МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет математики и компьютерных наук**

**Кафедра вычислительной математики и информатики**

Допустить к защите

Заведующий кафедрой

д-р. тех. наук, доцент

\_\_\_\_\_\_\_\_\_Н. А. Наумова

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г.

Руководитель ООП

д-р. тех. наук, профессор

\_\_\_\_\_\_\_\_\_Ю. М. Вишняков

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**(МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)**

**ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ   
(«ВНЕУРОЧНАЯ ДЕЯТЕЛЬНОСТЬ КЛАССНОГО РУКОВОДИТЕЛЯ»)**

Работу выполнила                           П. А. Королева

(подпись)

Направление подготовки 02.04.01 Математика и компьютерные науки .

Направленность (профиль) Вычислительная математика

Научный руководитель

канд. физ.- мат. наук, доцент                 О. В. Иванисова

(подпись)

Нормоконтролер

преподаватель                  А. А. Лахтина

(подпись)

Краснодар 2025

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc168439186)

[1 Анализ поставленной задачи и разработка алгоритма решения 5](#_Toc168439187)

[2 Программная формализация задачи 13](#_Toc168439188)

[3 Анализ скорости выполнения программы 26](#_Toc168439189)

[4 Составление контрольных примеров для отладки программы 28](#_Toc168439190)

[Заключение 45](#_Toc168439191)

[Список использованных источников 46](#_Toc168439192)

# ВВЕДЕНИЕ

Логические калькуляторы — важнейшие инструменты для обработки логических выражений и решения сложных задач, связанных с логикой. Они широко применяются в различных областях, включая математику, философию, информатику, электронику и другие.

В сети Интернет имеется большое количество пользовательских программ-«попыток» анализа логических формул, но ни одна не содержит всех необходимых компонентов анализа, либо вовсе предоставляет некорректный результат. Поэтому целью дипломной работы является разработка приложения, которое предоставит пользователю возможность эффективно обрабатывать логические формулы, строить таблицы истинности, определять тип формулы, приводить логические выражения к разным совершенным нормальным формам, проверять равносильность формул и правильность логического следования с минимальными затратами времени и памяти компьютера.

Для реализации задачи был выбран язык программирования С++, а для создания интерфейса приложения — фреймворк Qt Creator. В ходе работы будут выполнены следующие задачи:

1. изучение необходимой терминологии математической логики (алгебры высказываний);
2. анализ существующих программ логических калькуляторов;
3. решение основных задач для анализа алгоритма построения всех необходимых для разработки программы элементов;
4. выбор оптимального алгоритма решения поставленной задачи;
5. реализация выбранного алгоритма.

Из анализа пользовательских потребностей следует, что программа должна удовлетворять следующим требованиям:

1. иметь удобный интерфейс;
2. предоставлять различные варианты ввода исходных данных;
3. предоставлять возможность корректирования введенной формулы;
4. проверять правильность введенной формулы с выводом соответствующего сообщения;
5. предоставлять корректный ответ в зависимости от запроса пользователя;
6. иметь удобную и понятную инструкцию эксплуатации.

# 1 Анализ поставленной задачи и разработка алгоритма решения

По условию, задание предполагает создание приложения и написание программного кода, который будет обрабатывать введенное пользователем логическое выражение и выводить соответствующий результат в зависимости от запроса.

Для создания логического калькулятора необходимо определить основные задачи математической логики, которые должен решать калькулятор. Также стоит отметить, что алгоритм решения поставленной задачи может быть не единственным.

На первом этапе исследований необходимо определить все используемые параметры и найти возможности их перевода в программный код. Приведем основные определения и понятия из математической логики.

Логическое высказывание — это утверждение или предложение, которое может быть оценено как истинное (правдивое) или ложное (неправдивое), но не может быть одновременно истинным и ложным. Логическое высказывание выражает некоторое утверждение или факт, которое можно подвергать проверке и логическому анализу.

Логические высказывания могут быть простыми или сложными. Сложные логические высказывания состоят из комбинации нескольких простых высказываний и логических операций.

Построение из данных высказываний (или из данного высказывания) нового высказывания называется логической операцией. К основным логическим операциям можно отнести логическое отрицание (инверсия), конъюнкцию, дизъюнкцию, импликацию, эквиваленцию. Каждая из них имеет определенный приоритет выполнения, который учитывается при составлении таблицы истинности.

Логическое отрицание выполняется первым и применяется к отдельным операндам. Если исходное высказывание является истинным, то его отрицание будет ложным, и наоборот, если исходное высказывание является ложным, то его отрицание будет истинным.

Операция конъюнкции принимает значение «истина» только тогда, когда оба операнда истинны, в противном случае принимает значение «ложь».

Операция дизъюнкции принимает значение «истина», если хотя бы один из операндов истинен.

Импликация возвращает ложь только тогда, когда первый операнд истинен, а второй операнд ложен. В остальных случаях импликация возвращает истину.

Эквиваленция возвращает истину, если оба операнда имеют одинаковые значения (истина или ложь). Операция эквиваленции означает, что два высказывания эквивалентны и имеют одинаковую истинностную таблицу.

Таким образом, приоритет логических операций имеет следующий порядок:

1. логическое отрицание;
2. конъюнкция;
3. дизъюнкция;
4. импликация;
5. эквиваленция.

Существуют также и другие логические операции, такие как обратная импликация, исключающее или, и так далее. Однако они встречаются реже и обычно используются в специфических контекстах или при решении определённых задач. Основные же операции, описанные выше, являются базовыми и наиболее часто используемыми в математической логике. С помощью основных операций выражаются все остальные логические операции. Например, репликацию можно выразить через инверсию и дизъюнкцию следующим образом:

|  |
| --- |
|  |

При необходимости можно использовать скобки, чтобы явно указать порядок выполнения операций и избежать путаницы.

Путем комбинирования операций мы можем создавать логические выражения, которые отражают сложные логические связи между различными утверждениями или условиями.

Формулой алгебры высказываний называется логическое выражение, составленное из пропозициональных переменных, знаков логических операций и скобок по следующим правилам:

1. пропозициональная переменная есть формула алгебры высказываний;
2. если *A* — формула алгебры высказываний, то выражение является формулой алгебры высказываний;
3. если *A* и *B* — формулы алгебры высказываний, то выражения   
   *(A ∧ B)*, *(A ∨ B)*, *(A → B)* и *(A ↔ B)* являются формулами алгебры высказываний;
4. никаких других формул алгебры высказываний, кроме полученных согласно п. 1–3, нет.

Каждая формула может принимать значение истина или ложь в зависимости от значений входящих в нее пропозициональных переменных.

Логические формулы бывают следующих типов:

1. тождественно истинная (тавтология) — формула, принимающая значение «истина» при любом сочетании значений переменных;
2. тождественно ложная (противоречие) — формула, принимающая значение «ложь» при любом сочетании значений переменных;
3. выполнимая — формула, которая в результате анализа формулы принимает хотя бы одно значение «истина»;
4. опровержимая — формула, которая в результате анализа формулы принимает хотя бы одно значение «ложь».

Из вышесказанного следует, что проверка выполнимости формулы должна производиться посредством анализа таблицы истинности.

Таблица истинности — это таблица, используемая в логике для определения значений истинности составных высказываний, которые могут принимать значение «истина» или «ложь».

В таблице истинности каждой переменной высказывания назначается значение «истина» или «ложь», а затем определяется значение истинности всего высказывания на основе логических операций, используемых для объединения переменных.

Для логических высказываний существуют различные формы, которые позволяют представить эти высказывания в определенной структуре.

Дизъюнктивная нормальная форма (ДНФ) — это дизъюнкция попарно различных элементарных конъюнкций.

ДНФ называется совершенной (СДНФ), если:

1. все члены каждой дизъюнкции различны;
2. нет конъюнкции, одновременно содержащей переменную и ее отрицание;
3. каждая конъюнкция содержит все переменные, входящие в исходную формулу.

Конъюнктивная нормальная форма (КНФ) — это конъюнкция попарно различных элементарных дизъюнкций.

КНФ называется совершенной (СКНФ), если:

1. все члены каждой дизъюнкции различны;
2. нет дизъюнкции, одновременно содержащей переменную и отрицание этой переменной;
3. каждая дизъюнкция содержит все переменные, входящие в исходную формулу.

