**РАЗДЕЛ 3. ВИЗУАЛЬНЫЙ ЯЗЫК DRAKON**

Как показывает опыт преподавания информатики в средней школе, овладение искусством программирования и алгоритмизации наталкивается на серьезные трудности. В значительной степени они связаны с отсутствием навыков формулирования постановки задачи и разработки алгоритма ее решения. Еще в начале массового обучения программированию в школах и ВУЗах было замечено, что программистов условно можно разделить на две группы: «полевой офицер» и «штабной офицер». Полевой офицер, как правило, владея определенным запасом знаний программирования сразу же бросается писать программный код. Без комментариев и пояснений. Естественно, что в процессе программирования возникают непредвиденные ситуации, приводящие к постоянным изменениям в тексте программы. В конце концов и во многих случаях «полевой офицер» сдается или начинает подумывать о полезности побыть «штабным офицером». В свою очередь «штабной офицер» вначале обдумывает алгоритм программы. И здесь ему в помощь различные методы визуализации алгоритмов.

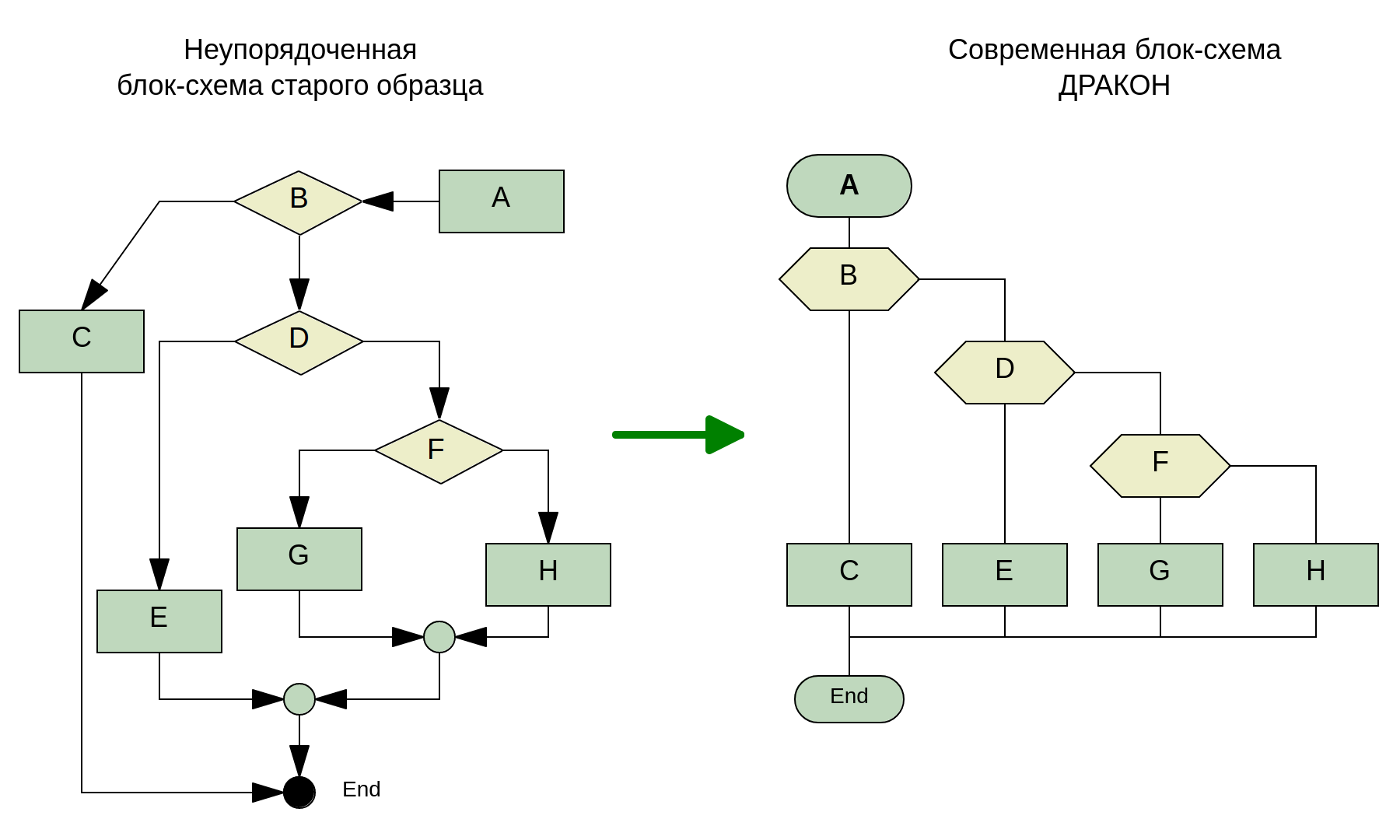
**3.1. Визуализация представления алгоритмов**

В теории и на практике сформировались следующие формы представления алгоритмов :

1. словесная (запись на естественном языке);
2. визуальная (изображения из графических символов);
3. псевдокоды (полуформализованные описания алгоритмов на условном алгоритмическом языке);
4. программная (тексты на языках программирования).

