**СОДЕРЖАНИЕ**

[1 Анализ литературных источников 3](#_Toc134984443)

[1.1 Анализ существующих аналогов 3](#_Toc134984444)

[1.1.1 Программное средство Structurizer 3](#_Toc134984445)

[1.1.2 Программное средство Edrawmax 4](#_Toc134984446)

[1.1.3 Программное средство Smartdraw 6](#_Toc134984447)

[1.2 Анализ средств разработки программного средства 7](#_Toc134984448)

[1.2.1 Работа со стеком 7](#_Toc134984449)

[1.2.2 Работа с N-арным деревом 8](#_Toc134984450)

[1.2.3 Работа с файлами 8](#_Toc134984451)

[1.3 Спецификация функциональных требований 10](#_Toc134984452)

[2 Проектирование и разработка программного средства 14](#_Toc134984453)

[2.1 Описание алгоритмов решения задачи 14](#_Toc134984454)

[2.2 Структура данных 19](#_Toc134984455)

[2.2.1 Структура типов программы 19](#_Toc134984456)

[2.2.2 Структура данных программы 20](#_Toc134984457)

[2.2.3 Структура данных алгоритма RedefineMainBlock(Self) 20](#_Toc134984458)

[2.2.4 Структура данных алгоритма ChangeGlobalSettings(Self, AOldDefaultAction) 21](#_Toc134984459)

[2.2.5 Структура данных алгоритма TryCutDedicated(Self) 21](#_Toc134984460)

[2.2.6 Структура данных алгоритма TryCopyDedicated(Self) 21](#_Toc134984461)

[2.2.7 Структура данных алгоритма TryDeleteDedicated(Self) 22](#_Toc134984462)

[2.2.8 Структура данных алгоритма TryInsertBufferBlock(Self) 22](#_Toc134984463)

[2.2.9 Структура данных алгоритма TryAddNewStatement (Self, AStatementClass, isAfterDedicated) 22](#_Toc134984464)

[2.2.10 Структура данных алгоритма TryChangeDedicatedText(Self) 23](#_Toc134984465)

[2.2.11 Структура данных алгоритма TrySortDedicatedCase (Self) 23](#_Toc134984466)

[2.2.12 Структура данных алгоритма ChangeDedicated(Self, AStatement) 24](#_Toc134984467)

[2.2.13 Структура данных алгоритма CreateCarryBlock(Self) 24](#_Toc134984468)

[2.2.14 Структура данных алгоритма MoveCarryBlock(Self, ADeltaX, ADeltaY) 24](#_Toc134984469)

[2.2.15 Структура данных алгоритма DefineHover(Self) 25](#_Toc134984470)

[2.2.16 Структура данных алгоритма TryDrawCarryBlock(Self, AVisibleImageRect) 25](#_Toc134984471)

[2.2.17 Структура данных алгоритма TryTakeAction(Self) 25](#_Toc134984472)

[2.2.18 Структура данных алгоритма DestroyCarryBlock(Self) 26](#_Toc134984473)

[2.2.19 Структура данных алгоритма CreateStatement(AStatementClass, ABaseBlock, Res) 26](#_Toc134984474)

[2.2.20 Структура данных алгоритма TryUndo(Self) 26](#_Toc134984475)

[2.2.21 Структура данных алгоритма TryRedo(Self) 27](#_Toc134984476)

[2.2.22 Структура данных алгоритма TryDrawCarryBlock(Self, AVisibleImageRect) 27](#_Toc134984477)

[2.3 Схемы алгоритмов решения задач по ГОСТ 19.701-90 28](#_Toc134984478)

# Анализ литературных источников

## Анализ существующих аналогов

### Программное средство Structurizer

**Structurizer** – это бесплатный графический редактор для создания схем Насси-Шнейдермана. Программа была разработана компанией H.J. Schulz & Co. и предлагает ряд функций, которые делают ее привлекательной для пользователей, занимающихся программированием и проектированием.

Среди основных функций Structurizer можно выделить:

* создание блоков и условных переходов, а также циклов;
* возможность настройки цвета, шрифта и размера элементов диаграммы;
* поддержка импорта и экспорта диаграмм в различных форматах, включая PNG, GIF, JPEG и другие.
* возможность использования дополнительных символов и иконок.

Достоинств Structurizer:

* простота и удобство использования программы;
* бесплатность программы;
* возможность импорта и экспорта диаграмм в различных форматах;
* наличие дополнительных символов и иконок.

Недостатки Structurizer:

* ограниченный функционал, необходимый только для создания схем Насси-Шнейдермана;
* не всегда стабильная работа программы;
* yстаревшее ПО. Structurizer не обновляется уже много лет и не поддерживается разработчиками.

