрастания (убывания) с помощью атрибутов типов этих компонент.

Ввиду скалярности машины фон Неймана обработка массивов осуществляется покомпонентно, как правило, с помощью циклов с параметром (for) с подстановкой параметра цикла в индексное выражение.

50. Тип запись

Тип записи – это комбинированный структурный тип с квалифицированным методом доступа. Комбинированность означает, что поля записи имеют различные типы. Квалифицированный доступ, в отличие от индексируемого, не гибок и не вычислим, так как требует явного указания статически определенных до компиляции имён полей. Это компенсируется большой семантической нагрузкой мнемонических имен полей. С учетом возможной вложенности определений комбинированных типов легко реализуются иерархические нереляционные структуры, лучше отражающие реальные соотношения объектов в обществе и технике.

51. Понятие о файлах

Структура файла является обобщением понятия последовательности. Поэтому файлы прямого доступа следует считать “массивами на диске”. Компоненты файла должны быть одного типа, и они доступны только путем последовательного прочтения. Поскольку файлы ввиду их потенциально большого размера размещаются на устройствах внешней памяти, в каждый момент времени доступна лишь текущая компонента файла, а другие компоненты могут быть получены лишь последовательным прочтением компонент файла вперёд, или, после перемотки, с начала. Движения “назад” нет ввиду инерционности электромеханических устройств.

Файлы принято подразделять на внешние и внутренние, текстовые и нетекстовые, входные и выходные.

Внешние файлы существуют до начала работы программы и сохраняются после окончания ее работы.

Внутренние файлы, так же, как и внешние, описываются в программе как файловые переменные. Их время жизни совпадает с временем работы программы, или, в случае динамического сопоставления физических файлов, короче его. Внутренние файлы используются в качестве рабочих файлов – временной памяти практически неограниченного размера.

Текстовые файлы – используются для ввода-вывода или для хранения в виде, пригодном для ввода-вывода. Часто бывают только входными или выходными.

52. Блочная структура программ. Локальные и глобальные переменные.

Использование минимального набора инструкций позволяет решить проблему эффективной организации программ **по управлению**.

Для организации программ **по данным** была введена специальная конструкция – **блок**.

Блок – простейшая форма программной единицы, представляющая собой обособленный фрагмент программы со своим локальным контекстом.

Определим блок как разновидность составной инструкции ЯП, обозначаемую открывающей скобкой { и закрывающей скобкой } и имеющей следующую структуру:

```
begin
    <последовательность описаний>
    <последовательность инструкций>
end
```

Внутри блока описываются **локальные объекты**. Локальные объекты позволяют не заботиться об уникальности имен и защищены от несанкционированного использования или непреднамеренной порчи тем, что не видны в глобальном контексте. Локализация области действия переменных и констант позволяет разгрузить глобальный контекст от второстепенных деталей и оптимизирует использование памяти.

Глобальные переменные уже размещены в области памяти охватывающего блока. После завершения выполнения блока память, отведенная под локальные переменные и константы, освобождается и может быть повторно использована, что сокращает общую потребность программы в памяти, так что она значительно меньше суммарной длины всех переменных и констант.

53. Процедуры и функции. Описание

Значение процедур и функций в том, что они, как и блоки, являются средством представления программ в виде логически связанных компонентов. Процедуры помогают избегать повторения кода, в некоторых случаях ускоряют работу программы (в сравнении с тем случаем, где процедуры не использованы) и упрощают понимание программистом самой программы. Каждый осмысленный набор процедур – интерпретированное расширение языка программирования.

Подпрограмма – именованный фрагмент программы, предназначенный для выполнения специальной инструкции обращения.

Перед использованием даём описание: заголовок + тело.

Заголовок – вид, имя, количество и виды параметров, способ их передачи, тип результата.

Тело – блок, содержит описание локальных и глобальных объектов вместе с алгоритмами действий.

54. Вызов процедур и функций. Способ передачи параметров

Формальные (в заголовке) и фактические (при вызове) параметры.

Передача параметров обеспечивает связь по данным между подпрограммой и внешней средой.

Способы передачи параметров:

1. Передача **по значению** (может быть охарактеризована как read-only) – формальному параметру выделяется память в области данных программы. Значение вычисляется в точке вызова и пересылается в область памяти этого компьютера.
2. **По результату** (передаётся значение, а потом вычисляется результат) – формальный параметр получает память в области данных программы, используется как локальная переменная, ему нужно присвоить начальное значение. После вычисления возвращается значение параметра-результата.
3. **По ссылке** – передаётся адрес объекта и все обращения к параметру в подпрограмме происходят по этому адресу. Всякое присваивание значения внутри программы приводит к одновременному изменению значения во всей программе.
4. **По имени** – откладывает обработку параметров ровно до того момента, когда они действительно потребуются. Этот метод является самой мощной формой передачи параметров, наиболее опасной и неэффективной.

55. Понятие рекурсии. Рекурсия и итерации. Примеры

Это не из методички, у Зайцева нет рекурсии, в лекции 24 (с. 185–194) точно; это из StackOverflow

В некоторых случаях функция может вызывать саму себя. Этот процесс называется **рекурсией**. Сложность её использования связана с доведением рекурсии до конца, так как функция, которая вызывает сама себя, может делать это бесконечно, если в коде не предусмотрена проверка условия завершения рекурсии.

