Штыкова Анна

А-13-18

Лабораторная работа №3

**НИУ МЭИ**

1. **Постановка задачи**

Разработать алгоритм и создать программу, реализующую построение таблицы истинности для логических функций четырёх аргументов.

1. **Состав данных**

*Описание всех функций, входящих в программу:*

|  |  |  |  |
| --- | --- | --- | --- |
| **Функция** | **Принимает** | **Возвращает** | **Смысл** |
| Evaluate | string& s1 - первая строка  string& s2  char oper | string | Функция, возвращающая результат операции |
| Negation | string& s | string | Выполняется операция отрицания |
| IsArgum | char c | bool | Является ли текущий символ переменной |
| IsHighPriority | const char& c1  const char& c2 | bool | Выше ли приоритет у одной операции по сравнению с другой операцией |
| Calculate | string& s | string | **функция, которая принимает выражение и возвращает результат** |
| IsCorrect | string& s | bool | Проверка на корректность исходного выражения |
| Output | ofstream& out, const string& s | **-** | Функция вывода в файл |

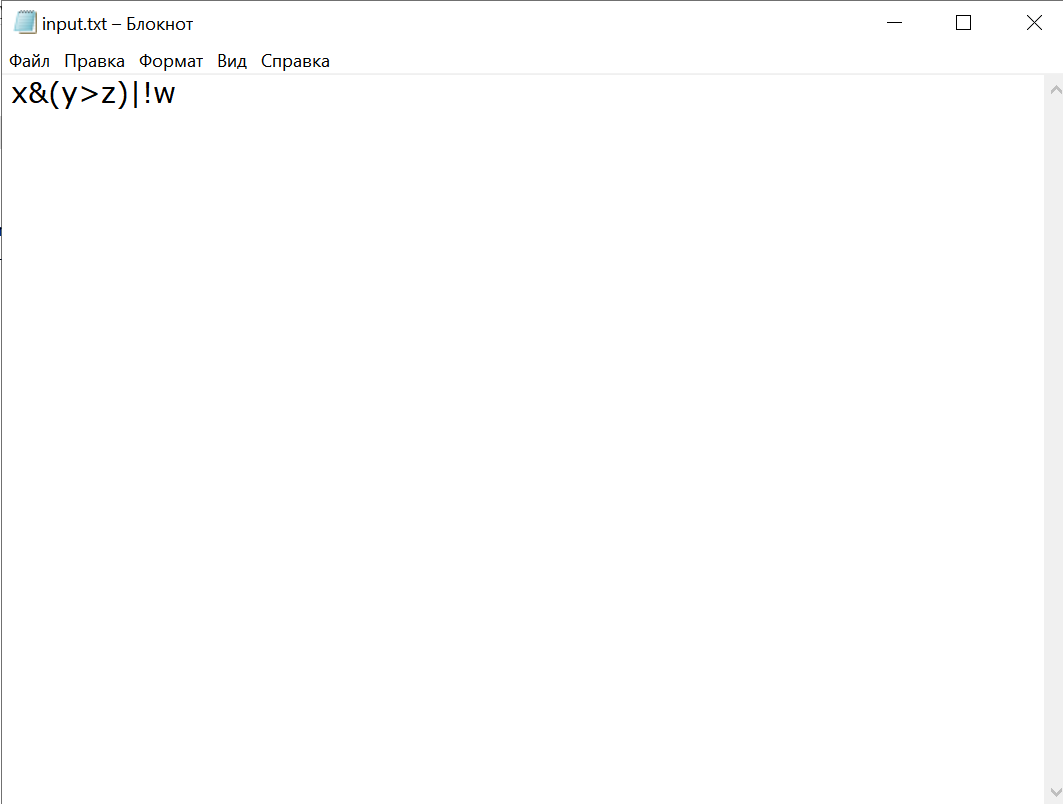
1. **Форма ввода**

Данные представляются в входном файле:

<Логическая функция> - логическая функция может содержать 4 переменные: x, y, z, w, открывающую скобку, закрывающую скобку, а также логические операции:

|  |  |
| --- | --- |
| ^ | Стрелка пирса |
| / | Штрих шеффера |
| > | Импликация |
| = | Эквивалентность |
| | | Дизъюнкция |
| & | Конъюнкция |
| ! | Отрицание |
| + | Сложение по модулю два |

Пример входных данных:



1. **Выходная форма**

Результат программы выводится в формате таблицы истинности для 4 переменных.

1. **Метод**

Введём приоритетность операций:

