

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных
систем

Лабораторная работа № 3
по дисциплине: Мат.лог. и теор. алгор
тема: «Формальные теории»

Выполнил: ст. группы ПВ-202
Аладиб Язан

Проверил:

Куценко Дмитрий Александрович
Бондаренко Татьяна Владимировна

Белгород 2022 г.

Задание варианта №2:

Номер варианта	Номера задач			
2	2.3	12.4	15.2	20.2

1)

2. Доказать, что:

$$\vdash (\bar{B} \rightarrow \bar{A}) \rightarrow (A \rightarrow B)$$

$\bar{B} \rightarrow \bar{A} \vdash A \rightarrow B$ по теореме, аффинной теореме дедукции

$\bar{B} \rightarrow \bar{A}, A \vdash B$

- $\bar{B} \rightarrow \bar{A}$ — гипотеза
- A — гипотеза
- $A \rightarrow (\bar{B} \rightarrow \bar{A})$ A1, B, \bar{B}
- $\bar{B} \rightarrow \bar{A}$ modus Ponens 2, 3
- $(\bar{B} \rightarrow \bar{A}) \rightarrow ((\bar{B} \rightarrow \bar{A}) \rightarrow B)$ A3
- $(\bar{B} \rightarrow \bar{A}) \rightarrow B$ modus Ponens 1, 5
- B modus Ponens 4, 6

2)

12. Доказать, что следующие формулы выводимы в исчислении высказываний.

4) $A \vee B \equiv B \vee A$.

Решение:

- $A \rightarrow (B \vee A)$ аксиома 7
- $B \rightarrow (B \vee A)$ аксиома 6
- $(A \rightarrow (B \vee A)) \rightarrow ((B \rightarrow (B \vee A)) \rightarrow (A \vee B) \rightarrow (B \vee A))$ аксиома 8
- $(B \rightarrow (B \vee A)) \rightarrow ((A \vee B) \rightarrow (B \vee A))$ modus Ponens 1, 3
- $(A \vee B) \rightarrow (B \vee A)$ modus Ponens 2, 4

3)

15. Построить таблицы истинности высказываний:

2) $\vdash (B \rightarrow C) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$.

A	B	C	$B \rightarrow C$	$A \rightarrow B$	$A \rightarrow C$	$A \rightarrow B \rightarrow (A \rightarrow C)$	F
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	0	1	1	1	1
0	1	1	1	1	1	1	1
1	0	0	1	0	0	1	1
1	0	1	1	0	1	1	1
1	1	0	0	1	0	0	1
1	1	1	1	1	1	1	1

Выражение тождественно истинно во всех случаях истинности:

$\vdash (A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C)). (*)$

Из того что (*)-аксиома следует, что отсюда выводится в этой теории

4)

20. Показать в исчислении предикатов.

2) $\exists x(A(x) \vee B(x)) \vdash \exists x A(x) \vee \exists x B(x)$.

Решение:

$\exists x(A(x) \vee B(x)) \vdash \exists x A(x) \vee \exists x B(x)$

1. $(A(x) \vee B(x)) \rightarrow (A(+)) \vee B(+)$
по 1 аксиоме Гейнхайса

2. $(A(+) \vee B(+)) \rightarrow (\exists x A(x) \vee \exists x B(x))$
по 2 аксиоме Гейнхайса

3. $(\exists x(A(x) \vee B(x))) \rightarrow (\exists x A(x) \vee \exists x B(x))$
по Modus Ponens из 1 и 2

```

#include <iostream>
#include <string>
#include <set>
#include <map>
#include <stack>
#include <queue>
#include <cassert>
#include <cstdlib>
#include <locale>
#include <windows.h>

// Объявление типов.
// Токен (лексема):
typedef char Token;
// Стек токенов:
typedef std::stack<Token> Stack;
// Последовательность токенов:
typedef std::queue<Token> Queue;
// Множество различных токенов:
typedef std::set<Token> Set;
// Таблица значений переменных:
typedef std::map<Token, Token> Map;
// Пара переменная-значение:
typedef std::pair<Token, Token> VarVal;
// Строка символов:
typedef std::string String;

// Является ли токен числом?
inline bool isNumber(Token t) {
    return t == '0' || t == '1';
}

// Является ли токен переменной?
inline bool isVariable(Token t) {
    return (t >= 'A' && t <= 'Z') || (t >= 'a' && t <= 'z');
}

// Является ли токен операцией?
inline bool isOperation(Token t) {
    return (t == '|' || t == '&' || t == '-' || t == '>' || t == '~');
}

// Является ли токен открывающей скобкой?
inline bool isOpeningPar(Token t) {
    return t == '(';
}

// Является ли токен закрывающей скобкой?
inline bool isClosingPar(Token t) {
    return t == ')';
}

// Вернуть величину приоритета операции
// (чем больше число, тем выше приоритет)
inline int priority(Token op) {