Наиболее эффективными и распространенными с точки зрения наглядности и понимания получили визуальные способы представления алгоритмов. В теоретическом отношении визуализация это форма представления информации, данных, знаний в виде изображений с целью достижения максимального удобства восприятия, понимания и анализа. Визуальное представление позволяет лучше видеть и понимать структурные элементы, используемые при проектировании алгоритма, включая логику их взаимодействия. Наиболее распространенными средствами визуализации алгоритмических конструкций являются блок-схемы и дракон-диаграммы. Преимуществом применения блок-схем является наличие соответствующего Государственного стандарта. Однако в последнее время блок-схемы подвергаются критике: утверждаются, что они непригодны для структурного программирования, с трудом поддаются формализации, их нельзя использовать для генерации программного кода.

Альтернативой визуализации алгоритмов с помощью блок-схем является технология конструирования алгоритмов на основе др*акон-технологии,* результатом которой являются дракон-диаграммы. Дракон-диаграммы пригодны для формализованной записи, автоматического получения кода и выполнения его на компьютере. Однако более важным аспектом применения дракон-технологии является ее когнитивное отличие от технологии блок-схем. Хотя блок-схемы время действительно улучшают ясность программ, однако это происходит не всегда, причем степень улучшения невелика. Кроме того, есть немало случаев, когда неудачно выполнены блок-схемы запутывают дело и затрудняют понимание. В отличие от них дракон-диаграммы гораздо более высоким уровнем понимания (рис.3.1.).

Рис. 3.1. Сравнение визуального представления алгоритма в виде блок-схемы и дракон-диаграммы

Благодаря использованию специальных формальных и неформальных когнитивных приемов дракон-диаграммы позволяют визуально представить решение любого сколь угодно сложного алгоритма или технологической проблемы в предельно ясной, наглядной и доходчивой форме. Особое значение визуализация алгоритмов с помощью дракон-диаграмм приобретает в учебном процессе как при изучении информатики, так и на уроках естественных, а также гуманитарных дисциплин.

**3.2. Редактор алгоритмического *языка* DRAKON Editor**

В настоящее время существует несколько платформ для построения дракон-диаграмм (DRAKON Editor, DRAKONHUB, Drakon.tech, «ИС Дракон». Генерация программного кода на одном из известных языков программирования, в том числе и на языке Golang, реализуется с помощью свободно распространяемого редактора DRAKON Editor WEB. В настоящее время актуальной версией редактора является DRAKON- Editor 1.31, которую можно загрузить с сайта <http://drakon-editor.sourceforge.net/> (рис. 3.2.).



Рис. 3.2. Архивированный файл редактора

Для запуска редактора DRAKON Editor 1.31. в операционной системе Linux and Mac необходимо:

- извлечь файлы из архива drakon editor 1.31.zip;

- запустить редактор: /drakon\_editor.tcl с терминала (внутри неархвированной папки.

Для операционной системы Windows необходимо:

- извлечь файлы из архива drakon editor 1.31.zip;

- открыть исполняемый файл редактора двойным кликом на фрагменте drakon\_tcl (рис. 3.3.):

.

Рис. 3.3. Запуск редактора DRAKON Editor 1.31.

После активизации редактора (DRAKON\_Editor) выбирается опция «Создать новый файл диаграммы», после чего открывается окно для сохранения файла диаграммы. Сразу же после создания нового файла его нужно обязательно сохранить, поскольку в редакторе DRAKON Editor нет необходимости сохранять все правки, - файлы сохраняются автоматически.

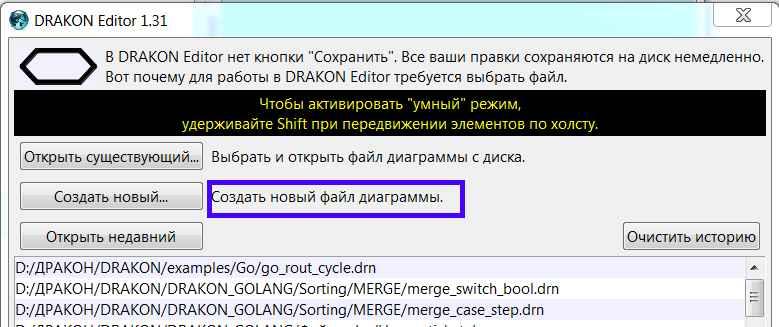
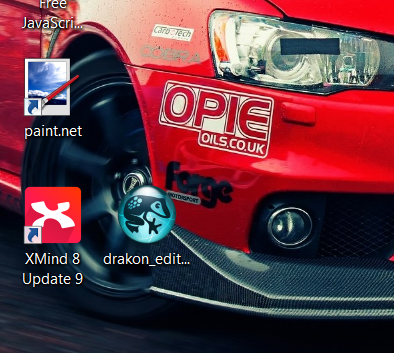


