Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Основы алгоритмизации и программирования (ОАиП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

Программное средство «Графический редактор Насси-Шнейдермана»

БГУИР КП I–40 01 01 322 ПЗ

Выполнил

студент гр. 251003 Панкратьев Е.С.

Проверил: Фадеева Е.П.

Минск 2023

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Лапицкая Н.В. 2023г.

ЗАДАНИЕ

по курсовому проектированию

Студенту Панкратьеву Егору Сергеевичу\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Тема работы Программное средство «Графический редактор Насси-\_\_\_

Шнейдермана»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Срок сдачи законченной работы *03.06.2023г.*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Исходные данные к работе *Среда программирования Delphi. Возможность создания графических схем по методу Насси-Шнейдермана из набора доступных блоков. Возможность сохранения истории изменений схемы и отката к предыдущим версиям. Отображение созданной схемы на экране. Автоматического выравнивания блоков на схеме. Создание и загрузка файлов схем для дальнейшей работы. Возможность экспорта схемы в различные форматы. Сохранение и отображение статистики пользовательской активности. Сохранение и загрузка настроек пользовательского интерфейса. Просмотр информации о каждом блоке на схеме. Редактирование информации оператора. Изменение внешнего вида схемы. Возможность просмотра краткого описания функциональных возможностей программы, а также краткие сведения об основных возможностях редактора, его инструментах и функций.* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

*Введение*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*1 Анализ литературных источников и формирование функциональных требований к разрабатываемому программному средству\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*2 Проектирование и разработка программного средства* \_\_\_\_\_\_*\_\_\_\_\_\_\_\_\_\_\_*

*3 Тестирование и проверка работоспособности программного средства\_\_\_*

*4 Руководство по установке и использованию программного средства\_\_\_\_\_*

*Заключение \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Список использованных источников\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Приложения* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

*Схема алгоритма в формате А1*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Консультант по курсовой работе *Фадеева Е.П.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

7.Дата выдачи задания *16.02.2023г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего обьема работы):

*Раздел 1, Введение к 28.02.2023г. – 10 % готовности работы;\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 2 к 15.03.2023г. – 30% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 3 к 15.04.2023г. – 60% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 4, Заключение,Приложения к 20.05.2023г. – 90% готовности работы;*

*оформление пояснительной записки и графического материала к 31.05.2023г. – 100% готовности работы.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Защита курсового проекта с 01.06.2023г. по 03.06.2023г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

РУКОВОДИТЕЛЬ *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Фадеева Е.П.*

*(подпись)*

Задание принял к исполнению *\_\_\_\_\_\_\_\_\_\_\_\_\_Панкратьев Е.С. 16.02.2023г.*

*(дата и подпись студента)*

**СОДЕРЖАНИЕ**

[Введение 7](#_Toc135980221)

[1 Анализ литературных источников и формирование функциональных требований к разрабатываемому программному средство 8](#_Toc135980222)

[1.1 Анализ существующих аналогов 8](#_Toc135980223)

[1.1.1 Программное средство Structurizer 8](#_Toc135980224)

[1.1.2 Программное средство Edrawmax 9](#_Toc135980225)

[1.1.3 Программное средство Smartdraw 10](#_Toc135980226)

[1.2 Описание средств разработки 12](#_Toc135980227)

[1.2.1 Работа со стеком 12](#_Toc135980228)

[1.2.2 Работа с N-арным деревом 13](#_Toc135980229)

[1.2.3 Работа с файлами 14](#_Toc135980230)

[1.3 Спецификация функциональных требований 16](#_Toc135980231)

[2 Проектирование и разработка программного средства 19](#_Toc135980232)

[2.1 Описание алгоритмов решения задачи 19](#_Toc135980233)

[2.2 Структура данных 24](#_Toc135980234)

[2.2.1 Структура типов программы 24](#_Toc135980235)

[2.2.2 Структура данных программы 25](#_Toc135980236)

[2.2.3 Структура данных алгоритма RedefineMainBlock(Self) 26](#_Toc135980237)

[2.2.4 Структура данных алгоритма ChangeGlobalSettings(Self, AOldDefaultAction) 26](#_Toc135980238)

[2.2.5 Структура данных алгоритма TryCutDedicated(Self) 26](#_Toc135980239)

[2.2.6 Структура данных алгоритма TryCopyDedicated(Self) 27](#_Toc135980240)

[2.2.7 Структура данных алгоритма TryDeleteDedicated(Self) 27](#_Toc135980241)

[2.2.8 Структура данных алгоритма TryInsertBufferBlock(Self) 27](#_Toc135980242)

[2.2.9 Структура данных алгоритма TryAddNewStatement (Self, AStatementClass, isAfterDedicated) 28](#_Toc135980243)

[2.2.10 Структура данных алгоритма TryChangeDedicatedText(Self) 28](#_Toc135980244)

[2.2.11 Структура данных алгоритма TrySortDedicatedCase (Self) 29](#_Toc135980245)

[2.2.12 Структура данных алгоритма ChangeDedicated(Self, AStatement) 29](#_Toc135980246)

[2.2.13 Структура данных алгоритма CreateCarryBlock(Self) 29](#_Toc135980247)

[2.2.14 Структура данных алгоритма MoveCarryBlock(Self, ADeltaX, ADeltaY) 30](#_Toc135980248)

[2.2.15 Структура данных алгоритма DefineHover(Self) 30](#_Toc135980249)

[2.2.16 Структура данных алгоритма TryDrawCarryBlock(Self, AVisibleImageRect) 30](#_Toc135980250)

[2.2.17 Структура данных алгоритма TryTakeAction(Self) 31](#_Toc135980251)

[2.2.18 Структура данных алгоритма DestroyCarryBlock(Self) 31](#_Toc135980252)

[2.2.19 Структура данных алгоритма CreateStatement(AStatementClass, ABaseBlock, Res) 31](#_Toc135980253)

[2.2.20 Структура данных алгоритма TryUndo(Self) 32](#_Toc135980254)

[2.2.21 Структура данных алгоритма TryRedo(Self) 32](#_Toc135980255)

[2.2.22 Структура данных алгоритма TryDrawCarryBlock(Self, AVisibleImageRect) 32](#_Toc135980256)

[2.3 Схемы алгоритмов решения задач по ГОСТ 19.701-90 33](#_Toc135980257)

[2.3.1 Схема алгоритма ChangeGlobalSettings 33](#_Toc135980258)

[2.3.2 Схема алгоритма TryAddNewStatement 34](#_Toc135980259)

[2.3.3 Схема алгоритма ChangeDedicated 35](#_Toc135980260)

[2.3.4 Схема алгоритма TrySortDedicatedCase 36](#_Toc135980261)

[2.3.5 Схема алгоритма MoveCarryBlock 37](#_Toc135980262)

[2.3.6 Схема алгоритма CreateCarryBlock 38](#_Toc135980263)

[2.3.7 Схема алгоритма TryTakeAction 39](#_Toc135980264)

[2.3.8 Схема алгоритма DestroyCarryBlock 40](#_Toc135980265)

[2.3.9 Схема алгоритма CreateStatement 41](#_Toc135980266)

[2.3.10 Схема алгоритма Draw 42](#_Toc135980267)

[2.4 Графический интерфейс 44](#_Toc135980268)

[2.4.1 Описание графических компонентов формы frmMain 44](#_Toc135980269)

[2.4.2 Описание графических компонентов формы frmGetAction 47](#_Toc135980270)

[2.4.3 Описание графических компонентов формы frmGetCaseConditions 47](#_Toc135980271)

[2.4.4 Описание графических компонентов формы frmGlobalSettings 48](#_Toc135980272)

[2.4.5 Описание графических компонентов формы frmPenSettings 49](#_Toc135980273)

[2.4.6 Описание графических компонентов формы frmHelp 50](#_Toc135980274)

[3 Тестирование и проверка работоспособности программного средства 51](#_Toc135980275)

[3.1 Тестирование основной формы 51](#_Toc135980276)

[3.1.1 Тест 1 51](#_Toc135980277)

[3.1.2 Тест 2 51](#_Toc135980278)

[3.1.3 Тест 3 52](#_Toc135980279)

[3.1.4 Тест 4 52](#_Toc135980280)

[3.1.5 Тест 5 53](#_Toc135980281)

[3.1.6 Тест 6 53](#_Toc135980282)

[3.1.7 Тест 7 54](#_Toc135980283)

[3.1.8 Тест 8 54](#_Toc135980284)

[3.1.9 Тест 9 55](#_Toc135980285)

[3.2 Тестирование формы frmGetAction 55](#_Toc135980286)

[3.2.1 Тест 1 55](#_Toc135980287)

[3.2.2 Тест 2 56](#_Toc135980288)

[3.3 Тестирование формы frmGetCaseConditions 56](#_Toc135980289)

[3.3.1 Тест 1 56](#_Toc135980290)

[3.3.2 Тест 2 57](#_Toc135980291)

[3.4 Тестирование формы frmGlobalSettings 57](#_Toc135980292)

[3.4.1 Тест 1 57](#_Toc135980293)

[3.4.2 Тест 2 58](#_Toc135980294)

[3.5 Тестирование формы frmPenSettings 58](#_Toc135980295)

[3.5.1 Тест 1 58](#_Toc135980296)

[3.5.2 Тест 2 59](#_Toc135980297)

[3.6 Тестирование формы frmHelp 60](#_Toc135980298)

[3.6.1 Тест 1 60](#_Toc135980299)

[3.6.2 Тест 2 60](#_Toc135980300)

[4 Руководство по установке 62](#_Toc135980301)

[4.1 Минимальные системные требования 62](#_Toc135980302)

[4.2 Установка 62](#_Toc135980303)

[4.3 Работа с приложением 67](#_Toc135980304)

[Заключение 71](#_Toc135980305)

[Список использованных источников 72](#_Toc135980306)

[Приложение А 73](#_Toc135980307)

[Приложение Б 99](#_Toc135980308)

[Приложение В 101](#_Toc135980309)

[Приложение Г 110](#_Toc135980310)

[Приложение Д 114](#_Toc135980311)

[Приложение Е 117](#_Toc135980312)

[Приложение Ж 120](#_Toc135980313)

[Приложение З 137](#_Toc135980314)

[Приложение И 140](#_Toc135980315)

[Приложение К 142](#_Toc135980316)

[Приложение Л 145](#_Toc135980317)

[Приложение М 152](#_Toc135980318)

[Приложение Н 163](#_Toc135980319)

[Приложение О 168](#_Toc135980320)

[Приложение П 169](#_Toc135980321)

[Приложение Р 173](#_Toc135980322)

[Приложение С 183](#_Toc135980323)

[Приложение Т 187](#_Toc135980324)

[Приложение У 190](#_Toc135980325)

[Приложение Ф 193](#_Toc135980326)

[Приложение Х 224](#_Toc135980327)

[Приложение Ц 233](#_Toc135980328)

[Приложение Ч 235](#_Toc135980329)

[Приложение Ш 240](#_Toc135980330)

[Приложение Щ 242](#_Toc135980331)

[Приложение Э 244](#_Toc135980332)

Введение

В настоящее время разработка программного обеспечения является важным аспектом современного мира информационных технологий. Графические редакторы играют ключевую роль в этом процессе, позволяя разработчикам создавать графические модели и схемы, что существенно облегчает процесс разработки. Одним из наиболее распространенных инструментов для создания графических схем является диаграмма Насси-Шнейдермана.

Диаграмма Насси-Шнейдермана является важным инструментом для программистов, поскольку она позволяет легко визуализировать и структурировать алгоритмы и программы. Этот графический подход позволяет разбить большую задачу на более мелкие подзадачи и связать их между собой.

Одним из главных преимуществ диаграмм Насси-Шнейдермана является то, что они делают процесс проектирования программ более наглядным и понятным. Благодаря графическому подходу, программист может быстро оценить сложность алгоритма и определить его эффективность. Кроме того, диаграммы Насси-Шнейдермана помогают быстро выявлять ошибки и улучшать код программы.

Цель данной курсовой работы заключается в разработке графического редактора для создания схем Насси-Шнейдермана с использованием векторной графики. Создание такого редактора может упростить процесс проектирования алгоритмов и облегчить работу программистов. Эта работа также включает исследование процесса разработки программного обеспечения, включая анализ аналогов и выбор наилучших решений в разработке.

В данной работе рассматривается процесс разработки графического интерфейса для программного редактора, который является важным аспектом программного обеспечения. В процессе реализации проекта также решаются задачи создания динамических структур данных, работы с файлами (текстовыми и типизированными), чтения/записи данных из файла, а также разработки пользовательского интерфейса для удобного взаимодействия с программой и повышения ее эффективности.

Таким образом, разработка программного обеспечения является важной сферой в информационных технологиях, и графические редакторы играют ключевую роль в этом процессе. Диаграмма Насси-Шнейдермана, являющаяся важным инструментом для программистов, позволяет легко визуализировать и структурировать алгоритмы и программы. Разработка графического редактора для создания схем Насси-Шнейдермана с использованием векторной графики может упростить процесс проектирования алгоритмов и облегчить работу программистов, что делает данную работу актуальной и востребованной.

# Анализ литературных источников и формирование функциональных требований к разрабатываемому программному средство

## 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.

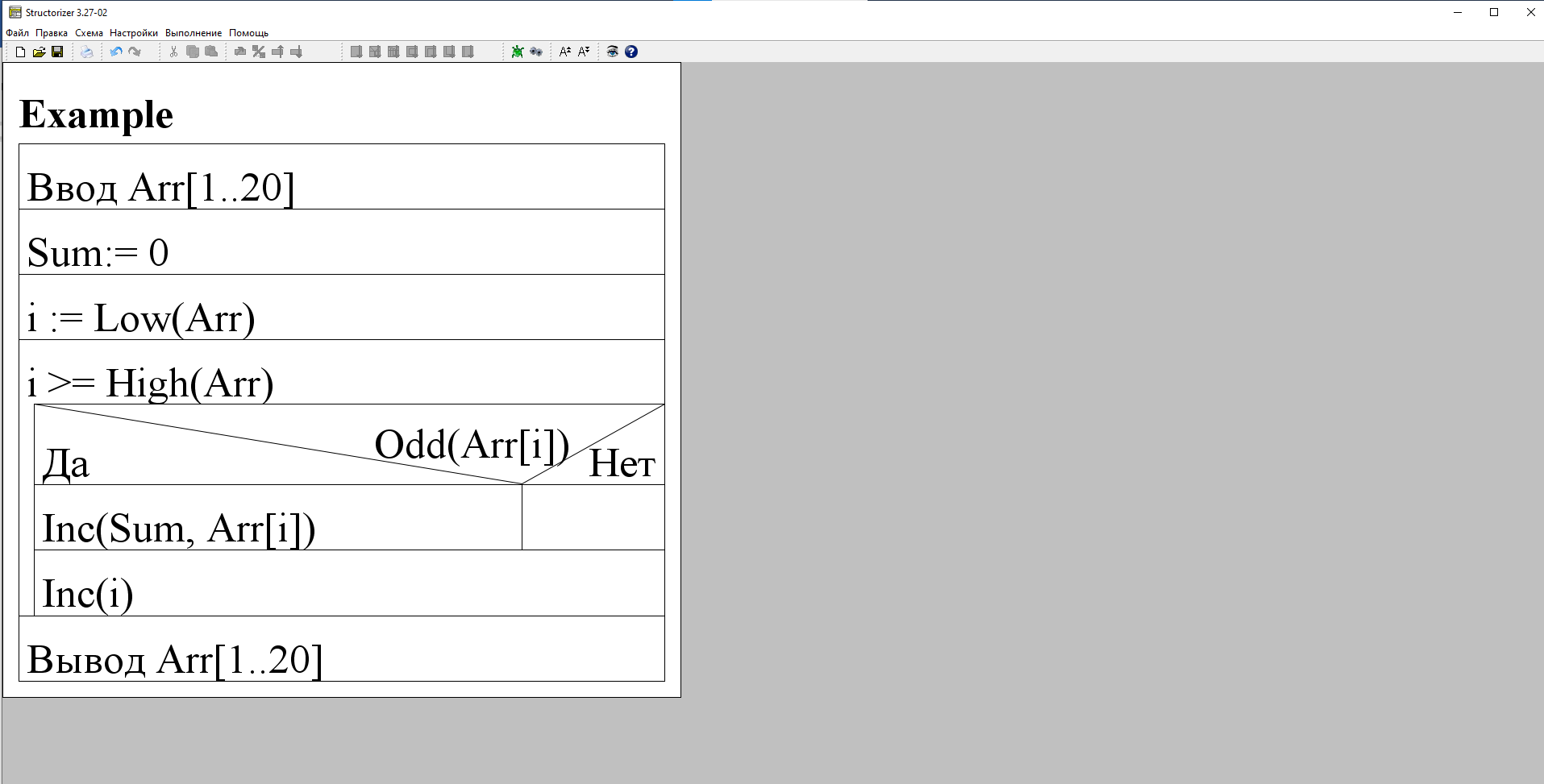


Рисунок 1.1 – Программное средство Structorizer

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.

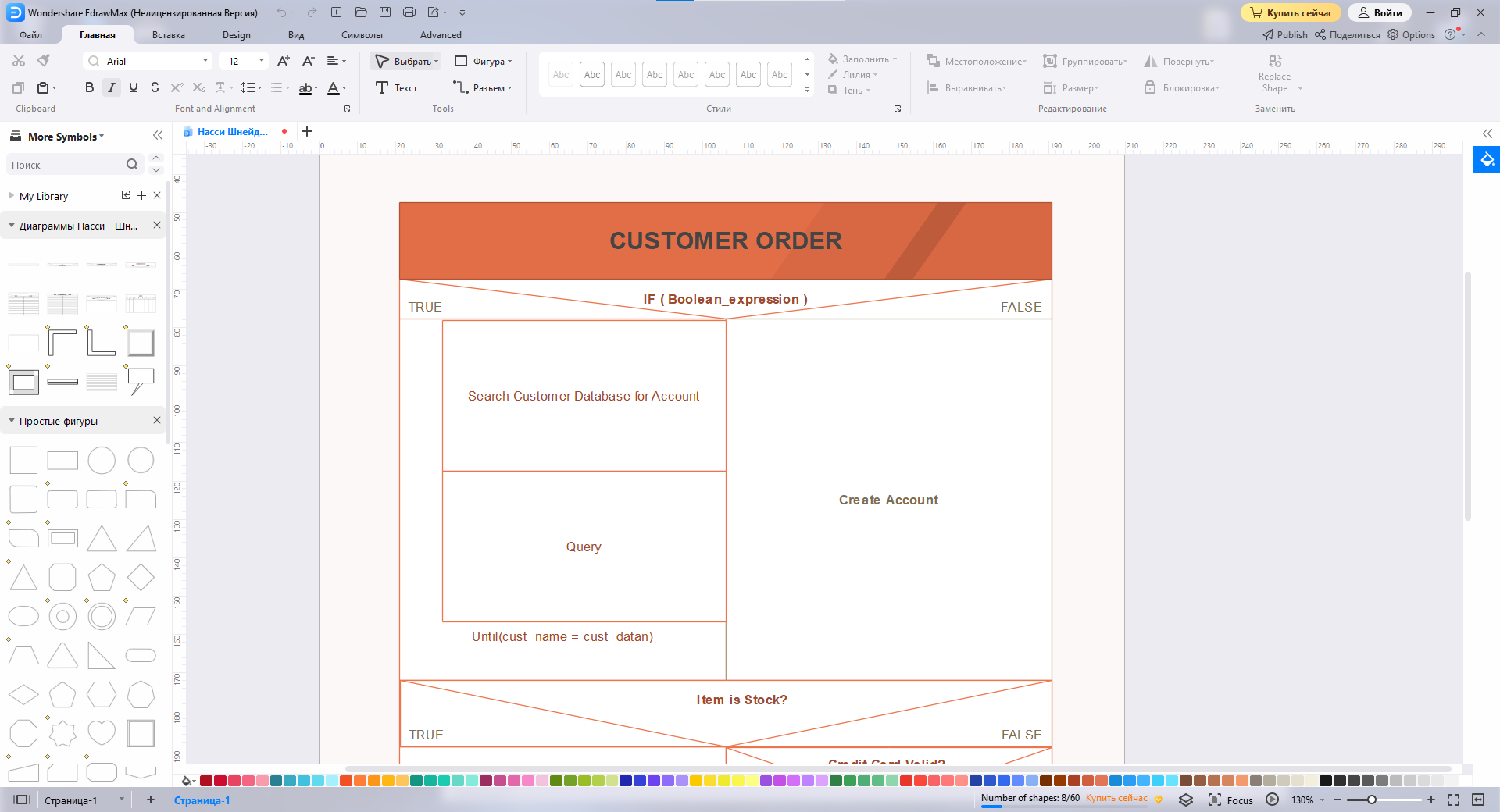


Рисунок 1.2 – Программное средство EdrawMax

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.

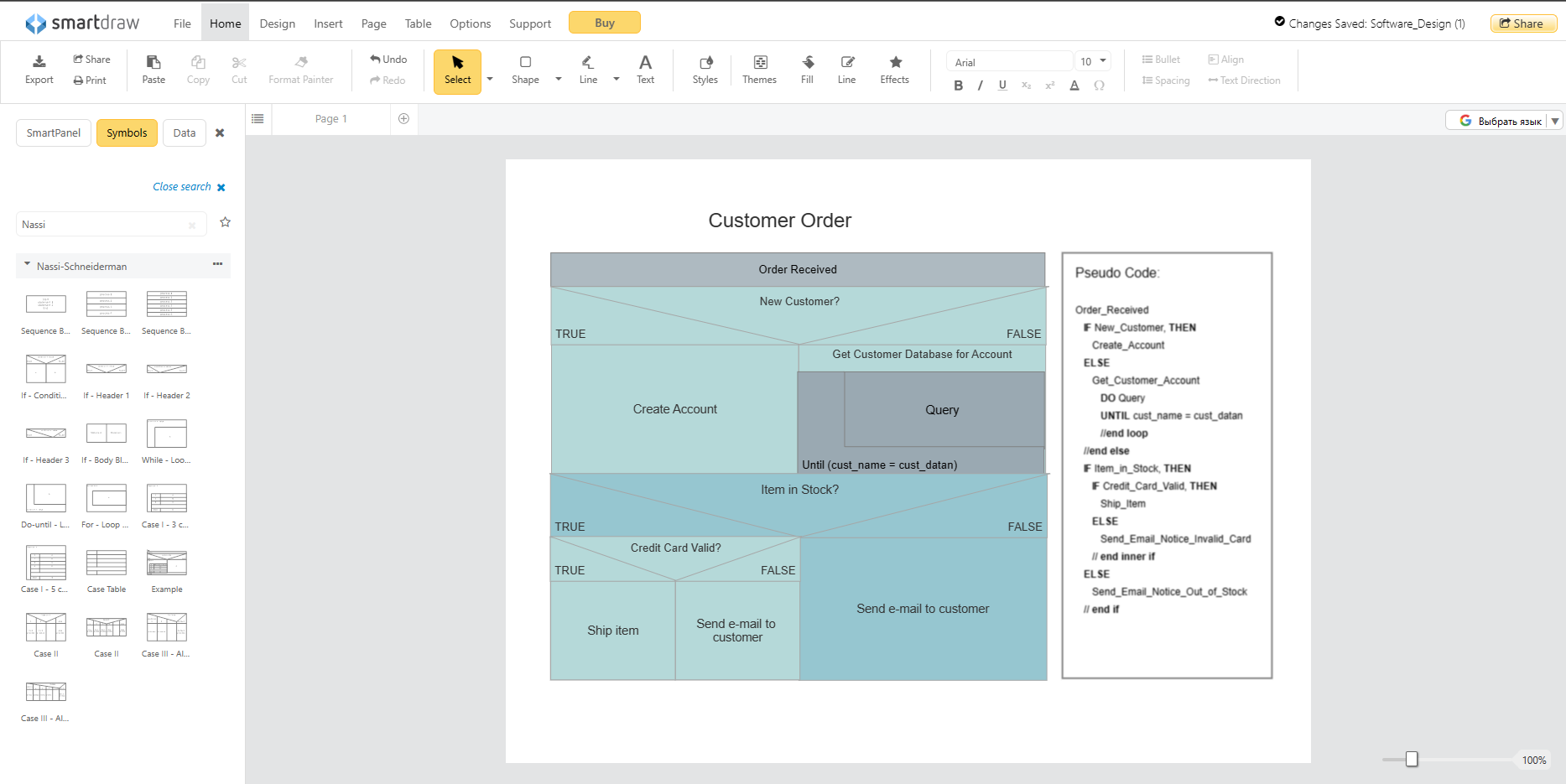


Рисунок 1.3 – Программное средство EdrawMax

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.

## Описание средств разработки

### 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.

Кроме того, в программном средстве реализована возможность работы с текстовым файлом, который используется для отображения лицензионного соглашения приложения. Данный файл содержит важную информацию о лицензии, политике конфиденциальности и других документах, которые необходимо ознакомиться пользователю перед началом использования программы.

## 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 – 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 – 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 |

Продолжение таблицы 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | ChangeGlobalSettings  (  Self,  AOldDefaultAction  ) | Изменяет глобальные настройки для объекта Self. В случае, если новое действие по умолчанию не равно значению параметра AOldDefaultAction, происходит вызов подпрограммы RedefineMainBlock с передачей объекта Self в качестве параметра | Self – получает от фактического параметра адрес с защитой.  AOldDefaultAction – получает от фактического параметра адрес с защитой | Процедура |
| 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 |

Продолжение таблицы 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 8 | TryMoveDedicated  (  Self,  ASetScrollPosProc,  AKey  ) | Перемещает выделенный оператор в переменной Self в соответствии со значением переменной AKey. Если перемещение выполнено успешно, то вызывает подпрограмму ASetScrollPosProc. | Self – получает от фактического параметра адрес с защитой.  ASetScrollPosProc – получает от фактического  параметра адрес с защитой.  AKey – получает от фактического параметра адрес с защитой | Процедура |
| 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 |

Продолжение таблицы 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 12 | ChangeDedicated  (  Self,  AStatement  ) | Меняет значение переменной Self на AStatement для выделенного оператора | Self – получает от фактического параметра адрес с защитой | Процедура |
| 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 |

Продолжение таблицы 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 17 | TryTakeAction  (  Self  ) | Выполняет действие с выделенным блоком переменной Self в зависимости от действия наведенного оператора переменной Self. | Self – получает от фактического параметра адрес с защитой | Процедура |
| 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 |

Продолжение таблицы 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 22 | InitializeBlocks (  Self,  AIndex  ) | Выполняет инициализацию блоков внутри оператора Self. Она устанавливает начальные позиции блоков и производит их выравнивание, начиная с индекса AIndex | Self – получает от фактического параметра адрес с защитой.  AIndex  – получает от фактического параметра значение | Процедура |

## Структура данных

### Структура типов программы

Таблица – 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 |

Продолжение таблицы 3

|  |  |  |
| --- | --- | --- |
| 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 |
| Item | 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 |
| 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 InitializeBlocks(Self, AIndex)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Self | TOperator | Оператор, требующий инициализации | Формальный |
| AIndex | Integer | Индекс, с которого начинается инициализация и выравнивание блоков внутри оператора Self | Формальный |
| I | Integer | Счетчик цикла | Локальный |
| BlockYStart | Integer | Вертикальная позиция, с которой начинается текущий блок | 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 Draw (часть 1)



Рисунок 2.11 – Схема алгоритма Draw (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

Форма Main является основной формой программного средства Насси-Шнейдермана, предоставляющей пользователю доступ к основным функциям программы. Эта форма позволяет пользователю вводить значения, отображать дерево, настраивать его параметры, а также вызывать другие формы и выполнять другие действия. Она обладает соответствующим интерфейсом, представленным на рисунке. Форма Main является ключевым элементом пользовательского опыта в программном средстве Насси-Шнейдермана и обеспечивает удобное и эффективное взаимодействие с функциональностью программы.

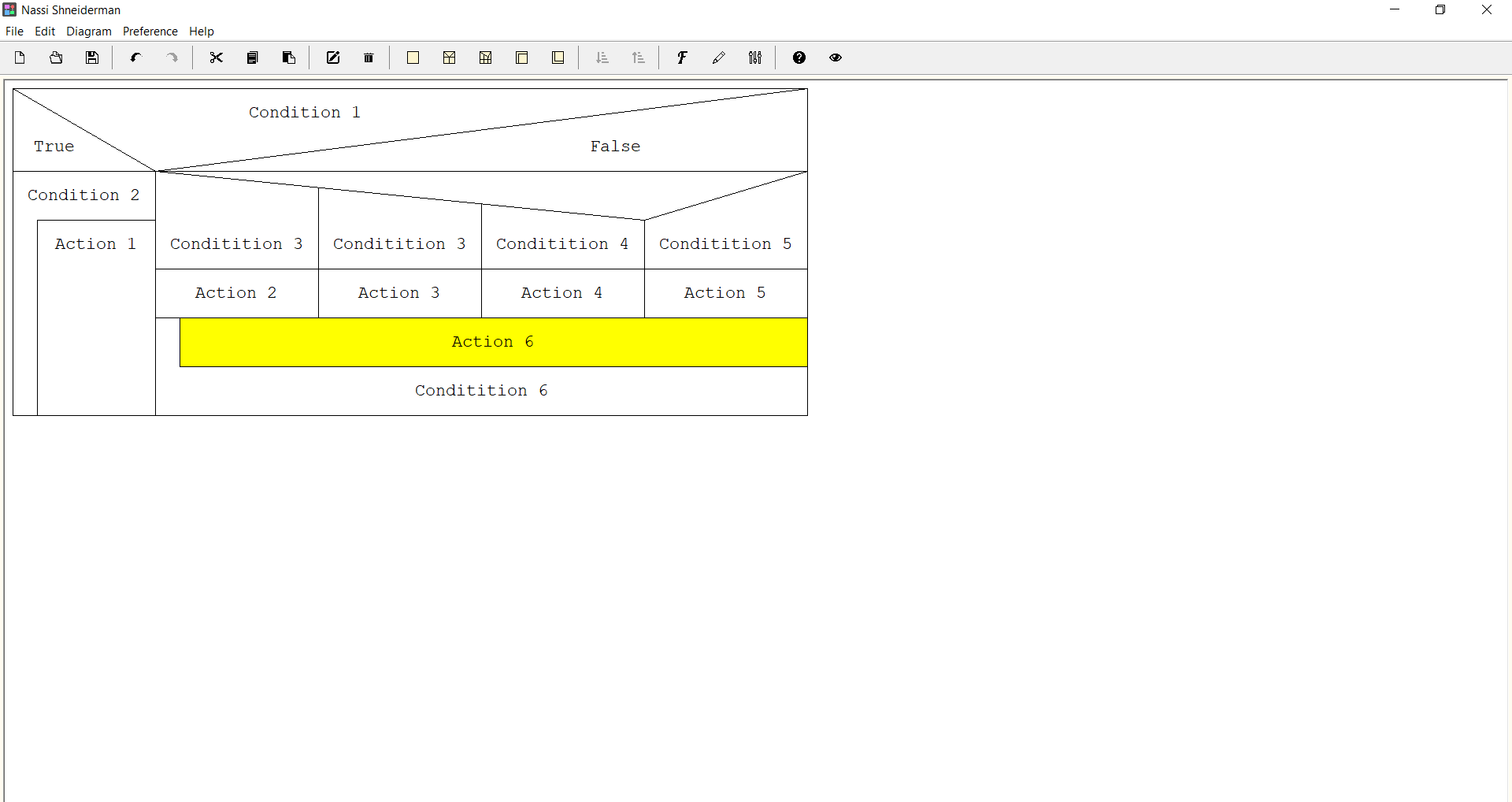


Рисунок 2.12 – Основное окно программы

Составляющие формы Main:

1. Панель «tbMain», на которой расположены следующие компоненты:

* кнопка «tbNew» очищает схему программного средства до начального состояния;
* кнопка «tbOpen» выполняет открытие файла из указанной директории;
* кнопка «tbSave» выполняет перезапись последнего файла либо вызывает «SaveAs»;
* кнопка «tbUndo» отменяет последнее действие;
* кнопка «tbRedo» возвращает отмененное действие;
* кнопка «tbCut» вырезает выделенный блок;
* кнопка «tbCopy» копирует выделенный блок;
* кнопка «tbInsert» вставляет копию после выделенного блока;
* кнопка «tbAction» изменяет действие у выделенного блока;
* кнопка «tbDelete» удаляет выделенный блок;
* кнопка «tbProcess» добавляет блок процесса после выделенного блока;
* кнопка «tbIfBranch» добавляет условный блок после выделенного блока;
* кнопка «tbMultBranch» добавляет блок множественного выбора после выделенного блока;
* кнопка «tbLoop» добавляет цикл с предусловием после выделенного блока;
* кнопка «tbRevLoop» добавляет цикл с постусловием после выделенного блока;
* кнопка «tbSortAsc» сортирует условия блока множественного выбора по возрастанию;
* кнопка «tbSortDecs» сортирует условия блока множественного выбора по убыванию;
* кнопка «tbFont» выполняет изменение шрифта;
* кнопка «tbPen» выполняет изменение кисти;
* кнопка «tbGlSettings» выполняет изменение глобальных настроек;
* кнопка «tbUserGuide» отображает информацию об приложении;
* кнопка «tbAbout» отображает информацию об авторею.