1. Отрицание

2. Конъюнкция

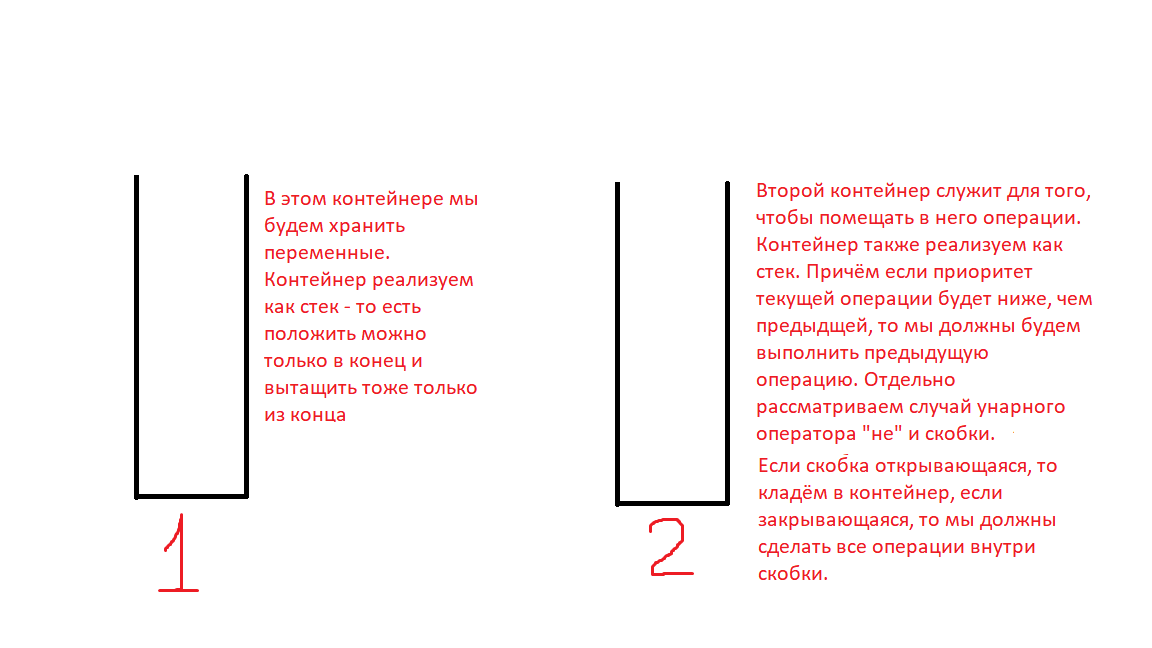
3. Дизъюнкция

4. Сложение по модулю два

5. Импликация

6. Эквивалентность

7. Штрих Шеффера, стрелка Пирса

Основой метода является создание двух контейнер, которые будут работать по принципу стеков. Этот алгоритм реализован в функции ***Calculate.* **

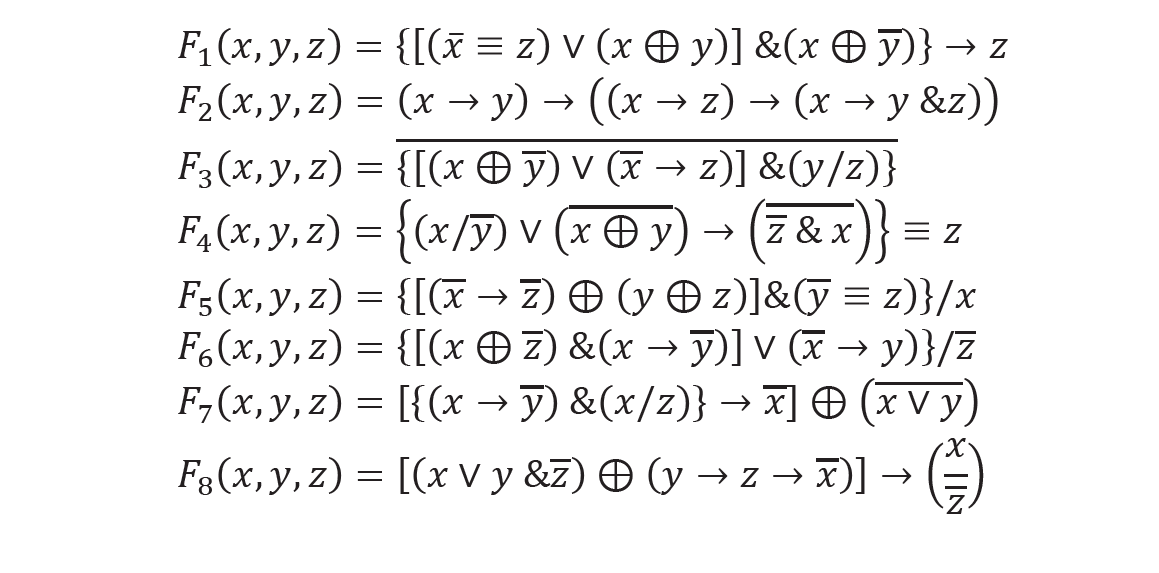
1. **Аномалии**

а) Неправильный формат записи выражения в текстовый файл

б) Нет данных в текстовом файле

1. **Тесты**

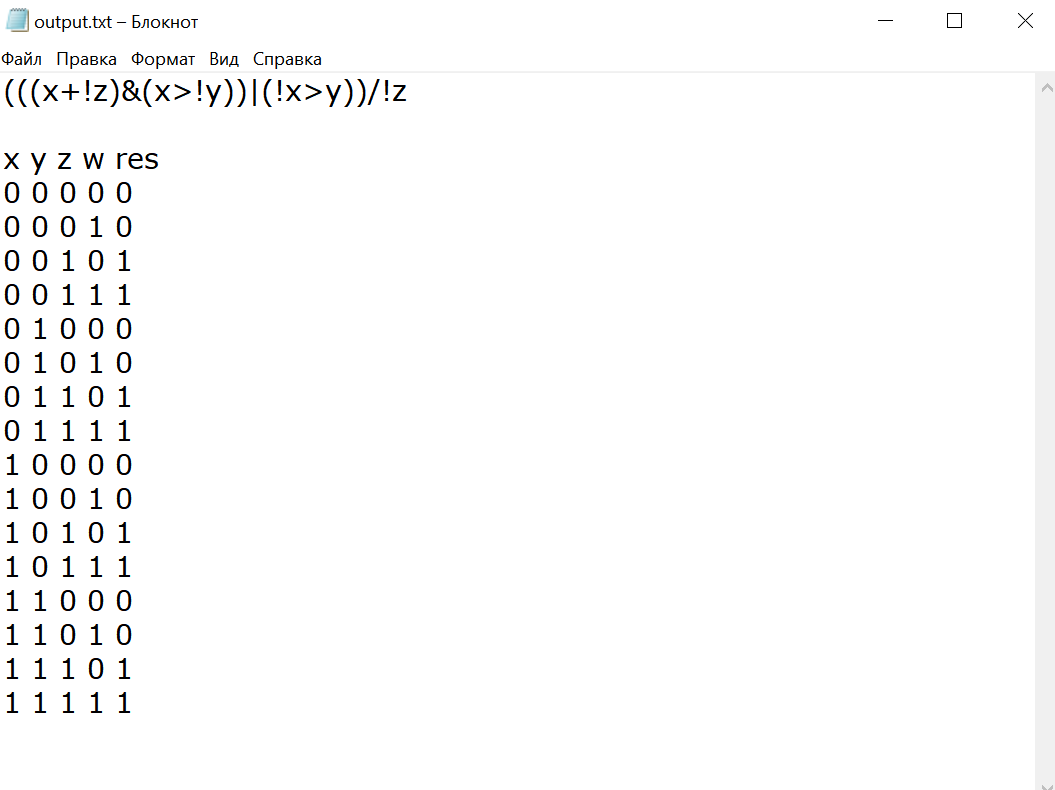
В качестве тестов возьмём приведённые функции из практического занятия 3.

