Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования «Тульский государственный университет»
Институт прикладной математики и компьютерных наук

Кафедра «Информационная безопасность»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе по дисциплине

«Технологии и методы программирования»

на тему

«Разработка приложения с графическим интерфейсом для вычисления логических операций и построения таблиц истинности»

Выполнила: ст. гр. 230711 _____ (подпись) Павлова В.С.

Проверил: доцент каф. ИБ _____ (подпись) Баранов А.Н.

Тула, 2023

ЗАДАНИЕ

на курсовую работу по дисциплине

«Технологии и методы программирования»

студента гр. 230711 Павловой Виктории Сергеевны

Тема курсовой работы

«Разработка приложения с графическим интерфейсом для вычисления логических операций и построения таблиц истинности»

Исходные данные

Интегрированная среда разработка Visual Studio Community 2022, интерфейс программирования приложений Windows Forms, язык программирования C#, операционная система Windows 10 Home (версия 21H2)

Задание получил _____ _____
(ФИО) (подпись)

Задание выдал _____ _____
(ФИО) (подпись)

Дата выдачи задания 21.02.2023 г.

График выполнения КР 21.02-28.02 – Получение и ознакомление с заданием

01.03-22.03 – Изучение литературы и других исходных материалов

23.03-03.05 – Изучение теории, раскрывающей тему курсовой работы

04.05-17.05 – Разработка программной реализации курсовой работы

18.05-24.05 – Анализ результатов

25.05-07.06 – Оформление пояснительной записки и сдача на проверку

27.06.2023 – Защита курсовой работы

Рекомендации и особые отметки _____

«__» _____20__г

Содержание

Введение

Развитие информационных технологий неумолимо набирает скорость: всего за полвека они изменились до неузнаваемости. От первых машинных языков и ассемблеров до современных высокоуровневых языков прошло много лет. Современные языки программирования отличаются высоким уровнем абстракции, что позволяет разработчикам создавать приложения быстро и эффективно. Новейшие среды разработки обрели большое количество инструментов и функций, таких как, например, автодополнение, отладка и система контроля версий, что тоже упростило и ускорило процесс создания приложений. Кроме того, существует большое количество библиотек и фреймворков, которые позволяют разработчикам использовать готовые решения для типовых задач – это также существенно ускоряет процесс разработки и повышает качество создаваемого приложения.

Приложения с графическим интерфейсом являются наиболее распространенным типом приложений. Неоспоримым фактом является то, что знание современных языков программирования и сред разработки – важный навык для разработчиков программного обеспечения, поэтому, актуальность выбранной темы и необходимость изучения современных языков программирования, а также сред разработки приложений не подлежит сомнению, в особенности в контексте направлений подготовки специалистов, чья работа связана с программированием сложных систем.

Данная курсовая работа посвящена изучению принципов и этапов разработки современных приложений, а также применения этих принципов на практике в ходе разработки приложения с графическим интерфейсом, его отладки и тестирования. Цель работы заключается в создании полностью функционального приложения для выполнения логических операций и построения таблиц истинности с использованием графического интерфейса.

I. Современные языки и среды разработки приложений

1.1 Эволюция языков программирования и программных средств

История языков программирования берёт своё начало в сороковых годах XX-го века. Программируемые машины того времени принимали на вход команды, состоящие лишь из нулей и единиц, что делало процесс написания программ трудоёмким и скрупулёзным процессом, можно даже сказать настоящим искусством. Переход к ассемблерам стал первым шагом в развитии языков программирования, поскольку ассемблер позволил отказаться от двоичных имён для команд и регистров в пользу более понятных лексем. Однако такие языки низкого уровня всё равно имели недостатки, в частности, привязка к архитектуре конкретной ЭВМ. [1]

Закономерным этапом развития языков низкого уровня стал переход к более универсальным машинонезависимым языкам, а также использование трансляторов. Первым высокоуровневым языком программирования стал Фортран (название образовано от «formula translator»), созданный в 1957 году. Он был специально разработан для научных вычислений и был ориентирован на математические операции. Далее в 1972 году появился язык C (Си), который был более универсальным языком программирования, и его можно было использовать для различных задач, а также Паскаль (Pascal), Бейсик (BASIC) и другие. [1]

Шло время, мир менялся, росла потребность в вычислительных мощностях и автоматизации бизнес-логики, а также, как пишет В.Ш. Кауфман, в различных программных услугах. Росли также объёмы получаемых и создаваемых данных. Появлялось всё больше новых языков программирования, большинство из которых поддерживало создание отдельных структур, содержащих переменные и работающие с ними функции. В 1970-1980 парадигма сменилась на объектно-ориентированную (ООП). Она включила в себя наработки предыдущего этапа, а

уровень абстракции развился до таких понятий, как полиморфизм и инкапсуляция. [2]

Современные высокоуровневые языки программирования, такие как Java, Python, C# и C++, используются для широчайшего спектра приложений, от веб-программирования до искусственного интеллекта и машинного обучения. Они позволяют программистам более эффективно и быстро создавать программы, что делает их незаменимыми в современном мире информационных технологий. С активным развитием Интернет-сети появляется потребность в реализации графических интерфейсов, появляется разделение на так называемый «фронтенд», с помощью которой происходит визуализация и создание внешней оболочки приложений, и «бэкенд», который занимается хранением и обработкой данных.

