Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №4**

Дисциплина: «Дискретная математика и математическая

логика»

Тема: Полнота системы функций

Выполнил работу

студент группы ИВТ-22-2б

Казанцев А.В.

Проверила

Старший преподаватель кафедры ИТАС

Рустамханова Г.И.

Пермь 2023

**Цель работы**

1. Создание консольного приложения «Полнота системы функций»

2. Изучение теории дискретной математики об свойствах логических функций

3. Изучение и отработка навыков минимальных форм

**Постановка задачи**

1. Программа должен иметь возможность задавать систему функций состоящую минимум из трех функции длиной 8 символов и меньше
2. Программа должна выполнять следующие функции:
   1. Создание и вывод на экран таблицы истинности функций.
   2. Показать к каким классам Поста относятся функции системы
   3. Проверять систему на полноту

**Ход работы**

Программа имеет возможность открыть файл с подходящий имеем.  
программа считывает файл в массив строк. Строки нормализируются для корректной работы. Файл закрывается.

Программа позволяет выводить таблицы истинности функций системы.

Программа проверяет

Методы, используемые в программе:

ReadIntoFile(string path)– метод открывает и считывает файл в массив строк после форматирует вектора полученных функций . Возвращает массив строк длиной не превышающих 8 символов из «0» или «1».

WriteTruthTable(string vector) выводит на экран таблицу истинности.

bool CheckConstZero(string vector) – возвращает истину если функция принадлежит к классу содержащий константу «0».

bool CheckConstOne(string vector) – возвращает истину если функция принадлежит к классу содержащий константу «1».

bool CheckSelfsimilarity(string vector)– возвращает истину если функция принадлежит к классу самодвойственных.

bool CheckMonotony(string vector)– возвращает истину если функция принадлежит к классу монотонности.

bool CheckLinearity(string vector) возвращает истину если функция принадлежит к классу линейных. Проверяет полином Жегалкина полученный через метод неопределённых коэффициентов на произведения.

bool[,] GetTableClassPosta(string[] FunctionSystem) – составляет таблицу содержащую информацию о классах функций системы.

bool CheckFullness(bool[,] CoverTab)- возвращает истину если система функций является полной.

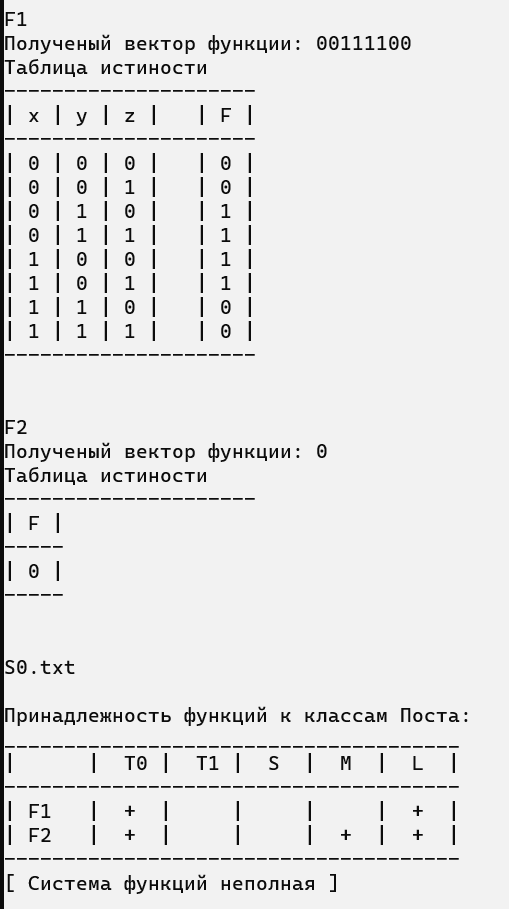
Дополнительные методы, используемые в программе:

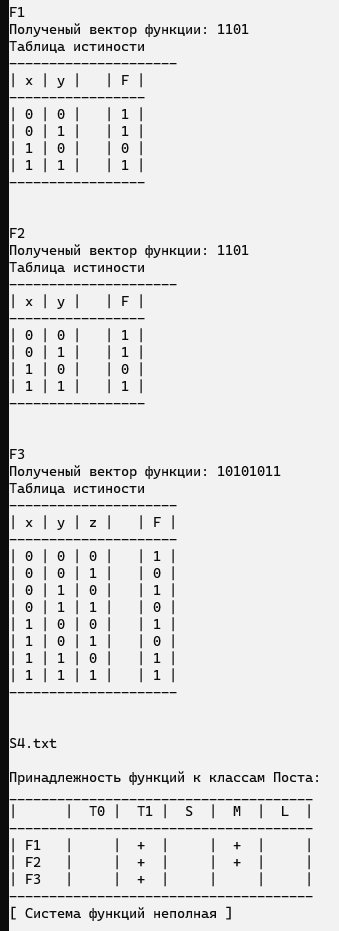
void CheckSystem(string path)- вызывает все методы для проверки системы на полноту. Выводит на экран таблицу классов Поста. Выводит таблицы истинности функций по команде пользователя.

