**Информатика. Зайцев**

**Билет 1**

**1. Формализация алгоритма (стр 39)**

Алгоритм - точно заданная последовательность правил, указывающая, каким образом можно за конечное число шагов получить выходное сообщение определенного вида, используя заданное входное сообщение. При этом подчеркивается, что действия, всем понятны, и легко выполнимы.

Формализация алгоритма реализуется с помощью построения алгоритмических моделей. Можно выделить три основных типа универсальных алгоритмических моделей:

• рекурсивные функции (понятия алгоритма связывается с вычислениями и числовыми функциями),

• машины Тьюринга (алгоритм представляется как описание процесса работы некоторой машины, способной выполнять лишь небольшое число весьма простых операций),

• нормальные алгоритмы Маркова (алгоритмы описываются как преобразования слов в произвольных алфавитах).

Этими моделями мы формализуем понятие алгоритма.

Использование алгоритмических моделей упирается на тезис Тьюринга-Чёрча о том, что всякий интуитивный алгоритм может быть выражен средствами одной из алгоритмических моделей. Также доказано, что всякий алгоритм, описанный средствами одной из моделей, может быть описан средствами другой. Проще говоря, одни модели сводятся к другим.

**2. Блочная структура**

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

Не менее важной проблемой является проблема организации программ по данным. Для организации программ по данным ещё в ранних языках программирования (АЛГОЛ60) была введена специальная конструкция — **блок**. **Блок** является простейшей формой программной единицы, представляющей собой обособленный фрагмент программы со своим локальным контекстом. Типичный пример объектов локального контекста — параметр цикла for или временная переменная для обмена значениями двух переменных. Иерархически вложенная система блоков позволяет организовать проектирование программы по данным как иерархический процесс детализации её функций. Каждая функция детализируется в виде блока с указанием неких характерных локальных данных. Если **блок** оказывается длинным и сложным, то в нём организуются дополнительные **блоки**.

Определим блок как разновидность составной инструкции языка программирования, обозначаемую открывающей скобкой (begin, {) и закрывающей скобкой (end, }) и имеющую следующую структуру:

begin / / Начало блока языка АЛГОЛ—60

<последовательность описаний>\

<последовательность инструкций>

end / / Конец блока

Известный нам составной оператор Паскаля блоком не является, так как не содержит описаний и не имеет локальных данных. Как и составной оператор, блок может быть вложенным и входить в другой, охватывающий блок. Во вложенном блоке доступны объекты охватывающего блока, если только они не экранированы омонимичными (одноименными) локальными переменными. Экранирование означает приоритет локальных объектов над глобальными и является средством разрешения конфликта имен. В то же время локальные переменные и константы блока недоступны в охватывающих блоках. В языках с развитой блочной и модульной структурой существуют средства явного экспорта/импорта глобальных и локальных объектов. Обычно программы, процедуры и функции представляют собой блоки, т. к. имеют описания локальных объектов. Локальные объекты позволяют не заботиться об уникальности имен при написании больших программ. Более того, они защищены от несанкционированного использования или от непреднамеренной порчи в большей части программы тем, что просто не видны оттуда. Локализация области действия переменных и констант позволяет не просто разгрузить глобальный контекст от второстепенных деталей, но и оптимизируют использование памяти. При использовании блочной структуры управление памятью локальных данных блоков происходит следующим образом: для каждого из блоков на время его выполнения выделяется область памяти для размещения локальных переменных и констант. Глобальные переменные и константы при этом 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. В каждый момент времени занимается память только под данные, необходимые текущему блоку и всем его прародителям.

**3. Цезарь на НАМ**

(для входного алфавита - русского, для английского аналогично)

\*A -> D\*

\*B -> E\*

\*C -> F\*

\*D -> G\*

\*E -> H\*

\*F -> I\*

\*G -> J\*

\*H -> K\*

\*I -> L\*

\*J -> M\*

\*K -> N\*

\*L -> O\*

\*M -> P\*

\*N -> Q\*

\*O -> R\*

\*P -> S\*

\*Q -> T\*

\*R -> U\*

\*S -> V\*

\*T -> W\*

\*U -> X\*

\*V -> Y\*

\*W -> Z\*

\*X -> A\*

\*Y -> B\*

\*Z -> C\*

-> \*

\* ->.

**Билет 2**

**1) Предмет информатики (стр 7)**

Информатика - наука об ЭВМ. Объединяет несколько более-менее независимых дисциплин. Одной из дисциплин является программирование.

Информатику определяют как науку об автоматической обработке информации при помощи ЭВМ:

Наука об осуществляемой с помощью преимущественно автоматических средств целесообразной обработке информации, рассматриваемой как представление знаний и сообщений в технических, экономических, и социальных областях.

Информатика во многом подобна математике. Подобно ей, она изучает законы, созданные человеком, и поддающиеся доказательству в отличие от естественных законов, знание которых всегда сопровождается некой долей неопределенности. Разница же заключается в предмете и подходе - математика обычно имеет дело с теоремами, бесконечными процессами и статистическими соотношениями, а информатика - с алгоритмами, конечными конструкциями, и динамическими соотношениями.

Как гласит программистский фольклор, математика делает то что можно так, как нужно, а информатика - то что нужно так, как можно.

**2) Универсальная МТ (106)**

Универсальной машиной Тью́ринга называют машину T, которая может заменить собой любую машину Tn. Получив на вход программу и входные данные, она вычисляет ответ, который вычислила бы по входным данным машина Tn, чья программа была дана на вход.

До начала работы на ее 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.

Построение универсальной МТ показало принципиальную возможность аппаратной реализации( в виде реально работающей аппаратуры) автоматического устройства, выполняющего любые алгоритмы по их описаниям.

Доказать универсальность некоторой системы - значит показать, что она может моделировать поведение некоторой системы, универсальность которой доказана, либо целого класса систем, который является универсальным (например всех МТ).

**3) Проверка палиндрома числа на Си**

**двоичное число**

#include <stdio.h>

#include <limits.h>

int main()

{

int a, b = 0, flag = 1;

scanf("%d", &a);

for (int a\_test = a; a\_test != 0;)

{

if (INT\_MAX/10 < b) { printf("Perepolneny\n"); flag = 0; break;}

else

{

b = b \* 10 + a\_test % 10;

a\_test /= 10;

}

}

if (flag == 1)

{

if (a == b) printf("Polindrom\n");

else printf("NonPolindrom\n");

}

system("pause");

return 0;

}

**Билет 3**

**1. НАМ (стр 56)**

В этой алгоритмической модели преобразуются текстовые сообщения. Элементарными тактами обработки алгоритмов Маркова являются замены подслов исходного сообщения на некоторые другие слова. Нормальные алгоритмы Маркова (НАМ) по существу являются детерминистическими (взаимосвязанными) текстовыми именами, которые для каждого выходного слова однозначно задают вычисления и, тем самым, в случае их завершения, порождают определенный результат. Марковская стратегия применения правил заключается в следующем: 1)Если применено несколько правил, то берется правило, которое встречается в описании алгоритма первым; 2)Если правило применимо в нескольких местах обрабатываемого слова, то выбирается самое левое из этих мест. НАМ представляет собой упорядоченный набор правил-продукций – пар слов, соединенных знаком ->. Слева от этого знака стоит слово, которое заменяется, а справа от этого знака стоит слово, на которое заменяется. Например строка az->z заменяет все слова az на z. Пример работы такой программы: на входе aaazazaz, на выходе zzz. НАМ не является нормируемым.

**2. Понятие файла (стр 179 либо 207)**

Структура файла является обобщением понятия последовательности. Поэтому файлы следует считать «массивами на диске». Компоненты все одного типа и они доступны только путем последовательного прочтения, движения «назад» нет ввиду инерционности электромеханических устройств. Аналог – магнитная лента. Внешние файлы обычно перечисляются в заголовке программы. Они существуют до начала работы программы и/или сохраняются после окончания ее работы. Внутренние файлы, также как и внешние, описываются в программе как файловые переменные. Их время жизни совпадает с временем работы программы. Текстовые – в них только символы, юзается спец. Знак для конца строки. Нетекстовые файлы не предназначены для ввода-вывода и хранят данные непосредственно во внутримашинном представлении.

**3. Целочисленное деление в кардинальной сс на Си. (НА ДТ)***2) Раз уж единички ограничены, то работаем с текстом (типа вертикальных палочек)*

/Целочисленное деление в кардинальной системе счисления на Си

Ввод : Палочки для двух чисел

Вывод: Палочки через пробел/

#include <stdio.h>

#include <stdlib.h>

int main() {

int i = ' ', flag = 0, a = -1, b = -1, c;

while((i = getchar())&&(i != EOF)) {

if (i == (int)'|' && flag == 0) {

a++;

}

if (i == (int)' ' && flag == 0) {

flag = 1;

}

if (i == (int)'|' && flag == 1) {

b++;

}

}

c = a / b;

printf("a=%d\nb=%d\nc=%d\n", a, b, c);

for (int i = -1; i < c; i++) {

printf("| ");

}

return 0;

}

[#include](https://vk.com/im?sel=134848551&st=%23include) <stdio.h>   
  
int sep(char c){   
if (c == ' ' || c == '\t' || c == '\n')   
return 1;   
return 0;   
}   
  
void write(int a, int b){   
a--; b--;   
if (b == 0)   
printf("Division by zero\n");   
else{   
printf("%d/%d = %d\n", a, b, a/b);   
for(int i = 1; i <= (a/b)+1; i++)   
printf("|");   
printf("\n");   
}   
}   
  
int main(){   
char c = '0', prec = '0';   
int a = 0, b = 0;   
  
while(scanf("%c", &c)){   
if (sep(c)){   
if (sep(prec))   
continue;   
if (a != 0){   
write(a, b);   
b = 0;   
}   
a = b;   
b = 0;   
}   
else if (c == '|')   
b += 1;   
prec = c;   
}   
return 0;   
}

**4 билет**

**1) II теорема Шеннона. Доказательство**

Для любой МТ Т=(A,Q,P,qo) можно эффективным образом построить МТ T =( A1 ,Q , P ,q0), моделирующую машину Т и имеющую всего однобуквенный алфавит A1={a1}={∨}.

