## **СОДЕРЖАНИЕ**

[СОДЕРЖАНИЕ 4](#_Toc514881437)

[ВВЕДЕНИЕ 6](#_Toc514881438)

[1.1. Метод дамке представления алгоритмов 8](#_Toc514881439)

[1.2. Анализ существующих аналогов 10](#_Toc514881440)

[1.3. Постановка задачи 13](#_Toc514881441)

[2.МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ 14](#_Toc514881442)

[2.1. Представление векторного изображения в формате SVG 14](#_Toc514881443)

[2.2. Описание функциональности ПС 15](#_Toc514881444)

[2.3. Спецификация функциональных требований 16](#_Toc514881445)

[3.ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 17](#_Toc514881446)

[3.1. Проектирование динамических структур данных 17](#_Toc514881447)

[3.2. Разработка алгоритма вставки фигуры 19](#_Toc514881448)

[3.3. Разработка алгоритма отрисовки фигур 20](#_Toc514881449)

[3.4. Разработка алгоритма получения координат по Y 22](#_Toc514881450)

[3.5. Разработка алгоритма сдвига блоков 23](#_Toc514881451)

[3.6. Разработка алгоритма вставки текста 24](#_Toc514881452)

[4.СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА 25](#_Toc514881453)

[4.1. Разработка интерфейса программы 25](#_Toc514881454)

[4.2. Описание модуля Draw 27](#_Toc514881455)

[4.3 Описание модуля Lists 30](#_Toc514881456)

[Модуль Lists осуществляет работу с динамическими структурами: создание головы списка, поиск фигуры по координатам, поиск поиск адреса по данным и так далее. Полный список подпрограмм прелставлен в таблице 4.3 30](#_Toc514881457)

[6.ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ 33](#_Toc514881458)

[6.1. Тестирование функционала добавления фигур 33](#_Toc514881459)

[6.2. Тестирование добавления фигуры вправо и из конструкций If-Then-Else, Repeat, For и While 35](#_Toc514881460)

[6.2. Тестирование алгоритмов смещения и поиска свободного столбца. 37](#_Toc514881461)

[6.3. Тестирование функционала добавления текста, удаления блока и экспорта в изоюражения 38](#_Toc514881462)

[6.4. Вывод из прохождения тестирования 39](#_Toc514881463)

[7.РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ 40](#_Toc514881464)

[ЗАКЛЮЧЕНИЕ 42](#_Toc514881465)

[СПИСОК ЛИТЕРАТУРЫ 43](#_Toc514881466)

[ПРИЛОЖЕНИЕ 1 44](#_Toc514881467)

[ПРИЛОЖЕНИЕ 2 44](#_Toc514881468)

# ВВЕДЕНИЕ

Задание алгоритмов с помощью блок-схем оказалось очень удобным средством изображения алгоритмов и получило широкое распространение. Блок-схема алгоритма — графическое изображение алгоритма в виде связанных между собой с помощью стрелок (линий перехода) и блоков — графических символов, каждый из которых соответствует одному шагу алгоритма. Внутри блока дается описание соответствующего действия.

Многие специалисты в области теории программирования считают, что графическое представление алгоритмов в соответствии с *ГОСТ 19.701-90* скрывают структуру структурированной программы, представляя структурированность недостаточно очевидной. Поэтому для представления структурированных схем алгоритмов были предложены специальные методы графических обозначений.

Для описания алгоритма любой сложности используются графические схемы алгоритмов. Основными принято считать схемы по Госту, схемы по Дамке и диаграмы Наси-Шнайдермана.

К недостаткам ГОСТа можно отнести возможность чертить неструктурированные схемы алогритма. Данного недостатка лишены схемы по Дамке и диаграмы Наси-Шнайдермана.

Данная курсовая работа посвящена разработке программного средства для построения схем алгоритмов по Дамке.

В ходе выполнения данной работы будут рассмотрены и применены следующие подходы:

1. Работа с графикой в Delphi
2. Разработка понятного, удобного, а главное быстрого интерфейса графического редактора
3. Работа с различными форматами растрового и векторного изображения

В пояснительной записке отображены этапы написания курсовой работы

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству;
2. Анализ требований к программному средству и разработка функциональных требований;
3. Проектирование программного средства;
4. Создание (конструирование) программного средства;
5. Тестирование, проверка работоспособности и анализ полученных результатов;
6. Руководство по установке и использованию.
7. **АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ**

## Метод дамке представления алгоритмов

«Эта одна книга “М. Дамке «Операционные системы микроЭВМ»” (прим. автора) по факту - одна из немногих, которые вообще были выпущены и стали настольными книгами тех времён. Во времена CP/M не было сегодняшних интернетов, где бы могли её распространять. Ну и не было stackoverflow, где можно было бы почувствовать себя крутым программистом, не зная ничего о программировании. Так что способ по факту использовался не в одной книге. Просто ещё появились ISO/IEC и ГОСТ, а заодно стало модным кодить без проектирования и документирования в связи с наплывом в профессию школотронов.»

Оношко Д.Е.

М. Дамке предложил для конструкций структурированных схем алгоритмов специальные обозначения.

**Три основных конструкции**структурного программирования изображаются следующим образом:

1) ***Функциональный блок***по-прежнемуобозначается прямоугольником

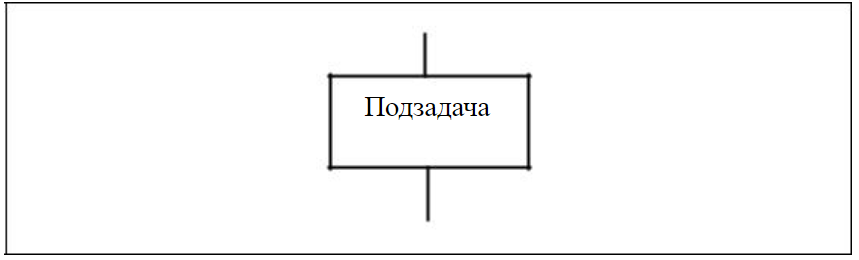


Рисунок 1.1 – Представление функционального блока по методу Дамке

2)***Конструкция If-Then-Else***изображается так, как иллюстрирует рисунок 1.2

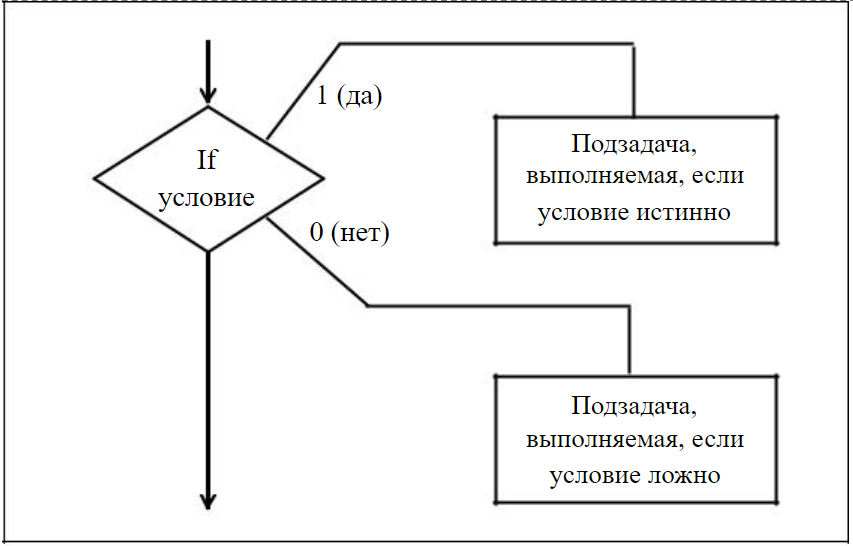
Элементы с выполняемыми действиями находятся справа от символа «Решение». Вход и выход из конструкции находятся соответственно сверху и снизу символя «Решение».

Рисунок 1.2 – Представление конструкции If-Then-Else по методу Дамке

3)***Конструкция Do-While(цикл с предусловием “Пока”)***изображается так, как показывает рисунок 1.3

Тело цикла выполняется до тех пор, пока условие истинно. Условие проверяется первым. Графически это изображается положением шестиугольника **над**выполняемым телом цикла.

Следует обратить внимание, что входы и выходы из всех конструкций метода Дамке находятся в левой части (сверху и снизу) графического представления конструкций. Расширения конструкций в правой части представления выходов не имеют.

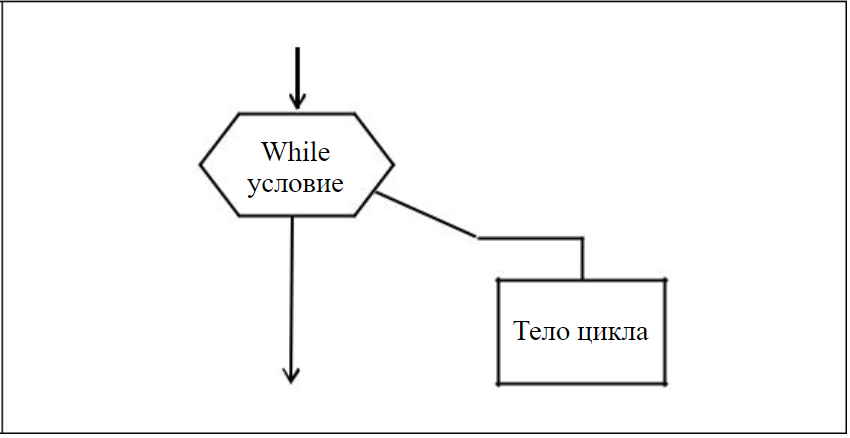


Рисунок 1.3 – Представление конструкции Do-While по методу Дамке

## Анализ существующих аналогов

В виду того, что рассматриваемый способ представления алгоритмов не пользуется популярностью, удобных программных средств для создания данных схем не существует. В качестве аналогов можно рассмотреть наиболее популярныег графические редакторы и сервисы.

**Draw.IO**

Один из самых удобных сервисов для разработки схем по Дамке. Разработчики - JGraph Ltd.

Плюсы:

* Не требует установки
* Стабильная работа сервиса
* Экспорт в большинство необходимых форматов

Минусы:

* Требует постоянного интернет-соединения
* Не предназначена для рисования схем по Дамке
* Бывают проблемы с открытием уже разработанных схем

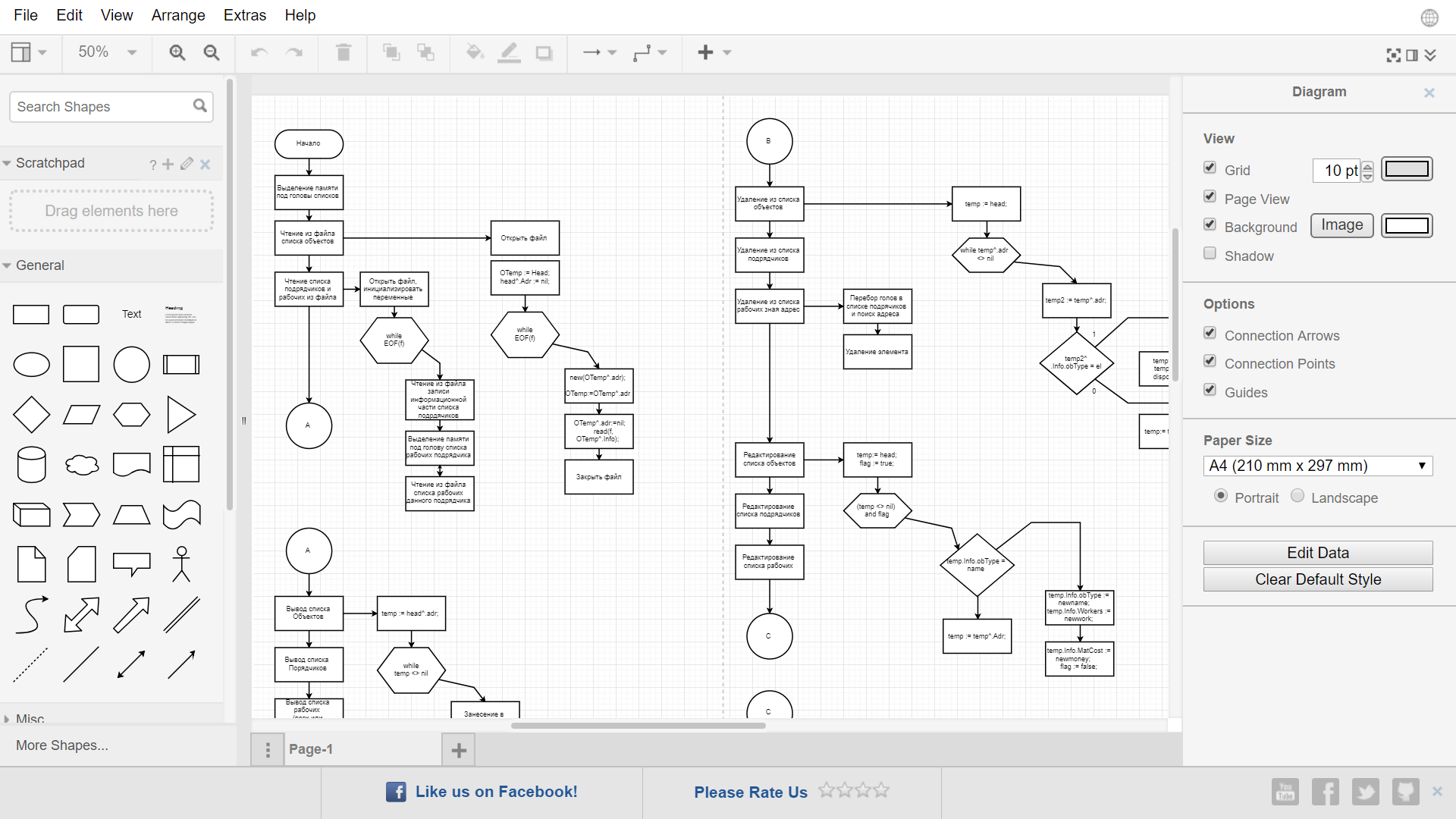


Рисунок 1.4 – Скриншот Draw.io

Популярный редактор Блок-схем от компании Gliffy Inc.