1. Компонент «ScrollBox» содержит в себе компонент «PaintBox», на котором отображается схема.
2. Компонент «MainMenu», содержащий следующие поля:

* поле «mnNew» очищает схему программного средства до начального состояния;
* поле «mnOpen» выполняет открытие файла из указанной директории;
* поле «mnSave» выполняет перезапись последнего файла либо вызывает «SaveAs»;
* поле «mnExpSVG» выполняет сохранение файла в формате SVG указанную директорию;
* поле «mnExpBMP» выполняет сохранение файла в формате BMP указанную директорию;
* поле «mnExpPNG» выполняет сохранение файла в формате PNG указанную директорию;
* поле «mnExit» выполняет закрытие приложения;
* поле «mnUndo» отменяет последнее действие;
* поле «mnRedo» возвращает отмененное действие;
* поле «mnCut» вырезает выделенный блок;
* поле «mnCopy» копирует выделенный блок;
* поле «mnInsert» вставляет копию после выделенного блока;
* поле «mnAction» изменяет действие у выделенного блока;
* поле «mnDelete» удаляет выделенный блок;
* поле «mnAction» изменяет действие у выделенного блока;
* поле «mnDelete» удаляет выделенный блок;
* поле «mnAftProcess» добавляет блок процесса после выделенного блока;
* поле «mnAftIfBranch» добавляет условный блок после выделенного блока;
* поле «mnAftMultBranch» добавляет блок множественного выбора после выделенного блока;
* поле «mnAftLoop» добавляет цикл с предусловием после выделенного блока;
* поле «mnAftRevLoop» добавляет цикл с постусловием после выделенного блока;
* поле «mnBefProcess» добавляет блок процесса перед выделенным блоком;
* поле «mnBefIfBranch» добавляет условный блок перед выделенным блоком;
* поле «mnBefMultBranch» добавляет блок множественного выбора перед выделенным блоком;
* поле «mnBefLoop» добавляет цикл с предусловием перед выделенным блоком;
* поле «mnBefRevLoop» добавляет цикл с постусловием перед выделенным блоком;
* поле «mnSortAsc» сортирует условия блока множественного выбора по возрастанию;
* поле «mnSortDecs» сортирует условия блока множественного выбора по поле;
* поле «mnFont» выполняет изменение шрифта;
* поле «mnPen» выполняет изменение кисти;
* поле «mnGlSettings» выполняет изменение глобальных настроек;
* поле «mnUserGuide» отображает информацию об приложении;
* поле «mnAbout» отображает информацию об авторе;
* поле «mnCurrentStat» отображает текущую статистику пользователя;
* поле «mnOtherStat» выполняет открытие файла статистика из указанной директории и отображает ее.

### Описание графических компонентов формы frmGetAction

Форма frmGetAction предоставляет пользователю возможность ввода действий для блока, позволяя определить необходимые операции или инструкции.

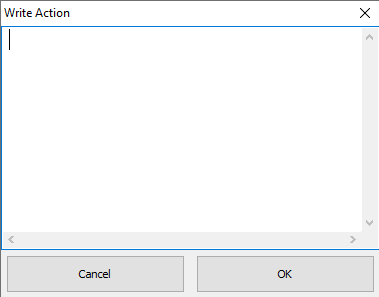


Рисунок 2.13 – Окно ввода действия

Составляющие формы frmGetAction:

* кнопка «btnCancel» отменяет ввод действия;
* кнопка «btnOk» сохраняет ввод действия;
* компонент «MemoAction» осуществляет ввод действия.

### Описание графических компонентов формы frmGetCaseConditions

Форма frmGetCaseConditions предоставляет пользователю возможность ввода условий, позволяя определить различные сценарии или варианты выполнения в зависимости от заданных условий.

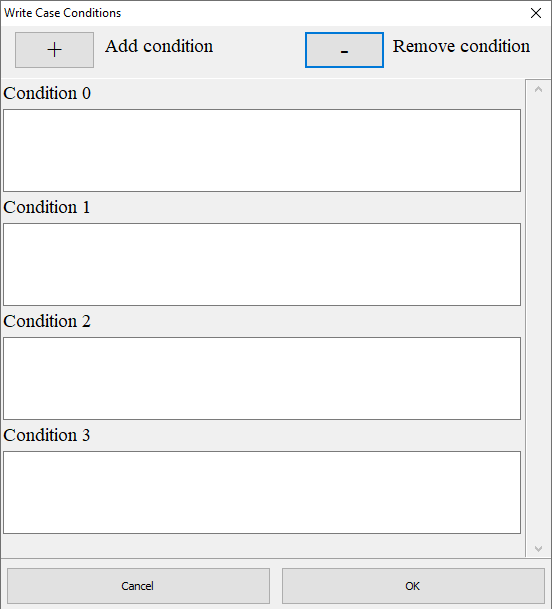


Рисунок 2.14 – Окно ввода условий

Составляющие формы frmGetCaseConditions:

* кнопка «btnCancel» отменяет ввод действия;
* кнопка «btnOk» сохраняет ввод действия;
* кнопка «btnAdd» добавляет условие;
* кнопка «btnDelete» удаляет условие;
* компоненты «mmFirst, mmSecond, mmThird, mmFourth» осуществляет ввод условий.

### Описание графических компонентов формы frmGlobalSettings

Форма "frmGlobalSettings" предоставляет пользователю возможность изменения глобальных настроек схемы, позволяя настраивать параметры и свойства, которые применяются ко всем блокам и элементам схемы. Это позволяет пользователю осуществлять широкий контроль над общими характеристиками схемы и адаптировать их под свои потребности и предпочтения.

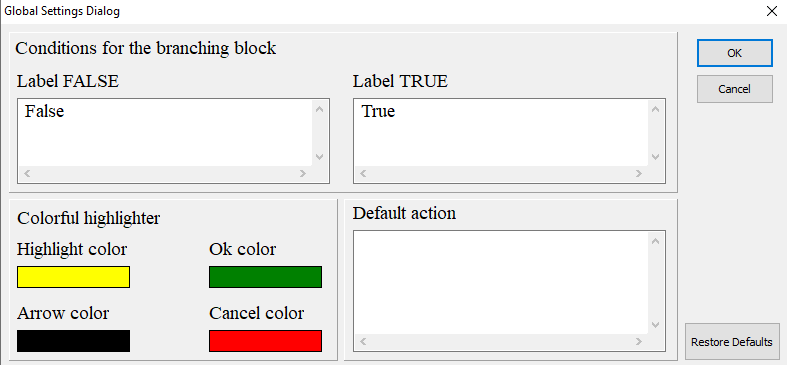


Рисунок 2.15 – Окно глобальных настроек

Составляющие формы frmGlobalSettings:

* кнопка «btnCancel» отменяет ввод действия;
* кнопка «btnOk» сохраняет ввод действия;
* кнопка «btnRestore» восстанавливает изначальные настройки программы;
* компонент «shpHighlight» устанавливает цвет для выделенного блока;
* компонент «shpOK» устанавливает цвет для отображения допустимо-сти операции;
* компонент «shpCancel» устанавливает цвет для отображения недопустимо-сти операции;
* компонент «shpArrow» устанавливает цвет для отображения стрелок;
* компонент «mmTrue» осуществляет ввод для истинного условия;
* компонент «mmFalse» осуществляет ввод для ложного условия;
* компонент «mmDefault» осуществляет ввод ввод для действия блока по умолчанию.

### Описание графических компонентов формы frmPenSettings

Форма frmPenSettings предоставляет пользователю возможность изменять настройки кисти, позволяя настраивать параметры и характеристики, связанные с рисованием и отображением элементов на схеме. Здесь пользователь может настраивать толщину кисти, тип линий и выбирать цвет, определяющий внешний вид рисуемых элементов. Это позволяет пользователю индивидуализировать стиль и эстетические аспекты схемы в соответствии с его предпочтениями и требованиями.

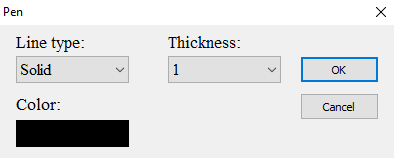


Рисунок 2.16 – Окно настройки кисти

Составляющие формы frmPenSettings:

* кнопка «btnCancel» отменяет ввод действия;
* кнопка «btnOk» сохраняет ввод действия;
* компонент «cbLineType» устанавливает тип линий кисти;
* компонент «cbThickness» устанавливает толщину кисти;
* компонент «CurrColor» устанавливает цвет кисти.

### Описание графических компонентов формы frmHelp

Форма frmHelp предоставляет пользователю возможность получить информацию о программе или авторе. Здесь пользователь может ознакомиться с справочными материалами, руководством пользователя или другой полезной информацией, которая поможет ему более полно использовать программное средство. Также форма может содержать информацию об авторе, его контактных данных или другие сведения, которые могут быть интересными для пользователей программы.

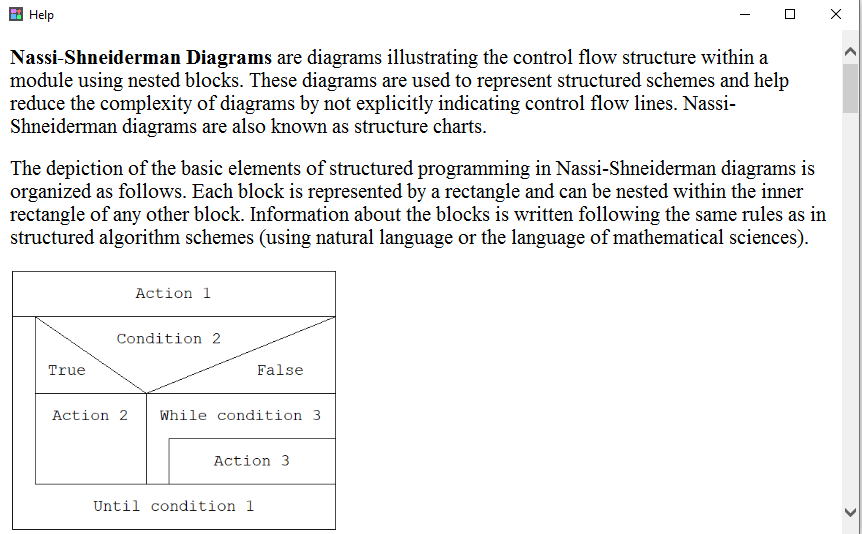


Рисунок 2.17 – Окно информации

Форма содержит компонент «WebBrowser» для отображения HTML документа.

# Тестирование и проверка работоспособности программного средства

## Тестирование основной формы

### Тест 1

Таблица 25 – Тест 1

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности поведения программы при запуске |
| Исходный набор данных: | Запуск программы |
| Ожидаемый результат: | Открытие формы Main, отображение схемы |
| Полученный результат: |  |

### Тест 2

Таблица 26 – Тест 2

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности поведения программы при добавлении блока процесса |
| Исходный набор данных: | Нажатие на кнопку tbProcess, ввод действия |
| Ожидаемый результат: | Добавление блока процесса |
| Полученный результат: |  |

### Тест 3

Таблица 27 – Тест 3

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности поведения программы при добавлении условного блока |
| Исходный набор данных: | Нажатие на кнопку tbIfBranch, ввод действия |
| Ожидаемый результат: | Добавление условного блока |
| Полученный результат: |  |

### Тест 4

Таблица 28 – Тест 4

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности поведения программы при добавлении блока множественного выбора |
| Исходный набор данных: | Нажатие на кнопку tbIfMultBranch, ввод действия и условий |
| Ожидаемый результат: | Добавление блока множественного выбора |
| Полученный результат: |  |
|  |  |

### Тест 5

Таблица 29 – Тест 5

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности поведения программы при сортировке условий блока множественного выбора |
| Исходный набор данных: | Нажатие на кнопку tbSortAsc |
| Ожидаемый результат: | Отсортированные условия по возрастанию |
| Полученный результат: |  |

### Тест 6

Таблица 30 – Тест 6

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности поведения программы при сортировке условий блока множественного выбора |
| Исходный набор данных: | Нажатие на кнопку tbSortDesc |
| Ожидаемый результат: | Отсортированные условия по убыванию |
| Полученный результат: |  |

### Тест 7

Таблица 31 – Тест 7

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности поведения программы при добавлении цикла с предусловием |
| Исходный набор данных: | Нажатие на кнопку tbLoop, ввод действия |
| Ожидаемый результат: | Добавление цикла с предусловием |
| Полученный результат: |  |

### Тест 8

Таблица 32 – Тест 8

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности поведения программы при добавлении цикла с постусловием |
| Исходный набор данных: | Нажатие на кнопку tbRevLoop, ввод действия |
| Ожидаемый результат: | Добавление цикла с постусловием |
| Полученный результат: |  |

### Тест 9

Таблица 33 – Тест 9

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности поведения программы при удалении блока |
| Исходный набор данных: | Нажатие на компонент PaintBox в области, где расположен условный блок. Затем нажатие на кнопку tbDelete |
| Ожидаемый результат: | Удаление условного блока |
| Полученный результат: |  |

## Тестирование формы frmGetAction

### Тест 1

Таблица 34 – Тест 1

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности поведения программы при изменении действия на однострочный текст у блока |
| Исходный набор данных: | Ввод однострочного текста |
| Ожидаемый результат: | Изменение действия у блока |
| Полученный результат: |  |

### Тест 2

Таблица 35 – Тест 2

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности поведения программы при изменении действия на многострочный текст у блока |
| Исходный набор данных: | Ввод многострочного текста |
| Ожидаемый результат: | Изменение действия у блока |
| Полученный результат: |  |

## Тестирование формы frmGetCaseConditions

### Тест 1

Таблица 36 – Тест 1

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности поведения программы при изменении условий на однострочный текст у блока множественного выбора |
| Исходный набор данных: | Ввод однострочного текста |
| Ожидаемый результат: | Изменение условий у блока множественного выбора |
| Полученный результат: |  |

### Тест 2

Таблица 37 – Тест 2

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности поведения программы при изменении условий на многострочный текст у блока множественного выбора |
| Исходный набор данных: | Ввод многострочного текста |
| Ожидаемый результат: | Изменение условий у блока множественного выбора |
| Полученный результат: |  |

## Тестирование формы frmGlobalSettings

### Тест 1

Таблица 38 – Тест 1

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности поведения программы при изменении глобальный настроек |
| Исходный набор данных: | Производится изменение глобальных настроек, включающих изменение условий правды и лжи для условного блока и изменение текста для блока по умолчанию на однострочный текст |
| Ожидаемый результат: | Изменение глобальный настроек |
| Полученный результат: |  |

### Тест 2

Таблица 39 – Тест 2

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности поведения программы при изменении глобальный настроек |
| Исходный набор данных: | Производится изменение глобальных настроек, включающих изменение условий правды и лжи для условного блока и изменение текста для блока по умолчанию на многострочный текст |
| Ожидаемый результат: | Изменение глобальный настроек |
| Полученный результат: |  |

## Тестирование формы frmPenSettings

### Тест 1

Таблица 40 – Тест 1

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности поведения программы при изменении кисти |
| Исходный набор данных: | Выполняется настройка кисти: толщина линии 10, цвет синий |
| Ожидаемый результат: | Изменение визуального отображения схемы |

Продолжение таблицы 40

|  |  |
| --- | --- |
| Полученный результат: |  |

### Тест 2

Таблица 41 – Тест 2

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности поведения программы при изменении кисти |
| Исходный набор данных: | Выполняется настройка кисти: толщина линии 4, цвет красный |
| Ожидаемый результат: | Изменение визуального отображения схемы |
| Полученный результат: |  |

## Тестирование формы frmHelp

### Тест 1

Таблица 42 – Тест 1

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности поведения программы при открытии формы Help с параметром «UserGuide» |
| Исходный набор данных: | Нажатие на кнопку tbUserGuide |
| Ожидаемый результат: | Отображение справочного материала и руководства |
| Полученный результат: |  |

### Тест 2

Таблица 43 – Тест 2

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности поведения программы при открытии формы Help с параметром «About» |
| Исходный набор данных: | Нажатие на кнопку tbAbout |
| Ожидаемый результат: | Отображение информации об авторе |

Продолжение таблицы 43

|  |  |
| --- | --- |
| Полученный результат: |  |

# Руководство по установке

## Минимальные системные требования

Для успешного запуска данного программного средства и комфортной работы с ним необходимо соответствие минимальным системным требованиям:

* процессор 1000 МГц или выше;
* объем оперативной памяти не менее 32 МБ;
* свободное место на диске не менее 21.5 МБ;
* операционная система Windows XP и выше.

## Установка

На установочном диске содержится установочный файл программного средства. После открытия этого пакета, на экране появляется окно, изображенное на рисунке 4.1. В этом окне предоставляется возможность выбрать язык установки. Установщик предлагает выбор между русским и английским языками.

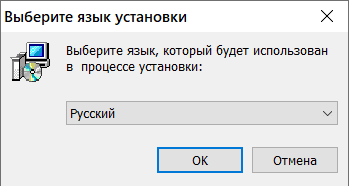


Рисунок 4.1 – Выбор языка

После выбора языка установки, появляется приветственное окно, изображенное на рисунке 4.2. В этом окне рекомендуется закрыть все активные приложения перед продолжением установки. Затем, предлагается нажать на кнопку «Далее», чтобы перейти к следующему шагу установки.

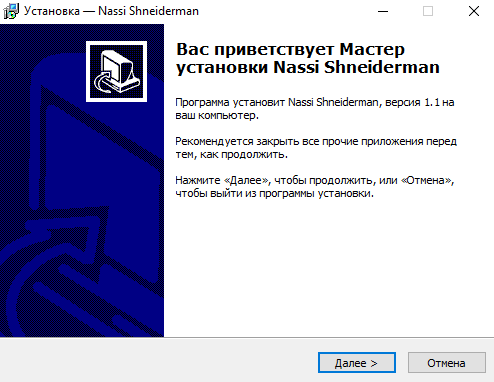


Рисунок 4.2 – Установка

После приветствия и рекомендации закрыть приложения, пользователь переходит к следующему этапу - просмотру и принятию лицензионного соглашения. Окно, изображенное на рисунке 4.3, содержит текст лицензионного соглашения, который может быть прочитан пользователем. После ознакомления с условиями лицензии, пользователь может принять их, нажав на соответствующую кнопку для продолжения установки.

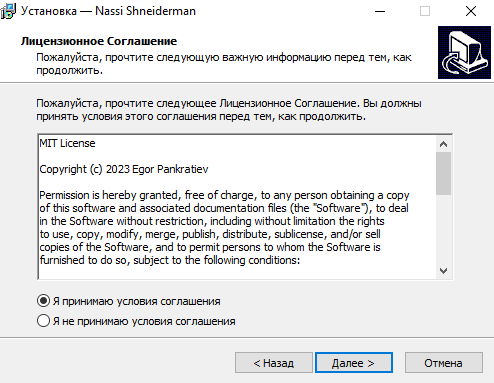


Рисунок 4.3 – Лицензионное Соглашение

На следующем этапе предлагается выбрать место для установки программного средства. На рисунке 4.4 иллюстрируется этот этап, где можно обратить внимание на информацию о минимальном требуемом объеме свободного дискового пространства, необходимого для загрузки приложения на диск (21.5 МБ).

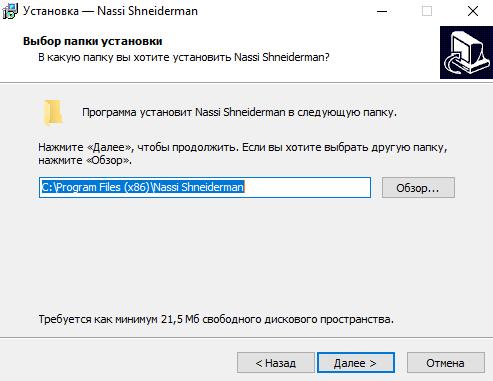


Рисунок 4.4 – Выбор папки установки

Далее предоставляется возможность выбрать дополнительные параметры загрузки. Один из параметров - создание ярлыка в меню «Пуск». Окно с настройкой этой функции представлено на рисунке 4.5. Это позволяет установить ярлык приложения в меню «Пуск» для более удобного доступа из главного меню операционной системы.

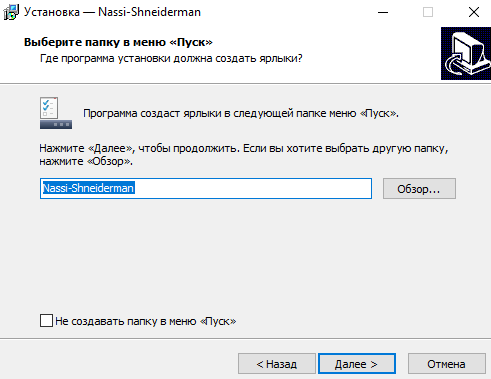


Рисунок 4.5 – Выбор папки в меню «Пуск»

На следующем этапе установки данного программного средства, предлагается создать ярлык на Рабочем столе. Окно, показанное на рисунке 4.6, отображает эту опцию. Здесь можно выбрать создание значка на Рабочем столе, что обеспечит удобный доступ к программе прямо с Рабочего стола операционной системы.

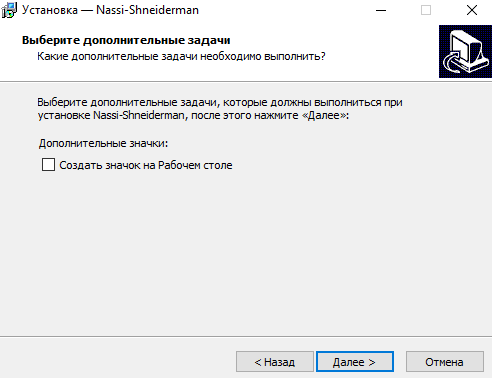


Рисунок 4.6 – Дополнительные задачи

По завершении этапа подготовки к установке, предлагается установить программное средство на компьютер. Окно, показанное на рисунке 4.7, отображает данное предложение. В этом окне можно подтвердить намерение установить программу и начать процесс установки на компьютере.

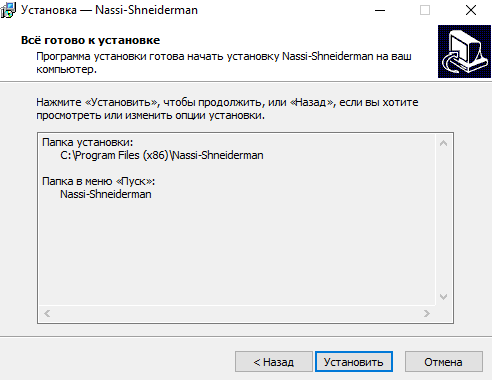


Рисунок 4.7 – Установка

По завершении установки программного средства, пользователю предоставляется информация об успешном завершении процесса. Эта информация отображается в окне, представленном на рисунке 4.8. Здесь пользователь получает подтверждение того, что установка прошла успешно и программное средство готово к использованию.

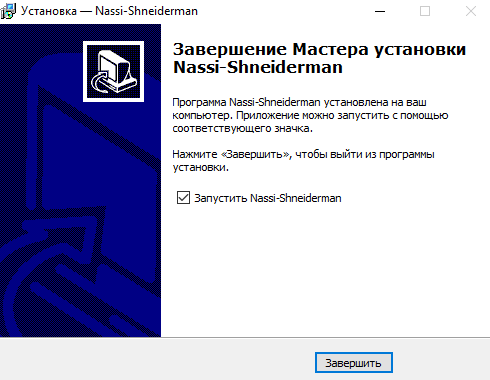


Рисунок 4.8 – Завершение

## Работа с приложением

После установки и запуска графического редактора для создания схем по методу Насси-Шнейдермана, пользователь попадает в рабочее окно, которое специально разработано для этой цели. Это окно, изображенное на рисунке 4.9, предоставляет пользователю все необходимые инструменты и функции для создания и редактирования схем, основанных на графическом подходе Насси-Шнейдермана. Здесь пользователь может легко проектировать и визуализировать структуру программы, используя блоки, связи и другие элементы, чтобы отобразить логику управления и последовательность действий в понятной и наглядной форме.

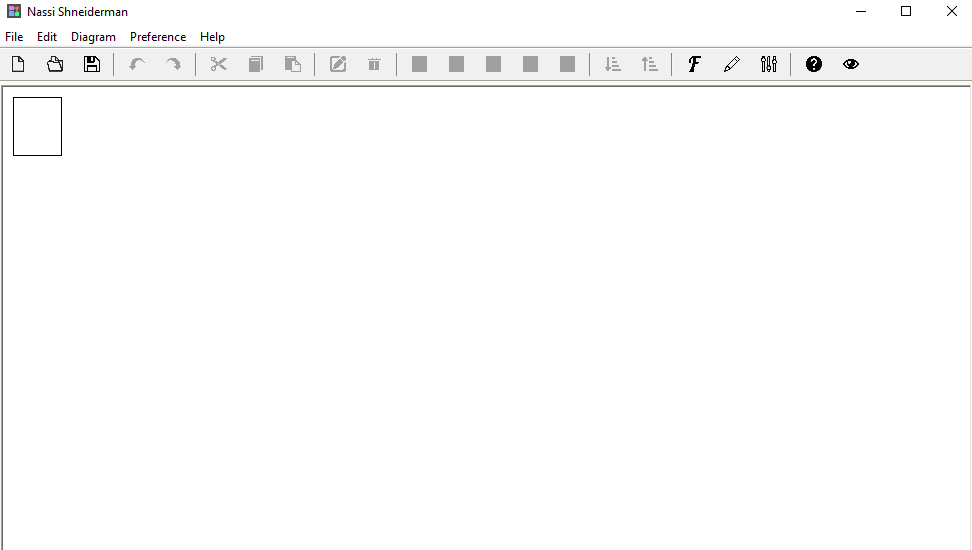


Рисунок 4.9 – Окно программы

Для добавления блоков в программу доступны следующие кнопки, которые можно увидеть на рисунке 4.10.



Рисунок 4.10 – Список доступных блоков

В наборе блоков имеются блок процесса, условный блок, блок множественного выбора, цикл с предусловием и цикл с постусловием. Блок процесса используется для представления последовательности действий или операций. Условный блок позволяет определить ветвление в программе в зависимости от выполнения определенного условия. Блок множественного выбора предоставляет возможность выбора из нескольких вариантов в зависимости от значения переменной или условия. Цикл с предусловием позволяет многократно выполнять блок кода, пока определенное условие остается истинным. Цикл с постусловием позволяет многократно выполнять блок кода, и затем проверять условие для продолжения или прерывания цикла. Эти блоки предоставляют программисту гибкость и функциональность для построения структуры программы в соответствии с требуемой логикой и потоком выполнения.

Для создания блока необходимо сначало указать соответствующее действие, как показано на рисунке 4.11.

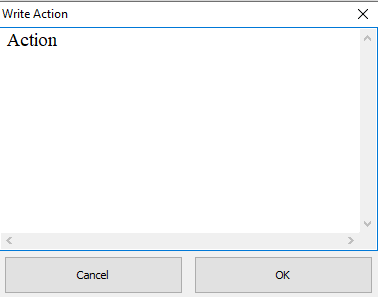


Рисунок 4.11 – Ввод действия

При создании блока множественного выбора также требуется задать условия для каждого варианта. Этот процесс наглядно демонстрируется на рисунке 4.12.

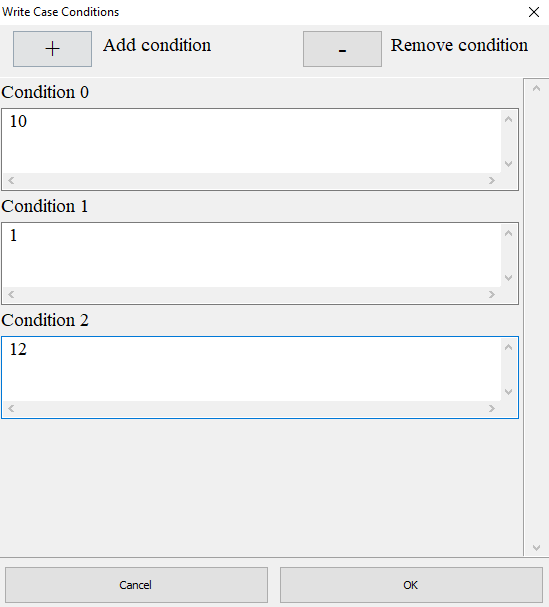


Рисунок 4.12 – Ввод условий

В программе предусмотрены различные настройки, которые позволяют настраивать отображение схемы в соответствии с предпочтениями пользователя. Настройки включают в себя возможность изменения шрифта, выбор определенной кисти, а также настройку различных цветов для разных сценариев. Эти настройки позволяют пользователю настроить внешний вид схемы в соответствии с их предпочтениями и создать уникальный стиль для своих сценариев. Благодаря этим настройкам, пользователи получают больше гибкости и контроля над визуальным представлением своих схем.



Рисунок 4.13 – Настройки

В заключение, для более подробной информации о функциях и настройках программного средства, рекомендуется обратиться к «User Guide» (Руководство пользователя). В нем вы найдете всю необходимую информацию о работе с программой, включая подробные инструкции, советы и рекомендации. Данное руководство является ценным ресурсом, который поможет вам максимально эффективно использовать программное средство и достичь желаемых результатов.

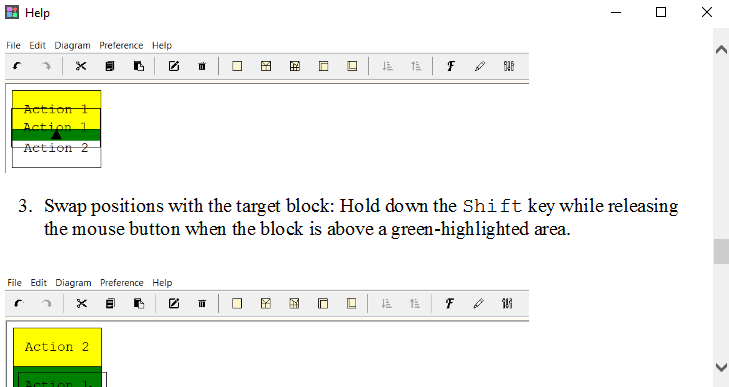


Рисунок 4.14 – Руководство пользователя

Заключение

В результате выполнения данного курсового проекта был разработан графический редактор для создания схем Насси-Шнейдермана. Разработка такого программного средства является важным шагом в области программирования и проектирования алгоритмов. Графический подход, предоставляемый этим редактором, позволяет программистам визуализировать и структурировать свои алгоритмы, что упрощает процесс разработки и повышает эффективность программ.

Редактор предоставляет несколько типов блоков, таких как блок процесса, условный блок, блок множественного выбора, цикл с предусловием и цикл с постусловием. Каждый блок может быть создан с помощью соответствующей кнопки на графическом интерфейсе программы.

Для удобства пользователей редактор предлагает настройки, которые позволяют настроить шрифт, кисть, а также выбрать различные цвета для различных сценариев. Настройки доступны через соответствующие кнопки в программе.

В ходе разработки данного программного средства были применены знания и навыки работы с векторной графикой, динамическими структурами данных, а также взаимодействием с файлами. Был учтен графический интерфейс пользователя, чтобы обеспечить удобство использования программы.

Данная работа не исчерпывает все возможности и потенциал разработки графического редактора для схем Насси-Шнейдермана. В будущем можно рассмотреть возможности оптимизации алгоритмов, добавления новых функций и улучшения интерфейса, чтобы обеспечить еще более удобное и эффективное использование программы.

В целом, разработка данного программного средства для создания схем Насси-Шнейдермана является актуальной и востребованной задачей, помогающей программистам в их повседневной работе и упрощающей процесс разработки программного обеспечения.

Список использованных источников

[1] Кнут Д.Э. Искусство программирования: Учеб. пособие. Т. 1. Основные алгоритмы. – М.: Вильямс, 2000. – 722 с.: ил.

[2] Вирт, Н. Алгоритмы и структуры данных. – М.: Мир, 1989. – 360 с.: ил.

[3] Серебряная, Л.В. Структуры и алгоритмы обработки данных : учеб.-метод. Пособие / Л. В. Серебряная, И. М. Марина. – Минск : БГУИР, 2013 – 51 с.

[4] Глухова, Л. А. Электронный учебно-методический комплекс по дисциплине «Основы алгоритмизации и программирования». Часть 1. [Электронный ресурс]. Режим доступа: <https://erud.bsuir.by/>. – Дата доступа: 15.03.2023..

[5] Фленов, М. Е. Библия Delphi. – 3-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2011. – 688 с.: ил.

[6] Тюкачев, Н. А. Программирование графики в Delphi/ Н. А. Тюкачев, В. Илларионов, В. Г. Хлебостроев. – СПб.:БХВ-Петербург, 2008. – 784 с.

[7] Глухова, Л.А. Основы алгоритмизации и программирования. Лабора-торный практикум: учеб. – метод. пособие. В 4 ч. Ч. 4/ Л. А. Глухова, Е.П. Фа-деева, Е.Е. Фадеева. – Минск: БГУИР, 2012. – 58 с.

[8] Фаронов, В.В. Delphi 6. Учебный курс.-М.: Издатель Молгачева С.В., 2001. – 672 с.