Вместе с самими языками развивались и изменялись среды программирования и программные средства, то есть так называемые IDE (Integrated Development Environment). На заре эпохи развития программирования, то есть до появления ассемблеров и высокоуровневых языков программирования, программисты писали код на языке машинных команд, который представлял собой набор чисел и инструкций, понятный только компьютеру. На тот момент не было сред разработки в современном понимании. С появлением ассемблеров, которые представляли собой низкоуровневый язык программирования, программисты стали писать код на нем, однако средства разработки в те времена были довольно простыми, состояли из текстовых редакторов и компиляторов, и не обладали такими функциями, как отладка кода, подсветка синтаксиса и т.д. [3]

В конце XX века развитие сред разработки ускорилось: были созданы новые интегрированные среды разработки, такие как Microsoft Visual Studio и Borland Delphi, которые предоставляли более продвинутые функции, среди которых средства отладки и профилирования, системы контроля версий, интеграция с другими средствами разработки, такими как базы данных, и т.д.

Сегодня среды разработки являются важным инструментом для программистов и разработчиков. Они предоставляют множество функций и инструментов для написания, отладки, тестирования и совместной работы над кодом. Одними из наиболее популярных сред разработки на сегодняшний день являются Visual Studio, Eclipse, IntelliJ IDEA, NetBeans и PyCharm, которые предоставляют различные функции и инструменты, чтобы помочь программистам работать максимально эффективно.

Появление средств разработки приложений с графическим интерфейсом связано с развитием графических пользовательских интерфейсов (GUI) в операционных системах. Создание приложений с GUI было сложным и трудоёмким процессом, требующим знаний не только языков программирования, но и графического дизайна и работы с пользовательским интерфейсом. Одним из первых средств разработки приложений с графическим интерфейсом был Visual Basic, созданный в 1991 году компанией Microsoft. Visual Basic позволял разработчикам создавать приложения с GUI, используя простой и интуитивно понятный язык программирования и инструменты для работы с элементами пользовательского интерфейса. Позже появился набор инструментов Windows Forms для языка программирования C#, который позволяет разработчикам создавать приложения с GUI для операционных систем Windows. Этот инструментарий также предоставляет библиотеку классов .NET для работы с элементами пользовательского интерфейса. [4]

Другим средством разработки приложений с GUI является Qt, кроссплатформенный набор инструментов и библиотек для разработки приложений на C++, который позволяет создавать приложения с GUI для разных операционных систем, включая Windows, Linux и macOS. Современные средства разработки приложений с GUI, такие как Microsoft Visual Studio и Qt Creator, предоставляют широкий набор инструментов и возможностей для работы с пользовательским интерфейсом, включая дизайнеры интерфейса, редакторы

кода и инструменты отладки. Они значительно упрощают и ускоряют процесс разработки приложений с графическим интерфейсом. [3]

1.2 Место языка С и оболочки QT-Creator среди других языков и программных сред

Значение языка программирования Си в программной индустрии сложно переоценить. Си является одним из самых популярных языков программирования и используется во многих областях, включая системное программирование, научные и инженерные вычисления, разработку приложений, работу с базами данных, создание игр и многие другие области. Основными особенностями языка Си являются: [5]

- возможность написания производительного кода;
- поддержка работы с указателями;
- простой синтаксис и низкий порог вхождения;
- модульность и возможность повторного использования кода.

Си является одним из самых популярных языков программирования в мире, и существует множество различных компиляторов, сред разработки и библиотек, поддерживающих Си. Одним из ключевых компиляторов для языка Си является gcc, который широко используется в Linux-среде и многих других операционных системах. Среди сред разработки для Си можно выделить, например, Code::Blocks, Visual Studio Code, Dev-C++ и другие. Также Си является основой для различных других языков программирования, включая C++, C#, Objective-C, Rust и другие.

Одной из интегрированных сред разработки (IDE) для создания приложений является QT-Creator. Это свободное программное обеспечение с открытым исходным кодом, разработанное компанией Qt Company. Говоря о Qt Creator, основное значение его заключается в том, что он позволяет разрабатывать кроссплатформенные приложения, которые могут работать на

различных операционных системах. Qt Creator обеспечивает интуитивно понятный интерфейс для создания, отладки и тестирования приложений.

1.3 Достоинства и недостатки языка С и оболочки QT-Creator

Си обеспечивает более прямой доступ к ресурсам компьютера, что позволяет программистам создавать более быстрые и эффективные программы, чем на более высокоуровневых языках, таких как Python или Ruby. Вопреки тому, что в последние годы появилось множество новых языков, Си по-прежнему остается популярным и актуальным языком программирования. Например, он используется в ядрах операционных систем, драйверах устройств, встроенных системах, в разработке приложений для мобильных устройств и других областях. Кроме того, С используется для написания системного программного обеспечения, операционных систем, встроенных систем, драйверов устройств и т.д. Языку Си присущи многие достоинства, в том числе: [5]

- высокая скорость и производительность;
- кроссплатформенность;
- низкий уровень абстракции;
- доступность и популярность.

