Московский авиационный институт
(национальный исследовательский университет)
Институт № 8 «Компьютерные науки и прикладная математика»

Курсовая работа
по курсу «Фундаментальная информатика»

1 семестр

Выполнила студентка группы М8О-104Б-23

Чечина Лилия Алексеевна

Преподаватель: Аносова Наталья Павловна

Москва, 2023

Содержание

ВВЕДЕНИЕ

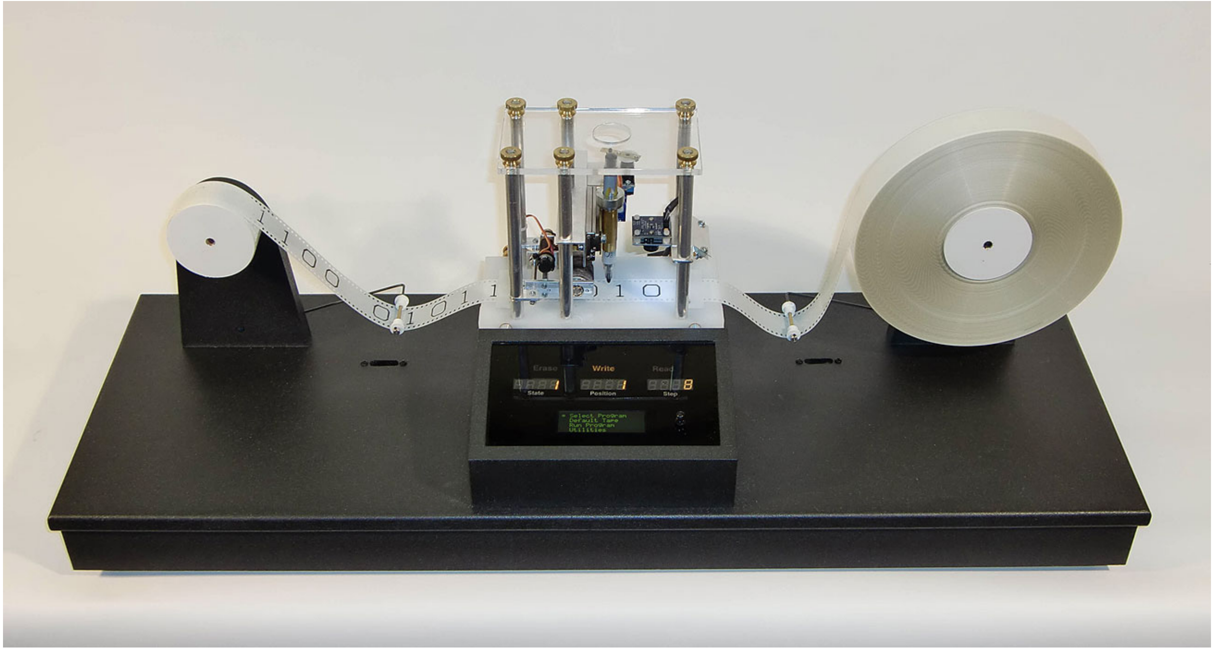
Цель курсовой работы: приобретение навыков разработки Машины Тьюринга, Диаграммера Тьюринга, разработки на языке программирования “Си”, понимание и применение численных методов.

Задача курсовой работы: составление эффективных алгоритмов решения поставленных задач.

2. МАШИНА ТЬЮРИНГА

Теория по Машине Тьюринга:

Машина Тьюринга – устройство, состоящее из ограниченной с одного конца ленты, разделённой на ячейки, и комбинированной читающей и пишущей головки, которая может перемещаться вдоль ленты от ячейки к ячейке.



В каждый момент времени головка располагается над некоторой ячейкой, называемой рабочей ячейкой, и находится в одном из конечного множества $Q = q$ дискретных состояний, среди которых выделено одно начальное состояние q_0. В зависимости от состояния головки и буквы в рабочей ячейки МТ выполняет одну из команд, составляющих её программу. Выполнение команды представляет из себя выполнение элементарного действия и перехода в новое состояние (которое может совпадать со старым). Определено три элементарных действия: сдвиг головки на одну ячейку влево (если лента ограничено слева, то данное действие не определено для крайней левой ячейки), сдвиг головки на одну ячейку вправо и запись в рабочую ячейку символа из рабочего алфавита МТ, либо пробела (λ). Перед началом работы МТ на её ленту записывается исходное сообщение так, что в каждой ячейке записывается либо одна буква сообщения, либо пробел. Начальной ячейкой становится ячейка, расположенная после конца исходного сообщения.

Постановка задачи: Выделение разрядов первого двоичного числа по маске, заданной вторым числом.