[9] Документация по SVG – MSDN [Электронный ресурс] Режим доступа: https://msdn.microsoft.com/library/bg124132(v=vs.85).aspx – Дата доступа: 06.04.18

[10] SVG – MDN web docs [Электронный ресурс] Режим доступа: https://developer.mozilla.org/docs/Web/SVG#Documentation – Дата доступа: 04.04.18

Приложение А

(обязательное)

Исходный код программы (модуль frmMain)

unit frmMain;

interface

uses

Winapi.Windows, Winapi.Messages, System.Classes,

Vcl.Graphics, Vcl.Controls,

Vcl.Forms, Vcl.Dialogs, Vcl.ExtCtrls, Vcl.StdCtrls,

Vcl.Menus, uConstants,

uBase, uFirstLoop, uIfBranching, uCaseBranching,

uLastLoop, uProcessStatement,

uStatementSearch, System.Actions, Vcl.ActnList,

Vcl.ToolWin, Types, uBlockManager,

Vcl.ComCtrls, uAdditionalTypes, frmPenSetting, Sys

tem.ImageList, Vcl.ImgList,

System.SysUtils, uGlobalSave, uLocalSave, frmHelp,

uStatementConverter, uDialogMessages,

frmGlobalSettings, System.UITypes, uExport, uStatis

tics;

type

TNassiShneiderman = class(TForm)

tbMain: TToolBar;

ilIcons: TImageList;

ScrollBox: TScrollBox;

PopupMenu: TPopupMenu;

MIAdd: TMenuItem;

MIAfter: TMenuItem;

MIBefore: TMenuItem;

MIAftProcess: TMenuItem;

MIAftBranch: TMenuItem;

MIAftMultBranch: TMenuItem;

MIAftTestLoop: TMenuItem;

MIAftRevTestLoop: TMenuItem;

MIBefProcess: TMenuItem;

MIBefBranch: TMenuItem;

MIBefMultBranch: TMenuItem;

MIBefTestLoop: TMenuItem;

MIBefRevTestLoop: TMenuItem;

MICut: TMenuItem;

MICopy: TMenuItem;

MIInset: TMenuItem;

alActions: TActionList;

actAfterProcess: TAction;

actAfterIfBranch: TAction;

actAfterMultBranch: TAction;

actAfterLoop: TAction;

actAfterRevLoop: TAction;

tbProcess: TToolButton;

tbIfBranch: TToolButton;

tbMultBranch: TToolButton;

tbLoop: TToolButton;

tbRevLoop: TToolButton;

actBeforeProcess: TAction;

actBeforeIfBranch: TAction;

actBeforeMultBranch: TAction;

actBeforeLoop: TAction;

actBeforeRevLoop: TAction;

actCopy: TAction;

actInsert: TAction;

actCut: TAction;

N1: TMenuItem;

N3: TMenuItem;

actDelete: TAction;

MIDelete: TMenuItem;

actSortAsc: TAction;

actSortDesc: TAction;

MIDescSort: TMenuItem;

MIAscSort: TMenuItem;

N2: TMenuItem;

actChangeAction: TAction;

MIChangeAction: TMenuItem;

PaintBox: TPaintBox;

actUndo: TAction;

actRedo: TAction;

N4: TMenuItem;

MIUndo: TMenuItem;

MIRedo: TMenuItem;

MainMenu: TMainMenu;

mnFile: TMenuItem;

mnNew: TMenuItem;

mnOpen: TMenuItem;

mnSave: TMenuItem;

mnSaveAs: TMenuItem;

mnExport: TMenuItem;

mnPrefer: TMenuItem;

mnFont: TMenuItem;

mnPen: TMenuItem;

actChngFont: TAction;

actChngPen: TAction;

FontDialog: TFontDialog;

ColorDialog: TColorDialog;

sep1: TToolButton;

tbFont: TToolButton;

tbPen: TToolButton;

sep2: TToolButton;

tbDelete: TToolButton;

tbAction: TToolButton;

sep3: TToolButton;

tbInsert: TToolButton;

tbCopy: TToolButton;

tbCut: TToolButton;

tbUndo: TToolButton;

tbRedo: TToolButton;

sep4: TToolButton;

tbSortDesc: TToolButton;

tbSortAsc: TToolButton;

sep5: TToolButton;

actChngGlSettings: TAction;

Globalsettings1: TMenuItem;

SaveDialog: TSaveDialog;

OpenDialog: TOpenDialog;

mnDiagram: TMenuItem;

mnAdd: TMenuItem;

mnAfter: TMenuItem;

mnBefore: TMenuItem;

mnAftProcess: TMenuItem;

mnAftBranchingBlock: TMenuItem;

mnAftMultBranchingBlock: TMenuItem;

mnAftReversedLoop: TMenuItem;

mnAftLoop: TMenuItem;

actBefProcess: TMenuItem;

mnBefBranchingBlock: TMenuItem;

mnBefMultBranchingBlock: TMenuItem;

mnBefLoop: TMenuItem;

mnBefReversedLoop: TMenuItem;

mnChangeAct: TMenuItem;

mnDelete: TMenuItem;

mnSortDesc: TMenuItem;

mnSortAsc: TMenuItem;

mnEdit: TMenuItem;

N5: TMenuItem;

mnUndo: TMenuItem;

mnRedo: TMenuItem;

N6: TMenuItem;

mnCut: TMenuItem;

mnCopy: TMenuItem;

mnInsert: TMenuItem;

mnHelp: TMenuItem;

mnUserGuide: TMenuItem;

mnAbout: TMenuItem;

actUserGuide: TAction;

actAbout: TAction;

sep6: TToolButton;

tbUserGuide: TToolButton;

tbAbout: TToolButton;

actExit: TAction;

Exit1: TMenuItem;

actSaveAs: TAction;

actSave: TAction;

actOpen: TAction;

actNew: TAction;

tbGlSettings: TToolButton;

sep7: TToolButton;

tbSaveAs: TToolButton;

tbOpen: TToolButton;

tbNew: TToolButton;

mnStatistics: TMenuItem;

mnCurrent: TMenuItem;

mnOther: TMenuItem;

actExpPNG: TAction;

actExpBMP: TAction;

actExpSVG: TAction;

mnExpSVG: TMenuItem;

mnExpBMP: TMenuItem;

mnExpPNG: TMenuItem;

procedure FormCreate(Sender: TObject);

procedure MouseDown(Sender: TObject; Button:

TMouseButton;

Shift: TShiftState; X, Y: Integer);

procedure DblClick(Sender: TObject);

procedure AddStatement(Sender: TObject);

procedure ScrollBoxMouseWheel(Sender: TObject;

Shift: TShiftState;

WheelDelta: Integer; MousePos: TPoint; var Han

dled: Boolean);

procedure MICopyClick(Sender: TObject);

procedure MICutClick(Sender: TObject);

procedure MIInsetClick(Sender: TObject);

procedure DeleteStatement(Sender: TObject);

procedure Sort(Sender: TObject);

procedure PopupMenuPopup(Sender: TObject);

procedure MouseUp(Sender: TObject; Button: TMouse

Button;

Shift: TShiftState; X, Y: Integer);

procedure MouseMove(Sender: TObject; Shift:

TShiftState; X,

Y: Integer);

procedure actChangeActionExecute(Sender: TObject);

procedure FormKeyDown(Sender: TObject; var Key:

Word; Shift: TShiftState);

procedure PaintBoxPaint(Sender: TObject);

procedure actUndoExecute(Sender: TObject);

procedure actRedoExecute(Sender: TObject);

procedure actChngFontExecute(Sender: TObject);

procedure FormShortCut(var Msg: TWMKey; var Han-

dled: Boolean);

procedure FormKeyUp(Sender: TObject; var Key: Word;

Shift: TShiftState);

procedure actChngPenExecute(Sender: TObject);

procedure FormClose(Sender: TObject; var Action:

TCloseAction);

procedure actChngGlSettingsExecute(Sender:

TObject);

procedure mnDiagramClick(Sender: TObject);

procedure mnEditClick(Sender: TObject);

procedure actHelpExecute(Sender: TObject);

procedure actExitExecute(Sender: TObject);

procedure actSaveAsExecute(Sender: TObject);

procedure actSaveExecute(Sender: TObject);

procedure actOpenExecute(Sender: TObject);

procedure actNewExecute(Sender: TObject);

procedure actExportExecute(Sender: TObject);

private type

TFileMode = (fmJSON = 0, fmSvg, fmBmp, fmPng,

fmStat, fmAll);

private

FPenDialog: TPenDialog;

FGlobalSettingsDialog: TGlobalSettingsDialog;

FPrevMousePos: TPoint;

FisPressed: Boolean;

FMayDrag, FWasDbClick: Boolean;

FBlockManager: TBlockManager;

FUserInfo: TUserInfo;

function GetVisibleImageScreen: TVisibleImageRect;

procedure SetScrollPos(const AStatement: TState

ment);

function isDragging: Boolean; inline;

procedure UpdateForDedicatedStatement;

procedure UpdateForStack;

procedure SetOpenFileMode(const AMode: TFileMode);

procedure SetSaveFileMode(const AMode: TFileMode);

function HandleSaveSchemePrompt: Boolean;

public

destructor Destroy; override;

end;

var

NassiShneiderman: TNassiShneiderman;

implementation

{$R \*.dfm}

{ TNassiShneiderman }

procedure TNassiShneiderman.FormClose(Sender:

TObject; var Action: TCloseAction);

var

Answer: integer;

begin

// Set the logout time for the user

FUserInfo.LogoutTime := Now;

// Save global settings

SaveGlobalSettings;

// Save statistics for the current user

SaveStatistics(FUserInfo);

// Check if the block manager has unsaved changes

// or if the default main block is being used

if not (FBlockManager.isSaved or FBlockMan

ager.isDefaultMainBlock) then

begin

// Display a warning message box with options to

// save, discard, or cancel

Answer := MessageDlg(rsExitDlg, mtWarning,

[mbYes, mbNo, mbCancel], 0);

case Answer of

mrYes:

begin

// Set the save file mode to JSON

SetSaveFileMode(fmJSON);

// Show the save dialog to choose a file path

if SaveDialog.Execute then

begin

// Set the path to the chosen file for the

// block manager

FBlockManager.PathToFile := SaveDia

log.FileName;

// Save the schema using the block manager

SaveSchema(FBlockManager);

// Free the form and close it

Action := caFree;

end

else

// If the user cancels the save dialog, do

// not close the form

Action := caNone;

end;

mrNo:

// Discard changes and free the form to close

// it

Action := caFree;

mrCancel:

// Cancel the form close action and keep the

// form open

Action := caNone;

end;

end;

end;

procedure TNassiShneiderman.FormCreate(Sender:

TObject);

const

MinFormWidth = 850 + 42;

MinFormHeight = 550 + 42;

begin

// Clear user information

ClearUserInfo(FUserInfo);

// Set the login time to the current time

FUserInfo.LoginTime := Now;

// Get the Windows user name and assign it to the

// user information

FUserInfo.UserName := GetWindowsUserName;

// Set titles for save and open dialogs

SaveDialog.Title := 'Save As';

OpenDialog.Title := 'Open';

// Load global settings

LoadGlobalSettings;

// Set the default statement to TProcessStatement

DefaultStatement := TProcessStatement;

// Set the UI language to English (United States)

SetThreadUILanguage(MAKELANGID(LANG\_ENGLISH, SUB

LANG\_ENGLISH\_US));

// Enable double buffering for smoother drawing

Self.DoubleBuffered := True;

// Initialize variables for mouse interaction

FisPressed := False;

FMayDrag := False;

FWasDbClick := False;

// Set minimum form width and height

Constraints.MinWidth := MinFormWidth;

Constraints.MinHeight := MinFormHeight;

// Set shortcuts for actions

actDelete.ShortCut := ShortCut(VK\_DELETE, []);

actChangeAction.ShortCut := ShortCut(VK\_RETURN, []);

actUndo.ShortCut := ShortCut(VK\_Z, [ssCtrl]);

actRedo.ShortCut := ShortCut(VK\_Z, [ssCtrl, ssShift]);

actChngFont.ShortCut := ShortCut(VK\_F, [ssShift, ssCtrl]);

actChngPen.ShortCut := ShortCut(VK\_P, [ssShift, ssCtrl]);

actChngGlSettings.ShortCut := ShortCut(VK\_G, [ssCtrl, ssShift]);

actSortAsc.ShortCut := ShortCut(VK\_RIGHT, [ssCtrl, ssShift]);

actSortDesc.ShortCut := ShortCut(VK\_LEFT, [ssCtrl, ssShift]);

// Create instances of dialog forms

FPenDialog := TPenDialog.Create(Self, ColorDialog);

FGlobalSettingsDialog := TGlobalSettingsDialog.Create(Self, ColorDialog);

// Create a buffer block and assign it to the block manager

TBlockManager.BufferBlock := TBlock.Create(0, PaintBox.Canvas);

TBlockManager.BufferBlock.AddStatement(uBase.DefaultStatement.Create(

DefaultAction, TBlockManager.BufferBlock));

// Create an instance of the block manager and initialize the main block

FBlockManager := TBlockManager.Create(PaintBox);

FBlockManager.InitializeMainBlock;

end;

procedure TNassiShneiderman.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);

begin

// Try to move the dedicated block based on the scroll position and the pressed key

FBlockManager.TryMoveDedicated(SetScrollPos, Key);

// Update the UI for the dedicated statement

UpdateForDedicatedStatement;

end;

procedure TNassiShneiderman.FormKeyUp(Sender: TObject; var Key: Word; Shift: TShiftState);

begin

// Set the flag indicating that no key is pressed

FisPressed := False;

end;

procedure TNassiShneiderman.actExitExecute(Sender: TObject);

begin

// Close the form

Close;

end;

procedure TNassiShneiderman.FormShortCut(var Msg: TWMKey; var Handled: Boolean);

begin

// If dragging is in progress, destroy the carry block

if isDragging then

FBlockManager.DestroyCarryBlock;

// Check if a key is already pressed

if FisPressed then

Handled := True

else if GetKeyState(VK\_RETURN) < 0 then

FisPressed := True

else

begin

// Handle specific key combinations

case Msg.CharCode of

VK\_Z, VK\_X, VK\_C, VK\_V:

FisPressed := True;

VK\_RIGHT, VK\_LEFT:

begin

FisPressed := True;

// Trigger FormKeyDown for specific keys only if CTRL or SHIFT is not pressed

if (GetKeyState(VK\_CONTROL) >= 0) or (GetKeyState(VK\_SHIFT) >= 0) then

FormKeyDown(nil, Msg.CharCode, []);

end;

end;

end;

end;

procedure TNassiShneiderman.PaintBoxPaint(Sender: TObject);

begin

// Draw the block manager content on the paint box

FBlockManager.Draw(GetVisibleImageScreen);

end;

procedure TNassiShneiderman.ScrollBoxMouseWheel(Sender: TObject;

Shift: TShiftState; WheelDelta: Integer; MousePos: TPoint;

var Handled: Boolean);

const

ScrollStep = 42 shl 1;

begin

// Check if Shift key is pressed

if ssShift in Shift then

begin

// Scroll horizontally based on the WheelDelta value

if WheelDelta > 0 then

ScrollBox.HorzScrollBar.Position := ScrollBox.HorzScrollBar.Position - ScrollStep

else

ScrollBox.HorzScrollBar.Position := ScrollBox.HorzScrollBar.Position + ScrollStep;

end

else

begin

// Scroll vertically based on the WheelDelta value

if WheelDelta > 0 then

ScrollBox.VertScrollBar.Position := ScrollBox.VertScrollBar.Position - ScrollStep

else

ScrollBox.VertScrollBar.Position := ScrollBox.VertScrollBar.Position + ScrollStep;

end;

// Convert mouse position to client coordinates of PaintBox

MousePos := PaintBox.ScreenToClient(Mouse.CursorPos);

// Trigger MouseMove event with updated mouse position

MouseMove(Sender, Shift, MousePos.X, MousePos.Y);

Handled := True;

end;

procedure TNassiShneiderman.PopupMenuPopup(Sender: TObject);

var

bool : Boolean;

begin

// Check if the dedicated statement is a TCaseBranching

bool := FBlockManager.DedicatedStatement is TCaseBranching;

// Set the visibility of menu items based on the statement type

MIAscSort.Visible:= bool;

MIDescSort.Visible:= bool;

// Check if there are undo actions in the undo stack

bool := FBlockManager.UndoStack.Count <> 0;

// Enable/disable menu items based on the availability of undo actions

MIUndo.Enabled:= bool;

MIRedo.Enabled:= bool;

// Check if the dedicated statement is non-default

bool := not isDefaultStatement(FBlockManager.DedicatedStatement);

// Enable/disable menu items based on the dedicated statement type

MIDelete.Enabled := bool;

MICut.Enabled := bool;

MICopy.Enabled := bool;

end;

procedure TNassiShneiderman.DblClick(Sender: TObject);

begin

// Try to change the dedicated text on double-click

FBlockManager.TryChangeDedicatedText;

FWasDbClick:= True;

end;

procedure TNassiShneiderman.MouseDown(Sender: TObject;

Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

begin

// Set the DedicatedStatement based on the coordinates (X, Y) using BinarySearchStatement

FBlockManager.DedicatedStatement := BinarySearchStatement(X, Y, FBlockManager.MainBlock);

if FBlockManager.DedicatedStatement <> nil then

begin

case Button of

mbLeft:

begin

// Check if it's a left mouse button click

FMayDrag := not FWasDbClick;

FWasDbClick := False;

FPrevMousePos := Point(X, Y);

end;

mbRight:

begin

// Check if it's a right mouse button click

if isDragging then

FBlockManager.DestroyCarryBlock;

PopupMenu.Popup(Mouse.CursorPos.X, Mouse.CursorPos.Y);

end;

end;

end;

// Update the UI based on the DedicatedStatement

UpdateForDedicatedStatement;

UpdateForStack;

end;

procedure TNassiShneiderman.MouseMove(Sender: TObject; Shift: TShiftState;

X, Y: Integer);

const

AmountPixelToMove = 42;

begin

if isDragging then

begin

// If dragging is in progress, update the hover position and move the carry block

FBlockManager.DefineHover(X, Y);

FBlockManager.MoveCarryBlock(X - FPrevMousePos.X, Y - FPrevMousePos.Y);

FPrevMousePos := Point(X, Y);

end

else if FMayDrag and ((Abs(FPrevMousePos.X - X) > AmountPixelToMove) or

(Abs(FPrevMousePos.Y - Y) > AmountPixelToMove)) and

(FBlockManager.DedicatedStatement <> nil) then

begin

// If the mouse has moved a sufficient distance and dragging is allowed, create the carry block

FMayDrag := False;

if isDragging then

FBlockManager.DestroyCarryBlock;

FBlockManager.CreateCarryBlock;

end;

end;

procedure TNassiShneiderman.MouseUp(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

begin

// Mouse button is released

FMayDrag := False;

if isDragging then

begin

// If dragging was in progress, take action, destroy the carry block, and update the interface

FBlockManager.TryTakeAction;

FBlockManager.DestroyCarryBlock;

end;

end;

procedure TNassiShneiderman.MICopyClick(Sender: TObject);

begin

// Copy the dedicated statement

FBlockManager.TryCopyDedicated;

UpdateForStack;

end;

procedure TNassiShneiderman.MICutClick(Sender: TObject);

begin

// Cut the dedicated statement

FBlockManager.TryCutDedicated;

UpdateForStack;

UpdateForDedicatedStatement;

Inc(FUserInfo.DeleteStatementCount);

end;

procedure TNassiShneiderman.MIInsetClick(Sender: TObject);

begin

// Insert the buffer block

FBlockManager.TryInsertBufferBlock;

UpdateForStack;

UpdateForDedicatedStatement;

Inc(FUserInfo.AddStatementCount);

end;

procedure TNassiShneiderman.AddStatement(Sender: TObject);

var

Tag: Integer;

begin

// Extract the tag value from the sender component

Tag := TComponent(Sender).Tag;

// Try to add a new statement to the block manager

FBlockManager.TryAddNewStatement(ConvertToStatementType(Tag mod 5), (Tag div 5) = 0);

// Update the interface to reflect the changes in the stack and dedicated statement

UpdateForStack;

UpdateForDedicatedStatement;

Inc(FUserInfo.AddStatementCount);

end;

procedure TNassiShneiderman.Sort(Sender: TObject);

begin

// Try to sort the dedicated case based on the tag value of the sender component

FBlockManager.TrySortDedicatedCase(TComponent(Sender).Tag);

// Update the interface to reflect the changes in the stack

UpdateForStack;

end;

procedure TNassiShneiderman.actRedoExecute(Sender: TObject);

begin

// Try to redo the previous action in the block manager

FBlockManager.TryRedo;

// Update the interface to reflect the changes in the stack and dedicated statement

UpdateForStack;

UpdateForDedicatedStatement;

end;

procedure TNassiShneiderman.actNewExecute(Sender: TObject);

begin

// Check if the current scheme is saved or a default main block, or prompt for saving

if FBlockManager.isSaved or FBlockManager.isDefaultMainBlock or HandleSaveSchemePrompt then

begin

// Destroy the current block manager

FBlockManager.Destroy;

// Create a new block manager and initialize the main block

FBlockManager := TBlockManager.Create(PaintBox);

FBlockManager.InitializeMainBlock;

end;

end;

procedure TNassiShneiderman.actOpenExecute(Sender: TObject);

begin

// Check if the current scheme is saved or a default main block, or prompt for saving

if FBlockManager.isSaved or FBlockManager.isDefaultMainBlock or HandleSaveSchemePrompt then

begin

// Set the open file mode to JSON

SetOpenFileMode(fmJSON);

// Execute the open dialog

if OpenDialog.Execute then

begin

// Set the path to the selected file in the block manager

FBlockManager.PathToFile := OpenDialog.FileName;

// Load the schema from the selected file

LoadSchema(FBlockManager);

end;

end;

end;

procedure TNassiShneiderman.actExportExecute(Sender: TObject);

var

FileMode: TFileMode;

begin

// Get the file mode from the tag value of the sender component

FileMode := TFileMode(TComponent(Sender).Tag);

// Set the save file mode based on the file mode

SetSaveFileMode(FileMode);

// Execute the save dialog

if SaveDialog.Execute then

begin

// Save the block manager's content to the selected file based on the file mode

case FileMode of

fmSVG: SaveSVGFile(FBlockManager, SaveDialog.FileName);

fmBMP: SaveBMPFile(FBlockManager, SaveDialog.FileName);

fmPNG: SavePNGFile(FBlockManager, SaveDialog.FileName);

end;

end;

end;

procedure TNassiShneiderman.actSaveAsExecute(Sender: TObject);

var

FileName: string;

FileExt: string;

begin

// Set the save file mode to all

SetSaveFileMode(fmAll);

// Execute the save dialog

if SaveDialog.Execute then

begin

FileName := SaveDialog.FileName;

FileExt := LowerCase(ExtractFileExt(FileName));

if FileExt = constExtJSON then

begin

// Set the path to the selected file in the block manager

FBlockManager.PathToFile := FileName;

// Save the schema to the selected file

SaveSchema(FBlockManager);

end

else if FileExt = constExtSVG then

SaveSVGFile(FBlockManager, FileName)

else if FileExt = constExtBmp then

SaveBMPFile(FBlockManager, FileName)

else if FileExt = constExtPNG then

SavePNGFile(FBlockManager, FileName);

end;

// Update the enabled state of the "Save As" toolbar button based on the block manager's save state

tbSaveAs.Enabled := not FBlockManager.isSaved;

end;

procedure TNassiShneiderman.actSaveExecute(Sender: TObject);

begin

if FBlockManager.PathToFile <> '' then

begin

// Save the schema to the current file

SaveSchema(FBlockManager);

end

else

actSaveAsExecute(Sender);

// Update the enabled state of the "Save As" toolbar button based on the block manager's save state

tbSaveAs.Enabled := not FBlockManager.isSaved;

end;

procedure TNassiShneiderman.actHelpExecute(Sender: TObject);

var

StartTime: TDateTime;

begin

StartTime := Now;

// Retrieve the tag value from the sender component

case TComponent(Sender).Tag of

0: Help.Execute(rsUseGuide);

1: Help.Execute(rsAbout);

2: ShowMessage(FormatStatistics(Self.FUserInfo));

3:

begin

SetOpenFileMode(fmStat);

var FilePath: string := IncludeTrailingPathDelimiter(ExtractFilePath(ParamStr(0))) + dirAppData;

if DirectoryExists(FilePath) then

OpenDialog.InitialDir := FilePath;

if OpenDialog.Execute then

ShowMessage(FormatStatistics(LoadStatistics(OpenDialog.FileName)));

OpenDialog.InitialDir := '';

end;

end;

// Update the help time by calculating the time difference between the current time and the start time

Inc(FUserInfo.HelpTime, SecondsBetween(Now, StartTime));

end;

procedure TNassiShneiderman.actUndoExecute(Sender: TObject);

begin

// Try to undo the previous action in the block manager

FBlockManager.TryUndo;

// Update the display for the block stack and the dedicated statement

UpdateForStack;

UpdateForDedicatedStatement;

end;

procedure TNassiShneiderman.DeleteStatement(Sender: TObject);

begin

// Try to delete the dedicated statement in the block manager

FBlockManager.TryDeleteDedicated;

// Update the display for the block stack and the dedicated statement

UpdateForStack;

UpdateForDedicatedStatement;

Inc(FUserInfo.DeleteStatementCount);

end;

procedure TNassiShneiderman.actChangeActionExecute(Sender: TObject);

begin

// Try to change the text of the dedicated statement in the block manager

FBlockManager.TryChangeDedicatedText;

// Update the display for the block stack

UpdateForStack;

Inc(FUserInfo.ChangeActionCount);

end;

procedure TNassiShneiderman.actChngFontExecute(Sender: TObject);

var

StartTime: TDateTime;

begin

StartTime := Now;

// Initialize the font dialog with the current font settings

FontDialog.Font := FBlockManager.Font;

// Prompt the user to select a new font using the font dialog

if FontDialog.Execute then

begin

// Update the font settings in the block manager with the selected font

FBlockManager.Font.Assign(FontDialog.Font);

// Redefine the main block to apply the new font settings

FBlockManager.RedefineMainBlock;

end;

// Update the font setting time by calculating the time difference between the current time and the start time

Inc(FUserInfo.FontSettingTime, SecondsBetween(Now, StartTime));

end;

procedure TNassiShneiderman.actChngPenExecute(Sender: TObject);

var

StartTime: TDateTime;

begin

StartTime := Now;

// Initialize the pen dialog with the current pen settings

FPenDialog.Pen := FBlockManager.Pen;

// Prompt the user to select new pen settings using the pen dialog

if FPenDialog.Execute then

begin

// Update the pen settings in the block manager with the selected pen

FBlockManager.RedefineMainBlock;

end;

// Update the pen setting time by calculating the time difference between the current time and the start time

Inc(FUserInfo.PenSettingTime, SecondsBetween(Now, StartTime));

end;

procedure TNassiShneiderman.actChngGlSettingsExecute(Sender: TObject);

var

PrevDefaultAction: string;

StartTime: TDateTime;

begin

StartTime := Now;

// Store the previous default action

PrevDefaultAction := DefaultAction;

// Prompt the user to change global settings using the global settings dialog

if FGlobalSettingsDialog.Execute then

begin

// Apply the changes in global settings to the block manager

FBlockManager.ChangeGlobalSettings(PrevDefaultAction);

end;

// Update the global settings time by calculating the time difference between the current time and the start time

Inc(FUserInfo.GlobalSettingsTime, SecondsBetween(Now, StartTime));

end;

procedure TNassiShneiderman.mnDiagramClick(Sender: TObject);

var

bool: Boolean;

begin

bool:= FBlockManager.DedicatedStatement is TCaseBranching;

mnSortAsc.Enabled := bool;

mnSortDesc.Enabled := bool;

bool := FBlockManager.DedicatedStatement <> nil;

mnAdd.Enabled := bool;

mnChangeAct.Enabled := bool;

mnDelete.Enabled := bool and not isDefaultStatement(FBlockManager.DedicatedStatement);

end;

procedure TNassiShneiderman.mnEditClick(Sender: TObject);

var

bool: Boolean;

begin

mnUndo.Enabled:= FBlockManager.UndoStack.Count <> 0;

mnRedo.Enabled:= FBlockManager.RedoStack.Count <> 0;

bool := FBlockManager.DedicatedStatement <> nil;

mnInsert.Enabled := bool;

bool := bool and not isDefaultStatement(FBlockManager.DedicatedStatement);

mnCut.Enabled := bool;

mnCopy.Enabled := bool;

end;

{ Private methods }

destructor TNassiShneiderman.Destroy;

begin

TBlockManager.BufferBlock.Destroy;

if TBlockManager.CarryBlock <> nil then

TBlockManager.CarryBlock.Destroy;

FBlockManager.Destroy;

FPenDialog.Destroy;

FGlobalSettingsDialog.Destroy;

inherited;

end;

function TNassiShneiderman.HandleSaveSchemePrompt: Boolean;

var

Answer: Integer;

begin

// Prompt the user with a message dialog and store the user's answer

Answer := MessageDlg(rsExitDlg, mtWarning, [mbYes, mbNo, mbCancel], 0);

// Handle the user's answer

case Answer of

mrYes:

begin

// Set the save file mode to JSON

SetSaveFileMode(fmJSON);

// Prompt the user to select a file for saving the schema

if SaveDialog.Execute then

begin

// Update the path to the file in the block manager

FBlockManager.PathToFile := SaveDialog.FileName;

// Save the schema using the block manager

SaveSchema(FBlockManager);

// Return True indicating successful saving

Result := True;

end

else

// Return False indicating saving was canceled

Result := False;

end;

mrNo:

// Return True indicating no saving is required

Result := True;

mrCancel:

// Return False indicating canceling the operation

Result := False;

end;

end;

procedure TNassiShneiderman.UpdateForStack;

begin

tbUndo.Enabled:= FBlockManager.UndoStack.Count <> 0;

tbRedo.Enabled:= FBlockManager.RedoStack.Count <> 0;

end;

procedure TNassiShneiderman.SetOpenFileMode(const AMode: TFileMode);

begin

OpenDialog.FileName := '';

case AMode of

fmJSON:

begin

OpenDialog.DefaultExt := constExtJSON;

OpenDialog.Filter := rsFMJSON;

end;

fmStat:

begin

OpenDialog.DefaultExt := constExtStat;

OpenDialog.Filter := rsFMStat;

end;

end;

end;

procedure TNassiShneiderman.SetSaveFileMode(const AMode: TFileMode);

begin

SaveDialog.FileName := '';

case AMode of

fmJSON:

begin

SaveDialog.DefaultExt := constExtJSON;

SaveDialog.Filter := rsFMJSON;

end;

fmSvg:

begin

SaveDialog.DefaultExt := constExtSVG;

SaveDialog.Filter := rsFMSVG;

end;

fmBmp:

begin

SaveDialog.DefaultExt := constExtBmp;

SaveDialog.Filter := rsFMBmp;

end;

fmPng:

begin

SaveDialog.DefaultExt := constExtPng;

SaveDialog.Filter := rsFMPng;

end;

fmAll:

begin

SaveDialog.DefaultExt := constExtJSON;

SaveDialog.Filter := rsFMJSON + '|' + rsFMSVG + '|' + rsFMBmp + '|' + rsFMPng + '|' + rsFMAll;

end;

end;

end;

procedure TNassiShneiderman.UpdateForDedicatedStatement;

var

bool: Boolean;

begin

bool:= FBlockManager.DedicatedStatement is TCaseBranching;

tbSortDesc.Enabled := bool;

tbSortAsc.Enabled := bool;

bool := FBlockManager.DedicatedStatement <> nil;

tbInsert.Enabled := bool;

tbAction.Enabled := bool;

tbProcess.Enabled := bool;

tbIfBranch.Enabled := bool;

tbMultBranch.Enabled := bool;

tbLoop.Enabled := bool;

tbRevLoop.Enabled := bool;

bool := bool and not isDefaultStatement(FBlockManager.DedicatedStatement);

tbCut.Enabled := bool;

tbCopy.Enabled := bool;

tbDelete.Enabled := bool;

tbSaveAs.Enabled := not FBlockManager.isSaved;

end;

function TNassiShneiderman.isDragging: Boolean;

begin

Result:= TBlockManager.CarryBlock <> nil;

end;

function TNassiShneiderman.GetVisibleImageScreen: TVisibleImageRect;

begin

Result.FTopLeft := PaintBox.ScreenToClient(ScrollBox.ClientToScreen(Point(0, 0)));

Result.FBottomRight := PaintBox.ScreenToClient(

ScrollBox.ClientToScreen(Point(ScrollBox.Width, ScrollBox.Height)));

end;

procedure TNassiShneiderman.SetScrollPos(const AStatement: TStatement);

const

Stock = 42;

var

VisibleImageScreen: TVisibleImageRect;

begin

VisibleImageScreen:= GetVisibleImageScreen;

case AStatement.BaseBlock.GetMask(VisibleImageScreen) of

$09 {1001}:

ScrollBox.HorzScrollBar.Position:= ScrollBox.HorzScrollBar.Position +

AStatement.BaseBlock.XLast - VisibleImageScreen.FBottomRight.X + Stock;

$06 {1100}:

ScrollBox.HorzScrollBar.Position:= AStatement.BaseBlock.XStart - Stock;

end;

case AStatement.GetMask(VisibleImageScreen, AStatement is TOperator) of

$09 {1001}:

ScrollBox.VertScrollBar.Position := ScrollBox.VertScrollBar.Position +

AStatement.GetYBottom - VisibleImageScreen.FBottomRight.Y + Stock;

$06 {1100}:

ScrollBox.VertScrollBar.Position := AStatement.YStart - Stock;

end;

end;

end.