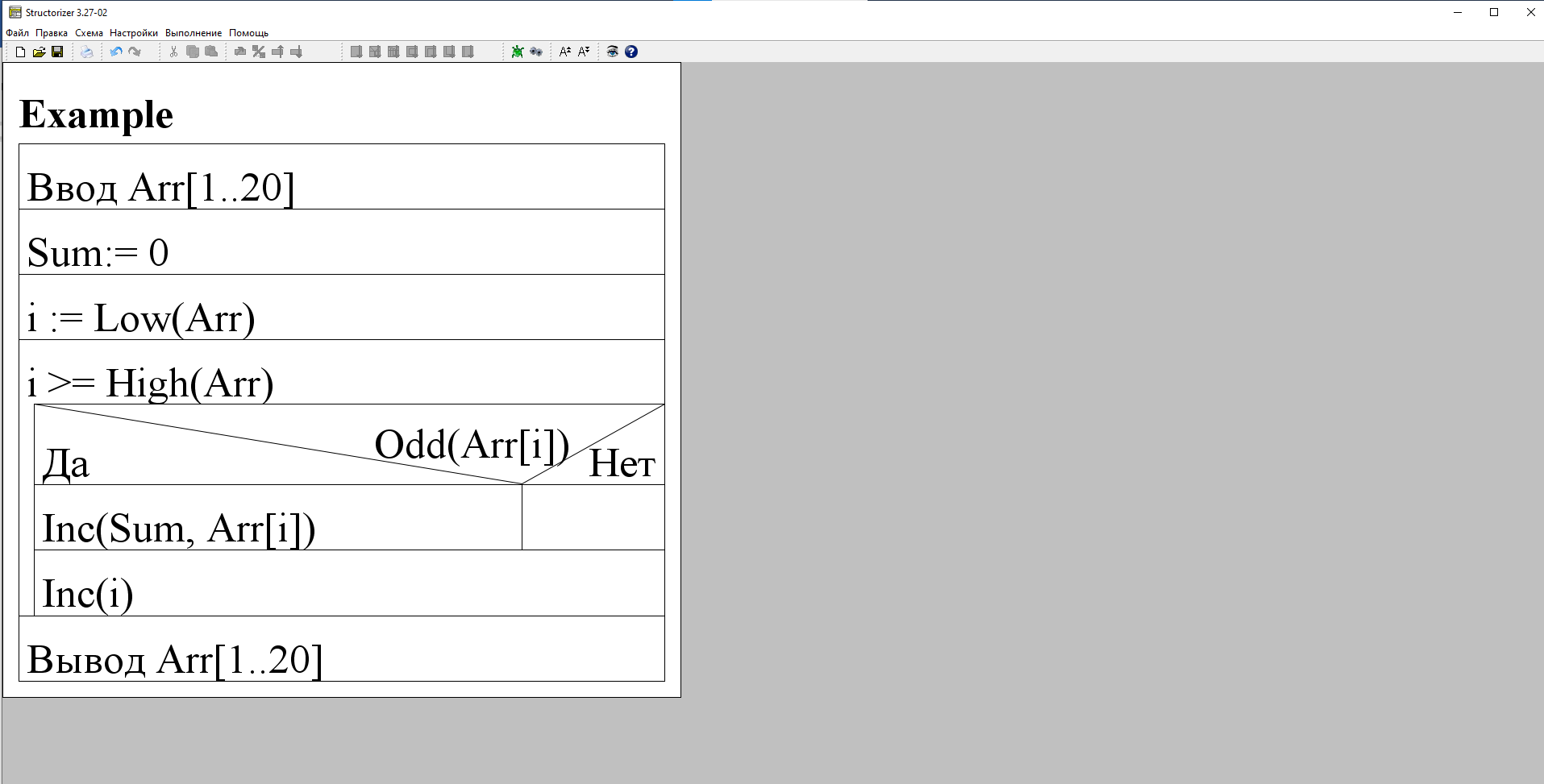


Рисунок 1.1 – Программное средство Structorizer

Structurizer – это простой и удобный инструмент для создания схем Насси-Шнейдермана. Программа имеет достаточно широкий набор функций для решения задач данного типа, и при этом является бесплатной и простой в использовании. Однако, несмотря на эти достоинства, программа иногда может работать нестабильно, что может привести к неудобствам при работе с ней.

### Программное средство Edrawmax

**Edrawmax** – это мощный графический редактор, который предоставляет пользователю возможность создавать широкий спектр диаграмм и схем, включая схемы Насси-Шнейдермана. Редактор создан компанией Edrawsoft и имеет ряд функций, которые делают его популярным среди пользователей, занимающихся программированием и проектированием.

Основные функции Edrawmax:

* создание блоков и условных переходов, а также циклов;
* поддержка различных типов соединений и рисунков;
* возможность редактирования цвета, размера, формы и других свойств элементов диаграммы;
* поддержка импорта и экспорта диаграмм в различных форматах, включая PNG, GIF, JPEG, SVG, PDF и другие;
* возможность использования дополнительных символов, шаблонов и шрифтов;
* встроенный набор готовых шаблонов и элементов для быстрого создания диаграмм.

Достоинства EdrawMax:

* мощный и многофункциональный редактор, позволяющий создавать широкий спектр диаграмм;
* большой выбор готовых шаблонов и элементов;
* возможность импорта и экспорта диаграмм в различных форматах;
* наличие дополнительных символов, шаблонов и шрифтов.

К сожалению, Edrawmax не является оптимальным инструментом для построения диаграмм Насси-Шнейдермана. Несмотря на то, что программа имеет некоторые функции, которые могут быть полезны при создании таких диаграмм, у нее есть ряд ограничений и недостатков, которые могут сделать этот процесс менее удобным и эффективным:

* неудобный интерфейс для создания диаграмм Насси-Шнейдермана. Edrawmax не имеет специализированных инструментов для создания диаграмм Насси-Шнейдермана;
* ограниченный функционал для создания диаграмм Насси-Шнейдермана. Edrawmax не имеет специализированных функций для создания диаграмм Насси-Шнейдермана.

Также стоит отметить, что это коммерческое программное обеспечение и для получения полного функционала и доступа к расширенным возможностям, пользователи должны приобрести платную версию программы.