Несмотря на то, что язык Си считается одним из самых популярных языков программирования в мире, он также имеет свои недостатки:

- необходимость вручную освобождать память, выделенную под объекты после их использования;
- отсутствие встроенной поддержки работы с строками;
- отсутствие проверки выхода за границы массива;
- отсутствие встроенной поддержки многопоточности;
- сложность работы с указателями;
- отсутствие встроенной поддержки ООП;

Среда разработки QT Creator позволяет создавать кроссплатформенные приложения с графическим интерфейсом, используя языки C++ и QML. Она обладает рядом преимуществ, в том числе: [3]

- интеграция с библиотекой Qt, которая предоставляет широкие возможности для создания графических интерфейсов;
- наличие встроенного отладчика и инструментов профилирования;
- удобная среда для создания и управления проектами.

Среди недостатков Qt Creator можно выделить тот факт, что он ориентирован на язык C++ и библиотеку Qt, что ограничивает использование других языков программирования и инструментов разработки. С учётом всех вышеперечисленных особенностей для разработки приложения с графическим интерфейсом (GUI) в качестве альтернативы QT-Creator в данной курсовой работе выберем Windows Forms и язык программирования C#.

II. Постановка задачи программирования приложения с графическим интерфейсом

2.1 Цель и задачи разработки

Целью данной курсовой работы является создание приложения с графическим интерфейсом для вычисления логических операций и построения таблицы истинности. Для достижения этой цели разработку можно разбить на следующие задачи:

1. Создание логики приложения:
    1) Реализация вычисления основных булевых функций;
    2) Создание обработчика входной строки с формулами алгебры логики;
    3) Реализация преобразования постфиксной записи из входной формулы;
    4) Распознавание основных операторов булевой алгебры;
2. Проектирование графического интерфейса приложения:
    1) Описание окна приложения и элементов управления;
    2) Настройка использованных элементов управления;
    3) Оформление визуальной части приложения;
3. Связь визуальных элементов приложения с логикой с помощью обработчиков событий;
4. Тестирование приложения:
    1) Проверка программных вычислений путём ручных расчётов;
    2) Оценка правильности получаемых таблиц истинности;
5. Оценка общей надёжности разработанного приложения:
    1) Подведение итогов разработки на основе полученных тестов;
    2) Оценка достижения поставленных целей и полноты выполнения задач.

2.2 Перечень автоматизированных функций

Поскольку конечной целью в рамках данной курсовой работы является разработка калькулятора булевых выражений, необходимо предусмотреть возможность ввода формул, описать основные правила синтаксиса калькулятора, реализовать их корректный парсинг, а также научить программу отличать операторы от операндов.

Наложим следующие ограничения на функциональность программы для обеспечения корректных вычислений:

1. Формула подаётся строго в соответствии с правилами синтаксиса калькулятора, приведёнными в инструкции пользователя в разделе IV.
2. Скобки следует использовать для явного указания порядка выполнения операций и для установления приоритетов в выражениях, в которых используются различные операции;
3. Среди операций должны быть лишь те, которые предусмотрены программной реализацией, а именно: отрицание, конъюнкция, дизъюнкция, функция Шеффера, функция Пирса, исключающее «ИЛИ», импликация и эквивалентность.

В качестве основной формы хранения данных будем использовать стек. Конечный вид формулы после обработки будет представлять собой постфиксную запись. Под постфиксной формой записи (также известной как обратная польская запись или Reverse Polish Notation, RPN) понимается форма записи математических выражений, в которой операторы следуют за операндами. [6]

Например, для выражения $f(x, y) = a \oplus b$ постфиксная запись будет иметь вид «ab⊕». Такой формат представления данных более практичен, поскольку в постфиксной записи операции выполняются последовательно, по мере обработки операндов, что делает вычисления проще и менее подверженным ошибкам. Помимо этого, в постфиксной записи отсутствуют скобки, поскольку порядок выполнения операций определяется порядком операндов, и,

следовательно, её проще обрабатывать. В качестве примера рассмотрим некоторую функцию $f(x, y, z) = \big((x \,\&\, y)\sim z\big)|\, 1$. Её постфиксная запись имеет вид: $f(x, y, z) = xy\&z\sim 1|$. С применением стека обработка данных будет выглядеть так, как показано на рисунке 1:

	y		z		1	
x	x	x&y	x&y	x&y~z	x&y~z	$f(x, y, z)$

Рисунок 1 – Заполнение стека постфиксной записью и вычисление выражения

Перечень функций, которые необходимо автоматизировать для достижения целей разработки, приведён ниже:

1) Метод для проверки, является ли символ оператором;
2) Метод для получения номера приоритета оператора;
3) Метод для формирования постфиксной записи;
4) Метод для вычисления выражения в постфиксной записи в соответствии с правилами передаваемой ему алгебры;
5) Общий метод обработки входной последовательности, который будет использован в обработчике событий входной строки.