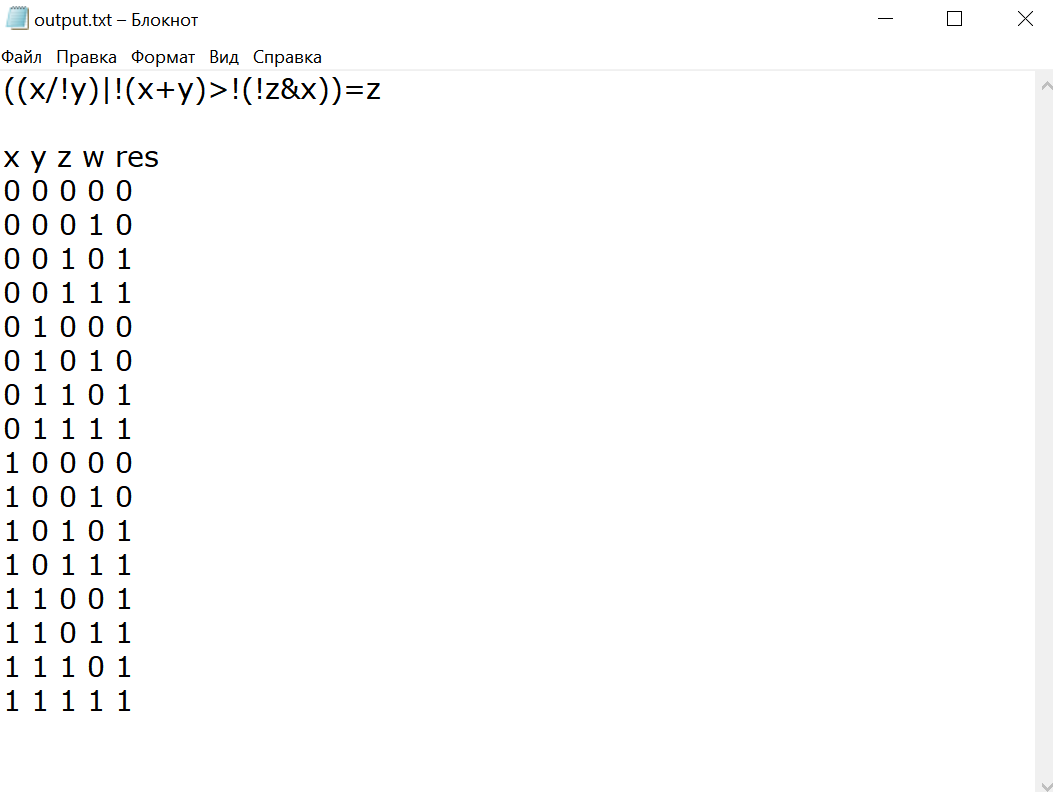
****

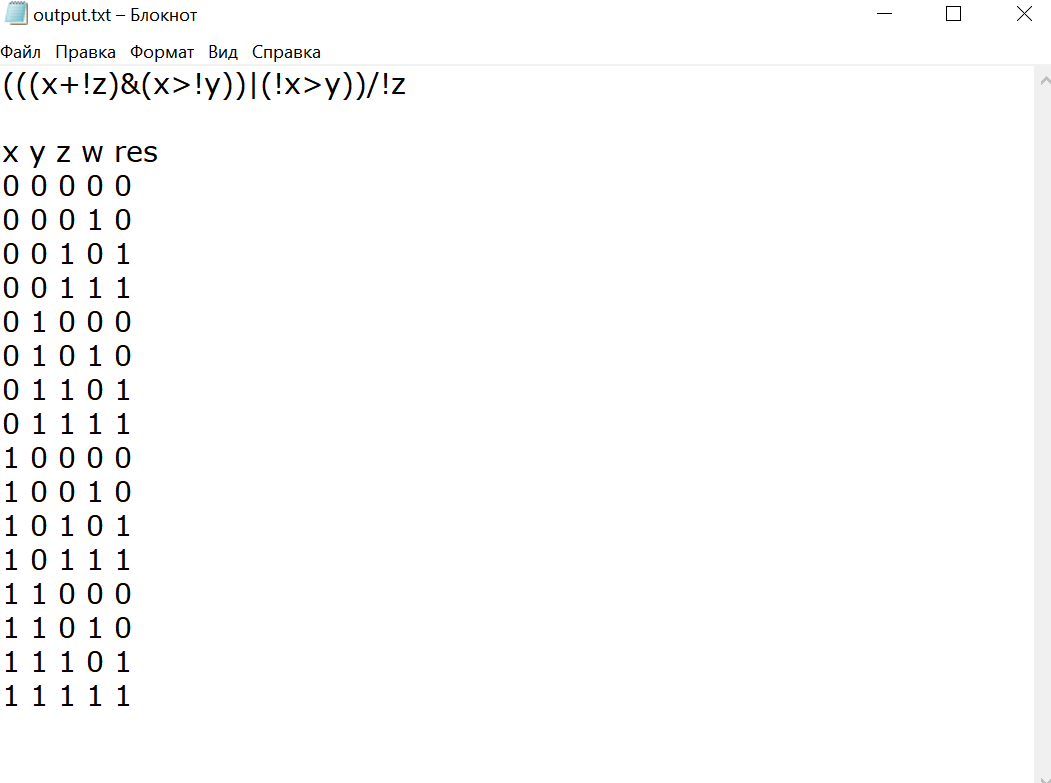
****

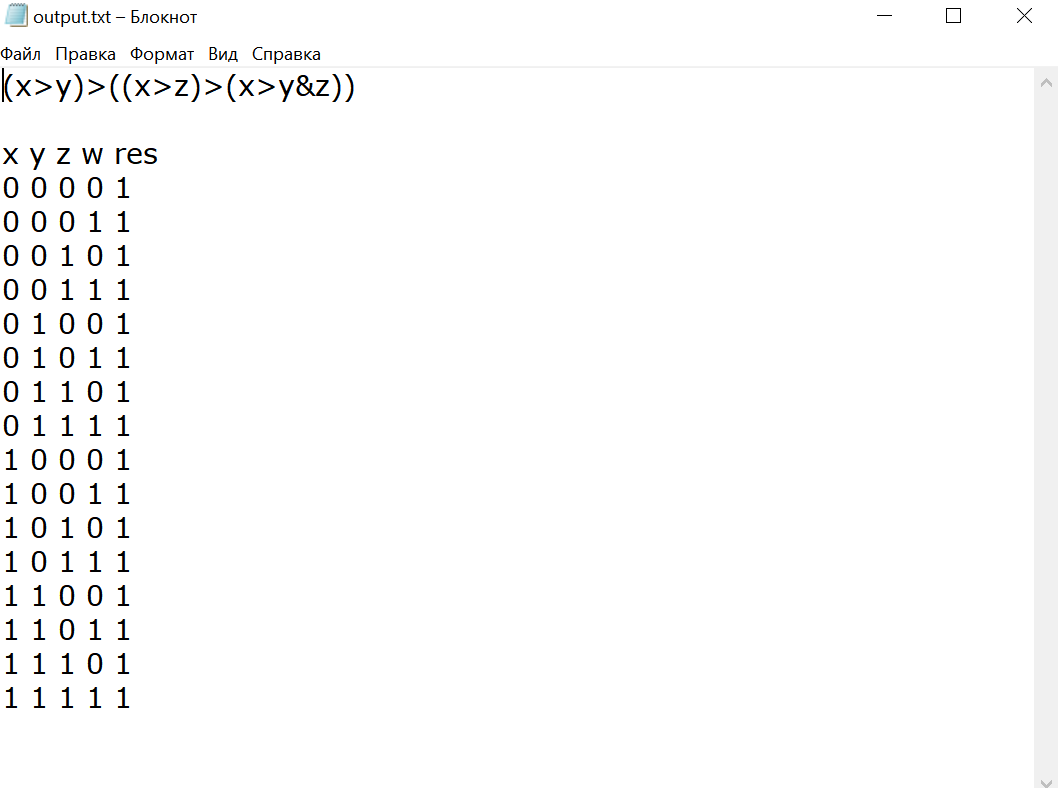
1. **Работа программы**