Использование таких форм позволяет удобно анализировать логические выражения, выявлять их особенности и свойства. Каждая из этих форм имеет свои преимущества и применяется в различных областях логики, математики, информатики.

В процессе исследования задачи было установлено, что для нахождения СДНФ и СКНФ существуют два метода: табличный, основанный на анализе таблицы истинности, и аналитический, основанный на приведении ДНФ к соответствующему виду с помощью элементарных преобразований. Учитывая необходимость переведения алгоритма на язык программирования, был выбран табличный метод.

Две формулы алгебры высказываний называются равносильными, если при любых значениях входящих в них пропозициональных переменных логические значения, получаемые из данных формул высказываний, совпадают.

Равносильность двух формул можно установить разными способами. Например, с помощью преобразований, использующих основные равносильности алгебры высказываний, установленные ранее, единственности совершенных нормальных форм, различных теорем или же с помощью таблицы истинности.

Формула алгебры высказываний *B* называется логическим следствием формул , если *B* принимает значение «истина» тогда, когда все принимают значение «истина». При этом формулы называются посылками (условиями), а формула *B* — заключением (следствием).

Существуют различные методы проверки верности логического следования, такие как метод от противного, использование необходимых и достаточных условий, с помощью таблицы истинности.

Для программной реализации оптимальным вариантом является установление логического следования через дефиницию.

Первые трудности в реализации задачи возникли уже на этапе ввода логической формулы. Для удобства пользователю предложены различные способы ввода данных:

1. ввод посредством клавиатуры;
2. ввод с помощью кнопок приложения;
3. запись и чтение из файла.

Но сразу же возник вопрос: в каком формате хранить и обрабатывать указанное пользователем выражение?

Полученное выражение хранится в переменной строкового типа string. Таблица истинности для введенного логического выражения должна быть универсальной, вне зависимости от сложности выражения и количества операций и операндов в нем. Для этого данная строка изучается с помощью вложенных условных операторов, откуда становится известно о количестве операндов и операций, входящих в данную логическую формулу.

Благодаря этой информации и знаниям из дискретной математики, на данном этапе можно задать размер статической матрицы, в которой будет храниться дальнейшие вычисления таблицы истинности.

Количество строк определим как , где *n* — количество переменных (операндов), входящих в введенную пользователем формулу (так как каждая переменная может принимать только два значения — истина или ложь). Количество столбцов будем считать как *n + 1* (итоговый столбец для вычисленных значений, которые принимает исходная формула).

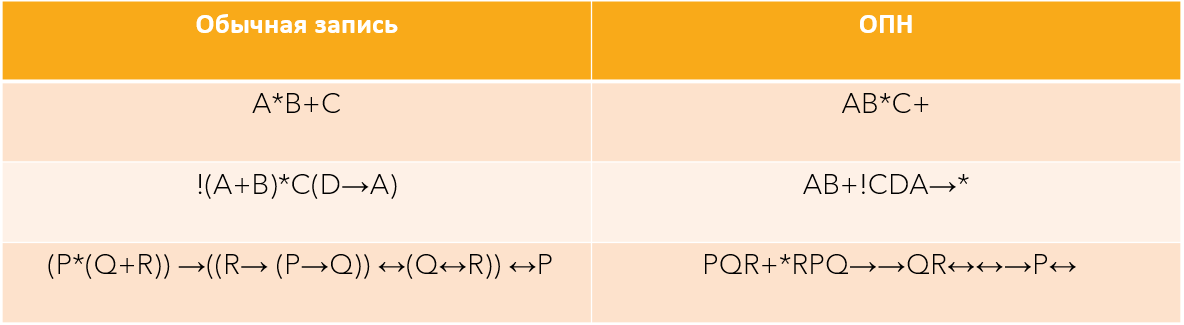
Одной из ключевых проблем при вычислении логической формулы был приоритет логических операций и необходимость правильного использования скобок. Например, когда пользователь видит логическое или математическое выражение с разными уровнями вложенных скобок, он интуитивно понимает, что нужно начинать вычисления с самых «глубоких» скобок. Однако написание такой программы, которая эффективно вычисляет подобное выражение с точки зрения затрачиваемых времени и памяти, задача не из легких. Для этого был реализован метод обратной польской нотации.

Обратная польская нотация (ОПН), также известная как постфиксная нотация, — это способ записи математических и логических выражений, в котором операнды предшествуют своим операторам.

В ОПН порядок выполнения операций однозначно определяется последовательностью элементов, без необходимости использования скобок. Это устраняет возможность неправильного интерпретирования выражений и повышает четкость их выполнения. Выражения в ОПН легче вычислять, так как операции выполняются сразу над операндами, без необходимости учитывать приоритеты операций или расставлять скобки. Это упрощает алгоритмы обработки выражений и ускоряет процесс вычислений.

Для перевода выражения в ОПН для хранения информации используется стек. Рассматриваемая логическая формула состоит из *k = l + m* символов, где *l* — количество операндов, *m* — количество логических операций.

Изначально формула записывается в инфиксной форме (двуместная операция находится между операндами *(A+B)*). ОПЗ предполагает перевод в постфиксную запись (операция находится после операндов *(AB+)*). Каждый операнд помещается в стек. Когда на вход поступает знак операции, из стека извлекается нужное количество переменных (для всех операций, кроме логического отрицания требуется две переменные) и к ним применяется данная операция, затем результат заносится в стек. Конечный результат вычислений будет находиться в вершине стека.

Рисунок 1 — Пример выражений в ОПН

Применение ОПН удобно для компьютерных программ, поскольку не требует использования дополнительного стека для хранения операций и операндов при расстановке скобок. Это позволяет снизить нагрузку на память и упрощает реализацию алгоритмов вычисления.

Написание программы было решено осуществить в свободной среде разработки (IDE) Qt Creator, так как в ней есть возможность создания удобного для пользователя интерфейса приложения логического калькулятора с различными кнопками и виджетами.

Языком программирования был выбран С++, который на данный момент является одним из популярных и востребованных языков программирования, предназначенных для создания программного обеспечения разного рода: от игр до операционных систем.

# 2 Программная формализация задачи

Код программы организован посредством связанных между собой функций. Каждая функция имеет особую роль и отвечает за определенную опцию в программе. Главной функцией является функция main для получения от пользователя логической формулы, её преобразования, а также для построения таблицы истинности. Остальные функции основываются на результате работы функции main.

В первую очередь при написании программы на языке C++ необходимо подключить соответствующие библиотеки. Это обеспечит доступ к необходимым функциям и классам, которые значительно упрощают разработку программы. Ниже приведены основные библиотеки, которые были использованы в программе.

#include <iostream> // Библиотека для работы с вводом и выводом в консоль

#include <fstream> // Библиотека для работы с файловыми потоками

#include <string.h> // Библиотека для работы со строками

#include <math.h> // Библиотека математических функций

#include <locale.h> // Библиотека для подключения русского языка

#include <stack> // Библиотека для работы со стеком

#include <iomanip> // Библиотека для форматированного ввода и вывода

#include <unordered\_set> // Библиотека для работы с хэш-таблицами

#include <set> // Библиотека для работы с отсортированными множествами

После подключения библиотек для удобства написания программы некоторые переменные были объявлены глобально. Это позволяет упростить их использование в различных функциях программы, уменьшая количество передаваемых аргументов и улучшая читаемость кода.

string str, logical\_operation = "(!\*+><~)";

string logical\_per = "ABCDEFGHIJKLMNOPQRSTUVWYXZabcdefghijklmnopqrstuvwyxz";

int num\_column; int num\_rows;

int \*\*table;

int kol\_per;

Программа содержит 3 основные функции для определения равносильности двух формул, проверки на верность логического следования и различные вычисления по таблице истинности (тип формулы, построение СКНФ, СДНФ). Основная задача каждой из этих функций — формирование таблицы истинности. Но, в зависимости от условия задачи, таблица истинности формируется по-разному.

Так, например, для различных вычислений по таблице истинности, таблица строится стандартным образом: первые *n* столбцов — все уникальные переменные, входящие в веденную логическую формулу, затем добавляется еще один столбец, содержащий итоговое значение формулы для каждого набора всевозможных значений логических переменных.

Для определения равносильности двух формул таблица истинности строится аналогичным образом: первые *n* столбцов — уникальные переменные, затем добавляется еще два столбца в таблицу для итоговых значений двух формул, равносильность которых подлежит проверке.