```

```

assert(isOperation(op));
int res = 0;
switch (op) {
    case '-':
        // Отрицание — наивысший приоритет
        res = 5;
        break;
    case '&':
        // Конъюнкция
        res = 4;
        break;
    case '|':
        // Дизъюнкция
        res = 3;
        break;
    case '>':
        // Импликация
        res = 2;
        break;
    case '~':
        // Эквивалентность — наинизший приоритет
        res = 1;
        break;
}
return res;
}

// Преобразовать последовательность токенов,
// представляющих выражение в инфиксной записи,
// в последовательность токенов, представляющих
// выражение в обратной польской записи
// (алгоритм Дейкстры «Сортировочная станция»)
Queue infixToPostfix(Queue input) {
    // Выходная последовательность
    Queue output;
    // Рабочий стек:
    Stack s;
    // Текущий входной токен:
    Token t;
    // Пока есть токены во входной последовательности:
    while (!input.empty()) {
        // Получить токен из начала входной последовательности
        t = input.front();
        input.pop();
        // Если токен — число или переменная, то:
        if (isNumber(t) || isVariable(t)) {
            // Добавить его в очередь вывода
            output.push(t);
        } else if (isOperation(t)) {
            // Пока на вершине стека присутствует токен-операция op2
            // и у op1 приоритет меньше либо равен приоритету op2, то:
            while (!s.empty() && isOperation(s.top()) && priority(t) <=
                priority(s.top())) {
                // переложить op2 из стека в выходную очередь
                output.push(s.top());
                s.pop();
            }
        }
    }
}

```

```

    }
    // Положить op1 в стек
    s.push(t);
    // Если token — открывающая скобка, то:
    } else if (isOpeningPar(t)) {
    // Положить его в стек
    s.push(t);
    // Если token — закрывающая скобка, то:
    } else if (isClosingPar(t)) {
    // Пока token на вершине стека не является открывающей скобкой:
    while (!s.empty() && !isOpeningPar(s.top())) {
    // Перекидывать токены-операции из стека
    // в выходную очередь
        assert(isOperation(s.top()));
        output.push(s.top());
        s.pop();
    }
    // Если стек закончился до того,
    // как был встречен token-«открывающая скобка», то:
    if (s.empty()) {
    // В выражении пропущена открывающая скобка
        throw String("Пропущена открывающая скобка!");
    } else {
        // Иначе выкинуть открывающую скобку из стека
        // (но не добавлять в очередь вывода)
        s.pop();
    }
    } else {
        // В остальных случаях входная последовательность
        // содержит token неизвестного типа
        String msg("Неизвестный символ '\"");
        msg += t + String("\'!");
        throw msg;
    }
    while (!s.empty()) {
    // Если token на вершине стека — открывающая скобка, то:
    if (isOpeningPar(s.top())) {
    // В выражении присутствует незакрытая скобка
        throw String("Незакрытая скобка!");
    } else {
        // Иначе переложить token-операцию из стека в выходную
        очередь
        assert(isOperation(s.top()));
        output.push(s.top());
        s.pop();
    }
    }
    // Конец алгоритма.
    // Выдать полученную последовательность
    return output;
}

// Напечатать последовательность токенов
void printSequence(Queue q) {
    while (!q.empty()) {
        std::cout << q.front();
        q.pop();
    }
}