Приложение Б

(обязательное)

Исходный код программы (модуль frmGetAction)

unit frmGetAction;

interface

uses

Winapi.Windows, System.Classes, uConstants,

Vcl.Controls, Vcl.Forms, Vcl.StdCtrls, Vcl.ExtCtrls;

type

TWriteAction = class(TForm)

MemoAction: TMemo;

btnOK: TButton;

btnCancel: TButton;

procedure MemoActionKeyDown(Sender: TObject; var Key: Word;

Shift: TShiftState);

procedure FormShow(Sender: TObject);

procedure FormCreate(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

function TryGetAction(var AAction: String): Boolean;

end;

var

WriteAction: TWriteAction;

implementation

{$R \*.dfm}

function TWriteAction.TryGetAction(var AAction: String): Boolean;

begin

MemoAction.Text := AAction;

MemoAction.SelStart := 0;

MemoAction.SelLength := Length(MemoAction.Text);

ShowModal;

if Self.ModalResult = MrOk then

begin

Result:= True;

AAction:= MemoAction.Lines.Text;

end

else

Result:= False;

end;

procedure TWriteAction.FormCreate(Sender: TObject);

begin

MemoAction.MaxLength := MaxTextLength;

MemoAction.Font.Size := mmFontSize;

MemoAction.Font.Name := mmFontName;

end;

procedure TWriteAction.FormShow(Sender: TObject);

begin

Left := (Screen.Width - Width) shr 1;

Top := (Screen.Height - Height) shr 1;

MemoAction.SetFocus;

end;

procedure TWriteAction.MemoActionKeyDown(Sender: TObject; var Key: Word;

Shift: TShiftState);

begin

if Key = VK\_ESCAPE then

ModalResult := mrCancel

else if (Key = VK\_RETURN) and not (ssShift in Shift) then

ModalResult := mrOk;

end;

end.

Приложение В

(обязательное)

Исходный код программы (модуль frmGetСaseСonditions)

unit frmGetСaseСonditions;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, uAdditionalTypes, uStack, uConstants,

Vcl.ExtCtrls, uMinMaxInt, Vcl.DBCtrls;

type

TWriteCaseConditions = class(TForm)

btnOK: TButton;

lbAdd: TLabel;

btnAdd: TButton;

lbDel: TLabel;

btnDelete: TButton;

btnCancel: TButton;

MainPanel: TPanel;

ScrollBar: TScrollBar;

procedure FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);

procedure FormCreate(Sender: TObject);

procedure btnAddClick(Sender: TObject);

procedure btnDeleteClick(Sender: TObject);

procedure FormShow(Sender: TObject);

procedure ScrollBarScroll(Sender: TObject; ScrollCode: TScrollCode;

var ScrollPos: Integer);

procedure FormMouseWheel(Sender: TObject; Shift: TShiftState;

WheelDelta: Integer; MousePos: TPoint; var Handled: Boolean);

procedure Click(Sender: TObject);

private const

constMinAmount = 2;

constMaxAmount = 442;

constMemoAmount = 4;

constMemoHigh = constMemoAmount - 1;

constLabelCaption = 'Condition ';

private type

TCondSet = record

LabelCaption : TLabel;

Memo : TMemo;

end;

private

{ Private declarations }

FPointer, FHigh: Integer;

FConds : TStringArr;

FCondsSet : array[0..constMemoHigh] of TCondSet;

procedure SetCondSetVisible(const ACondSetIndex: Integer; const AVisible: Boolean);

procedure RefreshCondSet(const AIndex: Integer);

procedure ScrollUp;

procedure ScrollDown;

procedure SetScrollPos(const ANewPointer: Integer);

procedure SaveCurrentCombination;

public

{ Public declarations }

function TryGetCond(var AConds: TStringArr): Boolean;

destructor Destroy; override;

end;

var

WriteCaseConditions: TWriteCaseConditions;

implementation

{$R \*.dfm}

procedure TWriteCaseConditions.SetCondSetVisible(const ACondSetIndex: Integer; const AVisible: Boolean);

begin

FCondsSet[ACondSetIndex].LabelCaption.Visible := AVisible;

FCondsSet[ACondSetIndex].Memo.Visible := AVisible;

end;

procedure TWriteCaseConditions.RefreshCondSet(const AIndex: Integer);

begin

FCondsSet[AIndex].Memo.Lines.Text := FConds[AIndex];

FCondsSet[AIndex].LabelCaption.Caption := constLabelCaption + IntToStr(AIndex);

end;

procedure TWriteCaseConditions.SetScrollPos(const ANewPointer: Integer);

var

I, J: Integer;

begin

SaveCurrentCombination;

FPointer := ANewPointer;

J := Low(FCondsSet);

for I := FPointer to FPointer + constMemoHigh do

begin

FCondsSet[J].Memo.Lines.Text := FConds[I];

FCondsSet[J].LabelCaption.Caption := constLabelCaption + IntToStr(I);

Inc(J);

end;

end;

procedure TWriteCaseConditions.ScrollDown;

var

I: Integer;

begin

FConds[FPointer] := FCondsSet[Low(FCondsSet)].Memo.Lines.Text;

Inc(FPointer);

for I := Low(FCondsSet) to constMemoHigh - 1 do

begin

FCondsSet[I].Memo.Lines.Text := FCondsSet[I + 1].Memo.Lines.Text;

FCondsSet[I].LabelCaption.Caption := constLabelCaption + IntToStr(FPointer + I);

end;

FCondsSet[constMemoHigh].Memo.Lines.Text := FConds[FPointer + constMemoHigh];

FCondsSet[constMemoHigh].LabelCaption.Caption := constLabelCaption + IntToStr(FPointer + constMemoHigh);

end;

procedure TWriteCaseConditions.ScrollUp;

var

I: Integer;

begin

FConds[FPointer + constMemoHigh] := FCondsSet[constMemoHigh].Memo.Lines.Text;

Dec(FPointer);

for I := constMemoHigh downto Low(FCondsSet) + 1 do

begin

FCondsSet[I].Memo.Lines.Text := FCondsSet[I - 1].Memo.Lines.Text;

FCondsSet[I].LabelCaption.Caption := constLabelCaption + IntToStr(FPointer + I);

end;

FCondsSet[Low(FCondsSet)].Memo.Lines.Text := FConds[FPointer];

FCondsSet[Low(FCondsSet)].LabelCaption.Caption := constLabelCaption + IntToStr(FPointer);

end;

procedure TWriteCaseConditions.ScrollBarScroll(Sender: TObject;

ScrollCode: TScrollCode; var ScrollPos: Integer);

begin

case ScrollCode of

scLineUp, scPageUp:

if FPointer <> Low(FCondsSet) then

ScrollUp;

scLineDown, scPageDown:

if FPointer <> FHigh - constMemoHigh then

ScrollDown;

scPosition, scTrack:

SetScrollPos(ScrollPos);

scTop:

SetScrollPos(Low(FCondsSet));

scBottom:

SetScrollPos(FHigh - constMemoHigh);

end;

end;

function TWriteCaseConditions.TryGetCond(var AConds: TStringArr): Boolean;

var

I, MinHigh: Integer;

begin

FPointer:= Low(FCondsSet);

ScrollBar.Position := FPointer;

if AConds = nil then

begin

ScrollBar.Enabled := False;

FHigh := constMinAmount - 1;

SetLength(FConds, FHigh shl 2);

for I := Low(FCondsSet) to FHigh do

begin

SetCondSetVisible(I, True);

FCondsSet[I].Memo.Lines.Text := '';

FCondsSet[I].LabelCaption.Caption := constLabelCaption + IntToStr(I);

end;

for I := FHigh + 1 to constMemoHigh do

SetCondSetVisible(I, False);

end

else

begin

FHigh := High(AConds);

if High(AConds) > constMemoHigh then

ScrollBar.Enabled := True

else

ScrollBar.Enabled := False;

SetLength(FConds, Length(AConds) shl 1);

for I := 0 to High(AConds) do

FConds[I] := AConds[I];

MinHigh := Min(FHigh, constMemoHigh);

for I := Low(FCondsSet) to MinHigh do

begin

SetCondSetVisible(I, True);

RefreshCondSet(I);

end;

for I := MinHigh + 1 to constMemoHigh do

SetCondSetVisible(I, False);

end;

ShowModal;

if Self.ModalResult = MrOk then

begin

Result:= True;

SetLength(AConds, FHigh + 1);

SaveCurrentCombination;

for I := 0 to FHigh do

AConds[I] := FConds[I];

end

else

Result:= False;

SetLength(FConds, 0);

end;

procedure TWriteCaseConditions.SaveCurrentCombination;

var

I: Integer;

begin

for I := Low(FCondsSet) to constMemoHigh do

FConds[FPointer + I] := FCondsSet[I].Memo.Lines.Text;

end;

procedure TWriteCaseConditions.FormKeyDown(Sender: TObject; var Key: Word;

Shift: TShiftState);

begin

if Key = VK\_ESCAPE then

ModalResult := mrCancel

else if (Key = VK\_RETURN) and not (ssShift in Shift) then

ModalResult := mrOk;

end;

procedure TWriteCaseConditions.FormMouseWheel(Sender: TObject;

Shift: TShiftState; WheelDelta: Integer; MousePos: TPoint;

var Handled: Boolean);

begin

if ScrollBar.Enabled and BtnOk.Focused then

begin

if (WheelDelta > 0) and (FPointer <> Low(FCondsSet)) then

begin

ScrollUp;

ScrollBar.Position := FPointer;

end

else if (WheelDelta < 0) and (FPointer <> FHigh - constMemoHigh) then

begin

ScrollDown;

ScrollBar.Position := FPointer;

end;

Handled := True;

end;

end;

procedure TWriteCaseConditions.btnAddClick(Sender: TObject);

begin

BtnOk.SetFocus;

if FHigh < constMaxAmount then

begin

Inc(FHigh);

if FHigh > High(FConds) then

SetLength(FConds, (FHigh + 1) shl 1);

if FHigh <= constMemoHigh then

begin

ScrollBar.Position := 0;

SetCondSetVisible(FHigh, True);

RefreshCondSet(FHigh);

end

else

begin

ScrollBar.Enabled := True;

ScrollBar.Max := FHigh - constMemoHigh;

end;

end;

end;

procedure TWriteCaseConditions.btnDeleteClick(Sender: TObject);

begin

BtnOk.SetFocus;

if FHigh >= constMinAmount then

begin

if FHigh <= constMemoAmount then

begin

ScrollBar.Position := 0;

ScrollBar.Enabled := False;

var I: Integer;

for I := Low(FCondsSet) to FHigh - 1 do

RefreshCondSet(I);

for I := FHigh to constMemoHigh do

SetCondSetVisible(I, False);

end

else if FPointer + constMemoHigh = FHigh then

begin

ScrollUp;

ScrollBar.Max := FHigh - constMemoAmount;

end

else

ScrollBar.Max := FHigh - constMemoAmount;

FConds[FHigh] := '';

Dec(FHigh);

end;

end;

procedure TWriteCaseConditions.FormCreate(Sender: TObject);

const

Indent = 5;

var

MemoHeight, FCondsSetWidth: Integer;

I, CurrPosY: Integer;

begin

MemoHeight := (MainPanel.Height - (mmFontSize + 3) \* constMemoAmount -

constMemoAmount \* Indent shl 1) div constMemoAmount - Indent shl 1;

FCondsSetWidth:= ScrollBar.Left - Indent shl 1;

CurrPosY:= Indent;

for I := Low(FCondsSet) to constMemoHigh do

begin

FCondsSet[I].LabelCaption := TLabel.Create(Self);

FCondsSet[I].Memo := TMemo.Create(Self);

with FCondsSet[I].LabelCaption do

begin

Parent := MainPanel;

Left := Indent;

Width := FCondsSetWidth;

Top := CurrPosY;

Font.Size := mmFontSize;

Font.Name := mmFontName;

end;

Inc(CurrPosY, FCondsSet[I].LabelCaption.Height + Indent);

with FCondsSet[I].Memo do

begin

Parent := MainPanel;

Left := Indent;

Width := FCondsSetWidth;

Top := CurrPosY;

Height := MemoHeight;

Font.Size := mmFontSize;

Font.Name := mmFontName;

MaxLength := MaxTextLength;

ScrollBars := ssBoth;

end;

Inc(CurrPosY, MemoHeight + Indent);

end;

end;

destructor TWriteCaseConditions.Destroy;

var

I: Integer;

begin

for I := Low(FCondsSet) to constMemoHigh do

begin

FCondsSet[I].LabelCaption.Destroy;

FCondsSet[I].Memo.Destroy;

end;

inherited;

end;

procedure TWriteCaseConditions.FormShow(Sender: TObject);

begin

Left := (Screen.Width - Width) shr 1;

Top := (Screen.Height - Height) shr 1;

end;

procedure TWriteCaseConditions.Click(Sender: TObject);

begin

BtnOk.SetFocus;

end;

end.

Приложение Г

(обязательное)

Исходный код программы (модуль frmGlobalSettings)

unit frmGlobalSettings;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes,

Vcl.Graphics, Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.ExtCtrls,

uIfBranching, uConstants, uBase, uBlockManager, UITypes, uGlobalSave;

type

TGlobalSettingsDialog = class(TForm)

btnOK: TButton;

btnCancel: TButton;

lbInfo: TLabel;

lbTrue: TLabel;

mmTrue: TMemo;

lbFalse: TLabel;

mmFalse: TMemo;

plIf: TPanel;

plDefault: TPanel;

lbDefAct: TLabel;

mmDefAct: TMemo;

plColors: TPanel;

lbColors: TLabel;

shpHighlight: TShape;

shpArrow: TShape;

shpOK: TShape;

shpCancel: TShape;

lnHighlightColor: TLabel;

lbArrow: TLabel;

lbOK: TLabel;

lbCancel: TLabel;

btnRestore: TButton;

procedure btnRestoreClick(Sender: TObject);

procedure FormShow(Sender: TObject);

procedure shpMouseDown(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

procedure KeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);

private

FColorDialog : TColorDialog;

procedure SetValues;

public

constructor Create(const AOwner: TComponent; const AColorDialog: TColorDialog);

function Execute: Boolean;

end;

implementation

{$R \*.dfm}

procedure TGlobalSettingsDialog.btnRestoreClick(Sender: TObject);

begin

ResetGlobalSettings;

SetValues;

btnOK.SetFocus;

end;

constructor TGlobalSettingsDialog.Create(const AOwner: TComponent; const AColorDialog: TColorDialog);

begin

inherited Create(AOwner);

FColorDialog := AColorDialog;

mmTrue.MaxLength := MaxTextLength;

mmTrue.Font.Size := mmFontSize;

mmTrue.Font.Name := mmFontName;

mmFalse.MaxLength := MaxTextLength;

mmFalse.Font.Size := mmFontSize;

mmFalse.Font.Name := mmFontName;

mmDefAct.MaxLength := MaxTextLength;

mmDefAct.Font.Size := mmFontSize;

mmDefAct.Font.Name := mmFontName;

end;

procedure TGlobalSettingsDialog.SetValues;

begin

mmTrue.Text := TIfBranching.TrueCond;

mmFalse.Text := TIfBranching.FalseCond;

mmDefAct.Text := DefaultAction;

with TBlockManager do

begin

shpHighlight.Brush.Color := HighlightColor;

shpArrow.Brush.Color := ArrowColor;

shpOK.Brush.Color := OKColor;

shpCancel.Brush.Color := CancelColor;

end;

end;

procedure TGlobalSettingsDialog.shpMouseDown(Sender: TObject;

Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

begin

if FColorDialog.Execute then

case TComponent(Sender).Tag of

0: shpHighlight.Brush.Color := FColorDialog.Color;

1: shpArrow.Brush.Color := FColorDialog.Color;

2: shpOK.Brush.Color := FColorDialog.Color;

3: shpCancel.Brush.Color := FColorDialog.Color;

end;

end;

function TGlobalSettingsDialog.Execute : Boolean;

begin

ShowModal;

if ModalResult = mrOk then

begin

Result:= True;

TIfBranching.TrueCond := mmTrue.Text;

TIfBranching.FalseCond := mmFalse.Text;

DefaultAction := mmDefAct.Text;

with TBlockManager do

begin

HighlightColor := shpHighlight.Brush.Color;

ArrowColor := shpArrow.Brush.Color;

OKColor := shpOK.Brush.Color;

CancelColor := shpCancel.Brush.Color;

end;

end

else

Result:= False;

end;

procedure TGlobalSettingsDialog.KeyDown(Sender: TObject; var Key: Word;

Shift: TShiftState);

begin

if Key = VK\_ESCAPE then

ModalResult := mrCancel

else if (Key = VK\_RETURN) and not (ssShift in Shift) then

ModalResult := mrOk;

end;

procedure TGlobalSettingsDialog.FormShow(Sender: TObject);

begin

Left := (Screen.Width - Width) shr 1;

Top := (Screen.Height - Height) shr 1;

SetValues;

end;

end.

Приложение Д

(обязательное)

Исходный код программы (модуль frmHelp)

unit frmHelp;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Menus, Vcl.OleCtrls, SHDocVw, ShellAPI, uConstants, System.IOUtils;

type

THelp = class(TForm)

WebBrowser: TWebBrowser;

pmHtmlMenu: TPopupMenu;

pmiClose: TMenuItem;

pmLicense: TMenuItem;

procedure pmiCloseClick(Sender: TObject);

procedure FormShow(Sender: TObject);

procedure FormCreate(Sender: TObject);

procedure WebBrowserBeforeNavigate2(ASender: TObject;

const pDisp: IDispatch; const URL, Flags, TargetFrameName, PostData,

Headers: OleVariant; var Cancel: WordBool);

procedure pmLicenseClick(Sender: TObject);

private const

MinFormWidth = 750;

MinFormHeight = 400;

private

procedure WMMouseActivate(var Msg: TMessage); message WM\_MOUSEACTIVATE;

public

procedure Execute(const AName: WideString);

end;

var

Help: THelp;

implementation

{$R \*.dfm}

procedure THelp.WebBrowserBeforeNavigate2(ASender: TObject;

const pDisp: IDispatch; const URL, Flags, TargetFrameName, PostData,

Headers: OleVariant; var Cancel: WordBool);

begin

if Pos('github.com', URL) > 0 then

begin

ShellExecuteW(Handle, 'open', PWideChar(WideString(URL)), nil, nil, SW\_SHOWNORMAL);

Cancel := True;

end;

end;

procedure THelp.WMMouseActivate(var Msg: TMessage);

begin

try

inherited;

if Msg.LParamHi = 516 then

pmHtmlMenu.Popup(Mouse.CursorPos.x, Mouse.CursorPos.y);

Msg.Result := 0;

except

end;

end;

procedure THelp.FormCreate(Sender: TObject);

begin

Constraints.MinWidth := MinFormWidth;

Constraints.MinHeight := MinFormHeight;

end;

procedure THelp.FormShow(Sender: TObject);

begin

WindowState := wsNormal;

Width := MinFormWidth;

Height := MinFormHeight;

Left := (Screen.Width - Width) shr 1;

Top := (Screen.Height - Height) shr 1;

end;

procedure THelp.pmiCloseClick(Sender: TObject);

begin

Close;

end;

procedure THelp.pmLicenseClick(Sender: TObject);

var

FilePath: string;

begin

FilePath := IncludeTrailingPathDelimiter(

ExtractFileDir(ExtractFileDir(ExtractFileDir(

IncludeTrailingPathDelimiter(

ExtractFileDir(ParamStr(0))))))) + PathToMITLicense;

if FileExists(FilePath) then

ShowMessage(TFile.ReadAllText(FilePath))

else

ShellExecuteW(Handle, 'open', PathToGitHubLicense, nil, nil, SW\_SHOWNORMAL);

end;

procedure THelp.Execute(const AName: WideString);

var

Flags, TargetFrameName, PostData, Headers: OleVariant;

begin

WebBrowser.Navigate('res://' + Application.ExeName + '/' + AName,

Flags, TargetFrameName, PostData, Headers);

ShowModal;

end;

end.

Приложение Е

(обязательное)

Исходный код программы (модуль frmPenSetting)

unit frmPenSetting;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.ExtCtrls, Types, System.UITypes;

type

TPenDialog = class(TForm)

btnOK: TButton;

btnCancel: TButton;

cbLineType: TComboBox;

CurrColor: TShape;

lbLineType: TLabel;

lbThickness: TLabel;

cbThickness: TComboBox;

lbColor: TLabel;

procedure FormShow(Sender: TObject);

procedure CurrColorMouseDown(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

procedure cbThicknessChange(Sender: TObject);

private

{ Private declarations }

FPen: TPen;

FColorDialog: TColorDialog;

class function GetIndexStyle(const AStyle: TPenStyle): Integer;

class function GetStyle(const AIndex: Integer): TPenStyle;

public

{ Public declarations }

property Pen: TPen write FPen;

constructor Create(const AOwner: TComponent; const AColorDialog: TColorDialog);

function Execute: Boolean;

end;

implementation

{$R \*.dfm}

class function TPenDialog.GetIndexStyle(const AStyle: TPenStyle): Integer;

begin

case AStyle of

psSolid: Result:= 0;

psDash: Result:= 1;

psDot: Result:= 2;

psDashDot: Result:= 3;

psDashDotDot: Result:= 4;

end;

end;

class function TPenDialog.GetStyle(const AIndex: Integer): TPenStyle;

begin

case AIndex of

0: Result:= psSolid;

1: Result:= psDash;

2: Result:= psDot;

3: Result:= psDashDot;

4: Result:= psDashDotDot;

end;

end;

constructor TPenDialog.Create(const AOwner: TComponent; const AColorDialog: TColorDialog);

begin

inherited Create(AOwner);

FColorDialog:= AColorDialog;

end;

procedure TPenDialog.FormShow(Sender: TObject);

begin

Left := (Screen.Width - Width) shr 1;

Top := (Screen.Height - Height) shr 1;

CurrColor.Brush.Color:= FPen.Color;

cbThickness.ItemIndex := FPen.Width - 1;

cbLineType.Enabled:= cbThickness.ItemIndex = 0;

if cbLineType.Enabled then

cbLineType.ItemIndex := GetIndexStyle(FPen.Style)

else

cbLineType.ItemIndex := 0;

end;

procedure TPenDialog.cbThicknessChange(Sender: TObject);

begin

cbLineType.Enabled:= cbThickness.ItemIndex = 0;

if not cbLineType.Enabled then

cbLineType.ItemIndex := 0;

end;

procedure TPenDialog.CurrColorMouseDown(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

begin

if FColorDialog.Execute then

CurrColor.Brush.Color:= FColorDialog.Color;

end;

function TPenDialog.Execute: Boolean;

begin

ShowModal;

if ModalResult = mrOk then

begin

Result:= True;

FPen.Color:= FColorDialog.Color;

FPen.Style:= GetStyle(cbLineType.ItemIndex);

FPen.Width:= cbThickness.ItemIndex + 1;

end

else

Result:= False;

end;

end.

Приложение Ж

(обязательное)

Исходный код программы (модуль uBlockManager)

unit uBlockManager;

interface

uses

uBase, uCommands, uAutoClearStack, Vcl.ExtCtrls, uSwitchStatements,

Winapi.Windows, uAdditionalTypes, uDrawShapes, Vcl.Graphics, frmGetAction,

frmGetСaseСonditions, uCaseBranching, uMinMaxInt, uStatementSearch, Types,

uIfBranching, uConstants;

type

TBlockManager = class

private type

TSetScrollPosProc = procedure(const AStatement: TStatement) of object;

THoveredStatement = record

private type

TState = (stBefore = 0, stAfter = 1, stSwap, stCancel);

public

Statement: TStatement;

Rect: TRect;

State: TState;

end;

private const

SchemeInitialFontSize = 13;

SchemeInitialFont = 'Courier new';

SchemeInitialFontColor: TColor = clBlack;

SchemeInitialFontStyles: TFontStyles = [];

SchemeInitialPenColor: TColor = clBlack;

SchemeInitialPenWidth = 1;

SchemeInitialPenStyle: TPenStyle = psSolid;

SchemeInitialPenMode: TPenMode = pmCopy;

private class var

FBufferBlock: TBlock;

FCarryBlock: TBlock;

FHoveredStatement: THoveredStatement;

FHighlightColor, FArrowColor, FOKColor, FCancelColor: TColor;

private

FMainBlock : TBlock;

FDedicatedStatement: TStatement;

FPaintBox: TPaintBox;

FUndoStack, FRedoStack: TAutoClearStack<ICommand>;

FPen: TPen;

FFont: TFont;

FPathToFile: string;

FisSaved: Boolean;

procedure AddToUndoStack(ACommand: ICommand);

procedure ChangeDedicated(const AStatement: TStatement);

procedure ChangeMainBlock(const ANewBlock: TBlock);

procedure SetPathToFile(const APath: string);

public

constructor Create(const APaintBox: TPaintBox);

destructor Destroy;

property MainBlock: TBlock read FMainBlock write ChangeMainBlock;

property DedicatedStatement: TStatement read FDedicatedStatement write ChangeDedicated;

property UndoStack: TAutoClearStack<ICommand> read FUndoStack;

property RedoStack: TAutoClearStack<ICommand> read FRedoStack;

property Font: TFont read FFont;

property Pen: TPen read FPen;

property PaintBox: TPaintBox read FPaintBox;

property PathToFile: string read FPathToFile write SetPathToFile;

property isSaved: Boolean read FisSaved;

class property CarryBlock: TBlock read FCarryBlock;

class property BufferBlock: TBlock read FBufferBlock write FBufferBlock;

class property HighlightColor: TColor read FHighlightColor write FHighlightColor;

class property ArrowColor: TColor read FArrowColor write FArrowColor;

class property OKColor: TColor read FOKColor write FOKColor;

class property CancelColor: TColor read FCancelColor write FCancelColor;

{ MainBlock }

procedure RedefineMainBlock;

procedure ChangeGlobalSettings(const AOldDefaultAction: string);

procedure InitializeMainBlock;

function isDefaultMainBlock: Boolean;

{ BufferBlock }

procedure TryCutDedicated;

procedure TryCopyDedicated;

procedure TryDeleteDedicated;

procedure TryInsertBufferBlock;

{ DedicatedStatement }

procedure TryMoveDedicated(const ASetScrollPosProc: TSetScrollPosProc; const AKey: Integer);

procedure TryChangeDedicatedText;

procedure TryAddNewStatement(const AStatementClass: TStatementClass;

const isAfterDedicated: Boolean);

procedure TrySortDedicatedCase(const ASortNumber: Integer);

{ CarryBlock }

procedure CreateCarryBlock;

procedure MoveCarryBlock(const ADeltaX, ADeltaY: Integer);

procedure DefineHover(const AX, AY: Integer);

procedure TryDrawCarryBlock(const AVisibleImageRect: TVisibleImageRect); inline;

procedure TryTakeAction;

procedure DestroyCarryBlock;

{ Interactions with statements }

class function CreateStatement(const AStatementClass: TStatementClass;

const ABaseBlock: TBlock): TStatement; static;

{ Stacks }

procedure TryUndo;

procedure TryRedo;

{ View update }

procedure Draw(const AVisibleImageRect: TVisibleImageRect);

end;

implementation

procedure TBlockManager.SetPathToFile(const APath: string);

begin

FisSaved:= True;

FPathToFile:= APath;

end;

procedure TBlockManager.AddToUndoStack(ACommand: ICommand);

begin

FisSaved:= False;

FRedoStack.Clear;

FUndoStack.Push(ACommand);

FUndoStack.Peek.Execute;

end;

destructor TBlockManager.Destroy;

begin

FPen.Destroy;

FFont.Destroy;

FUndoStack.Destroy;

FRedoStack.Destroy;

FMainBlock.Destroy;

inherited;

end;

constructor TBlockManager.Create(const APaintBox: TPaintBox);

begin

FPaintBox:= APaintBox;

FPen := TPen.Create;

FFont := TFont.Create;

FFont.Size := SchemeInitialFontSize;

FFont.Name := SchemeInitialFont;

FFont.Color := SchemeInitialFontColor;

FFont.Style := SchemeInitialFontStyles;

FPen.Color := SchemeInitialPenColor;

FPen.Width := SchemeInitialPenWidth;

FPen.Style := SchemeInitialPenStyle;

FPen.Mode := SchemeInitialPenMode;

FPaintBox.Canvas.Font := FFont;

FPaintBox.Canvas.Pen := FPen;

FUndoStack := TAutoClearStack<ICommand>.Create;

FRedoStack := TAutoClearStack<ICommand>.Create;

FDedicatedStatement:= nil;

FCarryBlock:= nil;

FMainBlock:= nil;

PathToFile:= '';

FisSaved:= False;

FPaintBox.Invalidate;

end;

{ MainBlock }

procedure TBlockManager.RedefineMainBlock;

begin

FPaintBox.Canvas.Font:= FFont;

FPaintBox.Canvas.Pen:= FPen;

MainBlock.RedefineSizes;

FPaintBox.Invalidate;

end;

procedure TBlockManager.ChangeGlobalSettings(const AOldDefaultAction: string);

begin

if AOldDefaultAction <> DefaultAction then

begin

if (FDedicatedStatement is DefaultStatement) and

(FDedicatedStatement.Action = DefaultAction) then

FDedicatedStatement := nil;

MainBlock.SetNewActionForDefaultStatements(AOldDefaultAction);

end;

TIfBranching.RedefineSizesForIfBranching(MainBlock);

FPaintBox.Invalidate;

end;

procedure TBlockManager.ChangeMainBlock(const ANewBlock: TBlock);

begin

if FMainBlock <> nil then

FMainBlock.Destroy;

FMainBlock:= ANewBlock;

FDedicatedStatement := nil;

PaintBox.Invalidate;

end;

procedure TBlockManager.InitializeMainBlock;

begin

FMainBlock:= TBlock.Create(SchemeIndent, FPaintBox.Canvas);

FMainBlock.AddUnknownStatement(uBase.DefaultStatement.Create(DefaultAction, FMainBlock),

SchemeIndent);

end;

function TBlockManager.isDefaultMainBlock: Boolean;

begin

Result:= (FUndoStack.Count = 0) and (FRedoStack.Count = 0) and isDefaultStatement(FMainBlock.Statements[0]);

end;

{ BufferBlock }

procedure TBlockManager.TryCutDedicated;

begin

TryCopyDedicated;

TryDeleteDedicated;

end;

procedure TBlockManager.TryCopyDedicated;

begin

if (FDedicatedStatement <> nil) and not isDefaultStatement(FDedicatedStatement) then

begin

FBufferBlock.Destroy;

FBufferBlock := TBlock.Create(nil);

FBufferBlock.Assign(FDedicatedStatement.BaseBlock);

FBufferBlock.AddStatement(FDedicatedStatement.Clone);

end;

end;

procedure TBlockManager.TryDeleteDedicated;

begin

if (FDedicatedStatement <> nil) and not isDefaultStatement(FDedicatedStatement) then

begin

AddToUndoStack(TCommandDelStatement.Create(FDedicatedStatement));

FDedicatedStatement:= nil;

FPaintBox.Invalidate;

end;

end;

procedure TBlockManager.TryInsertBufferBlock;

var

Statement: TStatement;

I: Integer;

begin

if (FBufferBlock.Statements.Count <> 0) and (FDedicatedStatement <> nil) then

begin

for I := FBufferBlock.Statements.Count - 1 downto 0 do

if isDefaultStatement(FBufferBlock.Statements[I]) then

begin

FBufferBlock.ExtractStatementAt(I);

FBufferBlock.Install(I);

end;

if (FBufferBlock.Statements.Count <> 1) or not isDefaultStatement(FBufferBlock.Statements[0]) then

begin

Statement:= FDedicatedStatement;

FDedicatedStatement:= FBufferBlock.Statements.GetLast;

AddToUndoStack(TCommandAddBlock.Create(Statement.BaseBlock,

Statement.BaseBlock.FindStatementIndex(Statement.YStart) + 1,

FBufferBlock));

FBufferBlock := TBlock.Create(nil);

FBufferBlock.Assign(FDedicatedStatement.BaseBlock);

FBufferBlock.AddStatement(FDedicatedStatement.Clone);

FPaintBox.Invalidate;

end;

end;

end;

{ DedicatedStatement }

procedure TBlockManager.TryMoveDedicated(const ASetScrollPosProc: TSetScrollPosProc; const AKey: Integer);

begin

case AKey of

VK\_LEFT:

begin

SetHorizontalMovement(FDedicatedStatement, FMainBlock, uSwitchStatements.BackwardDir);

ASetScrollPosProc(FDedicatedStatement);

FPaintBox.Invalidate;

end;

VK\_RIGHT:

begin

SetHorizontalMovement(FDedicatedStatement, FMainBlock, uSwitchStatements.ForwardDir);

ASetScrollPosProc(FDedicatedStatement);

FPaintBox.Invalidate;

end;

VK\_UP:

begin

SetVerticalMovement(FDedicatedStatement, FMainBlock, uSwitchStatements.BackwardDir);

ASetScrollPosProc(FDedicatedStatement);

FPaintBox.Invalidate;

end;

VK\_DOWN:

begin

SetVerticalMovement(FDedicatedStatement, FMainBlock, uSwitchStatements.ForwardDir);

ASetScrollPosProc(FDedicatedStatement);

FPaintBox.Invalidate;

end;

end;

end;

procedure TBlockManager.TryAddNewStatement(const AStatementClass: TStatementClass;

const isAfterDedicated: Boolean);

var

NewStatement: TStatement;

Block: TBlock;

begin

if FDedicatedStatement <> nil then

begin

NewStatement:= CreateStatement(AStatementClass,

FDedicatedStatement.BaseBlock);

if (NewStatement <> nil) and not isDefaultStatement(NewStatement) then

begin

Block:= FDedicatedStatement.BaseBlock;

AddToUndoStack(TCommandAddStatement.Create(Block,

Block.FindStatementIndex(FDedicatedStatement.YStart) +

Ord(isAfterDedicated),

NewStatement));

FDedicatedStatement:= NewStatement;

end;

FPaintBox.Invalidate;

end;

end;

procedure TBlockManager.TryChangeDedicatedText;

var

Action: String;

begin

if FDedicatedStatement <> nil then

begin

Action := FDedicatedStatement.Action;

if FDedicatedStatement is TCaseBranching then

begin

var CaseBranching: TCaseBranching:= TCaseBranching(FDedicatedStatement);

var Cond: TStringArr:= CaseBranching.Conds;

if (WriteAction.TryGetAction(Action)) and (WriteCaseConditions.TryGetCond(Cond)) then

begin

AddToUndoStack(TCommnadChangeContent.Create(FDedicatedStatement, Action, Cond));