Идея решения:

1) Двигаемся влево к правому числу, проходов у нас будет столько, сколько разрядов во втором (правом) числе.
2) Если в правом числе “1” — в левом заменяем “0” на “a”. Если в правом числе “0”, то в левом меняем “0” на “.”. Если в правом числе “1”, то в левом числе меняем “1” на b.
3) После заменяем “a” на “0”, “b” на “1”, а “.” на λ. Смещаем левую часть до правой.
4) Ответ готов.

Код программы:

```
{
   "alphabet": [
      "0",
      "1",
      "a",
      "b",
      "."
   ],
   "states": {
      "q0": {
         "0": "0 L q1",
         "1": "λ L q0",
         "comment": "",
         "a": "N",
         "b": "N",
         ".": "N",
         "λ": "L"
      },
      "q1": {
```

"0": "N",
"1": "L",
"comment": "",
"a": "N",
"b": "N",
".": "N",
"λ": "λ L q2"
},
"q2": {
"0": "a R q2",
"1": "b R q2",
"comment": "",
"a": "N",
"b": "N",
".": "N",
"λ": "λ R q3"
},
"q3": {
"0": "λ L q4",
"1": "R",
"comment": "",
"a": "N",
"b": "N",
".": "N",
"λ": "N"
},
"q4": {
"0": "N",

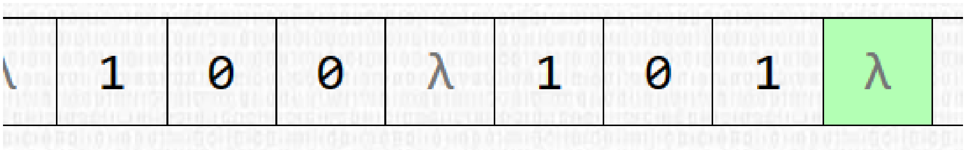
"1": "L",
"comment": "",
"a": "a L q5",
"b": "b L q5",
".": "N",
"λ": "L"
},
"q5": {
"0": ". R q5",
"1": ". R q5",
"comment": "",
"a": "R",
"b": "R",
".": "N",
"λ": "λ R q6"
},
"q6": {
"0": "N",
"1": "λ L q6",
"comment": "",
"a": "L",
"b": "L",
".": ". L q7",
"λ": "L"
},
"q7": {
"0": "a R q7",
"1": "b R q7",

```
    "comment": "",
    "a": "N",
    "b": "N",
    ".": ". R q8",
    "λ": "N"
},
"q8": {
    "0": "N",
    "1": "N",
    "comment": "",
    "a": "0 L q8",
    "b": "1 L q8",
    ".": "λ L q9",
    "λ": "N"
},
"q9": {
    "0": "N",
    "1": "N",
    "comment": "",
    "a": "0 N q10",
    "b": "1 N q10",
    ".": "N",
    "λ": "N"
},
"q10": {
    "0": "λ R q12",
    "1": "λ R q10",
    "comment": "",
```

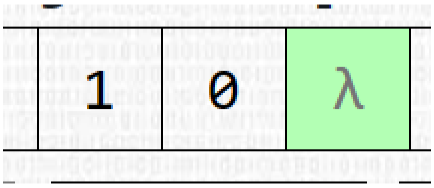
```
            "a": "N",
            "b": "N",
            ".": "N",
            "λ": "1 R q11"
        },
        "q11": {
            "0": "R",
            "1": "N",
            "comment": "",
            "a": "N",
            "b": "N",
            ".": "N",
            "λ": "N"
        },
        "q12": {
            "0": "N",
            "1": "R",
            "comment": "",
            "a": "N",
            "b": "N",
            ".": "N",
            "λ": "0 R q12"
        }
    },
    "word": ""
}
```


Тесты:

Входные данные:



Выходные данные:



Выводы: в ходе данной работы я научилась создавать программу на Машине Тьюринга.

3. ДИАГРАММЕР ТЬЮРИНГА

Теория:

Диаграмма МТ – визуально-топологический способ задания МТ через другие, более простые МТ, причём этот способ не менее полон и строг, чем “обычный”.

Элементарные МТ:

● МТ l (сдвиг головки на одну ячейку влево)

● МТ r (сдвиг головки на одну ячейку вправо)

● МТ λ, a (запись соответствующего знака на ленту

Дополнительные МТ:

● МТ $L_{\neq}$ (сдвиг влево до λ)

● МТ $R_{\neq}$ (сдвиг вправо до λ)

● МТ K (копирование предыдущего слова)

Обозначения:

● Состояние – точкоместо ·

● Символ МТ – ·Т·

● Соединение МТ – $T\,1 \cdot\!\!\longrightarrow T\,2$

● Развилка

● Повторение Упрощения записи диаграмм:

● Если над стрелкой МТ записаны все символы из алфавита A , то их $p \cup \{\lambda\}$ можно опустить