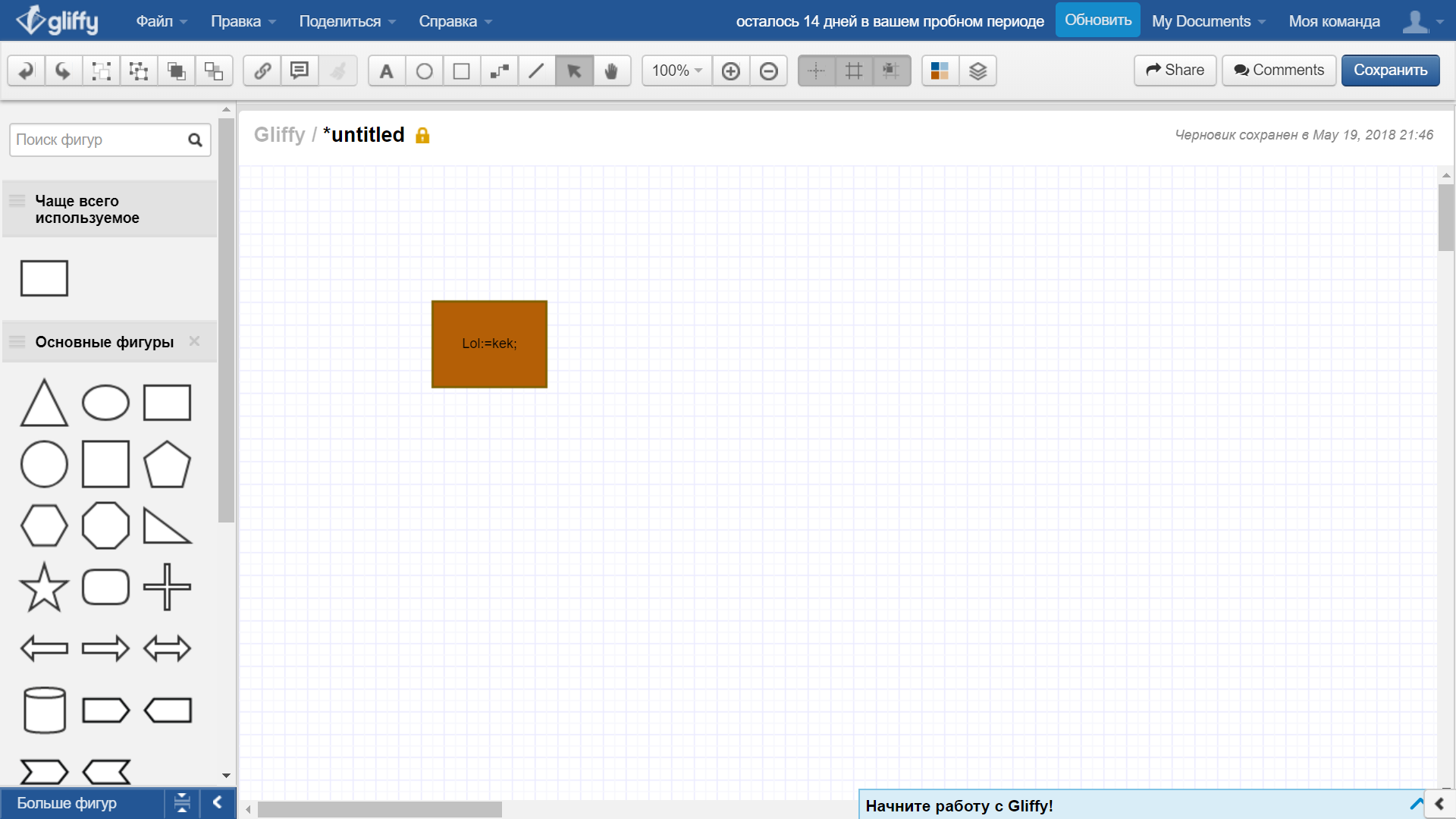


Рисунок 1.5 – Скриншот Gliffy.com

Плюсы:

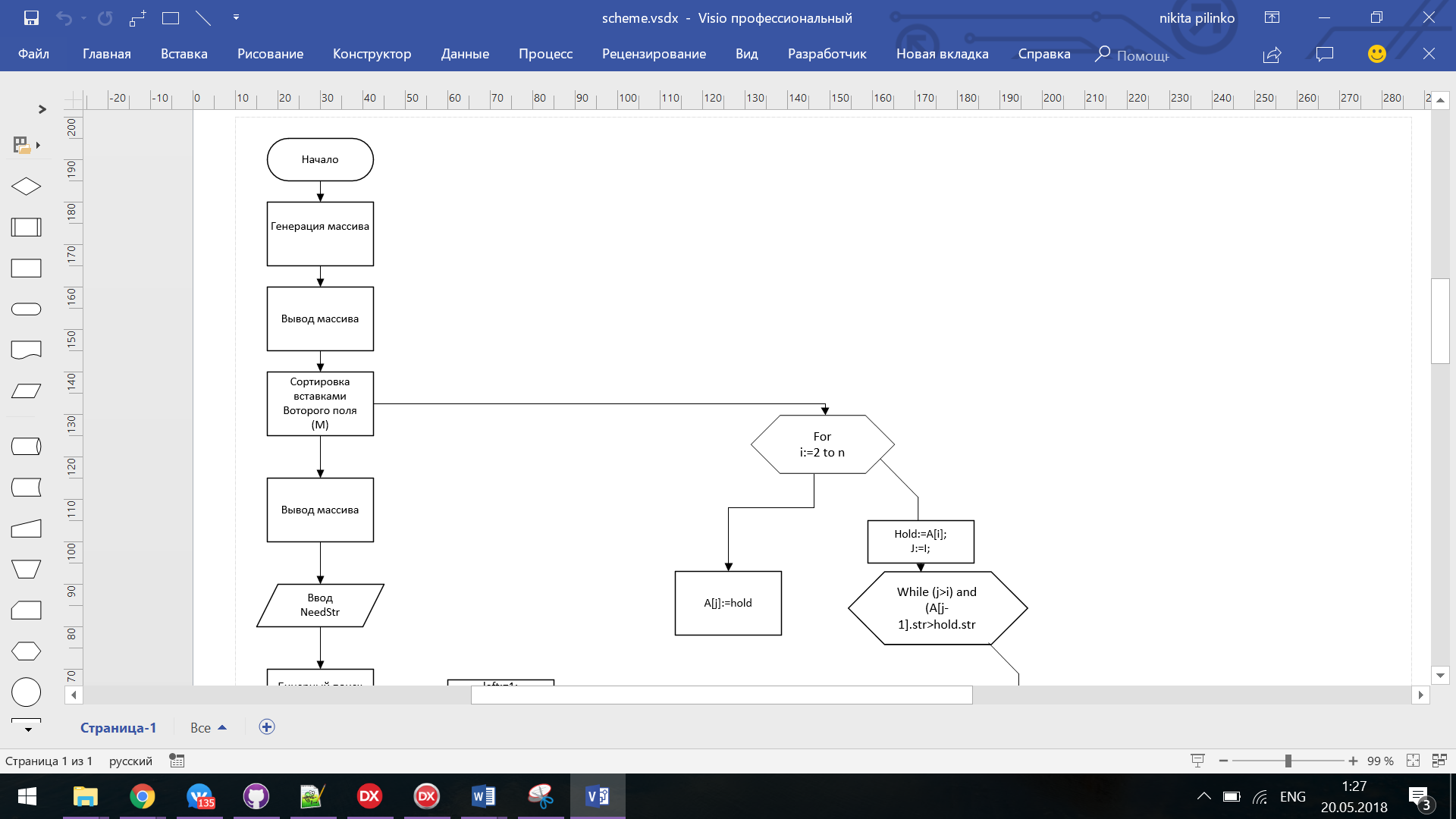
* Не требует установки
* Подробное руководство пользователя
* Экспорт в большинство необходимых форматов

Минусы:

* Требует постоянного интернет-соединения
* Не предназначена для рисования схем по Дамке
* Условно бесплатная – пробная версия (14 дней)

**MS Visio**

Стандартная программа из пакета Microsoft Office. Самая не пригодная к использованию программа для разработки схем по Дамке.



Плюсы:

* Большая палитра инструментов
* Автономная работа
* Экспорт в большинство необходимых форматов

Минусы:

* Не предназначена для рисования схем по Дамке
* Обладает интуитивно непонятным интерфейсом

## Постановка задачи

Так как проектируемое программное средство должно быть заточено под создание графических схем, необходимо создать максимально удобный дизайн взаимодействия с пользователем (UX). Программное средство должно автоматизированно прорисовывать основные конструкции и вести стрелки. Это должно положительно сказаться на скорости рисования схемы.

Так же будет реализована возмоность экспорта в векторные и растровые форматы изображения:

* Автоматизированное создание структуры данных, содержащей информацию для отрисовки блок схемы
* Сохранение исходников изображения в типизированный файл с возможностью дальнейшего использования;
* Открытие типизированного файла с исходниками;
* Экспорт изображение в векторный формат (svg);
* Экспорт изображения в растровые форматы (bmp, png);
* Операции копирования, вставки и отмены действия.
* Переключение языка (русский/английский)

Также в программе должно присутствовать руководство пользователя, включающее в себе инструкцию по использованию, описание программы и т.д.

В качестве языка программирования выбран язык Delphi.

# МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

## Представление векторного изображения в формате SVG

В отличие от растровых, векторные изображения состоят уже не из пикселей, а из множества опорных точек и соединяющих их кривых. Векторное изображение описывается математическими формулами и, соответственно, не требует наличия информации о каждом пикселе. Сколько ни увеличивай масштаб векторного изображения, вы никогда не увидите пикселей.Одним из самых распространенных форматов файлов векторной графики является формат SVG. Формат SVG предназначен для описание двумерной векторной и смешанной графики в текстовом формате XML.

Структура документа[2][3]:

* Первая строка – стандартный XML заголовок с указанием версии, кодировки.

Пример:

*<?xml version="1.0" encoding="UTF-8" standalone="no"?>*

* Вторая и третья строка – заголовок DOCTYPE, определяющий тип документа.

Пример:

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

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

* Четвертая строка – корневой элемент документ с указанием пространства имен SVG

Пример:

*<svg version="1.1"*

*baseProfile="full"*

*xmlns="http://www.w3.org/2000/svg"*

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

*xmlns:ev="http://www.w3.org/2001/xml-events"*

*width="100%" height="100%">*

* Далее идет остальной текст документа, завершающийся закрытием тега *</svg>*

Отрисовка основных фигур:

* **Описание путей**

Позволяет задать любую фигуру, описывая путь от начальной точки до конечной через промежуточные координаты. Строка с данными задается атрибутом d тега **path** и содержит команды, закодированные набором букв и чисел. Буквы – обозначают тип команды. Наиболее простые – M *(англ. moveto – переместить)*, L *(англ. lineto – нарисовать линию).* Цифры, чаще всего, содержат координаты точек по осям X и Y.

Пример: линия из точки (100,100) в точку (100,200)

*<path fill="none" stroke="black" d="M 100 100 L 100 200" />*

* **Прямоугольник**

Строка задается 4-мя основными атрибутами тега **rect**: координаты X,Y левой верхней точки (Атрибуты x, y), высота и ширина (Атрибуты height и width соответственно).

Пример:

*<rect fill="white" x="400" y="600" width="300" height="200" />*

* **Вывод текст**

Выводимый текст заключается в тег **text**,в котором в качестве атрибутов задаются свойства. Элементарные свойства для вывода текста – координаты левой верхней точки текста (атрибуты x, y).

Пример:

*<text x="30" y="12" >Syntax Diagrams Editor</text>*

Каждому тегу можно задать дополнительные свойства, описания и примеры которых находятся в документации по формату SVG.

## Описание функциональности ПС

Програмное средство должно обладать минмальным функционалом, достаточным для рисования схем по Дамке, но при этом обладать всеми необходимыми инструментами в том числе экспорт, настройка размеров холста.

Основные конструкции, необходимые для разработки схем по Дамке вынесены на ToolBar.

С самого запуска программы пользователю даётся возможность начинать работу. Блок начала алгоритма отрисовывается автоматически. Для навигации по блокам можно использовать стрелки, или явно указывать фигуру при помощи правого клика. Если на фигуре есть фокус, то по нажатию на соответсвующую кнопку ToolBarа будет отрисована выбранная конструкция. Для удаления блоков используется клавиша Delete. Так же реализована работа общепринятых клавишасочетаний Ctrl+Z, Ctrl+C, Ctrl+X, Ctrl+S.

Также программа предоставляет взаимодействие с меню В зависимости от того по какому элементу меню был произведен клик, программа может

* Создавать новый файл схемы
* Открывать существующий файл
* Экспортировать в векторные и растровые изображения
* Сохранять созданный файл в типизированный

## Спецификация функциональных требований

Основное функциональное требование - «Отображение фигур на полотне».