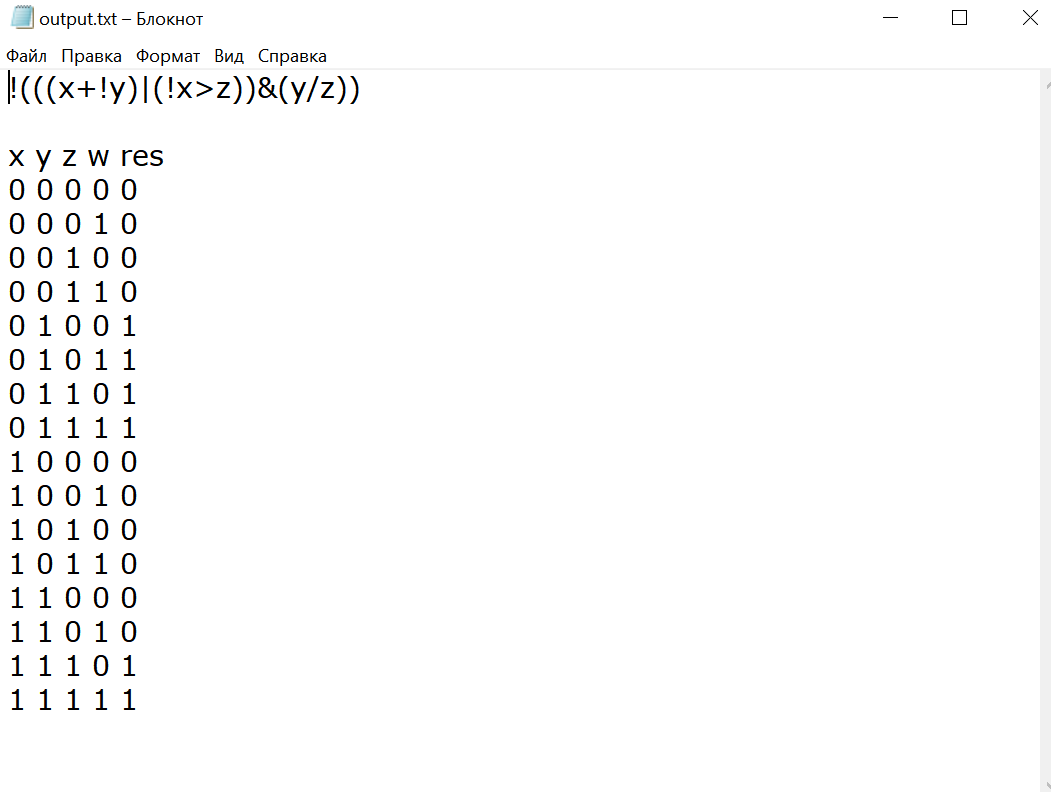
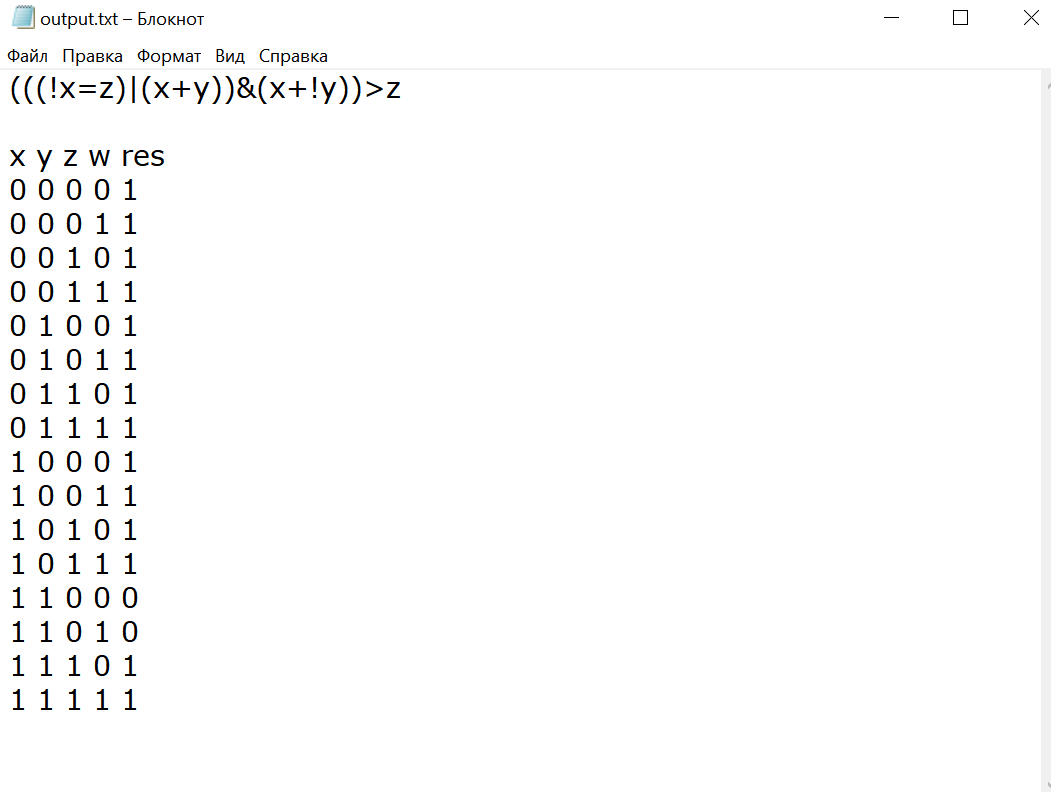
Введём все функции из пункта 7. Сравнивая, получаем, что программа работает корректно