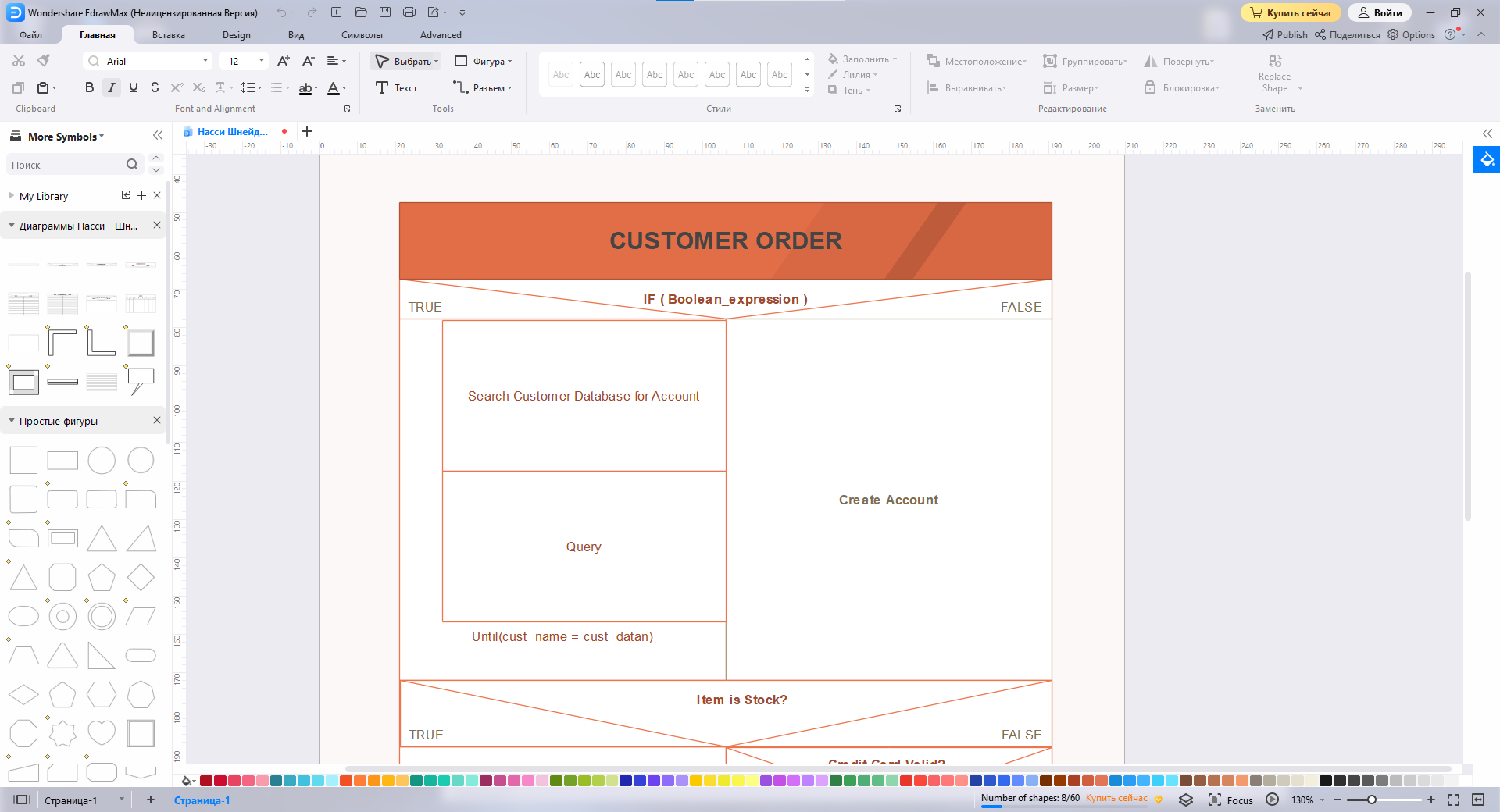


Рисунок 1.2 – Программное средство EdrawMax

В целом, хотя Edrawmax предлагает широкий набор инструментов для создания различных типов диаграмм, программа не является оптимальным выбором для создания диаграмм Насси-Шнейдермана.

### Программное средство Smartdraw

**Smartdraw** не только представлен в виде десктопного приложения, но и доступен в виде онлайн-версии, что обеспечивает более гибкую работу с программой и возможность доступа к проектам из любой точки с доступом в интернет. Кроме того, онлайн-версия Smartdraw позволяет работать в реальном времени с другими пользователями, обмениваться комментариями и совместно редактировать документы, что делает программу удобной для работы в коллективе.

Среди основных функций SmartDraw можно выделить:

* редактор предоставляет широкий набор инструментов для создания профессиональных диаграмм и схем;
* интеграция с другими приложениями, такими как Microsoft Word, Excel и PowerPoint, а также с Google Workspace, Jira и другими инструментами;
* предоставляет широкий выбор шаблонов для различных типов диаграмм и схем, что упрощает и ускоряет процесс создания;
* предоставляет возможность онлайн-совместной работы, что упрощает совместное использование диаграмм и схем с другими пользователями;
* возможность импорта и экспорта в различных форматах

Достоинства SmartDraw:

* обширный набор функций, необходимых для создания схем Насси-Шнейдермана;
* возможность настройки цвета, шрифта и размера элементов диаграммы;
* поддержка импорта и экспорта диаграмм в различных форматах;
* легкий интерфейс, понятный даже для новичков;
* возможность использования дополнительных символов и иконок;

Недостатки SmartDraw:

* платное программное обеспечение, необходимо приобретать лицензию для получения доступа к полному функционалу;
* отсутствие возможности редактирования схем в реальном времени, только локальное сохранение и загрузка диаграмм.

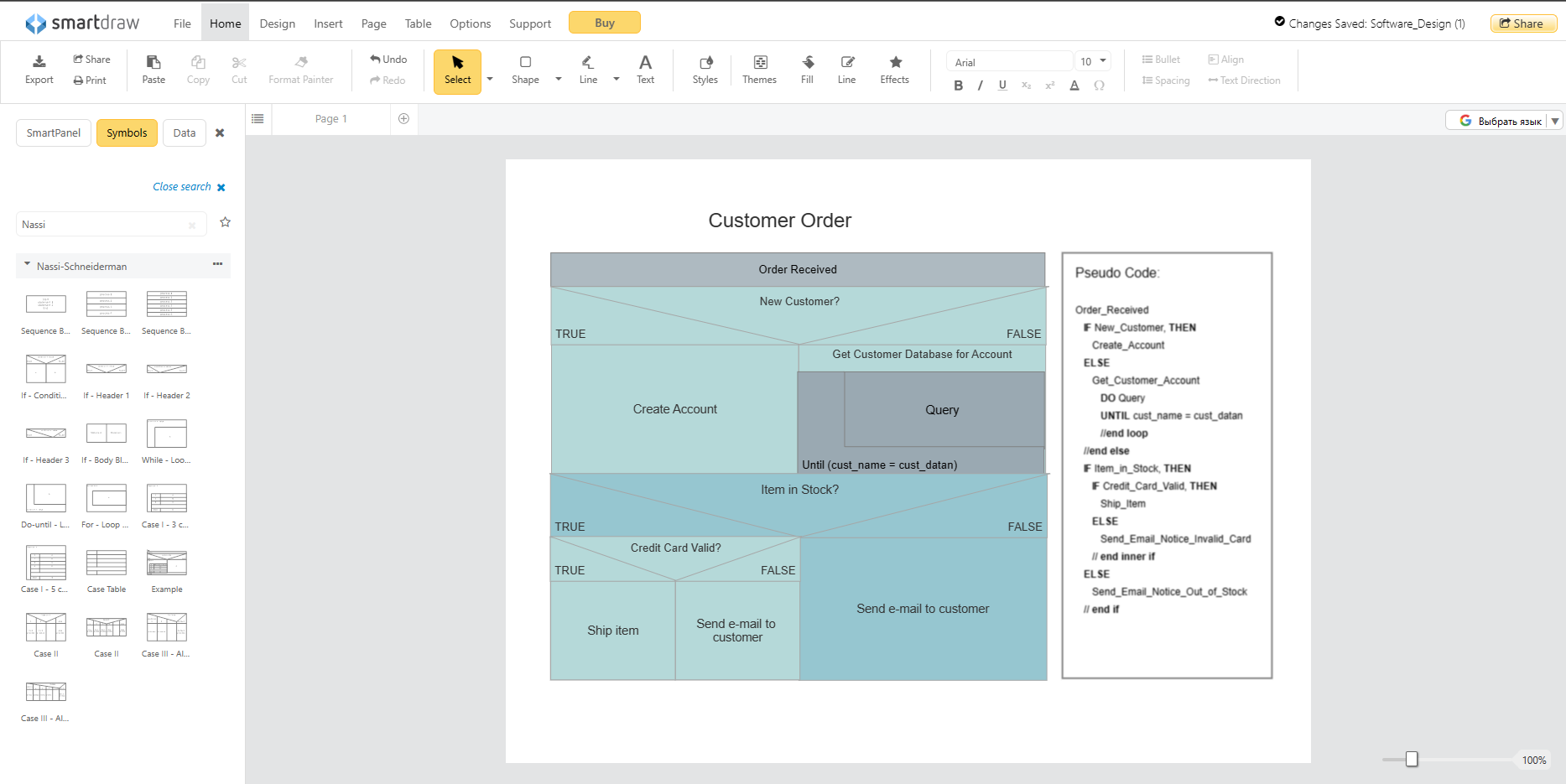


Рисунок 1.3 – Программное средство EdrawMax

Smartdraw – это многофункциональный графический редактор, который может использоваться для создания разнообразных диаграмм и схем, включая схемы Насси-Шнейдермана. Однако, программа не является специализированным инструментом для создания схем Насси-Шнейдермана, поэтому ее функциональность в этой области может быть ограничена.