Рис. 3.4. Создание нового файла диаграммы

После создания нового файла открывается окно интерфейса редактора (Рис.3.5.):

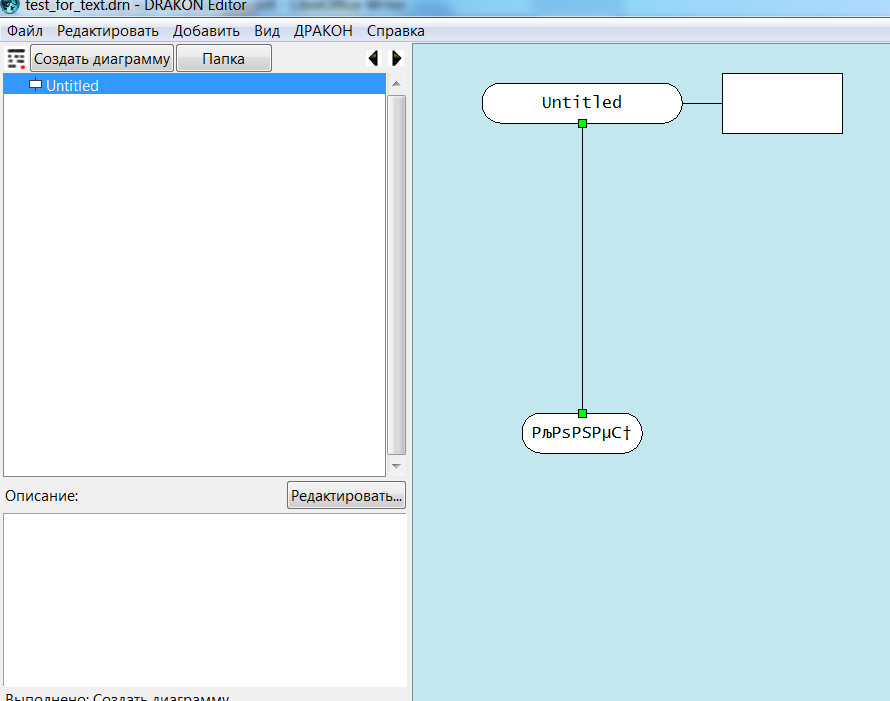


Рис. 3.5. Окно интерфейса редактора DRAKON WEB Editor

Далее нужно нажать иконку “New diagram” и во вновь открывшемся меню выбрать вид диаграммы (Примитив /Primitive/ или Силуэт /Silhuette/) (рис.3.6.).

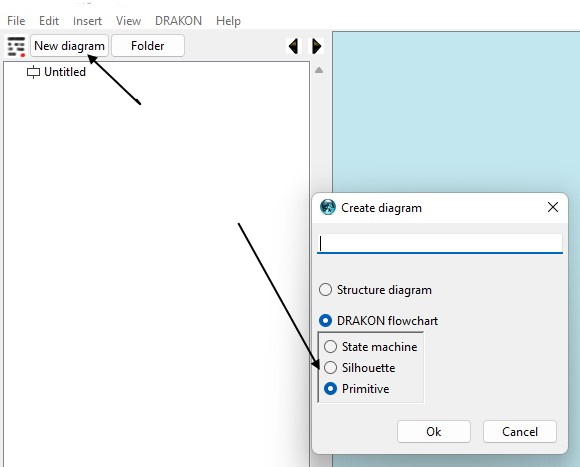
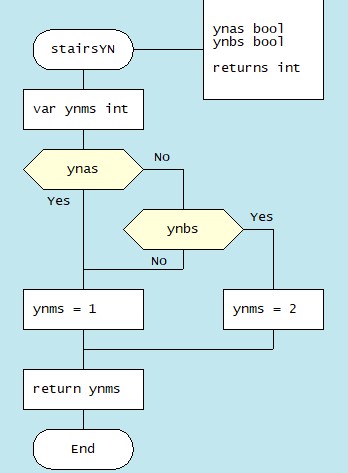
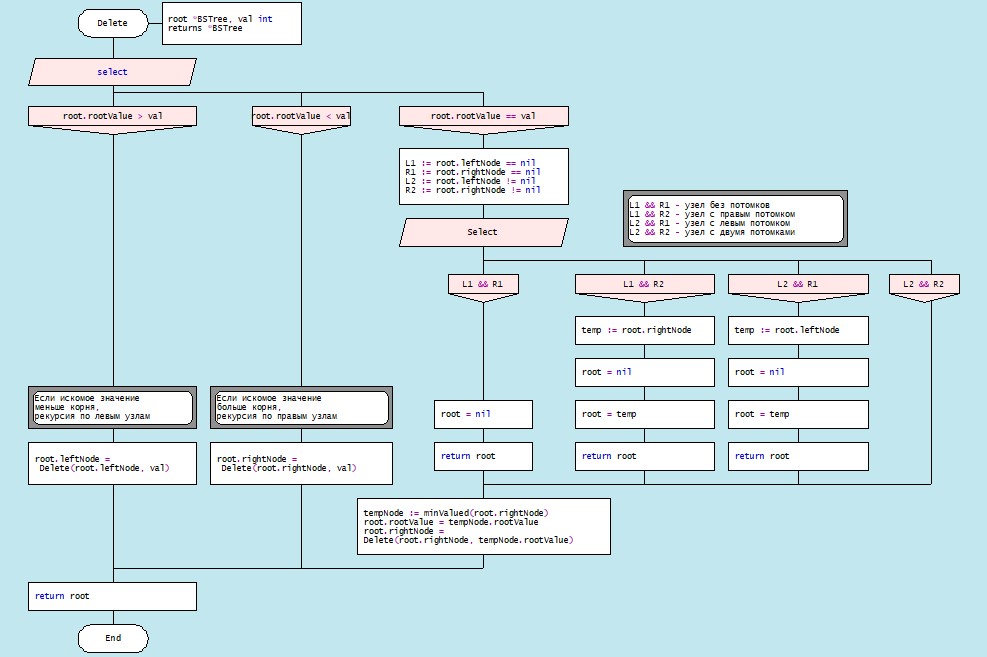


