Алгоритмический язык ДАЛВЯЗ 2
Силуэтное программирование

Барановский Дмитрий Владимирович
инженер-программист
г. Москва, 2016 г.

БЛАГОДАРНОСТИ

Большое спасибо Владимиру Даниеловичу Паронджанову за его идеи в области визуального программирования, которые и побудили меня серьезно задуматься над вопросом – а для чего нужно визуальное программирование и каким оно должно быть.

Благодарю Владислава Жаринова, Алексея Донского, Андрея Тюгашева, Эдуарда Ильченко, Степана Митькина и других участников форума http://forum.oberoncore.ru за активное обсуждение вопросов и интересные творческие идеи в области визуального программирования.

Содержание

Введение

Наконец-то в 2016, без малого через четыре года после того, как я выложил на форуме http://forum.oberoncore.ru описание алгоритмического языка ДАЛВЯЗ 2 (файл dalvjaz2_description.pdf), я смог сформулировать, какой смысл применительно к языку ДАЛВЯЗ 2 имеет термин «силуэтное программирование».

Силуэтное программирование на языке ДАЛВЯЗ 2 – это совокупность приемов визуального программирования, применяемая для разработки в графическом редакторе ДАЛВЯЗ 2 в формате структурных блок-схем процедур с циклом сложного условия и обеспечивающая программисту более комфортные с точки зрения эргономики, чем при вводе в текстовом редакторе, условия ввода исходного кода процедуры.

Ну, такое утверждение хорошо бы доказать, а еще лучше проверить на практике. А то уже вон сколько графических систем программирования с редактированием исходного кода в формате блок-схем понапридумывали, а они у программистов широкого признания не завоевали, говорят – неудобно.

Почему неудобно и как удобно ?

Рассуждения на эту тему и составляют основное содержание этого материала, в котором я объединил в одно целое уже имевшиеся у меня разные варианты описания языка ДАЛВЯЗ 2, задуманного мной как сильно упрощенное и удобное для практического программирования на языках семейств Pascal и C/C++ подобие алгоритмического языка ДРАКОН.

ЧАСТЬ 1. ТЕОРИЯ

1.1. Почему неудобны стандартные блок-схемы

Блок-схемы формата ГОСТ 19.701-90 неудобны по следующим причинам:

1) трудоемкость ввода и исправления блок-схемы в графическом редакторе: нужно вручную задавать позиции элементов и связи между элементами блок-схемы;

2) отсутствие четких правил группировки элементов на схеме, что при нарастании сложности алгоритма ухудшает читаемость схемы;

3) допустимость пересечения линий на схеме, что ухудшает читаемость схемы;

4) неоднозначность направления потока выполнения от одного блока схемы к другому, что ухудшает читаемость схемы; для разрешения этой неоднозначности нужно использовать стрелки для указания последовательности выполнения действий,

5) значительное число предопределенных типов элементов схемы, что затрудняет восприятие схемы;

6) несоответствие блок-схем по ГОСТ 19.701-90 правилам структурного программирования, что затрудняет получение качественного, с точки зрения правил структурного программирования, исходного кода при генерации его на основе введенной в графическом редакторе блок-схемы;

7) построение блок-схемы из произвольно взятого исходного кода является, в общем случае, не такой уж простой задачей, от которой разработчики многих графических сред программирования просто отмахиваются: мол, мы строим блок-схемы только из ранее сгенерированного нами кода.

1.2. Что такое силуэт

В СССР наиболее известная, если судить по материалам сети Интернет, попытка усовершенствовать блок-схемы с точки зрения эргономичности их восприятия была предпринята В.Д. Пароджановым при создании им алгоритмического языка ДРАКОН (Дружелюбный Русский Алгоритмический язык, Который Обеспечивает Наглядность).

Ключевым понятием языка ДРАКОН является введенное В.Д. Пароджановым понятие «силуэт» (рис. 1.1 и рис. 1.2).

Силуэт в языке ДРАКОН представляет собой визуально-логическую конструкцию, позволяющую изменить компоновку элементов блок-схемы с преимущественно вертикальной на горизонтальную, что дает возможность просмотра схемы, умещающейся по высоте в один экран, при помощи только горизонтальной полосы прокрутки; такой способ просмотра схемы с точки зрения эргономики является более предпочтительным по сравнению с просмотром схемы как с применением вертикальной полосы прокрутки (т.к. поле зрения вытянуто по горизонтали и, соответственно, зрительным осям глаз чаще приходится совершать горизонтальные движения, чем вертикальные) так и с применением обеих полос прокрутки одновременно.

Введение в языке ДРАКОН визуальной схемы типа «силуэт», запрета на пересечение линий схемы и формализации направления выполнения действий на схеме сверху вниз и слева направо, кроме обратных петель цикла и обратной петли силуэта, позволило устранить недостатки 2, 3, 4 блок-схем по ГОСТ 19.701-90, но недостатки 1, 5, 6, 7 создателю языка ДРАКОН преодолеть так и не удалось.