```

```

    }
    std::cout << std::endl;
}
// Является ли символ пробельным?
inline bool isSpace(char c){
    return c <= ' ';
}
// Если символ – маленькая буква, преобразовать её в большую,
// иначе просто вернуть этот же символ
inline char toUpperCase(char c) {
    if (c >= 'a' && c <= 'z') {
        return c - 'a' + 'A';
    } else {
        return c;
    }
}
// Преобразовать строку с выражением в последовательность токенов
// (лексический анализатор)
Queue stringToSequence(const String& s) { Queue res;
    for (size_t i = 0; i < s.size(); ++i) {
        if (!isSpace(s[i])) {
            res.push(toUpperCase(s[i]));
        }
    }
    return res;
}
// Напечатать сообщение об ошибке
inline void printErrorMessage(const String& err) {
    std::cerr << "*** ОШИБКА! " << err << std::endl;
}
// Ввести выражение с клавиатуры
inline String inputExpr() {
    String expr;
    std::cout << "Формула логики высказываний: ";
    std::getline(std::cin, expr);
    return expr;
}
// Выделить из последовательности токенов переменные
unsigned getVariables(Queue s, Set& res) {
    //Set res;
    unsigned c = 0;
    while (!s.empty()) {
        if (isVariable(s.front()) && res.count(s.front()) == 0) {
            res.insert(s.front());
            c++;
        }
        s.pop();
    }
    return c;
}
// Получить значения переменных с клавиатуры
Map inputVarValues(const Set& var) {
    Token val; Map res;
    for (Set::const_iterator i = var.begin();
        i != var.end(); ++i) {
        do {
            std::cout << *i << " = ";

```

```

        std::cin >> val;
        if (!isNumber(val)) {
            std::cerr << "Введите 0 или 1!" << std::endl;
        }
    } while (!isNumber(val));
    res.insert(VarVal(*i, val));
}
return res;
}
// Заменить переменные их значениями
Queue substValues(Queue expr, Map& varVal) {
    Queue res;
    while (!expr.empty()) {
        if (isVariable(expr.front())) {
            res.push(varVal[expr.front()]);
        }
        else {
            res.push(expr.front());
        }
    }
    expr.pop();
    return res;
}
// Является ли операция бинарной?
inline bool isBinOp(Token t) {
    return t == '&' || t == '|' || t == '>' || t == '~';
}
// Является ли операция унарной?
inline bool isUnarOp(Token t) {
    return t == '-';
}
// Получить bool-значение токена-числа (true или false)
inline bool logicVal(Token x) {
    assert(isNumber(x)); return x == '1';
}
// Преобразовать bool-значение в токен-число
inline Token boolToToken(bool x) {
    if (x) {
        return '1';
    }
    else {
        return 0;
    }
}
// Вычислить результат бинарной операции
inline Token evalBinOp(Token a, Token op, Token b) {
    assert (isNumber(a) && isBinOp(op) && isNumber(b));
    bool res;
    // Получить bool-значения операндов
    bool left = logicVal(a);
    bool right = logicVal(b);
    switch (op){
        case '&':
            // Конъюнкция
            res = left && right;
            break;
        case '|':

```



```

        // Дизъюнкция
        res = left || right;
        break;
    case '>':
        // Импликация
        res = !left || right;
        break;
    case '~':
        // Эквивалентность
        res = (!left || right) && (!right || left);
        break;
    }
    return boolToToken(res);
}
// Вычислить результат унарной операции
inline Token evalUnarOp(Token op, Token a) {
    assert (isUnarOp(op) && isNumber(a));
    bool res = logicVal(a);
    switch (op){
        case '-':
            // Отрицание
            res = !res;
            break;
    }
    return boolToToken(res);
}
void evalOpUsingStack(Token op, Stack &s) {
    assert (isOperation(op));
    // Если операция бинарная, то:
    if (isBinOp(op)) {
        // В стеке должны быть два операнда
        if (s.size() >= 2) {
            // Если это так, то извлекаем правый операнд-число
            Token b = s.top();
            if (!isNumber(b)) {
                throw String("Неверное выражение!");
            }
            s.pop();
            // Затем извлекаем левый операнд-число
            Token a = s.top();
            if (!isNumber(a)) {
                throw String("Неверное выражение!");
            }
            s.pop();
            // Помещаем в стек результат операции
            s.push(evalBinOp(a, op, b));
        } else {
            throw String("Неверное выражение!");
        }
        // Иначе операция унарная
    } else if (isUnarOp(op) && !s.empty()) {
        // Извлекаем операнд
        Token a = s.top();
        if (!isNumber(a)) {
            throw String("Неверное выражение!");
        }
        s.pop();
    }
}