FPaintBox.Invalidate;

end;

end

else if WriteAction.TryGetAction(Action) then

begin

AddToUndoStack(TCommnadChangeContent.Create(FDedicatedStatement, Action, nil));

FPaintBox.Invalidate;

end;

end;

end;

procedure TBlockManager.TrySortDedicatedCase(const ASortNumber: Integer);

begin

if FDedicatedStatement is TCaseBranching then

begin

AddToUndoStack(TCommandCaseSort.Create(TCaseBranching(FDedicatedStatement),

ASortNumber));

FPaintBox.Invalidate;

end;

end;

procedure TBlockManager.ChangeDedicated(const AStatement: TStatement);

begin

FDedicatedStatement:= AStatement;

FPaintBox.Invalidate;

end;

{ CarryBlock }

procedure TBlockManager.CreateCarryBlock;

begin

FCarryBlock:= TBlock.Create(nil);

FCarryBlock.Assign(FDedicatedStatement.BaseBlock);

FCarryBlock.AddStatement(FDedicatedStatement.Clone);

end;

procedure TBlockManager.MoveCarryBlock(const ADeltaX, ADeltaY: Integer);

begin

FCarryBlock.MoveRight(ADeltaX);

FCarryBlock.MoveDown(ADeltaY);

FPaintBox.Invalidate;

end;

procedure TBlockManager.DefineHover(const AX, AY: Integer);

const

Indent = 5;

begin

FHoveredStatement.Statement:= nil;

FHoveredStatement.State := stCancel;

if FDedicatedStatement is TOperator then

begin

var Block: TBlock:= BinarySearchBlock(TOperator(FDedicatedStatement).Blocks, AX);

if Block <> nil then

FHoveredStatement.Statement:= BinarySearchStatement(AX, AY, Block);

end;

if FHoveredStatement.Statement = nil then

begin

FHoveredStatement.Statement:= BinarySearchStatement(AX, AY, FMainBlock);

if FHoveredStatement.Statement = nil then

Exit;

FHoveredStatement.Rect:= CreateRect(FHoveredStatement.Statement);

if FHoveredStatement.Statement <> FDedicatedStatement then

begin

var YStart: Integer := FHoveredStatement.Statement.YStart;

var YLast: Integer := FHoveredStatement.Statement.YLast;

if (FHoveredStatement.Statement is TOperator) and

not TOperator(FHoveredStatement.Statement).IsPreсOperator then

begin

YStart:= TOperator(FHoveredStatement.Statement).Blocks[0].Statements.GetLast.GetYBottom;

YLast := FHoveredStatement.Statement.GetYBottom;

end;

if AY >= YLast - FHoveredStatement.Statement.YIndentText then

begin

FHoveredStatement.Rect.Top:= YLast - FHoveredStatement.Statement.YIndentText;

FHoveredStatement.Rect.Bottom:= YLast;

FHoveredStatement.State := stAfter;

end

else if AY <= YStart + FHoveredStatement.Statement.YIndentText then

begin

FHoveredStatement.Rect.Top:= YStart;

FHoveredStatement.Rect.Bottom:= YStart + FHoveredStatement.Statement.YIndentText;

FHoveredStatement.State := stBefore;

end

else if FHoveredStatement.Statement is TOperator then

begin

var BaseOperator: TOperator:= TOperator(FHoveredStatement.Statement);

var CurrBlock: TBlock := FDedicatedStatement.BaseBlock;

while CurrBlock.BaseOperator <> nil do

begin

if CurrBlock.BaseOperator = BaseOperator then

Exit;

CurrBlock := CurrBlock.BaseOperator.BaseBlock;

end;

FHoveredStatement.State := stSwap;

end

else if not isDefaultStatement(FHoveredStatement.Statement) then

FHoveredStatement.State := stSwap;

end;

end

else

FHoveredStatement.Rect:= CreateRect(FHoveredStatement.Statement);

if isDefaultStatement(FDedicatedStatement) then

begin

FHoveredStatement.State := stCancel;

if FHoveredStatement.Statement <> nil then

FHoveredStatement.Rect:= CreateRect(FHoveredStatement.Statement);

end;

end;

procedure TBlockManager.TryDrawCarryBlock(const AVisibleImageRect: TVisibleImageRect);

const

Offset = 3;

begin

if FCarryBlock <> nil then

begin

if FHoveredStatement.Statement <> nil then

case FHoveredStatement.State of

stAfter:

begin

ColorizeRect(FPaintBox.Canvas, FHoveredStatement.Rect, FOKColor);

DrawArrow(FPaintBox.Canvas,

FHoveredStatement.Rect.Width shr 1 +

FHoveredStatement.Rect.Left,

FHoveredStatement.Rect.Bottom,

FHoveredStatement.Rect.Top + Offset,

FArrowColor);

if (FHoveredStatement.Statement is TOperator) and

TOperator(FHoveredStatement.Statement).IsPreсOperator then

begin

FHoveredStatement.Rect.Right:= FHoveredStatement.Statement.BaseBlock.XStart +

TOperator(FHoveredStatement.Statement).GetOffsetFromXStart;

FHoveredStatement.Rect.Bottom:= FHoveredStatement.Statement.GetYBottom;

ColorizeRect(FPaintBox.Canvas, FHoveredStatement.Rect, FOKColor);

end;

end;

stBefore:

begin

ColorizeRect(FPaintBox.Canvas, FHoveredStatement.Rect, FOKColor);

DrawArrow(FPaintBox.Canvas,

FHoveredStatement.Rect.Width shr 1 +

FHoveredStatement.Rect.Left,

FHoveredStatement.Rect.Top,

FHoveredStatement.Rect.Bottom - Offset,

FArrowColor);

if (FHoveredStatement.Statement is TOperator) and

not TOperator(FHoveredStatement.Statement).IsPreсOperator then

begin

FHoveredStatement.Rect.Right:= FHoveredStatement.Statement.BaseBlock.XStart +

TOperator(FHoveredStatement.Statement).GetOffsetFromXStart;

FHoveredStatement.Rect.Top:= FHoveredStatement.Statement.YStart;

ColorizeRect(FPaintBox.Canvas, FHoveredStatement.Rect, FOKColor);

end;

end;

stSwap:

ColorizeRect(FPaintBox.Canvas, FHoveredStatement.Rect, FOKColor);

stCancel:

ColorizeRect(FPaintBox.Canvas, FHoveredStatement.Rect, FCancelColor);

end;

FCarryBlock.DrawBlock(AVisibleImageRect);

end;

end;

procedure TBlockManager.TryTakeAction;

begin

case FHoveredStatement.State of

stBefore, stAfter:

AddToUndoStack(TCommandTransferAnotherBlock.Create(

FHoveredStatement.Statement,

Boolean(Ord(FHoveredStatement.State)),

FDedicatedStatement));

stSwap:

AddToUndoStack(TCommandSwapStatements.Create(

FHoveredStatement.Statement,

FDedicatedStatement));

end;

end;

procedure TBlockManager.DestroyCarryBlock;

begin

FCarryBlock.Destroy;

FCarryBlock:= nil;

FHoveredStatement.State := stCancel;

FHoveredStatement.Statement := nil;

FPaintBox.Invalidate;

end;

{ Interactions with statements }

class function TBlockManager.CreateStatement(const AStatementClass: TStatementClass;

const ABaseBlock: TBlock): TStatement;

var

Action: String;

begin

Result:= nil;

Action := '';

if WriteAction.TryGetAction(Action) then

begin

if AStatementClass = TCaseBranching then

begin

var Cond: TStringArr:= nil;

if WriteCaseConditions.TryGetCond(Cond) then

Result:= TCaseBranching.Create(Action, Cond);

end

else

Result:= AStatementClass.Create(Action);

end;

end;

{ Stacks }

procedure TBlockManager.TryUndo;

var

Commamd: ICommand;

begin

if FUndoStack.Count <> 0 then

begin

FisSaved:= False;

Commamd:= FUndoStack.Pop;

Commamd.Undo;

FRedoStack.Push(Commamd);

FDedicatedStatement := nil;

FPaintBox.Invalidate;

end;

end;

procedure TBlockManager.TryRedo;

var

Commamd: ICommand;

begin

if FRedoStack.Count <> 0 then

begin

FisSaved:= False;

Commamd:= FRedoStack.Pop;

Commamd.Execute;

FUndoStack.Push(Commamd);

FDedicatedStatement := nil;

FPaintBox.Invalidate;

end;

end;

{ View update }

procedure TBlockManager.Draw(const AVisibleImageRect: TVisibleImageRect);

const

Stock = 42 shl 2;

Correction = 5;

begin

FPaintBox.Canvas.Font := FFont;

FPaintBox.Canvas.Pen := FPen;

FPaintBox.Width := Max(FMainBlock.XLast + Stock,

AVisibleImageRect.FBottomRight.X -

AVisibleImageRect.FTopLeft.X - Correction);

FPaintBox.Height := Max(FMainBlock.Statements.GetLast.GetYBottom + Stock,

AVisibleImageRect.FBottomRight.Y -

AVisibleImageRect.FTopLeft.Y - Correction);

AVisibleImageRect.Expand(Stock);

if FDedicatedStatement <> nil then

ColorizeRect(FPaintBox.Canvas,CreateRect(FDedicatedStatement), FHighlightColor);

TryDrawCarryBlock(AVisibleImageRect);

FMainBlock.DrawBlock(AVisibleImageRect);

end;

end.

Приложение З

(обязательное)

Исходный код программы (модуль uArrayList)

unit uArrayList;

interface

type

TArrayList<T> = class

private

FArray: array of T;

FCount: Integer;

procedure CapacityInc;

function GetItem(const AIndex: Integer): T;

procedure SetItem(const AIndex: Integer; const AValue: T);

public

constructor Create(const AInitialCapacity: Integer = 0);

destructor Destroy; override;

procedure Delete(const AIndex: Integer);

procedure Add(const Item: T);

procedure Insert(const AItem: T; const AIndex: Integer);

procedure Clear;

property Items[const Index: Integer]: T read GetItem write SetItem; default;

property Count: Integer read FCount;

function GetLast: T;

end;

implementation

constructor TArrayList<T>.Create(const AInitialCapacity: Integer = 0);

begin

FCount := 0;

SetLength(FArray, AInitialCapacity);

end;

destructor TArrayList<T>.Destroy;

begin

SetLength(FArray, 0);

inherited;

end;

procedure TArrayList<T>.Add(const Item: T);

begin

if FCount = Length(FArray) then

CapacityInc;

FArray[FCount] := Item;

Inc(FCount);

end;

procedure TArrayList<T>.Delete(const AIndex: Integer);

var

I: Integer;

begin

for I := AIndex to FCount - 2 do

FArray[I] := FArray[I + 1];

Dec(FCount);

end;

procedure TArrayList<T>.Insert(const AItem: T; const AIndex: Integer);

var

I: Integer;

begin

if FCount = Length(FArray) then

CapacityInc;

for I := FCount - 1 downto AIndex do

FArray[I + 1] := FArray[I];

FArray[AIndex] := AItem;

Inc(FCount);

end;

procedure TArrayList<T>.Clear;

begin

FCount := 0;

end;

function TArrayList<T>.GetItem(const AIndex: Integer): T;

begin

Result := FArray[AIndex];

end;

procedure TArrayList<T>.SetItem(const AIndex: Integer; const AValue: T);

begin

FArray[AIndex] := AValue;

end;

function TArrayList<T>.GetLast: T;

begin

Result:= FArray[FCount - 1];

end;

procedure TArrayList<T>.CapacityInc;

begin

SetLength(FArray, Length(FArray) shl 1 + 4);

end;

end.

Приложение И

(обязательное)

Исходный код программы (модуль uStack)

unit uStack;

interface

type

TStack<T> = class

private type

PItem = ^TItem;

TItem = record

FData: T;

FNext: PItem;

end;

private

FTop: PItem;

FCount: Integer;

public

constructor Create;

destructor Destroy; override;

procedure Push(const AItem: T);

procedure Clear;

function Pop: T;

function Peek: T;

property Count: Integer read FCount;

end;

implementation

constructor TStack<T>.Create;

begin

FTop := nil;

FCount := 0;

end;

destructor TStack<T>.Destroy;

begin

Clear;

inherited;

end;

procedure TStack<T>.Clear;

var

I: Integer;

begin

for I := FCount - 1 downto 0 do

Pop;

end;

procedure TStack<T>.Push(const AItem: T);

var

NewItem: PItem;

begin

New(NewItem);

NewItem^.FData := AItem;

NewItem^.FNext := FTop;

FTop := NewItem;

Inc(FCount);

end;

function TStack<T>.Pop: T;

var

Item: PItem;

begin

Item := FTop;

FTop := FTop^.FNext;

Result := Item^.FData;

Dispose(Item);

Dec(FCount);

end;

function TStack<T>.Peek: T;

begin

Result := FTop^.FData;

end;

end.

Приложение К

(обязательное)

Исходный код программы (модуль uGlobalSave)

unit uGlobalSave;

interface

uses

UBlockManager, uBase, System.JSON, uIfBranching, Vcl.Graphics, System.SysUtils,

System.IOUtils, uConstants;

procedure ResetGlobalSettings;

procedure LoadGlobalSettings;

procedure SaveGlobalSettings;

implementation

const

constIfTrueCond = 'True';

constIfFalseCond = 'False';

constDefaultAction = '';

constHighlightColor = clYellow;

constArrowColor = clBlack;

constOKColor = clGreen;

constCancelColor = clRed;

constGbSettingsNameWithExt = 'GlobalSettings' + constExtJSON;

procedure ResetGlobalSettings;

begin

TIfBranching.TrueCond := constIfTrueCond;

TIfBranching.FalseCond := constIfFalseCond;

DefaultAction := constDefaultAction;

with TBlockManager do

begin

HighlightColor := constHighlightColor;

ArrowColor := constArrowColor;

OKColor := constOKColor;

CancelColor := constCancelColor;

end;

end;

procedure LoadGlobalSettings;

var

Json: TJSONObject;

AppDataPath: string;

begin

AppDataPath := IncludeTrailingPathDelimiter(ExtractFilePath(ParamStr(0))) + dirAppData;

AppDataPath := TPath.Combine(AppDataPath, constGbSettingsNameWithExt);

if FileExists(AppDataPath) then

begin

Json := TJSONObject(TJSONObject.ParseJSONValue(TFile.ReadAllText(AppDataPath, TEncoding.UTF8)));

try

with Json do

begin

DefaultAction := GetValue('DefaultAction').Value;

TIfBranching.TrueCond := GetValue('TrueCond').Value;

TIfBranching.FalseCond := GetValue('FalseCond').Value;

with TBlockManager do

begin

HighlightColor := StringToColor(GetValue('HighlightColor').Value);

ArrowColor := StringToColor(GetValue('ArrowColor').Value);

OKColor := StringToColor(GetValue('OKColor').Value);

CancelColor := StringToColor(GetValue('CancelColor').Value);

end;

end;

except

ResetGlobalSettings;

end;

Json.Destroy;

end

else

ResetGlobalSettings;

end;

procedure SaveGlobalSettings;

var

Json: TJSONObject;

AppDataPath: string;

begin

Json := TJSONObject.Create;

try

with Json do

begin

AddPair('DefaultAction', DefaultAction);

AddPair('TrueCond', TIfBranching.TrueCond);

AddPair('FalseCond', TIfBranching.FalseCond);

with TBlockManager do

begin

AddPair('HighlightColor', ColorToString(HighlightColor));

AddPair('ArrowColor', ColorToString(ArrowColor));

AddPair('OKColor', ColorToString(OKColor));

AddPair('CancelColor', ColorToString(CancelColor));

end;

end;

AppDataPath := IncludeTrailingPathDelimiter(ExtractFilePath(ParamStr(0))) + dirAppData;

if not TDirectory.Exists(AppDataPath) then

TDirectory.CreateDirectory(AppDataPath);

AppDataPath := TPath.Combine(AppDataPath, constGbSettingsNameWithExt);

TFile.WriteAllText(AppDataPath, Json.ToJSON, TEncoding.UTF8);

finally

Json.Destroy;

end;

end;

end.

Приложение Л

(обязательное)

Исходный код программы (модуль uLocalSave)

unit uLocalSave;

interface

uses

UBlockManager, uBase, System.JSON, System.Classes, uIfBranching, Vcl.Graphics,

System.SysUtils, System.IOUtils, System.UITypes, uCaseBranching, uAdditionalTypes,

uStatementConverter, System.Generics.Collections, Vcl.Dialogs, uDialogMessages;

procedure SaveSchema(const ABlockManager: TBlockManager);

procedure LoadSchema(const ABlockManager: TBlockManager);

implementation

{ Pen }

// Load

procedure JSONToPen(const AJSON: TJSONObject; const APen: TPen);

begin

with APen do

begin

Color := StringToColor(AJSON.GetValue('Color').Value);

Width := AJSON.GetValue('Width').Value.ToInteger;

Style := TPenStyle(AJSON.GetValue('Style').Value.ToInteger);

Mode := TPenMode(AJSON.GetValue('Mode').Value.ToInteger);

end;

end;

// Save

function PenToJSON(const APen: TPen): TJSONObject;

begin

Result := TJSONObject.Create;

with Result do

begin

AddPair('Color', ColorToString(APen.Color));

AddPair('Width', TJSONNumber.Create(APen.Width));

AddPair('Style', TJSONNumber.Create(Ord(APen.Style)));

AddPair('Mode', TJSONNumber.Create(Ord(APen.Mode)));

end;

end;

{ Font }

function GetOrdFontStyle(AFontStyles: TFontStyles): Integer;

begin

Result := Ord(fsBold in AFontStyles) shl 3 or

Ord(fsItalic in AFontStyles) shl 2 or

Ord(fsUnderline in AFontStyles) shl 1 or

Ord(fsStrikeOut in AFontStyles);

end;

function GetFontStyleFromOrd(AOrd: Integer): TFontStyles;

begin

Result := [];

if AOrd and $01 = $01 then

Include(Result, fsStrikeOut);

if AOrd and $02 = $02 then

Include(Result, fsUnderline);

if AOrd and $04 = $04 then

Include(Result, fsItalic);

if AOrd and $08 = $08 then

Include(Result, fsBold);

end;

// Load

procedure JSONToFont(const AJSON: TJSONObject; const AFont: TFont);

begin

with AFont do

begin

Size := AJSON.GetValue('Size').Value.ToInteger;

Name := AJSON.GetValue('Name').Value;

Color := StringToColor(AJSON.GetValue('Color').Value);

Style := GetFontStyleFromOrd(AJSON.GetValue('Style').Value.ToInteger);

Charset := AJSON.GetValue('Charset').Value.ToInteger;

end;

end;

// Save

function FontToJSON(const AFont: TFont): TJSONObject;

begin

Result := TJSONObject.Create;

with Result do

begin

AddPair('Size', TJSONNumber.Create(AFont.Size));

AddPair('Name', AFont.Name);

AddPair('Color', ColorToString(AFont.Color));

AddPair('Style', TJSONNumber.Create(GetOrdFontStyle(AFont.Style)));

AddPair('Charset', TJSONNumber.Create(Ord(AFont.Charset)));

end;

end;

{ Statement }

// Load

function JSONToBlock(const JsonObject: TJSONObject; const ABaseOperator: TOperator;

const ACanvas: TCanvas): TBlock; forward;

function JSONToStatement(const JsonObject: TJSONObject; const ACanvas: TCanvas): TStatement;

var

CurrOperator: TOperator;

MyJSONArray: TJSONArray;

I: Integer;

StatementClass: TStatementClass;

begin

StatementClass := ConvertToStatementType(JsonObject.GetValue('StatementIndex').Value.ToInteger);

if StatementClass = TCaseBranching then

begin

MyJSONArray := TJSONArray(JsonObject.GetValue('Conds'));

var StringArr : TStringArr;

SetLength(StringArr, MyJSONArray.Count);

for I := 0 to MyJSONArray.Count - 1 do

StringArr[I] := MyJSONArray.Items[I].Value;

Result:= TCaseBranching.Create(JsonObject.GetValue('Action').Value, StringArr);

end

else

Result:= StatementClass.Create(JsonObject.GetValue('Action').Value);

Result.SetCoords(JsonObject.GetValue('YStart').Value.ToInteger,

JsonObject.GetValue('YLast').Value.ToInteger);

if Result is TOperator then

begin

CurrOperator:= TOperator(Result);

MyJSONArray := TJSONArray(JsonObject.GetValue('Blocks'));

for I := 0 to MyJSONArray.Count - 1 do

CurrOperator.Blocks[I] := JSONToBlock(TJSONObject(MyJSONArray.Items[I]), CurrOperator, ACanvas);

end;

end;

// Save

function BlockToJSON(const ABlock: TBlock): TJSONObject; forward;

function StatementToJSON(const AStatement: TStatement): TJSONObject;

var

CurrOperator: TOperator;

MyJSONArray: TJSONArray;

I, StatementIndex: Integer;

begin

Result := TJSONObject.Create;

StatementIndex := AStatement.GetSerialNumber;

Result.AddPair('StatementIndex', TJSONNumber.Create(StatementIndex));

if StatementIndex = 2 {2: TCaseBranching} then

begin

MyJSONArray := TJSONArray.Create;

var StringArr : TStringArr := TCaseBranching(CurrOperator).Conds;

for I := 0 to High(StringArr) do

MyJSONArray.Add(StringArr[I]);

Result.AddPair('Conds', MyJSONArray);

end;

Result.AddPair('Action', AStatement.Action);

Result.AddPair('YStart', TJSONNumber.Create(AStatement.YStart));

Result.AddPair('YLast', TJSONNumber.Create(AStatement.YLast));

if AStatement is TOperator then

begin

CurrOperator:= TOperator(AStatement);

MyJSONArray := TJSONArray.Create;

for I := 0 to High(CurrOperator.Blocks) do

MyJSONArray.AddElement(BlockToJSON(CurrOperator.Blocks[I]));

Result.AddPair('Blocks', MyJSONArray);

end;

end;

{ Block }

// Load

function JSONToBlock(const JsonObject: TJSONObject; const ABaseOperator: TOperator;

const ACanvas: TCanvas): TBlock;

var

JsonArray: TJSONArray;

I: Integer;

begin

Result := TBlock.Create(

JsonObject.GetValue('XStart').Value.ToInteger,

JsonObject.GetValue('XLast').Value.ToInteger,

ABaseOperator,

ACanvas);

JsonArray := TJSONArray(JsonObject.GetValue('Statements'));

for I := 0 to JsonArray.Count - 1 do

Result.AddStatement(JSONToStatement(TJSONObject(JsonArray.Items[I]), ACanvas));

end;

// Save

function BlockToJSON(const ABlock: TBlock): TJSONObject;

var

I: Integer;

JsonArray: TJsonArray;

begin

Result := TJSONObject.Create;

Result.AddPair('XStart', TJSONNumber.Create(ABlock.XStart));

Result.AddPair('XLast', TJSONNumber.Create(ABlock.XLast));

JsonArray := TJsonArray.Create;

for I := 0 to ABlock.Statements.Count - 1 do

JsonArray.AddElement(StatementToJSON(ABlock.Statements[I]));

Result.AddPair('Statements', JsonArray);

end;

{ Schema }

// Save

procedure SaveSchema(const ABlockManager: TBlockManager);

var

Json: TJSONObject;

begin

if ABlockManager.PathToFile <> '' then

begin

Json := TJSONObject.Create;

try

with Json do

begin

with ABlockManager do

begin

AddPair('Pen', PenToJSON(Pen));

AddPair('Font', FontToJSON(Font));

AddPair('MainBlock', BlockToJSON(MainBlock));

end;

AddPair('DefaultAction', uBase.DefaultAction);

end;

TFile.WriteAllText(ABlockManager.PathToFile, Json.ToJSON, TEncoding.UTF8);

finally

Json.Destroy;

end;

end;

end;

// Load

procedure LoadSchema(const ABlockManager: TBlockManager);

var

Json: TJSONObject;

OldDefaultAction: string;

begin

if TFile.Exists(ABlockManager.PathToFile) and

SameText(TPath.GetExtension(ABlockManager.PathToFile), '.json') then

begin

Json := TJSONObject(TJSONObject.ParseJSONValue(TFile.ReadAllText(ABlockManager.PathToFile, TEncoding.UTF8)));

try

with Json do

begin

with ABlockManager do

begin

JSONToPen(TJSONObject(GetValue('Pen')), Pen);

JSONToFont(TJSONObject(GetValue('Font')), Font);

MainBlock := JSONToBlock(TJSONObject(GetValue('MainBlock')), nil, PaintBox.Canvas);

TIfBranching.RedefineSizesForIfBranching(MainBlock);

end;

OldDefaultAction := Json.GetValue('DefaultAction').Value;

if OldDefaultAction <> uBase.DefaultAction then

ABlockManager.MainBlock.SetNewActionForDefaultStatements(OldDefaultAction);

end;

except

ShowMessage(rsErrorFile);

end;

Json.Destroy;

end;

end;

end.

Приложение М

(обязательное)

Исходный код программы (модуль uExport)

unit uExport;

interface

uses

uBlockManager, Vcl.Graphics, Vcl.ExtCtrls, uAdditionalTypes, uConstants, PNGImage,

System.SysUtils, Classes, uBase, uCaseBranching, uFirstLoop, uIfBranching, uLastLoop,

uProcessStatement, System.UITypes, System.IOUtils, System.Types;

procedure SaveBMPFile(const ABlockManager: TBlockManager; const AFileName: string);

procedure SavePNGFile(const ABlockManager: TBlockManager; const AFileName: string);

procedure SaveSVGFile(const ABlockManager: TBlockManager; const AFileName: string);

implementation

procedure InitializeVisibleImageRect(const ABitmap: TBitmap;

out AVisibleImageRect: TVisibleImageRect);

begin

AVisibleImageRect.FTopLeft.X := 0;

AVisibleImageRect.FTopLeft.Y := 0;

AVisibleImageRect.FBottomRight.X := ABitmap.Width;

AVisibleImageRect.FBottomRight.Y := ABitmap.Height;

end;

procedure InitializeBitmap(const ABitmap: TBitmap; const ABlockManager: TBlockManager);

begin

ABitmap.Width := ABlockManager.MainBlock.XLast + SchemeIndent;

ABitmap.Height := ABlockManager.MainBlock.Statements[ABlockManager.MainBlock.

Statements.Count - 1].GetYBottom + SchemeIndent;

ABitmap.Canvas.Font := ABlockManager.Font;

ABitmap.Canvas.Pen := ABlockManager.Pen;

end;

procedure SaveBMPFile(const ABlockManager: TBlockManager; const AFileName: string);

var

VisibleImageRect: TVisibleImageRect;

Bitmap: TBitmap;

begin

Bitmap := TBitmap.Create;

try

InitializeBitmap(Bitmap, ABlockManager);

InitializeVisibleImageRect(Bitmap, VisibleImageRect);

ABlockManager.MainBlock.InstallCanvas(Bitmap.Canvas);

ABlockManager.MainBlock.DrawBlock(VisibleImageRect);

Bitmap.SaveToFile(AFileName);

finally

Bitmap.Destroy;

end;

ABlockManager.MainBlock.InstallCanvas(ABlockManager.PaintBox.Canvas);

end;

procedure SavePNGFile(const ABlockManager: TBlockManager; const AFileName: string);

var

Bitmap: TBitmap;

PNG: TPNGImage;

VisibleImageRect: TVisibleImageRect;

begin

Bitmap := TBitmap.Create;

try

InitializeBitmap(Bitmap, ABlockManager);

InitializeVisibleImageRect(Bitmap, VisibleImageRect);

ABlockManager.MainBlock.InstallCanvas(Bitmap.Canvas);

ABlockManager.MainBlock.DrawBlock(VisibleImageRect);

PNG := TPNGImage.Create;

try

PNG.Assign(Bitmap);

PNG.SaveToFile(AFileName);

finally

PNG.Destroy;

end;

finally

Bitmap.Destroy;

end;

ABlockManager.MainBlock.InstallCanvas(ABlockManager.PaintBox.Canvas);

end;

{ SVG }

const

SVGHead = '<?xml version="1.0" encoding="UTF-8" standalone="no"?>' +

'<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" ' +

'"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">';

clrFill: TColor = clWhite;

CorrectionToSvg = 1.333;

CorrectionToCanv = 1 / CorrectionToSvg;

function ColorToRGBString(const AColor: TColor): string;

var

RGB: Longint;

R, G, B: Byte;

begin

RGB := ColorToRGB(AColor);

R := (RGB and $000000FF);

G := (RGB and $0000FF00) shr 8;

B := (RGB and $00FF0000) shr 16;

Result := Format('#%.2x%.2x%.2x', [R, G, B]);

end;

procedure SetSVGOpenTag(const SVG: TStringList; const AWidth, AHeight: Integer);

begin

SVG.Add('<svg xmlns="http://www.w3.org/2000/svg" ' +

'xmlns:xlink="http://www.w3.org/1999/xlink" ' +

'width="' + IntToStr(AWidth) + 'px" ' +

'height="' + IntToStr(AHeight) + 'px" ' +

'version="1.1">');

end;

procedure DrawRectangle(const SVG: TStringList; const AXStart, AXLast, AYStart, AYLast: Integer; const APen: TPen);

begin

SVG.Add('<rect x="' + IntToStr(AXStart) + 'px" y="' + IntToStr(AYStart) + 'px" width="' + IntToStr(AXLast - AXStart) + 'px" height="' + IntToStr(AYLast - AYStart) + 'px" stroke="' + ColorToRGBString(APen.Color) + '" stroke-width="' + IntToStr(APen.Width) + 'px" fill="' + ColorToRGBString(clrFill) + '" />');

end;

procedure DrawLine(const SVG: TStringList; const AXStart, AXLast, AYStart, AYLast: Integer; const APen: TPen);

begin

SVG.Add('<line x1="' + IntToStr(AXStart) + 'px" y1="' + IntToStr(AYStart) + 'px" x2="' + IntToStr(AXLast) + 'px" y2="' + IntToStr(AYLast) + 'px" stroke="' + ColorToRGBString(APen.Color) + '" stroke-width="' + IntToStr(APen.Width) + 'px" />');

end;

procedure DrawText(const SVG: TStringList; AXStart, AYStart: Integer; const ACanvas: TCanvas; const AText: string);

var

Lines: TStringDynArray;

I, Indent: Integer;

begin

Lines := AText.Split([sLineBreak]);

Indent := Round(CorrectionToCanv \* ACanvas.TextHeight(Space));

for I := 0 to High(Lines) do

SVG.Add('<text x="' + IntToStr(AXStart) + 'px" y="' + IntToStr(AYStart + Indent \* I) + 'px" font-family="' + ACanvas.Font.Name + '" font-size="' + IntToStr(ACanvas.Font.Size) + 'px" fill="' + ColorToRGBString(ACanvas.Font.Color) + '">' + Lines[I] + '</text>');

end;

procedure DrawInvertedTriangle(const SVG: TStringList; const AXStart, AXMiddle, AXLast, AYStart, AYLast: Integer; const APen: TPen);

begin

SVG.Add('<line x1="' + IntToStr(AXStart) + 'px" y1="' + IntToStr(AYStart) + 'px" x2="' + IntToStr(AXMiddle) + 'px" y2="' + IntToStr(AYLast) + 'px" stroke="' + ColorToRGBString(APen.Color) + '" stroke-width="' + IntToStr(APen.Width) + 'px" />');

SVG.Add('<line x1="' + IntToStr(AXLast) + 'px" y1="' + IntToStr(AYStart) + 'px" x2="' + IntToStr(AXMiddle) + 'px" y2="' + IntToStr(AYLast) + 'px" stroke="' + ColorToRGBString(APen.Color) + '" stroke-width="' + IntToStr(APen.Width) + 'px" />');

end;

procedure DrawUnfinishedHorRectForLoop(const SVG: TStringList; const AXStart, AXMiddle, AXLast, AYStart, AYLast: Integer; const APen: TPen);

begin

SVG.Add('<line x1="' + IntToStr(AXStart) + 'px" y1="' + IntToStr(AYStart) + 'px" x2="' + IntToStr(AXLast) + 'px" y2="' + IntToStr(AYStart) + 'px" stroke="' + ColorToRGBString(APen.Color) + '" stroke-width="' + IntToStr(APen.Width) + 'px" />');

SVG.Add('<line x1="' + IntToStr(AXLast) + 'px" y1="' + IntToStr(AYStart) + 'px" x2="' + IntToStr(AXLast) + 'px" y2="' + IntToStr(AYLast) + 'px" stroke="' + ColorToRGBString(APen.Color) + '" stroke-width="' + IntToStr(APen.Width) + 'px" />');

SVG.Add('<line x1="' + IntToStr(AXLast) + 'px" y1="' + IntToStr(AYLast) + 'px" x2="' + IntToStr(AXMiddle) + 'px" y2="' + IntToStr(AYLast) + 'px" stroke="' + ColorToRGBString(APen.Color) + '" stroke-width="' + IntToStr(APen.Width) + 'px" />');

end;

procedure DrawUnfinishedVertRectForLoop(const SVG: TStringList; const AXStart, AXLast, AYStart, AYMiddle, AYLast: Integer; const APen: TPen);

begin

SVG.Add('<line x1="' + IntToStr(AXStart) + 'px" y1="' + IntToStr(AYStart) + 'px" x2="' + IntToStr(AXStart) + 'px" y2="' + IntToStr(AYLast) + 'px" stroke="' + ColorToRGBString(APen.Color) + '" stroke-width="' + IntToStr(APen.Width) + 'px" />');

SVG.Add('<line x1="' + IntToStr(AXStart) + 'px" y1="' + IntToStr(AYLast) + 'px" x2="' + IntToStr(AXLast) + 'px" y2="' + IntToStr(AYLast) + 'px" stroke="' + ColorToRGBString(APen.Color) + '" stroke-width="' + IntToStr(APen.Width) + 'px" />');