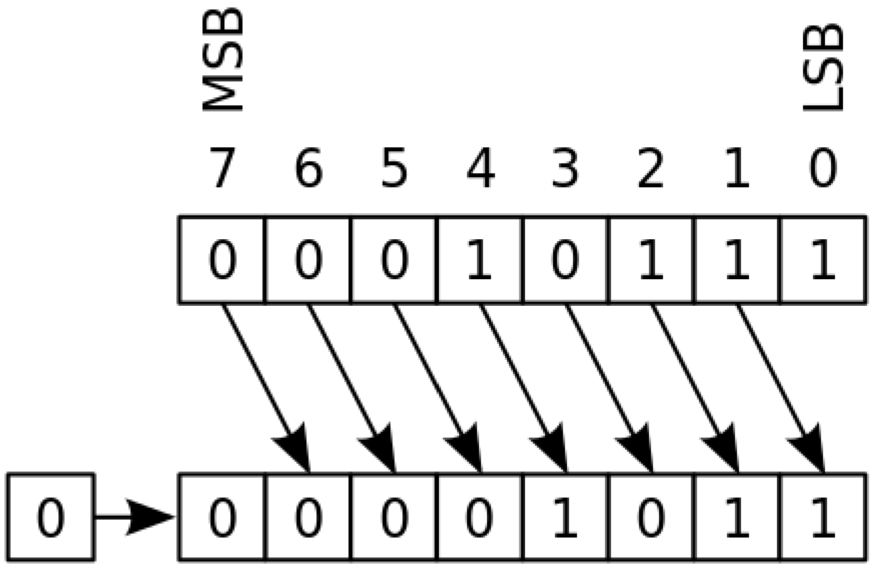
● Если стрелка, над которой ничего не написано, соединяет две соседние точки, то её можно опустить, слив точки в одну

● Если на диаграмме расположены подряд k символов одной МТ M, то их можно заменить одним символом М k

Постановка задачи: разработать диаграмму Тьюринга решения задачи в среде интерпретатора VTM или jdt (или VisualTuring 2.0!) с использованием стандартных машин (r, l, R, L, Kn, i a) и вспомогательных машин, определяемых поставленной задачей. Вычислить двоичный логический сдвиг первого числа вправо на число разрядов, равное второму.

Идея решения:

Если битовая последовательность 0001 0111 (десятичная 23) логически сдвинута на одну битовую позицию, то:



Сдвиг вправо дает: 0000 1011 (десятичная дробь 11).

1) Алфавит будет состоять из “0” и “1”.

2) Копируем правое число

3) Вычитаем число с помощью машины “sub”

3) Делаем проверку не является ли число нулевым с помощью доп. Машины “Check_zero

4) Зануляем скопированное 2 число

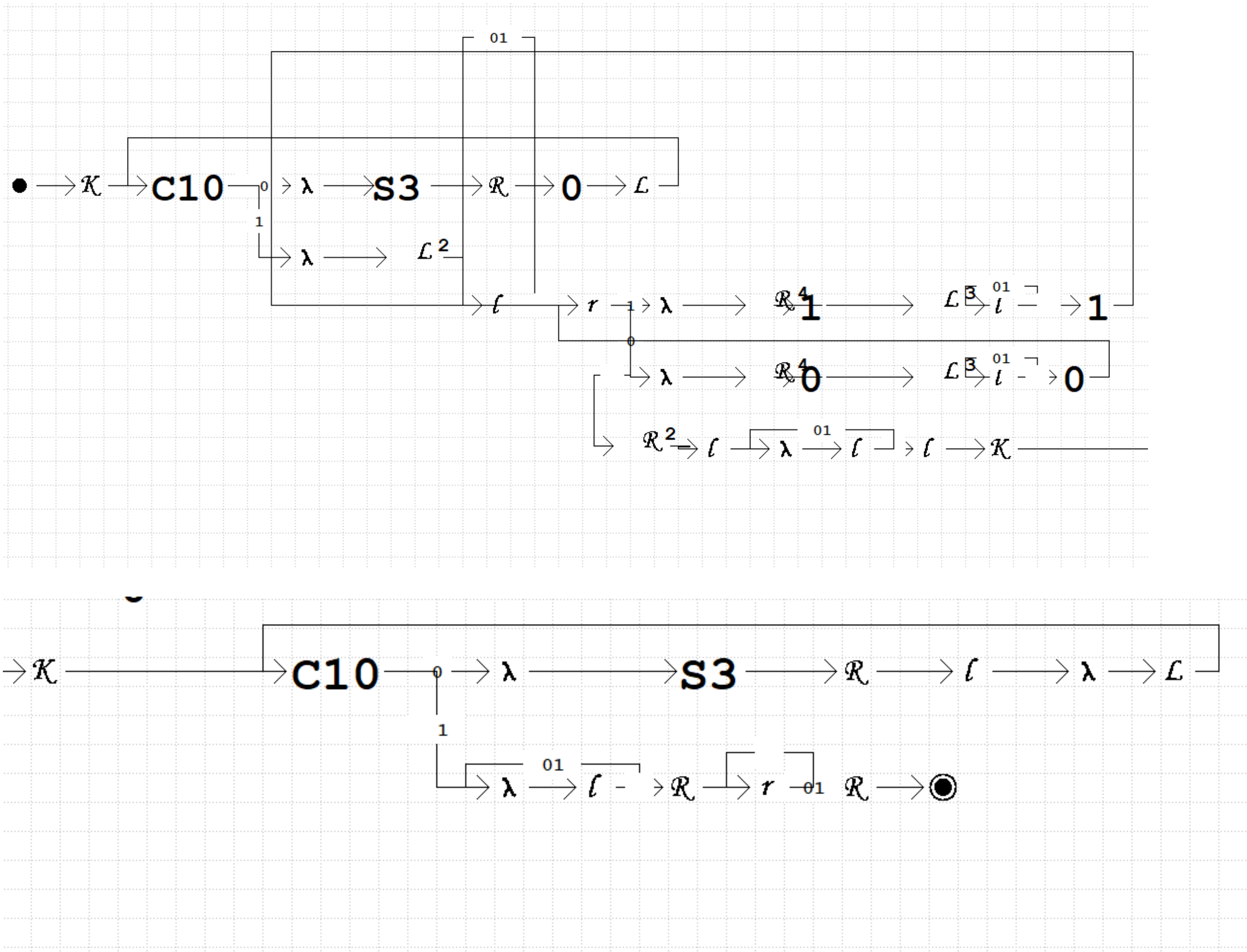
5) Справа делаем число из количества разрядов, равное первому