Доказательство теоремы основывается на следующих рассуждениях. Первоначально последовательность X={xi} кодируется символами из В так, что достигается максимальная пропускная способность (канал не имеет помех). Затем в последовательность из В длины n вводится r символов по каналу передается новая последовательность из n + r символов. Число возможных последовательностей длины n + r больше числа возможных последовательностей длины n. Множество всех последовательностей длины n + r может быть разбито на n подмножеств, каждому из которых сопоставлена одна из последовательностей длины n. При наличии помехи на последовательность из n + r выводит ее из соответствующего подмножества с вероятностью сколь угодно малой.

Теорема позволяет определять на приемной стороне канала, какому подмножеству принадлежит искаженная помехами принятая последовательность n + r, и тем самым восстановить исходную последовательность длины n.

Как и в случае Теоремы I, доказательством будет схема построения. Покажем, что можно построить ма­шину ***С****,* работающую подобно любой заданной машине Тьюринга ***А***и использующую только два символа внешний алфавит, например символы 0 и 1.

*Пусть машина А содержит: n внутренних состояний Sj, m символов внешнего алфавита ai. Тогда машина С будет содержать: n внутренних состояний Tj, являющихся аналогами Sj, не более чем 8nm специальных внутренних состояний, 2 символа внешнего алфавита: 0 и 1*

**Общая идея построения**

Пусть *l* - наименьшее целое число, для которого m≤2*l*. Тогда символам машины ***А***можно сопоставить двоичные последовательности длины *l* таким образом, что различным символам будут соответствовать различные же последовательности. При этом пустому символу машины ***А***мы ставим в соответствие последовательность из *l* нулей. Машина ***С***будет работать с дво­ичными последовательностями. Элементарная операция ма­шины ***А***будет соответствовать в машине ***С***переходу счи­тывающей головки на *l* - 1 клеток вправо (c сохранением считанной информации во внутреннем состоянии головки), затем обратному переходу на *l* - 1 клеток влево, записи новых символов по пути и, наконец, движению вправо или влево на *l*клеток, в соответствии с движением считывающей головки машины ***А****.* В течение этого процесса состояние машины ***А****,* конечно, сохраняется и в машине ***С****.* Замена старого состояния новым происходит в конце опера­ции считывания.

**2) Тип логический**

Boolean. Тип данных, который может принимать всего два значения, 0 или 1, True или False, T или F. Занимает всего 1 бит памяти. Популярные операции: Конъюнкция, дизъюнкция, тождественность и отрицание. Операции логического типа обладают следующими свойствами:

**1.** Если хотя бы один операнд имеет значение NULL, то и результат имеет значение NULL

**2.** X&X=X, XVX=X при X=И, Л

**3.** X = И, Y=любое значение кроме NULL, то, XVY=И

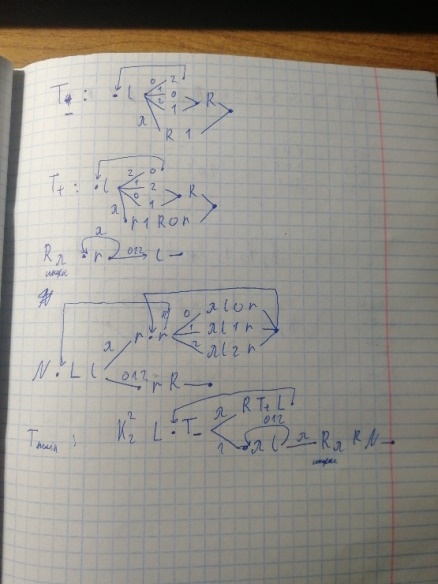
**4.**  X = Л, Y=любое значение кроме NULL, то, X&Y=Л

**5.** Дизъюнкция и конъюнкция коммутативны.

**6.** XV-X=И, кроме Х= NULL

**7.** X&-X=Л, кроме Х= NULL

**8.** -(-X)=X

**3) Сложение в троичной системе счисления на ДТ**

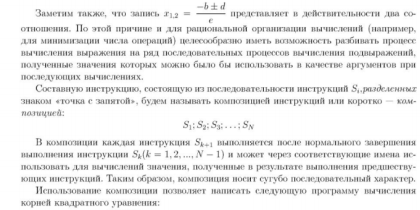
**5 билет**

**1)Знаки и символы (стр 18)**

Письменные языковые сообщения представляют из себя последовательности знаков. Мы будем различать атомарные и составные знаки. Атомарным знаком или буквой называется «неделимый» символ 1-ого уровня. Пример: буквы (А, Б, В….). Они складываются в знаки второго уровня (слова). Знаки второго уровня в знаки третьего (предложения) и т.д. (абзацы и пр.). Все знаки большего 1 уровня называются составными знаками. Множество слов также можно рассматривать как набор знаков. Для записи сообщения, которое содержит знаки k-го уровня необходимо иметь k-1 различных типов знаков разделителей. Есть специальный знак, чтобы разделять одни составные знаки от других. Для слов это пробел, для предложений – точка. С каждым знаком связывается его смысл или семантика. Знак вместе с сопоставленной ему семантикой называется символом. (А – буква, пи – символ).

**2) Критика моделей вычислений Тьюринга**

Недостатки МТ довольно существенны.

**1.** Описание программы, выполняющая алгоритм сложения двух дробей занял целую страницу, поэтому для более сложных математических операций понадобится большая трудоемкость программиста, даже при составленном алгоритме.

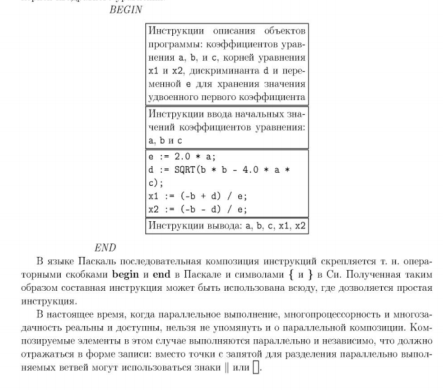
**2.** Необходимость многочисленных копирований. При описании нашего алгоритма около половины всех действий были именно действиями копирования. Из-за этого поиск нужных слов среди всего это громоздкого кода становится довольно затруднительным.

**3.** Необходимость при составлении программы выписывать ситуации на ленте, иначе можно легко запутаться. МТ удобно для построения красивой теории алгоритмов, но не для практических задач.

**3)Вывести первые 500 простых чисел на Си**

#include <stdio.h>

int main(){

**** int i=2, k, m, n=500;

while (n>0){

m=i;

k=0;

while (m>1){

if (i%m==0) k=k+1;

if (k>1) break;

m=m-1;

}

if (k==1) {

printf ("%d ",i);

n=n-1;

}

i=i+1;

}

}

Второй вариант: zaliupa

#include <stdio.h>

int main()

{

int i,j;

i=1;

j=0;

while(j<500)

{

if ((i == 1) || (i == 2) || (i == 3) || (i == 5) || (i == 7)) {

printf("%d ", i);

j++;

}

else if ((i % 2 != 0) && (i % 3 != 0) && (i % 5 != 0) && (i % 7 != 0)) {

printf("%d ", i);

j++;

}

i++;

}

printf("\n");

printf("%d",j);

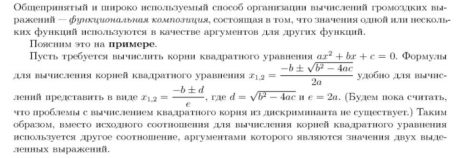
return 0;

}

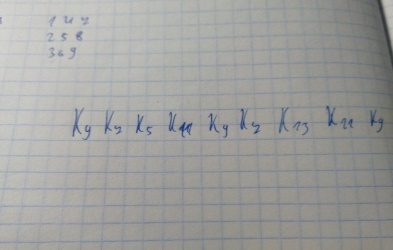
**Билет 6**

**1. Информация и сообщение (стр 16)**

Информация и сообщение – основные (неопределяемые) понятия информатики. Информация передается посредством сообщения и наоборот, сообщение – то, что несет информацию. Информация может существовать только в форме некоторого сообщения. Соответствие между информацией и несущим ее сообщением не является взаимно однозначным: 1)Одна и та же информация может передаваться с помощью различных сообщений; 2) одно и то же сообщение может передавать различную информацию

**2 . Обобщенная инструкция композиции**

**3.Транспонирование матрицы 3\*3 на ДТ**



**7 билет**

**1. Эквивалентность диаграмм и программ(стр 53)**

Каждой ДМТ, задающей программу Р, можно эффективным образом сопоставить МТ Т=(A,Q,P,qo) , образованную строками команд МТ так, чтобы МТ, определяющая эту диаграмму, смоделировала бы машину Т. Т.е. нам нужно указать способ эффективного построения программы по ДМТ программы Р, а потом убедиться, что для исходной ДМТ и МТ, выполняются все пять условий моделирования МТ. Построение: Каждому элементу диаграммы .v. сопоставим строку команды A ,Q, P,qo ,при этом нам нужно будет поставить вместо последней точки ДМТ заключающую строку программы МТ. Док-во, что оно эквивалентно. Все 5 условий моделирования должны быть выполнены. Условие 1 выполняется потому, что алфавиты МТ и ДМТ совпадают. Условие 2 выполняется потому, что точкам ДМТ соответствуют состояния МТ. Условие 3-5 выполняются потому, что МТ определяет ту же последовательность элементарных действий, что и программа Р.