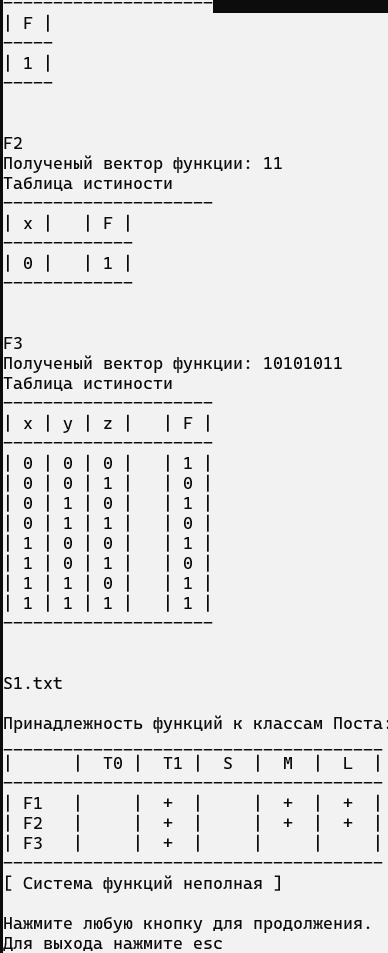
void ShowClassPosta(bool[,] CoverTab) – показывает таблицу с информацией о классах функций системы.

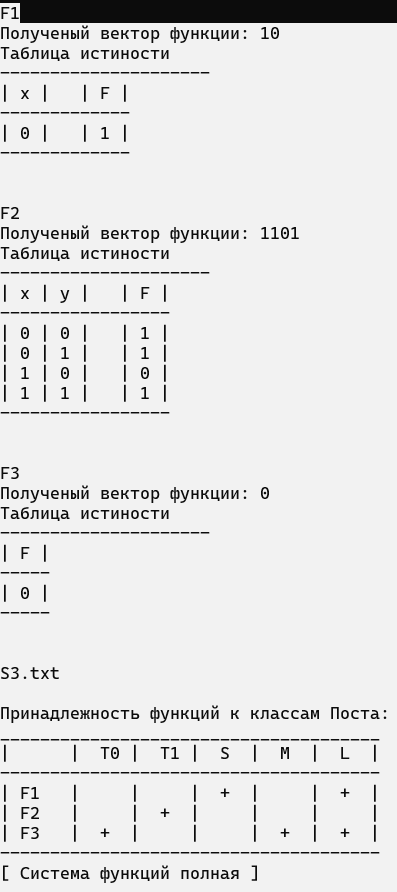
bool IsCheckGradeTwo(int number)- возвращает истину если число является степенью «2» и не больше 8.

Тесты







****

**Код программы**

using System;

using System.IO;

using System.Text.RegularExpressions;

internal class Program

{

private static void Main(string[] args)

{

Console.BackgroundColor = ConsoleColor.White;

Console.ForegroundColor = ConsoleColor.Black;

Console.Clear();

Console.WriteLine("Полнота системы функций");

CheckSystem("S1.txt");

Console.WriteLine("\n\n\n");

for (int i = 2;i<=4;i++)

{

string path = "S" + i.ToString() + ".txt";

CheckSystem(path);

Console.WriteLine("\n\n\n");

}

}

public static void CheckSystem(string path)

{

string[] FunctionSystem= ReadIntoFile(path);

if (FunctionSystem.Length == 3) { Console.WriteLine("Недостаточно функций в системе..."); return; }

Console.WriteLine("\nЕсли желаете отобразить таблицы истиности текущей системы нажмите Enter.\n");

if (Console.ReadKey(true).Key == ConsoleKey.Enter)

{

int i = 1;

foreach (var item in FunctionSystem)

{

Console.WriteLine("F" + (i++));