Таблица истинности для проверки на правильность логического следования строится несколько иначе. Неизменными остаются первые *n* столбцов, содержащие уникальные переменные. После этого добавляется еще *k + 1* столбец, в котором находятся значения *k* посылок и значение одного заключения.

В программе имеются десять дополнительных функций, каждая из которых отвечает за определенное действие (проверка на корректность введенной формулы, вычисление таблицы истинности, определения типа формулы и другие).

Также код программы содержит четыре вспомогательные функции для задания приоритета логических операций и проверки символа на принадлежность к латинскому алфавиту, которые вызываются в основных программах.

Ниже приведено описание основных алгоритмов, реализованных в коде программы.

Логическая функция bool proverka (string str) реализует алгоритм проверки введенной логической формулы на корректность. Введенная пользователем логическая формула должна удовлетворять ряду условий, которые гарантируют ее синтаксическую правильность для дальнейшей работы программы. В алгоритме реализованы следующие проверки логической формулы.

Проверка на пустые скобки.

for (int i = 0; i < str.length() - 1; i++)

if (str[i] == '(' && str[i + 1] == ')')

return false;

Данный фрагмент кода анализирует всю строку, в которой содержится введенная формула и проверяет, есть ли где-либо пара пустых скобок (). Если такие скобки найдены, функция возвращает false, так как это делает формулу некорректной.

Проверка на одинаковое количество открывающих и закрывающих скобок.

int skobka\_1=0, skobka\_2=0;

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

{if (str[i]=='(') skobka\_1++;

else if (str[i]==')') skobka\_2++;}

if (skobka\_1!=skobka\_2)

return false;

В данном фрагменте кода считываются все открывающиеся и закрывающиеся скобки в строке. Если их количество не совпадает, функция возвращает false.

Проверка на знаки логической операции.

string logical\_operation2 = "\*+><~";

for(int j=0; j<logical\_operation2.length(); j++)

if ((str[0]!='!' && str[0] == logical\_operation2[j]) || (str[str.length()]== logical\_operation2[j])) return false;

for (int i = 0; i < str.length() - 1; i++)

for(int j=0; j<logical\_operation2.length(); j++)

for(int k=0; k<logical\_operation2.length(); k++)

{if (str[i] == logical\_operation2[j] && str[i + 1] == logical\_operation2[k]) return false;}

for (int i = 0; i < str.length() - 1; i++)

if (str[i] == '!' && str[i + 1] == '!')

return false;

Указанный фрагмент проверяет, чтобы первый и последний символы не были операциями, за исключением логического отрицания, которое может быть в начале.

Проверка на связь двух соседних операндов, связанных логической операцией:

for (int i = 0; i < str.length() - 1; i++)

for(int j = 0; j < logical\_per.length(); j++)

for(int k = 0; k < logical\_per.length(); k++)

{if (str[i] == logical\_per[j] && str[i + 1] == logical\_per[k]) return false;}

В данной части кода осуществляется проверка, что бы не было двух подряд идущих переменных, что также делает формулу некорректной.

Проверка на посторонние символы:

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

if (logical\_per.find(str[i]) == std::string::npos && logical\_operation.find(str[i]) == std::string::npos)

return false;

Данный блок кода проверяет, чтоб в строке не было символов, не принадлежащих указанному допустимому набору (переменные, логические операции, скобки).

Функция возвращает true («истинна»), если строка проходит все проверки, и false («ложь») в противном случае, что свидетельствует о некорректности введенной логической формулы.

Функция string opz (const string& str) реализует алгоритм преобразования введенной пользователем логической формулы из инфиксной записи в постфиксную или, другими словами, реализует алгоритм обратной польской нотации (ОПЗ).

Преобразование логической формулы происходит по следующему алгоритму:

1. создается пустой стек для временного хранения операторов;
2. введенное пользователем выражение считывается посимвольно слева направо;
3. если очередной символ является операндом (логической переменной), то он добавляется в выходную строку;
4. если очередной символ является открывающейся скобкой, то он помещается в стек;
5. если очередной символ является оператором (логической операцией), то:
6. пока стек не пуст и верхний элемент стека не является открывающейся скобкой, извлекаются операторы из стека и добавляются в выходную строку с учетом приоритета логических операций;
7. текущий оператор помещается в стек;
8. если очередной символ является закрывающейся скобкой, то:
9. пока верхний элемент стека не является открывающейся скобкой, извлекаются операторы из стека и добавляются в выходную строку;
10. открывающая скобка извлекается из стека и удаляется;
11. шаги 3–6 повторяются для оставшихся символов в строке;
12. пока стек не пуст, из него извлекаются операторы и добавляются в выходную строку.

Ниже приведен программный код описанного алгоритма перевода логической формулы в обратную польскую нотацию:

stack<char> operators;

string result;

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

char currentChar = str[i];

if (logical\_per.find(currentChar) != string::npos) {

result += currentChar;}

else if (currentChar == '(') {

operators.push(currentChar);}

else if (currentChar == ')') {

while (!operators.empty() && operators.top() != '(')

{result += operators.top();

operators.pop();}

operators.pop();}

else if (currentChar == '!') {

operators.push(currentChar);}

else if (logical\_operation.find(currentChar) != string::npos) {

while (!operators.empty() && priority(operators.top()) >= priority(currentChar))

{result += operators.top();

operators.pop();}

operators.push(currentChar);}}

while (!operators.empty()) {

result += operators.top();

operators.pop();}

return result;

Результатом данной функции является переменная строкового типа, содержащая введенную пользователем логическую формулу, записанную в постфиксной форме, с сохранением правильного порядка операций и учетом их приоритета.

Функция void calculate (string postfix, int \*\*table, int num\_rows, map<char, int> &var\_positions, int num\_vars) реализует алгоритм заполнения таблицы истинности.

Данная функция вычисляет значения логических выражений, представленных в ОПН, для каждой строки таблицы истинности. Результаты вычислений добавляются в таблицу.

set<char> variables;

for (char c : postfix) {

if (isVariable(c) && logical\_per.find(c) != string::npos) variables.insert(c);}

В данном фрагменте кода создается множество variables для хранения уникальных переменных, присутствующих в выражении ОПН.

int var\_index = 0;

for (char var : variables) {

var\_positions[var] = var\_index++;}

Переменные из множества variables связываются с их позициями в таблице истинности. Для этого создается словарь var\_positions, где каждая переменная получает свой уникальный индекс.

for (int i = 0; i < num\_rows; i++) {

string expression = "";

for (char c : postfix) {

if (isVariable(c)) {

int var\_position = var\_positions[c];

bool var\_value = table[i][var\_position];

expression += to\_string(var\_value);}

else {expression += c;}}

Для каждой строки таблицы истинности (т.е. каждого набора значений переменных) создается строка expression, где логические переменные заменяются на их значения («0» или «1»).

stack<bool> operands;

for (char c : postfix) {

if (isVariable(c)) {

int var\_position = var\_positions[c];

bool var\_value = table[i][var\_position];

operands.push(var\_value);}

else if (isOperator(c)) {

if (c == '!') {

bool op = operands.top();

operands.pop();

operands.push(!op);}

else {

bool op2 = operands.top();

operands.pop();

bool op1 = operands.top();

operands.pop();

bool result = evaluate(c, op1, op2);

operands.push(result);}}}

bool result = operands.top();

operands.pop();

table[i][num\_vars] = result ? 1 : 0;}}

Для каждой строки таблицы истинности реализуется следующий алгоритм:

1. создается стек operands для хранения промежуточных значений операндов;
2. для каждого символа в строке postfix: если символ является переменной, ее значение добавляется в стек operands;
3. если символ является оператором:
4. если оператор — отрицание (!), извлекается верхний элемент стека, применяется операция отрицания, и результат помещается обратно в стек;
5. в противном случае (другие логические операторы), извлекаются два верхних элемента стека, применяется операция, и результат помещается обратно в стек.

После обработки всех символов вычисляется конечный результат выражения для текущей строки таблицы истинности и добавляется в соответствующую ячейку таблицы.

Функция string SKNF (int num\_rows, int num\_columns, int \*\*table, string str\_copy) строит совершенную конъюнктивную нормальную форму (СКНФ) для введенной пользователем логической формулы на основе ее таблицы истинности.