```

```

        // Помещаем в стек результат операции
        s.push(evalUnarOp(op, a));
    } else {
        throw String("Неверное выражение!");
    }
}

Token evaluate(Queue expr) {
    // Рабочий стек
    Stack s;
    // Текущий токен
    Token t;
    // Пока входная последовательность содержит токены:
    while (!expr.empty()) {
        // Считать очередной токен
        t = expr.front();
        assert (isNumber(t) || isOperation(t));
        expr.pop();
        // Если это число, то:
        if (isNumber(t)) {
            // Поместить его в стек
            s.push(t);
        } else if (isOperation(t)) {
            // Вычислить её, модифицируя стек
            // (результат также помещается в стек)
            evalOpUsingStack(t, s);
        }
    }
    // Результат – единственный элемент в стеке
    if (s.size() == 1) {
        // Вернуть результат
        return s.top();
    } else {
        throw String("Неверное выражение!");
    }
}

void printResult(Token r) {
    assert (isNumber(r));
    std::cout << "Значение выражения: " << r << std::endl;
}

//построение матрицы СКНФ формул
int SKNF(Queue expr, int** matr, Set vars, unsigned countVars) {
    unsigned mask;
    unsigned lim = 1 << countVars;
    unsigned c = 0;

    for (size_t i = 0; i < lim; i++) {
        Map varVals;
        mask = lim;
        Set::const_iterator k = vars.begin();
        for (size_t j = 0; j < countVars; j++) {
            mask >>= 1;
            bool t = i & mask;
            Token T = boolToToken(t);
            varVals.insert(VarVal(*k, T));
            // std::cout << t << ' ';
            k++;
        }
    }
}

```

```

    }
    Queue sExpr = substValues(expr, varVals);
    Token res = evaluate(sExpr);
    if (res == '0') {
        mask = lim;
        for (size_t j = 0; j < countVars; j++) {
            mask >>= 1;
            bool t = i & mask;
            if (t == true) {
                (*matr)[j] = -1;
            } else
                (*matr)[j] = 0;
        }
        matr++;
        c++;
    }
}
return c;
}
//проверка является ли резольвента пустой
bool EmptySequence(int* a, unsigned countVars) {
    for (unsigned i = 0; i < countVars; i++)
        if (a[i] != 1)
            return false;
    return true;
}
//проверка на равенство массивов
bool EqvivArr(int* a, int* b, unsigned countVars) {
    for (int i = 0; i < countVars; i++)
        if (a[i] != b[i])
            return false;
    return true;
}
//поиск массива в матрице
bool SearchArr(int** matr, int n, unsigned countVars, int* a) {
    for (int i = n - 1; i >= 0; i--) {
        if (EqvivArr(matr[i], a, countVars))
            return true;
    }
    return false;
}
//запись в матрицу дизъюнкта
void WriteArr(int** matr, int* n, unsigned countVars, int* a) {
    for (int i = 0; i < countVars; i++) {
        matr[*n][i] = a[i];
    }
    (*n)++;
}
//ВЫВОД ДИЗЪЮНКТА
void PrintfArr(int** matr, int n, unsigned countVars, Set vars) {
    std::set<int>::iterator it;
    std::cout << "(";
    for (int i = 0, it = *vars.begin(); i < countVars; i++, it++)
        if (matr[n][i] != 1) {
            if (matr[n][i] == -1)
                std::cout << '-' << (Token)(it);
            else