Спецификация данной функции может иметь следующий вид:

* Функциональный блок выглядит как Прямоугольник с прозрачным фоном и обводкой шириной в 3 px. Как и все остальные фигуры имеет цвет clBlack. Представлен на рисунке 2.1

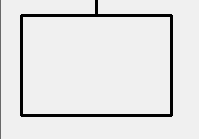


Рисунок 2.1 – Функциональный блок

* Конструкция If-Then-Else – самая сложная для отрисовки фигура, может иметь три выхода: при истинном и ложном значении и так же следующие за ней блоки.

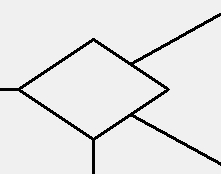


Рисунок 2.2 –конструкция If-Then-Else

* Конструкция Do – While, цикла с параметром и Repeat - until имеют схожее строение – это вытянутый по горизонтали шестиугольник, разница лишь в исходящих из блока линий – у Do – While она начинается из нижней левой границы как показано на рисунке 2.3, у Repeat – until из верхней левой.

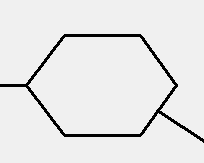


Рисунок 2.3 –конструкция Do-While

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## Проектирование динамических структур данных

Поскольку память под данные отводятся во время выполнения работы программы, возникла необходимость использовать динамические структуры данных. Использование динамических структур позволяет организовать иерархию объектов, упрощает создание и удаление. Однако нужно предусмотреть следующие особенности:

* Функциональные блоки могут иметь различное количество дочерних компонентов
* Информативную часть нужно сохранять в файл, при этом сохранить связанность фигур между собой.

Поэтому было принято решение использовать в качестве записи ссылочной части списка запись с несколькими полями адреса. Структура представлена на рисунке 3.1.

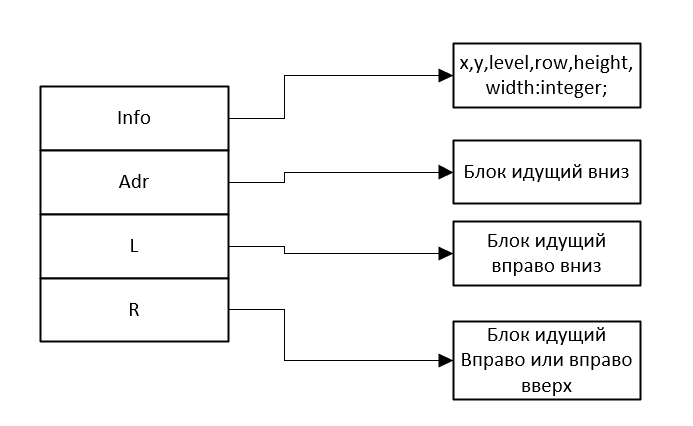


Рисунок 3.1 – структура основного списка программы

Так как фигуры могут иметь произвольную степень вложенности, то для прохода по структуре используются рекурсивные алгоритмы. Структура записи схемы алгоритма представлена на рисунке 3.2.

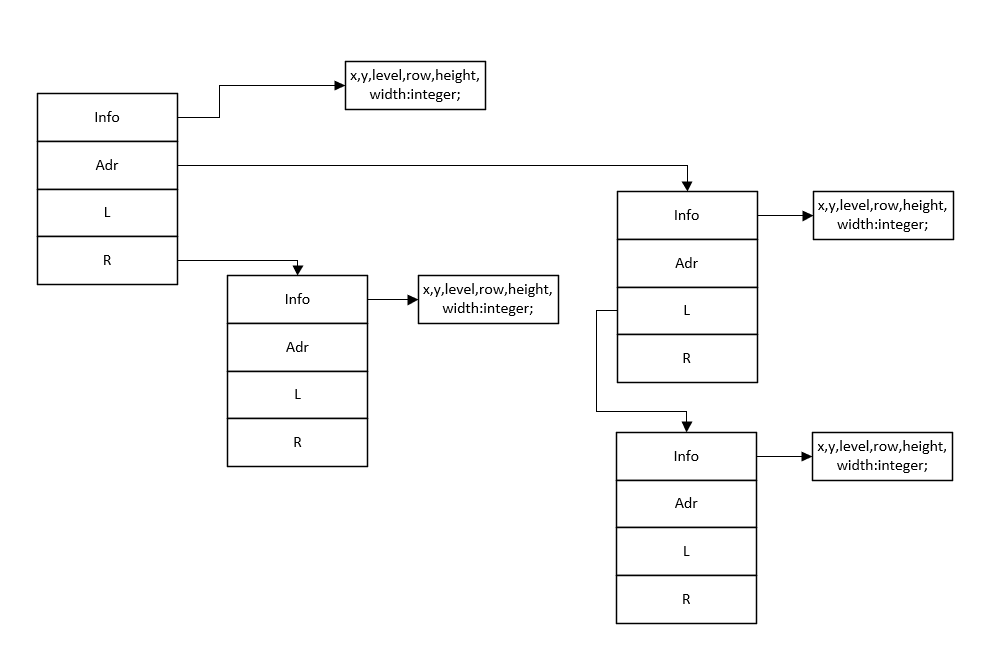


Рисунок 3.2. – Пример связанности списка

## Разработка алгоритма вставки фигуры

При активном фокусе на блоке, после нажатия на одну из кнопок ToolBara, управление передаётся процедуре, которой передаётся только тип фигуры. Процедура формирует запись типа TFigureinfo, содержащую всю необходимую информацию для формирования записи. Схема представлена на рисуноке 3.4

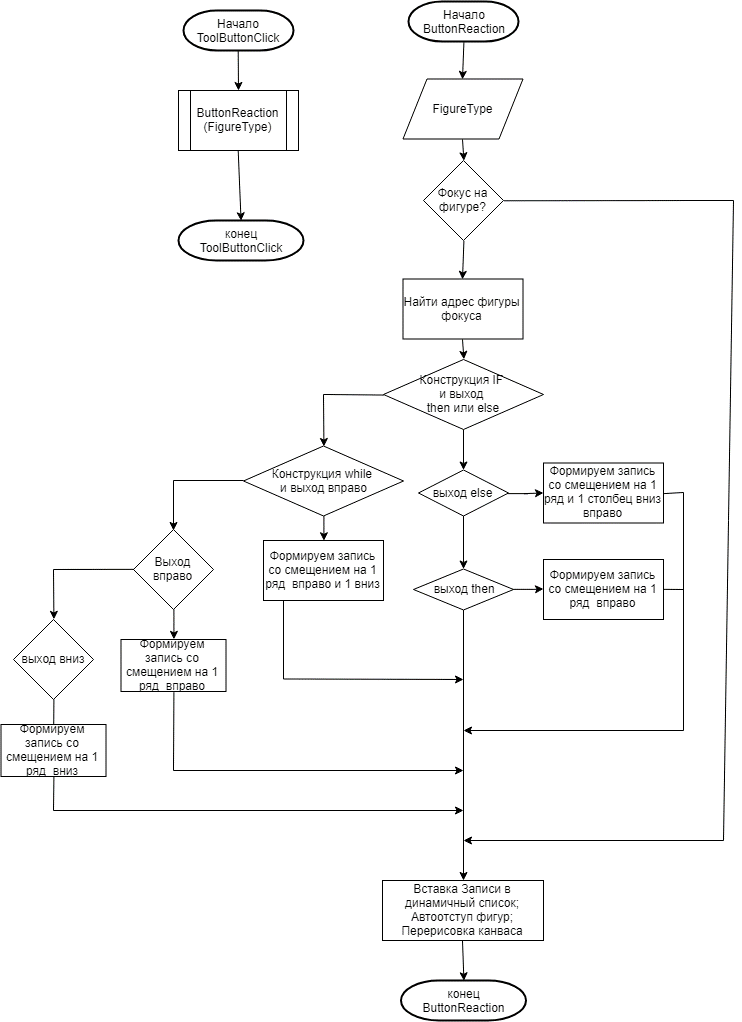


Рисунок 3.4 -схема алгоритма реакции на нажатие

## Разработка алгоритма отрисовки фигур

Используемый подход отрисовки фигур имеет наименьшие затраты по памяти. Так как каждая запись содержит лишь координаты начала, высоту и ширину блока, тип блока, процедура вызывает внутри себя процедуры, которые отрисовывают необходимые фигуры внутри заданного прямоугольника. Метод Дамке предполагает произвольную степень вложенности блоков, необходимо использовать рекурсивный алгоритм прохода по записи. Схема алгоритма представлена на Рисуноке 3.5

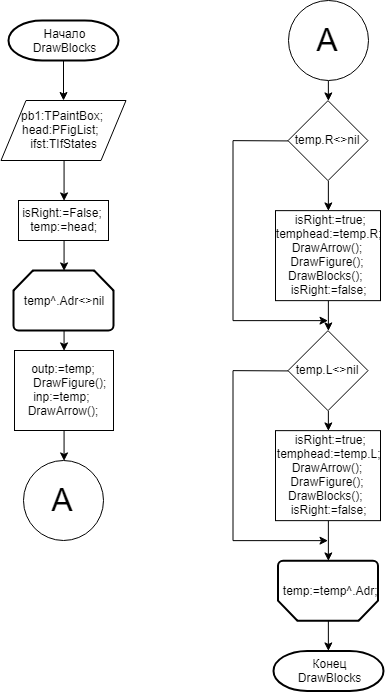


Рисунок 3.5 – схема алгоритма отрисовки

## Разработка алгоритма получения координат по Y

Чтобы предотвратить наложение фигур разной степени вложенности возникла необходимостьв разработки алгоритма, который бы позволил получать координаты свободного ряда. Функция, реализующая данный алгоритм вызывается каждый раз, когда блок вставляется вниз.

Основная идея состоит в определении Максимальной координаты Y блоков большего уровня вложенности. На основании полученного значения, и константы стандартного отсутпа очередная вставляемая фигура получает Y координату. Работа алгоритма представлена на рисунке 3.6а, 3.6б

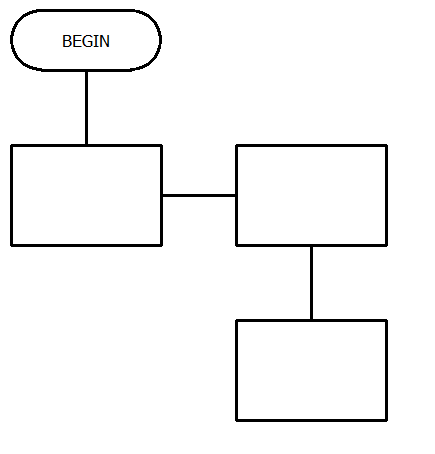
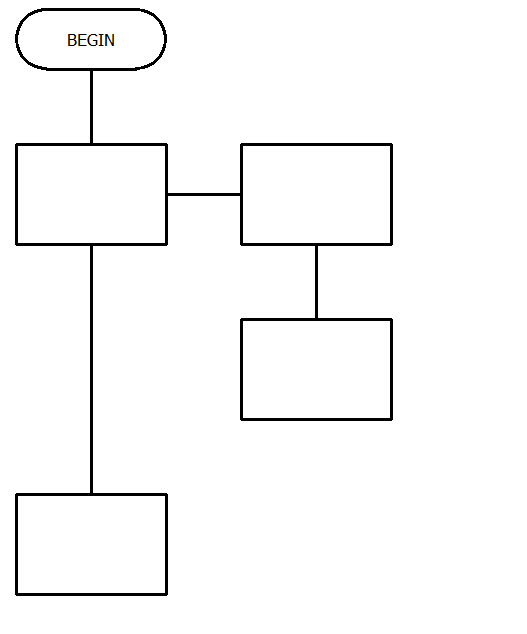
 

Рисунок 3.6 а,б –исходная схема и результат работы алгоритма

Как видно из рисунков, применение такого алгоритма исключает возможность наложения блоков друг на друга

## Разработка алгоритма сдвига блоков

После вставки блока, вложенность которого больше единицы так же может возникнуть ситуация, при которой блоки наложаться друг на друга. Для избежания этого был разработан алгоритм, который сдвигает фигуры, обладающие меньшей степенью вложенности. Так же меняется в соответствии со сдвигом поле ‘row’ сдвинутых блоков, алгоритм представлен на Рисуноке 3.7