Рис.3.6. Создание новой диаграммы

Выбор вида диаграммы определяется сложностью алгоритма (рис.3.7.):



а) «Примитив»



б) «Силуэт»

Рис. 3.7. Виды диаграмм: а) «Притимив»; б) «Силуэт»

Для создания дракон-диаграммы пользователь вызывает контекстное меню (правая кнопка мыши), и в соответствии с алгоритмом выбирает необходимые графоэлементы, наполняя их соответствующими операторами выбранного языка программирования (Рис.3.6).

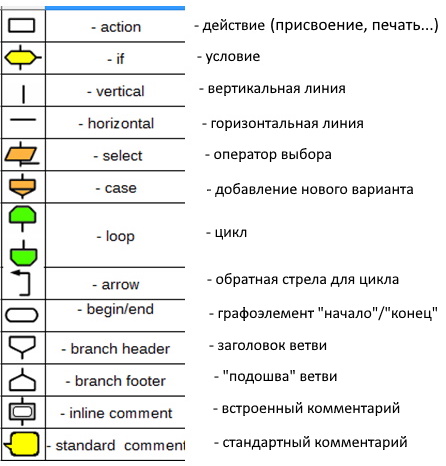


Рис. 3.8. Массив графоэлементов редактора DRAKON WEBrakon Web Editor

Выбранный графоэлемент перемещается мышью на вертикальную линию (шампур — по терминологии DRAKON-технологии) в нужное место. После этого графоэлемент заполняется соответствующим текстом, отображающим тот или иной фрагмент на конкретном языке программирования.

Создание дракон-диаграммы должно происходить по определенным правилам:

1. Создание дракон-диаграммы начинается с названия, которое должно отражать назначение алгоритма (функции) и располагаться на самом верху.

2. На диаграмме должны быть только одно начало и один конец. Графоэлемент «*Конец*» помещается внизу диаграммы (рис.3.8):

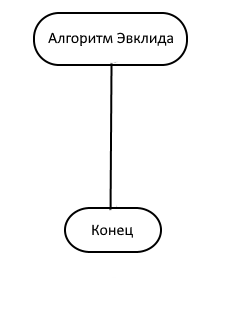


Рис. 3.8. Начало диаграммы

3. Поток действий представленный на диаграмме должен идти только сверху вниз. Такой подход является более удобным, поскольку в нашем культурном ареале тексты читаются именно таким образом.

4. Необходимо избегать поворотов. Единственный случай когда линии должны изменить направление, - это место принятия решений. Повороты нужны только тогда когда по алгоритму требуется осуществить выбор между различными действиями. Если же решений нет, нужно идти вниз. Во всяком случае необходимо минимизировать число поворотов.

5. **Категорически** не допускается пересечений линий. Все попытки применить пересечения должны пресекаться. Впрочем в случае пересечения редактор выдаст ошибку.