6) С первого числа переносим на наше новое часть правого первого числа в левую часть нового

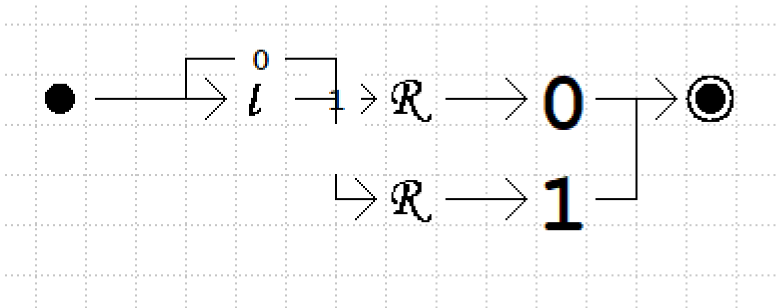
7) Смещаем влево и готово

Программа:

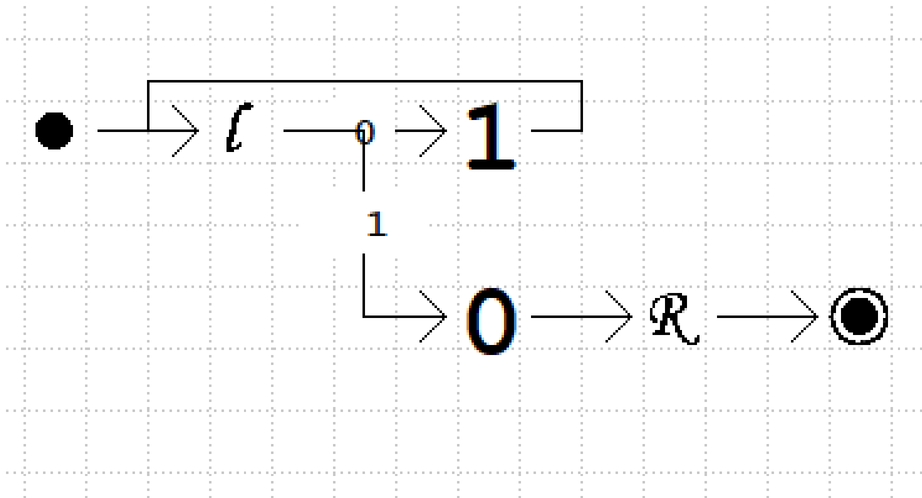
Main Mashine:



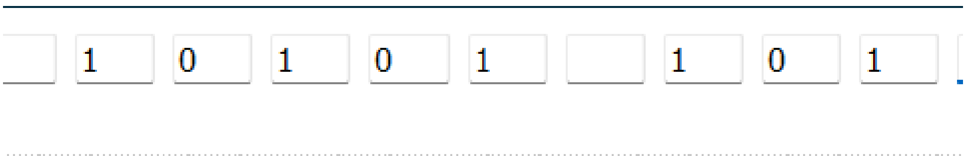
Check_zero

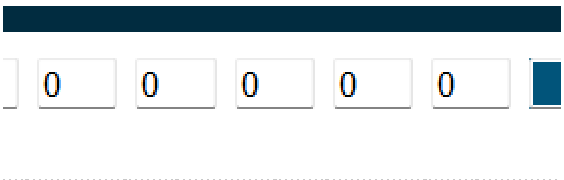


Sub



Тесты:





Выводы: в ходе данной работы мы освоили диаграммер Тьюринга, научились подключать к основной машине дополнительные.