## Анализ средств разработки программного средства

### Работа со стеком

Стек – это структура данных, которая работает по принципу «последним пришел, первым ушел». В стек можно добавлять элементы только на вершину, а удалять – только верхний элемент. Стек используется в программировании для реализации операции отката, когда нужно отменить последнее действие. Для этого в программе можно использовать стек для хранения истории действий пользователя. Каждое действие представляется как элемент стека, и при нажатии на кнопку «Отменить» из стека извлекается последнее действие и выполняется обратное действие, чтобы отменить его эффект. Таким образом, стек позволяет эффективно реализовывать операцию отмены действий в программе.

В программном средстве для построения схем по методу Насси-Шнейдермана используется стек для поддержки отмены (undo) и повтора (redo) действий пользователя при создании и редактировании схемы. Каждый раз, когда пользователь выполняет какое-либо действие, такое как добавление блока или изменение связей между блоками, состояние схемы сохраняется в стеке. Если пользователь захочет отменить последнее действие, программа извлечет состояние схемы из стека и вернет ее в предыдущее состояние.

Кроме того, стек используется для создания новых условий в блоке case. При создании нового условия в блоке case, программа помещает его в вершину стека. Если пользователь захочет удалить условие, программа просто извлечет его из стека.

Для реализации стека в программе используется список, который позволяет легко добавлять новые элементы и удалять уже существующие.

### Работа с N-арным деревом

N-арное дерево – это структура данных, которая представляет собой дерево, в котором каждый узел может иметь несколько дочерних узлов. Каждый узел в n-арном дереве содержит данные и ссылки на его дочерние узлы.

В отличие от двоичных деревьев, где каждый узел имеет не более двух дочерних узлов, в n-арном дереве каждый узел может иметь до n дочерних узлов. N-арное дерево может быть использовано для представления иерархических структур, таких как файловая система или структура сайта.

Каждый узел в n-арном дереве имеет родительский узел, за исключением корневого узла, который не имеет родительского узла. Узлы, у которых нет дочерних узлов, называются листьями.

Одним из преимуществ n-арного дерева является возможность эффективной вставки и удаления узлов в любом месте дерева. Однако, n-арное дерево может иметь более высокую памятьовую стоимость, чем двоичное дерево, так как каждый узел должен хранить ссылки на несколько дочерних узлов.

В программе построения схем Насси-Шнейдермана используется дерево для представления структуры схемы. Каждый узел в дереве представляет блок схемы, такой как условный оператор, цикл или оператор присваивания. Каждый узел также содержит информацию о своих дочерних узлах.

Например, условный оператор может иметь два дочерних узла: один для блока кода, который выполняется, если условие истинно, и другой для блока кода, который выполняется, если условие ложно. Цикл может иметь только один дочерний узел, который представляет тело цикла.

При построении схемы Насси-Шнейдермана программа использует дерево для отображения структуры схемы и для определения последовательности выполнения операторов. Дерево также используется для упрощения построения схемы и валидации ее структуры, что помогает избежать ошибок и упрощает процесс отладки.

Кроме того, в программе также может использоваться дерево для оптимизации процесса построения схемы, так как это позволяет ускорить поиск и доступ к определенным узлам в дереве.

### Работа с файлами

Файлы могут быть разделены на две основные категории: логические и физические файлы. Логический файл – это файл, который представляет собой логически связанные данные, имеющие определенную структуру. Он может содержать различные типы данных, такие как текстовые документы, изображения, аудио- и видеофайлы, базы данных и другие. Логический файл определяет формат, структуру и ограничения для данных, которые он содержит. Физический файл, с другой стороны, это непосредственно файл, хранящийся на жестком диске или другом устройстве хранения информации. Физический файл содержит набор битов, которые могут быть интерпретированы как данные, которые он представляет.

Логические и физические файлы тесно связаны друг с другом. Логический файл описывает формат и структуру данных, которые он содержит, а физический файл представляет собой место хранения этих данных на устройстве. Когда данные записываются в логический файл, они сохраняются в соответствующем физическом файле, который затем может быть прочитан для получения этих данных. При чтении данных из логического файла, система оперирует на физическом файле, считывая данные из определенного участка жесткого диска, и затем интерпретирует их в соответствии с форматом логического файла.

Существует три типа файлов: типизированные файлы, текстовые файлы и не типизированные файлы. Типизированные файлы связываются с файловыми переменными, объявленными как «file of <Тип>». Файл считается состоящим из элементов, каждый из которых имеет тип <Тип>. Не типизированные файлы могут быть связаны только с файловыми переменными, которые были объявлены как «file». Файл считается состоящим из элементов, размер которых определяется при открытии файла. Текстовый файл представляет собой последовательность символов, которая может быть разделена на строки. Строки могут быть различной длины (в том числе пустые). В конце каждой строки помещается специальный управляющий символ: возврат каретки (#13 или М международное обозначение CR) и перехода новую строку (#10 или международное обозначение LF). С наличием этого маркера связана логическая функция Eoln (End of line). Эта функция возвращает значение True, если текущая позиция в файле находится в конце строки (т.е. перед символом перехода на новую строку). Текстовые файлы могут быть открыты в одном из двух режимов: для чтения или для записи. Когда файл открывается для чтения, указатель позиции устанавливается на начало файла. Когда файл открывается для записи, содержимое файла удаляется, а указатель позиции устанавливается на начало файла.

Кроме того, текстовые файлы могут быть открыты в режиме добавления, который позволяет добавлять данные в конец файла без удаления его содержимого. Файл, открытый в режиме добавления, всегда открывается для записи, но указатель позиции устанавливается на конец файла.

Важно отметить, что для работы с файлами в программировании нужно уметь открывать и закрывать файлы, читать и записывать данные в файлы, а также обрабатывать ошибки, связанные с файлами, такие как отсутствие файла или ошибка доступа. Также нужно следить за использованием ресурсов компьютера при работе с файлами, чтобы избежать проблем с памятью или производительностью.