SVG.Add('<line x1="' + IntToStr(AXLast) + 'px" y1="' + IntToStr(AYLast) + 'px" x2="' + IntToStr(AXLast) + 'px" y2="' + IntToStr(AYMiddle) + 'px" stroke="' + ColorToRGBString(APen.Color) + '" stroke-width="' + IntToStr(APen.Width) + 'px" />');

end;

procedure DrawProcess(const SVG: TStringList; const AProcessStatement: TProcessStatement);

begin

with AProcessStatement do

begin

DrawRectangle(SVG, BaseBlock.XStart, BaseBlock.XLast,

YStart, YLast, BaseBlock.Canvas.Pen);

DrawText(SVG,

BaseBlock.XStart + ((BaseBlock.XLast - BaseBlock.XStart) shr 1)

- (ActionSize.Width shr 1),

YStart + Round(BaseBlock.Canvas.Font.Size \* CorrectionToCanv) + YIndentText,

BaseBlock.Canvas, Action);

end;

end;

procedure DrawBlock(const SVG: TStringList; const ABlock: TBlock); forward;

procedure DrawIfBranching(const SVG: TStringList; const AIfBranching: TIfBranching);

begin

with AIfBranching do

begin

DrawRectangle(SVG, BaseBlock.XStart, BaseBlock.XLast,

YStart, YLast, BaseBlock.Canvas.Pen);

DrawInvertedTriangle(SVG, BaseBlock.XStart, Blocks[1].XStart, BaseBlock.XLast,

YStart, YLast, BaseBlock.Canvas.Pen);

DrawText(SVG,

Blocks[0].XStart +

GetAvailablePartWidth(Blocks[0].XLast - Blocks[0].XStart, TrueSize.Height + YIndentText) +

GetAvailablePartWidth(BaseBlock.XLast - BaseBlock.XStart, ActionSize.Height) shr 1 -

ActionSize.Width shr 1,

YStart + Round(BaseBlock.Canvas.Font.Size \* CorrectionToCanv) + YIndentText,

BaseBlock.Canvas, Action);

DrawText(SVG,

Blocks[0].XStart + GetAvailablePartWidth(

Blocks[0].XLast - Blocks[0].XStart, TrueSize.Height) shr 1 -

TrueSize.Width shr 1,

YStart + Round(BaseBlock.Canvas.Font.Size \* CorrectionToCanv) + YIndentText shl 1 + ActionSize.Height,

BaseBlock.Canvas, TIfBranching.TrueCond);

DrawText(SVG,

Blocks[1].XLast - GetAvailablePartWidth(

Blocks[1].XLast - Blocks[1].XStart, FalseSize.Height) shr 1 -

FalseSize.Width shr 1,

YStart + Round(BaseBlock.Canvas.Font.Size \* CorrectionToCanv) + YIndentText shl 1 + ActionSize.Height,

BaseBlock.Canvas, TIfBranching.FalseCond);

DrawBlock(SVG, Blocks[0]);

DrawBlock(SVG, Blocks[1]);

end;

end;

procedure DrawCaseBranching(const SVG: TStringList; const ACaseBranching: TCaseBranching);

var

I: Integer;

YTriangleHeight : Integer;

LeftTriangleWidth : Integer;

PartLeftTriangleWidth : Integer;

begin

with ACaseBranching do

begin

// Calculate the height of a triangle

YTriangleHeight:= YStart + ActionSize.Height + YIndentText shl 1;

// Drawing the main block

DrawRectangle(SVG, BaseBlock.XStart, BaseBlock.XLast,

YStart, YLast, BaseBlock.Canvas.Pen);

// Drawing a triangle

DrawInvertedTriangle(SVG, BaseBlock.XStart, Blocks[High(Blocks)].XStart,

BaseBlock.XLast, YStart, YTriangleHeight, BaseBlock.Canvas.Pen);

// Draw a line that connects the vertex of the triangle and

// the lower base of the operator

DrawLine(SVG, Blocks[High(Blocks)].XStart,

Blocks[High(Blocks)].XStart, YTriangleHeight, YLast, BaseBlock.Canvas.Pen);

{ Draw the lines that connect the side of the triangle to the side of the block }

// Calculate the width to the left of the vertex of the triangle

LeftTriangleWidth:= 0;

for I := 0 to High(Blocks) - 1 do

Inc(LeftTriangleWidth, Blocks[I].XLast - Blocks[I].XStart);

// Find the Y coordinate for each block

PartLeftTriangleWidth:= LeftTriangleWidth;

for I := 0 to High(Blocks) - 2 do

begin

Dec(PartLeftTriangleWidth, Blocks[I].XLast - Blocks[I].XStart);

DrawLine(SVG, Blocks[I].XLast, Blocks[I].XLast,

YTriangleHeight - (YTriangleHeight - YStart) \*

PartLeftTriangleWidth div LeftTriangleWidth,

YLast, BaseBlock.Canvas.Pen);

end;

{ End }

// Drawing the action

DrawText(SVG,

BaseBlock.XStart

+

LeftTriangleWidth \* (ActionSize.Height + YIndentText) div (YTriangleHeight - YStart)

+

(BaseBlock.XLast - BaseBlock.XStart) \* YIndentText div (YTriangleHeight - YStart) shr 1

-

ActionSize.Width shr 1

,

YStart + Round(BaseBlock.Canvas.Font.Size \* CorrectionToCanv) + YIndentText, BaseBlock.Canvas, Action);

// Drawing the conditions

Inc(YTriangleHeight, YIndentText);

for I := 0 to High(Conds) do

DrawText(SVG,

Blocks[I].XStart + ((Blocks[I].XLast - Blocks[I].XStart) shr 1)

- (CondsSizes[I].Width shr 1),

YTriangleHeight, BaseBlock.Canvas, Conds[I]);

for I := 0 to High(Blocks) do

DrawBlock(SVG, Blocks[I]);

end;

end;

procedure DrawFirstLoop(const SVG: TStringList; const AFirstLoop: TFirstLoop);

begin

with AFirstLoop do

begin

DrawUnfinishedVertRectForLoop(SVG, BaseBlock.XStart, BaseBlock.XLast, YStart,

YLast, GetYBottom, BaseBlock.Canvas.Pen);

DrawUnfinishedHorRectForLoop(SVG, BaseBlock.XStart, Blocks[0].XStart,

BaseBlock.XLast, YStart, YLast, BaseBlock.Canvas.Pen);

DrawText(SVG,

BaseBlock.XStart + ((BaseBlock.XLast - BaseBlock.XStart) shr 1)

- (ActionSize.Width shr 1),

YStart + Round(BaseBlock.Canvas.Font.Size \* CorrectionToCanv) + YIndentText,

BaseBlock.Canvas, Action);

DrawBlock(SVG, Blocks[0]);

end;

end;

procedure DrawLastLoop(const SVG: TStringList; const ALastLoop: TLastLoop);

begin

with ALastLoop do

begin

DrawUnfinishedVertRectForLoop(SVG, BaseBlock.XStart, BaseBlock.XLast, YLast,

GetYBottom, YStart, BaseBlock.Canvas.Pen);

DrawUnfinishedHorRectForLoop(SVG, BaseBlock.XStart, Blocks[0].XStart,

BaseBlock.XLast, YLast, YStart, BaseBlock.Canvas.Pen);

DrawText(SVG,

BaseBlock.XStart + ((BaseBlock.XLast - BaseBlock.XStart) shr 1)

- (ActionSize.Width shr 1),

GetBlockYBottom + Round(BaseBlock.Canvas.Font.Size \* CorrectionToCanv) + YIndentText,

BaseBlock.Canvas, Action);

DrawBlock(SVG, Blocks[0]);

end;

end;

procedure DrawBlock(const SVG: TStringList; const ABlock: TBlock);

var

I: Integer;

begin

for I := 0 to ABlock.Statements.Count - 1 do

case ABlock.Statements[I].GetSerialNumber of

0: DrawProcess(SVG, TProcessStatement(ABlock.Statements[I]));

1: DrawIfBranching(SVG, TIfBranching(ABlock.Statements[I]));

2: DrawCaseBranching(SVG, TCaseBranching(ABlock.Statements[I]));

3: DrawFirstLoop(SVG, TFirstLoop(ABlock.Statements[I]));

4: DrawLastLoop(SVG, TLastLoop(ABlock.Statements[I]));

end;

end;

procedure SaveSVGFile(const ABlockManager: TBlockManager; const AFileName: string);

var

SVG: TStringList;

begin

SVG := TStringList.Create;

try

SetSVGOpenTag(SVG, ABlockManager.MainBlock.XLast + SchemeIndent,

ABlockManager.MainBlock.Statements[ABlockManager.MainBlock.

Statements.Count - 1].GetYBottom + SchemeIndent);

ABlockManager.PaintBox.Canvas.Pen := ABlockManager.Pen;

ABlockManager.PaintBox.Canvas.Font := ABlockManager.Font;

ABlockManager.PaintBox.Canvas.Font.Size := Round(CorrectionToSvg \* ABlockManager.Font.Size);

DrawBlock(SVG, ABlockManager.MainBlock);

SVG.Add('</svg>');

TFile.WriteAllText(AFileName, SVG.Text, TEncoding.UTF8);

finally

SVG.Destroy;

end;

end;

end.

Приложение Н

(обязательное)

Исходный код программы (модуль uStatistics)

unit uStatistics;

interface

uses

System.SysUtils, Windows, uConstants, System.IOUtils, Vcl.Dialogs, uDialogMessages;

type

TUserInfo = record

UserName: ShortString;

LoginTime: TDateTime;

LogoutTime: TDateTime;

GlobalSettingsTime: Integer;

HelpTime: Integer;

FontSettingTime: Integer;

PenSettingTime: Integer;

ChangeActionCount: Integer;

DeleteStatementCount: Integer;

AddStatementCount: Integer;

end;

procedure SaveStatistics(const AUserInfo: TUserInfo);

function LoadStatistics(const AFilePath: string): TUserInfo;

procedure ClearUserInfo(var UserInfo: TUserInfo);

function FormatStatistics(const AUserInfo: TUserInfo): string;

function SecondsBetween(const ANow, AThen: TDateTime): Integer;

function GetWindowsUserName: string;

implementation

const

constMaxCount = 142;

constStatisticsName = 'Statistics';

function SecondsBetween(const ANow, AThen: TDateTime): Integer;

begin

Result := Round((ANow - AThen) \* 86400);

end;

function CountFilesWithExtension(const AFolderPath: string): Integer;

var

SearchRec: TSearchRec;

ResultCode: Integer;

begin

Result := 0;

ResultCode := FindFirst(AFolderPath + '\\*' + constExtStat, faAnyFile, SearchRec);

try

while ResultCode = 0 do

begin

if (SearchRec.Name <> '.') and (SearchRec.Name <> '..') and

(SearchRec.Attr and faDirectory = 0) then

Inc(Result);

ResultCode := FindNext(SearchRec);

end;

finally

FindClose(SearchRec.FindHandle);

end;

end;

procedure DeleteFilesWithExtension(const AFolderPath: string);

var

SearchRec: TSearchRec;

ResultCode: Integer;

begin

ResultCode := FindFirst(AFolderPath + '\\*' + constExtStat, faAnyFile, SearchRec);

try

while ResultCode = 0 do

begin

if (SearchRec.Name <> '.') and (SearchRec.Name <> '..') and

(SearchRec.Attr and faDirectory = 0) then

DeleteFile(PWideChar(AFolderPath + '\' + SearchRec.Name));

ResultCode := FindNext(SearchRec);

end;

finally

FindClose(SearchRec.FindHandle);

end;

end;

procedure SaveStatistics(const AUserInfo: TUserInfo);

var

Count: Integer;

UserInfoFile: file of TUserInfo;

FilePath: string;

begin

FilePath := IncludeTrailingPathDelimiter(ExtractFilePath(ParamStr(0))) + dirAppData;

Count:= CountFilesWithExtension(FilePath);

if Count >= constMaxCount then

begin

DeleteFilesWithExtension(FilePath);

Count:= 0;

end;

FilePath := TPath.Combine(FilePath, constStatisticsName +'\_'+ IntToStr(Count) + constExtStat);

AssignFile(UserInfoFile, FilePath);

Rewrite(UserInfoFile);

try

Write(UserInfoFile, AUserInfo);

finally

CloseFile(UserInfoFile);

end;

end;

function LoadStatistics(const AFilePath: string): TUserInfo;

var

UserInfoFile: file of TUserInfo;

begin

AssignFile(UserInfoFile, AFilePath);

Reset(UserInfoFile);

try

Read(UserInfoFile, Result);

except

ShowMessage(rsErrorFile);

end;

CloseFile(UserInfoFile);

end;

function FormatStatistics(const AUserInfo: TUserInfo): string;

begin

Result := 'User Name: ' + AUserInfo.UserName + sLineBreak;

if AUserInfo.LoginTime = 0 then

Result := Result + 'Login Time: None' + sLineBreak

else

Result := Result + 'Login Time: ' + FormatDateTime('dd/MM/yyyy HH:mm:ss', AUserInfo.LoginTime) + sLineBreak;

if AUserInfo.LogoutTime = 0 then

Result := Result + 'Logout Time: None' + sLineBreak

else

Result := Result + 'Logout Time: ' + FormatDateTime('dd/MM/yyyy HH:mm:ss', AUserInfo.LogoutTime) + sLineBreak;

Result := Result +

'Global Settings Time: ' + IntToStr(AUserInfo.GlobalSettingsTime) + ' seconds' + sLineBreak +

'Help Time: ' + IntToStr(AUserInfo.HelpTime) + ' seconds' + sLineBreak +

'Font Setting Time: ' + IntToStr(AUserInfo.FontSettingTime) + ' seconds' + sLineBreak +

'Pen Setting Time: ' + IntToStr(AUserInfo.PenSettingTime) + ' seconds' + sLineBreak +

'Change Action Count: ' + IntToStr(AUserInfo.ChangeActionCount) + sLineBreak +

'Delete Statement Count: ' + IntToStr(AUserInfo.DeleteStatementCount) + sLineBreak +

'Add Statement Count: ' + IntToStr(AUserInfo.AddStatementCount);

end;

function GetWindowsUserName: string;

var

UserName: array[0..255] of Char;

UserNameLen: DWORD;

begin

UserNameLen := SizeOf(UserName);

if GetUserName(UserName, UserNameLen) then

Result := UserName

else

Result := '';

end;

procedure ClearUserInfo(var UserInfo: TUserInfo);

begin

FillChar(UserInfo, SizeOf(UserInfo), 0);

end;

end.

Приложение О

(обязательное)

Исходный код программы (модуль uDetermineDimensions)

unit uDetermineDimensions;

interface

uses

Vcl.graphics, System.Types, System.SysUtils, uMinMaxInt, uConstants,

uAdditionalTypes;

function GetTextSize(const ACanvas: TCanvas; const AText: string): TSize;

implementation

function GetTextSize(const ACanvas: TCanvas; const AText: string): TSize;

var

Lines: TStringDynArray;

I: Integer;

begin

if AText = '' then

begin

Result.Height:= ACanvas.TextHeight(Space);

Result.Width:= ACanvas.TextWidth(Space);

end

else

begin

Lines := AText.Split([sLineBreak]);

Result.Width := 0;

for I := 0 to High(Lines) do

Result.Width := Max(Result.Width, ACanvas.TextWidth(Lines[I]));

Result.Height:= ACanvas.TextHeight(Space) \* Length(Lines);

end;

end;

end.

Приложение П

(обязательное)

Исходный код программы (модуль uDrawShapes)

unit uDrawShapes;

interface

uses

Vcl.graphics, System.Types, System.SysUtils, uConstants, uAdditionalTypes,

uBase, System.UITypes;

procedure DrawRect(const AXStart, AXLast, AYStart, AYLast : Integer;

const ACanvas: TCanvas);

procedure DrawInvertedTriangle(const AXStart, AXMiddle, AXLast, AYStart,

AYLast : Integer; const ACanvas: TCanvas);

procedure ColorizeRect(const ACanvas: TCanvas; const ARect: TRect; const AColor: TColor);

procedure DrawUnfinishedVertRectForLoop(const AXStart, AXLast, AYStart, AYMiddle,

AYLast: Integer; const ACanvas: TCanvas);

procedure DrawUnfinishedHorRectForLoop(const AXStart, AXMiddle, AXLast,

AYStart, AYLast: Integer; const ACanvas: TCanvas);

procedure DrawLine(const AXStart, AXLast, AYStart, AYLast : Integer; const ACanvas: TCanvas);

procedure DrawText(const ACanvas: TCanvas; const AX, AY: Integer; const AText: string);

function CreateRect(const AStatement: TStatement): TRect; inline;

procedure DrawArrow(const ACanvas: TCanvas; const AX, AStartY, AEndY: Integer;

const AColor: TColor);

implementation

procedure DrawUnfinishedVertRectForLoop(const AXStart, AXLast, AYStart, AYMiddle,

AYLast: Integer; const ACanvas: TCanvas);

begin

ACanvas.MoveTo(AXStart, AYStart);

ACanvas.LineTo(AXStart, AYLast);

ACanvas.LineTo(AXLast, AYLast);

ACanvas.LineTo(AXLast, AYMiddle);

end;

procedure DrawUnfinishedHorRectForLoop(const AXStart, AXMiddle, AXLast,

AYStart, AYLast: Integer; const ACanvas: TCanvas);

begin

ACanvas.MoveTo(AXStart, AYStart);

ACanvas.LineTo(AXLast, AYStart);

ACanvas.LineTo(AXLast, AYLast);

ACanvas.LineTo(AXMiddle, AYLast);

end;

procedure DrawLine(const AXStart, AXLast, AYStart, AYLast : Integer;

const ACanvas: TCanvas);

begin

ACanvas.MoveTo(AXStart, AYStart);

ACanvas.LineTo(AXLast, AYLast);

end;

procedure DrawRect(const AXStart, AXLast, AYStart, AYLast : Integer;

const ACanvas: TCanvas);

begin

ACanvas.MoveTo(AXStart, AYStart);

ACanvas.LineTo(AXLast, AYStart);

ACanvas.LineTo(AXLast, AYLast);

ACanvas.LineTo(AXStart, AYLast);

ACanvas.LineTo(AXStart, AYStart);

end;

procedure DrawInvertedTriangle(const AXStart, AXMiddle, AXLast, AYStart,

AYLast : Integer; const ACanvas: TCanvas);

begin

ACanvas.MoveTo(AXStart, AYStart);

ACanvas.LineTo(AXMiddle, AYLast);

ACanvas.LineTo(AXLast, AYStart);

end;

function CreateRect(const AStatement: TStatement): TRect; inline;

begin

Result:= Rect(AStatement.BaseBlock.XStart, AStatement.YStart,

AStatement.BaseBlock.XLast, AStatement.GetYBottom);

end;

procedure ColorizeRect(const ACanvas: TCanvas; const ARect: TRect; const AColor: TColor);

begin

ACanvas.Brush.Color := AColor;

ACanvas.FillRect(ARect);

end;

procedure DrawText(const ACanvas: TCanvas; const AX, AY: Integer; const AText: string);

var

Lines: TStringDynArray;

Indent: Integer;

I: Integer;

begin

ACanvas.Brush.Style := bsClear;

Lines := AText.Split([sLineBreak]);

Indent := ACanvas.TextHeight(Space);

for I := 0 to High(Lines) do

ACanvas.TextOut(AX, AY + I \* Indent, Lines[I]);

end;

procedure DrawArrow(const ACanvas: TCanvas; const AX, AStartY, AEndY: Integer;

const AColor: TColor);

var

Points: array [0..2] of TPoint;

Offset, ArrowHeight, ArrowWidth: Integer;

begin

ACanvas.Brush.Color:= AColor;

ArrowHeight := Abs(AEndY - AStartY);

ArrowWidth := ArrowHeight div 2;

if AStartY > AEndY then

Offset := -ArrowHeight

else

Offset := ArrowHeight;

Points[0] := Point(AX, AStartY);

Points[1] := Point(AX - ArrowWidth, AStartY + Offset);

Points[2] := Point(AX + ArrowWidth, AStartY + Offset);

ACanvas.Polygon(Points);

end;

end.

Приложение Р

(обязательное)

Исходный код программы (модуль uCommands)

unit uCommands;

interface

uses

uAdditionalTypes, uBase, uStack, uCaseBranching, uMinMaxInt;

type

ICommand = interface

procedure Execute;

procedure Undo;

end;

{ TCommnadChangeContent }

TCommnadChangeContent = class(TInterfacedObject, ICommand)

private

FAction: string;

FConds: TStringArr;

FStatement: TStatement;

public

constructor Create(const AStatement: TStatement; const AAct: String;

const AConds: TStringArr);

procedure Execute;

procedure Undo;

End;

{ TCommandAddStatement }

TCommandAddStatement = class(TInterfacedObject, ICommand)

private

FNewStatement: TStatement;

FBaseBlock: TBlock;

FIndex : Integer;

public

constructor Create(const ABaseBlock: TBlock; const AIndex : Integer;

const ANewStatement: TStatement);

procedure Execute;

procedure Undo;

destructor Destroy; override;

End;

{ TCommandDel }

TCommandDelStatement = class(TInterfacedObject, ICommand)

private

FBaseBlock: TBlock;

FStatement: TStatement;

FIndex : Integer;

public

constructor Create(const AStatement: TStatement);

procedure Execute;

procedure Undo;

destructor Destroy; override;

End;

{ TCommandAddBlock }

TCommandAddBlock = class(TInterfacedObject, ICommand)

private

FInsertedBlock, FBaseBlock: TBlock;

FIndex, FHigh: Integer;

public

constructor Create(const ABaseBlock: TBlock; const AIndex : Integer;

const AInsertedBlock: TBlock);

procedure Execute;

procedure Undo;

destructor Destroy; override;

End;

{ TCommandCaseSort }

TCommandCaseSort = class(TInterfacedObject, ICommand)

private

FCaseBranching : TCaseBranching;

FSortNumber: Integer;

FPrevConds: TStringArr;

FPrevBlocks: TBlockArr;

public

constructor Create(const ACaseBranching: TCaseBranching; const ASortNumber : Integer);

procedure Execute;

procedure Undo;

End;

{ TCommandTransferAnotherBlock }

TCommandTransferAnotherBlock = class(TInterfacedObject, ICommand)

private

FCommandAddStatement : TCommandAddStatement;

FCommandDelStatement : TCommandDelStatement;

FOldBaseBlock : TBlock;

public

constructor Create(const AHoveredStatement : TStatement; const isAfter: Boolean;

const AStatement: TStatement);

procedure Execute;

procedure Undo;

end;

{ TCommandSwapStatements }

TCommandSwapStatements = class(TInterfacedObject, ICommand)

private

FFirstStatement, FSecondStatement : TStatement;

FFirstIndex, FSecondIndex: Integer;

procedure SortStatements;

public

constructor Create(const AFirstStatement, ASecondStatement: TStatement);

procedure Execute;

procedure Undo;

end;

implementation

{ TChangeContent }

constructor TCommnadChangeContent.Create(const AStatement: TStatement; const AAct: String;

const AConds: TStringArr);

begin

FAction:= AAct;

FConds:= AConds;

FStatement:= AStatement;

end;

procedure TCommnadChangeContent.Execute;

var

PrevAction: string;

begin

PrevAction:= FStatement.Action;

if FConds = nil then

FStatement.ChangeAction(FAction)

else

begin

var CaseBranching: TCaseBranching:= TCaseBranching(FStatement);

var FPrevConds: TStringArr := CaseBranching.Conds;

CaseBranching.ChangeActionWithConds(FAction, FConds);

FConds:= FPrevConds;

end;

FAction:= PrevAction;

end;

procedure TCommnadChangeContent.Undo;

begin

Execute;

end;

{ TCommandAdd }

destructor TCommandAddStatement.Destroy;

begin

FNewStatement.DecRefCount;

if (FNewStatement.BaseBlock = nil) and (FNewStatement.RefCount = 0) then

FNewStatement.Destroy;

inherited;

end;

constructor TCommandAddStatement.Create(const ABaseBlock: TBlock; const AIndex : Integer;

const ANewStatement: TStatement);

begin

ANewStatement.IncRefCount;

FNewStatement:= ANewStatement;

FIndex:= AIndex;

FBaseBlock:= ABaseBlock;

end;

procedure TCommandAddStatement.Execute;

begin

FBaseBlock.InsertWithResizing(FIndex, FNewStatement);

end;

procedure TCommandAddStatement.Undo;

var

WasDefaultStatementRemoved: Boolean;

begin

WasDefaultStatementRemoved:= FIndex >= FBaseBlock.Statements.Count;

Dec(FIndex, Ord(WasDefaultStatementRemoved));

FBaseBlock.ExtractStatementAt(FIndex);

FBaseBlock.Install(FIndex - Ord(FIndex = FBaseBlock.Statements.Count));

Inc(FIndex, Ord(WasDefaultStatementRemoved));

end;

{ TCommandDel }

destructor TCommandDelStatement.Destroy;

begin

FStatement.DecRefCount;

if (FStatement.BaseBlock = nil) and (FStatement.RefCount = 0) then

FStatement.Destroy;

inherited;

end;

constructor TCommandDelStatement.Create(const AStatement: TStatement);

begin

AStatement.IncRefCount;

FStatement:= AStatement;

FBaseBlock:= AStatement.BaseBlock;

end;

procedure TCommandDelStatement.Execute;

begin

FIndex:= FStatement.BaseBlock.Extract(FStatement);

FBaseBlock.Install(FIndex - Ord(FIndex = FBaseBlock.Statements.Count));

end;

procedure TCommandDelStatement.Undo;

begin

FBaseBlock.InsertWithResizing(FIndex, FStatement);

end;

{ TCommandAddBlock }

destructor TCommandAddBlock.Destroy;

var

I: Integer;

WasDefaultStatementRemoved: Boolean;

begin

if FInsertedBlock.Statements.Count = 0 then

begin

WasDefaultStatementRemoved:= FIndex >= FBaseBlock.Statements.Count;

Dec(FIndex, Ord(WasDefaultStatementRemoved));

Dec(FHigh, Ord(WasDefaultStatementRemoved));

for I := FIndex to FIndex + FHigh do

FBaseBlock.Statements[I].DecRefCount;

end

else

for I := 0 to FHigh do

FInsertedBlock.Statements[I].DecRefCount;

FInsertedBlock.Destroy;

inherited;

end;

constructor TCommandAddBlock.Create(const ABaseBlock: TBlock; const AIndex : Integer;

const AInsertedBlock: TBlock);

var

I: Integer;

begin

for I := 0 to AInsertedBlock.Statements.Count - 1 do

AInsertedBlock.Statements[I].IncRefCount;

FBaseBlock:= ABaseBlock;

FIndex:= AIndex;

FInsertedBlock:= AInsertedBlock;

FHigh:= FInsertedBlock.Statements.Count - 1;

end;

procedure TCommandAddBlock.Execute;

begin

FBaseBlock.InsertBlock(FIndex, FInsertedBlock);

FInsertedBlock.Statements.Clear;

end;

procedure TCommandAddBlock.Undo;

var

I: Integer;

WasDefaultStatementRemoved: Boolean;

begin

WasDefaultStatementRemoved:= FIndex >= FBaseBlock.Statements.Count;

Dec(FIndex, Ord(WasDefaultStatementRemoved));

for I := 0 to FHigh do

FInsertedBlock.Statements.Add(FBaseBlock.ExtractStatementAt(FIndex + I));

FBaseBlock.Install(FIndex - Ord(FIndex = FBaseBlock.Statements.Count));

Inc(FIndex, Ord(WasDefaultStatementRemoved));

end;

{ TCommandCaseSort }

constructor TCommandCaseSort.Create(const ACaseBranching: TCaseBranching; const ASortNumber : Integer);

begin

FCaseBranching:= ACaseBranching;

FSortNumber:= ASortNumber;

end;

procedure TCommandCaseSort.Execute;

begin

FPrevConds:= Copy(FCaseBranching.Conds);

FPrevBlocks:= Copy(FCaseBranching.Blocks);

FCaseBranching.SortConditions(FSortNumber);

end;

procedure TCommandCaseSort.Undo;

begin

FCaseBranching.RestoreСonditions(FPrevConds, FPrevBlocks);

end;

{ TCommandTransferAnotherBlock }

constructor TCommandTransferAnotherBlock.Create(const AHoveredStatement : TStatement;

const isAfter: Boolean; const AStatement: TStatement);

var

NewIndex, OldIndex: Integer;

begin

NewIndex:= AHoveredStatement.BaseBlock.

FindStatementIndex(AHoveredStatement.YStart);

FOldBaseBlock:= AStatement.BaseBlock;

OldIndex := FOldBaseBlock.FindStatementIndex(AStatement.YStart);

if AHoveredStatement.BaseBlock = FOldBaseBlock then

case isAfter of

True:

Inc(NewIndex, Ord(OldIndex - 1 >= NewIndex));

False:

Dec(NewIndex, Ord(OldIndex + 1 <= NewIndex));

end

else

Inc(NewIndex, Ord(isAfter));

FCommandAddStatement:= TCommandAddStatement.Create(AHoveredStatement.BaseBlock,

NewIndex, AStatement);

FCommandDelStatement := TCommandDelStatement.Create(AStatement);

end;

procedure TCommandTransferAnotherBlock.Execute;

begin

FCommandDelStatement.Execute;

if FCommandAddStatement.FNewStatement is TOperator then

begin

var CurrOperator: TOperator := TOperator(FCommandAddStatement.FNewStatement);

CurrOperator.MoveRightChildrens(FCommandAddStatement.FBaseBlock.XStart -

FOldBaseBlock.XStart);

CurrOperator.SetXLastForChildrens(FCommandAddStatement.FBaseBlock.XLast);

end;

FCommandAddStatement.Execute;

end;

procedure TCommandTransferAnotherBlock.Undo;

begin

FCommandAddStatement.Undo;

if FCommandAddStatement.FNewStatement is TOperator then

begin

var CurrOperator: TOperator := TOperator(FCommandAddStatement.FNewStatement);

CurrOperator.MoveRightChildrens(FOldBaseBlock.XStart -

FCommandAddStatement.FBaseBlock.XStart);

CurrOperator.SetXLastForChildrens(FOldBaseBlock.XLast);

end;

FCommandDelStatement.FStatement.BaseBlock := FOldBaseBlock;

FCommandDelStatement.Undo;

end;

{ TCommandSwapStatements }

constructor TCommandSwapStatements.Create(const AFirstStatement,

ASecondStatement: TStatement);

begin

FFirstStatement := AFirstStatement;

FFirstIndex := FFirstStatement.BaseBlock.FindStatementIndex(FFirstStatement.YStart);

FSecondStatement := ASecondStatement;

FSecondIndex := FSecondStatement.BaseBlock.FindStatementIndex(FSecondStatement.YStart);

end;

procedure TCommandSwapStatements.SortStatements;

var

TempStatement: TStatement;

TempIndex: Integer;

begin

if FFirstStatement.BaseBlock.XStart < FSecondStatement.BaseBlock.XStart then

begin

TempStatement := FFirstStatement;

FFirstStatement := FSecondStatement;

FSecondStatement:= TempStatement;

TempIndex := FFirstIndex;

FFirstIndex := FSecondIndex;

FSecondIndex := TempIndex;

end;

end;

procedure TCommandSwapStatements.Execute;

var

SecondBaseBlock: TBlock;

CurrOperator: TOperator;

Offset: Integer;

TempIndex: Integer;

begin

SortStatements;

SecondBaseBlock := FSecondStatement.BaseBlock;

Offset := FFirstStatement.BaseBlock.XStart - SecondBaseBlock.XStart;

FFirstStatement.BaseBlock.AssignStatement(FFirstIndex, FSecondStatement);

SecondBaseBlock.AssignStatement(FSecondIndex, FFirstStatement);

TempIndex := FFirstIndex;

FFirstIndex := FSecondIndex;

FSecondIndex := TempIndex;

if FFirstStatement is TOperator then

begin

CurrOperator := TOperator(FFirstStatement);

CurrOperator.MoveRightChildrens(-Offset);

CurrOperator.SetXLastForChildrens(CurrOperator.BaseBlock.XLast);

end;

if FSecondStatement is TOperator then

begin

CurrOperator := TOperator(FSecondStatement);

CurrOperator.MoveRightChildrens(Offset);

CurrOperator.SetXLastForChildrens(CurrOperator.BaseBlock.XLast);

end;

FFirstStatement.SwapYStart(FSecondStatement);

SecondBaseBlock.Install(FFirstIndex);

FSecondStatement.BaseBlock.Install(FSecondIndex);

end;

procedure TCommandSwapStatements.Undo;

begin

Execute;

end;

end.

Приложение С

(обязательное)

Исходный код программы (модуль uSwitchStatements)

unit uSwitchStatements;

interface

uses

uBase, uAdditionalTypes;

type

TDirection = (ForwardDir = 1, BackwardDir = -1);

procedure SetHorizontalMovement(out AStatement: TStatement; const AMainBlock: TBlock;

const ADirection: TDirection);

procedure SetVerticalMovement(out AStatement: TStatement; const AMainBlock: TBlock;

const ADirection: TDirection);

implementation

procedure SetHorizontalMovement(out AStatement: TStatement; const AMainBlock: TBlock;

const ADirection: TDirection);

var

LastBlock: Integer;

BlockIndex, StatementIndex : Integer;

begin

if (AStatement = nil) or (AStatement.BaseBlock.BaseOperator = nil) then

AStatement:= AMainBlock.Statements[0]

else

begin

case ADirection of

BackwardDir: LastBlock:= 0;

ForwardDir: LastBlock:= High(AStatement.BaseBlock.BaseOperator.Blocks);

end;

BlockIndex:= AStatement.BaseBlock.BaseOperator.