6. Выполнение действий сверху вниз позволяет не прибегать к использованию стрелок. Единственное исключение — цикл типа while (рис.3.8.):

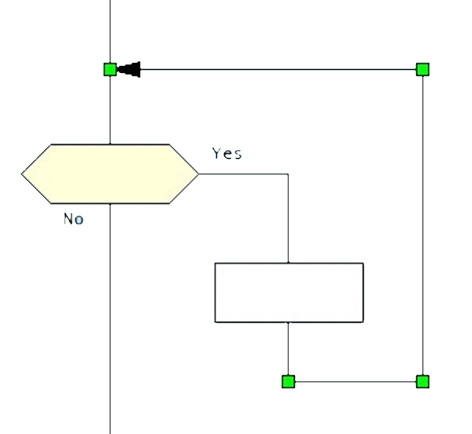


Рис. 3.9. Поток данных снизу-вверх (цикл типа While)

7. При создании дракон-диаграммы должны использоваться исключительно прямые вертикальные и горизонтальные линии, поскольку прямые линии понятней кривых.

8. Необходимо придерживаться одинакового расстояния между соседними графоэлементами. Редактор DRAKON WEB Editor позволяет «умно» оптимизировать диаграмму с помощью клавиш «Shift + мышь).

9. Разветвление осуществляется только направо. Разветвление налево должно быть исключено. Выполнение этого правила существенно повышает предсказуемость диаграмм и их однообразие.

У читателя может сложиться впечатление, что создание дракон-диаграмм является очень сложным процессом. Следует напомнить еще раз, что в редакторе DRAKON WEB Editor все происходит достаточно просто и понятно. Редактор не допустит нарушение изложенных правил, что проверяется опцией *verify (на рисунке указана ошибка — нет пространства между графоэлементами* (рис.3.10)):

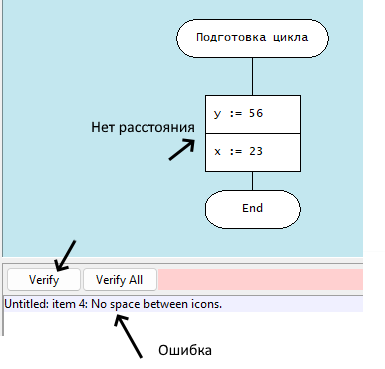


Рис. 3.10. Работа редактор, указывающего ошибку

В редакторе DRAKON WEB Editor реализован полный набор правил визуального синтаксиса, что освобождает пользователя от необходимости детально помнить синтаксические правила. DRAKON WEB Editor создает только правильно созданные диаграммы и не допускает диаграммы, в которых нарушен синтаксис, в генерацию программного кода.

**3.3. Базовые структуры алгоритмов редактора DRAKON WEB Editor**

Логическая структура какого-либо алгоритма может быть представлена комбинацией трех базовых структур: *линейной, разветвленной и циклической.* Характерной особенностью базовых структур является наличие в них одного входа и одного вихода.

3.3.1. Линейная структура

Линейная структура алгоритмического процесса реализует операции, выполняемые последовательно в порядке их записи. Типичным примером такого процесса является стандартная вычислительная схема, состоящая из трех этапов:

a) введение входных данных; б) вычисление по формулам; в) вывод результата. Графическое изображение базового элемента линейной структуры в программных продуктах линейки DRAKON имеет вид простых прямоугольников, Например, в алгоритме нахождения минимального и максимального значений массива переменным minim и maxim присваивается значение нулевого элемента массива array (рис.3.11).

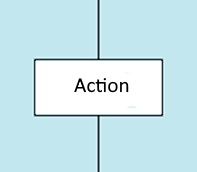
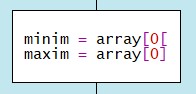
 

Рис.3.11. Линейная структура дракон-диаграммы

3.3.2. Базовые конструкции разветвленной структуры

Разветвленная конструкция содержит хотя бы одну проверку условия, в результате которой обеспечивается переход на один из возможных вариантов решения. Каждый из вариантов ведет к общему выходу, то есть работа алгоритма будет продолжаться независимо от того, какой путь будет выбран. Структура разветвления существует в двух основных вариантах:

а). дракон-фрагмент конструкции “если (*условие*), то (*действия*) иначе (*действия*)”, то-есть в языках программирования это оператор *if...else.* Пример применения конструкции if-else в дракон-диаграмме в алгоритме нахождения минимального и максимального значений массива (рис. 3.12):

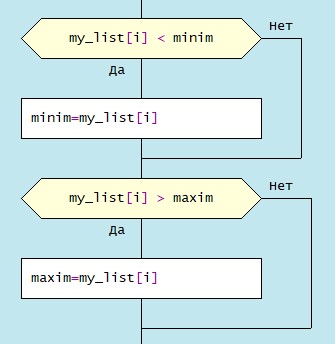


Рис. 3.12. Фрагмент диаграммы с конструкцией if-else

б). дракон-фрагмент конструкции “Выбор”

Значения, с которыми будет сравниваться выражение в графоэлементе «Выбор», нахолятся в графоэлементах «Вариант». Если в крайнем правом варианте отсутствует текст, это означает «все другие значения». Ниже — пример выбора разветвления алгоритма сортировки (merge sort) (3.13).