Алгоритм функции выглядит следующим образом:

1. выбираются строки таблицы истинности, в которых формула принимает значение «0»;

string SKNF;

int k = 0;

for (int i = 0; i < num\_rows; i++)

if (table[i][num\_columns - 1] == 0) k++;

1. если таких строк нет, пользователю выводится результат, что СКНФ для данной формулы не существует;

if (k == 0)

SKNF = "СКНФ НЕ СУЩЕСТВУЕТ!";

1. иначе по каждой из выбранных строк строят элементарную дизъюнкцию, содержащую все переменные формулы, по следующему правилу: если переменная принимает значение «1», то она входит в дизъюнкцию вместе со своим отрицанием, если же она принимает значение «0», то она входит без отрицания;
2. из полученных элементарных дизъюнкций составляют конъюнкцию.

else {

k = 0;

for (int i = 0; i < num\_rows; i++)

{if (table[i][num\_columns - 1] == 0)

{if (k > 0) SKNF.push\_back('\*');

SKNF.push\_back('(');

for (int j = 0; j < num\_columns - 1; j++)

{if (table[i][j] == 0)

SKNF.push\_back(str\_copy[j]);

else

{SKNF.push\_back('!');

SKNF.push\_back(str\_copy[j]);}

if (j < num\_columns - 2)

SKNF.push\_back('+');}

SKNF.push\_back(')');

k++;}}}

Результатом работы функции является переменная строкового типа, которая либо содержит построенную СКНФ функции, либо сообщение о том, что СКНФ для данной логической формулы построить нельзя.

Аналогичным образом работает функция для построения СДНФ введенной формулы string SDNF (int num\_rows, int num\_columns, int \*\*table, string str\_copy).

С помощью функции string tip (int num\_rows, int num\_columns, int \*\*table) определяется тип логической формулы. Она содержит ряд различных вложенных условных операторов для проверки на принадлежность формулы к определенному классу. Эти условия напрямую диктуются правилами, описанными в первой главе. Главным элементом проверки является вычисленная таблица истинности.

Для проверки логического следования пользователь вводит в переменную строкового типа посылки, перечисленные через запятую, и заключение, которое отделяется от посылок значком « / ». Из-за синтаксических отличий строки, в которой теперь хранится не одно выражение, а некоторое количество посылок и заключение, необходимо было написать код программы для проверки корректности ввода такого рода выражения.

Для этой цели была реализована логическая функция bool proverka\_logical (string str), в которой осуществляются проверки на единственность заключения и корректность перечисления посылок через запятую.

Функция возвращает true («истинна»), если строка проходит все проверки, и false («ложь») в противном случае, что свидетельствует о некорректности введенного выражения.

Функция string log\_sled (int num\_rows, int num\_columns, int \*\*table) осуществляет проверку правильности логического следования в соответствии с его определением, т.е. для всех строк таблицы истинности, где все посылки истинны, заключение также должно быть истинно.

Данная функция итерируется по всем строкам таблицы истинности и в каждой строке подсчитывается количество первых num\_columns-1 столбцах (посылках).

string type;

for (int i = 0; i < num\_rows; i++)

{int ist = 0;

for (int j = 0; j < num\_columns-1; j++)

if (table[i][j] == 1) ist++;

Если все посылки истинны и заключение также истинно, то функция возвращает переменную строкового типа, в которой содержится сообщение о том, что введенное выражение является логическим следствием.

if (ist == (num\_columns-1) && table[i][num\_columns-1] == 1)

{type = "Является логическим следствием";

return type; }}

type = "Не является логическим следствием";

return type;

Для проверки равносильности двух логических формул пользователь вводит в переменную строкового типа две формулы, отделенные между собой значком « = ». Аналогично логическому следствию, для данного выражения необходимо дополнительная проверка на корректность.

Данную проблему помогает решить логическая функция bool proverka\_ravn (string str), в которой осуществляется проверка на единственность оператора сравнивания двух логических формул.

После успешных проверок на корректность введенного выражения, функция string ravnosilnost (int num\_rows, int num\_columns, int \*\*table) осуществляет проверку равносильности двух формул с помощью вычисленной таблицы истинности. Если формулы принимают одинаковые значения в каждой строке таблицы истинности при подстановке одинакового набора пропозициональных переменных, содержащихся в данных логических формулах, то они являются равносильными. Поэтому для каждой строки сравниваются значения в последних двух столбцах таблицы. Если значения совпадают, увеличивается счетчик.

string ravn;

int k;

for (int i = 0; i < num\_rows; i++)

if (table[i][num\_columns - 2] == table[i][num\_columns - 1])

k++;

После перебора всех строк проверяется, равен ли счетчик количеству строк таблицы истинности. Если это так, значит, результаты всех строк совпадают, и формулы являются равносильными.

if(k%num\_rows==0)

ravn.append("Формулы являются равносильными :)");

else ravn.append("Формулы не являются равносильными :(");

return ravn;

Результатом работы функции является переменная строкового типа, которая содержит информацию о равносильности двух формул.

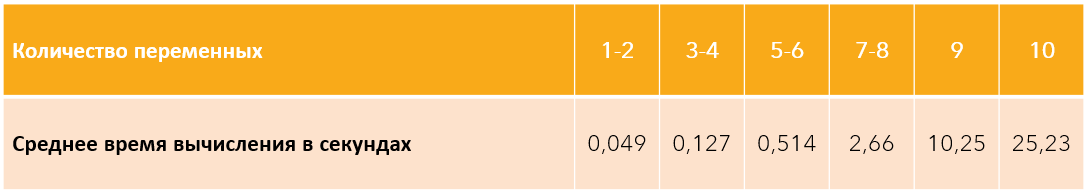
# 3 Анализ скорости выполнения программы

Язык программирования С++ является одним из самых мощных и производительных языков, что позволяет создавать высокоэффективные и быстрые программы.

Язык программирования C++ славится своей скоростью и эффективностью благодаря следующим особенностям:

1. программы на данном языке компилируются в машинный код, который выполняется напрямую процессором, что минимизирует задержки выполнения;
2. язык С++ имеет низкоуровневый контроль, т.е. предоставляет программисту возможность работать с памятью напрямую, что позволяет оптимизировать использование ресурсов;
3. стандартная библиотека C++ включает множество оптимизированных алгоритмов и структур данных, что ускоряет разработку и улучшает производительность программ;
4. компиляторы C++ обладают широкими возможностями для оптимизации кода, что позволяет значительно улучшить скорость выполнения программ.

Для анализа быстродействия программы логического калькулятора была проведена серия экспериментов с различным количеством уникальных переменных. В каждом эксперименте измерялось время вычисления в программе на примере построении таблицы истинности конъюнкции всех уникальных переменных. Результаты измерений показаны на рисунке 2.

Рисунок 2 — Время работы программы

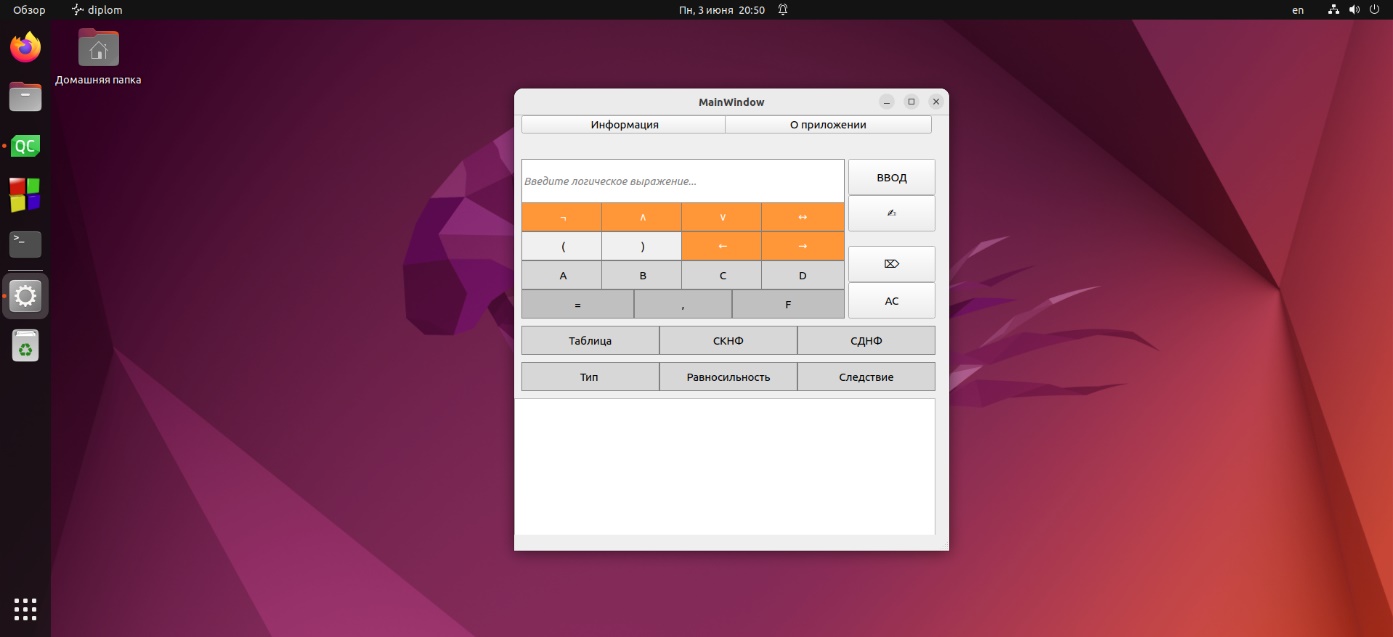
Из таблицы видно, что время выполнения программы увеличивается с ростом количества уникальных переменных. Это ожидаемое поведение, так как большее количество переменных требует дополнительных вычислений.