Итерация – это способ организации обработки данных, при котором определенные действия повторяются многократно, не приводя при этом к рекурсивным вызовам программ.

Рекурсия и итерация взаимозаменяемы, но не всегда с одинаковыми затратами по ресурсам и скорости.

Пример. Факториал числа:

С помощью рекурсии:

```
long fact(int n)
{
        long ans;
        if (n > 0)
                ans = n * fact(n - 1);
        else
                ans = 1;
        return ans;
}
```

Оно же с помощью итерации:

```
long fact(int n)
{
        long ans;
        for (ans = 1; n > 1; n--)
                ans *= n;
        return ans;
}
```

Рекурсивные решения более элегантные, но менее эффективные по сравнению с итеративными.

56. Рекурсивный вызов процедур

ХЗ, что тут писать...У ВЕ ни слова.

Принцип работы рекурсивного вызова:

1. Каждый уровень вызова функции имеет собственные переменные. То есть переменная, n уровня 1 отличается от переменной n уровня 2, так что программа создает несколько разных переменных,

каждая из которых имеет имя n и собственное значение, отличающееся от других. Когда в конечном итоге программа возвращается к вызову функции первого уровня, исходная переменная n по-прежнему имеет значение, с которого начинала.

2. Каждый вызов функции сбалансирован с возвратом. Когда поток управления достигает оператора `return` в конце последнего уровня рекурсии, управление переходит на предыдущий уровень рекурсии.
3. Операторы в рекурсивной функции, которые предшествуют рекурсивному вызову, выполняются в том же самом порядке, в котором эти функции вызывались.
4. Операторы в рекурсивной функции, которые находятся после рекурсивного вызова, выполняются в порядке, обратном тому, в каком эти функции вызывались.
5. Хотя каждый уровень рекурсии обладает собственным набором переменных, сам код не дублируется.
6. Очень важно, чтобы рекурсивная функция содержала код, который мог бы остановить последовательность рекурсивных вызовов.

57. Критика ЯП Паскаль и Си

Паскаль:

1. ЯП Паскаль представляет собой программно-компилируемую реализацию машины фон Неймана, и к нему можно отнести всю критику этой модели.
2. Оператор присваивания (:=) постоянно отвлекает программиста от решения задачи (как?)
3. Паскаль является строго типизированным языком. Все объекты программ на Паскале должны быть описаны и употребляются в строгом соответствии с описаниями. Паскаль содержит только такие языковые средства, которые эффективно компилируются на аппаратуру. Поэтому многие нужные программисту типы данных (такие как строки и массивы переменной длины) либо не реализованы вообще, либо их реализация наталкивается на ряд проблем.
4. Модель идеального компьютера, представляемая Паскалем, слишком далека от современной программно-аппаратной среды, представляемой современными ОС. Паскаль проигрывает Си в выразительной силе и лаконичности. Библиотека языка Си представляет собой весьма мощную интерпретируемую компоненту.

Критика языка Си: в методичке нет. Язык Си идеален.

58. Критика алгоритмической модели фон Неймана

Критика традиционных языков программирования всегда основывается на исследовании их интеллектуального предка – компьютера фон Неймана. Несмотря на прошедшие с тех пор 60 лет, суть фон неймановского компьютера не изменилась: по-прежнему он состоит из процессора, памяти и соединяющей их шины, которая за один такт может передавать только одно слово между ними. Это слово – либо данные, либо команда, либо адрес. Задача программы состоит в некотором существенном изменении содержимого памяти процессором исключительно посредством перекачки данных через шину из памяти и обратно. Большую часть потока составляют не полезные данные, а всего лишь имена (адреса) данных, а также команды и данные, служащие лишь для вычисления таких адресов.

Разумеется, должен существовать менее примитивный способ внесения в память больших изменений, чем мельтешение множества слов туда и обратно. Эта шина является не только узким местом для потока команд и данных задачи, но, что более важно, и интеллектуальным сужением, привязывающим нас к мышлению “слово за словом” вместо того, чтобы вдохновлять работу с более крупными концептуальными понятиями решаемой задачи. Программирование по фон Нейману большей частью заключается в планировании и спецификации огромного потока слов через это сужение, причём большая часть этого потока состоит не из самих значащих данных, а из сведений о том, где их искать. Обычные языки программирования в основном являются более высокоуровневыми и сложными программно реализованными версиями компьютера фон Неймана. В языках программирования фон Неймана переменные используются для имитации ячеек памяти компьютера; операторы управления выражают его команды передачи управления и проверки, а операторы присваивания имитируют загрузку содержимого ячейки и некоторую арифметику с последующим запоминанием результата.

Оператор присваивания. Именно он вынуждает нас программировать как это имело место на аппаратном компьютере фон Неймана с его шиной обмена данными между памятью и процессором.

Сохранение фон неймановских компьютеров сделало другие концепции неэкономичными и ограничила их развитие. Как альтернатива был предложен функциональный стиль программирования. Языки функционального программирования были задуманы как средство для рекурсивных построений. Процедуры в них могут служить данными, подставляемыми на место других аргументов. Каждое действие порождает некое значение, которое, в свою очередь, становится аргументом следующего действия и т.д. Основными средствами функционального программирования являются композиция и рекурсия. Функциональные языки хорошо подходят для смешанных вычислений, когда вычисления над данными приводят к получению программного кода, который может быть выполнен.