**2. Структура программ для машины фон Неймана**

Программа и данные помещаются в одном и том же устройстве памяти, но на регистрах процессора слова с командами программы и данными существенно различаются. Таблица имен, программа и данные могут быть записаны в виде, непосредственно переносимом в память машины. Процесс составления таблицы имен и размещения данных в памяти машины может быть автоматизирован. Каждый объект, используемый для получения значений других объектов при выполнении программы обработки данных должен иметь определенное значение. Начальные значения таких объектов могут быть заданы либо при размещении в их памяти машины (константы), либо путем выполнения инструкции ввода данных\*/ А теперь к сути. Программа для машины фон Неймана представляет собой текст, обязательно включающий в себя инструкции описания объектов (данных, которые обрабатывает программа) инструкции ввода данных, инструкции обработки данных и инструкции вывода значений объектов-результатов. Начало => инструкции описания объектов программы => инструкции ввода => инструкции обработки данных => инструкции вывода => КОНЕЦ

**3. Реверс байтов целого числа на Си**

**unsigned rev(unsigned n)  
{  
 unsigned r = 0;  
 for (int i = sizeof(n) \* CHAR\_BIT; i--;) {  
 r \*= 2;  
 r += n % 2;  
 n /= 2;  
 }  
 return r;  
}**

#include <stdio.h>

int main()

{

int num1, num2;

printf("Введите исходное целое число ");

scanf("%d", &num1);

// вариант со сдвигом

num2 = ((num1-(num1>>8))<<8+(num1>>8));

printf("Результат реверса байтов %d", num2);

return 0;

}

**8 билет**

**1)Системы счисления**

Система счисления – это способ числовой интерпретации цифровых сообщений. Непозиционные: а) Натуральная система счисления. Число принадлежит целым от единицы до бесконечности. Пример – пересчет предметов. II яблока. б) Кардинальная система счисления. Для нуля отдельная палочка. Недостатки при мат. Операциях, (после сложения надо убрать одну палочку) Проблемы: для числа 1000 потребуется половина экрана терминала. Позиционные: An=ak∗nk−1+…+a2∗n1+a1∗n0 . Для отрицательных чисел вся эта сумма домножается на -1. Пример (десятичная) 34910=3∗102+4∗101+9∗100 . Перевод из двоичной в десятичную. 1012=1∗22+0∗21+1∗20 . Как мы видим, позиция влияет на значение числа. Поэтому эта система счисления называется позиционной

**2)Построение процессора фон Неймана**

Для построения процессора необходимо:

**1.** Выбрать и зафиксировать рабочий алфавит процессора. Обычно это первые p чисел, включая ноль. Т.е. от 0 до p-1.

**2.** Зафиксировать конечное или бесконечное множество допустимых слов над алфавитом. Если множество допустимых слов конечно и включает лишь слова фиксированной длины к (либо слова, имеющие длину не больше, чем к(, то удобно добавить к множеству допустимых слов еще одно слово, называемое «переполнением».

**3.** Составить и записать на ленту программы МТ, выполняющие операции процессора, включая операции РИМ и АДР

**4.** Сообщить обозначения операций универсальной МТ, точнее, управляющей программе универсальной МТ, на основе которой строится процессор.

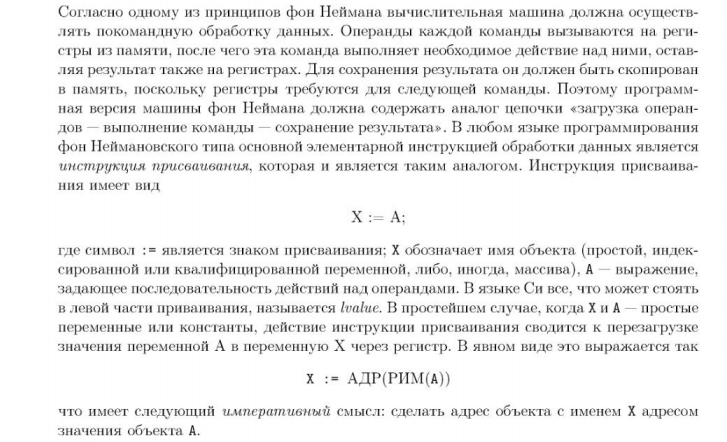
**3)Поразрядная конъюнкция двоичных чисел на ДТ**

**9 билет**

**1) обработка сообщений**

Обработка сообщений. Примеры: кодирование, перевод с одного языка на другой, перепечатывание, чтение вслух и пр. Т.е. обработка сообщений состоит в выделении в исходном сообщений знаков некоторого уровня, составляющих это сообщение, и замене каждого выделенного знака другим знаком. При чтении вслух каждый знак заменяется звуком,, фонемой. Следовательно любая обработка сообщений может рассматриваться как кодирование в широком смысле

**2) обобщенная инструкция присваивания**

****

**3) НАМ проверка лексикографического упорядочения двоичных слов**

на вход поступают двоичные числа равной длины через символ \_

#\_->#

#&->#\*

\*0&->0\*

\*1&->1\*

\*0\_->0\_

\*1\_->1\_

\*1->/

\*0->/

/1->/

/0->/

/\_->/

\_/->/

1/->/

0/->/

#/->/

/->.YES

%1->%

%0->%

%\_->%

\_%->%

1%->%

0%->%

%&->%

&%->%

%->.NO

#0->>

#1->@

@0->0@

@1->1@

@\_->\_!

>0->0>

>1->1>

>\_->\_^

^&0->&0^

^&1->&1^

^0-><&0

^1-><&1

!&0->&0!

!&1->&1!

!0->%

!1->?&1

&0<-><&0

&1<-><&1

&0?->?&0

&1?->?&1

\_?->?\_

\_<-><\_

0<-><0

1<-><1

<->#

0?->?0

1?->?1

?->#

&0->0

&1->1

**10 билет:**

**1) Свойства алгоритмов**

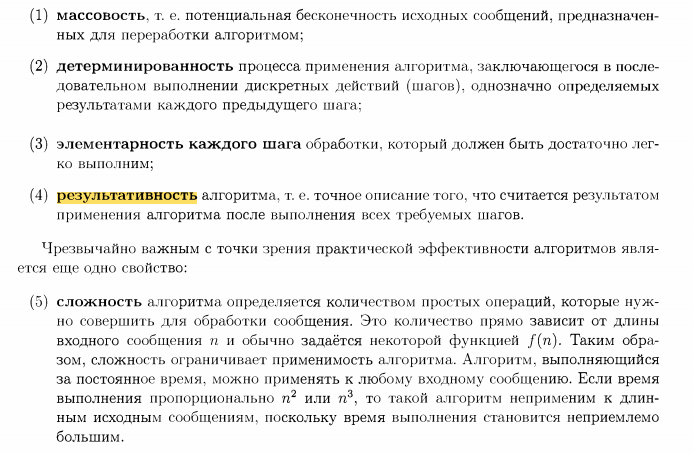
**1.** Массовость, т.е. потенциальная бесконечность исходных сообщений для обработки

**2**. Детерминированность – заключается в последовательном применении шагов, однозначно определяемых результатами каждого предыдущего шага

**3.** Элементарность каждого шага, каждый шаг алгоритма должен выполнять только одно определенное действие

**4.** Результативность алгоритма - точное описание того, что считается результатом применения алгоритма после выполнения всех требуемых шагов.

**5.** Сложность алгоритма определяется кол-вом простых операций, которые нужно совершить для обработки сообщения. От него зависит время выполнения алгоритма.



**2) Построение машины фон Неймана (такого нет в методичке,точнее частично)**

Машиной фон Неймана называется аппаратная реализация процессора фон Неймана. Машина состоит из управляющего устройства, устройства памяти и одного или нескольких регистров. Память машины фон Неймана содержит конечное число ячеек ограниченного размера. В состав машины должны входить устройства записи сообщений в память и вывода данных из памяти на носитель, доступный для восприятия органами чувств человека. В результате аппаратной реализации процессора фон Неймана мы перешли от сообщений к данным, потеряв абсолютную вычислимость и лишившись возможности непосредственно созерцать процесс обработки данных в ЭВМ и тем более участвовать в нём.   
Но приобретения более значительны: мы получили возможность быстрой и безошибочной

автоматической обработки данных без участия человека. Абсолютная вычислимость в машине фон Неймана заменена частичной. Вычислимым по фон Нейману считается все то,

что может быть получено за приемлемое время на доступных ресурсах памяти и, весьма

часто, в пределах ограниченного бюджета имеющихся средств.

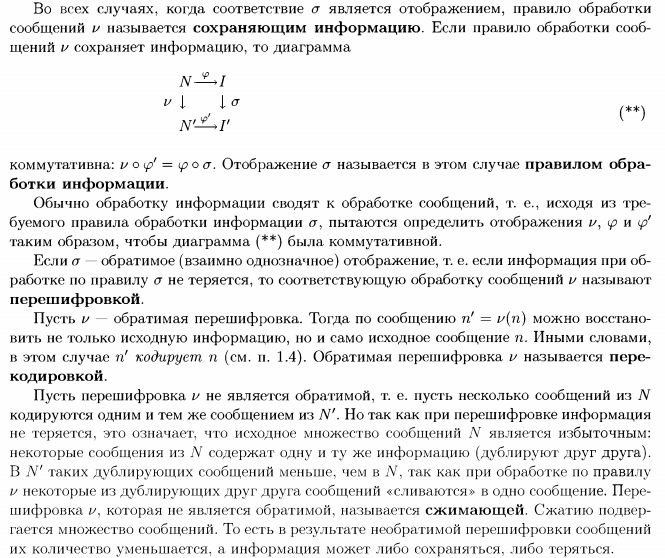
**3) ДТ проверки делимости на 3**

**11 билет---**

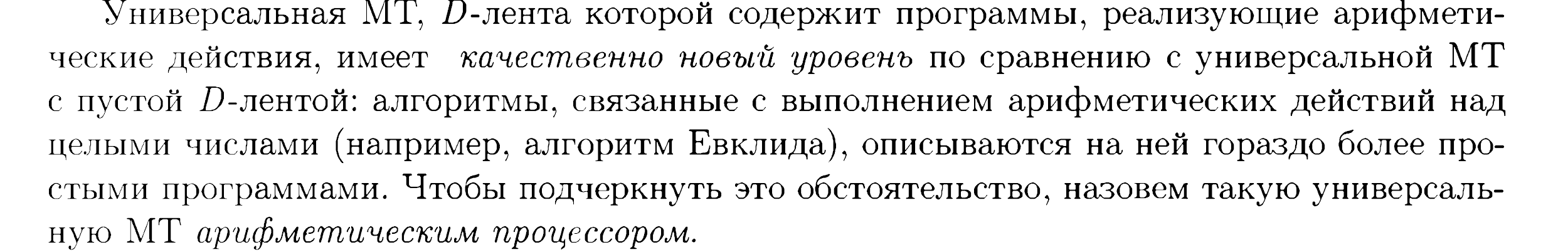
**1) обработка информации**

Во всех случаях, когда правило обработки σ является отображением, правило обработки сообщений v называется сохраняющим информацию. Если v сохраняет информацию, то диаграмма коммутативна: v °φ’=φ°σ . Отображение σ называется в таком случае правилом обработки информации.