4. НОРМАЛЬНЫЕ АЛГОРИТМЫ МАРКОВА

Теория:

Нормальные алгоритмы Маркова – это алгоритмическая модель, представляющая из себя упорядоченный набор правил-продукций (пар слов), соединённых между собой символами → или ↦. Каждая продукция является формулой замены части входного слова, совпадающего с левой частью формулы, на её правую часть. При этом и левая, и правая часть могут быть пустыми.

Пример:

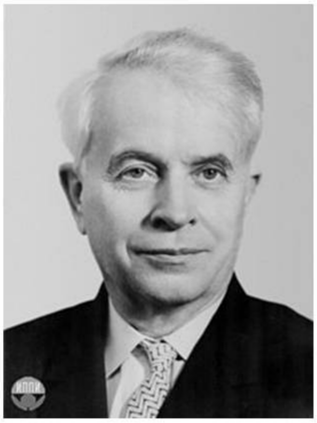
Алфавит из трех символов: 0, 1 и дополнительного символа | (палочки). Цепочка замен:

|0 → 0||
1 → 0|
0 →

Исходный объект: 101.

Работа алгоритма:

0|01
00||1
00||0|
00|0|||
000|||||
00|||||
0|||||
|||||



Андрей Марков мл.
(9 [22].09.1903 – 11.10.1979)

Процесс выполнения НАМ заканчивается в двух случаях:

1. все правила-продукции неприменимы к обрабатываемому слову
2. была применена терминальная продукция (со знаком ↦) Если в процессе выполнения НАМ нетерминальные правила выполняются бесконечно долго, то данный алгоритм неприменим к данному входному слову.

Приоритет правил НАМ:

1. Из нескольких применимых на данном шаге правил выбирается то, что встречено в описании алгоритма раньше

2. Из всех частей преобразуемого слова, к которым применимо правило, выбирается самое левое.

Для сокращения записи НАМ используются метасимволы, не входящие в алфавит алгоритма, но обозначающие любую букву из этого алфавита.

Постановка задачи:

Входное слово представляет собой два троичных числа без знака, разделённые знаком “^”. Следует обменять числа местами

Код программы:

```
0^ -> ^*a
1^ -> ^*b
2^ -> ^*c



*a0 -> 0*a
*a1 -> 1*a
*a2 -> 2*a

*b0 -> 0*b
*b1 -> 1*b
*b2 -> 2*b

*c0 -> 0*c
*c1 -> 1*c
*c2 -> 2*c

a* -> a
b* -> b
c* -> c
```

a -> 0

b -> 1

c -> 2

^ ->.

Тесты:

```
10210^1002
```

```
1002^10210
```

Выводы: в ходе работы мы овладели НАМ и способами решения задач с помощью него.

5. ВЕЩЕСТВЕННЫЙ ТИП. ПРИБЛИЖЁННЫЕ ВЫЧИСЛЕНИЯ. ТАБУЛИРОВАНИЕ ФУНКЦИЙ.

Теория:

Машинный эпсилон (англ. Machine epsilon) — числовое значение, меньше которого невозможно задавать относительную точность для любого алгоритма, возвращающего вещественные числа. Абсолютное значение «машинного эпсилон» зависит от разрядности сетки применяемой ЭВМ, типа (разрядности) используемых при расчетах чисел, и от принятой в конкретном трансляторе структуры представления вещественных чисел (количества бит, отводимых на мантиссу и на порядок).[2] Формально машинный эпсилон обычно определяют как минимальное из чисел ε, для которого 1+ε>1 при машинных расчетах с числами данного типа[3]. Альтернативное определение — максимальное ε, для которого справедливо равенство 1+ε=1.

Практическая важность машинного эпсилон связана с тем, что два (отличных от нуля) числа являются одинаковыми с точки зрения машинной арифметики, если их относительная разность по модулю меньше (при определении первого типа) или не превосходит (при определении второго типа) машинного эпсилон.