FindBlockIndex(AStatement.BaseBlock.XStart);

if BlockIndex = LastBlock then

AStatement:= AStatement.BaseBlock.BaseOperator

else

begin

StatementIndex:= AStatement.BaseBlock.FindStatementIndex(

AStatement.YStart);

if AStatement.BaseBlock.BaseOperator.Blocks[BlockIndex +

Ord(ADirection)].Statements.Count <= StatementIndex then

StatementIndex:= AStatement.BaseBlock.BaseOperator.

Blocks[BlockIndex + Ord(ADirection)].Statements.Count - 1;

AStatement:= AStatement.BaseBlock.

BaseOperator.Blocks[BlockIndex + Ord(ADirection)].Statements[StatementIndex];

end;

end;

end;

procedure SetVerticalMovement(out AStatement: TStatement; const AMainBlock: TBlock;

const ADirection: TDirection);

var

StatementIndex: Integer;

CurrStatement: TStatement;

Blocks: TBlockArr;

begin

if AStatement = nil then

AStatement:= AMainBlock.Statements[0]

else

begin

StatementIndex:= AStatement.BaseBlock.FindStatementIndex(

AStatement.YStart);

case ADirection of

BackwardDir:

begin

if StatementIndex > 0 then

begin

AStatement:= AStatement.BaseBlock.Statements[StatementIndex + Ord(ADirection)];

if AStatement is TOperator then

begin

Blocks:= TOperator(AStatement).Blocks;

AStatement:= Blocks[High(Blocks)].Statements.GetLast;

end;

end

else if AStatement.BaseBlock.BaseOperator = nil then

AStatement:= AStatement.BaseBlock.Statements[0]

else

AStatement:= AStatement.BaseBlock.BaseOperator;

end;

ForwardDir:

begin

if AStatement is TOperator then

AStatement:= TOperator(AStatement).Blocks[0].Statements[0]

else if StatementIndex < AStatement.BaseBlock.Statements.Count - 1 then

AStatement:= AStatement.BaseBlock.Statements[StatementIndex + Ord(ADirection)]

else

begin

if AStatement.BaseBlock.BaseOperator <> nil then

begin

CurrStatement := AStatement;

repeat

CurrStatement := CurrStatement.BaseBlock.BaseOperator;

StatementIndex := CurrStatement.BaseBlock.FindStatementIndex(

CurrStatement.YStart);

if StatementIndex < CurrStatement.BaseBlock.Statements.Count - 1 then

begin

AStatement := CurrStatement.BaseBlock.Statements[StatementIndex + Ord(ADirection)];

Exit;

end;

until CurrStatement.BaseBlock.BaseOperator = nil;

AStatement := AStatement.BaseBlock.BaseOperator;

end;

end;

end;

end;

end;

end;

end.

Приложение Т

(обязательное)

Исходный код программы (модуль uStatementSearch)

unit uStatementSearch;

interface

uses

uBase;

function BinarySearchStatement(const AX, AY: Integer; const ABlock: TBlock): TStatement;

function BinarySearchBlock(const Blocks: TBlockArr; const AX: Integer): TBlock;

implementation

function BinarySearchBlock(const Blocks: TBlockArr; const AX: Integer): TBlock;

var

L, R, M: Integer;

begin

Result := nil;

L := 0;

R := High(Blocks);

while L <= R do

begin

M := (L + R) shr 1;

if (AX >= Blocks[M].XStart) and (AX <= Blocks[M].XLast) then

Exit(Blocks[M])

else if AX < Blocks[M].XStart then

R := M - 1

else

L := M + 1;

end;

end;

function BinarySearchStatement(const AX, AY: Integer; const ABlock: TBlock): TStatement;

var

L, R, M: Integer;

CurrOperator: TOperator;

CurrStatement: TStatement;

begin

Result := nil;

if (AX >= ABlock.XStart) and (AX <= ABlock.XLast) then

begin

L := 0;

R := ABlock.Statements.Count - 1;

while L <= R do

begin

M := (L + R) shr 1;

CurrStatement := ABlock.Statements[M];

if (AY >= CurrStatement.YStart) and (AY <= CurrStatement.GetYBottom) then

begin

if CurrStatement is TOperator then

begin

CurrOperator:= TOperator(CurrStatement);

case CurrOperator.IsPreсOperator of

True:

if AY <= CurrOperator.YLast then

Exit(CurrStatement);

False:

if AY >= CurrOperator.Blocks[0].Statements.GetLast.GetYBottom then

Exit(CurrStatement);

end;

if AX <= CurrOperator.BaseBlock.XStart + CurrOperator.GetOffsetFromXStart then

Exit(CurrStatement);

Exit(BinarySearchStatement(AX, AY, BinarySearchBlock(CurrOperator.Blocks, AX)));

end

else

Exit(CurrStatement);

end

else if AY < CurrStatement.YStart then

R := M - 1

else

L := M + 1;

end;

end

end;

end.

Приложение У

(обязательное)

Исходный код программы (модуль uCaseBlockSorting)

unit uCaseBlockSorting;

interface

uses

uAdditionalTypes, uStack, uBase;

type

TCompareFunction = function(const AFirstStr, ASecondStr: String): Boolean;

procedure QuickSort(const AStr: TStringArr; const ABlocks : TBlockArr;

const ACompare: TCompareFunction);

function CompareStrAsc(const AFirstStr, ASecondStr: string): Boolean;

function CompareStrDesc(const AFirstStr, ASecondStr: string): Boolean;

implementation

procedure QuickSort(const AStr: TStringArr; const ABlocks : TBlockArr;

const ACompare: TCompareFunction);

type

TIndexRange = record

LeftIndex: Integer;

RightIndex: Integer;

end;

var

I, J: Integer;

Pivot: string;

TempStr: string;

TempBlock: TBlock;

Stack: TStack<TIndexRange>;

IndexRange: TIndexRange;

begin

Stack := TStack<TIndexRange>.Create;

IndexRange.LeftIndex := Low(AStr);

IndexRange.RightIndex := High(AStr);

Stack.Push(IndexRange);

while Stack.Count > 0 do

begin

IndexRange := Stack.Pop;

Pivot := AStr[(IndexRange.LeftIndex + IndexRange.RightIndex) shr 1];

I := IndexRange.LeftIndex;

J := IndexRange.RightIndex;

repeat

while ACompare(AStr[I], Pivot) do

Inc(I);

while ACompare(Pivot, AStr[J]) do

Dec(J);

if I <= J then

begin

TempStr := AStr[I];

AStr[I] := AStr[J];

AStr[J] := TempStr;

TempBlock := ABlocks[I];

ABlocks[I] := ABlocks[J];

ABlocks[J] := TempBlock;

Inc(I);

Dec(J);

end;

until I > J;

if IndexRange.LeftIndex < J then

begin

I:= IndexRange.RightIndex;

IndexRange.RightIndex := J;

Stack.Push(IndexRange);

IndexRange.LeftIndex := IndexRange.RightIndex + 1;

IndexRange.RightIndex := I;

Stack.Push(IndexRange);

end

else if IndexRange.RightIndex > I then

begin

J := IndexRange.LeftIndex;

IndexRange.LeftIndex := I;

Stack.Push(IndexRange);

IndexRange.RightIndex := IndexRange.LeftIndex - 1;

IndexRange.LeftIndex := J;

Stack.Push(IndexRange);

end;

end;

Stack.Destroy;

end;

function CompareStrAsc(const AFirstStr, ASecondStr: string): Boolean;

begin

if Length(AFirstStr) = Length(ASecondStr) then

Result := AFirstStr < ASecondStr

else

Result := Length(AFirstStr) < Length(ASecondStr);

end;

function CompareStrDesc(const AFirstStr, ASecondStr: string): Boolean;

begin

if Length(AFirstStr) = Length(ASecondStr) then

Result := AFirstStr > ASecondStr

else

Result := Length(AFirstStr) > Length(ASecondStr);

end;

end.

Приложение Ф

(обязательное)

Исходный код программы (модуль uBase)

unit uBase;

interface

uses

Vcl.graphics, uArrayList, uMinMaxInt, uDetermineDimensions, System.Types,

uAdditionalTypes;

type

TBlock = class;

{ TBaseStatement }

// Define abstract class TStatement

// This class is a base class for all statements and is abstract

// DefaultSymbol is a constant field used to represent an unknown value

// in the statement

TStatement = class abstract

private

FRefCount : Integer;

protected

// FYStart and FYLast are used to store the Y position of the statement

FYStart, FYLast: Integer;

// FAction stores the text of the statement

FAction: String;

FActionSize: TSize;

FYIndentText, FXMinIndentText: Integer;

// FBaseBlock is a reference to the block that the statement belongs to

FBaseBlock: TBlock;

// Set the bottommost part

procedure SetYBottom(const AYBottom: Integer); virtual;

// Get the optimal lower part

function GetMaxOptimalYBottom: Integer; virtual;

// Lowers the statement on Offset

procedure Lower(const AOffset: Integer);

// Returns whether the current Y last is optimal

function HasOptimalYLast: boolean;

// Get the optimal Y last

function GetOptimalYLast: Integer; virtual; abstract;

// Returns the optimal block width

function GetOptimaWidth: Integer; virtual; abstract;

procedure RedefineStatement; virtual;

procedure SetTextSize; virtual;

procedure SetActionSizes;

// These methods are abstract and will be implemented by subclasses to draw

procedure Draw; virtual; abstract;

procedure Initialize; virtual;

public

property RefCount: Integer read FRefCount;

// Create

constructor Create(const AAction : String; const ABaseBlock: TBlock); overload;

constructor Create(const AAction : String); overload; virtual;

// These properties return the text of the statement and base block

property Action: String read FAction;

property ActionSize: TSize read FActionSize;

property BaseBlock: TBlock read FBaseBlock write FBaseBlock;

// Returnts the Y statrt coordinate

property YStart: Integer read FYStart;

property YLast: Integer read FYLast;

property YIndentText: Integer read FYIndentText;

procedure SetCoords(const AYStart, AYLast: Integer);

// Returns the Y coordinate of the bottommost part

function GetYBottom: Integer; virtual;

// Change action

procedure ChangeAction(const AAction: String);

// Set the optimal Y last

procedure SetOptimalYLast;

function GetSerialNumber: Integer; virtual; abstract;

procedure SwapYStart(const AStatement: TStatement);

function Clone: TStatement; virtual;

function GetMask(const AVisibleImageRect: TVisibleImageRect;

const isTOperator: Boolean): Integer; inline;

procedure IncRefCount; inline;

procedure DecRefCount; inline;

end;

{ TStatementClass }

TStatementClass = class of TStatement;

{ TBlockArr }

TBlockArr = array of TBlock;

{ TOperator }

TOperator = class abstract(TStatement)

protected

FBlocks: TBlockArr;

procedure CreateBlock; virtual; abstract;

procedure SetYBottom(const AYBottom: Integer); override;

function GetMaxOptimalYBottom: Integer; override;

function GetOptimalWidthForBlock(const ABlock: TBlock): Integer; virtual; abstract;

function GetBlockYStart: Integer;

procedure DrawBlocks(const AVisibleImageRect: TVisibleImageRect);

procedure RedefineStatement; override;

procedure Initialize; override;

procedure InitializeBlocks(StartIndex: Integer = 0);

procedure InstallCanvas(const ACanvas: TCanvas);

procedure SetBlockTextSize;

procedure AlignBlocksToXStart;

public

constructor Create(const AAction : String); override;

destructor Destroy; override;

function IsPreсOperator : Boolean; virtual; abstract;

function GetYBottom: Integer; override;

property Blocks: TBlockArr read FBlocks;

function Clone: TStatement; override;

function FindBlockIndex(const AXStart: Integer): Integer;

function GetOffsetFromXStart: Integer; virtual;

procedure MoveRightChildrens(const AOffset : Integer);

procedure MoveDownChildrens(const AOffset : Integer);

procedure SetXLastForChildrens(const AXLast : Integer);

procedure AlignBlocks;

end;

{ TBlock }

TBlock = class

private

// FCanvas is a reference to the canvas used for drawing

FCanvas: TCanvas;

FXStart, FXLast: Integer;

FStatements: TArrayList<TStatement>;

FBaseOperator: TOperator;

procedure MoveRightExceptXLast(const AOffset: Integer);

function GetLastStatement: TStatement;

function Clone(const ABaseOperator: TOperator): TBlock;

procedure Insert(var AIndex: Integer;const AInsertedStatement: TStatement);

procedure RemoveStatementAt(const Index: Integer);

procedure ChangeXStartBlock(const ANewXStart: Integer);

// After changing the Y coordinate, need to call the procedure in order to

// change the Y coordinates of others

procedure FixYInBlock(const Index: Integer);

procedure PromptFixYInBaseBlocks;

procedure FixYStatement(AIndex: Integer = 0);

procedure RedefineBlock(const AIndex: Integer = 0);

public

constructor Create(const ABaseOperator: TOperator); overload;

constructor Create(const AXStart: Integer; const ACanvas: TCanvas); overload;

constructor Create(const AXStart: Integer; const ABaseOperator: TOperator); overload;

constructor Create(const AXStart, AXLast: Integer; const ABaseOperator: TOperator;

const ACanvas: TCanvas); overload;

destructor Destroy;

property XStart: Integer read FXStart;

property XLast: Integer read FXLast;

property Canvas: TCanvas read FCanvas;

property BaseOperator: TOperator read FBaseOperator;

property Statements: TArrayList<TStatement> read FStatements;

procedure InsertBlock(AIndex: Integer; const AInsertedBlock: TBlock);

procedure InsertWithResizing(AIndex: Integer;const AInsertedStatement: TStatement);

procedure AddUnknownStatement(const AStatement: TStatement; const AYStart: Integer);

procedure AddStatement(const AStatement: TStatement);

procedure AssignStatement(const AIndex: Integer; const AStatement : TStatement);

function Extract(const AStatement: TStatement): Integer;

function ExtractStatementAt(const AIndex: Integer) : TStatement;

procedure MoveRight(const AOffset: Integer);

procedure MoveDown(const AOffset: Integer);

procedure ChangeXLastBlock(const ANewXLast: Integer);

function FindOptimalXLast: Integer;

procedure SetOptimalXLastBlock;

procedure DrawBlock(const AVisibleImageRect: TVisibleImageRect);

procedure Assign(const ASource: TBlock);

function FindStatementIndex(const AFYStart: Integer): Integer;

procedure SetNewActionForDefaultStatements(const AOldDefaultAction: string);

procedure RedefineSizes;

procedure AdjustStatements;

// Set the dimensions after adding and if this statement is the last one,

// it asks the previous to set the optimal height

procedure Install(const Index: Integer);

function GetMask(const AVisibleImageRect: TVisibleImageRect): Integer; inline;

procedure InstallCanvas(const ACanvas: TCanvas);

end;

var

DefaultStatement: TStatementClass = nil;

DefaultAction : string;

function isDefaultStatement(const AStatement: TStatement): Boolean;

implementation

function isDefaultStatement(const AStatement: TStatement): Boolean;

begin

Result:= (AStatement is DefaultStatement) and

(AStatement.FAction = DefaultAction);

end;

{ TStatement }

procedure TStatement.IncRefCount;

begin

Inc(FRefCount);

end;

procedure TStatement.DecRefCount;

begin

Dec(FRefCount);

end;

constructor TStatement.Create(const AAction : String; const ABaseBlock: TBlock);

begin

FRefCount := 0;

FBaseBlock:= ABaseBlock;

Create(AAction);

end;

constructor TStatement.Create(const AAction : String);

begin

FRefCount := 0;

FAction := AAction;

end;

procedure TStatement.SetTextSize;

const

Stock = 5;

begin

SetActionSizes;

FYIndentText:= BaseBlock.FCanvas.Font.Size + BaseBlock.FCanvas.Pen.Width + Stock;

FXMinIndentText:= BaseBlock.FCanvas.Font.Size + BaseBlock.FCanvas.Pen.Width + Stock;

end;

procedure TStatement.SetActionSizes;

begin

FActionSize:= GetTextSize(BaseBlock.Canvas, FAction);

end;

procedure TStatement.RedefineStatement;

begin

SetTextSize;

SetOptimalYLast;

end;

function TStatement.HasOptimalYLast : Boolean;

begin

Result:= FYLast = GetOptimalYLast;

end;

procedure TStatement.ChangeAction(const AAction: String);

begin

FAction := AAction;

SetActionSizes;

SetOptimalYLast;

BaseBlock.SetOptimalXLastBlock;

BaseBlock.FixYInBlock(BaseBlock.FindStatementIndex(FYStart));

BaseBlock.PromptFixYInBaseBlocks;

end;

procedure TStatement.SetCoords(const AYStart, AYLast: Integer);

begin

Self.FYStart := AYStart;

Self.FYLast := AYLast;

end;

function TStatement.GetYBottom: Integer;

begin

Result:= FYLast;

end;

procedure TStatement.SetYBottom(const AYBottom: Integer);

begin

FYLast:= AYBottom;

end;

function TStatement.GetMaxOptimalYBottom: Integer;

begin

Result:= GetOptimalYLast;

end;

procedure TStatement.SetOptimalYLast;

begin

FYLast := GetOptimalYLast;

end;

procedure TStatement.Lower(const AOffset: Integer);

begin

Inc(FYStart, AOffset);

Inc(FYLast, AOffset);

end;

procedure TStatement.Initialize;

begin

SetOptimalYLast;

BaseBlock.SetOptimalXLastBlock;

end;

procedure TStatement.SwapYStart(const AStatement: TStatement);

var

Temp: Integer;

begin

Temp := Self.YStart;

Self.FYStart := AStatement.FYStart;

AStatement.FYStart := Temp;

end;

function TStatement.Clone: TStatement;

begin

Result:= TStatementClass(Self.ClassType).Create(DefaultAction, Self.BaseBlock);

Result.FAction:= Self.FAction;

Result.FActionSize:= Self.FActionSize;

Result.FYIndentText := Self.FYIndentText;

Result.FXMinIndentText := Self.FXMinIndentText;

Result.FYStart:= Self.FYStart;

Result.FYLast:= Self.FYLast;

end;

function TStatement.GetMask(const AVisibleImageRect: TVisibleImageRect;

const isTOperator: Boolean): Integer;

var

YLast: Integer;

begin

if isTOperator and (TOperator(Self).GetOffsetFromXStart <> 0) then

YLast := GetYBottom

else

YLast:= FYLast;

Result :=

{X--- : }

Ord(FYStart >= AVisibleImageRect.FTopLeft.Y) shl 3 or

{-X-- : }

Ord(YLast <= AVisibleImageRect.FBottomRight.Y) shl 2 or

{--X- : }

Ord(FYStart <= AVisibleImageRect.FBottomRight.Y) shl 1 or

{---X : }

Ord(YLast >= AVisibleImageRect.FTopLeft.Y);

end;

{ TBlock }

destructor TBlock.Destroy;

var

I: Integer;

begin

for I := 0 to FStatements.Count - 1 do

if FStatements[I].FRefCount = 0 then

FStatements[I].Destroy;

FStatements.Destroy;

inherited;

end;

constructor TBlock.Create(const ABaseOperator: TOperator);

begin

FStatements := TArrayList<TStatement>.Create(14);

FBaseOperator := ABaseOperator;

end;

constructor TBlock.Create(const AXStart: Integer; const ACanvas: TCanvas);

begin

FStatements := TArrayList<TStatement>.Create(14);

FCanvas := ACanvas;

FXStart := AXStart;

end;

constructor TBlock.Create(const AXStart: Integer; const ABaseOperator: TOperator);

begin

Create(ABaseOperator);

FXStart := AXStart;

end;

constructor TBlock.Create(const AXStart, AXLast: Integer; const ABaseOperator: TOperator;

const ACanvas: TCanvas);

begin

Create(AXStart, ABaseOperator);

FXLast := AXLast;

FCanvas := ACanvas;

end;

procedure TBlock.RedefineBlock(const AIndex: Integer = 0);

var

I: Integer;

begin

for I := AIndex to FStatements.Count - 1 do

FStatements[I].RedefineStatement;

SetOptimalXLastBlock;

end;

procedure TBlock.RedefineSizes;

var

I, Offset : Integer;

begin

RedefineBlock;

for I := 1 to FStatements.Count - 1 do

begin

Offset:= FStatements[I - 1].GetYBottom - FStatements[I].FYStart;

FStatements[I].Lower(Offset);

if FStatements[I] is TOperator then

TOperator(FStatements[I]).MoveDownChildrens(Offset);

end;

end;

procedure TBlock.Insert(var AIndex: Integer; const AInsertedStatement: TStatement);

begin

AInsertedStatement.FBaseBlock:= Self;

AInsertedStatement.SetTextSize;

FStatements.Insert(AInsertedStatement, AIndex);

if (FStatements.Count = 2) and (isDefaultStatement(Statements[AIndex xor 1])) then

begin

AInsertedStatement.FYStart:= Statements[AIndex xor 1].FYStart;

Self.RemoveStatementAt(AIndex xor 1);

AIndex:= 0;

end

else if AIndex = FStatements.Count - 1 then

begin

FStatements[AIndex - 1].

SetYBottom(FStatements[AIndex - 1].GetMaxOptimalYBottom);

AInsertedStatement.FYStart:= Statements[AIndex - 1].GetYBottom;

end

else if AIndex <> 0 then

AInsertedStatement.FYStart:= Statements[AIndex - 1].GetYBottom

else

AInsertedStatement.FYStart:= Statements[AIndex + 1].FYStart;

end;

procedure TBlock.InsertWithResizing(AIndex: Integer; const AInsertedStatement: TStatement);

begin

Insert(AIndex, AInsertedStatement);

AInsertedStatement.Initialize;

FixYInBlock(AIndex + 1);

PromptFixYInBaseBlocks;

end;

procedure TBlock.AddUnknownStatement(const AStatement: TStatement; const AYStart: Integer);

begin

AddStatement(AStatement);

AStatement.FYStart:= AYStart;

AStatement.Initialize;

FixYInBlock(0);

PromptFixYInBaseBlocks;

end;

procedure TBlock.AddStatement(const AStatement: TStatement);

begin

FStatements.Add(AStatement);

AStatement.FBaseBlock := Self;

AStatement.SetTextSize;

end;

procedure TBlock.AssignStatement(const AIndex: Integer; const AStatement : TStatement);

begin

FStatements[AIndex] := AStatement;

AStatement.FBaseBlock := Self;

AStatement.SetTextSize;

end;

procedure TBlock.InsertBlock(AIndex: Integer; const AInsertedBlock: TBlock);

var

I: Integer;

begin

AInsertedBlock.MoveRight(Self.FXStart - AInsertedBlock.FXStart);

AInsertedBlock.ChangeXLastBlock(Self.FXLast);

for I := 0 to AInsertedBlock.FStatements.Count - 1 do

begin

Inc(AIndex, I);

Self.Insert(AIndex, AInsertedBlock.FStatements[I]);

if AInsertedBlock.FStatements[I] is TOperator then

TOperator(AInsertedBlock.FStatements[I]).InstallCanvas(FCanvas);

end;

Dec(AIndex, AInsertedBlock.FStatements.Count - 1);

RedefineBlock(AIndex);

FixYInBlock(AIndex);

PromptFixYInBaseBlocks;

end;

function TBlock.Extract(const AStatement: TStatement): Integer;

begin

Result:= FindStatementIndex(AStatement.FYStart);

ExtractStatementAt(Result);

end;

function TBlock.ExtractStatementAt(const AIndex: Integer) : TStatement;

begin

Result:= FStatements[AIndex];

Result.FBaseBlock := nil;

FStatements.Delete(AIndex);

if FStatements.Count = 0 then

begin

AddStatement(DefaultStatement.Create(DefaultAction, Self));

FStatements[0].FYStart:= Result.FYStart;

end

else if (BaseOperator = nil) and (AIndex = 0) then

FStatements[0].FYStart:= Result.FYStart;

end;

procedure TBlock.RemoveStatementAt(const Index: Integer);

begin

ExtractStatementAt(Index).Destroy;

end;

procedure TBlock.FixYStatement(AIndex: Integer = 0);

var

I: Integer;

procedure FixYBlocks(const ABlocks: TBlockArr); inline;

var

I: Integer;

begin

for I := 0 to High(ABlocks) do

ABlocks[I].FixYStatement;

end;

begin

if AIndex = 0 then

begin

if BaseOperator <> nil then

FStatements[AIndex].Lower(BaseOperator.GetBlockYStart - FStatements[AIndex].FYStart);

if FStatements[AIndex] is TOperator then

FixYBlocks(TOperator(FStatements[AIndex]).FBlocks);

Inc(AIndex);

end;

for I := AIndex to FStatements.Count - 1 do

begin

FStatements[I].Lower(FStatements[I - 1].GetYBottom - FStatements[I].FYStart);

if FStatements[I] is TOperator then

FixYBlocks(TOperator(FStatements[I]).FBlocks);

end;

if (BaseOperator <> nil) and not BaseOperator.IsPreсOperator then

BaseOperator.SetOptimalYLast;

end;

function TBlock.FindOptimalXLast: Integer;

var

I, CurrOptimalX: Integer;

Blocks: TBlockArr;

procedure CheckNewOptimalX(var AResult: Integer; const ACurrOptimalX: Integer); inline;

begin

if ACurrOptimalX > AResult then

AResult:= ACurrOptimalX;

end;

begin

Result:= -1;

for I := 0 to FStatements.Count - 1 do

begin

CurrOptimalX:= FXStart + FStatements[I].GetOptimaWidth;

CheckNewOptimalX(Result, CurrOptimalX);

if FStatements[I] is TOperator then

begin

Blocks:= TOperator(FStatements[I]).FBlocks;

CurrOptimalX:= Blocks[High(Blocks)].FindOptimalXLast;

CheckNewOptimalX(Result, CurrOptimalX);

end;

end;

if BaseOperator <> nil then

begin

CurrOptimalX:= FXStart + BaseOperator.GetOptimalWidthForBlock(Self);

CheckNewOptimalX(Result, CurrOptimalX);

end;

end;

procedure TBlock.MoveRightExceptXLast(const AOffset: Integer);

var

I, J: Integer;

Blocks: TBlockArr;

begin

Inc(FXStart, AOffset);

for I := 0 to FStatements.Count - 1 do

if FStatements[I] is TOperator then

begin

Blocks:= TOperator(FStatements[I]).FBlocks;

for J := 0 to High(Blocks) - 1 do

Blocks[J].MoveRight(AOffset);

Blocks[High(Blocks)].MoveRightExceptXLast(AOffset);

end;

end;

procedure TBlock.AdjustStatements;

var

I: Integer;

procedure AdjustOtherStatements(const ABlocks: TBlockArr); inline;

var

I: Integer;

begin

for I := 0 to High(ABlocks) do

ABlocks[I].AdjustStatements;

end;

begin

if BaseOperator <> nil then

FStatements[0].Lower(BaseOperator.GetBlockYStart - FStatements[0].FYStart);

if FStatements[0] is TOperator then

AdjustOtherStatements(TOperator(FStatements[0]).FBlocks);

for I := 1 to FStatements.Count - 1 do

begin

FStatements[I].Lower(FStatements[I - 1].GetYBottom - FStatements[I].FYStart);

if FStatements[I] is TOperator then

AdjustOtherStatements(TOperator(FStatements[I]).FBlocks);

end;

end;

procedure TBlock.MoveRight(const AOffset: Integer);

var

I: Integer;

begin

Inc(FXStart, AOffset);

Inc(FXLast, AOffset);

for I := 0 to FStatements.Count - 1 do

if FStatements[I] is TOperator then

TOperator(FStatements[I]).MoveRightChildrens(AOffset);

end;

procedure TBlock.MoveDown(const AOffset: Integer);

var

I: Integer;

begin

for I := 0 to FStatements.Count - 1 do

begin

FStatements[I].Lower(AOffset);

if FStatements[I] is TOperator then

TOperator(FStatements[I]).MoveDownChildrens(AOffset);

end;

end;

procedure TBlock.ChangeXStartBlock(const ANewXStart: Integer);

var

I: Integer;

Blocks: TBlockArr;

begin

FXStart:= ANewXStart;

for I := 0 to FStatements.Count - 1 do

if FStatements[I] is TOperator then

begin

Blocks:= TOperator(FStatements[I]).FBlocks;

Blocks[High(Blocks)].ChangeXStartBlock(ANewXStart);

end;

end;

procedure TBlock.ChangeXLastBlock(const ANewXLast: Integer);

var

I: Integer;

Blocks: TBlockArr;

begin

FXLast:= ANewXLast;

for I := 0 to FStatements.Count - 1 do

if FStatements[I] is TOperator then

begin

Blocks:= TOperator(FStatements[I]).FBlocks;

Blocks[High(Blocks)].ChangeXLastBlock(ANewXLast);

end;

end;

procedure TBlock.SetOptimalXLastBlock;

var

CurrBlock: TBlock;

NewXLast, OldXLast: Integer;

I, Index: Integer;

Blocks: TBlockArr;

begin

OldXLast:= Self.FXLast;

CurrBlock:= Self;

while (CurrBlock.BaseOperator <> nil) and

(CurrBlock.BaseOperator.BaseBlock.FXLast = OldXLast) do

CurrBlock:= CurrBlock.BaseOperator.BaseBlock;

NewXLast:= CurrBlock.FindOptimalXLast;

CurrBlock.ChangeXLastBlock(NewXLast);

if CurrBlock.FBaseOperator <> nil then

begin

Blocks:= CurrBlock.FBaseOperator.FBlocks;

Index:= CurrBlock.FBaseOperator.FindBlockIndex(CurrBlock.FXStart) + 1;

for I := Index to High(Blocks) - 1 do

Blocks[I].MoveRight(Blocks[I - 1].FXLast - Blocks[I].FXStart);

I:= High(Blocks);

Blocks[I].MoveRightExceptXLast(Blocks[I - 1].FXLast - Blocks[I].FXStart);

Blocks[I].SetOptimalXLastBlock;

end;

end;

procedure TBlock.SetNewActionForDefaultStatements(const AOldDefaultAction: string);

var

I: Integer;

procedure CheckForOperator(const AStatement: TStatement; const AOldDefaultAction: string); inline;

var

Blocks : TBlockArr;

J: Integer;

begin

if AStatement is TOperator then

begin

Blocks := TOperator(AStatement).Blocks;

for J := 0 to High(Blocks) do

Blocks[J].SetNewActionForDefaultStatements(AOldDefaultAction);

end;

end;

procedure CheckForDefault(const ABlock: TBlock; var AIndex: Integer; const AOldDefaultAction: string); inline;

begin

with ABlock do

begin

if FStatements[AIndex] is DefaultStatement then

begin

if FStatements[AIndex].FAction = AOldDefaultAction then

FStatements[AIndex].ChangeAction(DefaultAction)

else if FStatements[AIndex].FAction = DefaultAction then

begin

RemoveStatementAt(AIndex);

Dec(AIndex);

end;

end;

end;

end;

begin

I:= 0;

while I < Statements.Count - 1 do

begin

CheckForOperator(Statements[I], AOldDefaultAction);

CheckForDefault(Self, I, AOldDefaultAction);

Inc(I);

end;

I:= Statements.Count - 1;

CheckForOperator(Statements[I], AOldDefaultAction);

CheckForDefault(Self, I, AOldDefaultAction);

end;

function TBlock.FindStatementIndex(const AFYStart: Integer): Integer;

var

L, R, M: Integer;

begin

L := 0;

R := FStatements.Count - 1;

Result := -1;

while L <= R do

begin

M := (L + R) shr 1;

if FStatements[M].FYStart = AFYStart then

Exit(M)

else if FStatements[M].FYStart < AFYStart then

L := M + 1

else

R := M - 1;

end;

end;

procedure TBlock.PromptFixYInBaseBlocks;

var

CurrBlock: TBlock;

CurrOperator: TOperator;

begin

CurrBlock:= Self;

while CurrBlock.BaseOperator <> nil do

begin

CurrOperator:= CurrBlock.BaseOperator;

CurrBlock:= CurrOperator.BaseBlock;

CurrOperator.AlignBlocks;

CurrBlock.FixYStatement(CurrBlock.FindStatementIndex(CurrOperator.FYStart) + 1);

end;

end;

procedure TBlock.FixYInBlock(const Index: Integer);

begin

// Shift all statements after and childrens

FixYStatement(Index);

if FStatements[Index] is TOperator then

TOperator(FStatements[Index]).AlignBlocks;

end;

procedure TBlock.Install(const Index: Integer);

var

I: Integer;

Blocks: TBlockArr;

CurrOperator: TOperator;

begin

FStatements[Index].SetOptimalYLast;

if FStatements[Index] is TOperator then

begin

CurrOperator:= TOperator(FStatements[Index]);

Blocks:= CurrOperator.Blocks;

Blocks[0].SetOptimalXLastBlock;

Blocks[0].GetLastStatement.SetYBottom(Blocks[0].GetLastStatement.GetOptimalYLast);

for I := 1 to High(Blocks) - 1 do

begin

Blocks[I].SetOptimalXLastBlock;

Blocks[I].GetLastStatement.SetYBottom(Blocks[I].GetLastStatement.GetOptimalYLast);

end;

CurrOperator.AlignBlocks;

end

else

Self.SetOptimalXLastBlock;

FixYInBlock(Index);

PromptFixYInBaseBlocks;

end;

procedure TBlock.DrawBlock(const AVisibleImageRect: TVisibleImageRect);

var

L, R, M: Integer;

CurrStatement: TStatement;

isTOperator: Boolean;

begin

L := 0;