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

Пусть v – обратимая перешифровка. Тогда обработка обратимой перешифровкой называется перекодировкой. Иначе она называются сжимающей. Т.е. в результате необратимой перешифровки сообщений их кол-во уменьшается, а информация может либо сохраняться, либо теряться.  
  


**2) специализированные процессоры для обработки сообщений (стр 118)**

****

**3) НАМ: умножение двух чисел в натуральной системе счисления**

решение, м? :) 1111#11111 через решетку два t числа

|ab->ab|b

ab->

|b->b|

#||->ab#|

#|->

b->|

**12 билет**

**1) сложность алгоритмов**

Сложность алгоритма определяет, как долго придется ждать, пока будет получен результат его работы. Всего выделяют 7 классов сложности:

1) О (1) – алгоритмы с постоянным временем выполнения, например доступ к элементу массива. При увеличении размера задачи вдвое время выполнения не меняется

2) О (log n) – Алгоритмы с логарифмическим временем выполнения, например, бинарный поиск. Время выполнения удваивается при увеличении размера задачи в n раз O(n) –

3) O (n) - Алгоритмы с линейным временем выполнения, например, последовательный поиск или схема Горнера. При увеличении размера задачи вдвое, время выполнения также удваивается.

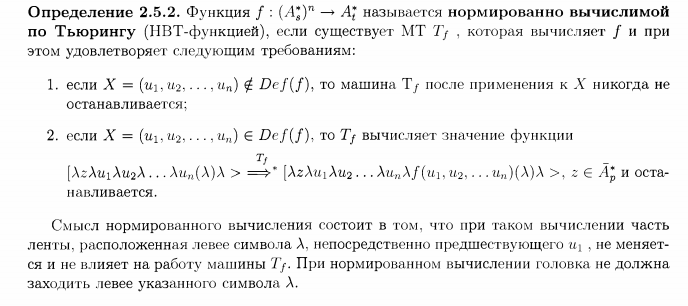
4) O (n\*logn) - Алгоритмы с линеарифмическим временем выполнения, например, быстрая сортировка. При увеличении размера вдвое, время выполнения увеличивается немного больше, чем вдвое.

5) O(n^2) - Алгоритмы с квадратичным временем выполнения, например, простые алгоритмы сортировки. При увеличении размера задачи вдвое, время выполнения увеличивается в 4 раза.

6) O (n^3) – Алгоритмы с кубическим временем выполнения, например, умножение матриц. Время выполнения увеличивается в 8 раз при увеличении размера задачи в 2 раза.

7) O (2^n) - Алгоритмы с экспоненциальным временем выполнения, например, задача о составлении расписания. Время выполнения увеличивается в 2n при увеличении размера задачи в 2 раза

**2) нормированные функции, теорема**



**3) написать Дифференциал многочлена на си**

//Дифференциал многочлена (на вход поступают

//коэффициенты многочлена, на выход - коэффициенты

//производной от многочлена)

#include <stdio.h>

int main(){

int a[100];

int power = -1;

while(power > 100 || power < 0){

printf("Введите степень многочлена:\n");

scanf("%d",&power);

}

printf("Введите коэффициенты: \n");

for(int i = 0; i <= power; i++)

scanf("%d",&a[i]);

printf("Результат:\n");

for(int i = 0; i < power; i++)

printf("%d ", a[i]\*(power-i));

printf("0\n");

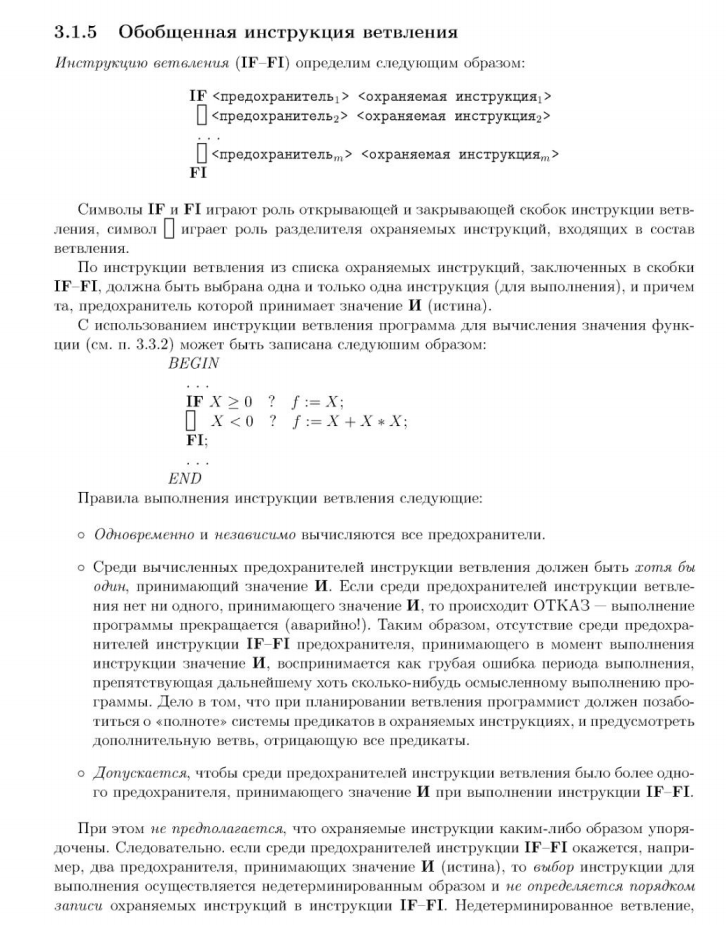
}

**Билет 13**

**1)Конструктивное описание процесса обработки дискретных сообщений**

Обработка дискретных сообщений состоит в применении отображения **v** или в последовательном применении отображений **C P Q**. Чтобы отображение **Р** могло служить основой для автоматизации обработки, оно должно задавать некоторый способ построения сообщения, исходя из сообщения **D**. Для этого нужно представить **Р** в виде последовательности элементарных шагов обработки или тактов, каждый из которых достаточно прост для выполнения физ. Объектом., такие шаги называются операциями. Например – переписывание слова: Переписываем последовательно каждую букву.

**2)Обобщенная инструкция ветвления**

****

****

**3)Проверка палиндрома с помощью НАМ**

либо двоичное число

двоичное число

решение, м? :) всего три буквы, а то алфавит очень большой, много кода. Тесты: abba ababa abbc bacb bbb cab.

##a->a##

##b->b##

##c->c##

a(a)->(a)a

b(a)->(a)b

с(a)->(a)c

a(b)->(b)a

b(b)->(b)b

с(b)->(b)c

a(c)->(c)a

b(c)->(c)b

с(c)->(c)c

#(a)a->#

#(b)b->#

#(c)c->#

#(a)#->##

#(b)#->##

#(c)#->##

a#->(a)#

b#->(b)#

c#->(c)#

#-0->-0

)-0->-0

(-0->-0

a-0->-0

b-0->-0

c-0->-0

1###->.1

##->-0

1-0->.0

->1###

**14 билет**

**1. Линейная запись схем МТ**

Ветвление диаграммы представляется в виде: **IF a1?S1 | a2?S2 | … | ak?Sk FI**. Где **а** – условие, **S** – действие. **IF – FI** – начало и конец ветвления. Цикл представляется как **DO a1?S1 | a2?S2 | … | ak?Sk OD**. Все то же самое, только **DO** и **OD** – начало и конец цикла. Это позволит нам преобразовать двумерную диаграмму в одномерную строку

**2. Согласование типов**

Согласование целого и вещественного типа, как близких с математической точки зрения, осуществляется с помощью округления или отбрасывания дробной части **(round() или trunc())**. Обратное преобразование (из целого в вещественный) производится по умолчанию в случае необходимости приведения смешанного выражения к более сложному вещественному типу.

**3. Двоичный реверс машинного слова на Си**

Предположим, на вход подаётся целое число в десятичной системе счисления, целого типа с длиной 32 бита. Тогда реверс двоичного слова будет выглядеть так:

#include <stdio.h>

int a, b;

int main()

{

scanf("%d",&a);

b=0;

while (b<32){

printf("%d", ((a>>b)&1));

b++;

}

return(0);

}

**15 билет**

**1)эквивалентность диаграмм и программ**

Каждой ДМТ, задающей программу Р, можно эффективным образом сопоставить МТ Т=(A,Q,P,qo) , образованную строками команд МТ так, чтобы МТ, определяющая эту диаграмму, смоделировала бы машину Т. Т.е. нам нужно указать способ эффективного построения программы по ДМТ программы Р, а потом убедиться. Что для исходной ДМТ и МТ, выполняются все пять условий моделирования МТ. Построение: Каждому элементу диаграммы .v. сопоставим строку команды A ,Q, P,qo ,при этом нам нужно будет поставить вместо последней точки ДМТ заключающую строку программы МТ. Док-во, что оно эквивалентно. Все 5 условий моделирования должны быть выполнены. Условие 1 выполняется потому, что алфавиты МТ и ДМТ совпадают. Условие 2 выполняется потому, что точкам ДМТ соответствуют состояния МТ. Условие 3-5 выполняются потому, что МТ определяет ту же последовательность элементарных действий, что и программа Р.