Постановка задачи:

Составить программу на СИ, которая печатает таблицу значений элементарной функции, вычисленной двумя способами: по формуле Тейлора и с помощью встроенных внутри функции языка программирования. В качестве аргумента в таблице взять точки разбиения отрезка [a, b] на n равных частей (n + 1 точка, включая концы отрезка), находящихся в рекомендованной области хорошей точности формулы Тейлора. Вычисления по формуле Тейлора проводить по экономной в сложностном смысле схеме с точностью ε*k, где ε — машинное эпсилон, аппаратнореализованного вещественного типа для данной ЭВМ, а k-экспериментально подбираемый коэффициент, обеспечивающий приемлемую сходимость. Число итераций должно ограничиваться сверху порядка 100. Программа должна сама определять машинное ε и обеспечивать корректные размеры и генерируемой таблицы.

РЯД	a	b	функция
$1+\frac{2x}{1!}+\ldots+\frac{(2x)^n}{n!}$	0.1	0.6	e^{2x}

Идея решения:

1) Подключаем заголовки «math.h» и «stdio.h»

2) Определяем функцию вычисления машинного эпсилон

3) Определяем функцию для вычисления члена ряда Тейлора

4) Определяем функцию для вычисления функции при помощи встроенных функций

5) Вычисляем машинное эпсилон и выводим

6) Печатаем таблицу аргументов функций, значений полученных средствами языка С и ряда Тейлора, количество итераций запрошенное машиной для вычисления значения функции

Программа:

```
#include <math.h>

#include <stdio.h>

// функция подсчета машинного эпсилон. Пока единица меньше Эпсилон +
1, делим эпсилон пополам и повторяем

long double mashine_epsilon() {
```

```
}

long double eps = 1;
while (1 < 1 + eps) {
    eps /= 2;
}
    return eps;
}


// функция, считающая значение функции в точке икс
long double function_value(long double x) {
    return (expl(2*x));  // expl даёт самую высокую точность
}


// функция подсчета факториала числа
int factorial(long long n) {
    long double ans = 1;
    for (long long i = 2; i <= n; ++i) {
        ans *= i;
    }
    return ans;
}
```

```
// функция, вычисляющая ряд Тейлора в точке икс
long double teilor_raw(long double x, int n) {
    long double v = pow(2*x, n);
    v /= (long double)factorial(n);
    return v;
}


// функция, выводящая таблицу с полученными значениями
void table(long double k, long double a, long double b, int steps, int max_iters)
{
    long double step = (b - a) / steps; // считаем количество шагов [длину
отрезка делим на количество разбиений(шагов)]
    long double eps = mashine_epsilon(); // eps = подсчитанного машинного
эпсилон
    printf("Машинный эпсилон равен %.20Lf\n", eps); // выводим машинный
эпсилон
    printf("
_______________________________________________________________________
___________________________\n");
```

```
    }

    printf("|x   |част. sum  ряда      |          значения функции  e^2x           |
число итераций |\n"); // выводим верхнюю часть таблицы

printf("|____|_____________________|________________________________
_____________|________________|\n");

    for (long double x = a; x < b + step; x += step) { // перебираем по шагам
нужное количество иксов, принадлежащих отрезку

        int n = 0;

        long double current_member = 1; // текущий член

        long double sum = 0; // сумма

    /* пока модуль текущего члена превосходит произведение машинного
эпсилон и коэффициента точности, и

    n не превосходит максимального значения итераций (100), или если n =
2 */

        while ((fabsl(current_member) > eps * k && n < max_iters) || n == 2) {

            current_member = teilor_raw(x, n); // присваиваем текущему члену
значение n-ного элемента ряда Тейлора в точке x

            sum += current_member; // прибавляем к сумме текущий член

            n++; // переходим к следующей итерации, увеличив n

        }

        printf("|%.2Lf|%.19Lf|%.43Lf|%3d            |\n", x, sum, function_value(x),
n); // выводим необходимые значения с нужной точностью

    }
```

```
printf("|____|______________________|________________________________
______________|________________|\n"); // закрываем строку таблицы
}
int main() {
   long double k = 1000000000000000000000; // коэффициент точности
   long double a = 0.1l; // левая границы отрезка
   long double b = 0.6l; // правая граница отрезка
   int steps; // количество разбиений

   printf("Количество разбиений отрезка ");
   scanf("%d", &steps);
   int max_iters = 100; // максимальное число итераций
   table(k, a, b, steps, max_iters); // вывод таблицы
}
```

6. ПРОЦЕДУРЫ И ФУНКЦИИ В КАЧЕСТВЕ ПАРАМЕТРОВ

Теория:

Краткие сведения из численных методов

Рассматривается уравнение вида $F(x)=0$. Предполагается, что функция $F(x)$ достаточно гладкая, монотонная на этом отрезке и существует единственный корень уравнения $x^* \in [a,b]$. На отрезке $[a,b]$ ищется приближенное решение x с точностью ε, т.е. такое, что $\left|x-x^*\right|<\varepsilon$.