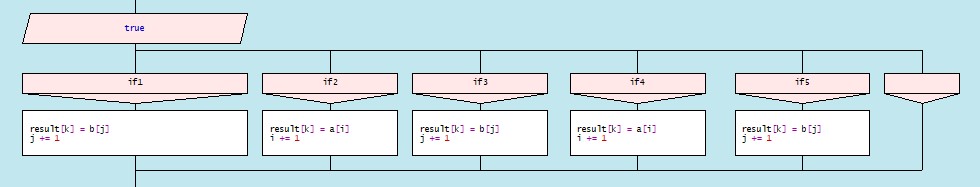


Рис.3.13 Фрагмент выбора разветвления (оператор Select)

3.1.3. Базовые конструкции циклической структуры

Циклическая структура предполагает многократное повторение одной и той же последовательности действий. Количество повторений определяется входными данными или условиями задачи. К циклическим структурам относятся, прежде всего, конструкция "Цикл-Для" (“for”), “Цикл для каждого” (“foreach”), составные конструкции "Цикл-До" (“while-do”) и "Цикл-Пока" (“do-while”).

а). “Цикл-Для” состоит из трех частей. В первой части фиксируется инициализация цикла. Во второй следует проверка условия завершения цикла. Если оно истинно, то выполняются операторы тела цикла до тех пор, пока это выражение не станет ложным. Если оно ложно, цикл заканчивается и управление передается следующему оператору. В третьей части увеличивается параметр цикла. Фрагмент дракон-диаграммы с конструкцией “Цикл-Для” имеет вид (рис. 3.14):

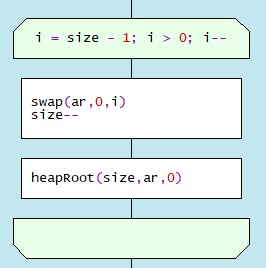


Рис.3.14. Фрагмент «Цикл-Для»

б). “Цикл для каждого” (“foreach”) выполняет оператор или блок операторов для каждого элемента массива или списка данных (3.15).

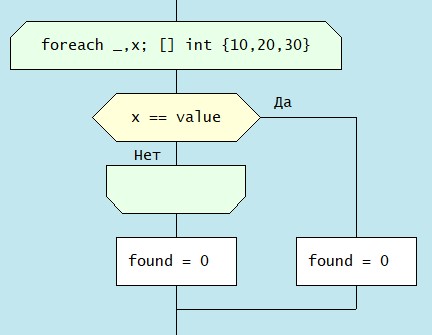


Рис. 3.15. Фрагмент диаграммы «Цикл для каждого»

в). пример составной конструкции “Цикл-До” (рис.3.16):

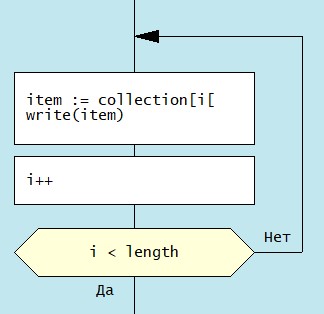


Рис. 3.16. Фрагмент диаграммы «Цикл-До»

г). пример составной конструкции “Цикл-После” (рис.3.17):

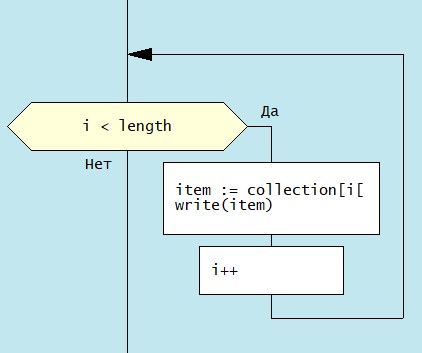


Рис. 3.18 Фрагмент диаграммы «Цикл-После»

3.1.4. Пример построения DRAKON-диаграммы

Покажем на конкретном примере построение диаграммы алгоритма вычисления наибольшего общего делителя методом Евклида. Наибольший общий делитель (НОД) – это число, которое делит без остатка два числа и делится само без остатка на любой другой делитель данных двух чисел. Проще говоря, это самое большое число, на которое можно без остатка разделить два числа, для которых ищется НОД.

Алгоритм нахождения НОД делением методом Евклида заключается в следующем:

1. Большее число делим на меньшее.
2. Если делится без остатка, то меньшее число и есть НОД (следует выйти из цикла).
3. Если есть остаток, то большее число заменяем на остаток от деления.
4. Переходим к пункту 1.

Дракон-диаграмма алгоритма нахождения наибольшего общего делителя показана на рис.3.19.