**2)тип литерный**

Константы литерного типа — самоопределённые термы — заключаются в апострофы или кавычки: ’3’, ’а’. Множество значений определяется конкретной кодировкой: ASCII, КОИ-8 и т.д. Эта кодировка задаёт порядковый номер литеры, определяемой функцией преобразования типа ord, связывающей литерный тип с поддиапазоном целого (обычно [0..255]), и предопределяет упорядоченность множества значений так, что имеет смысл понятия следующей и предыдущей литеры (succ и pred соответственно). Одновременно кодовая таблица задаёт и обратное соответствие литер и внутренних кодов (в Паскале это функция chr). Таким образом, для всякой литеры с имеет место соотношение с = chr(ord(c)) и наоборот (для малых положительных целых, обычно в диапазоне [0..255]) i = ord(chr(i) ). Кроме того, с = succ(pred(c)) = pred(succ(c)). Для языка Си все условности отброшены и транслятор осуществляет неявное автоматическое преобразование int 4Ф- char: i = с или с = i. 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 (в том числе и для изображений значений других типов словами — цепочками знаков). По этой причине в языке Паскаль литерный тип является «более элементарным», чем другие типы. Основные входной и выходной файлы Паскаль-программы (input и output) — литерного типа. Имеется соответствующий предопределенный файловый тип text. В настоящее время ввод-вывод утрачивает свою литерную ориентацию, приобретая графический и мультимедийный характер. Однако литерный тип весьма важен как простой и удобный стандарт.

**3)проверка на палиндромию машинного слова на си**

**Написано из предположения что на вход поступает целое число типа int.**

#include <stdio.h>

int a, b, c;

int main()

{

scanf("%d",&a);

b=0;

c=1;

while (b<16){

if (((a>>b)&1)=(a>>(31-b)&1)) c=0;

b++;

}

if (c==0) printf ("NOT PALINDROM\n");

if (c==1) printf ("PALINDROM\n");

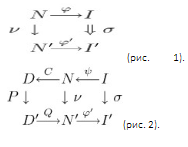
return 0;

}

**16 билет**

**1) обработка данных(информации)**

Во всех случаях, когда правило обработки σ является отображением, правило обработки сообщений v называется сохраняющим информацию. Если v сохраняет информацию, то диаграмма коммутативна: v °φ=φ°σ . Отображение σ называется правилом обработки информации. Если σ обратимое отображение, т.е. информация при обработке не теряется, то соответствующая обработка называется дешифровкой. Пусть v – обратимая перешифровка. Тогда обработка обратимой перешифровкой называется перекодировкой. Иначе она называются сжимающей. Т.е. в результате необратимой перешифровки сообщений их кол-во уменьшается, а информация может либо сохраняться, либо теряться.



**2) теорема бойма якопини миллса**

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

Доказательство:

Пусть машина T имеет алфавит А = {a1, a2, …, ap} и множество состояний Q = {q0, q1, …, qs}. Тогда алфавит машины T’ имеет еще (p+1)(s+1) знаков bij. Каждой нетерминальной команде Т сопоставляем машину Bij:

1.Машина перезаписи буквы

2. Машина движения.

3. Стоп машина

Затем составляем диаграмму T’.

Следствие: Любая диаграмма может быть преобразована к виду, не использующему произвольные передачи управления - стрелки, соединяющие элементы диаграммы неструкт. способами.

**3) написать нам переводящий число в двоичной системе счисления в восьмеричную**

Решение:

a1->1a

a0->0a

a->b

000b->b0

001b->b1

010b->b2

011b->b3

100b->b4

101b->b5

110b->b6

111b->b7

10b->b2

11b->b3

01b->b1

1b->b1

0b->b

b=>

->a

**17 билет**

**1)Конструктивное описание процесса обработки дискретных сообщений**

Обработка (дискретных) сообщений состоит в применении отображения (правила обработки) и , или в последовательном применении отображений С, P и Q . Чтобы отображение P могло служить основой для автоматизации обработки сообщений, оно должно задавать некоторый способ построения сообщения d' = P(d), исходя из сообщения d G D. Если D — конечное множество, то отображение P может быть задано таблицей, в которой перечисляются все сообщения d G D и соответствующие им сообщения P(d). Примером такой таблицы может служить таблица умножения (или таблица сложения) однозначных чисел в позиционной системе счисления. Если же D бесконечно или по крайней мере так велико, что задание P с помощью таблицы оказывается непрактичным, то нужно представить P в виде последовательности элементарных шагов обработки (или тактов), каждый из которых состоит в выполнении одного или нескольких достаточно простых отображений, называемых операциями: P = Р\ о Р2 о ... о Рк. Примеры операций: переписывание (копирование) буквы или слова, приписывание буквы к слову, приписывание слова к другому слову. Отметим, что любое отображение, допускающее описание с помощью достаточно простой таблицы, может быть объявлено операцией. Конечно же, оператор языка программирования высокого уровня, машинная команда, директива или системный вызов операционной системы также удовлетворяют критерию элементарности шагов обработки.

Наверное, интерпретация дискретных сообщений.

**2)Тип целый**

(integer, long int и пр). Значениями машинного целого типа являются все  
математические целые числа, заключенные между двумя фиксированными значениями Min  
и Max, являющимися атрибутами конкретного целого типа (разные границы у integer и long  
long). Формульно: Z={z∈Z|MIN ≤ z≤ MAX}∪{Null, передопределенное значение }  
Над такими типами можно проводить следующие операции: +,-,\*,/,%.

**Некоторые свойства операций и отношений:**

**1.** коммутативность сложения и умножения:УХ,К 6 Z X + Y = Y + X ,X \* Y = Y \* X ;

**2.** если X > Y > 0 или X < Y < 0, то (X - Y) + Y = X;

**3.** монотонность операций: VX, Y £ Z из 0 < X < А и 0 < Y < В следует X + Y < A + B-, X - B < A - Y - X \*Y < A \*B ; X + B < A + Y; Всё это верно только если А + В < М А Х и А \* В < М А Х соответственно.

**4.** если хотя бы один из операндов операции целого типа имеет значение \_L или T, то результат операции тоже имеет значение \_L или T соответственно. Если один операнд двуместной операции имеет значение \_L, а другой T, то результат операции имеет значение \_L;

**5.** если один из операндов отношения целого типа имеет значение \_L или T, то результат имеет значение \_L. Строго говоря, это неопределенное значение принадлежит логическому типу.

**6.** Если один из операндов Null или inf (переполнение), то результат будет тоже Null или inf  
соответственно

**3)Составить ДТ по сложению двоичных чисел**

**Билет 18 (вроде)---**

**1) Необходимость формального определения алгоритма**

Основной недостаток неформального определения алгоритма является его расплывчатость, т.к. мы не знаем, что значит «Понимать и выполнять действия одинаково» и что значит «всем понятные и легко выполнимые действия». Поэтому нам нужно формализовать алгоритмы и его понятие. Формализация алгоритма реализуется с помощью построения алгоритмических моделей. Можно выделить три основных типа универсальных алгоритмических моделей: рекурсивные функции (понятия алгоритма связывается с вычислениями и числовыми функциями), машины Тьюринга (алгоритм представляется как описание процесса работы некоторой машины, способной выполнять лишь небольшое число весьма простых операций), нормальные алгоритмы Маркова (алгоритмы описываются как преобразования слов в произвольных алфавитах). Этими моделями мы формализуем понятие алгоритма и у нас остается гораздо меньше вариантов интерпретации одной модели.

**2) Вызов процедур и функций. Передача аргументов(нет вЗайчике)**

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. В заголовке процедуры пишется имя, количество и виды параметров на вход, тип результата, а также некоторые необязательные атрибуты программы, которых довольно много. В теле находится сам код этой процедуры в виде блока. При вызове подпрограммы на место формальных параметров подставляются фактические параметры, так что их значения или ссылки на них замещают формальные параметры в период данного вызова подпрограммы. При другом вызове на место этих же формальных параметров могут подставляться другие фактические параметры, что и обеспечивает динамизм и универсальность подпрограмм.

1)Передача по значению может быть охарактеризована как read-only. При передаче сначала выделяется память под значение, потом значение каждого параметра вычисляется в точке вызова подпрограммы и пересылаются в выделенную область памяти. 2)По результату. Вычисляется результат функции, а потом передается уже сам результат, а не как в передаче по значению (передается значение, а потом вычисляется результат). 3)По ссылке. При передаче параметров по ссылке передается адрес объекта и все обращения к параметру в подпрограмме происходят по этому адресу. При этом всякое присваивание значения внутри программы приводит к одновременному изменению значения во всей программе. 4)По имени. Он откладывает обработку параметров ровно до того момента, когда они действительно потребуются. Этот метод является самой мощной формой передачи параметров, наиболее опасной и неэффективной.

**3) ДТ: перевод числа из двоичной системы в натуральную**

**19 билет**

**1. Кодирование**

Кодом называется правило, описывающее отображение С=А → А’ алфавита (набора знаков) А на другой набор знаков А’ . Например, отображение восьмеричного числа 08→0002 ,78→1112 . При кодировании сообщения n получается новое сообщение C(n), которое содержит ту же информацию, что и n, но изменяется правило интерпретации сообщений. (Например – кодирование по Цезарю). Кодирование, при котором каждый образ является отдельным знаком, называется шифрованием. В симметричных системах шифрования операции кодирования и декодирования являются взаимно обратными функциями. Т.е. зная способ шифрования, действуя наоборот, расшифровать сообщения. Ассиметричное кодирование, когда один ключ используется для кодирования, второй для декодирования. Одностороннее шифрование, используемое для хранения паролей, не является кодированием. Для каждого шифрованного пароля получается множество вариантов дешифрованных.

**2. Обобщенная инструкция цикла**

**1)** Количество предохранителей, принимающих значения И может быть более одного

**2)** Если среди предохранителей инструкции нет ни одного, принимающего значение И, то выполнение инструкции заканчивается естественным образом без прекращения работы программы

**3)** Выполнение каждой охраняемой инструкции должно приводить к изменению аргументов предохранителей, в противном случае выполнение инструкции цикла может никогда не закончиться

**3. Написать НАМ построения машинного слова по заданной двоичной маске**

**Билет 20**

**1. Диаграммы Тьюринга**

Диаграммы машин Тьюринга (ДМТ). ДМТ могут быть представлены через МТ.