R := FStatements.Count - 1;

while L < R do

begin

M := (L + R) shr 1;

case FStatements[M].GetMask(AVisibleImageRect, FStatements[M] is TOperator) of

$0F {1111}, $03 {0011}, $07 {0111}, $0B {1011}:

R := M;

$09 {1001}:

R := M - 1;

else

L := M + 1;

end;

end;

if R >= 0 then

begin

if (R <> 0) and (FStatements[R - 1] is TOperator) then

TOperator(FStatements[R - 1]).DrawBlocks(AVisibleImageRect);

for M := R to FStatements.Count - 1 do

begin

CurrStatement:= FStatements[M];

isTOperator:= CurrStatement is TOperator;

if isTOperator then

TOperator(CurrStatement).DrawBlocks(AVisibleImageRect);

case CurrStatement.GetMask(AVisibleImageRect, isTOperator) of

$0F {1111}, $03 {0011}, $07 {0111}, $0B {1011}:

CurrStatement.Draw;

else

Break;

end;

end;

end;

end;

function TBlock.GetLastStatement: TStatement;

begin

if (BaseOperator = nil) or BaseOperator.IsPreсOperator then

Result:= FStatements.GetLast

else

Result:= BaseOperator;

end;

function TBlock.Clone(const ABaseOperator: TOperator): TBlock;

var

I: Integer;

NewStatements: TStatement;

begin

Result:= TBlock.Create(ABaseOperator);

Result.FCanvas:= Self.FCanvas;

Result.FXStart:= Self.FXStart;

Result.FXLast:= Self.FXLast;

Result.FStatements:= TArrayList<TStatement>.Create(Self.FStatements.Count);

for I := 0 to Self.FStatements.Count - 1 do

begin

NewStatements:= Self.FStatements[I].Clone;

NewStatements.FBaseBlock:= Result;

Result.FStatements.Add(NewStatements);

end;

end;

procedure TBlock.Assign(const ASource: TBlock);

begin

Self.FXStart:= ASource.FXStart;

Self.FXLast:= ASource.FXLast;

Self.FCanvas:= ASource.FCanvas;

end;

function TBlock.GetMask(const AVisibleImageRect: TVisibleImageRect): Integer;

begin

Result :=

{X--- : }

Ord(FXStart >= AVisibleImageRect.FTopLeft.X) shl 3 or

{-X-- : }

Ord(FXLast <= AVisibleImageRect.FBottomRight.X) shl 2 or

{--X- : }

Ord(FXStart <= AVisibleImageRect.FBottomRight.X) shl 1 or

{---X : }

Ord(FXLast >= AVisibleImageRect.FTopLeft.X);

end;

procedure TBlock.InstallCanvas(const ACanvas: TCanvas);

var

I: Integer;

begin

FCanvas := ACanvas;

for I := 0 to FStatements.Count - 1 do

if FStatements[I] is TOperator then

TOperator(FStatements[I]).InstallCanvas(ACanvas);

end;

{ TOperator }

constructor TOperator.Create(const AAction : String);

begin

inherited;

CreateBlock;

end;

destructor TOperator.Destroy;

var

I: Integer;

begin

for I := 0 to High(FBlocks) do

FBlocks[I].Destroy;

inherited;

end;

function TOperator.FindBlockIndex(const AXStart: Integer): Integer;

var

L, R, M: Integer;

begin

L := 0;

R := High(FBlocks);

Result := -1;

while L <= R do

begin

M := (L + R) shr 1;

if FBlocks[M].FXStart = AXStart then

Exit(M)

else if FBlocks[M].FXStart < AXStart then

L := M + 1

else

R := M - 1;

end;

end;

function TOperator.GetBlockYStart: Integer;

begin

case IsPreсOperator of

True: Result := FYLast;

False: Result := FYStart;

end;

end;

procedure TOperator.InitializeBlocks(StartIndex: Integer = 0);

var

I: Integer;

BlockYStart: Integer;

procedure SetYPos(const ABlock: TBlock; const AYStart: Integer); inline;

begin

ABlock.Statements[0].FYStart := AYStart;

ABlock.Statements[0].SetOptimalYLast;

ABlock.FixYStatement;

end;

begin

InstallCanvas(FBaseBlock.FCanvas);

SetBlockTextSize;

BlockYStart:= GetBlockYStart;

if StartIndex = 0 then

begin

SetYPos(Blocks[StartIndex], BlockYStart);

Blocks[StartIndex].FXStart:= BaseBlock.FXStart + GetOffsetFromXStart;

Blocks[StartIndex].ChangeXLastBlock(Blocks[StartIndex].FindOptimalXLast);

Inc(StartIndex);

end;

for I := StartIndex to High(Blocks) - 1 do

begin

SetYPos(Blocks[I], BlockYStart);

Blocks[I].FXStart:= Blocks[I - 1].FXLast;

Blocks[I].ChangeXLastBlock(Blocks[I].FindOptimalXLast);

end;

if Length(Blocks) > 1 then

Blocks[High(Blocks)].FXStart:= Blocks[High(Blocks) - 1].FXLast;

SetYPos(Blocks[High(Blocks)], BlockYStart);

Blocks[High(Blocks)].FXLast:= BaseBlock.FXLast;

AlignBlocks;

end;

function TOperator.GetYBottom: Integer;

begin

case IsPreсOperator of

True: Result:= FBlocks[0].FStatements.GetLast.GetYBottom;

False: Result := FYLast;

end;

end;

procedure TOperator.SetYBottom(const AYBottom: Integer);

var

I: Integer;

begin

case IsPreсOperator of

True:

for I := 0 to High(FBlocks) do

FBlocks[I].Statements.GetLast.SetYBottom(AYBottom);

False:

FYLast := AYBottom;

end;

end;

function TOperator.GetMaxOptimalYBottom: Integer;

var

I: Integer;

begin

Result := -1;

case IsPreсOperator of

True:

for I := 0 to High(FBlocks) do

Result := Max(Result, FBlocks[I].Statements.GetLast.GetMaxOptimalYBottom);

False: Result := GetOptimalYLast;

end;

end;

procedure TOperator.DrawBlocks(const AVisibleImageRect: TVisibleImageRect);

var

L, R, M: Integer;

begin

L := 0;

R := High(FBlocks);

while L < R do

begin

M := (L + R) shr 1;

case FBlocks[M].GetMask(AVisibleImageRect) of

$0F {1111}, $03 {0011}, $07 {0111}, $0B {1011}:

R := M;

$09 {1001}:

R := M - 1;

else

L := M + 1;

end;

end;

if R >= 0 then

for M := R to High(FBlocks) do

begin

case FBlocks[M].GetMask(AVisibleImageRect) of

$0F {1111}, $03 {0011}, $07 {0111}, $0B {1011}:

FBlocks[M].DrawBlock(AVisibleImageRect);

else

Break;

end;

end;

end;

procedure TOperator.AlignBlocks;

var

I, MaxYLast, CurrYLast: Integer;

begin

if Length(FBlocks) > 1 then

begin

MaxYLast := FBlocks[0].GetLastStatement.GetMaxOptimalYBottom;

for I := 1 to High(FBlocks) do

begin

CurrYLast := FBlocks[I].GetLastStatement.GetMaxOptimalYBottom;;

if MaxYLast < CurrYLast then

MaxYLast := CurrYLast;

end;

for I := 0 to High(FBlocks) do

if FBlocks[I].GetLastStatement.GetYBottom <> MaxYLast then

FBlocks[I].GetLastStatement.SetYBottom(MaxYLast);

end;

end;

procedure TOperator.Initialize;

begin

case IsPreсOperator of

True:

begin

SetOptimalYLast;

InitializeBlocks;

end;

False:

begin

InitializeBlocks;

SetOptimalYLast;

end;

end;

BaseBlock.SetOptimalXLastBlock;

end;

procedure TOperator.AlignBlocksToXStart;

var

I, CurrXStart: Integer;

begin

CurrXStart:= BaseBlock.FXStart + GetOffsetFromXStart;

if CurrXStart <> FBlocks[0].FXStart then

if Length(FBlocks) = 1 then

FBlocks[0].ChangeXStartBlock(CurrXStart)

else

begin

CurrXStart := CurrXStart - FBlocks[0].FXStart;

for I := 0 to High(FBlocks) - 1 do

FBlocks[I].MoveRight(CurrXStart);

FBlocks[High(FBlocks)].ChangeXStartBlock(FBlocks[High(FBlocks) - 1].XLast);

end;

end;

procedure TOperator.RedefineStatement;

procedure GlueBlock(const ABlock: TBlock); inline; forward;

procedure GlueBlocks(const ABlocks: TBlockArr); inline;

var

I: Integer;

begin

for I := 0 to High(ABlocks) do

GlueBlock(ABlocks[I]);

end;

procedure GlueBlock(const ABlock: TBlock); inline;

var

I: Integer;

begin

with ABlock do

begin

FStatements[0].Lower(BaseOperator.GetBlockYStart - FStatements[0].FYStart);

if FStatements[0] is TOperator then

GlueBlocks(TOperator(FStatements[0]).Blocks);

for I := 1 to FStatements.Count - 1 do

begin

FStatements[I].Lower(FStatements[I - 1].GetYBottom - FStatements[I].FYStart);

if FStatements[I] is TOperator then

GlueBlocks(TOperator(FStatements[I]).Blocks);

end;

end;

end;

procedure RedefineBlocks;

var

I: Integer;

begin

AlignBlocksToXStart;

for I := 0 to High(FBlocks) do

begin

FBlocks[I].RedefineBlock;

GlueBlock(FBlocks[I]);

end;

AlignBlocks;

end;

begin

SetTextSize;

case Self.IsPreсOperator of

True:

begin

SetOptimalYLast;

RedefineBlocks;

end;

False:

begin

RedefineBlocks;

SetOptimalYLast;

end;

end;

end;

function TOperator.Clone: TStatement;

var

I: Integer;

ResultOperator: TOperator;

begin

Result:= inherited;

ResultOperator:= TOperator(Result);

SetLength(ResultOperator.FBlocks, Length(Self.Blocks));

for I := 0 to High(Self.Blocks) do

ResultOperator.FBlocks[I]:= Self.FBlocks[I].Clone(ResultOperator);

end;

function TOperator.GetOffsetFromXStart: Integer;

begin

Result:= 0;

end;

procedure TOperator.InstallCanvas(const ACanvas: TCanvas);

var

I: Integer;

begin

for I := 0 to High(Blocks) do

Blocks[I].InstallCanvas(ACanvas);

end;

procedure TOperator.SetBlockTextSize;

var

I, J: Integer;

begin

for I := 0 to High(FBlocks) do

for J := 0 to FBlocks[I].Statements.Count - 1 do

begin

FBlocks[I].Statements[J].SetTextSize;

if FBlocks[I].Statements[J] is TOperator then

TOperator(FBlocks[I].Statements[J]).SetBlockTextSize;

end;

end;

procedure TOperator.MoveRightChildrens(const AOffset : Integer);

var

I: Integer;

begin

for I := 0 to High(FBlocks) do

FBlocks[I].MoveRight(AOffset);

end;

procedure TOperator.MoveDownChildrens(const AOffset : Integer);

var

I: Integer;

begin

for I := 0 to High(FBlocks) do

FBlocks[I].MoveDown(AOffset);

end;

procedure TOperator.SetXLastForChildrens(const AXLast : Integer);

begin

FBlocks[High(FBlocks)].ChangeXLastBlock(AXLast);

end;

end.

Приложение Х

(обязательное)

Исходный код программы (модуль uCaseBranching)

unit uCaseBranching;

interface

uses

uBase, uAdditionalTypes, uDrawShapes, uDetermineDimensions, uMinMaxInt,

uCaseBlockSorting;

type

TCaseBranching = class(TOperator)

private

FConds: TStringArr;

FCondsSizes: TSizeArr;

function GetMaxHeightOfConds: Integer;

procedure SetCondSize(const AIndex: Integer);

procedure RestoreBlocksAfterRearrangement;

procedure RepairChildBlocks(const AHigh: Integer);

protected

procedure SetTextSize; override;

procedure CreateBlock; override;

procedure CreateBlockStarting(AStartIndex: Integer);

function GetOptimaWidth: Integer; override;

function GetOptimalWidthForBlock(const ABlock: TBlock): Integer; override;

function GetOptimalYLast: Integer; override;

procedure Draw; override;

public

constructor Create(const AAction : String; const AConds: TStringArr);

function IsPreсOperator: Boolean; override;

procedure ChangeActionWithConds(const AAction: String; const AConds: TStringArr);

function Clone: TStatement; override;

procedure SortConditions(const SortNumber: Integer);

procedure RestoreСonditions(const AConds: TStringArr; const ABlocks: TBlockArr);

function GetSerialNumber: Integer; override;

property Conds: TStringArr read FConds write FConds;

property CondsSizes: TSizeArr read FCondsSizes;

end;

implementation

constructor TCaseBranching.Create(const AAction : String; const AConds: TStringArr);

begin

FConds:= AConds;

SetLength(FCondsSizes, Length(AConds));

inherited Create(AAction);

end;

procedure TCaseBranching.SetCondSize(const AIndex: Integer);

begin

FCondsSizes[AIndex] := GetTextSize(BaseBlock.Canvas, FConds[AIndex]);

end;

procedure TCaseBranching.RestoreСonditions(const AConds: TStringArr; const ABlocks: TBlockArr);

var

LastBlock: TBlock;

begin

// Finding the last block before sorting

LastBlock:= FBlocks[High(FBlocks)];

// Decrease the last x by 1 to untie it from the base block

SetXLastForChildrens(LastBlock.XLast - 1);

// Set old values

FConds:= AConds;

FBlocks:= ABlocks;

// Move the blocks in a new order

RestoreBlocksAfterRearrangement;

// Set the optimal length for the last block before sorting

LastBlock.SetOptimalXLastBlock;

// Stretch the new last block to the base

SetXLastForChildrens(FBaseBlock.XLast);

end;

procedure TCaseBranching.SortConditions(const SortNumber: Integer);

var

Compare: TCompareFunction;

LastBlock: TBlock;

begin

case SortNumber of

0: Compare:= CompareStrAsc;

1: Compare:= CompareStrDesc;

end;

// Finding the last block before sorting

LastBlock:= FBlocks[High(FBlocks)];

// Decrease the last x by 1 to untie it from the base block

SetXLastForChildrens(LastBlock.XLast - 1);

// Sorting blocks

QuickSort(FConds, FBlocks, Compare);

// Move the blocks in a new order

RestoreBlocksAfterRearrangement;

// Stretch the new last block to the base

SetXLastForChildrens(FBaseBlock.XLast);

// Set the optimal length for the last block before sorting

LastBlock.SetOptimalXLastBlock;

end;

procedure TCaseBranching.RestoreBlocksAfterRearrangement;

var

I: Integer;

begin

FBlocks[0].MoveRight(BaseBlock.XStart - FBlocks[0].XStart);

SetCondSize(0);

for I := 1 to High(FBlocks) do

begin

FBlocks[I].MoveRight(FBlocks[I - 1].XLast - FBlocks[I].XStart);

SetCondSize(I);

end;

end;

procedure TCaseBranching.SetTextSize;

var

I: Integer;

begin

inherited;

for I := 0 to High(FConds) do

SetCondSize(I);

end;

procedure TCaseBranching.RepairChildBlocks(const AHigh: Integer);

var

I, StartIndex: Integer;

begin

StartIndex := AHigh + 1;

for I := 1 to AHigh do

if FBlocks[I - 1].XLast - FBlocks[I].XStart <> 0 then

begin

StartIndex:= I;

Break;

end;

for I := StartIndex to AHigh do

FBlocks[I].MoveRight(FBlocks[I - 1].XLast - FBlocks[I].XStart);

FBlocks[AHigh].ChangeXLastBlock(BaseBlock.XLast);

end;

procedure TCaseBranching.ChangeActionWithConds(const AAction: String;

const AConds: TStringArr);

var

I, MinHigh: Integer;

PrevConds: TStringArr;

begin

PrevConds:= FConds;

FConds:= AConds;

SetLength(FCondsSizes, Length(AConds));

// Check what conditions have changed

MinHigh := Min(High(PrevConds), High(FConds));

for I := 0 to MinHigh do

if FConds[I] <> PrevConds[I] then

begin

SetCondSize(I);

FBlocks[I].ChangeXLastBlock(FBlocks[I].FindOptimalXLast);

end;

// Repair of children after X change

RepairChildBlocks(MinHigh);

// Remove blocks if the amount of conditions has decreased

for I := Length(FConds) to High(FBlocks) do

FBlocks[I].Destroy;

// Setting a new amount for blocks

SetLength(FBlocks, Length(AConds));

// Add new blocks if the amount of conditions has increased

if Length(PrevConds) < Length(FConds) then

begin

// Set the width to one, to untie the X of the last block. In the future

// will set the optimal width

FBlocks[High(PrevConds)].ChangeXLastBlock(FBlocks[High(PrevConds)].XStart + 1);

for I := Length(PrevConds) to High(FConds) do

SetCondSize(I);

// Сreate and initialize new blocks. Set the width to one. In the future

// will set the optimal width

CreateBlockStarting(Length(PrevConds));

InitializeBlocks(Length(PrevConds));

// Set the optimal width of the last block

FBlocks[High(PrevConds)].SetOptimalXLastBlock;

end;

// Changing the action

ChangeAction(AAction);

end;

function TCaseBranching.Clone: TStatement;

var

ResultCase: TCaseBranching;

begin

Result:= inherited;

ResultCase:= TCaseBranching(Result);

ResultCase.FConds:= Copy(Self.FConds);

ResultCase.FCondsSizes:= Copy(Self.FCondsSizes);

end;

function TCaseBranching.GetMaxHeightOfConds: Integer;

var

I: Integer;

begin

Result:= FCondsSizes[0].Height;

for I := 1 to High(FConds) do

if FCondsSizes[I].Height > Result then

Result:= FCondsSizes[I].Height;

end;

function TCaseBranching.GetOptimalYLast: Integer;

begin

Result:= FYStart + GetMaxHeightOfConds + FActionSize.Height + FYIndentText shl 2;

end;

function TCaseBranching.GetOptimaWidth: Integer;

begin

Result:= (FActionSize.Width + FXMinIndentText shl 1) \*

(FActionSize.Height + FYIndentText shl 1) div FYIndentText;

end;

function TCaseBranching.GetOptimalWidthForBlock(const ABlock: TBlock): Integer;

begin

Result:= FCondsSizes[FindBlockIndex(ABlock.XStart)].Width + FXMinIndentText shl 1;

end;

procedure TCaseBranching.CreateBlock;

begin

SetLength(FBlocks, Length(FConds));

CreateBlockStarting(0);

end;

procedure TCaseBranching.CreateBlockStarting(AStartIndex: Integer);

var

I, HighIndex: Integer;

begin

HighIndex:= High(FBlocks);

for I := HighIndex downto AStartIndex do

begin

FBlocks[I]:= TBlock.Create(MaxInt - (HighIndex - I), Self);

FBlocks[I].Statements.Add(DefaultStatement.Create(DefaultAction, FBlocks[I]));

end;

end;

function TCaseBranching.IsPreсOperator: Boolean;

begin

Result:= True;

end;

function TCaseBranching.GetSerialNumber: Integer;

begin

Result:= 2;

end;

procedure TCaseBranching.Draw;

var

I: Integer;

YTriangleHeight : Integer;

LeftTriangleWidth : Integer;

PartLeftTriangleWidth : Integer;

begin

// Calculate the height of a triangle

YTriangleHeight:= FYStart + FActionSize.Height + FYIndentText shl 1;

// Drawing the main block

DrawRect(BaseBlock.XStart, BaseBlock.XLast, FYStart, FYLast, BaseBlock.Canvas);

// Drawing a triangle

DrawInvertedTriangle(BaseBlock.XStart, FBlocks[High(FBlocks)].XStart,

BaseBlock.XLast, FYStart, YTriangleHeight, BaseBlock.Canvas);

// Draw a line that connects the vertex of the triangle and

// the lower base of the operator

DrawLine(FBlocks[High(FBlocks)].XStart, FBlocks[High(FBlocks)].XStart,

YTriangleHeight, FYLast, BaseBlock.Canvas);

{ Draw the lines that connect the side of the triangle to the side of the block }

// Calculate the width to the left of the vertex of the triangle

LeftTriangleWidth:= 0;

for I := 0 to High(FBlocks) - 1 do

Inc(LeftTriangleWidth, FBlocks[I].XLast - FBlocks[I].XStart);

// Find the Y coordinate for each block

PartLeftTriangleWidth:= LeftTriangleWidth;

for I := 0 to High(FBlocks) - 2 do

begin

Dec(PartLeftTriangleWidth, FBlocks[I].XLast - FBlocks[I].XStart);

DrawLine(FBlocks[I].XLast, FBlocks[I].XLast,

YTriangleHeight - (YTriangleHeight - FYStart) \*

PartLeftTriangleWidth div LeftTriangleWidth,

FYLast, BaseBlock.Canvas);

end;

{ End }

// Drawing the action

DrawText(BaseBlock.Canvas,

BaseBlock.XStart

+

LeftTriangleWidth \* (FActionSize.Height + FYIndentText) div (YTriangleHeight - FYStart)

+

(BaseBlock.XLast - BaseBlock.XStart) \* FYIndentText div (YTriangleHeight - FYStart) shr 1

-

FActionSize.Width shr 1

,

FYStart + FYIndentText, Action);

// Drawing the conditions

Inc(YTriangleHeight, FYIndentText);

for I := 0 to High(FConds) do

DrawText(BaseBlock.Canvas,

FBlocks[I].XStart + ((FBlocks[I].XLast - FBlocks[I].XStart) shr 1)

- (FCondsSizes[I].Width shr 1),

YTriangleHeight, FConds[I]);

end;

end.

Приложение Ц

(обязательное)

Исходный код программы (модуль uFirstLoop)

unit uFirstLoop;

interface

uses

uDrawShapes, uLoop;

type

TFirstLoop = class(TLoop)

protected

function GetOptimalYLast: Integer; override;

procedure Draw; override;

public

function IsPreсOperator: Boolean; override;

function GetSerialNumber: Integer; override;

end;

implementation

function TFirstLoop.IsPreсOperator: Boolean;

begin

Result:= True;

end;

function TFirstLoop.GetOptimalYLast: Integer;

begin

Result := FYStart + FActionSize.Height + FYIndentText shl 1;

end;

function TFirstLoop.GetSerialNumber: Integer;

begin

Result:= 3;

end;

procedure TFirstLoop.Draw;

begin

DrawUnfinishedVertRectForLoop(BaseBlock.XStart, BaseBlock.XLast, FYStart,

FYLast, GetYBottom, BaseBlock.Canvas);

DrawUnfinishedHorRectForLoop(BaseBlock.XStart, Blocks[0].XStart,

BaseBlock.XLast, FYStart, FYLast, BaseBlock.Canvas);

DrawText(BaseBlock.Canvas, BaseBlock.XStart + ((BaseBlock.XLast - BaseBlock.XStart) shr 1)

- (FActionSize.Width shr 1), FYStart + FYIndentText, Action);

end;

end.

Приложение Ч

(обязательное)

Исходный код программы (модуль uIfBranching)

unit uIfBranching;

interface

uses

uBase, uDrawShapes, uMinMaxInt, uDetermineDimensions, uAdditionalTypes,

uConstants;

type

TIfBranching = class(TOperator)

private const

FBlockCount = 2;

private class var

FTrueCond, FFalseCond: string;

private

FTrueSize, FFalseSize: TSize;

procedure SetCondsSize;

function GetMinValidPartWidth(const ATextHeight, ATextWidth: Integer): Integer;

class procedure RedefineConds(const ABlock: TBlock); static;

protected

procedure SetTextSize; override;

function GetOptimaWidth: Integer; override;

procedure CreateBlock; override;

function GetOptimalWidthForBlock(const ABlock: TBlock): Integer; override;

function GetOptimalYLast: Integer; override;

procedure Draw; override;

public

property TrueSize: TSize read FTrueSize;

property FalseSize: TSize read FFalseSize;

function IsPreсOperator: Boolean; override;

function Clone: TStatement; override;

function GetAvailablePartWidth(const APartWidth, ATextHeight: Integer): Integer;

function GetSerialNumber: Integer; override;

class property TrueCond: string read FTrueCond write FTrueCond;

class property FalseCond: string read FFalseCond write FFalseCond;

class procedure RedefineSizesForIfBranching(const ABlock: TBlock); static;

end;

implementation

class procedure TIfBranching.RedefineSizesForIfBranching(const ABlock: TBlock);

begin

RedefineConds(ABlock);

ABlock.AdjustStatements;

end;

class procedure TIfBranching.RedefineConds(const ABlock: TBlock);

var

I, J: Integer;

CurrOperator: TOperator;

Statement: TStatement;

begin

for I := 0 to ABlock.Statements.Count - 1 do

begin

Statement := ABlock.Statements[I];

if Statement is TOperator then

begin

CurrOperator:= TOperator(Statement);

if CurrOperator is TIfBranching then

begin

TIfBranching(CurrOperator).SetCondsSize;

CurrOperator.SetOptimalYLast;

for J := 0 to High(CurrOperator.Blocks) do

begin

RedefineConds(CurrOperator.Blocks[J]);

CurrOperator.Blocks[J].SetOptimalXLastBlock;

end;

end

else

for J := 0 to High(CurrOperator.Blocks) do

RedefineConds(CurrOperator.Blocks[J]);

CurrOperator.AlignBlocks;

end;

end;

end;

procedure TIfBranching.SetCondsSize;

begin

FTrueSize := GetTextSize(BaseBlock.Canvas, FTrueCond);

FFalseSize := GetTextSize(BaseBlock.Canvas, FFalseCond);

end;

procedure TIfBranching.SetTextSize;

begin

inherited;

SetCondsSize;

end;

function TIfBranching.Clone: TStatement;

var

ResultIf: TIfBranching;

begin

Result:= inherited;

ResultIf:= TIfBranching(Result);

ResultIf.FTrueSize := Self.FTrueSize;

ResultIf.FFalseSize := Self.FFalseSize;

end;

function TIfBranching.GetOptimalYLast: Integer;

begin

Result := FYStart + Max(FTrueSize.Height, FFalseSize.Height) +

FActionSize.Height + 3 \* FYIndentText;

end;

function TIfBranching.IsPreсOperator: Boolean;

begin

Result:= True;

end;

procedure TIfBranching.CreateBlock;

begin

SetLength(FBlocks, FBlockCount);

FBlocks[0]:= TBlock.Create(Self);

FBlocks[1]:= TBlock.Create(Self);

FBlocks[0].Statements.Add(DefaultStatement.Create(DefaultAction, FBlocks[0]));

FBlocks[1].Statements.Add(DefaultStatement.Create(DefaultAction, FBlocks[1]));

end;

function TIfBranching.GetAvailablePartWidth(const APartWidth, ATextHeight: Integer): Integer;

begin

Result:= APartWidth \*

(FYLast - FYStart - ATextHeight - FYIndentText) div (FYLast - FYStart);

end;

function TIfBranching.GetMinValidPartWidth(const ATextHeight,

ATextWidth: Integer): Integer;

begin

Result:= (ATextWidth + FXMinIndentText shl 1) \*

(FYLast - FYStart) div (FYLast - FYStart - ATextHeight - FYIndentText);

end;

function TIfBranching.GetOptimaWidth: Integer;

begin

Result:= GetMinValidPartWidth(FActionSize.Height, FActionSize.Width);

end;

function TIfBranching.GetOptimalWidthForBlock(const ABlock: TBlock): Integer;

begin

if ABlock = FBlocks[0] then

Result:= GetMinValidPartWidth(FTrueSize.Height, FTrueSize.Width)

else

Result:= GetMinValidPartWidth(FFalseSize.Height, FFalseSize.Width);

end;

function TIfBranching.GetSerialNumber: Integer;

begin

Result:= 1;

end;

procedure TIfBranching.Draw;

begin

// Drawing the main block

DrawRect(BaseBlock.XStart, BaseBlock.XLast, FYStart, FYLast, BaseBlock.Canvas);

// Drawing a triangle

DrawInvertedTriangle(BaseBlock.XStart, FBlocks[1].XStart, BaseBlock.XLast,

FYStart, FYLast, BaseBlock.Canvas);

// Drawing the action

DrawText(BaseBlock.Canvas,

FBlocks[0].XStart +

GetAvailablePartWidth(FBlocks[0].XLast - FBlocks[0].XStart, FTrueSize.Height + FYIndentText) +

GetAvailablePartWidth(BaseBlock.XLast - BaseBlock.XStart, FActionSize.Height) shr 1 -

FActionSize.Width shr 1,

FYStart + FYIndentText, Action);

// Drawing the True text

DrawText(BaseBlock.Canvas,

FBlocks[0].XStart + GetAvailablePartWidth(

FBlocks[0].XLast - FBlocks[0].XStart, FTrueSize.Height) shr 1 -

FTrueSize.Width shr 1,

FYStart + FYIndentText shl 1 + FActionSize.Height, FTrueCond);

// Drawing the False text

DrawText(BaseBlock.Canvas,

FBlocks[1].XLast - GetAvailablePartWidth(

FBlocks[1].XLast - FBlocks[1].XStart, FFalseSize.Height) shr 1 -

FFalseSize.Width shr 1,

FYStart + FYIndentText shl 1 + FActionSize.Height, FFalseCond);

end;

end.

Приложение Ш

(обязательное)

Исходный код программы (модуль uLastLoop)

unit uLastLoop;

interface

uses

uDrawShapes, uLoop;

type

TLastLoop = class(TLoop)

protected

function GetOptimalYLast: Integer; override;

procedure Draw; override;

public

function GetBlockYBottom: Integer;

function IsPreсOperator: Boolean; override;

function GetSerialNumber: Integer; override;

end;

implementation

function TLastLoop.IsPreсOperator: Boolean;

begin

Result:= False;

end;

procedure TLastLoop.Draw;

begin

DrawUnfinishedVertRectForLoop(BaseBlock.XStart, BaseBlock.XLast, FYLast,

GetYBottom, FYStart, BaseBlock.Canvas);

DrawUnfinishedHorRectForLoop(BaseBlock.XStart, Blocks[0].XStart,

BaseBlock.XLast, FYLast, FYStart, BaseBlock.Canvas);

DrawText(BaseBlock.Canvas, BaseBlock.XStart + ((BaseBlock.XLast - BaseBlock.XStart) shr 1)

- (FActionSize.Width shr 1), GetBlockYBottom + FYIndentText, Action);

end;

function TLastLoop.GetOptimalYLast: Integer;

begin

Result := GetBlockYBottom + FActionSize.Height + FXMinIndentText shl 1;

end;

function TLastLoop.GetSerialNumber: Integer;

begin

Result := 4;

end;

function TLastLoop.GetBlockYBottom: Integer;

begin

Result:= FBlocks[0].Statements.GetLast.GetYBottom;

end;

end.

Приложение Щ

(обязательное)

Исходный код программы (модуль uLoop)

unit uLoop;

interface

uses

uBase;

type

TLoop = class abstract(TOperator)

private const

FBlockCount = 1;

protected

FCountPixelCorrection: Integer;

procedure CreateBlock; override;

function GetOptimalWidthForBlock(const ABlock: TBlock): Integer; override;

function GetOptimaWidth: Integer; override;

procedure SetTextSize; override;

function GetXLastStrip: Integer;

public

property CountPixelCorrection: Integer read FCountPixelCorrection;

function Clone: TStatement; override;

function GetOffsetFromXStart: Integer; override;

end;

implementation

procedure TLoop.CreateBlock;

begin

SetLength(FBlocks, FBlockCount);

FBlocks[0] := TBlock.Create(Self);

FBlocks[0].Statements.Add(DefaultStatement.Create(DefaultAction, FBlocks[0]));

end;

procedure TLoop.SetTextSize;

begin

inherited;

FCountPixelCorrection:= BaseBlock.Canvas.Font.Size shl 1 + 5;

end;

function TLoop.GetOptimaWidth: Integer;

begin

Result := FActionSize.Width + FXMinIndentText shl 1;

end;

function TLoop.GetXLastStrip: Integer;

begin

Result:= FBaseBlock.XStart + FCountPixelCorrection;

end;

function TLoop.GetOffsetFromXStart: Integer;

begin

Result:= FCountPixelCorrection;

end;

function TLoop.Clone: TStatement;

begin

Result:= inherited;

TLoop(Result).FCountPixelCorrection := Self.FCountPixelCorrection;

end;

function TLoop.GetOptimalWidthForBlock(const ABlock: TBlock): Integer;

begin

Result:= -1;

if ABlock = FBlocks[0] then

Result:= GetOptimaWidth - GetXLastStrip;

end;

end.

Приложение Э

(обязательное)

Исходный код программы (модуль uProcessStatement)

unit uProcessStatement;

interface

uses

uBase, uDrawShapes;

type

TProcessStatement = class(TStatement)

protected

function GetOptimaWidth: Integer; override;

function GetOptimalYLast: Integer; override;

procedure Draw; override;

public

function GetSerialNumber: Integer; override;

end;

implementation

function TProcessStatement.GetOptimaWidth: Integer;

begin

result:= FActionSize.Width + FXMinIndentText shl 1;

end;

procedure TProcessStatement.Draw;

begin

DrawRect(BaseBlock.XStart, BaseBlock.XLast, FYStart, FYLast, BaseBlock.Canvas);

DrawText(BaseBlock.Canvas, BaseBlock.XStart + ((BaseBlock.XLast - BaseBlock.XStart) shr 1)

- (FActionSize.Width shr 1), FYStart + FYIndentText, Action);

end;

function TProcessStatement.GetOptimalYLast: Integer;

begin

Result := FYStart + FActionSize.Height + FYIndentText shl 1;

end;

function TProcessStatement.GetSerialNumber: Integer;

begin

Result:= 0;

end;

end.