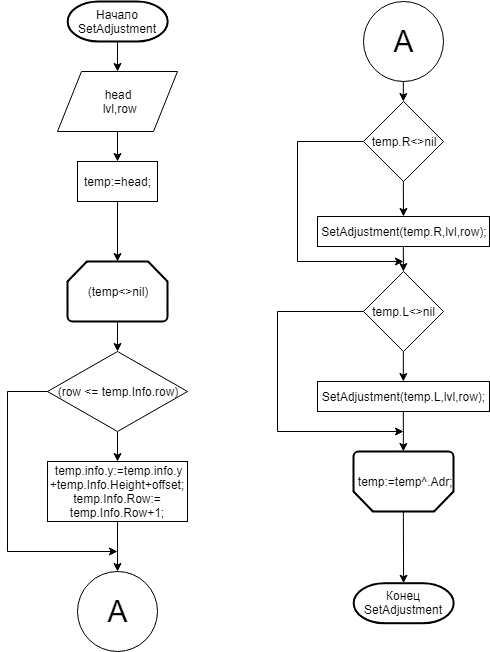


Рисунок 3.7 – схема алгоритма сдвига блоков

Рисунок 3.10 – Cхема алгоритма отрисовки линий

## Разработка алгоритма вставки текста

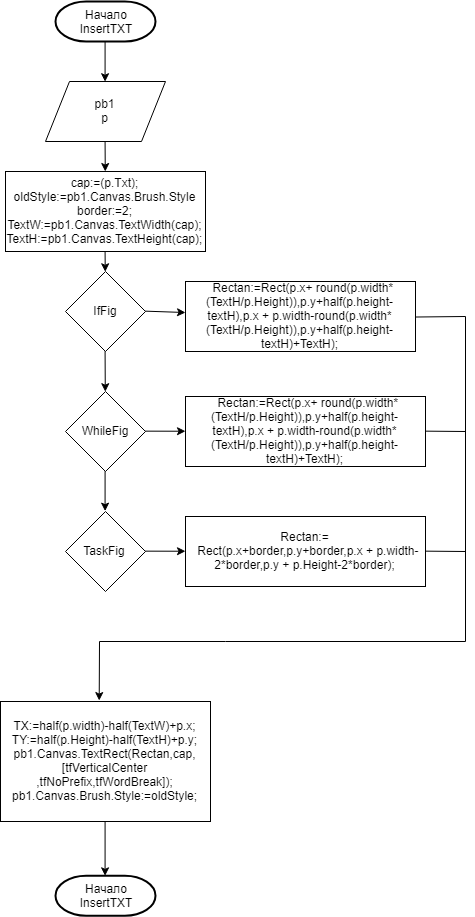
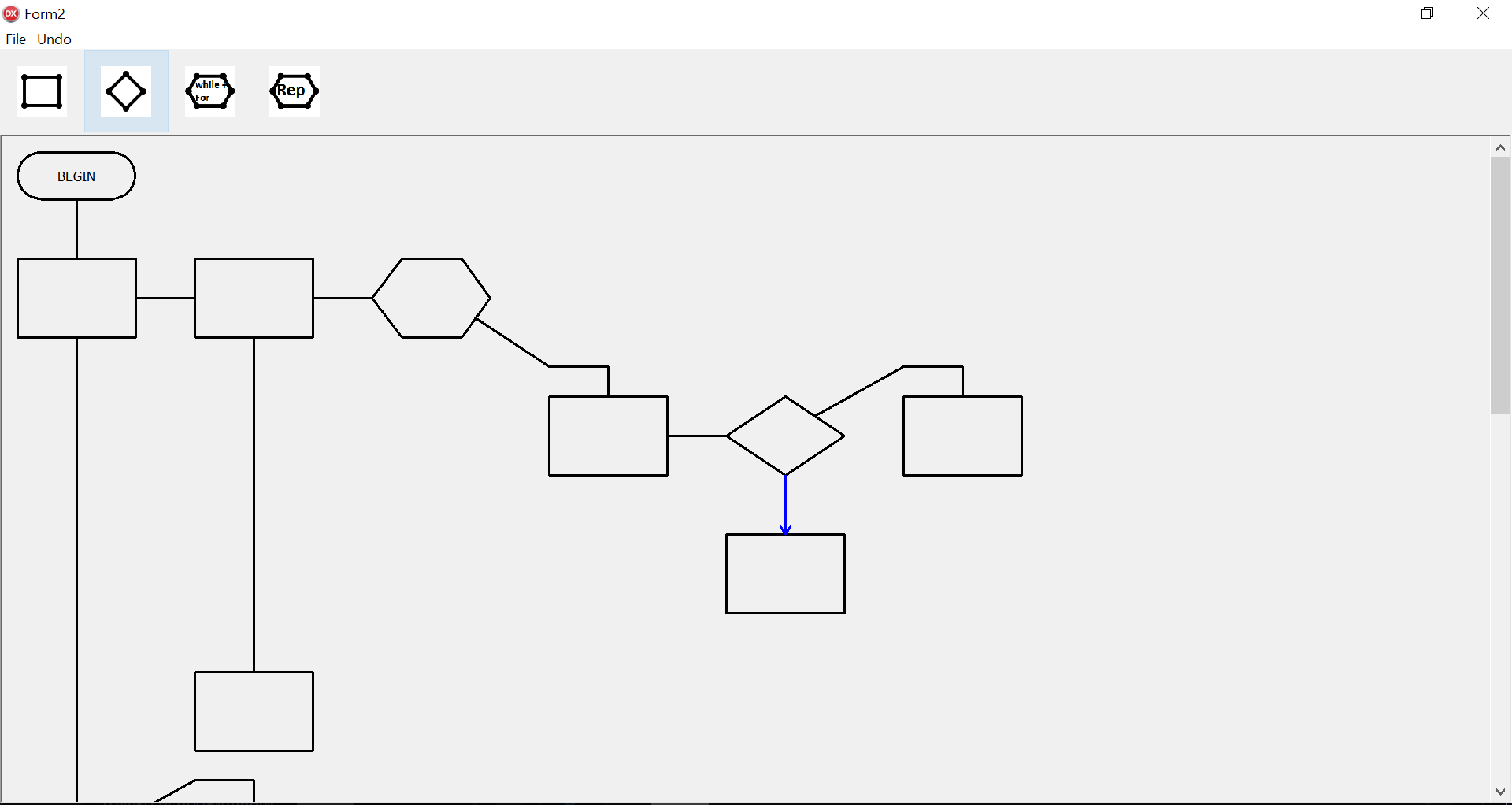
Текст должен вставляться по середине блока по вертилки и горизонтали. Для каждой фигуры область вставки текста различная. Процедура определяет границы области, при помощи определения параметров текста функциями Canvas. Для конструкции IF-Then-Else область расчитывается по математической

Рисунок 3.8 – схема алгоритма вставки текста.

# СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА

## Разработка интерфейса программы

Интерфейс программы предельно простой, представляет собой главную форму с меню, tool bar и областью рисования. Интерфейс представлен на Рисуноке 4.1. Описание основных подпрограмм представленно в таблице 4.1



Интерфейс разработанной программы - Рисунок 4.1

В форме использованы следующие элементы:

* TImagelist – Хранит упорядоченный набор иконок кнопок
* TMainMenu -Позволяет назначит меню с выпадающим списком
* TScrollBox – Подложка для виртуального расширения поля
* TMemo – многострочное поле для редактирования текста.
* TPaintBox - Область рисования
* TToolBar – Панель инструментов

Таблица 4.1 – основные подпрограммы модуля Main

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
| clrscreen | $R \*.dfm | procedure clrscreen(pb1:TPaintBox); | pb1 |  |
| FormCreate | Событие при создании формы | procedure TKek.FormCreate(Sender: TObject); | Sender | Объект, который сгенерировал событие |
| getCharParams | Процедура получает ширину и высоту шрифта при помощи функций Windows | procedure TKek.getCharParams(var Chrwidth, Chrheight:Integer); | Chrwidth | Ширина |
| Chrheight | Высота |
| FormKeyDown |  | procedure TKek.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState); | Sender | Объект, который сгенерировал событие |
| Key |  |
| Shift |  |
| pb1MouseDown | Обработка нажатие мыши | procedure TKek.pb1MouseDown(Sender: TObject; Button:TMouseButton; Shift: TShiftState; X, Y: Integer); | Sender | Объект, который сгенерировал событие |
| Button | кнопка |
| Shift |  |
| X |  |
| Y |  |
| ButtonReaction | Процедура | procedure ButtonReaction(Figure: TFigType); | Figure | Тип фигуры |
| pb1Paint | Процедура отрисовки канваса | procedure TKek.pb1Paint(Sender: TObject); | Sender | Объект, который сгенерировал событие |
| btntaskClick | Обработка клика, вызывает ButtonReaction | procedure TKek.btntaskClick(Sender: TObject); | Sender | Объект, который сгенерировал событие |
| btnWhileClick | Обработка клика, вызывает ButtonReaction | procedure TKek.btnWhileClick(Sender: TObject); | Sender | Объект, который сгенерировал событие |
| btnIfClick | Обработка клика, вызывает ButtonReaction | procedure TKek.btnIfClick(Sender: TObject); | Sender | Объект, который сгенерировал событие |
| ScrollBox1MouseWheelDown | Обработка прокрутки | procedure TKek.ScrollBox1MouseWheelDown(Sender: TObject;Shift: TShiftState; MousePos: TPoint; var Handled: Boolean); | Sender | Объект, который сгенерировал событие |
| Shift | Состоние кнопки Shift |
| MousePos | Позиция мыши |
| Handled |  |
| ScrollBox1MouseWheelUp | Обработка прокрутки | procedure TKek.ScrollBox1MouseWheelUp(Sender: TObject; Shift:TShiftState; MousePos: TPoint; var Handled: Boolean); | Sender | Объект, который сгенерировал событие |
| Shift | Состоние кнопки Shift |
| MousePos | Позиция мыши |
| Handled |  |
| ToolButton1Click | Обработка клика, вызывает ButtonReaction | procedure TKek.ToolButton1Click(Sender: TObject); | Sender | Объект, который сгенерировал событие |

Основное назначение модуля – обработка событий, которые генерирует пользователь.

## Описание модуля Draw

Основное назначение модуля – отрисовывать фигуры и стрелки на объекте канвас компонента Paintbox. Модуль осуществляет вставку текста в фигуру.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1. **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
| TextUtil | Определяет длинну введённой строки | function TextUtil(cap:string):string; | cap |  |
| InsertTXT | Процедура, отрисовывающая текст на  Канвасе | procedure InsertTXT(pb1:TPaintBox;var p:TFigureInfo); | pb1 | Объект, который сгенерировал событие |
| getCharParams | Получает ширину и высоту символов | procedure TKek.getCharParams(var Chrwidth, Chrheight:Integer); | p | Ширина |
| IsEmptyTXT | Возвращает true если запись не содержит текста. | function IsEmptyTXT(p:TFigureInfo):Boolean; | p | Высота |
| HorizontalAdjust | Выравнивает блоки п огоризонтали | procedure HorizontalAdjust(head: PFigList; NewSize:PFigList); | head | Голова |
| NewSize |  |
| drawRect | Рисует прямоугольник | procedure drawRect(pb1:TPaintBox;p:TFigureInfo;Color:Tcolor); | pb1 | Объект, который сгенерировал событие |
| p |  |
| Color | Колонка |
| drawA | Обработка нажатие мыши | procedure drawA(pb1:TPaintBox;p:TArrowInfo; Color:TColor); | pb1 | Объект, который сгенерировал событие |
| p | кнопка |
| Color | Колонка |
| defaultDraw | Обработка клика | procedure defaultDraw(head:PFigList;pb1:TPaintBox); | head | Голова |
| DrawIF | Отрисовывает блок IF-Then-else | procedure DrawIF(pb1:TPaintBox; p:TFigureInfo; color:TColor); | pb1 | Область рисования |
| p |  |
| color | Колонка |
| DrawWhile | Рисует блок While | procedure DrawWhile(pb1:TPaintBox; p:TFigureInfo; color:TColor); | pb1 | Область рисования |
| DrawFigure |  | procedure DrawFigure(pb1:TPaintBox; p:TFigureInfo; color:TColor); | pb1 | Область рисования |
| p |  |
| DrawBlocks | Рекурсивная процедура, вызывающая отрисовку блоков и стрелок | procedure DrawBlocks(pb1:TPaintBox; head:PFigList; varmaxX,maxY:integer; ifst:TIfStates); | pb1 | Область рисования |
| head | Голова |
| axX |  |
| maxY | максимальный |
| ifst |  |
| DrawDirectArrows | Рисует стрелки для фигур фокуса ввода | procedure DrawDirectArrows(pb1:TPaintBox; p:TFigureInfo;left:boolean; ifState:TIfStates); | pb1 | Область рисования) |
| p |  |
| left |  |
| ifState | Статус |
| DrawArrow | Процедура отрисовывает стрелки | procedure DrawArrow(pb1:TPaintBox;inp,outp:TFigureInfo;isRight:Boolean;ifstate:TIfStates); | pb1 |  |
| inp | Исходящая фигура |
| :TFigureInfo;isRight |  |
| Boolean;ifstate | Статус |
|  |  |  | TIfStates |  |

## 4.3 Описание модуля Lists

## Модуль Lists осуществляет работу с динамическими структурами: создание головы списка, поиск фигуры по координатам, поиск поиск адреса по данным и так далее. Полный список подпрограмм прелставлен в таблице 4.3

Таблица 4.3 – Список подпрограмм модуля Lists

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
| createFigureHead |  | procedure createFigureHead(var head:PFigList); | head | Голова |
| GetAdjust |  | function GetAdjust(head:PFigList;lvl:integer;p:TFigureInfo;vard:Integer):integer; | head | Голова |
| lvl |  |
| p |  |
| :Integer |  |
| GetAdj |  | function GetAdj(head:PFigList;lvl:integer; varmax,d:integer):integer; | head | Голова |
| lvl |  |
| ax |  |
| d |  |
| eqInfo |  | function eqInfo(a,b:TFigureInfo):Boolean; | a |  |
| b |  |
| GetRow |  | function GetRow(head:PFigList;lvl:integer;p:TFigureInfo):integer; | head | Голова |
| lvl |  |
| p |  |
| GetR |  | function GetR(head:PFigList;lvl:integer; var max:Integer):integer; | head | Голова |
| lvl |  |
| max | максимальный |
| JustAdjust |  | procedure JustAdjust( head:PFigList); | head | Голова |
| SetAdjustment |  | procedure SetAdjustment(const head:PFigList; lvl,row:Integer); | head | Голова |
| lvl |  |
| row | Строка |
| GetParentAdr |  | function GetParentAdr(const head:PFigList; P:PFigList):PFigList; | head | Голова |
| P |  |
| GetP |  | function GetP(const head:PFigList; P:PFiglist):PFigList; | head | Голова |
| P |  |
| GetAdr |  | function GetAdr(const head:PFigList; info:TFigureInfo):PFigList; | head | Голова |
| info |  |
| Levelwidth |  | function Levelwidth(head:PFigList; lvl:integer):Integer; | head | Голова |
| lvl |  |
| insertFigure |  | procedure insertFigure(head:PFigList;points:TFigureInfo;LRec:TFigureInfo); | head | Голова |
| points |  |
| LRec |  |
| GetClickFig | Обработка клика | function GetClickFig(x,y:integer; const head:PFigList; varselected:Boolean):TFigureInfo; | x |  |
| y |  |
| head | Голова |
| elected |  |
| GClickFig | Обработка клика | function GClickFig(x,y:integer; const head:PFigList; varselected:Boolean):TFigureInfo; | x |  |
| y |  |
| head | Голова |
| elected |  |
| aHead |  | function aHead(head: PFigList;lol:TFigureInfo):boolean; | head | Голова |
| lol |  |
| SetLevelWidth |  | procedure SetLevelWidth(var head:PFigList;lvl,width,increment:Integer); | head | Голова |
| lvl |  |
| width | Ширина |
| increment |  |
| DeleteBlock |  | procedure DeleteBlock(head: PFigList; P:TFigureInfo); | head | Голова |
| P |  |
| CreateNode |  | procedure CreateNode(head: PFigList;Points:TFigureInfo;LRec:TFigureInfo); | head | Голова |
| Points |  |
| LRec |  |
| LeftBorn |  | function LeftBorn(const head:PFigList;p:TFigureInfo):boolean; | head | Голова |
| p |  |
| LeftB |  | function LeftB(const head:PFigList;p:TFigureInfo;varex:Boolean):boolean; | head | Голова |
| p |  |
| x |  |
| createleft |  | procedure createleft(head: PFigList;Points:TFigureInfo;LRec:TFigureInfo); | head | Голова |
| Points |  |
|  |  |  | LRec |  |

# 5.ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

Проведено тестирование программного средства. Тестирование программного средства производилась на компьютере с установленной операционной системой Windows 10.