В ДМТ есть:

**1)** Машина копирования. Копирует слово слева от головки в конец сообщения, была создана для удобства нормирования программы.

**2)** Машина L – Двигает голову МТ влево до первого попавшегося пробелаhttps://docs.google.com/document/d/1k5RkTDkCRjpQYFjLnR93uvJtn5Hyp-kGJKp-fhSTdys/edit#

**3)** Машина R – Двигает голову МТ вправо до первого попавшегося пробела

**4)** Элементарная l – двигает головку МТ влево на одну ячейку

**5)** Элементарная r – двигает головку МТ вправо на одну ячейку

**6)** Элементарная λ – производит запись буквы или пробела Надо заметить, что все машины – группа трех элементарных команд, которые были созданы для удобства программиста.

**2. Небазовые типы данных**

**Поскольку предусмотреть в языке программирования встроенных базовых типов на все** случаи жизни невозможно, обычно предоставляют возможность более или менее тривиального конструирования новых типов данных на основе базовых.

**1) Отрезок (диапазон)**

Можно было бы объявить отрезок типа, например, следующим образом: type diap = (0.0 .. 2.0; real; real); Здесь в качестве множества значений берётся отрезок вещественного типа, операции и отношения также заимствуются из вещественного типа.

**2) Перечисление**

В программе управления уличным движением удобно объявить type StreetLight = ((Red, Yellow, Green) ; := ; integer);

**3. Выдача байта на си**

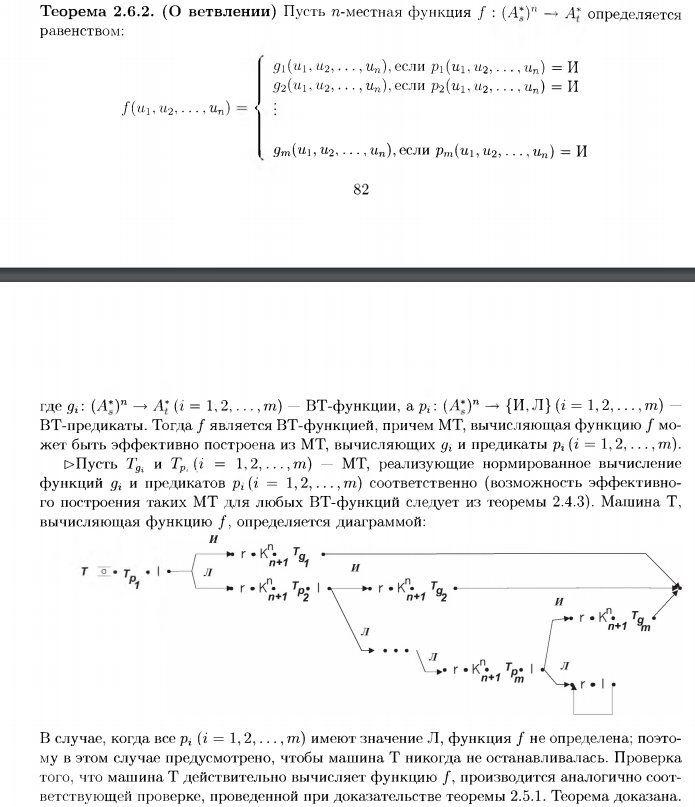
[#include](https://vk.com/im?sel=154279619&st=%23include) <stdio.h>   
  
int main(){   
  
int number,byte;   
printf("введите число и номер байта считая справа\n");   
scanf("%d %d",&number,&byte);   
while (byte < 1 || byte > 4) {   
printf("байт некорректно введен\n");   
scanf("%d",&byte);   
}   
number = number » (byte - 1) \* 8;   
for (int i = 7; i >= 0; —i) {   
printf("%d", (number » i) & 1u);   
}   
printf("\n");   
return 0;   
}

**21 билет:**

**1) Знаки и Символы**

Письменные языковые сообщения представляют из себя последовательности знаков. Мы будем различать атомарные и составные знаки. Атомарным знаком или буквой называется «неделимый» символ 1-ого уровня. Пример: буквы (А, Б, В….). Они складываются в слова второго уровня (слова). Знаки второго уровня в знаки третьего (предложения) и т.д. (абзацы и пр.). Все знаки большего уровня называются составными знаками. Множество слов также можно рассматривать как набор знаков. Для записи сообщения, которое содержит знаки k-го уровня необходимо иметь k-1 различных типов знаков разделителей. Есть специальный знак, чтобы разделять одни составные знаки от других. Для слов это пробел, для предложений – точка. С каждым знаком связывается его смысл или семантика. Знак вместе с сопоставленной ему семантикой называется символом. (А – буква, пи – символ).

**2) Теорема о ветвлении**

****

**3) НАМ реверса слова**

(A и В - любые (не обязательно различные) символы исходного слова отличные от $ ? #)

?|->.

?A -> >A

?B -> >B

>AA -> A>A

>AB -> B>A

>BA -> A>B

>BB -> B>B

>A| -> <|A

>B| -> <|B

>A -> <|A

>B -> <|B

A< -> <A

B< -> <B

< -> ?

->?

**Билет 22**

**1. Моделирование машины Тьюринга**

Рассмотрим две МТ Т=( A,Q ,P , qo ) и T =( A ,Q ,P , qo ) . Будем говорить что машина T моделирует машину Т и обозначать это T ≅Т , если выполняются следующие условия.

1)Указан способ кодирования знаков алфавита А знаками А`

2)Каждому состоянию q∋Q машины Т поставлено в соответствие некоторое состояние q ∋Q машины T`, т.е. определено отображение Q→Q

3)Если С0 – начальная конфигурация машины Т, то ее образ С0 - Начальная конфигурация машины T`

4)Если машина Е из начальной конфигурации С0 после конечного числа тактов останавливается на конфигурации Сk , то машина T` из начальной конфигурации С0 , являющейся образом С0 , после конечного числа тактов также останавливается на конфигурации Сk , являющейся образом Сk

5)Если из начальной конфигурации С0 машина Т пробегает последовательность конфигураций C0,C1 ,…,Ck ,…., то каждая последовательность конфигураций, пробегаемая машиной T` из начальной конфигурации С0 содержит в качестве подпоследовательности конфигураций Ci i=(0,1,…, k ,…)−¿ образы конфигураций Ci машины Т. Т.е. машина Т` описывает тот же алгоритм, что и Т, но, возможно, проходит при выполнении алгоритма большее число промежуточных конфигураций. Таким образом, понятие моделирования вводит бинарное отношение алгоритмического равенства между машинами Тьюринга.

**2. Понятие о структурном типе данных**

**Структурные значения – упорядоченные систематически организованные совокупности** других значений, рассматриваемых как единое целое. Компоненты структурных значений называются полями или элементами. Пример структурного типа – комплексный тип, состоящий из двух частей, вещественная и мнимая. Существует несколько методов структурирования:

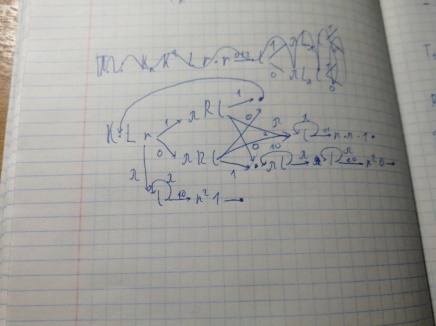
**1)** Регулярный содержит только компоненты базового типа.

**2)** Комбинированный содержит компоненты различных типов. Его различают на:

**2а)** Индексированный (все компоненты имеют свой индекс)

**2б)** Квалифицированный (компонент идентифицируется именем)

2в)Секвенциальный (все последовательно)

**3. Составить ДТ проверки палиндромии слова**

**Билет 23**

**1. Информация и сообщения. Интерпретация сообщения**

Информация и сообщение – основные (неопределяемые) понятия информатики. Информация передается посредством сообщения и наоборот, сообщение – то, что несет информацию. Информация может существовать только в форме некоторого сообщения. Соответствие между информацией и несущим ее сообщением не является взаимно однозначным:

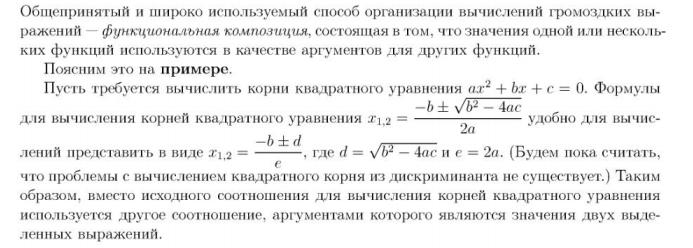
**1)** Одна и та же информация может передаваться с помощью различных сообщений;

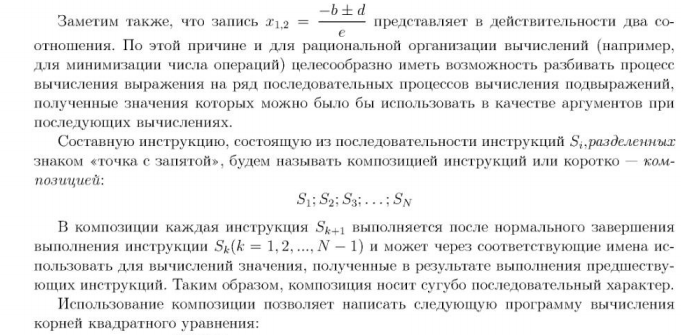
**2)** одно и то же сообщение может передавать различную информацию.