PrintTruthTable(item);

}

}

bool[,] ClassTab = GetTableClassPosta(FunctionSystem);

Console.WriteLine(path);

ShowClassPosta(ClassTab);

if (CheckFullness(ClassTab))

Console.WriteLine("[ Система функций полная ]");

else Console.WriteLine("[ Система функций неполная ]");

Console.WriteLine("\nНажмите любую кнопку для продолжения.\nДля выхода нажмите esc");

if (Console.ReadKey(true).Key == ConsoleKey.Escape) Environment.Exit(0);

}

public static bool CheckFullness(bool[,] CoverTab)

{

bool isCheck = true;

for (int i = 0; i < CoverTab.GetLength(1); i++)

{

isCheck = true;

for (int j = 0; j < CoverTab.GetLength(0)&&isCheck; j++)

{

isCheck = isCheck&&CoverTab[j,i];

}

if (isCheck) return false;

}

return true;

}

public static void ShowClassPosta(bool[,] CoverTab)

{

Console.Write("\nПринадлежность функций к классам Поста:\n" + new string('\_', 38));

Console.Write($"\n| | T0 | T1 | S | M | L |\n" + new string('-', 38));

for (int i = 0; i < CoverTab.GetLength(0); i++)

{

Console.Write($"\n| F" + (i + 1) + new string(' ', 4 - (i + 1).ToString().Length) + "|");

for (int j = 0; j < CoverTab.GetLength(1); j++)

{

if (CoverTab[i,j])

Console.Write($" + |");

else Console.Write($" |");

}

}

Console.WriteLine("\n" + new string('-', 38));

}

public static bool[,] GetTableClassPosta(string[] FunctionSystem)

{

bool[,] TableClass = new bool[FunctionSystem.Length, 5];

for (int i = 0; i < TableClass.GetLength(0); i++)

{

int j = 0;

TableClass[i, j++] = CheckConstZero(FunctionSystem[i]);

TableClass[i,j++]= CheckConstOne(FunctionSystem[i]);

TableClass[i,j++]= CheckSelfsimilarity(FunctionSystem[i]);

TableClass[i,j++]= CheckMonotony(FunctionSystem[i]);

TableClass[i, j] = CheckLinearity(FunctionSystem[i]);

}

return TableClass;

}

public static void PrintTruthTable(string vector)

{

Console.WriteLine("Полученый вектор функции: " + vector);

int i = 0;

int len = vector.Length;

Console.WriteLine($"Таблица истиности\n---------------------");

switch (len)

{

case 1:

Console.WriteLine($"| F | ");

Console.WriteLine($"-----");

Console.WriteLine($"| {vector[i++]} | ");

Console.WriteLine(new string('-', 5) + "\n\n");

return;

case 2:

Console.WriteLine($"| x | | F | ");

Console.WriteLine(new string('-', 13 + len - 2));

break;

case 4:

Console.WriteLine($"| x | y | | F | ");

Console.WriteLine(new string('-', 13 + len));

break;

case 8:

Console.WriteLine($"| x | y | z | | F | ");

Console.WriteLine(new string('-', 13 + len));

break;

}

for (int x = 0; x <= 1 && len >= 2; x++)

{

for (int y = 0; y <= 1 && len >= 4; y++)

{

for (int z = 0; z <= 1 && len == 8; z++)

{

Console.WriteLine($"| {x} | {y} | {z} | | {vector[i++]} | ");

}

if (len == 4)

{

Console.WriteLine($"| {x} | {y} | | {vector[i++]} | ");

}

}

if (len == 2)

{

Console.WriteLine($"| {x} | | {vector[i++]} | ");

Console.WriteLine(new string('-', 13 + len - 2) + "\n\n");

return;

}

}

Console.WriteLine(new string('-', 13 + len) + "\n\n");

}

public static bool CheckConstZero(string vector)

{

if (vector[0] == '0')

return true;

else return false;

}

public static bool CheckConstOne(string vector)

{

if (vector[vector.Length - 1] == '1')

return true;

else return false;

}

public static bool CheckSelfsimilarity(string vector)

{

if (vector.Length == 1) return false;

bool isCheck = true;

int len = vector.Length;

for (int i = 0; i < len / 2 && isCheck; i++)

isCheck = isCheck && vector[i] != vector[vector.Length - 1 - i];

return isCheck;

}

public static bool CheckMonotony(string vector)

{

if (vector.Length == 1) return true;

bool isCheck = true;

int len = vector.Length;

switch (len)