```

```

        std::cout << (Token)(it);
        std::cout << " | ";
    }
    std::cout << "\b\b\b)";
}
//метод резолюций
bool MethodResolution(int** matr, int n, unsigned countVars, Set vars) {
    int k = n, tk = n;
    int* a = new int[countVars];
    const time_t TIMEOUT_LIMIT = 100;
    // Лимит времени, которое отводится на поиск решения
    time_t start = time(NULL);
    // Время начала поиска решения
    bool is_solved = false;
    while (time(NULL) - start < TIMEOUT_LIMIT and !is_solved) {
        for (int i = 0; i < k - 1 and !is_solved; i++)
            for (int j = i + 1; j < k and !is_solved; j++) {
                for (int z = 0; z < countVars; z++) {
                    if (matr[i][z] == 0 && matr[j][z] == -1 || matr[i][z] ==
-1 && matr[j][z] == 0)
                        a[z] = 1;
                    else if (matr[i][z] == 0 || matr[j][z] == 0)
                        a[z] = 0;
                    else if (matr[i][z] == -1 && matr[j][z] == -1)
                        a[z] = -1;
                    else if (matr[i][z] == 1 && matr[j][z] == 1)
                        a[z] = 1;
                }
                is_solved = EmptySequence(a, countVars);
                if (is_solved) {
                    std::cout << "Выполним склеивание ";
                    PrintfArr(matr, i, countVars, vars);
                    std::cout << " и ";
                    PrintfArr(matr, j, countVars, vars);
                    std::cout << ": пустая резольвента\n";
                    return true;
                }
                else {
                    if (!SearchArr(matr, tk, countVars, a)) {
                        WriteArr(matr, &tk, countVars, a);
                        std::cout << "Выполним склеивание ";
                        PrintfArr(matr, i, countVars, vars);
                        std::cout << " и ";
                        PrintfArr(matr, j, countVars, vars);
                        std::cout << ": ";
                        PrintfArr(matr, tk, countVars, vars);
                        std::cout << " \n";
                    }
                }
            }
        k = tk;
    }
    return is_solved;
}
//Вывод множества дизъюнктов
void PrintfSetDis(int** matr, int n, int countVars, Set vars) {

```

```

std::cout << "{";
for (int i = 0; i < n; i++) {
    PrintfArr(matr, i, countVars, vars);
    std::cout << ", ";
}
std::cout << "\b\b}";
}
// Главная программа
int main() {
    SetConsoleOutputCP(CP_UTF8); //Подключение русского языка
    try {
        std::cout << "Введите число посылок: ";
        int n;
        String expres;
        std::cin >> n;
        getchar();
        expres += "-(";
        //ВВОД формул
        String exp;
        for (int i = 0; i < n - 1; i++) {
            exp = inputExpr();
            expres += '(' + exp + "&";
        }
        exp = inputExpr();
        expres += '(' + exp + ')';
        expres += ">";
        std::cout << "Введите проверяемое следствие\n";
        exp = inputExpr();
        expres = expres + '(' + exp + ")";
        expres += ")";
    }
    // Преобразовать выражение в последовательность токенов
    Queue input = stringToSequence(expres);
    // Преобразовать последовательность токенов в ОПЗ
    Queue output = infixToPostfix(input);
    Set vars;
    unsigned countVars = getVariables(output, vars);
    unsigned t = 1 << countVars;
    //создание матрицы
    int **matr = new int *[3 * t];
    for (int i = 0; i < 3 * t; i++) {
        matr[i] = new int[countVars];
    }
    //построение матрицы СКНФ
    unsigned k = SKNF(output, matr, vars, countVars);
    std::cout << "Множество дизъюнктов:\n";
    PrintfSetDis(matr, k, countVars, vars);
    std::cout << "\n";
    bool isSloved = MethodResolution(matr, k, countVars, vars);
    if (isSloved)
        std::cout << "Получена пустая резольвента, теорема доказана";
    else
        std::cout << "Пустую резольвенту невозможно получить, теорема опровергнута";
    //удаление матрицы
    for (int i = 0; i < 3 * t; i++) {
        delete[] matr[i];
    }
}

```

```
}  
delete[] matr;  
catch (const String& err) {  
    // Если возникла ошибка, вывести сообщение  
    printErrorMessage(err);  
    // И выйти из программы с неудачным кодом завершения  
    exit(1);  
}  
// Конец программы  
return 0;  
}
```