Можно сделать вывод о том, что файлы являются важными элементами программирования, которые позволяют хранить и загружать данные в различных форматах. Функции чтения и записи данных в файлы, а также функции управления файлами позволяют программистам создавать приложения, которые сохраняют пользовательские настройки и данные, что делает их более удобными и гибкими в использовании. Типизированные файлы позволяют сохранять данные в определенном формате, что упрощает работу с ними в дальнейшем. Кроме того, использование файлов позволяет пользователям сохранять свою работу и продолжать работать с ней в будущем, что является важным элементом при создании любых приложений.

Пользователи программного средства построения схем Насси-Шнейдермана имеют возможность сохранять свои работы, чтобы впоследствии открыть и продолжить работу с ними. Это позволяет сохранить текущее состояние схемы, включая настройки шрифта, параметры кисти и размещение блоков и операторов. Для этой цели используется специальный формат файла, который обеспечивает структурированное хранение данных схемы. Один из таких форматов – JSON, широко используемый для обмена данными между программами и платформами. JSON позволяет представить данные схемы в виде объектов и массивов, обеспечивая удобство чтения и восстановления информации при открытии сохраненного файла.

Сохранение статистики пользовательской активности также является важной функцией программного средства, позволяющей отслеживать и анализировать взаимодействие пользователей с программой. Это позволяет разработчикам исследовать популярные функции, обнаруживать возможные проблемы или узкие места, а также определять предпочтения пользователей и улучшать функциональность программы на основе этой информации.

В типизированный файл сохраняется статистика пользовательской активности с использованием следующих полей:

* имя пользователя (тип: String): Имя пользователя, который создал или работал с схемой;
* время входа в систему (тип: TDateTime): Дата и время, когда пользователь вошел в программу для работы с схемами;
* время выхода из системы (тип: TDateTime): Дата и время, когда пользователь завершил работу в программе и вышел из нее;
* общее время настройки системы (тип: Integer): Суммарное время, затраченное пользователем на настройку программного средства, включая изменение общих параметров и настройку пользовательского интерфейса;
* время использования справки (тип: Integer): Количество времени, которое пользователь потратил на использование справочной информации в программе;
* время настройки шрифта (тип: Integer): Время, затраченное на выбор и настройку параметров шрифта для текста на схеме;
* время настройки кисти (тип: Integer): Количество времени, затраченное на выбор и настройку параметров кисти, используемой для рисования элементов схемы;
* количество совершенных изменений (тип: Integer): Общее количество изменений, которые пользователь внес в схемы, включая добавление, удаление или изменение блоков и операторов;
* количество удаленных операторов (тип: Integer): Количество операторов, которые были удалены пользователем в процессе работы с схемами;
* количество добавленных операторов (тип: Integer): Количество операторов, которые были добавлены пользователем на схемы в процессе работы;

## Спецификация функциональных требований

Программное обеспечение обеспечивает пользователю возможность создавать и редактировать схемы Насси-Шнейдермана.

Функциональные требования – это определенные задачи и функции, которые программа должна выполнять для обеспечения желаемого функционала. Функциональные требования определяются на основе бизнес- и пользовательских требований.

Разрабатываемое программное обеспечение обеспечивает создание приложения, которое позволяет пользователю создавать схемы по методу Насси-Шнейдермана и сохранять их в нужном формате. Бизнес-требования представляют собой общее видение, не включающее детализации поведения системы и технических характеристик.

Также программное также обеспечение должно иметь интуитивно понятный и простой интерфейс, обеспечивающий удобство работы с ним. Это может быть достигнуто с помощью использования понятных иконок и кнопок, простой навигации, интуитивно понятного меню.

Функциональные требования к разрабатываемому ПС приведены в таблице 1.

Таблица 1 – Функциональные требования к программному средству

|  |  |
| --- | --- |
| Идентификатор | Требование |
| ФТ-1 | Создание схем по методу Насси-Шнейдермана из каталога блоков |
| ФТ-2 | Сохранение истории изменение схемы и отката к предыдущим версиям |
| ФТ-3 | Отображение схемы |
| ФТ-4 | Создание и загрузка файлов схем |
| ФТ-5 | Экспорт в различные форматы |
| ФТ-6 | Сохранение и отображение статистики пользователя |
| ФТ-7 | Сохранения и загрузки настроек пользовательского интерфейса |
| ФТ-8 | Просмотра информации оператора |
| ФТ-9 | Редактирование информации оператора |
| ФТ-10 | Изменения внешнего вида схемы |

ФТ-1 Создание схем по методу Насси-Шнейдермана из каталога блоков.

Это функциональное требование означает, что пользователь должен иметь возможность создавать схемы по методу Насси-Шнейдермана, используя доступные блоки из каталога. Каталог должен содержать набор стандартных блоков. Пользователь должен иметь возможность выбирать нужные блоки и вставлять их на рабочую область, чтобы создавать схемы.

ФТ-2 Сохранение истории изменение схемы и отката к предыдущим версиям.

Это функциональное требование означает, что пользователь должен иметь возможность сохранять все изменения, внесенные в схему, и возвращаться к предыдущим версиям схемы при необходимости. Это важно, чтобы предотвратить потерю данных и иметь возможность вернуться к предыдущему рабочему состоянию. Для этого можно использовать стек или другие механизмы хранения истории изменений.

ФТ-3 Отображение схемы.

Это функциональное требование означает, что пользователь должен иметь возможность просмотреть созданную им схему. Схема должна быть отображена в удобном для восприятия формате, который позволяет пользователю понимать структуру и последовательность выполнения операций в схеме. Можно использовать графический интерфейс или другой способ отображения схемы.

ФТ-4 Создание и загрузка файлов схем.

Это функциональное требование означает, что пользователь должен иметь возможность сохранять схемы в файлы и загружать их из файлов. Это важно, чтобы пользователь мог сохранить свою работу и поделиться ею с другими пользователями.

ФТ-5 Экспорт в различные форматы.

Это функциональное требование позволяет пользователю экспортировать созданную схему в различные форматы, такие как PNG, JPEG, PDF и другие. Экспортирование схемы в различные форматы позволяет пользователю сохранить ее в удобном для просмотра формате и поделиться с другими людьми.

ФТ-6 Сохранение и отображение статистики пользователя.

Это функциональное требование позволяет сохранять информацию о действиях и активности пользователя для последующего отображения. Статистика включает такие данные как имя пользователя, время входа и выхода из системы, время настройки системы, использование справки, настройку шрифта и кисти, количество совершенных изменений, удаленных и добавленных операторов. Это позволяет пользователям отслеживать свою активность и проделанную работу.