{

case 2:

if (vector[0] <= vector[1])

return true;

else return false;

case 4:

for (int i = 0; i < len && isCheck; i += 2)

{

isCheck = isCheck && vector[i] <= vector[i + 1];

}

isCheck = isCheck && vector.Substring(0, len / 2).CompareTo(vector.Substring(len / 2)) != -1;

return isCheck;

case 8:

for (int i = 0; i < len && isCheck; i += 2)

{

isCheck = isCheck && vector[i] <= vector[i + 1];

}

for (int i = 0; i < len && isCheck; i += 4)

{

isCheck = isCheck && vector.Substring(i, len / 4).CompareTo(vector.Substring(i + len / 4, len / 4)) != 1;

}

isCheck = isCheck && vector.Substring(0, len / 2).CompareTo(vector.Substring(len / 2, len / 2)) != 1;

return isCheck;

}

return false;

}

public static bool CheckLinearity(string vector)

{

if (vector.Length == 1 || vector.Length == 2) return true;

int len = vector.Length;

int[] polinom = new int[len];

switch (len)

{

case 4:

polinom[0] = int.Parse(vector[0].ToString());

polinom[1] = int.Parse(vector[2].ToString()) == 0 ? polinom[0] : polinom[0] == 0 ? 1 : 0;

polinom[2] = int.Parse(vector[1].ToString()) == 0 ? polinom[0] : polinom[0] == 0 ? 1 : 0;

polinom[3] = int.Parse(vector[3].ToString()) == 0 ? (polinom[0] + polinom[1] + polinom[2]) % 2 == 0 ? 0 : 1 : (polinom[0] + polinom[1] + polinom[2]) % 2 == 1 ? 0 : 1;

if (polinom[3] == 1)

return false;

else return true;

case 8:

polinom[0] = int.Parse(vector[0].ToString());

polinom[1] = int.Parse(vector[4].ToString()) == 0 ? polinom[0] : polinom[0] == 0 ? 1 : 0;

polinom[2] = int.Parse(vector[2].ToString()) == 0 ? polinom[0] : polinom[0] == 0 ? 1 : 0;

polinom[3] = int.Parse(vector[1].ToString()) == 0 ? polinom[0] : polinom[0] == 0 ? 1 : 0;

polinom[4] = int.Parse(vector[6].ToString()) == 0 ? (polinom[0] + polinom[1] + polinom[2]) % 2 == 0 ? 0 : 1 : (polinom[0] + polinom[1] + polinom[2]) % 2 == 1 ? 0 : 1; if (polinom[4] == 1) return false;

polinom[5] = int.Parse(vector[5].ToString()) == 0 ? (polinom[0] + polinom[1] + polinom[3]) % 2 == 0 ? 0 : 1 : (polinom[0] + polinom[1] + polinom[3]) % 2 == 1 ? 0 : 1; if (polinom[5] == 1) return false;

polinom[6] = int.Parse(vector[3].ToString()) == 0 ? (polinom[0] + polinom[2] + polinom[3]) % 2 == 0 ? 0 : 1 : (polinom[0] + polinom[2] + polinom[3]) % 2 == 1 ? 0 : 1; if (polinom[6] == 1) return false;

polinom[7] = int.Parse(vector[7].ToString()) == 0 ? (polinom[0] + polinom[1] + polinom[2] + polinom[3] + polinom[4] + polinom[5] + polinom[6] + polinom[7]) % 2 == 0 ? 0 : 1 : (polinom[0] + polinom[1] + polinom[2] + polinom[3] + polinom[4] + polinom[5] + polinom[6] + polinom[7]) % 2 == 1 ? 0 : 1;

if (polinom[7] == 1) return false;

else return true;

}

return false;

}

public static string[] ReadIntoFile(string path)

{

string[] lines = File.ReadAllLines(path);

for (int i = 0; i < lines.Length; i++)

{

string vector = Regex.Replace(lines[i], "[^01]", "");

while (!IsCheckGradeTwo(vector.Length))

{

if (vector.Length == 0) { vector = "0"; break; }

vector = vector.Remove(vector.Length - 1);

}

lines[i] = vector;

}

return lines;

}

public static bool IsCheckGradeTwo(int number)

{

if (number == 0) return false;

string s = Convert.ToString(number, 2);

if (Regex.IsMatch(s, "^\\b1{1}0{0,3}$"))

return true;

else return false;

}

}