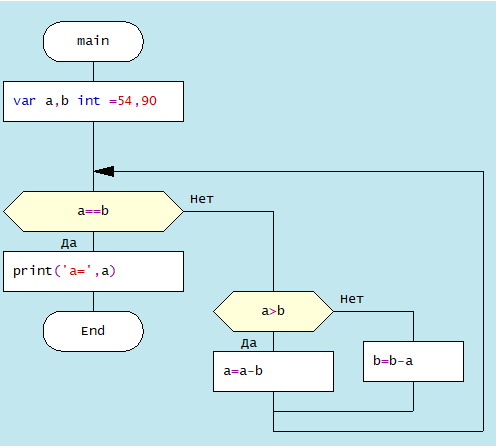


Рис.3.19 Дракон-диаграмма алгоритма Эвклида

**3.4. От дракон-диаграммы до программного кода**

Каждая дракон-диаграмма соответствует программному модулю. На рис.3.20 показана структура алгоритма сортировки массива, состоящая из ряда отдельных модулей (дракон-диаграмм) и дракон-диаграммы main, содержащей конструкции головной программы main на языке Go:

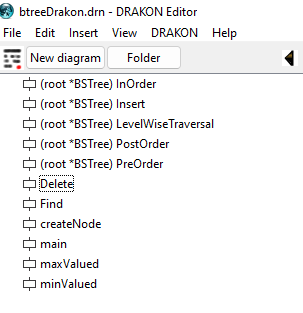


Рис. 3.20. Структура программы обработки деревьев

Как уже было изложено выше, структура программы на языке программирования Golang состоит из пакетов. Конструкции языка, входящие в пакет заносятся в раздел File/Description программного интерфейса редактора DRAKON WEB Editor (рис. 3.21.):

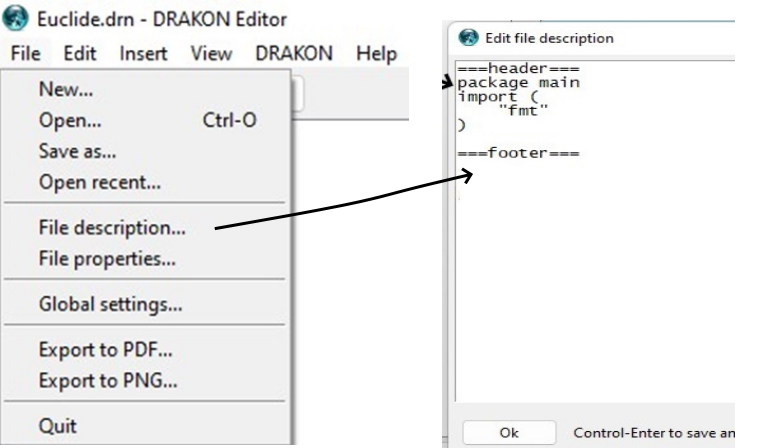


Рис.3.21. Порядок описания структуры программы через File/Description

В поле между метками ===header=== и ===footer=== следует располагать конструкции языка, доступные в каждом модуле программы, например, глобальные переменные.

Далее пользователь должен указать используемый язык программирования. Для этого нужно открыть опцию File properties (рис. 3.22):

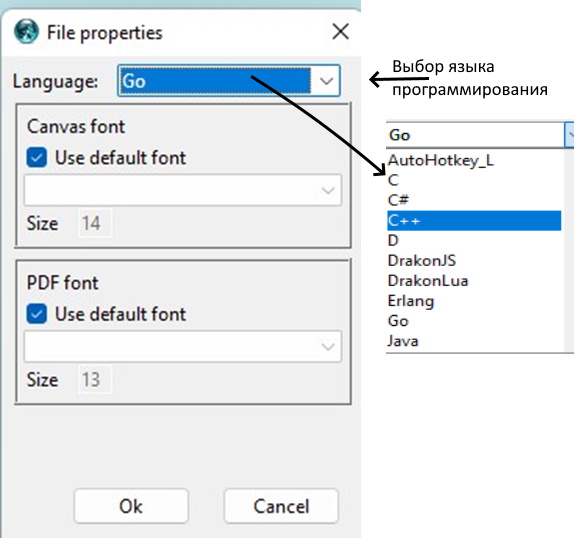


Рис. 3.22. Выбор языка программирования

Завершающим этапом разработки является генерирование программного кода (рис.3.23):

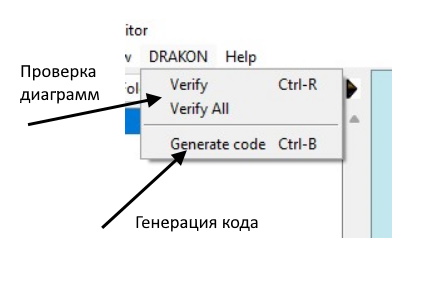


Рис. 3.23. Опция «Генерация кода»

Исходный код в формате .drn и сгенерированный код в формате .go сохраняются в одной и той же той директории (рис. 3.24):