Таким образом, программа работает для выражения любой сложности. Нет ограничения на количество операций и повторения операндов. Количество возможных уникальных операндов — 52 буквы (английские буквы верхнего и нижнего регистра). Скорость вычисления программы напрямую зависит от количество уникальных операндов, так как количество строк в таблице истинности (т.е. вычислений в таблице) вычисляется по формуле , где *n* — количество уникальных операндов в формуле.

Для учебных целей, как правило, достаточно 3–5 уникальных операндов. Для такого количества вычислений таблицы истинности программа работает достаточно быстро, поэтому является удобной в использовании. Более того, даже при увеличении количества уникальных операндов, использование эффективных алгоритмов программы позволяют поддерживать приемлемую производительность. Это делает её подходящей не только для учебных, но и для исследовательских целей, где требуется обработка более сложных логических выражений.

# 4 Составление контрольных примеров для отладки программы

При запуске программы в QT Creator открывается главное окно — приложение логического калькулятора. Оно включает в себя все необходимые кнопки, к которым подключены описанные выше функции (обработчики событий, активирующиеся при нажатии на соответствующую кнопку). Интерфейс программы интуитивно понятен, поэтому не требует детального описания. Внешний вид главного окна программы логического калькулятора продемонстрирован на рисунке 3.

Рисунок 3 — Главное окно программы

В левом верхнем углу находится кнопка «Информация». При нажатии на данную кнопку открывается дополнительное окно, в котором находится краткая инструкция эксплуатации приложения, которая предлагается пользователю для ознакомления перед использованием программы. В ней описываются основные тонкости корректного ввода логических формул, чтобы предупредить возможность ошибочного ввода формулы.

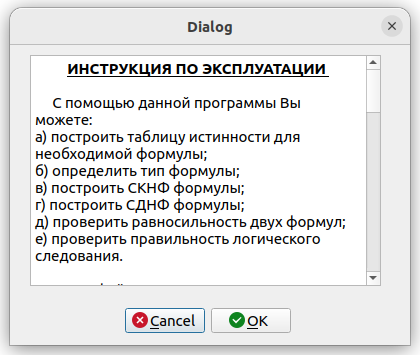


Рисунок 4 — Инструкция к приложению

При запуске программы автоматически подсвечивается окно для ввода логической формулы. Пользователю предлагается ввести формулу, содержащую до четырех переменных, с помощью кнопок в приложении. Однако есть возможность ввода формулы с клавиатуры, что увеличивает максимальное количество используемых переменных до 52 букв латинского алфавита (латинские буквы верхнего и нижнего регистра). Также логическую формулу можно ввести с помощью текстового файла, записав в него формулу согласно инструкции, или же скопировать и вставить нажатием правой кнопкой мыши по окну ввода.

Интерфейс калькулятора разработан таким образом, чтобы минимизировать возможность ошибочного ввода формулы. Для этого созданы кнопки редактирования формулы «Очистить» и «Удалить» (удаление одного символа или всей строки). Так же для удаления одного символа в случае ввода формулы с клавиатуры пользователь может использовать кнопку Backspace в окне ввода формул.

Проверка корректности введенных пользователем формул осуществляется при попытке нажатия кнопки «Ввести». Таким образом, пользователь может водить логическую формулу любым из доступных способов без каких-либо ограничений на данную формулу, которая может быть введена некорректно. После этого пользователю необходимо нажать на указанную выше кнопку, запустив обработчик событий для проверки корректности введенной формулы. Если формула введена некорректно, высвечивается окно QMessageBox::warning с соответствующим предупреждением и программа приостанавливает свою работу, пока не будет введена новая формула (рисунок 6).

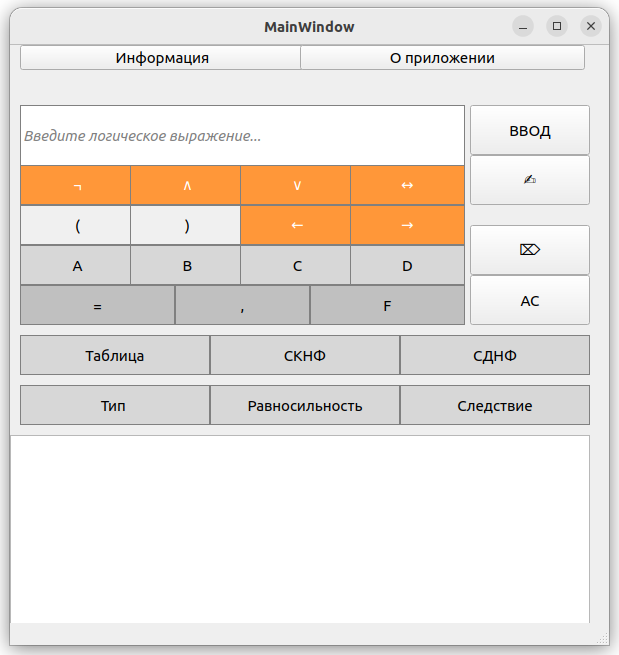


Рисунок 5 — Интерфейс приложения

Программа может обрабатывать шесть логических операций (инверсия, конъюнкция, дизъюнкция, импликация, репликация, эквивалентность) причем их приоритетность соответствует порядку следования операций в формате чтения (слева направо). Следует отметить, что пользователь имеет возможность изменять приоритетность на свое усмотрение, используя скобки.

Для отладки программы были разработаны примеры, тестирующие ее в тех ситуациях, в которых предполагается наличие ошибок.

Для функции, отвечающей за построение таблицы истинности примером проверки корректной работы является введенная логическая формула, в которой находятся переменные без каких-либо логических операций. В данном случае при попытке нажатия на кнопку «Ввести» отобразиться новое окно с предупреждением, показанное на рисунке 6.