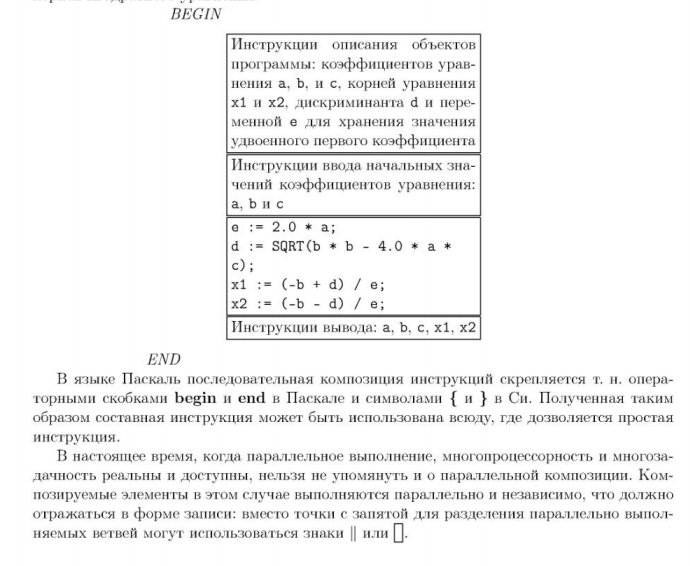
**Интерпретация сообщений:**

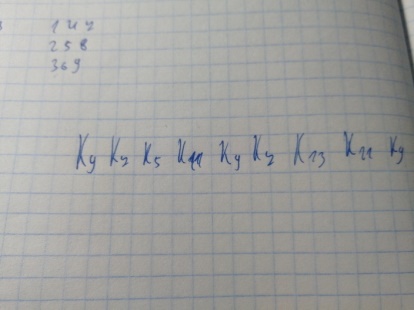
информация i, которая передается сообщением n, устанавливается с помощью правила интерпретации, которое представляет собой отображение рассматриваемого множества сообщений В в множество сведений. (Сведения – множество информации). φ=N→ I Есть конечные языки и бесконечные языки. Пример конечного языка – светофор. Зеленый, желтый и красный. Интерпретация этих сообщений как «Едь», «внимание» и «Стой» соответственно. Это φ конечного языка. Пример бесконечного языка – наш родной язык. Бесконечный он только теоретически, слов в этом языке «бесконечно» много, он постоянно расширяется и всегда можно придумать новые слова с новыми значениями.

**2. Обобщенная инструкция композиции**

****

****

****

**3. Транспонирование квадратной матрицы 3x3 в ДТ**

**Билет 25**

**1. Понятие файла**

Структура файла является обобщением понятия последовательности. Поэтому файлы следует считать «массивами на диске». Компоненты все одного тип и они доступны только путем последовательного прочтения, движения «назад» нет ввиду инерционности электромеханических устройств. Аналог – магнитная лента. Внешние файлы обычно перечисляются в заголовке программы. Они существуют до начала работы программы и/или сохраняются после окончания ее работы. Внутренние файлы, также как и внешние, описываются в программе как файловые переменные. Их время жизни совпадает с временем работы программы. Текстовые – в них только символы, юзается спец. Знак для конца строки. Нетекстовые файлы не предназначены для ввода-вывода и хранят данные непосредственно во внутримашинном представлении.

**2. Эквивалентность программ и диаграмм**

Каждой ДМТ, задающей программу Р, можно эффективным образом сопоставить МТ Т=(A,Q,P,qo) , образованную строками команд МТ так, чтобы МТ, определяющая эту диаграмму, смоделировала бы машину Т. Т.е. нам нужно указать способ эффективного построения программы по ДМТ программы Р, а потом убедиться. Что для исходной ДМТ и МТ, выполняются все пять условий моделирования МТ. Построение: Каждому элементу диаграммы .v. сопоставим строку команды Aэ

,Q, P,qo ,при этом нам нужно будет поставить вместо последней точки ДМТ заключающую строку программы МТ. Док-во, что оно эквивалентно. Все 5 условий моделирования должны быть выполнены. Условие 1 выполняется потому, что алфавиты МТ и ДМТ с°ххх 3-5 выполняются потому, что МТ определяет ту же последовательность элементарных действий, что и программа Р.

**3. Умножение многочленов на Си**

**26 билет:**

**1)Автоматическая обработка информации**

Чтобы выполнить автоматическую обработку информации, нужно располагать тремя физ. Представлениями

1)D – множество исходных данных

2) D - Множество результирующих данных

3)Обработка из D в D будет Р и оно будет выполняться на физ. Устройстве. Таким образом, для того, чтобы выполнить автоматическую обработку сообщений, необходимо автоматизировать выполнение трех отображений:

**1)** Отображения С, которое позволит представить сообщение N в множестве данных D

**2)** Отображения Р, которое переводит элемент D в D

**3)** Отображения декодирования Q,которое позволяет интерпретировать результат в элемент N`

**2) тип запись**

Это комбинированный структурный тип с квалифицированным методом доступа. Комбинированность означает, что поля записи имеют различные типы. Квалифицированный доступ гибок и невычислим, т.к. требует явного указания. С помощью него можно делать удобные и понятные, в частности погромисту, структуры.

**3)обратный код 16 числа нам**

\*0->F\*

\*1->E\*

\*2->D\*

\*3->C\*

\*4->B\*

\*5->A\*

\*6->9\*

\*7->8\*

\*8->7\*

\*9->6\*

\*A->5\*

\*B->4\*

\*C->3\*

\*D->2\*

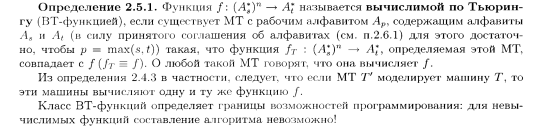
\*E->1\*

\*F->0\*

\*->.

->\*

**Билет 27:**

**1) вычислимые функции(а или б)**

**2) тип массива**

Массив – регулярный структурный тип с индексированным методом доступа. Т.е. все элементы одного типа и все они имеют индекс. Для массивов одного типа определена операция присваивания (A=B) без поиндексированного копирования каждой переменной. Обработка массива в основном осуществляется покомпонентно, с помощью циклов.

**3) НАМ сложение двух двоичных чисел**

1c->c0

0c->1

c->1

1a->a1

0a->a0

+a->a+

1b->b1

0b->b0

+b->b+

0#a->#1

0#b->#0

1#b->#1

1#a->c#0

\*1->1\*

\*0->0\*

\*+->#+\*

1\*->a\*

0\*->b\*

0#->#0

1#->#1

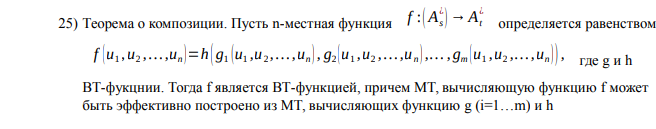
#->

+\*->.

->\*

**28 билет:**

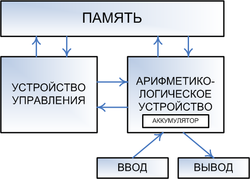
**1.Теорема о композиции**

****

**2. Модель фон Неймана.Имена и адреса. (стр 113)**

Модель фон Неймана широко известный принцип совместного хранения команд и данных.

Память разделена на ячейки, в которых хранятся двоичные данные и команды, именуемые единицами информации. Доступ к ячейке в памяти осущесвляется по адресу.



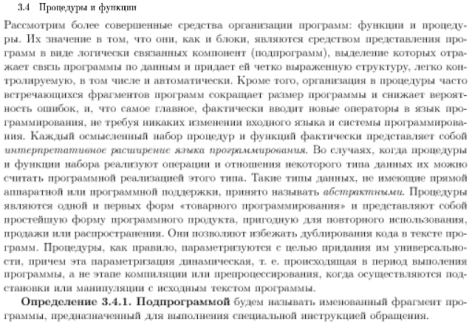
**3. Си: лексикографическая упорядоченность последовательности слов**

БЕЗ ПРОВЕРКИ СЛОВ, НАЧИНАЮЩИХСЯ НА ОДНУ И ТУ ЖЕ БУКВУ

#include <stdio.h>

#include <ctype.h>

#define OUT 0

****#define IN 1

unsigned int char\_to\_set(char c) {

c=tolower(c);

if (c< 'a'||c>'z')

return 0;

else return 1u<<(c - 'a');

}

int main () {

int alpha=1;

int c, state, sim;

state = OUT;

unsigned int min = char\_to\_set('a');

while ((c=getchar())!=EOF){

sim=char\_to\_set(c);

if (state==OUT&&sim!=0){

if(sim>=min) min = sim;

else {alpha=0;break;}

state = IN;

}

if (state==IN&&(c=='\n'||c=='\t'||c==' '||c==',')) state=OUT;

}

if (alpha==1) printf("Упорядочена");

if (alpha==0) printf("НЕ упорядочена");

}

**Билет 29**

**1. Теорема о ветвлении**

**2. Понятие типа данных**

Тип данных – множество изображений(Сам объект), для которых определено правило их интерпретации, позволяющее каждому изображению сопоставить его значение, и множество атрибутов(Что с этим объектом делать), которые позволяют одному или нескольким элементам типа данных сопоставить либо изображения данных того же типа, либо изображения данных другого типа.

**3. Инкремент 16-чного числа на НАМ**

\*0 -> 0\*

\*1 -> 1\*

\*2 -> 2\*

\*3 -> 3\*

\*4 -> 4\*

\*5 -> 5\*

\*6 -> 6\*

\*7 -> 7\*

\*8 -> 8\*

\*9 -> 9\*

\*a -> a\*

\*b -> b\*

\*c -> c\*

\*d -> d\*

\*e -> e\*

\*f -> f\*

\* -> @

0@ -> 1!

1@ -> 2!

2@ -> 3!

3@ -> 4!

4@ -> 5!

5@ -> 6!

6@ -> 7!

7@ -> 8!

8@ -> 9!