ФТ-7 Сохранения и загрузки настроек пользовательского интерфейса.

Это функциональное требование позволяет пользователю сохранять свои настройки пользовательского интерфейса, такие как цвет заднего фона, настройки кисти и другие. После сохранения пользователь может загрузить эти настройки и сразу начать работу со схемой в своей привычной среде.

ФТ-8 Просмотр информации о блоке.

Это функциональное требование позволяет пользователю просматривать информацию о выбранном блоке. Такую как наименование действия и условия, если имеются.

ФТ-9 Редактирование информации о блоке.

Это функциональное требование позволяет пользователю редактировать информацию о выбранном блоке. Такую как наименование действия и условия, если имеются.

ФТ-10 Изменения внешнего вида блоков.

Это функциональное требование позволяет пользователю изменять внешний вид блоков, такой как цвет, шрифт, размер и т.д. Это может быть полезно, если пользователю нужно выделить определенные блоки на схеме или изменить их внешний вид для удобства восприятия.

# Проектирование и разработка программного средства

## Описание алгоритмов решения задачи

Таблица 2 – Описание алгоритмов решения задачи

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| №  п.п. | Наименование алгоритма | Название алгоритма | Формальные  параметры | Предпола-гаемый тип реализации |
| 1 | Основной алгоритм | Вызывает следующие подпрограммы:  RedefineMainBlock, ChangeGlobalSettings, TryCut-  Dedicated, TryCopyDedicated,TryDeleteDedicated, TryInsertBuerBlock, TryMoveDedicated, TryChangeDedicatedText, TryAdd-  NewStatement, TrySortDedicatedCase, CreateCarryBlock, MoveCarryBlock, DefineHover, TryDrawCarryBlock, TryTakeAction, DestroyCarryBlock, CreateStatement, TryUndo, TryRedo, Draw. |  |  |
| 2 | RedefineMainBlock  (  Self  ) | Производит переопределение размера схемы для Self | Self – получает от фактического параметра адрес с защитой | Процедура |
| 3 | ChangeGlobalSettings  (  Self,  AOldDefaultAction  ) | Изменяет глобальные настройки для объекта Self. В случае, если новое действие по умолчанию | Self – получает от фактического параметра адрес с защитой.  AOldDefaultAction – получает от | Процедура |

Продолжение таблицы 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | не равно значению параметра AOldDefaultAction, происходит вызов подпрограммы RedefineMainBlock с передачей объекта Self в качестве параметра | фактического параметра адрес с защитой |  |
| 4 | TryCutDedicated  (  Self  ) | Вызывает подпрограммы TryCopy- Dedicated и TryDelete-  Dedicated, передавая Self в качестве параметра. | Self – получает от фактического параметра адрес с защитой | Процедура |
| 5 | TryDeleteDedicated  (  Self  ) | Производит удаление выделенного оператора у объекта Self, если такой оператор существует. | Self – получает от фактического параметра адрес с защитой | Процедура |
| 6 | TryCopyDedicated  (  Self  ) | Копирует выделенный оператор, находящийся в объекте Self, если такой оператор существует. | Self – получает от фактического параметра адрес с защитой | Процедура |
| 7 | TryInsertBufferBlock  (  Self  ) | Добавляет в переменную Self новый буферный оператор после выделенного оператора, если таковой имеется. | Self – получает от фактического параметра адрес с защитой | Процедура |
| 8 | TryMoveDedicated  (  Self,  ASetScrollPosProc,  AKey  ) | Перемещает выделенный оператор в переменной Self в соответствии со значением переменной AKey. Если перемещение выполнено | Self – получает от фактического параметра адрес с защитой.  ASetScrollPosProc – получает от фактического | Процедура |

Продолжение таблицы 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | успешно, то вызывает подпрограмму ASetScrollPosProc. | параметра адрес с защитой.  AKey – получает от фактического параметра адрес с защитой |  |
| 9 | TryAddNewStatement  (  Self,  AStatementClass,  isAfterDedicated  ) | Вызывает подпрограмму CreateStatement с параметрами AStatementClass и базовым блоком выделенного оператора переменной Self для создания нового оператора. Он добавляется после или до выделенного оператора в зависимости от значения переменной isAfterDedicated | Self – получает от фактического параметра адрес с защитой.  AStatementClass – получает от фактического параметра адрес с защитой.  isAfterDedicated  – получает от фактического параметра адрес с защитой | Процедура |
| 10 | TryChangeDedicatedText  (  Self  ) | Обновляет действие у выделенного оператора переменной Self, если он существует | Self – получает от фактического параметра адрес с защитой | Процедура |
| 11 | TrySortDedicatedCase  (  Self  ) | Если выделенный оператор переменной Self является оператором множественного выбора, то сортирует его условия | Self – получает от фактического параметра адрес с защитой | Процедура |
| 12 | ChangeDedicated  (  Self,  AStatement  ) | Меняет значение переменной Self на AStatement для выделенного оператора | Self – получает от фактического параметра адрес с защитой | Процедура |

Продолжение таблицы 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 13 | CreateCarryBlock  ( Self ) | Создает у переменной Self переносимый блок. | Self – получает от фактического параметра адрес с защитой | Процедура |
| 14 | MoveCarryBlock  (  Self,  ADeltaX,  ADeltaY  ) | Смещает переносимый блок переменной Self на ADeltaX по оси X и на ADeltaY по оси Y | Self – получает от фактического параметра адрес с защитой.  ADeltaX – получает от фактического параметра адрес с защитой.  ADeltaY– получает от фактического параметра адрес с защитой | Процедура |
| 15 | DefineHover  (  Self,  AX,  AY  ) | Поиск оператора, содержащего координаты (AX, AY) и, если оператор найден, присваивает его в наведенный оператор переменной Self и определяет для него действие | Self – получает от фактического параметра адрес с защитой.  AX – получает от фактического параметра адрес с защитой.  AY – получает от фактического параметра адрес с защитой | Процедура |
| 16 | TryDrawCarryBlock  (  Self,  AVisibleImageRect  ) | Отрисовывает все операторы переносимого блока у переменной Self, которые входят в границы AVisibleImageRect | Self – получает от фактического параметра адрес с защитой.  AVisibleImageRect – получает от фактического параметра адрес с защитой. | Процедура |
| 17 | TryTakeAction  (  Self  ) | Выполняет действие с выделенным блоком переменной Self в зависимости от действия наведенного | Self – получает от фактического параметра адрес с защитой. | Процедура |