Необходимо также предусмотреть такую структуру данных для вычисления функций алгебры логики, которая сможет работать со списком переменных и их значениями. Такой класс должен иметь некоторый ассоциативный контейнер для связи булевой переменной с её значением. Помимо этого, класс должен реализовывать следующие методы:

1) Метод доступа к словарю с переменными;
2) Метод для присвоения новых значений переменным;
3) Метод для получения данных о переменных от обработчика входной строки;

4) Методы вычисления булевых функций, перечисленных ранее;

III. Описание этапов разработки

3.1 Описание форм и виджетов

Сперва создадим и настроим саму форму: заменим стандартный фон на розовый, установив параметр BackColor в MistyRose, установим ей размер 818x497, название «Boolean Algebra Calculator», а также сменим иконку приложения, как показано на рисунке 2:

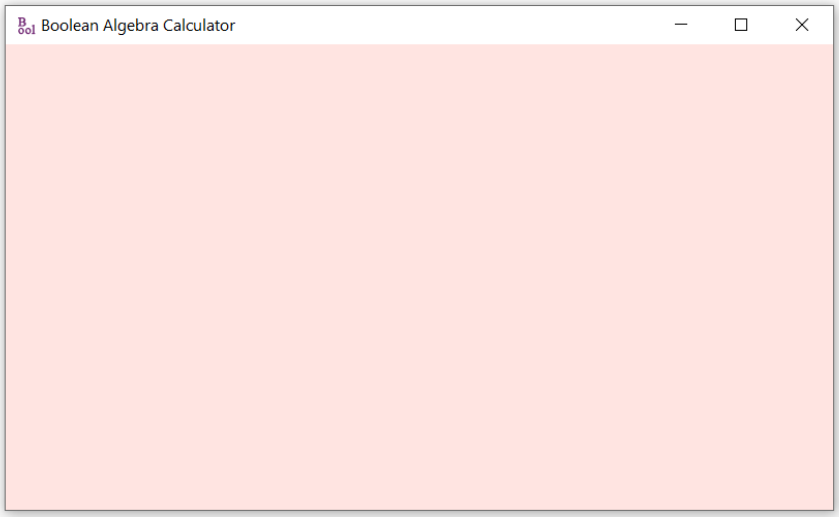

Рисунок 2 – Первичная форма без элементов управления

Теперь разместим на форме первые элементы управления: добавим строку ввода формул formulaTextBox – она приведена на рисунке 3 под цифрой 2. Соответствующая ей пояснительная надпись обозначена цифрой 1. Ниже, как показано на рисунке 3, добавим кнопку Button (цифра 3), которая будет использоваться для запуска вычислений:

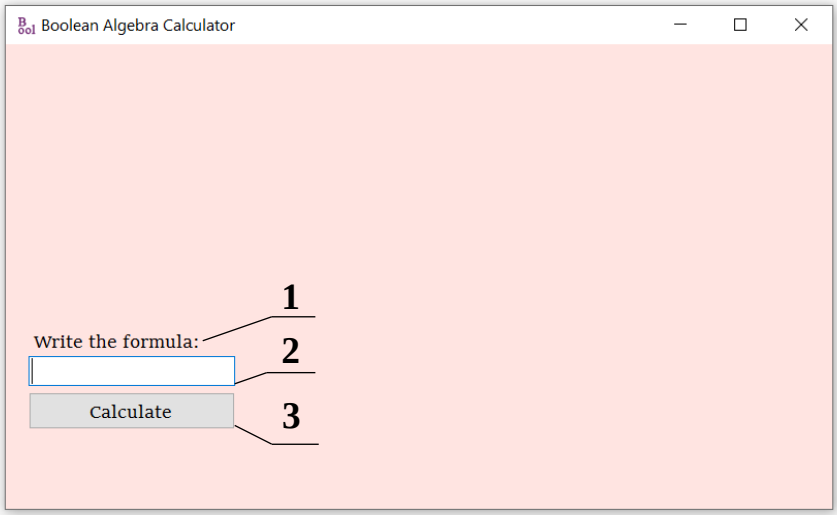

Рисунок 3 – Форма с первыми элементами управления

Над строкой ввода разместим ещё одно текстовое поле textBoxLabel, как показано на рисунке 4 (цифра 4), содержащее пояснение правил синтаксиса калькулятора. Необходимо использовать именно элемент TextBox для того, чтобы обеспечить возможность копирования обозначений функций из этого поля. Правее расположим таблицу формата DataGridView для отображения таблицы истинности. Она изображена на рисунке 4 пунктиром и не отображается на форме ввиду отсутствия введённых в программу данных.

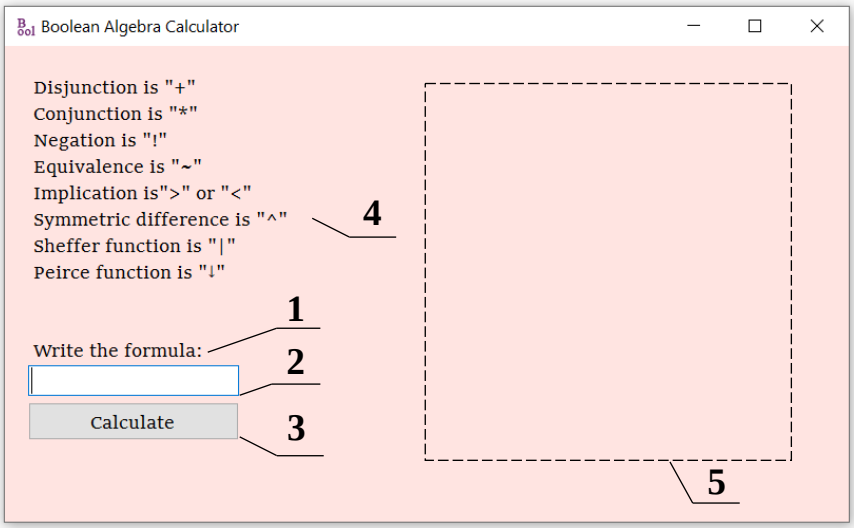

Рисунок 4 – Итоговый вид приложения-калькулятора при запуске

3.2 Описание кодовых конструкций

Разработанное приложение содержит следующие кодовые конструкции:

1) Основная программа-загрузчик (Program.cs);
2) Класс-обработчик Handler (Handler.cs);
3) Класс, описывающий булевы функции (BooleanFunctions.cs);
4) Класс, описывающий форму (Form.cs)

Код описания класса Handler на языке программирования C# приведён в листинге 1 приложения А. В таблице 1 приведено описание данного класса и список реализованных функций в соответствии с перечнем функций, подверженных автоматизации, из раздела II.