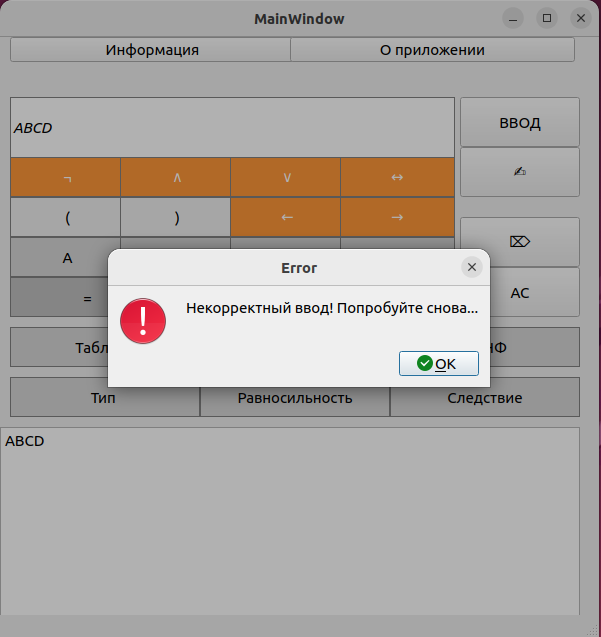


Рисунок 6 — Демонстрация некорректного вода формулы

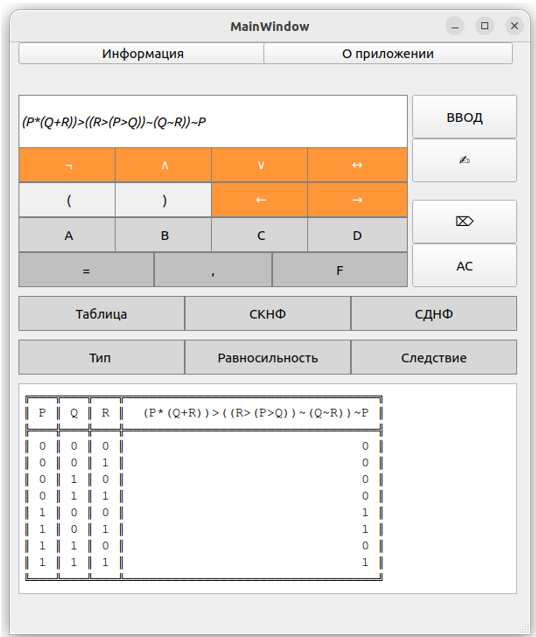


Рисунок 7 — Демонстрация корректного вода формулы

Для тестирования функции проверки равносильности двух логических формул необходимо проверить оба возможных случая: случай, когда формулы являются равносильными и случай, когда они таковыми не являются.

Для выявления конфликтных ситуаций было принято решение использовать для проверки на равносильность две похожие логические формулы, различающихся в одной логической операции.

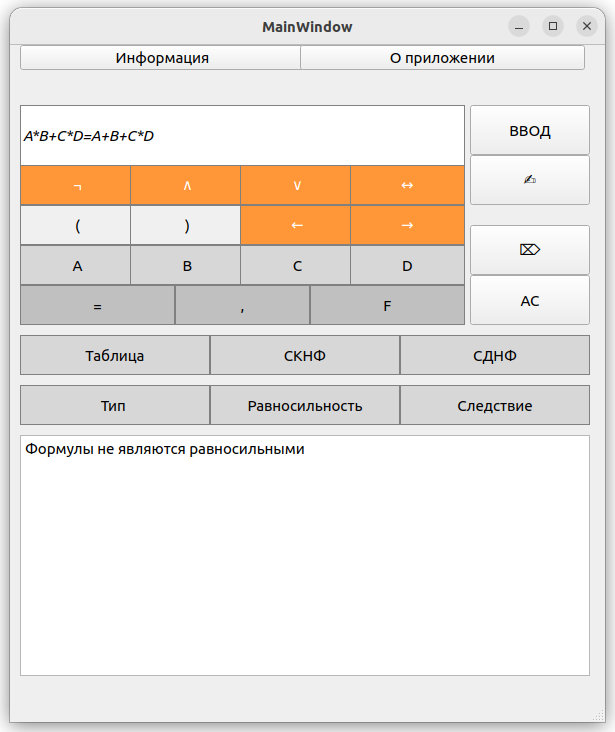


Рисунок 8 — Демонстрация проверки равносильности формул

Как и ожидалось, программа выдает корректный результат, так как то, что для человеческого взгляда может показаться похожим и, в случае ошибки, идентичным, для запрограммированного алгоритма является абсолютно разным.

Для тестирования функции проверки верности логического следования необходимо проверить соответственно два случая: заключение является логическим следствием посылок и оно таковым не является.

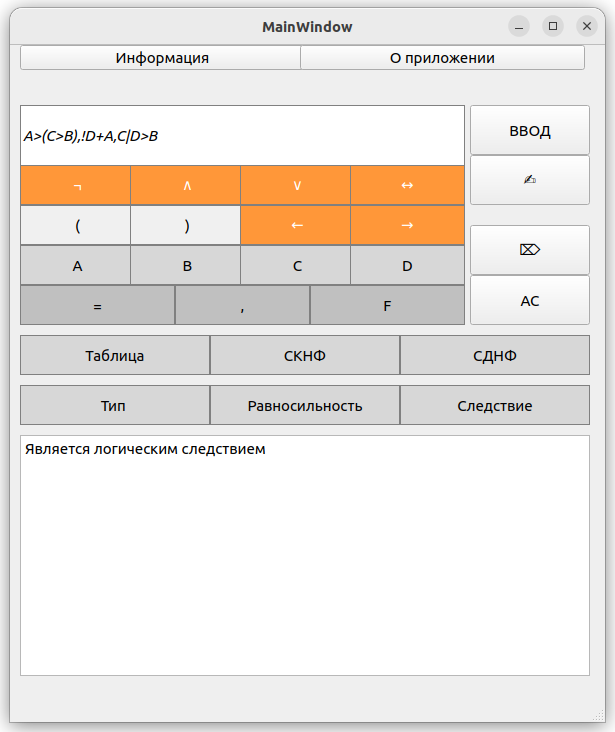


Рисунок 9 — Демонстрация верного логического следования

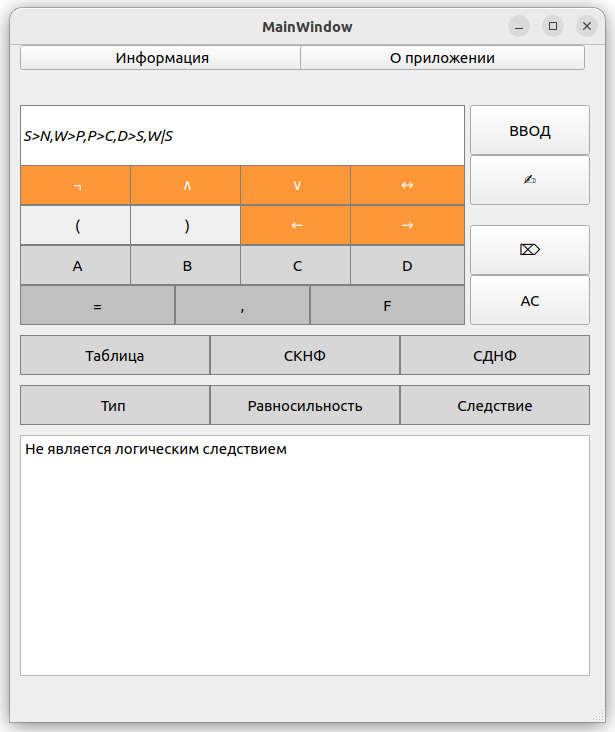


Рисунок 10 — Демонстрация неверного логического следования

Для проверки функции, отвечающей за определение типа логической формулы, необходимо подобрать три различных тестовых примера: для определения тождественно истинной, тождественно ложной и выполнимой (не тождественно истинной) формулы.

Формула является тождественно истинной, так как при подстановке любого набора пропозициональных переменных, входящих в данную формулу, итоговый столбец таблицы истинности заполнен только значениями «1».

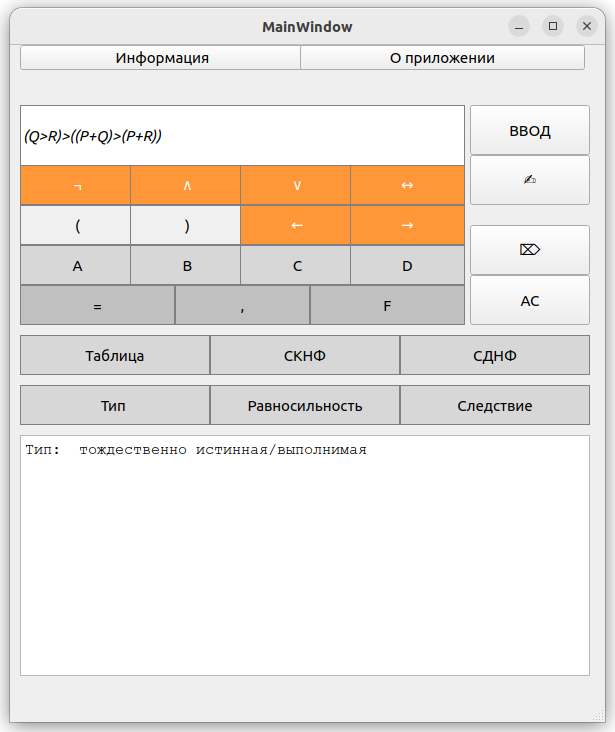


Рисунок 11 — Демонстрация определения типа формулы на примере   
тавтологии

Формула является тождественно ложной, так как при подстановке любого набора пропозициональных переменных, входящих в данную формулу, итоговый столбец таблицы истинности заполнен только значениями «0».