Продолжение таблицы 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | оператора переменной Self. |  |  |
| 18 | DestroyCarryBlock  (  Self  ) | Удаляет переносимый блок переменной Self | Self – получает от фактического параметра адрес с защитой. | Процедура |
| 19 | CreateStatement  (  AStatementClass,  ABaseBlock,  Res  ) | Создает оператор типа AStatementClass с базовым блоком ABaseBlock и записывает его в переменную Res | AStatementClass – получает от фактического параметра адрес с защитой.  ABaseBlock – получает от фактического параметра адрес с защитой.  Res – получает от фактического параметра адрес, возвращаемый параметр | Функций. Res – возвращаемый функцией параметр |
| 20 | TryUndo  (  Self  ) | Отменяет последнее действие, выполненное над переменной Self | Self – получает от фактического параметра адрес с защитой. | Процедура |
| 21 | TryRedo  (  Self  ) | Выполняет отмену последнего ранее отмененного действия, связанного с переменной Self. | Self – получает от фактического параметра адрес с защитой. | Процедура |
| 22 | Draw  (  Self,  AVisibleImageRect  ) | Функция вызывает подпрограмму TryDrawCarryBlock, передавая в параметрах переменную Self и AVisibleImageRect. Эта подпрограмма отрисовывает главный блок и выделенный оператор. | Self – получает от фактического параметра адрес с защитой.  AVisibleImageRect – получает от фактического параметра адрес с защитой. | Процедура |

## Структура данных

### Структура типов программы

Таблица 3 – Структура типов программы

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| TBlock | Record  FXStart, FXLast:  Integer;  FCanvas: TCanvas;  FStatements: array of  ^Statement  FBaseOperator:  ^TOperator  End; | Блок, в котором содержатся операторы, хранит адрес на базовый оператор. Задает ограничение для вложенный операторов по Х |
| PBlock | ^TBlock | Тип, предназначенный для обозначения указателя на блок |
| TStatement | Record  FYStart, FYLast: Integer;  FAction: String;  BaseBlock: ^TBlock End; | Оператор, который хранит действие и задает коодинаты по Y. Хранит адрес на базовый блок |
| PStatement | ^TStatement | Предназначен для обозначения указателя на блок |
| TOperator | Record(PStatement)  FBlocks: array of  ^TBlock  End | Оператор, который содержит в себе блоки |
| POperator | ^TOperator | Предназначен для обозначения указателя на блок |
| THovered-  Statement | Record  Statement: TStatement;  Rect: TRect;  State: TState;  End; | Определяет наведенный оператор и его состояние |
| TSetScrollPosProc | Procedure(const AStatement: TStatement) of object | Процедурный тип, который управляет положением скролла в заданной позиции |
| PItem | ^TItem | Указатель на элемент |

Продолжение таблицы 3

|  |  |  |
| --- | --- | --- |
| TItem | Record  FData: T;  FNext: PItem;  End; | Элемент, содержащий данные и указатель на следующий элемент |
| TStack | Record  FTop: PItem;  FCount: Integer;  End | Представляет стек, хранящий указатель на первый элемент и количество элементов в стеке |
| TBlockManager | Record  FBufferBlock: TBlock;  FCarryBlock: TBlock;  FHoveredStatement:  THoveredStatement;  FMainBlock : TBlock;  FDedicatedStatement:  TStatement;  FPaintBox: TPaintBox;  FPen: TPen;  FFont: TFont;  End; | Этот объект предназначен для работы с операторами и блоками внутри схемы |

### Структура данных программы

Таблица 4 – Структура данных программы

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| FBlockManager | TBlockManager | Работа с операторами и блоками внутри схемы |
| FUndoStack | TStack | Стек для отмены действий |
| FRedoStac | TStack | Стек для отмен последних отмененных действий |

### Структура данных алгоритма RedefineMainBlock(Self)

Таблица 5 – Структура данных RedefineMainBlock(Self)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Self | TBlockManager | Обеспечивает доступ к информации о блоках и операторах | Формальный |

### Структура данных алгоритма ChangeGlobalSettings(Self, AOldDefaultAction)

Таблица 6 – Структура данных ChangeGlobalSettings(Self, AOldDefaultAction)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Self | TBlockManager | Обеспечивает доступ к информации о блоках и операторах | Формальный |
| AOldDefaultAction | String | Предыдущее значение действия по умолчанию | Формальный |

### Структура данных алгоритма TryCutDedicated(Self)

Таблица 7 – Структура данных TryCutDedicated(Self)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Self | TBlockManager | Обеспечивает доступ к информации о блоках и операторах | Формальный |

### Структура данных алгоритма TryCopyDedicated(Self)

Таблица 8 – Структура данных TryCopyDedicated(Self)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Self | TBlockManager | Обеспечивает доступ к информации о блоках и операторах | Формальный |

### Структура данных алгоритма TryDeleteDedicated(Self)

Таблица 9 – Структура данных TryDeleteDedicated(Self)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Self | TBlockManager | Обеспечивает доступ к информации о блоках и операторах | Формальный |

### Структура данных алгоритма TryInsertBufferBlock(Self)

Таблица 10 – Структура данных TryInsertBufferBlock(Self)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Self | TBlockManager | Обеспечивает доступ к информации о блоках и операторах | Формальный |
| BaseBlock | TBlock | Получение базового блока выделенного оператора | Локальный |
| I | Integer | Счетчик цикла | Локальный |

### Структура данных алгоритма TryAddNewStatement (Self, AStatementClass, isAfterDedicated)

Таблица 11 – Структура данных TryAddNewStatement (Self, AStatementClass, isAfterDedicated)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Self | TBlockManager | Обеспечивает доступ к информации о блоках и операторах | Формальный |
| AStatementClass | Integer | Переменная указывает тип оператора, который будет создан | Формальный |
| isAfterDedicated | Boolean | Флаг, указывающий, добавлять ли новый оператор | Формальный |

Продолжение таблицы 11

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | после выделенного оператора |  |
| NewStatement | TStatement | Хранит созданный оператор | Локальный |
| Block | TBlock | Хранит базовый блок созданного оператора | Локальный |

### Структура данных алгоритма TryChangeDedicatedText(Self)

Таблица 12 – Структура данных TryChangeDedicatedText(Self)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Self | TBlockManager | Обеспечивает доступ к информации о блоках и операторах | Формальный |
| Action | String | Хранит новое действие для выделенного оператора | Локальный |

### Структура данных алгоритма TrySortDedicatedCase (Self)

Таблица 13 – Структура данных TrySortDedicatedCase (Self)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Self | TBlockManager | Обеспечивает доступ к информации о блоках и операторах | Формальный |

### Структура данных алгоритма ChangeDedicated(Self, AStatement)

Таблица 14 – Структура данных ChangeDedicated(Self, AStatement)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Self | TBlockManager | Обеспечивает доступ к информации о блоках и операторах | Формальный |
| AStatement | TStatement | Хранит новый выделенный блок | Формальный |