Таблица 1 – Описание разработанного класса Handler

Автоматизированные методы (функции-элементы) класса	
Название, тип возвращаемого значения, аргументы	**Описание**
bool IsOperator(char s)	Метод для проверки, является ли символ s оператором
byte GetPriority(char s)	Метод для получения приоритета оператора s
string ConvertToRPN(string input)	Метод для формирования постфиксной записи с использованием стека из входной строки input
int Counting(string input, BooleanFunctions booleanFunctions)	Метод для вычисления выражения input в постфиксной записи согласно правилам передаваемой ему алгебры booleanFunctions класса BooleanFunctions
int Calculate(string input, BooleanFunctions booleanFunctions)	Общий метод обработки входной последовательности input, который используется в обработчике событий входной строки и передает правила booleanFunctions

Описание класса, назначение которого вычислять функции алгебры логики, приведено в таблице 2, а код программной реализации на языке программирования C# приведён в листинге 1 приложения Б.

Таблица 2 – Описание разработанного класса BooleanFunctions

Поля/свойства (элементы данных) класса	
Название и тип	**Описание**
Dictionary<char, int> variables	Ассоциативный контейнер (словарь), хранящий пару ключ-значение для связи переменной и её значения
Автоматизированные методы (функции-элементы) класса	
Название, тип возвращаемого значения, аргументы	Описание
Dictionary<char, int> Variables	Метод доступа к словарю с переменными
void SetVariables(char[] keys, int[] values)	Метод для присвоения новых значений переменным
List<char> GetAllVariablesFromString(string str)	Метод для получения данных о переменных от обработчика входной строки
int Disjunction(int A, int B)	Метод для реализации дизъюнкции
int Conjunction(int A, int B)	Метод для реализации конъюнкции
int Negation(int A)	Метод для реализации логического отрицания
int Equivalence(int A, int B)	Метод для реализации эквивалентности
int Implication(int A, int B)	Метод для реализации импликации
int Nonequivalence(int A, int B)	Метод для реализации симметрической разности
int ShefferFunction(int A, int B)	Метод для реализации штриха Шеффера
int PeirceFunction(int A, int B)	Метод для реализации стрелки Пирса

В листинге 1 данного раздела описана основная программа, реализующая загрузку приложения, а в листинге 2 приведены кодовые конструкции,

описывающие форму, с помощью которой непосредственно реализуется функционал приложения.

Листинг 1. Описание программы-загрузчика Program.cs

```
using System;
using System.Collections.Generic;

namespace BooleanAlgebra
{
    class Program
    {
        static void Main(string[] args)
        {
            ApplicationConfiguration.Initialize();
            Form form = new Form();
            Application.Run(form);
        }
    }
}
```

Листинг 2. Описание формы программы-калькулятора Form.cs

```
using System.Data;
using System.Linq.Expressions;
using System.Windows.Forms;

namespace BooleanAlgebra
{

    public partial class Form : System.Windows.Forms.Form
    {

        private List<char> variables;
        private List<string> operators;

        public Form()
        {
            InitializeComponent();
        }
        private void CalculateClickEventHandler(object sender, EventArgs e)
        {
            try
            {   // Считать выражение из строки ввода
                string expression = formulaTextBox.Text;
                var variables =
BooleanFunctions.GetAllVariablesFromString(expression);
                var booleanFunctions = new BooleanFunctions();
                // Получить количество переменных и создать таблицу
                int numVariables = variables.Count;
                DataTable table = new DataTable();
                for (int i = 0; i < numVariables; i++)
                {
                    table.Columns.Add(variables[i].ToString());
                }
                table.Columns.Add("Result");
                // Генерация всех возможных комбинаций значений таблицы истинности
                int numRows = (int)Math.Pow(2, numVariables);
                for (int i = 0; i < numRows; i++)
                {
```

Листинг 2. Код описания формы Form.cs (продолжение)

```
                    var values = new List<int>();
                    DataRow row = table.NewRow();

                    for (int j = 0; j < numVariables; j++)
                    {
                        row[j] = Convert.ToString(i, 2).PadLeft(numVariables,
'0')[j];
                    }
                    for (int j = numVariables - 1; j >= 0; j--)
                    {
                        int bit = (i >> j) & 1;
                        values.Add(bit);
                    }
                    booleanFunctions.SetVariables(variables.ToArray(),
values.ToArray());
                    row["Result"] = Handler.Calculate(expression, booleanFunctions);
                    table.Rows.Add(row);
                }
                // Отображение таблицы с помощью DataGridView
                dgvTruthTable.DataSource = table;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Error");
            }
        }
        private int GetNumVariables(string expression)
        {
            // Получить количество всех уникальных символов в верхнем регистре
            return expression.Where(c => Char.IsUpper(c)).Distinct().Count();
        }
        private void calculateButton_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (e.KeyChar == (char)Keys.Enter)
            {
                CalculateClickEventHandler(sender, e);
                e.Handled = true;
            }
        }
    }
}
```