## Тестирование функционала добавления фигур

Таблица 5.1 - тестирование функционала добавления фигур

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер**  **теста** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 0 | Добавление блока следования | 1. 1. Выбор фигуры из которой будет следовать блок а также выбор направление при помощи стрелок. 2. 2. Нажатие на необходимую добавляемую фигуру | Фигура добавится, прорисуется линия соединяющая блок | Тест пройден! |
| 1 | Добавление блока If-Then-Else | 1. 1. Выбор фигуры из которой будет следовать блок а также выбор направление при помощи стрелок.   2. Нажатие на необходимую добавляемую фигуру | Фигура добавится, прорисуется линия соединяющая блок | Тест пройден! |
| 2 | Добавление блока While/For/Repeat | 1. 1. Выбор фигуры из которой будет следовать блок а также выбор направление при помощи стрелок.   2. Нажатие на необходимую добавляемую фигуру | Фигура добавится, прорисуется линия соединяющая блок | Тест пройден! |

Программа прошла данный этап тестирования, никаких проблем, ошибок и багов выявленно не было.

## 5.2. Тестирование добавления фигуры вправо и из конструкций If-Then-Else, Repeat, For и While

Таблица 5.1 - тестирование функционала добавления фигур

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер**  **теста** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 0 | Добавление блока вправо | 1. 1. Выбор фигуры из которой будет следовать блок а также выбор направление при помощи стрелок. 2. 2. Нажатие на необходимую добавляемую фигуру | Фигура добавится, прорисуется линия соединяющая блок | Тест пройден! |
| 1 | Добавление блока из блока If-Then-Else вправо вверх | 1. 1. Выбор фигуры из которой будет следовать блок а также выбор направление при помощи стрелок.   2. Нажатие на необходимую добавляемую фигуру | Фигура добавится, прорисуется линия соединяющая блок | Тест пройден! |
| 2 | Добавление блока из блока If-Then-Else вправо вниз | 1. 1. Выбор фигуры из которой будет следовать блок а также выбор направление при помощи стрелок.   2. Нажатие на необходимую добавляемую фигуру | Фигура добавится, прорисуется линия соединяющая блок | Тест пройден! |
| 3 | Добавление блока из блока while/for | 1. 1. Выбор фигуры из которой будет следовать блок а также выбор направление при помощи стрелок. 2. 2. Нажатие на необходимую добавляемую фигуру | Фигура добавится, прорисуется линия соединяющая блок | Тест пройден! |
| 4 | Добавление блока из блока Repeat-until | 1. 1. Выбор фигуры из которой будет следовать блок а также выбор направление при помощи стрелок. 2. 2. Нажатие на необходимую добавляемую фигуру | Фигура добавится, прорисуется линия соединяющая блок | Тест пройден! |

## 5.3.Тестирование алгоритмов смещения и поиска свободного столбца.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер**  **теста** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 0 | Вставка вниз, когда на следующем ряду уже есть фигура | 1. 1. Выбор фигуры из которой будет следовать блок а также выбор направление при помощи стрелок. 2. 2. Нажатие на необходимую добавляемую фигуру | Алгоритм расчитает смещение по которому необходимо размещать фигуру | Тест пройден! |
| 1 | Добавление блока вниз при наличии блока меньшей степени вложенности занимающей необходимый ряд | 1. 1. Выбор фигуры из которой будет следовать блок а также выбор направление при помощи стрелок.   2. Нажатие на необходимую добавляемую фигуру | Алгоритм сместит фигуру меньшей степени вложенности, тем самым освободив ряд для вставки фигуры большей степени вложенности | Тест пройден! |

В результате тестирования были выявленны некоторые недочёты со смещением блоков, которые впоследствии были исправлены.

## Тестирование функционала добавления текста, удаления блока и экспорта в изоюражения

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер**  **теста** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 0 | Вставка текст в блоки программы | 1. 1. Выбор фигуры, в которую будет вставляться текст 2. 2. Нажатие на Enter 3. 3. Редактирование текста в TMemo 4. 4.Нажатие на произвольное место TPaintBox для заверщения редактирования и вставки текста | Текст вставляется в выбранную ячейку | Тест пройден! |
| 1 | Удаление блока | 1. 1. Выбор фигуры которая подлежит удалению 2. 2. Нажатие кнопки Delete | Блок удалиться со всеми вложенными блоками | Тест пройден!(от бдока осталось только выделение) |

## Вывод из прохождения тестирования

Программа успешно прошла все тесты, что показывает корректность работы программы и соответствие функциональным требованиям.

# РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ

Для начала работы с программой необходимо открыть файл DamkeEditor.exe Интерфейс показан на рисунке 7.1



Рисунок 7.1 – Интерфейс программы

Для добавления фигур используются кнопки с соответсвующими блокам графическими представлениями. Кнопки выделены на рисунке 7.2



Рисунок 7.2 – Кнопки для добавлния фигур

Но нажатия на кнопки блоков недостаточно, необходимо предварительно выбрать фигуру, к которой присоединяется создаваемый блок а также выбрать направление. Например на рисунке 7.3 выбран блок If-Then-Else и направление вниз вправо из него.



Рисунок 7.3 Выбор направления добавления фигуры

Для удаления достаточно придать фигуре фокус (нажать на неё), после чего можно нажать на кнопку Delete и фигура удалиться как и все её дочерние элементы(Элементы ниже и справа)

Программа предусматривает экспорт в изображения в форматы png и bmp. Для экспорта необходимо раскрыть выпадающий список TMainMenu, там нажать на кнопку экспорта и выбрать необходимое разрешение. Пример показан на рисунке 7.4

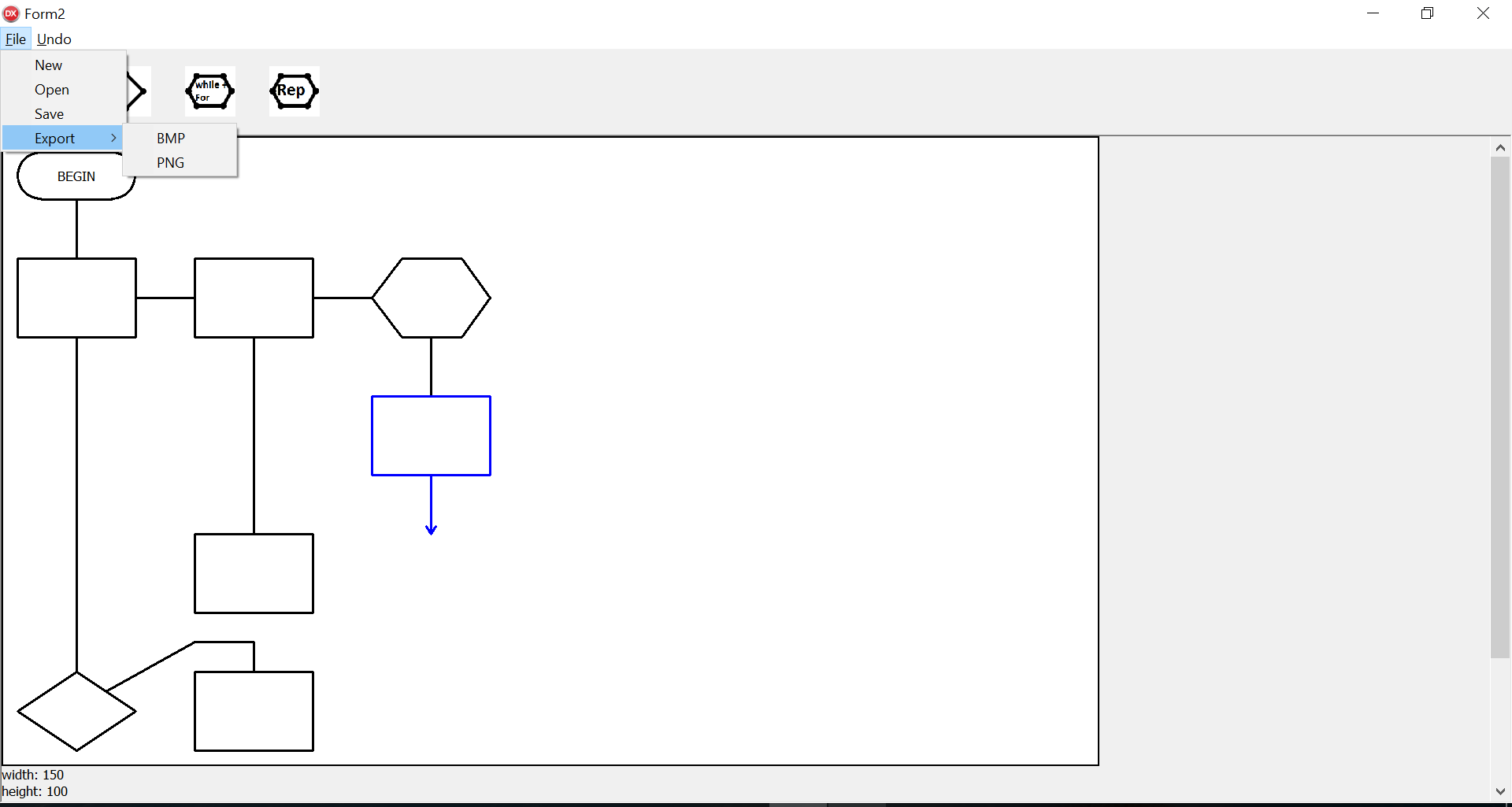


Рисунок 7.4 – Расположение кнопок для экспорта изображения

# ЗАКЛЮЧЕНИЕ

В процессе выполнения курсового проекта было разработано приложение, имеющий минимальный функционал требуемый для создания, редактирования и экспорта блок схем, созданных по методу Дамке.

При тестировании и отладке не были выявлены случаи некорректной работы программы, однако большинство из них были исправлены.

Простой и удобный пользовательский интерфейс позволяет пользователю легко привыкнуть к приложению и начать рисовать схемы.

Написанный код легко модифицируется, а изменения вида может быть модифицирован путем изменения констант. В дальнейшем возможны улучшения и доработки, вносящие новый функционал в программу.

В процессе разработки, я изучил принципы работы с графикой в языке программирования Delphi, принцип работы с растровыми форматами BMP и PNG, принципы правильной организации пользовательского интерфейса. Полученные знания и опыт позволят легко применить их в новых проектах.

Разработанное программное средство является одним из немногих средств, позволяющих быстро качественно и правильно нарисовать схему алгоритма методом Дамке, обладая минимальными знаниями в принципах построения. Потому разботанное средство рекоммендовано к использованию студентам, использующим данный способ в своих отчётах

# СПИСОК ЛИТЕРАТУРЫ

[1] Глухова, Л.А. Основы алгоритмизации и программирования: учебное пособие. В 2 Ч. / Л.А. Глухова. – БГУИР, 2006 – Ч. 1. – 195 с.