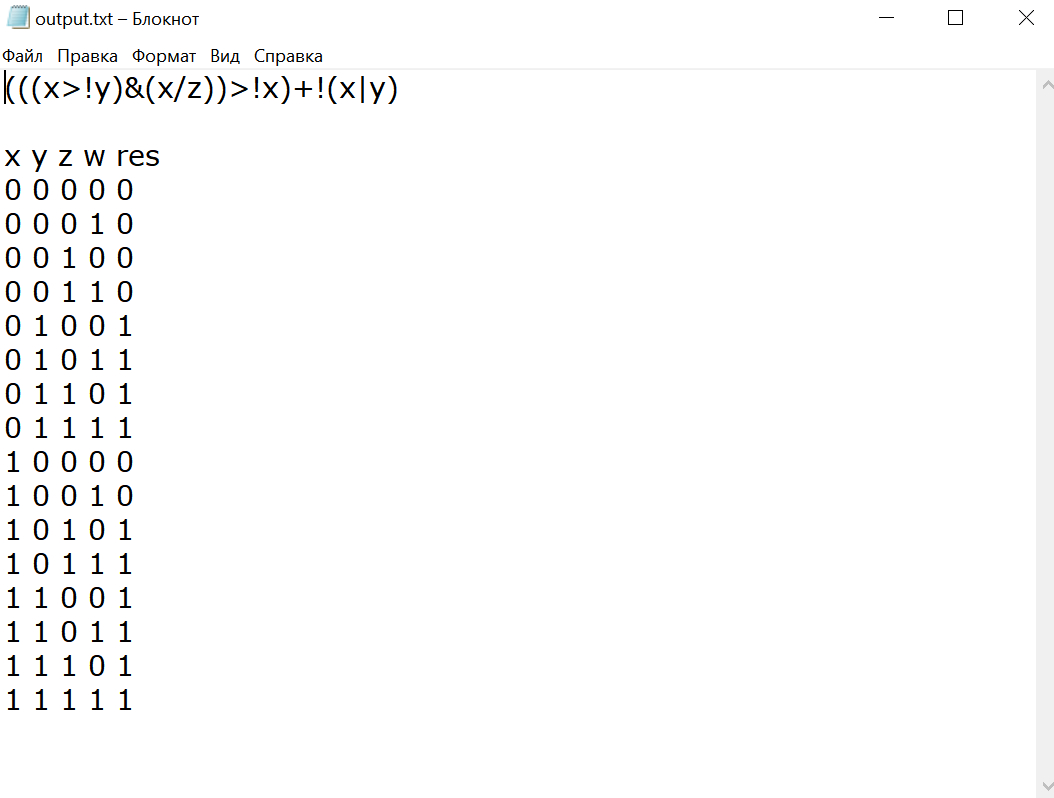
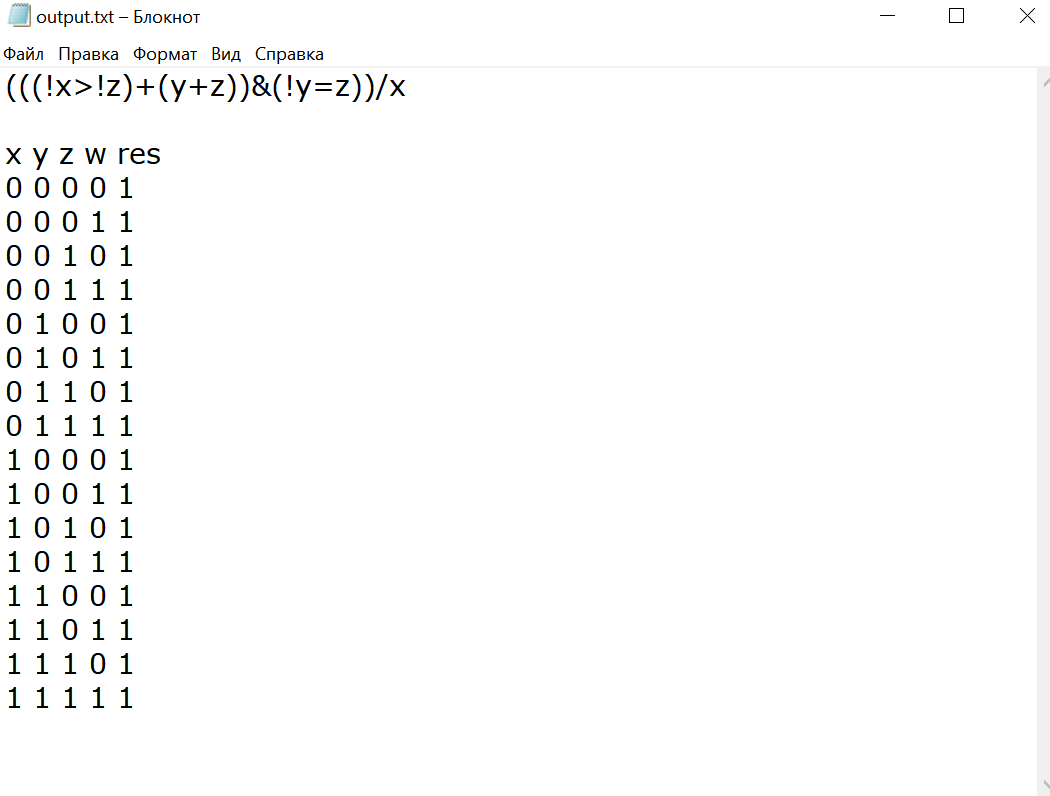
****