**3.5. Работа с генерированным кодом**

Сгенерированный код можно открыть в одной из интегрированных сред разработки (IDE): Visual Studio Code, VIM, Eclipse, Atom, Sublime Text и ряда других [[Top 5 Best Golang IDE - GoLang Docs](https://golangdocs.com/best-golang-ide)]. В данной работе используется IDE Visual Studio Code (VSC), описание которой не входит в содержание книги. Внешний вид среды программирования VSC представлен на рис.3.24:

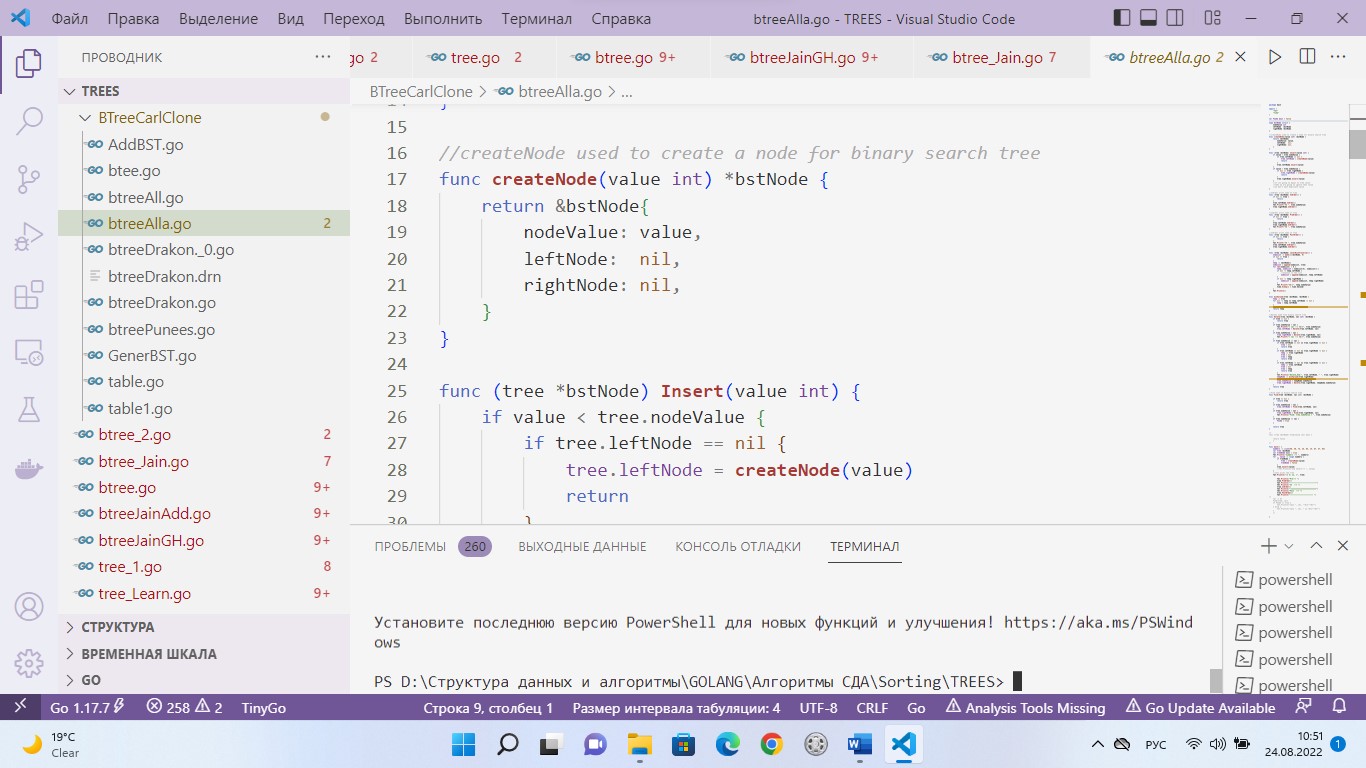


Рис.3.24. Внешний вид программы в среде Visual Studio Code

Практически в рамках гибридного подхода отладку программы зачастую приходится выполнять параллельно в двух средах: DRAKON WEB Editor и Visual Studio Code. Естественно, что при наличии графических синтаксических ошибок а дракон диаграмме генерация кода производиться не будет, в то же время наличие смысловых ошибок в алгоритме код сгенерируется, но результаты окажутся неверными.

Процесс отладки сгенерированного кода рекомендуется проводить c помощью инструментария соответствующей среды программирования. После завершения отладки все исправления необходимо внести в соответствующие графоэлементы дракон-диаграммы. Во всяком случае необходимо всегда следить за соответствием содержимого графоєлементов диаграммы и программного кода в соответствующей среде программирования.

[Download Visual Studio Code - Mac, Linux, Windows](https://code.visualstudio.com/Download)