IV. Тестирование разработанного приложения

4.1 Описание инструкции пользователя

Разработанное приложение представляет из себя одно окно, на котором расположена строка для ввода формулы, а также кнопка для старта вычислений. Формула должна вводиться с учётом условных обозначений программы, которые приведены над строкой ввода. Логические конструкции должны быть отделены друг от друга скобками в соответствии с правилами булевой алгебры.

Список поддерживаемых операций, приоритет и их условные обозначения приведены в таблице 3:

Таблица 3 – Список определённых в программе операций и их обозначение

Приоритет	**Название**	**Условное обозначение**
0	Скобки	()
1	Отрицание	!
2	Конъюнкция	*
2	Штрих Шеффера	|
2	Стрелка Пирса	↓
3	Исключающее ИЛИ	^
3	Дизъюнкция	+
4	Импликация (x -> y)	>
4	Импликация (x <- y)	<
5	Эквивалентность	~

4.2 Оценка надежности разработанного приложения

Проведём тестирование разработанного калькулятора алгебры логики с помощью выражений различной сложности. Для начала рассмотрим простейшее выражение – дизъюнкцию: $f(x, y) = x \vee y$. Результат вычислений приложения, а также таблица истинности (т/и) дизъюнкции для сравнения представлены на рисунке 5. Как видно из рисунка, вычисления калькулятора корректны.

А) Б)

x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

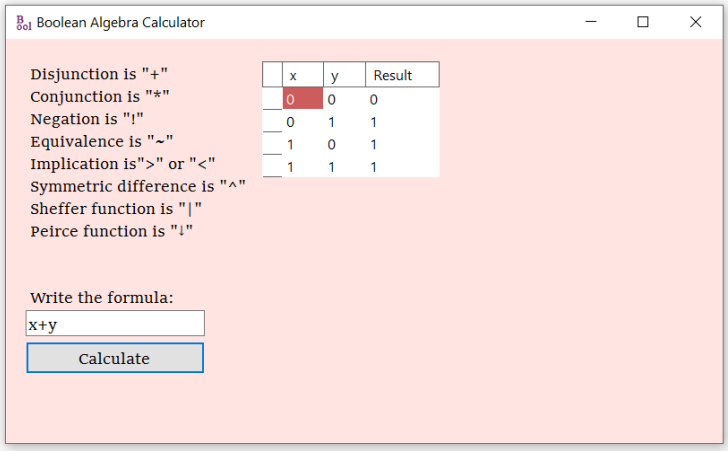

Рисунок 5 – Т/и для первого выражения, составленная: а) вручную; б) программно

Рассмотрим более сложное выражение $f(x,y,z) = \overline{x \sim y} \oplus z$, что на синтаксисе приложения можно записать как $f(x,y,z) = (!\,(x \sim y))\hat{}\,z$. Результат вычислений приложения, а также таблица истинности данной функции для сравнения представлены на рисунке 6. Как видно из рисунка, вычисления калькулятора снова оказались корректны.

А) Б)

x	y	z	$x \sim y$	$\overline{x \sim y}$	$f(x,y,z)$
0	0	0	1	0	0
0	0	1	1	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	0	1	1
1	0	1	0	1	0
1	1	0	1	0	0
1	1	1	1	0	1

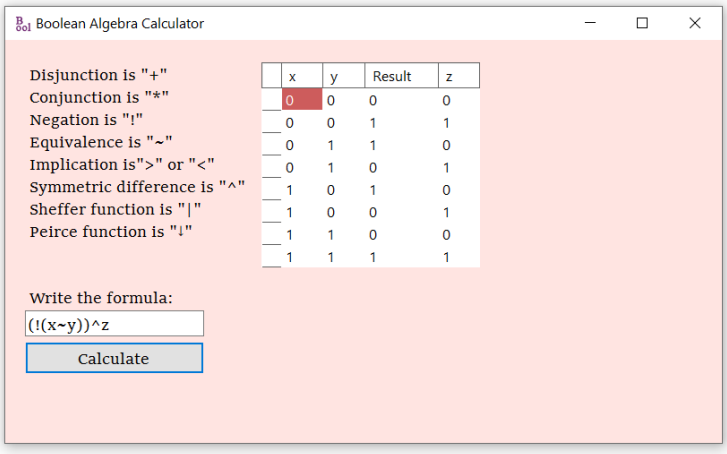

Рисунок 6 – Т/и для второго выражения, составленная: а) вручную; б) программно

В завершение тестирования аналогичным образом проверим правильность работы калькулятора на ещё одном выражении $f(x, y, z,) = (x \wedge \bar{z}) \downarrow \mathrm{z}) \rightarrow \mathrm{y}$, что на синтаксисе приложения можно записать как $f(x, y, z) = \Big(\big(x * (!\, z)\big) \downarrow z\Big) > y$. Результаты приведены на рисунке 7:

А)

x	y	z	$x \wedge \bar{z}$	$(x \wedge \bar{z}) \downarrow \mathrm{z}$	$f(x, y, z)$
0	0	0	0	1	0
0	0	1	0	1	1
0	1	0	0	0	1
0	1	1	0	0	1
1	0	0	1	0	1
1	0	1	1	0	1
1	1	0	0	0	1
1	1	1	0	0	1

Б)

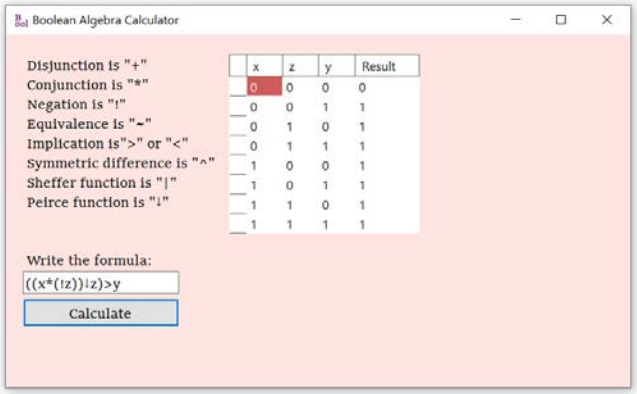

Рисунок 7 – Т/и для третьего выражения, составленная: а) вручную; б) программно

По итогам проведённой проверки можно сделать вывод о том, что приложение полностью работоспособно при соблюдении правил его синтаксиса: калькулятор верно вычисляет логические выражения и составляет для них корректные таблицы истинности.

Заключение

В настоящее время программирование является одним из наиболее востребованных и перспективных направлений развития информационных технологий. Эволюция языков и инструментов программирования проходила через различные этапы, начинаясь с машинных кодов и ассемблера и дойдя до современных высокоуровневых языков. Среди них особое место занимает язык C, который является одним из самых популярных языков программирования в мире, а также среда разработки QT-Creator, которая представляет собой мощный инструмент для разработки кроссплатформенных приложений.

Язык C используется для создания операционных систем, компиляторов, драйверов и других системных приложений, а QT-Creator – для разработки кроссплатформенных приложений с графическим пользовательским интерфейсом.

Исходя из знаний, полученных по курсу предмета «Технологии и методы программирования», для разработки приложения с графическим интерфейсом в данной курсовой работе был выбран Windows Forms – интерфейс программирования приложений, отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework. В ходе курсовой работы на основе изученных принципов было создано и протестировано приложение с графическим интерфейсом для вычисления логических операций и построения таблиц истинности. Его ключевой особенностью является преобразование данных к виду постфиксной записи, оно способно вычислять выражения булевой алгебры для различных наборов переменных и строить таблицы истинности на основе полученных данных.

Список использованных источников

1. А.А. Тюгашев. Основы программирования. Часть I. – СПб: Университет ИТМО, 2016. – 160 с.
2. Кауфман В. Ш. Языки программирования. – М.: ДМК Пресс, 2010. – 464 с.
3. Программирование на языке C++ в среде Qt Creator: / Е. Р. Алексеев, Г. Г. Злобин, Д. А. Костюк,О. В. Чеснокова, А. С. Чмыхало — М. : ALT Linux, 2015. — 448 с. : ил. — (Библиотека ALT Linux).
4. Brown E. Windows Forms Programming with C#. – Greenwich: Manning Publications C, 2002. – 720 с.
5. Шилдт Г. Полный справочник по C++, 4-е издание. Пер. с англ. — М. : Издательский дом “Вильямс”, 2006. — 800 с.
6. Яблонский С. В. Введение в дискретную математику: учебное пособие для вузов / С. В. Яблонский. – 5-е изд., стер. – Москва: Высшая школа, 2008. – 384 с.

Приложение А

Листинг 1. Описание класса-обработчика Handler

```
using System;
using System.Collections.Generic;

namespace BooleanAlgebra
{
    class Handler
    {
        // Метод возвращает true, если проверяемый символ - оператор
        static private bool IsOperator(char s) => ("+*!~><()^↓|".IndexOf(s) != -1);

        // Метод возвращает приоритет оператора
        static private byte GetPriority(char s)
        {
            return s switch
            {
                '(' => 0,
                ')' => 1,
                '!' => 2,
                '|' => 3,
                '↓' => 3,
                '*' => 3,
                '+' => 4,
                '^' => 4,
                '>' => 5,
                '<' => 5,
                '~' => 6,
                _ => 7,
            };
        }
        // Метод формирует постфиксную запись
        static private string ConvertToRPN(string input)
        {
            string output = string.Empty;
            Stack<char> stackForOperations = new Stack<char>();
            for (int i = 0; i < input.Length; i++)
            {
                if (" ".IndexOf(input[i]) != -1) //Пропуск пробела
                {
                    continue;                    //Переход к следующему символу
                }
                //Если символ - буква, то добавляем к строке хранения выражения
                if (char.IsLetter(input[i]))
                {
                    output += input[i];
                    i++;

                    if (i == input.Length)
                    {
                        break; //Если символ - последний, то выход из цикла
                    }
                }
                //Если символ - оператор
                if (IsOperator(input[i]))
                {
                    if (input[i] == '(')
                    {
                        stackForOperations.Push(input[i]); //Записываем открывающую
скобку в стек
                    }
                    else if (input[i] == ')')
```