****

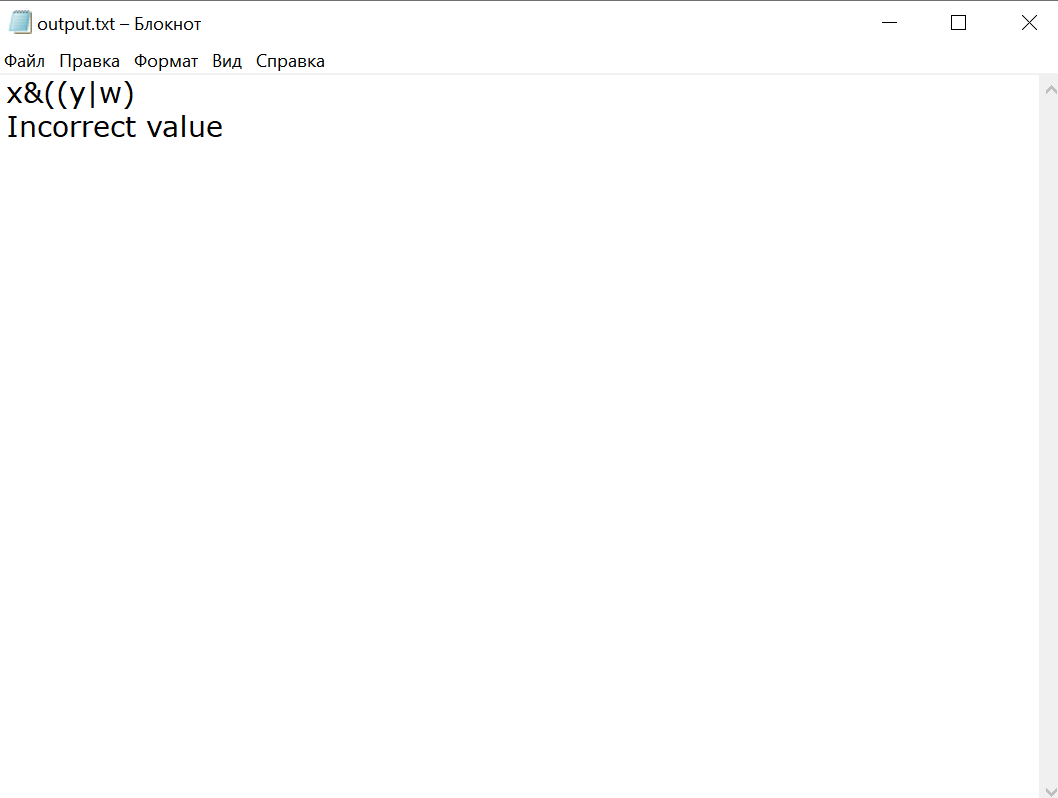
****

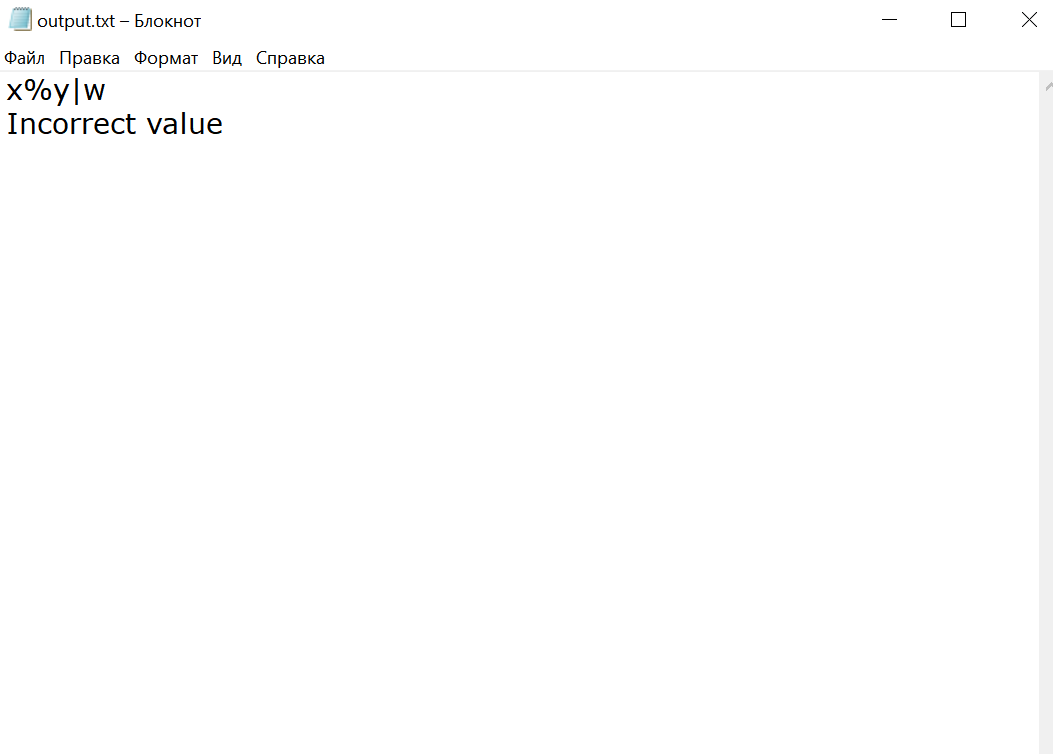
****

****

****

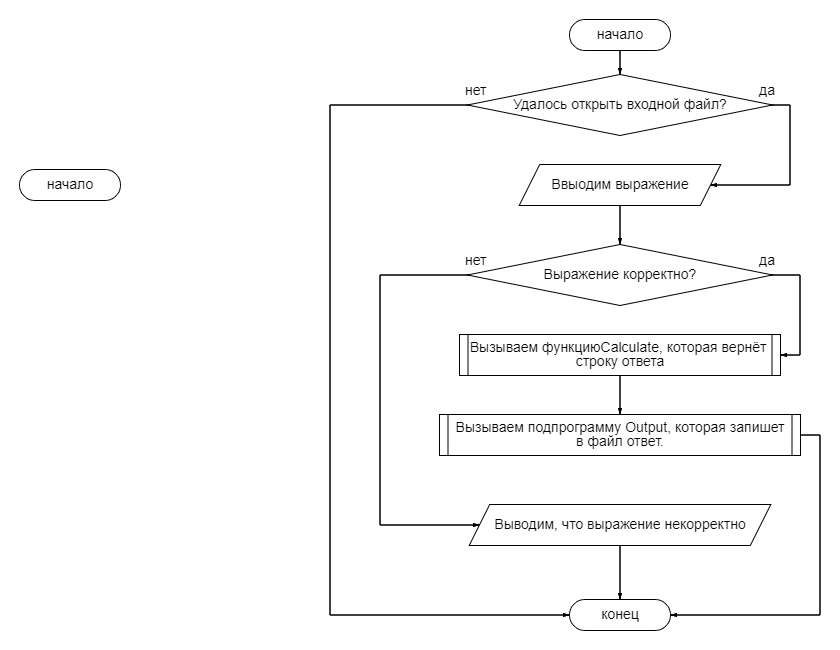
**Также приведём пример с неверными входными данными:**

****

****

1. **Блок-схема**

*Блок-схема основной программы:*

****

1. **Программа**

#include <iostream>

#include <string>

#include <fstream>

#include <iomanip>

#include <vector>

#include <exception>

#include <map>

#include <stack>

#include <algorithm>

using namespace std;

string x = "0000000011111111";

string y = "0000111100001111";

string z = "0011001100110011";

string w = "0101010101010101";

vector<bool> res = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

map<char, int> priority = { {'(', 0}, {'^', 1}, {'/', 1}, {'^', 2}, {'>', 3}, {'+', 4}, {'|', 5}, {'&', 6}, {'!', 7} };

string Evaluate(string& s1, string& s2, char oper) {

string result = s2;

if (oper == '=') { //экивалентность

for (int i = 0; i < s1.length(); i++) {

if (s1[i] != result[i]) {

result[i] = '0';

}

else {

result[i] = '1';

}

}

}

else if (oper == '>') { //импликация

for (int i = 0; i < s1.length(); i++) {

if ((s1[i] == '0') && (result[i] == '1')) {

result[i] = '0';

}

else {

result[i] = '1';

}

}

}

else if (oper == '^') { //стрелка пирса

for (int i = 0; i < s1.length(); i++) {

if ((s1[i] == '0') && (result[i] == '0')) {

result[i] = '1';