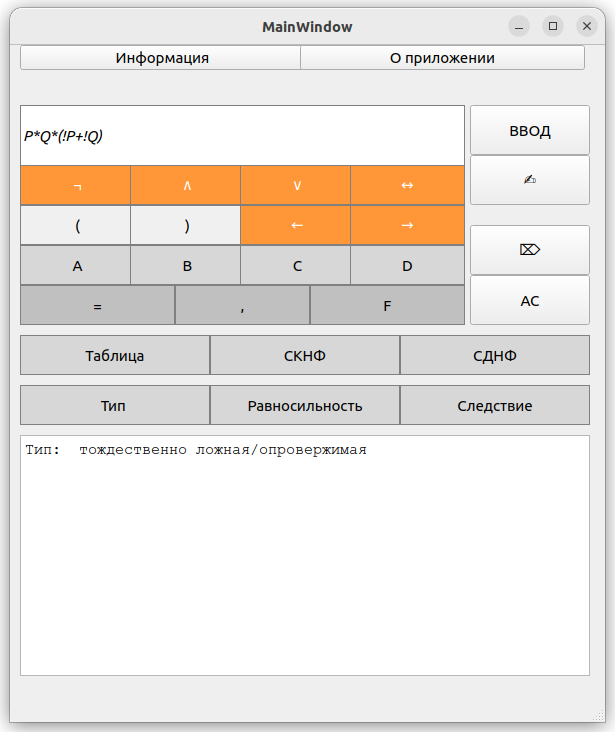


Рисунок 12 — Демонстрация определения типа формулы на примере   
противоречия

Формула является и выполнимой, и опровержимой одновременно, так как при подстановке любого набора пропозициональных переменных, входящих в данную формулу, итоговый столбец таблицы истинности содержит хотя бы одно значение «1», но в то же время состоит не только из единиц.

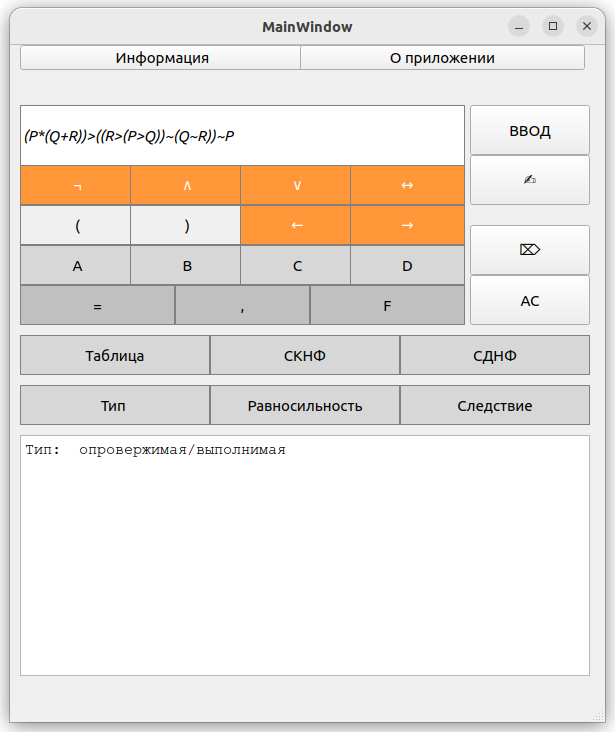


Рисунок 13 — Демонстрация определения типа формулы на примере   
опровержимой/выполнимой логической формулы

Тестирование функции построения СКНФ было принято осуществить на двух примерах: в случае, когда СКНФ существует и когда её не существует для данной логической формулы.

Тождественно истинная формула, рассмотренная ранее, станет примером для вычисления функции в случае, когда СКНФ формулы не существует.

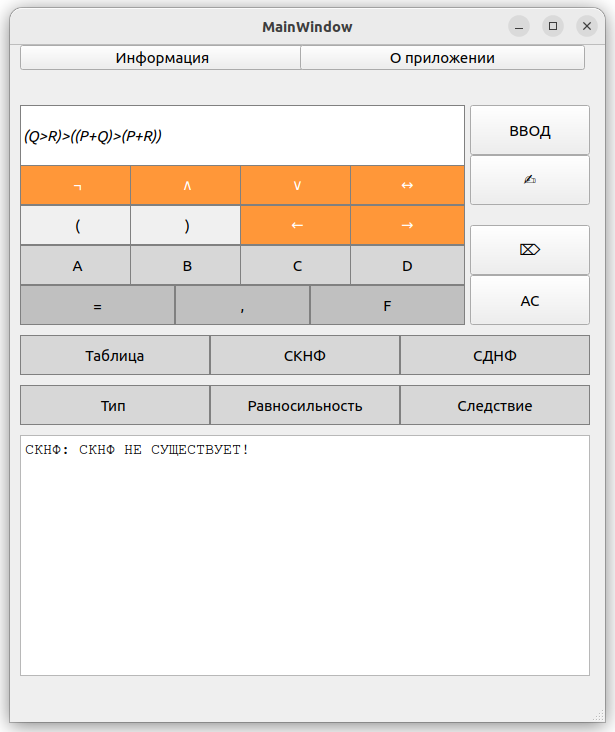


Рисунок 14 — Демонстрация построения СКНФ для тавтологии

Тождественно ложная формула станет примером для вычисления функции, когда СКНФ формулы существует.

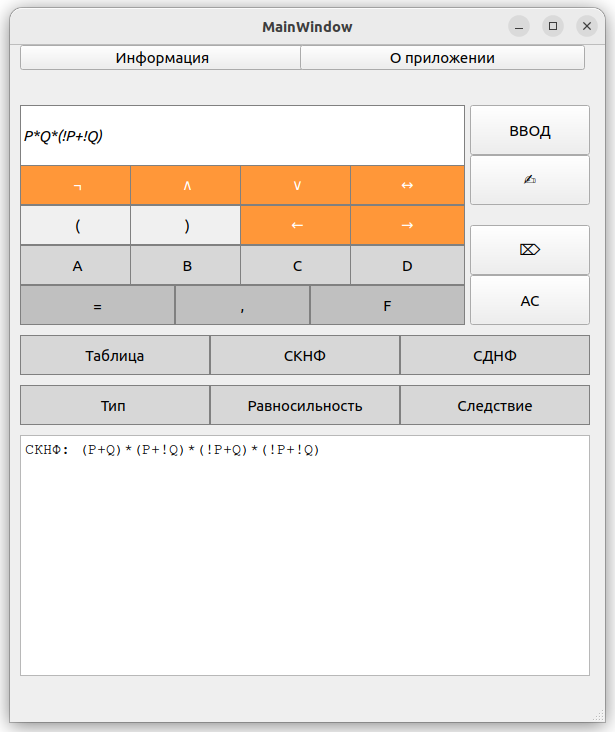


Рисунок 15 — Демонстрация построения СКНФ для противоречия

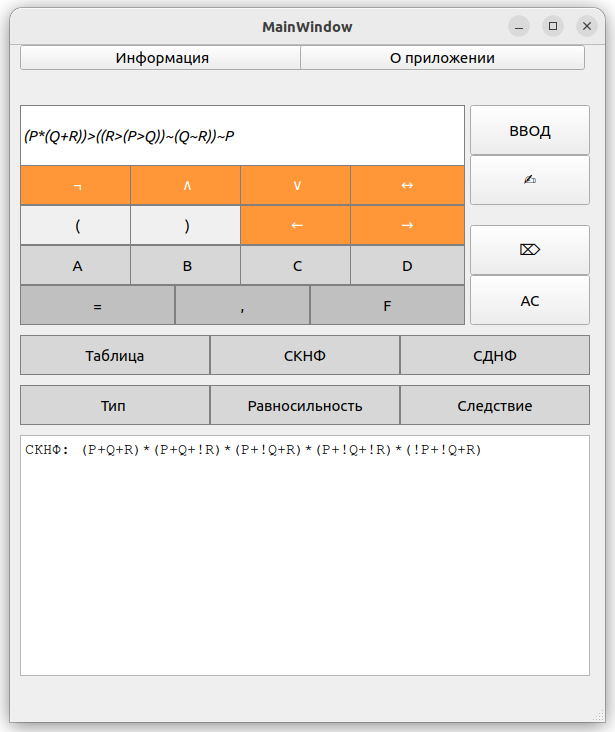


Рисунок 16 — Демонстрация построения СКНФ для   
выполнимой/опровержимой формулы

Аналогичным образом была протестирована функция, результатом которой является вычисленная СДНФ для введенной логической формулы.