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

[3] Help for RAD Studio 10.2 Tokyo [Электронный ресурс] Режим доступа: http://docwiki.embarcadero.com/RADStudio/Tokyo/en/Main\_Page— Дата доступа: 16.04.18

[4] А.Н.Вальвачев, К.А.Сурков, Д.А.Сурков, Ю.М.Четырько. Программирование на языке Delphi. Учебное пособие[Электронный ресурс] Режим доступа: http://www.rsdn.ru/?summary/3165.xml. Дата доступа: 20.04.18

# ПРИЛОЖЕНИЕ 1

**Схема алгоритма работы программы**

# ПРИЛОЖЕНИЕ 2

**Исходный код программы**

**unit Unit2;**

interface

uses

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

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

Vcl.Menus,Vcl.Buttons, Lists, System.ImageList, Vcl.ImgList, Vcl.ComCtrls,

Vcl.ToolWin,Types\_const, DrawItems, Vcl.Imaging.pngimage;

type

TKek = class(TForm)

mm1: TMainMenu;

File1: TMenuItem;

New1: TMenuItem;

Open1: TMenuItem;

Save1: TMenuItem;

Export1: TMenuItem;

Undo1: TMenuItem;

ScrollBox1: TScrollBox;

pb1: TPaintBox;

tlb1: TToolBar;

il1: TImageList;

btn1: TToolButton;

btn2: TToolButton;

btnWhile: TToolButton;

lbl1: TLabel;

mmoInput: TMemo;

ToolButton1: TToolButton;

mniExportBmp: TMenuItem;

dlgOpen1: TOpenDialog;

mniExportPNG: TMenuItem;

procedure getCharParams(var Chrwidth, Chrheight:Integer);

procedure FormCreate(Sender: TObject);

procedure ScrollBox1MouseWheelDown(Sender: TObject; Shift: TShiftState;

MousePos: TPoint; var Handled: Boolean);

procedure ScrollBox1MouseWheelUp(Sender: TObject; Shift: TShiftState;

MousePos: TPoint; var Handled: Boolean);

procedure pb1Paint(Sender: TObject);

procedure btntaskClick(Sender: TObject);

procedure btnIfClick(Sender: TObject);

procedure pb1MouseDown(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

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

procedure btnWhileClick(Sender: TObject);

procedure ToolButton1Click(Sender: TObject);

procedure inptext(pb1:TPaintBox; pinf:TFigureInfo);

procedure mniExportBmpClick(Sender: TObject);

procedure mniExportPNGClick(Sender: TObject);

private

public

end;

var

increment:Integer;

k:Integer;

IfState: TIfstates;

MaxInlineChars:integer = 30;

Maxlines:integer = 5;

FieldLeft:Integer = 0;

FieldTop:Integer = 0;

FieldWidth:Integer = 750;

FieldHeight:Integer = 1500;

ButtonWidth:Integer;

RectMinWidth:Integer = 150;

RectMinHeight:Integer = 100;

offset:Integer = 75;

startX:Integer = 50;

startY:Integer = 20;

Radius:Integer = 30;

arrowk:Integer = 12;

arrAngel: real = Pi/6;

IsRight,Selected,isUp:Boolean;

Kek: TKek;

Rect: TRect;

FigureHead:PFigList;

maxX, maxY:Integer;

ClickFigure: TFigureInfo;

lol:PFigList;

implementation

procedure clrscreen(pb1:TPaintBox);

begin

pb1.Canvas.Brush.Color:=clWhite;

pb1.Canvas.Rectangle(0,0,pb1.width,pb1.Height);

end;

procedure TKek.FormCreate(Sender: TObject);

var INIT:TFigureInfo;

begin

pb1.canvas.Pen.Width:=3;

createFigureHead(FigureHead);

ClickFigure:=FigureHead.Info;

clrscreen(pb1);

Selected:=False;

IfState:=RUP;

end;

procedure TKek.getCharParams(var Chrwidth, Chrheight:Integer);

var DC: HDC;

SaveFont: HFONT;

TTM: TTextMetric;

begin

DC := GetDC(mmoInput.Handle);

if (DC <> 0) then

begin

SaveFont := SelectObject(DC,mmoInput.Font.Handle);

if (GetTextMetrics(DC,TTM)) then

begin

ChrWidth:=TTM.tmAveCharWidth+TTM.tmExternalLeading;

Chrheight:=TTM.tmHeight;

end;

SelectObject(DC,SaveFont);

ReleaseDC(mmoinput.handle,DC);

end;

end;

procedure TKek.inptext(pb1:TPaintBox; pinf:TFigureInfo);

var p:PFigList;

var lvlwidth:integer;

ChrWidth:integer;

var i,maxStrLength:Integer;

Chrheight,CurrStrWidth:Integer;

begin

getCharParams(ChrWidth,Chrheight);

p:=GetAdr(figurehead,pinf);

mmoInput.Left:=pb1.left;

mmoInput.Top:=pb1.Top;

mmoInput.Width:=MaxInlineChars\* ChrWidth;

mmoInput.Height:=RectMinWidth;

mmoInput.Visible:=True;

mmoInput.Lines.Clear;

mmoInput.SelStart:=0;

mmoInput.Lines.Add(p.Info.Txt);

mmoInput.MaxLength:=30;

end;

procedure TKek.mniExportBmpClick(Sender: TObject);

var

kek, lol: integer;

begin

if dlgOpen1.Execute then

begin

ClickFigure := GetClickFig(0,0,FigureHead,Selected);

with TBitmap.Create do

begin

Width := pb1.Width;

Height := pb1.Height;

defaultDraw(FigureHead, canvas);

DrawBlocks(Canvas,FigureHead,kek,lol, IfState);

SaveToFile(dlgOpen1.FileName);

free;

end;

end;

end;

procedure TKek.mniExportPNGClick(Sender: TObject);

var

Bitmap: TBitmap;

png: TPNGImage;

kek,lol:Integer;

Rect: TRect;

begin

if dlgOpen1.Execute then

begin

try

ClickFigure := GetClickFig(0,0,FigureHead,Selected);

Bitmap := TBitmap.Create;

png := TPngImage.Create;

with bitmap do

begin

png := TPngImage.Create;

Width := pb1.Width;

Height := pb1.Height;

defaultDraw(FigureHead, canvas);

DrawBlocks(Canvas,FigureHead,kek,lol, IfState);

end;

png.Assign(bitmap);

rect.Left := 0;

rect.Top :=0;

rect.Right := Bitmap.Width;

Rect.Bottom := Bitmap.Height;

png.Draw(Bitmap.Canvas, rect);

png.SaveToFile(dlgOpen1.FileName);

finally

bitmap.Free;

png.Free;

end;

end;

end;

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

Shift: TShiftState);

var ClickAdr:PFigList;

begin

if Selected then

begin

ClickAdr:=GetAdr(FigureHead,ClickFigure);

if Key = VK\_DELETE then

try

DeleteBlock(Figurehead, Clickfigure);

finally

end;

end;

if key = 38 then

begin

if Selected then

begin

if ClickFigure.FigType = IfFig then

begin

if ord(ifstate)>0 then

IfState:=pred(Ifstate)

else

begin

IfState:=Succ(ifstate);

IfState:=Succ(ifstate);

end;

pb1.Repaint;

exit

end;

end;

try

ClickAdr:=GetParentAdr(figurehead, ClickAdr);

ClickFigure:=Clickadr.Info;

except

end;

end;

end;

if selected and (key = 13) then

begin

Inptext(pb1,ClickFigure);

end;

if key = VK\_DOWN then

begin

IsRight:=False;

if Selected then

begin

if ClickFigure.FigType = IfFig then

begin

if ord(ifstate)<2 then

IfState:=succ(Ifstate)

else

begin

IfState:=pred(Ifstate);

IfState:=pred(Ifstate);

end;

pb1.Repaint;

Exit;

end;

end;

end;

if Key = 39 then

IsRight:= True;

pb1.Repaint;

end;

procedure TKek.pb1MouseDown(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

var x0,y0:Integer;

var lol:PFigList;

var sel,ifst,maxStr:string;

var prex:TFigureInfo;

begin

if Selected then

begin

prex:=GetClickFig(x,y,FigureHead,Selected);

if (ClickFigure.x = prex.x) and (ClickFigure.y = prex.y) then

begin

Exit;

end;

end;

if (mmoInput.Visible) then

begin

GetAdr(FigureHead,ClickFigure).Info.Txt:=mmoInput.Text;

mmoInput.Visible:=False;

end;

case IfState of

RUP: ifst:='RUP';

RDOWN: ifst:='Rdown';

Down: ifst:='down';

end;

ClickFigure:=GetClickFig(x,y,FigureHead,Selected);

if selected then

sel:= 'selected' else sel:='Not selected';

lbl1.Caption:=

'width: '+ IntToStr(ClickFigure.width) +#10#13

+'height: '+ IntToStr(ClickFigure.height) +#10#13

+'x: '+ IntToStr(ClickFigure.x) +#10#13

+'y: '+ IntToStr(ClickFigure.y) +#10#13

+'level: '+ IntToStr(ClickFigure.level) +#10#13

+'pb1width: '+ IntToStr(Kek.pb1.width) +#10#13

+'Selected: '+ sel +#10#13

+'Ifstate: '+ ifst +#10#13

+'row: '+ IntToStr(ClickFigure.Row) +#10#13

+'maxX: ' + IntToStr(maxX) + #10#13

+'Memo1.CaretPos.y: ' + IntToStr(mmoInput.CaretPos.y) + #10#13 ;

pb1.Repaint;

end;

procedure ButtonReaction(Figure: TFigType);

var p:TFigureInfo;

var prex:PFigList;

tempP:TFigureInfo;

var dRow:integer;

begin

if Selected then

begin

prex:=GetAdr(FigureHead,ClickFigure);

p.height:=rectMinHeight;

p.FigType:=Figure;

p.Txt:='';

if (ClickFigure.FigType = IfFig) and ((IfState = RUP) or (IfState = RDOWN)) then

begin

case IfState of

RUP:

begin

p.x:=prex.Info.x +prex.Info.width+ offset;

p.y:=prex.Info.y;

p.Row:=prex.Info.row;

p.width:=RectMinWidth;

p.level:=prex.Info.level+1;

CreateNode(FigureHead,p,prex.Info);

end;

RDOWN:

begin

p.x:=ClickFigure.x +ClickFigure.width+ offset;

p.y:=ClickFigure.y+ClickFigure.Height + offset;

p.Row:=ClickFigure.row+1;

p.width:=RectMinWidth;

p.level:=ClickFigure.level+1;

if p.level>1 then

SetAdjustment(FigureHead,p.level,p.Row);

CreateLeft(FigureHead,p,ClickFigure);

end;

end;

Kek.pb1.repaint;

end

else

if (ClickFigure.FigType = WhileFig) and IsRight then

begin

p.x:=ClickFigure.x +ClickFigure.width+ offset;

p.y:=ClickFigure.y+ClickFigure.Height + offset;

p.Row:=ClickFigure.row+1;

p.width:=RectMinWidth;

p.level:=ClickFigure.level+1;

if p.level>1 then

SetAdjustment(FigureHead,p.level,p.Row);

CreateLeft(FigureHead,p,ClickFigure);

end else

if (ClickFigure.FigType = RepeatFig) and IsRight then

begin

p.x:=prex.Info.x +prex.Info.width+ offset;

p.y:=prex.Info.y;

p.Row:=prex.Info.row;

p.width:=RectMinWidth;

p.level:=prex.Info.level+1;

CreateNode(FigureHead,p,prex.Info);

end

else

if IsRight and (prex.R = nil) then

begin

p.x:=ClickFigure.x+ClickFigure.width+offset;

p.y:=ClickFigure.y+half(ClickFigure.Height)-half(RectMinHeight);

p.Row:=ClickFigure.row;

p.width:=RectMinWidth;

p.level:=ClickFigure.level+1;

CreateNode(FigureHead,p,ClickFigure);

end

else if (prex.adr = nil) then

begin

p.x:=ClickFigure.x;

p.width:=ClickFigure.width;

p.level:=ClickFigure.level;

p.y:=GetAdjust(FigureHead,p.level,ClickFigure,dRow);

p.Row:= GetRow(FigureHead,p.level,p);

if p.level>1 then

SetAdjustment(FigureHead,p.level,p.Row);

insertFigure(FigureHead,p,ClickFigure);

end;

Kek.pb1.Repaint;

end;

end;

procedure TKek.pb1Paint(Sender: TObject);

var temp:PFigList;

begin

clrscreen(pb1);

if maxY>(Kek.pb1.Height- 400) then

begin

Kek.ScrollBox1.Height:=Kek.ScrollBox1.Height+200;

Kek.pb1.Height:=Kek.pb1.Height+200;

end;

if maxX>(Kek.pb1.Width-600) then

begin

Kek.ScrollBox1.width:=Kek.ScrollBox1.width+500;

Kek.pb1.width:=Kek.pb1.width+500;

end;

DrawBlocks(pb1.Canvas,FigureHead,maxX,maxY,IfState);

if Selected then

begin

DrawDirectArrows(pb1.Canvas,ClickFigure,IsRight,ifState );

drawRect(pb1.Canvas,ClickFigure,clBlue);

end;

defaultDraw(FigureHead,pb1.Canvas);

end;

procedure TKek.btntaskClick(Sender: TObject);

begin

ButtonReaction(TaskFig);

end;

procedure TKek.btnWhileClick(Sender: TObject);

begin

ButtonReaction(WhileFig);

end;

procedure TKek.btnIfClick(Sender: TObject);

begin

ButtonReaction(IfFig);

end;

procedure TKek.ScrollBox1MouseWheelDown(Sender: TObject; Shift: TShiftState;

MousePos: TPoint; var Handled: Boolean);

begin

if ssShift

in Shift then

begin

scrollbox1.HorzScrollBar.Position:=scrollbox1.HorzScrollBar.position + scrollbox1.VertScrollBar.Increment;

end

else

begin

scrollbox1.VertScrollBar.Position:=scrollbox1.VertScrollBar.position + scrollbox1.VertScrollBar.Increment;

end;

end;

procedure TKek.ScrollBox1MouseWheelUp(Sender: TObject; Shift: TShiftState;

MousePos: TPoint; var Handled: Boolean);

begin

if ssShift in Shift then

begin

scrollbox1.HorzScrollBar.Position:=scrollbox1.HorzScrollBar.position - scrollbox1.VertScrollBar.Increment;

end

else

begin

scrollbox1.VertScrollBar.Position:=scrollbox1.VertScrollBar.position - scrollbox1.VertScrollBar.Increment;

end;

end;

procedure TKek.ToolButton1Click(Sender: TObject);

begin

ButtonReaction(RepeatFig);

end;

end.