}

else {

result[i] = '0';

}

}

}

else if (oper == '/') { //штрих шеффера

for (int i = 0; i < s1.length(); i++) {

if ((s1[i] == '1') && (result[i] == '1')) {

result[i] = '0';

}

else {

result[i] = '1';

}

}

}

else if (oper == '+') { //сложение по модулю 2

for (int i = 0; i < s1.length(); i++) {

if (s1[i] != result[i]) {

result[i] = '1';

}

else {

result[i] = '0';

}

}

}

else if (oper == '|') { //лог. или

for (int i = 0; i < s1.length(); i++) {

if ((s1[i] == '0') && (result[i] == '0')) {

result[i] = '0';

}

else {

result[i] = '1';

}

}

}

else { //лог. и

for (int i = 0; i < s1.length(); i++) {

if ((s1[i] == '1') && (result[i] == '1')) {

result[i] = '1';

}

else {

result[i] = '0';

}

}

}

return result;

}

string Negation(string& s) {

for (int i = 0; i < s.size(); i++) {

if (s[i] == '0') {

s[i] = '1';

}

else {

s[i] = '0';

}

}

return s;

}

bool IsArgum(char c) {

if (c == 'x' || c == 'y' || c == 'z' || c == 'w') {

return true;

}

return false;

}

bool IsHighPriority(const char& c1, const char& c2) {

if (priority[c2] > priority[c1]) {

return true;

}

return false;

}

string Calculate(string& s) {

vector<string> arguments; //аргументы

vector<char> operations; //операции

for (int i = 0; i < s.length(); i++) {

if (IsArgum(s[i])) { //

if (s[i] == 'x') {

arguments.push\_back(x);

}

else if (s[i] == 'y') {

arguments.push\_back(y);

}

else if (s[i] == 'z') {

arguments.push\_back(z);

}

else {

arguments.push\_back(w);

}

if (!operations.empty()) {

if (operations[operations.size() - 1] == '!') {

arguments[arguments.size() - 1] = Negation(arguments[arguments.size() - 1]);

operations.pop\_back();

}

}

}

else {

if (operations.empty()) {

operations.push\_back(s[i]);

}

else {

if (s[i] == ')') {

char pred = operations[operations.size()-1]; //вытащили последнюю операцию

bool flag = true;

while (flag) {

string arg1 = arguments[arguments.size() - 1]; arguments.pop\_back();

string arg2 = arguments[arguments.size() - 1]; arguments.pop\_back();

string add = Evaluate(arg1, arg2, pred);

arguments.push\_back(add);

operations.pop\_back();

pred = operations[operations.size() - 1];

if (pred == '(') {

operations.pop\_back();

flag = false;

}

}

if (!operations.empty()) {

if (operations[operations.size() - 1] == '!') {

string arg = arguments[arguments.size() - 1]; arguments.pop\_back();

string add = Negation(arg);

arguments.push\_back(add);

operations.pop\_back();

}

}

}

else if (s[i] == '(') {

operations.push\_back(s[i]);

}

else {

char pred = operations[operations.size() - 1];

if (IsHighPriority(pred, s[i])) { // больше ли текущий приоритет в сравнении с предыдщуем

operations.push\_back(s[i]);

}

else {

bool flag = true;

char pred = operations[operations.size() - 1];

while (flag && !operations.empty()) {

if (!IsHighPriority(pred, s[i])) {

string arg1 = arguments[arguments.size() - 1]; arguments.pop\_back();

string arg2 = arguments[arguments.size() - 1]; arguments.pop\_back();

string add = Evaluate(arg1, arg2, pred);

arguments.push\_back(add);

operations.pop\_back();

if (!operations.empty()) {

pred = operations[operations.size() - 1];

}

}

else {

flag = false;

}

}

operations.push\_back(s[i]);

}

}

}

}

}

if (arguments.size() != 1) {

while (operations.size() != 0) {

char oper = operations[operations.size() - 1];

string arg1 = arguments[arguments.size() - 1]; arguments.pop\_back();

string arg2 = arguments[arguments.size() - 1]; arguments.pop\_back();

string add = Evaluate(arg1, arg2, oper);

arguments.push\_back(add);

operations.pop\_back();

}

return arguments[0];

}

return arguments[0];

}

bool IsCorrect(string& s) {

int c1 = count(s.begin(), s.end(), '('); //кол-во скобок разное

int c2 = count(s.begin(), s.end(), ')');

if (c1 != c2) {

cout << "1";

return false;

}

for (int i = 0; i < s.size(); i++) {

if (s[i] != 'x' && s[i] != 'y' && s[i] != 'z' && s[i] != 'w' &&

s[i] != '=' && s[i] != '^' && s[i] != '>' && s[i] != '!' &&

s[i] != '|' && s[i] != '&' && s[i] != '/' && s[i] != '+' && s[i] != '(' && s[i] != ')') {

cout << "2";

cout << s[i]<< endl;

return false;

}

}

return true;

}

void Output(ofstream& out, const string& s) {

string x = "0000000011111111";

string y = "0000111100001111";

string z = "0011001100110011";

string w = "0101010101010101";

out << "x y z w res" << endl;

for (int i = 0; i < x.length(); i++) {

out << x[i] << ' ' << y[i] << ' ' << z[i] << ' ' << w[i] << ' ' << s[i] << endl;

}

}

int main()

{

ifstream input("input.txt");

string line;

if (!input.is\_open()) {

cout << "file is not open" << endl;

}

getline(input, line);

const string path = "output.txt";

ofstream output(path);

if (IsCorrect(line)) {

output << line << endl << endl;

string result = Calculate(line);

Output(output, result);

}

else {

output << "Incorrect value";

}

return 0;

}