9@ -> a!

a@ -> b!

b@ -> c!

c@ -> d!

d@ -> e!

e@ -> f!

f@ -> @0

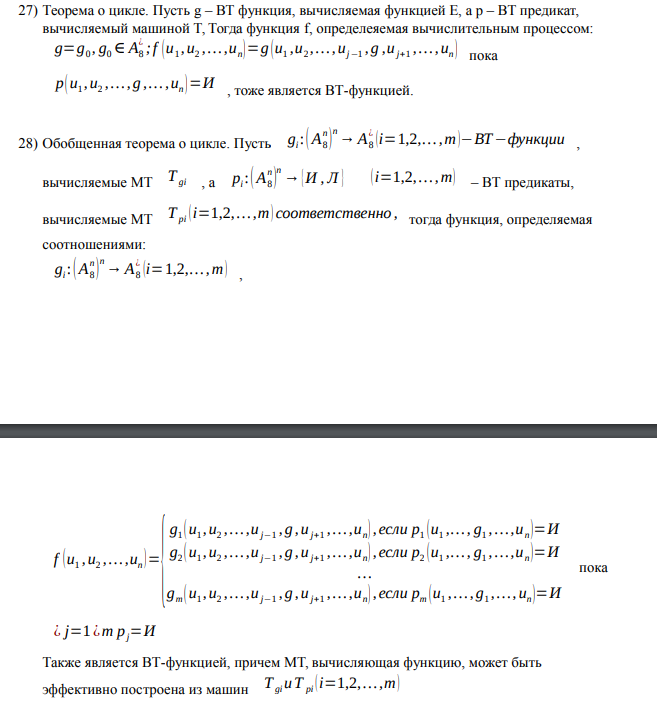
@ -> 1!

! ->.

-> \*

**30 билет**

**1)теорема о цикле**

**2) 2) структуры и функции объяснение (в методичке стр 185)**

Процедуры помогают избегать повторения кода, упрощают понимание программистом самой программы В заголовке процедуры пишется имя, количество и виды параметров на вход, тип результата В теле находится сам код этой процедуры в виде блока.

**3)написать на си конвертацию тьюринговской машины из пятерок в четверки**

Компилировать через gcc, на предупреждения внимание не обращать.

#include <stdio.h>

int main() {

int q0, q1;

char a, b, move;

while (scanf("%d,%c,%c,%c,%d", &q0, &a, &b, &move, &q1)) {

if ((int)move == (int)'u') {

printf("(%d,%c,%c,%d)\n", q0, a, b, q1);

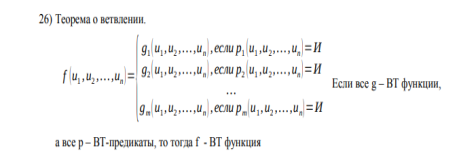
}

else if ((int)a == (int)b) {

printf("(%d,%c,%c,%d)\n", q0, a, move, q1);

}

else {

**** printf("(%d,%c,%c,%d)\n", q0, a, b, q0 + 100);

printf("(%d,%c,%c,%d)\n", q0 + 100, b, move, q1);

}

if ((int)move == (int)'s') {

break;

}

}

}

**31 билет**

**1. Схемы Тьюринга. Нисходящая разработка**

Нисходящая разработка состоит в следующем. Мы функцию f выражаем через f1,f2,..,fk и составляет диаграмму машины Т, включающую символы машины Ti (i=1…k). Каждую из функций fi выражаем через новые, более простые. И так до тех пор, пока на каком-то уровне не получатся диаграммы, включающие только символы элементарных МТ.

**2. Критика языков Паскаль и Си**

Язык высокого уровня - тип языка компьютерного программирования

**Паскаль** Недостатки этого замечательного языка являются продолжением его достоинств к нему можно отнести критику машины фон Неймана, так как он представляет собой ее реализацию скалярный оператор присваивания, строгая типизация содержит только языковые средства, которые эффективно компилируются на аппаратуру товарное программирование на Паскаль невозможно из-за отсутствия модульности и внешних процедур

**1)** Язык Паскаль представляет собой программно-компилируемую реализацию машины фон Неймана и к нему можно отнести всю критику этой алгоритмической модели. Скалярный оператор присваивания, через бутылочное горлышко которого надо прокачивать сложные математические объекты, заставляет программиста постоянно заботиться о рациональном использовании этой шины (:=) и отвлекает от решения самой задачи. Язык Паскаль является строго типизированным языком. Все объекты программ на Паскале должны быть описаны и употребляются в строгом соответствии с описаниями. Это предполагает строгую дисциплину программирования, неудобную в задачах системного программирования. Однако в своей строгости Паскаль непоследователен. Например, записи с вариантными частями образуют брешь в системе типизации. Тем не менее, особенности Паскаля таковы, что возможна реализация быстрого однопроходного эффективного компилятора. Языковая среда Паскаля также невелика, проста и эффективна. Паскаль содержит только такие языковые средства, которые эффективно компилируется на аппаратуру. Но это превращается в недостаток, потому что многие нужные любому программисту типы данных (такие как строки и массивы переменной длины) либо не реализованы вообще, либо их эффективная реализация проблематична. Товарное программирование на Паскале невозможно ввиду отсутствия модульности и внешних процедур. Паскаль проигрывает Си не только в выразительной силе и лаконичности. Библиотека языка Си представляет собой весьма мощную интерпретируемую компоненту.

**2)** Если по пунктам, то: **Си** Языки высокого уровня предназначены для выражения потребностей программиста, а не возможностей компьютера Они используют абстрактные данные и контролируют структуры, символические обозначения и переменные

1. **Примитивная поддержка модульности** 2. **Предупреждения вместо ошибок** 3. **Высокий порог вхождения** 4. **Отсутствие контроля инициализации переменных** 5. **Отсутствие контроля над адресной арифметикой** 6. **Динамически выделяемая память** 7. **Неудобные и небезопасные нуль-терминированные строки** 8. **Небезопасная реализация функций с переменным числом аргументов** 9. **Отсутствие унификации обработки ошибок**

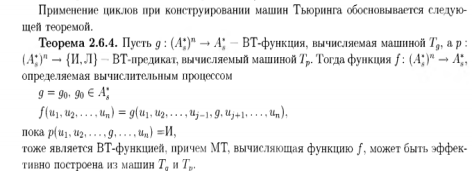
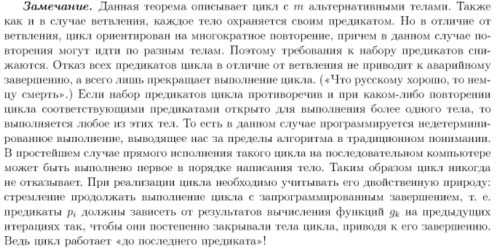
**3. Написать программу, выводящую график функции на Си**

**#include <stdio.h>  
#include <math.h>  
  
#define WIGTH\_FIELD 20 // ширина поля  
  
int main() {  
 int span = 0; // отступ  
 double real = 0.0;  
 for (double i = 0.0; i < 100; i += 0.4) {  
 real = (sin(i) > 0 ? 1 + sin(i) : 1 - (-sin(i))) / 2.0;  
 if (real == 0.0) {  
 for (int j = 0; j < WIGTH\_FIELD / 2; ++j)  
 printf(" ");  
 printf("\*\n");  
 } else {  
 span = WIGTH\_FIELD \* real;  
 for (int j = 0; j < span; ++j)  
 printf(" ");  
 printf("\*\n");  
 }  
 }  
 return 0;  
}**

**32 билет**

**1)Обобщенная теорема о цикле.**

Просто теорема о цикле

****это создание цикла, который при отказе всех предикатов всего лишь прекращает свое выполнение, а не включает аварийное завершение у этого цикла альтернативные тела если предикатами открыто для выполнения более одного тела, то выполняется любое из этих тел(обычно самое первое) таким образом, цикл никогда не отказывает но нужно следить за программируемым завершением то есть предикаты должны постепенно закрывать тело цикла, приводя его к завершению****

**Ы**

Модель фон Неймана пришла на замену модели МТ, дабы избавиться от недостатков предыдущей модели. К модели фон Неймана относят как аппаратные компьютеры, так и традиционные языки программирования. Их математические основания считаются сложными, громоздкими и концептуально бесполезными. Модель фон Неймана использует память и чувствительна к предыстории. Семантика также заключается в переходах из состояния в состояние, только они более сложные. Ясность программ гораздо выше. Наибольшая проблема модели фон Неймана – шина, которая за один раз может передавать только один определенный элемент памяти и является «Узким горлышком». Таким же горлышком в программировании является оператор присваивания, именно он вынуждает нас программировать на уровне «слово за слово».

**3) Написать на Си табуляцию многочлена и его производной на отрезке.**

#include <stdio.h>

#include <math.h>

typedef double(\*function\_pointer) (double);

double dichotomy\_method(function\_pointer function,double a,double b,double eps){

double x;

while(fabs(a-b)>=eps){

x=(a+b)/2;

if(function(a)\*function(x) > 0){

a = x;

}else{

b = x;

}

}

return x;

}

double iteration\_method(function\_pointer function,double a, double b, double eps){

double x = (a+b)/2.;

while(fabs(x-function(x))>=eps){

x = function(x);

}

return x;

}

double newton\_method(function\_pointer function, function\_pointer derivative, double a, double b,double eps){

double x=(a+b)/2.;

while(fabs(function(x)/derivative(x)) >= eps){

x-=function(x)/derivative(x);

}

return x;

}

double function\_1(double x){

return cos(x)-pow(exp(1),x\*x/-2.)+x-1.;

}

double function\_2(double x){

return 1-x+sin(x)-log(1.+x);

}

[#include](https://vk.com/im?sel=162450565&st=%23include) <stdio.h>  
  
int function(int mas[100],int n, int x){  
int i, j, z, s;  
for (i=0;i<=n;i++){  
if(i!=0){  
s=x;  
for(j=1;j<=i;j++){  
s=s\*s;  
}  
}else{  
s=1;  
}  
z=z+x\*mas[i];  
}  
return z;  
}  
  
int main(){  
int i,n, funct[100], proiz[100], min, max;  
scanf("%d",&n);  
scanf("%d",&min); scanf("%d",&max);  
for(i=n-1;i>=0;i--){  
scanf("%d",&funct[i]);  
if(i>1) scanf("%d",&proiz[i-1]);  
}  
printf("%d, %d, %d\n",n,min,max);  
for(i=min;i<=max;i++){  
printf("x=%d, funct=%d\n",i,function(funct,n-1,i));  
}  
printf("\n");  
for(i=min;i<=max;i++){  
printf("x=%d\n, proiz=%d\n",i,function(funct,n-2,i));  
}  
return 0;  
}