### Структура данных алгоритма CreateCarryBlock(Self)

Таблица 15 – Структура данных CreateCarryBlock(Self)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Self | TBlockManager | Обеспечивает доступ к информации о блоках и операторах | Формальный |

### Структура данных алгоритма MoveCarryBlock(Self, ADeltaX, ADeltaY)

Таблица 16 – Структура данных MoveCarryBlock(Self, ADeltaX, ADeltaY)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Self | TBlockManager | Обеспечивает доступ к информации о блоках и операторах | Формальный |
| ADeltaX | Integer | Смещение по координате X | Формальный |
| ADeltaY | Integer | Смещение по координате Y | Формальный |

### Структура данных алгоритма DefineHover(Self)

Таблица 17 – Структура данных DefineHover(Self)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Self | TBlockManager | Обеспечивает доступ к информации о блоках и операторах | Формальный |
| Indent | Integer | Отступ от начала оператора | Локальный |

### Структура данных алгоритма TryDrawCarryBlock(Self, AVisibleImageRect)

Таблица 18 – Структура данных TryDrawCarryBlock(Self, AVisibleImageRect)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Self | TBlockManager | Обеспечивает доступ к информации о блоках и операторах | Формальный |
| AVisibleImageRect | TVisibleImageRect | Информация о видимой границе | Формальный |

### Структура данных алгоритма TryTakeAction(Self)

Таблица 19 – Структура данных TryTakeAction(Self)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Self | TBlockManager | Обеспечивает доступ к информации о блоках и операторах | Формальный |

### Структура данных алгоритма DestroyCarryBlock(Self)

Таблица 20 – Структура данных DestroyCarryBlock(Self)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Self | TBlockManager | Обеспечивает доступ к информации о блоках и операторах | Формальный |

### Структура данных алгоритма CreateStatement(AStatementClass, ABaseBlock, Res)

Таблица 21 – Структура данных CreateStatement(AStatementClass, ABaseBlock, Res)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| AStatementClass | Integer | Переменная указывает тип оператора, который будет создан | Формальный |
| ABaseBlock | TBlock | Базовый блок, в котором будет создан оператор | Формальный |
| Res | TStatement | Созданный оператор | Формальный |
| Action | String | Хранит новое действие для нового оператора | Локальный |

### Структура данных алгоритма TryUndo(Self)

Таблица 22 – Структура данных TryUndo(Self)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Self | TBlockManager | Обеспечивает доступ к информации о блоках и операторах | Формальный |

### Структура данных алгоритма TryRedo(Self)

Таблица 23 – Структура данных TryRedo(Self)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Self | TBlockManager | Обеспечивает доступ к информации о блоках и операторах | Формальный |

### Структура данных алгоритма TryDrawCarryBlock(Self, AVisibleImageRect)

Таблица 24 – Структура данных TryDrawCarryBlock(Self, AVisibleImageRect)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Self | TBlockManager | Обеспечивает доступ к информации о блоках и операторах | Формальный |
| AVisibleImageRect | TVisibleImageRect | Информация о видимой границе | Формальный |
| Stock | Integer | Определяет дополнительное значение, на которое нужно изменить текущую границу | Локальный |
| Correction | Integer | Величина корректировки в пикселях, необходимая для правильного отображения границы | Локальный |

## Схемы алгоритмов решения задач по ГОСТ 19.701-90

### Схема алгоритма ChangeGlobalSettings



Рисунок 2.1 – Схема алгоритма ChangeGlobalSettings

### Схема алгоритма TryAddNewStatement



Рисунок 2.2 – Схема алгоритма TryAddNewStatement

### Схема алгоритма ChangeDedicated



Рисунок 2.3 – Схема алгоритма ChangeDedicated

### Схема алгоритма TrySortDedicatedCase



Рисунок 2.4 – Схема алгоритма TrySortDedicatedCase

### Схема алгоритма MoveCarryBlock



Рисунок 2.5 – Схема алгоритма MoveCarryBlock

### Схема алгоритма CreateCarryBlock



Рисунок 2.6 – Схема алгоритма CreateCarryBlock

### Схема алгоритма TryTakeAction



Рисунок 2.7 – Схема алгоритма TryTakeAction

### Схема алгоритма DestroyCarryBlock



Рисунок 2.8 – Схема алгоритма DestroyCarryBlock

### Схема алгоритма CreateStatement



Рисунок 2.9 – Схема алгоритма CreateStatement

### Схема алгоритма Draw



Рисунок 2.10 – Схема алгоритма CreateStatement (часть 1)



Рисунок 2.11 – Схема алгоритма CreateStatement (часть 2)

## Графический интерфейс

Для организации графического интерфейса программного средства было использовано 6 формы: frmMain, frmGetAction, frmGetCaseConditions, frmGlobalSettings, frmPenSettings, frmHelp.

### Описание графических компонентов формы frmMain

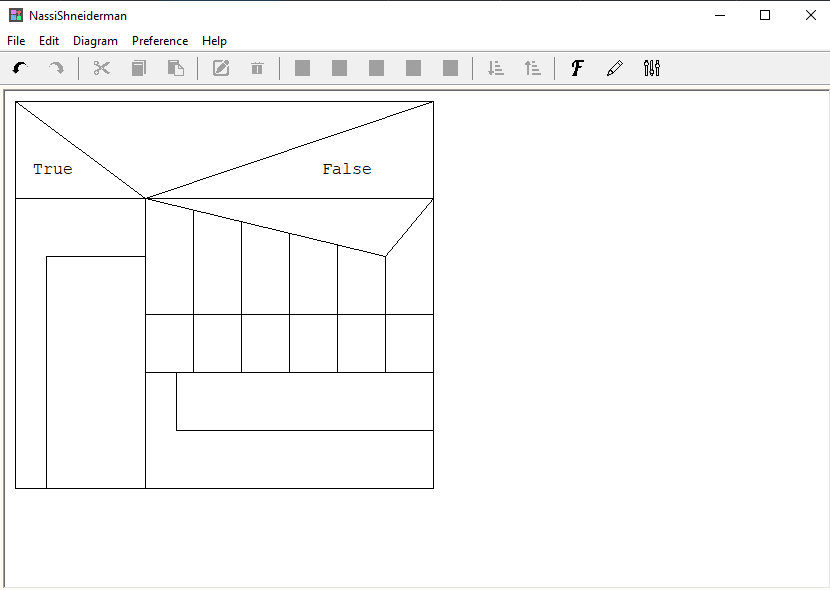


Рисунок 2.12 – Основное окно программы