**unit Lists;**

interface

uses System.SysUtils,Vcl.Graphics,Vcl.ExtCtrls,vcl.Dialogs,Types\_const;

function aHead(head: PFigList;lol:TFigureInfo):boolean;

procedure DeleteBlock(head: PFigList; P:TFigureInfo);

function Levelwidth(head:PFigList; lvl:integer):Integer;

procedure SetLevelWidth(var head:PFigList; lvl,width,increment:Integer);

procedure CreateNode(head:PFigList; points:TFigureInfo; LRec:TFigureInfo);

function GetClickFig(x,y:integer; const head:PFigList; var selected:Boolean):TFigureInfo;

procedure createFigureHead(var head:PFigList);

procedure insertFigure(

function GetAdr(const head:PFigList; info:TFigureInfo):PFigList;

function GetParentAdr(const head:PFigList; P:PFigList):PFigList;

procedure createleft(head: PFigList;Points:TFigureInfo; LRec:TFigureInfo);

procedure SetAdjustment(const head:PFigList; lvl,row:Integer);

function LeftBorn(const head:PFigList;p:TFigureInfo):boolean;

function GetAdjust(head:PFigList;lvl:integer;p:TFigureInfo;var d:Integer):integer;

function GetRow(head:PFigList;lvl:integer;p:TFigureInfo):integer;

implementation

procedure createFigureHead(var head:PFigList);

begin

new(head);

head.Adr:=nil;

head.Info.x:=startX-radius;

head.Info.y:=startY;

head.Info.width:=RectMinWidth;

head.Info.Height:=2\*radius;

head.Info.level:=1;

head.R:=nil;

head.Info.FigType:=TaskFig;

head.Info.Row:=0;

end;

function GetAdjust(head:PFigList;lvl:integer;p:TFigureInfo;var d:Integer):integer;

var max:Integer;

function GetAdj(head:PFigList;lvl:integer; var max,d:integer):integer;

var temp,temphead:Pfiglist;

begin

temp:=head;

while (temp<>nil) do

begin

if (temp.Info.level>lvl) and (temp.Info.y>max )then

begin

max:=temp.Info.y+temp.Info.Height;

d:=temp.info.row;

end;

if temp.R<>nil then

begin

result:=GetAdj(temp.R,lvl,max,d);

end;

if temp.L<>nil then

begin

result:=GetAdj(temp.L,lvl,max,d);

end;

temp:=temp^.Adr;

end;

result:=max;

end;

begin

max:=p.y+p.Height;

result:=GetAdj(head,lvl,max,d)+offset;

end;

function eqInfo(a,b:TFigureInfo):Boolean;

begin

result:= False;

if (a.x = b.x) and (a.y = b.y) then

result:=True;

end;

function GetRow(head:PFigList;lvl:integer;p:TFigureInfo):integer;

var max:Integer;

function GetR(head:PFigList;lvl:integer; var max:Integer):integer;

var temp,temphead:Pfiglist;

begin

temp:=head;

while (temp<>nil) do

begin

begin

if (temp.Info.level>=lvl) and (temp.Info.Row>max) and not(eqInfo(p,temp.Info))then

begin

max:=temp.Info.Row

end;

if temp.R<>nil then

begin

result:=GetR(temp.R,lvl,max);

end;

if temp.L<>nil then

begin

result:=GetR(temp.L,lvl,max);

end;

temp:=temp^.Adr;

end;

end;

result:=max;

end;

begin

max:=0;

result:=GetR(head,lvl,max)+1;

end;

procedure JustAdjust( head:PFigList);

var temp,temphead:PFigList;

begin

temp:=head;

while (temp<>nil) do

begin

temp.info.y:=temp.info.y + temp.Info.Height + offset;

temp.Info.Row:=temp.Info.Row + 1;

temp:=temp^.Adr;

end;

end;

procedure SetAdjustment(const head:PFigList; lvl,row:Integer);

var temp,temphead:PFigList;

begin

temp:=head;

while (temp<>nil) do

begin

if

begin

temp.info.y:=temp.info.y + temp.Info.Height + offset;

temp.Info.Row:=temp.Info.Row + 1;

end;

if temp.R<>nil then

begin

SetAdjustment(temp.R,lvl,row);

end;

if temp.L<>nil then

begin

SetAdjustment(temp.L,lvl,row);

end;

temp:=temp^.Adr;

end;

end;

function GetParentAdr(const head:PFigList; P:PFigList):PFigList;

function GetP(const head:PFigList; P:PFiglist):PFigList;

var temp:PFigList;

begin

temp:=head;

while temp<> nil do

begin

if (temp^.Adr = P) then

begin

result:=temp;

exit;

end;

if temp.R<>nil then

begin

Result:=GetP(temp.R,P);

end;

if temp.L<>nil then

begin

Result:=GetP(temp.L,P);

end;

temp:=temp^.Adr;

end;

end;

begin

result:=nil;

result:=GetP(head,p);

end;

function GetAdr(const head:PFigList; info:TFigureInfo):PFigList;

var temp:PFigList;

begin

temp:=head;

while temp<> nil do

begin

if (temp^.Info.x = info.x) and (temp^.Info.y = info.y) then

begin

result:=temp;

exit;

end;

if temp.R<>nil then

begin

Result:=Getadr(temp.R,info);

end;

if temp.L<>nil then

begin

Result:=Getadr(temp.L,info);

end;

temp:=temp^.Adr;

end;

end;

function Levelwidth(head:PFigList; lvl:integer):Integer;

var temp,temphead:PFigList;

begin

with TBitmap.Create do

begin

temp:=head;

if (head.Info.level = lvl) and (Canvas.TextWidth(head.Info.Txt)>Result) then

Result:=Canvas.TextWidth(head.Info.Txt);

while (temp^.Adr<>nil) do

begin

if (temp.Info.level = lvl) and (Canvas.TextWidth(temp.Info.Txt)>Result) then

Result:=Canvas.TextWidth(temp.Info.Txt);

if temp.R<>nil then

begin

temphead:=temp.R;

if (temphead.Info.level = lvl) and (Canvas.TextWidth(temphead.Info.Txt)>Result) then

Result:=Canvas.TextWidth(temphead.Info.Txt);

Result:=Levelwidth(temphead,lvl);

end;

if temp.L<>nil then

begin

temphead:=temp.L;

if (temphead.Info.level = lvl) and (Canvas.TextWidth(temphead.Info.Txt)>Result) then

Result:=Canvas.TextWidth(temphead.Info.Txt);

Result:=Levelwidth(temphead,lvl);

end;

temp:=temp^.Adr;

end;

if (Result = 0) and (temp.Info.level = lvl) then

Free;

end;

end;

procedure insertFigure(head:PFigList; points:TFigureInfo;LRec:TFigureInfo);

var LRecAdr,temp,temp2:PFigList;

begin

LRecAdr:=getadr(head,LRec);

if LRecadr.Adr = nil then

begin

new(LRecAdr^.Adr);

temp:=LRecAdr^.Adr;

temp^.Adr:=nil;

temp^.L:=nil;

temp^.R:=nil;

temp^.Info:=points;

temp^.Info.Txt:='';

end

else

end; }

end;

function GetClickFig(x,y:integer; const head:PFigList; var selected:Boolean):TFigureInfo;

function GClickFig(x,y:integer; const head:PFigList; var selected:Boolean):TFigureInfo;

var temp:PfigList;

begin

selected:=False;

temp:=head;

if (x > temp.Info.x) and

(x < temp.Info.x+temp.Info.width) and

(y > temp.Info.y) and

(y < temp.Info.y + temp.Info.Height)

then

begin

Result:=temp.Info;

selected:=True;

Exit;

end;

while (temp<> nil) do

begin

if temp.R<>nil then

begin

Result:=GClickFig(x,y,temp.R,selected);

if selected then exit;

end;

if temp.L<>nil then

begin

Result:=GClickFig(x,y,temp.L,selected);

if selected then exit;

end;

if (x > temp.Info.x) and

(x < temp.Info.x+temp.Info.width) and

(y > temp.Info.y) and

(y < temp.Info.y + temp.Info.Height)

then

begin

Result:=temp.Info;

selected:=True;

Exit;

end

else temp:=temp^.Adr;

end;

if selected then

Exit;

Result.x:=-1;

end;

begin

Result:=GClickFig(x,y,head,selected);

end;

function aHead(head: PFigList;lol:TFigureInfo):boolean;

begin

result:= false;

if (head.Info.x = lol.x) and (head.info.y = lol.y) then

result:=true;

end;

procedure SetLevelWidth(var head:PFigList; lvl,width,increment:Integer);

var temp,temphead:PFigList;

var isRight:boolean;

var max:integer;

begin

temp:=head;

while (temp<>nil) do

begin

if temp.info.level=lvl then

temp.info.width:=width;

if head.Info.level > lvl then

head.info.x:=head.info.x+increment;

if temp.R<>nil then

begin

temphead:=temp.R;

if temphead.info.level=lvl then

temphead.info.width:=width;

if head.Info.level > lvl then

head.info.x:=head.info.x+increment;

SetLevelWidth(temphead,lvl,width,increment);

end;

temp:=temp^.Adr;

end;

end;

procedure DeleteBlock(head: PFigList; P:TFigureInfo);

var temp: PFigList;

begin

temp:=GetParentAdr(head,GetAdr(head,P));

temp.adr:=nil;

end;

procedure CreateNode(head: PFigList;Points:TFigureInfo; LRec:TFigureInfo);

var LRecAdr,temphead,temp:PFigList;

begin

LRecAdr:=getadr(head,LRec);

begin

new(LRecAdr^.R);

temp:=LRecAdr^.R;

temp^.L:=nil;

temp^.Adr:=nil;

temp^.R:=nil;

temp^.Info:=points;

temp^.Info.Txt:='';

end;

end;

function LeftBorn(const head:PFigList;p:TFigureInfo):boolean;

var exitbool:boolean;

function LeftB(const head:PFigList;p:TFigureInfo;var ex:Boolean):boolean;

var temp,temp2:PFigList;

begin

if not(ex) then

begin

temp2:=GetAdr(head,p);

temp:=head;

while temp<> nil do

begin

if temp.R<>nil then

begin

Result:=LeftB(temp.R,p,ex);

end;

if temp.L<>nil then

begin

Result:=LeftB(temp.L,p,ex);

end;

if (temp^.r = temp2) then

begin

result:=True;

ex:=True;

Exit;

end;

temp:=temp^.Adr;

end;

end;

end;

begin

exitbool:=False;

LeftB(head,p,exitbool);

end;

procedure createleft(head: PFigList;Points:TFigureInfo; LRec:TFigureInfo);

var LRecAdr,temphead,temp:PFigList;

begin

LRecAdr:=getadr(head,LRec);

begin

new(LRecAdr^.L);

temp:=LRecAdr^.L;

temp^.L:=nil;

temp^.Adr:=nil;

temp^.R:=nil;

temp^.Info:=points;

temp^.Info.Txt:='';

end;

end;

end.

**unit Types\_const;**

interface

uses System.classes;

const strBegin = 'BEGIN';

FieldLeft:Integer = 0;

FieldTop:Integer = 0;

FieldWidth:Integer = 750;

FieldHeight:Integer = 1500;

RectMinWidth:Integer = 150;

RectMinHeight:Integer = 100;

offset:Integer = 75;

startX:Integer = 50;

startY:Integer = 20;

Radius:Integer = 30;

arrowk:Integer = 12;

arrAngel: real = Pi/6;

type

TIfStates = (RUP, RDOWN, DOWN);

TFigType = (TaskFig,IfFig,WhileFig,StartFig,untilFig, RepeatFig);

TDirection = (Horizontal, Vertical);

TArrowInfo = record

id,x,y,length: integer;

Direction: TDirection;

ArrowType:TFigType;

end;

PArrowList = ^TArrowList;

TArrowList = record

Info: TArrowInfo;

Adr:PArrowList;

end;

TFigureInfo = record

Txt: string ;

x,width,y,Height,level,Row:integer;

FigType: TFigType;

end;

PFigList = ^FigList;

FigList = record

Info: TFigureInfo;

Adr: PFigList;

L: PFigList;

R: PFiglist;

end;

function third(x:integer):integer;

function half(x:integer):integer;

function forth(x:integer):integer;

implementation

function half(x:integer):integer;

begin

result:=Round(x/2)

end;

function third(x:integer):integer;

begin

result:=Round(x/3)

end;

function forth(x:integer):integer;

begin

result:=Round(x/4)

end;

end.