Листинг 1. Описание класса-обработчика Handler (продолжение)

```
                    {
                        //Выписываем все операторы до открывающей скобки в строку
                        char s = stackForOperations.Pop();

                        while (s != '(')
                        {
                            output += s.ToString() + ' ';
                            s = stackForOperations.Pop();
                        }
                    }
                    else
                    {
                        if (stackForOperations.Count > 0) //Если стек не пустой
                        {
                            //Приоритет оператора не больше приоритета оператора на
вершине стека
                            if (GetPriority(input[i]) <=
GetPriority(stackForOperations.Peek()) && input[i] != '!')
                            {
                                output += stackForOperations.Pop().ToString() + " ";
//Добавляем последний оператор в строку с выражением
                            }
                        }
                        //В противном случае добавляем оператор на вершину стека
                        stackForOperations.Push(char.Parse(input[i].ToString()));
                    }
                }
            }
            //Считываем из стека все оставшиеся операторы
            while (stackForOperations.Count > 0)
            {
                output += stackForOperations.Pop() + " ";
            }
            return output; //Возврат выражения в постфиксной записи
        }
        //Вычисление выражения алгебры логики
        static private int Counting(string input, BooleanFunctions booleanFunctions)
        {
            int result = 0;
            Stack<int> temp = new Stack<int>();

            for (int i = 0; i < input.Length; i++)
            {
                if (char.IsLetter(input[i]))
                {
                    temp.Push(booleanFunctions.Variables[input[i]]);
                }
                else if (IsOperator(input[i]))
                {
                    int a = temp.Pop();
                    switch (input[i])
                    {
                        case '+':
                            {
                                int b = temp.Pop();
                                result = booleanFunctions.Disjunction(b, a);
                                break;
                            }
                        case '*':
                            {
                                int b = temp.Pop();
                                result = booleanFunctions.Conjunction(b, a);
```

Листинг 1. Описание класса-обработчика Handler (продолжение)

```
                                break;
                            }
                        case '~':
                            {
                                int b = temp.Pop();
                                result = booleanFunctions.Equivalence(b, a);
                                break;
                            }
                        case '>':
                            {
                                int b = temp.Pop();
                                result = booleanFunctions.Implication(b, a);
                                break;
                            }
                        case '<':
                            {
                                int b = temp.Pop();
                                result = booleanFunctions.Implication(a, b);
                                break;
                            }
                        case '!':
                            result = booleanFunctions.Negation(a);
                            break;
                        case '^':
                            {
                                int b = temp.Pop();
                                result = booleanFunctions.Nonequivalence(a, b);
                                break;
                            }
                        case '|':
                            {
                                int b = temp.Pop();
                                result = booleanFunctions.ShefferFunction(a, b);
                                break;
                            }
                        case '↓':
                            {
                                int b = temp.Pop();
                                result = booleanFunctions.PeirceFunction(a, b);
                                break;
                            }
                    }
                    temp.Push(result);
                }
            }
            return temp.Peek();
        }

        static public int Calculate(string input, BooleanFunctions booleanFunctions)
        {
            string output = ConvertToRPN(input);             //Перевод выражения в
постфиксный вид
            int result = Counting(output, booleanFunctions); //Вычисление выражения
в постфискной записи
            return result;                                   //Возврат результата
        }
    }
}
```

Приложение Б

Листинг 1. Описание класса BooleanFunctions

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace BooleanAlgebra
{
    class BooleanFunctions
    {
        private Dictionary<char, int> variables = new Dictionary<char, int>();

        public Dictionary<char, int> Variables
        {
            get { return variables; }
        }

        //Присвоение значений переменным
        public void SetVariables(char[] keys, int[] values)
        {
            variables.Clear();
            for (int i = 0; i < keys.Length; i++)
            {
                variables.Add(keys[i], values[i]);
            }
        }

        //Получение переменных из строки
        public static List<char> GetAllVariablesFromString(string str)
        {
            List<char> variables = new List<char>();
            for (int i = 0; i < str.Length; i++)
            {
                if (char.IsLetter(str[i]))
                {
                    variables.Add(str[i]);
                }
            }
            return variables.Distinct().ToList();
        }
        //Логичсекое ИЛИ
        public int Disjunction(int A, int B)
```

Листинг 1. Описание класса BooleanFunctions (продолжение)

```
        {
            return Math.Max(A, B);
        }
        //Логическое И
        public int Conjunction(int A, int B)
        {
            return Math.Min(A, B);
        }
        //Отрицание
        public int Negation(int A)
        {
            return (A + 1) % 2;
        }
        //Эквивалентность
        public int Equivalence(int A, int B)
        {
            return A == B ? 1 : 0;
        }
        //Импликация
        public int Implication(int A, int B)
        {
            return Disjunction(Negation(A), B);
        }
        //Неравнозначность
        public int Nonequivalence(int A, int B)
        {
            return Disjunction(Conjunction(Negation(A), B), Conjunction(A,
Negation(B)));
        }
        //Штрих Шеффера
        public int ShefferFunction(int A, int B)
        {
            return Negation(Conjunction(A, B));
        }
        //Стрелка Пирса
        public int PeirceFunction(int A, int B)
        {
            return Negation(Disjunction(A, B));
        }
    }
}
```