При решении реальных задач, где поведение функции $F(x)$ неизвестно, сначала производят исследование функции (аналитическое, численное, или графическое (`gnuplot`, `MathLab`, `MathCAD`, `Maple`)) и т. н. отделение корней, т. е. разбивают область определения функции на отрезки монотонности, на каждом из которых имеется ровно один корень и выполняются другие условия применимости численных методов (гладкость). Различные численные методы предъявляют разные требования к функции $F(x)$, обладают различной скоростью сходимости и поведением.

В данном задании предлагается изучить и запрограммировать три простейших численных метода решения алгебраических уравнений и провести вычислительные эксперименты по определению корней уравнений на указанных в задании отрезках монотонности и, в качестве дополнительного упражнения, вне их.

1. Метод дихотомии (половинного деления).

Очевидно, что если на отрезке $[a,b]$ существует корень уравнения, то значения функции на концах отрезка имеют разные знаки: $F(a)\cdot F(b)<0$. Метод заключается в делении отрезка пополам и его сужении в два раза на каждом шаге итерационного процесса в зависимости от знака функции в середине отрезка.

Итерационный процесс строится следующим образом: за начальное приближение принимаются границы исходного отрезка $a^{(0)}=a$, $b^{(0)}=b$. Далее вычисления проводятся по формулам: $a^{(k+1)}=(a^{(k)}+b^{(k)})/2$, $b^{(k+1)}=b^{(k)}$, если $F(a^{(k)})\cdot F((a^{(k)}+b^{(k)})/2)>0$; или по формулам: $a^{(k+1)}=a^{(k)}$, $b^{(k+1)}=(a^{(k)}+b^{(k)})/2$, если $F(b^{(k)})\cdot F((a^{(k)}+b^{(k)})/2)>0$.

Процесс повторяется до тех пор, пока не будет выполнено условие окончания $\left|a^{(k)}-b^{(k)}\right|<\varepsilon$.

Приближенное значение корня к моменту окончания итерационного процесса получается следующим образом $x^* \approx (a^{(конечное)}+b^{(конечное)})/2$.

2. Метод итераций.

Идея метода заключается в замене исходного уравнения $F(x)=0$ уравнением вида $x=f(x)$.

Достаточное условие сходимости метода: $\left|f'(x)\right|<1, x\in[a,b]$. Это условие необходимо проверить перед началом решения задачи, так как функция $f(x)$ может быть выбрана неоднозначно, причем в случае неверного выбора указанной функции метод расходится.

Начальное приближение корня: $x^{(0)}=(a+b)/2$ (середина исходного отрезка).

Итерационный процесс: $x^{(k+1)}=f(x^{(k)})$.

Условие окончания: $\left|x^{(k)}-x^{(k-1)}\right|<\varepsilon$.

Приближенное значение корня: $x^* \approx x^{(конечное)}$.

3. Метод Ньютона.

Метод Ньютона является частным случаем метода итераций.

Условие сходимости метода: $\left|F(x)\cdot F''(x)\right|<\left(F'(x)\right)^2$ на отрезке $[a,b]$.

Итерационный процесс: $x^{(k+1)}=x^{(k)}-F(x^{(k)})/F'(x^{(k)})$.

Более совершенное с программистской точки зрения решение задачи может быть получено с помощью изучаемого в курсе «Языки программирования» (II семестр) процедурного типа данных. В этом случае различные уравнения и методы как переменные процедурного типа подставляются в качестве фактических параметров соответствующих подпрограмм. Решение задачи на языке Си, фактически базирующееся на указателях на функции, близко к этому.

Постановка задачи:

Составить программу на языке Си с процедурами решения трансцендентных алгебраических уравнений различными численными методами (итераций, Ньютона и половинного деления—дихотомии). Нелинейные уравнения оформить как параметры-функции, разрешив относительно неизвестный величины в случае необходимости. Применять каждую процедуру к решению двух уравнений, заданных двумя строками таблицы, начиная с варианта с заданным номером. Если метод неприменим, дать математическое обоснование графическую иллюстрацию, например, с использованием gnuplot.

20	$0{,}1x^2 - x\ln x = 0$	[1, 2]	Ньютона	1.1183
21	$\operatorname{tg} x - \frac{1}{3}\operatorname{tg}^3 x + \frac{1}{5}\operatorname{tg}^5 x - \frac{1}{3} = 0$	[0, 0.8]	дихотомии	0.3333

Идея решения:

1) Подключение библиотеки <stdio.h>, <math.h> и <float.h>.

2) Определяем функции, данные в задании, видоизменяем функции для определенных методов.

3) Машинный эпсилон

4) Реализуем функции для различных методов с помощью указателей.