**unit DrawItems;**

interface

uses System.SysUtils,System.Types, Types\_const,Vcl.Graphics,Lists,Vcl.ExtCtrls,vcl.Dialogs;

procedure DrawBlocks(canv:TCanvas; head:PFigList; var maxX,maxY:integer; ifst:TIfStates);

procedure defaultDraw(head:PFigList;canv:TCanvas);

procedure DrawDirectArrows(canv:TCanvas; p:TFigureInfo; left:boolean; ifState:TIfStates);

procedure drawRect(canv:TCanvas;p:TFigureInfo;Color:Tcolor);

procedure InsertTXT(canv:TCanvas;var p:TFigureInfo);

function IsEmptyTXT(p:TFigureInfo):Boolean;

procedure DrawArrow(canv:TCanvas; inp,outp:TFigureInfo;isRight:Boolean;ifstate:TIfStates);

implementation

var

FieldLeft:Integer = 0;

FieldTop:Integer = 0;

FieldWidth:Integer = 750;

FieldHeight:Integer = 1500;

ButtonWidth:Integer;

RectMinWidth:Integer = 75;

RectMinHeight:Integer = 100;

ratio:real=1.5;

offset:Integer = 75;

startX:Integer = 50;

startY:Integer = 20;

Radius:Integer = 30;

arrowk:Integer = 12;

arrAngel: real = Pi/6;

function TextUtil(cap:string):string;

var lengthkek,i:Integer;

begin

lengthkek:=Length(cap);

i:=1;

while i<Lengthkek do

begin

if ( cap[i]=':') and (cap[i+1]= '=') then

begin

Insert('#10#13',cap,i+2);

lengthkek:=Length(cap);

end;

result:=cap;

inc(i);

end;

end;

procedure InsertTXT(canv:TCanvas;var p:TFigureInfo);

var

txt:TLabeledEdit;

TX,TY:Integer;

TextW,TextH: Integer;

cap:String;

Rectan:Trect;

border:Integer;

oldStyle:TBrushStyle;

begin

cap:=(p.Txt);

oldStyle:=Canv.Brush.Style;

border:=2;

TextW:=Canv.TextWidth(cap);

TextH:=Canv.TextHeight(cap);

case p.FigType of

TaskFig:

Rectan:=

Rect(p.x+border,p.y+border,p.x + p.width-2\*border,p.y + p.Height-2\*border);

IfFig:

Rectan:=

Rect(p.x+ round(p.width\*(TextH/p.Height)),p.y+half(p.height-textH),p.x + p.width-round(p.width\*(TextH/p.Height)),p.y+half(p.height-textH)+TextH);

WhileFig: Rectan:=

Rect(p.x+ round(p.width\*(TextH/p.Height)),p.y+half(p.height-textH),p.x + p.width-round(p.width\*(TextH/p.Height)),p.y+half(p.height-textH)+TextH);

StartFig: ;

untilFig: ;

end;

TX:=half(p.width)-half(TextW)+p.x;

TY:=half(p.Height)-half(TextH)+p.y;

Canv.TextRect(Rectan,cap,[tfVerticalCenter ,tfNoPrefix,tfWordBreak]);

Canv.Brush.Style:=oldStyle;

end;

function IsEmptyTXT(p:TFigureInfo):Boolean;

begin

Result:=False;

if p.Txt= '' then

result:=True;

end;

procedure HorizontalAdjust(head: PFigList; NewSize:PFigList);

var temp:PFigList;

begin

end;

procedure drawRect(canv:TCanvas;p:TFigureInfo;Color:Tcolor);

var prev:TColor;

begin

if p.FigType <> IfFig then

begin

prev:=Canv.Pen.Color;

Canv.Pen.Color:=color;

with Canv do

begin

MoveTo(p.x,p.y);

LineTo(p.x,p.y + p.Height);

LineTo(p.x +p.width,p.y + p.Height);

LineTo(p.x +p.width,p.y);

LineTo(p.x,p.y);

end;

Canv.Pen.Color:=prev;

InsertTXT(canv,p);

end;

end;

procedure drawA(canv:TCanvas;p:TArrowInfo; Color:TColor);

var x,y:Integer;

var prevColor:TColor;

begin

prevColor:=Canv.Pen.Color;

Canv.Pen.Color:=Color;

x:=Round(arrowk\*sin(arrAngel));

y:=Round(arrowk\*cos(arrAngel));

with Canv do

begin

MoveTo(p.x,p.y);

case p.Direction of

Vertical:

begin

LineTo(p.x,p.y+p.length);

LineTo(p.x+x,p.y+p.length-y);

MoveTo(p.x,p.y+p.length);

LineTo(p.x-x,p.y+p.length-y);

end;

Horizontal:

begin

LineTo(p.x+p.length,p.y);

LineTo(p.x+p.length-y,p.y+x);

MoveTo(p.x+p.length,p.y);

LineTo(p.x+p.length-y,p.y-x)

end;

end;

Canv.Pen.Color:=prevColor;

end;

end;

procedure drawEllipse(canv:TCanvas;x,y,width,height:Integer);

var R:Integer;

begin

R:=half(height);

with Canv do

begin

Arc(x,y,x+height,y+height,x+R,y,x+R,y+height);

Arc(x+width-height,y,x+width,y+height,x+width-R,y+height,x+width-R,y);

MoveTo(x+r,y);

LineTo(x+width-r,y);

MoveTo(x+r,y+height);

LineTo(x+width-r,y+height);

end;

end;

procedure defaultDraw(head:PFigList;canv:TCanvas);

var p:TFigureInfo;

x,y:integer;

TX,TY:Integer;

TextW,TextH: Integer;

begin

canv.Pen.Width:=3;

with Canv do

begin

drawEllipse(canv,head.Info.x ,head.Info.y,Head.info.width,Head.Info.Height);

p.Txt:=strBegin;

TextW:=Canv.TextWidth(strBegin);

TextH:=Canv.TextHeight(strBegin);

TX:=head.Info.x+half(Head.info.width)-half(TextW);

TY:=head.Info.y+half(Head.Info.Height)-half(TextH);

Canv.Brush.Style:=bsClear;

TextOut(TX,TY,strBegin);

Canv.Brush.Style:=bsSolid;

end;

end;

procedure DrawIF(canv:TCanvas; p:TFigureInfo; color:TColor);

var prev:TColor;

begin

prev:=Canv.Pen.Color;

Canv.Pen.Color:=color;

with Canv do

begin

MoveTo(p.x+half(p.width),p.y);

LineTo(p.x+p.width,p.y + half(p.Height));

LineTo(p.x +half(p.width),p.y + p.Height);

LineTo(p.x ,p.y+half(p.Height));

LineTo(p.x+half(p.width),p.y);

end;

InsertTXT(canv,p);

Canv.Pen.Color:=prev;

end;

procedure DrawWhile(canv:TCanvas; p:TFigureInfo; color:TColor);

var prev:TColor;

begin

prev:=Canv.Pen.Color;

Canv.Pen.Color:=color;

with Canv do

begin

MoveTo(p.x+forth(p.width),p.y);

LineTo(p.x+3\*forth(p.width),p.y);

LineTo(p.x+p.width,p.y + half(p.Height));

LineTo(p.x +3\*forth(p.width),p.y + p.Height);

LineTo(p.x +forth(p.width),p.y + p.Height);

LineTo(p.x ,p.y+half(p.Height));

LineTo(p.x+forth (p.width),p.y);

end;

InsertTXT(canv,p);

Canv.Pen.Color:=prev;

end;

procedure DrawFigure(canv:TCanvas; p:TFigureInfo; color:TColor);

begin

case p.FigType of

TaskFig:

begin

drawRect(canv,p,color);

end;

IfFig:

begin

DrawIF(canv,p,color);

end;

WhileFig:

begin

drawWhile(canv,p,color);

end;

RepeatFig:

begin

DrawWhile(canv,p,color);

end;

end;

end;

procedure DrawBlocks(canv:TCanvas; head:PFigList; var maxX,maxY:integer; ifst:TIfStates);

var temp,temphead:PFigList;

var isRight:boolean;

var inp,outp:PFigList;

var flag:Boolean;

begin

maxX:=0;

maxY:=0;

isRight:=False;

temp:=head;

if temp.Info.x > maxX then

maxX:=temp.Info.x;

if temp.R<>nil then

begin

isRight:=true;

temphead:=temp.R;

DrawArrow(canv,temp.r.info,temp.info,isRight,ifst);

DrawFigure(canv,temphead.info,clblack);

DrawBlocks(canv,temp.R,maxX,maxY,ifst);

isRight:=false;

if temp.Info.x > maxX then

maxX:=temp.Info.x;

end;

if temp.L<>nil then

begin

isRight:=true;

temphead:=temp.l;

DrawArrow(canv,temp.l.info,temp.info,isRight,ifst);

DrawFigure(canv,temphead.info,clblack);

DrawBlocks(canv,temp.l,maxX,maxY,ifst);

isRight:=false;

if temp.Info.x > maxX then

maxX:=temp.Info.x;

end;

while (temp^.Adr<>nil) do

begin

outp:=temp;

temp:=temp^.Adr;

if temp.Info.y > maxY then

maxY:=temp.Info.y;

if temp.Info.x > maxX then

maxX:=temp.Info.x;

DrawFigure(canv,temp.Info,clBlack);

inp:=temp;

DrawArrow(canv,inp.info,outp.info,isRight,ifst);

if temp.R<>nil then

begin

isRight:=true;

temphead:=temp.R;

DrawArrow(canv,temp.r.info,temp.info,isRight,ifst);

DrawFigure(canv,temphead.info,clblack);

DrawBlocks(canv,temp.R,maxX,maxY,ifst);

isRight:=false;

end;

if temp.L<>nil then

begin

isRight:=true;

temphead:=temp.l;

DrawArrow(canv,temp.l.info,temp.info,isRight,ifst);

DrawFigure(canv,temphead.info,clblack);

DrawBlocks(canv,temp.l,maxX,maxY,ifst);

isRight:=false;

end;

end;

end;

procedure DrawDirectArrows(canv:TCanvas; p:TFigureInfo; left:boolean; ifState:TIfStates);

var x,y:Integer;

var arrow:TArrowInfo;

var predC: TColor;

begin

predC:=Canv.Pen.Color;

Canv.Pen.Color:=clBlue;

case p.FigType of

TaskFig: ;

IfFig:

begin

case ifState of

RUP:

begin

with Canv do

begin

MoveTo((p.width+p.x-half(half(p.width))),p.y+half(half(p.Height)));

LineTo(p.x+p.width+2\*offset,p.y-half(offset));

LineTo(p.x+p.width+2\*offset+Rectminwidth+half(RectMinWidth),p.y-half(offset));

LineTo(p.x+p.width+2\*offset+Rectminwidth+half(RectMinWidth),p.y);

end;

end;

RDOWN:

begin

with Canv do

begin

MoveTo((p.width+p.x-half(half(p.width))),p.y+3\*forth(p.Height));

LineTo(p.x+p.width,PenPos.y+forth(p.height));

LineTo(penpos.x+RectMinWidth,penpos.y);

LineTo(PenPos.X ,penpos.y + offset);

end;

end;

DOWN:

begin

arrow.Direction:=Vertical;

arrow.x:=p.x+half(p.width);

arrow.y:=p.y+p.Height;

arrow.length:=offset;

drawA(canv,arrow,clBlue);

end

end;

end;

end; }

StartFig: ;

untilFig: ;

RepeatFig: ;

end;

if p.FigType = IfFig then

begin

end

else

begin

if left then

begin

arrow.Direction:=Horizontal;

arrow.x:=p.x+p.width;

arrow.y:=p.y+half(p.Height);

end

else

begin

arrow.Direction:=Vertical;

arrow.x:=p.x+half(p.width);

arrow.y:=p.y+p.Height;

end;

arrow.length:=offset;

drawA(canv,arrow,clBlue);

end;

Canv.Pen.Color:=predC;

end;

procedure DrawArrow(canv:TCanvas; inp,outp:TFigureInfo;isRight:Boolean;ifstate:TIfStates);

var x,y:Integer;

begin

if isRight then

begin

x:=inp.x;

y:=inp.y+half(inp.Height);

with Canv do

begin

case outp.FigType of

taskFig:

begin

MoveTo((outp.width+outp.x),outp.y+half(outp.Height));

LineTo(x,y);

end;

IfFig:

begin

if (outp.x < inp.x) and(outp.y = inp.y) then

begin

MoveTo((outp.width+outp.x-half(half(outp.width))),outp.y+half(half(outp.Height)));

LineTo(inp.x,inp.y-half(offset));

LineTo(inp.x+half(inp.width),inp.y-half(offset));

LineTo(inp.x+half(inp.width),inp.y);

end;

if (outp.x < inp.x) and(outp.y < inp.y) then

begin

MoveTo((outp.width+outp.x-half(half(outp.width))),outp.y+3\*forth(outp.Height));

LineTo(inp.x,inp.y-half(offset));

LineTo(inp.x+half(inp.width),inp.y-half(offset));

LineTo(inp.x+half(inp.width),inp.y);

end;

end;

WhileFig:

begin

MoveTo((outp.width+outp.x -half(forth(outp.width))),outp.y+3\*forth(outp.Height));

LineTo(inp.x,inp.y-half(offset));

LineTo(inp.x+half(inp.width),inp.y-half(offset));

LineTo(inp.x+half(inp.width),inp.y);

end;

RepeatFig:

begin

MoveTo((outp.width+outp.x-half(forth(outp.width))),outp.y+half(half(outp.Height)));

LineTo(inp.x,inp.y-half(offset));

LineTo(inp.x+half(inp.width),inp.y-half(offset));

LineTo(inp.x+half(inp.width),inp.y);

end;

end;

end;

end

else

begin

x:=half(inp.width)+inp.x;

y:=inp.y;

with Canv do

begin

case outp.FigType of

taskFig:

begin

MoveTo((half(outp.width)+outp.x),outp.y+outp.Height);

LineTo(x,y);

end;

IfFig:

Begin

MoveTo((half(outp.width)+outp.x),outp.y+outp.Height);

LineTo(x,y);

End;

WhileFig:

begin

MoveTo((half(outp.width)+outp.x),outp.y+outp.Height);

LineTo(x,y);

end;

RepeatFig:

begin

MoveTo((half(outp.width)+outp.x),outp.y+outp.Height);

LineTo(x,y);

end;

end;

end;

end;

end;

end.