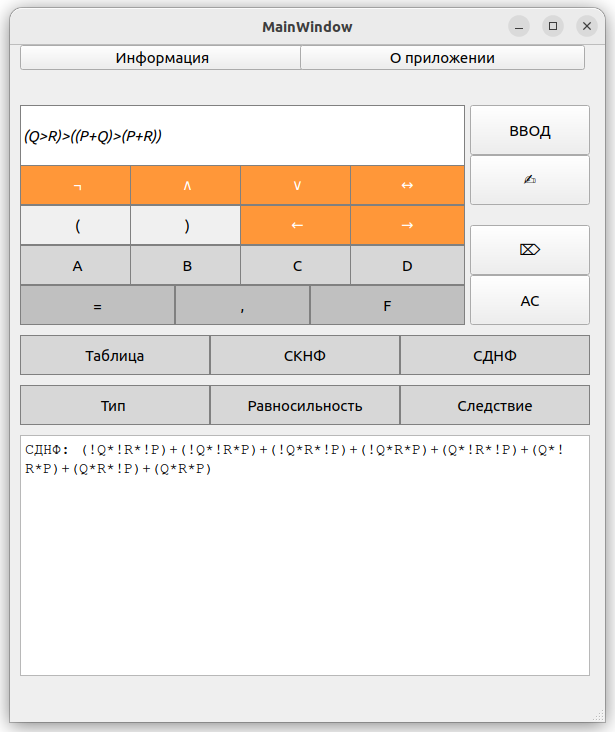


Рисунок 17 — Демонстрация построения СДНФ для тавтологии

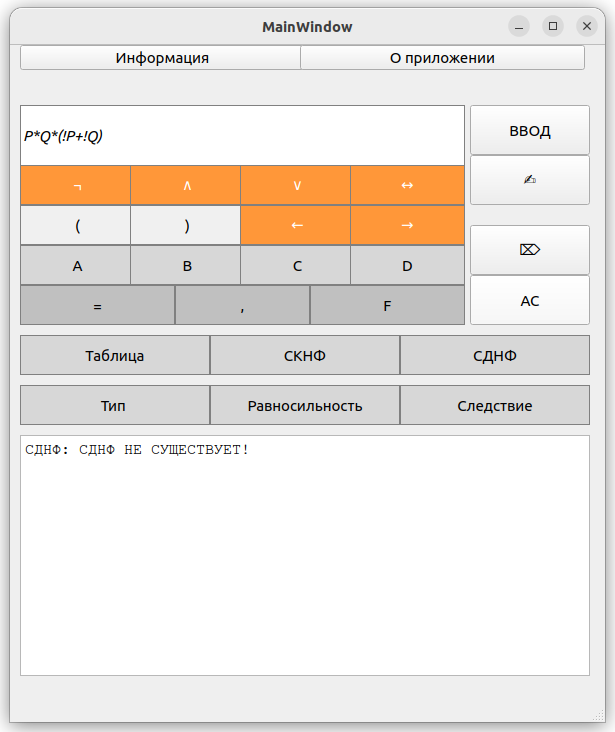


Рисунок 18 — Демонстрация построения СДНФ для противоречия

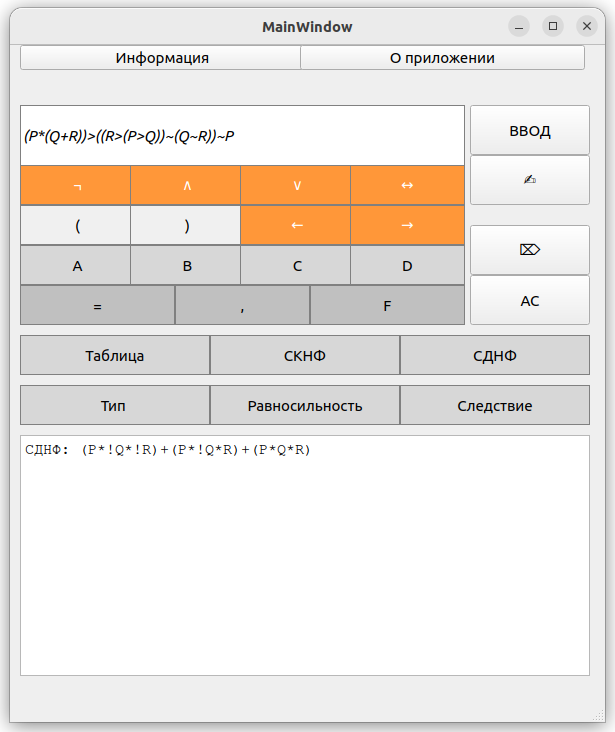


Рисунок 19 — Демонстрация построения СДНФ для   
выполнимой/опровержимой формулы

Таким образом, все функции программы были протестированы на различных наборах данных. Результаты тестирования подтвердили, что алгоритмы программы устойчивы к различным входным данным и правильно решают логические задачи в соответствии с заданными условиями.

# ЗАКЛЮЧЕНИЕ

В процессе исследования был проведен комплексный анализ поставленной задачи, разработаны индивидуальные алгоритмы решения всех компонентов задачи с последующим описанием на языке программирования С++. Программа была протестирована, а ее возможности продемонстрированы в соответствии с поставленными требованиями. Таким образом, цель исследования — создание логического калькулятора — была достигнута.

Созданная программа выгодно выделяется на фоне подобных разработок скоростью выполнения программы и алгоритмами обработки логических формул. Интерфейс приложения и его составляющие адаптированы под пользователя, которым может быть как студент в целях обучения, так и преподаватель для более быстрого и эффективного анализа проверочных работ обучающихся. Установка программы не займет много времени, так как она занимает небольшой объем памяти на жестком диске. Программа адаптирована для любой ОС семейства Linux и Windows.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Иванисова, О. В. Дискретная математика и математическая логика: учебное пособие / О. В. Иванисова, И. В. Сухан. — Москва: Берлин: Директ-Медиа, 2020. — 353 с. — ISBN 978-5-4499-1729-4.
2. Иванисова, О. В. Математическая логика. Алгебра высказываний: учебное пособие / О. В. Иванисова, И. В. Сухан, Г. Г. Кравченко; Кубанский государственный университет. — Краснодар: КубГУ, 2017. — 130 с.
3. Алексеев, Е. Р. Программирование на языке С++ в среде Qt Creator: учебное пособие / Е. Р. Алексеев, Г. Г. Злобин, Д. А. Костюк [и др.]. – Москва: ALT Linux, 2015. – 448 с. – ISBN 978-5-905167-16-4.
4. Обратная польская нотация — простое объяснение и практическое применение: сайт. – URL: https://nauchniestati.ru/spravka/obratnaya-polskaya-notacziya (дата обращения: 18.02.2024).
5. Игошин, В. И. Математическая логика и теория алгоритмов: учебное пособие / В. И. Игошин. – Москва: Издательский центр «Академия», 2004. –447 с. – ISBN 978-5-7695-4593-1.
6. Игошин, В.И. Задачи и упражнения по математической логике и теории алгоритмов: учебное пособие / В. И. Игошин. – Москва: Издательский центр «Академия», 2006. – 303 с. – ISBN 978-5-7695-3728-0.
7. ГОСТ Р 52653 Информационно-коммуникационные технологии в образовании. Термины и определения : национальный стандарт Российской Федерации: издание официальное : утвержден и введен в действие Приказом Федерального агентства по техническому регулированию и метрологии от 27 декабря 2006 г. N 419-ст : дата введения 2008–07–01 / подготовлен Техниче-ским комитетом по стандартизации ТК 461 «Информационно-коммуникационные технологии в образовании» – Москва : Стандартинформ, 2008, 18 с. – (Отраслевой специализированный ресурсный центр регистрации, стандартизации и сертификации информационных ресурсов системы образования).