5) Выводим значения, полученные тремя различными методами для 20 и 21 варианта

Программа:

```
#include <stdio.h>
#include <math.h>
#include <float.h>
double function_1(double x)
   {
```

```
   return ((0.1 * pow(x, 2)) - x * log(x));  /*Тут внимательно с аргументами
функций*/
   }
double function_1_iter(double x)
   {
    return  (0.1 * pow(x, 2)) - (x * log(x)) + x;  // прибавили слева и справа x
   }
double function_1_diff(double x)
   {
     return (1/5)*x - log(x) - 1; // производная функции
   }
double function_2(double x)
   {
    return (tan(x) - (1./3)*pow(tan(x), 3) + (1./5)*pow(tan(x), 5) - (1./3));
   }
double function_2_iter(double x)
   {
    return x - 0.5*function_2(x);
   }
double function_2_diff(double x)
   {
   return (cos(2*x)*pow(cos(x), 2)+pow(sin(x), 4))/pow(cos(x), 6);
   }
double dichotomy(double(*Func_main)(double), double a, double b)
   {
    double x;
    while (fabs(a - b) > DBL_EPSILON) { //пока значения концов отрезка
отличаются с заданной точностью
    x = (a + b) / 2.0;
```

```
    if ((*Func_main)(a) * (*Func_main)(x) < 0.0) //если значение функции на
концах отрезка разных знаков – обновление значений концов отрезка
        {b = x;}
    else{
    a = x;}
    }
 return x;
}
double iteration(double(*func)(double), double a, double b)
{
 double x_cur = (a + b) / 2.0, x_last=b; //первое и последующие приближения
 //x_cur=a;
 while (fabs(x_cur - x_last) > DBL_EPSILON) { //пока значения отличаются
с заданной точностью, вычислять следующее приближение
 x_last = x_cur;
 x_cur = (*func)(x_last);
 }
 return x_cur;
}
double newton(double(*Func_main)(double), double (*func_diff)(double),
double a, double b)
{
 double x_cur = (a + b) / 2.0, x_last; //первое и последующие приближения
 while (fabs(x_cur - x_last) > DBL_EPSILON) { //пока значения отличаются
с заданной точностью, вычислять следующее приближение (с помощью
касательной)
 x_last = x_cur;
 x_cur -= (*Func_main)(x_last) / (*func_diff)(x_cur);
 }
```

```
 return x_cur;
}
int main() {
 printf("Для перого уравнения на отрезке [1, 2]:\n");
 printf("Методом диохтомии: %.10lf\n", dichotomy(function_1, 1.0,
2.0)); //вывод корня уравнения, вычисленного методом диохтомии для
варианта 20
 printf("Методом итераций: %.10lf\n", iteration(function_1_iter,
1.0, 2.0)); //вывод корня уравнения, вычисленного методом итераций для
варианта 20
 printf("Методом Ньютона: %.10lf\n", newton(function_1,
function_1_diff, 1.0, 2.0)); //вывод корня уравнения, вычисленного методом
Ньютона для варианта 20
printf("Для второго уравнения на отрезке [0, 0.8]:\n");
printf("Методом диохтомии: %.10lf\n", dichotomy(function_2, 0.0,
0.8)); //вывод корня уравнения, вычисленного методом диохтомии для
варианта 21
 printf("Методом итераций: %.10lf\n", iteration(function_2_iter,
0.0, 0.8)); //вывод корня уравнения, вычисленного методом итераций для
варианта 21
 printf("Методом Ньютона: %.10lf\n", newton(function_2,
function_2_diff, 0.0, 0.8)); //вывод корня уравнения, вычисленного методом
Ньтона для варианта 21
}
```

Тесты:

```
Для перого уравнения на отрезке [1, 2]:
Методом диохтомии: 1.1183255916
Методом итераций: 1.1183255916
Методом Ньютона: 1.1183255916
Для второго уравнения на �трезке [0, 0.8]:
Методом диохтомии: 0.3332554647
Методом ите�аций: 0.3332554647
Методом Ньютона: 0.3332554647
```

Вывод: в ходе данной работы я научилась использовать указатели для функции, овладела различными методами решения уравнения, которые мне пригодятся в дальнейшем обучении и работе.

7. Заключение

В ходе выполнения курсовой работы я познакомилась с важнейшими алгоритмическими моделями: Машина Тьюринга, Диаграммер Тьюринга, НАМ, которые необходимы для формального определения алгоритма. Также я познакомилась с численными методами и научилась улучшила свои качества разработки на Си.

8. Список используемых источников:

1. Машинный ноль — Википедия (wikipedia.org)

2. С. С. Гайсарян; В. Е. Зайцев - Курс информатики

3. В.Ю. Гидаспов; И.Э. Иванов; Д.Л. Ревизников; В.Ю. Стрельцов; В.Ф. Формалев - Чилсенные методы

4. Андрей андреевич марков презентация - 89 фото (triptonkosti.ru)

5. https://dzen.ru/a/YUjmEc6UNFgAX_LR