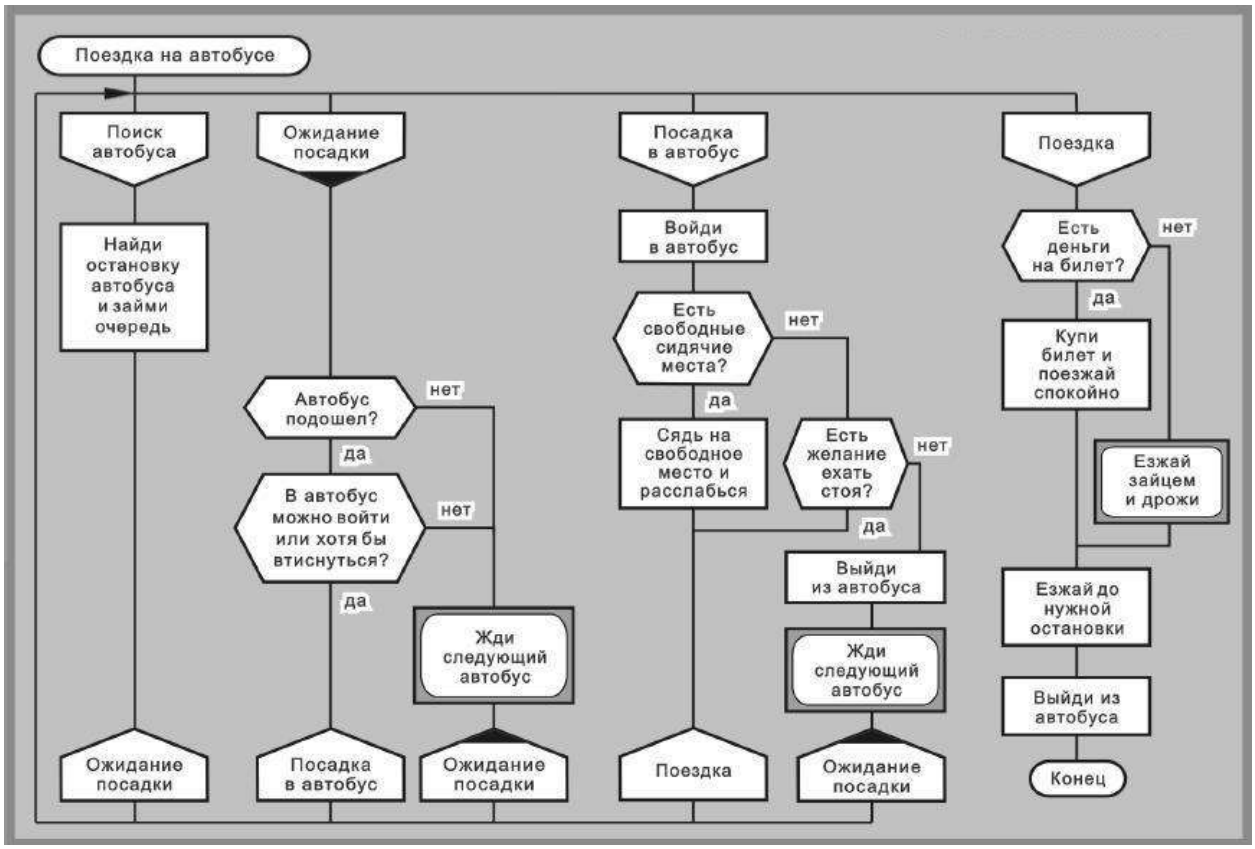

Рисунок 1.1 – Полная форма схемы «Силуэт»

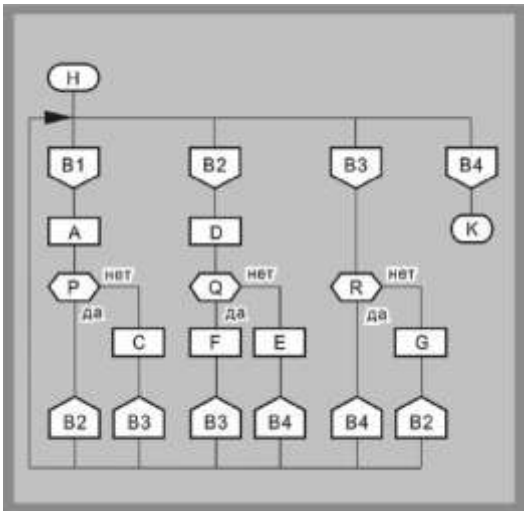

Рисунок 1.2 – Минимизированная форма схемы «Силуэт»

1.3. Алгоритмический язык ДАЛВЯЗ 2

Алгоритмический язык ДАЛВЯЗ 2 (Дружелюбный АЛгоритмический Визуальный ЯЗык, версия 2) определяет следующие правила визуальной разработки исходного текста процедур в формате структурных блок-схем:

- структурные блок-схемы создаются с использованием трех основных логических конструкций (маршрутных операторов) структурного программирования:

1) действие;

2) сложное условие;

3) цикл;

- для структурных блок-схем определены следующие типы действий:

1) начало схемы;

2) конец схемы;

3) стандартное действие (в том числе задание номера ветки сложного условия);

4) действие для выполнения одного из дополнительных маршрутных операторов (выход из цикла, продолжение цикла, выход из процедуры, переход goto, точка перехода goto, переход к обработке ошибок, блок обработки ошибок); каждому дополнительному маршрутному оператору соответствует своя пиктограма, наиболее часто употребляемыми дополнительными маршрутными операторами являются «выход из цикла» (break) и «выход из процедуры» (return); подробнее работа с пиктограммами рассматривается в пункте 2.6 настоящего материала;

- для всех блоков условий всегда выполняется правило: если условие выполняется, то переход вниз, а если условие не выполняется, то переход вправо, поэтому для блоков условий в ДАЛВЯЗ 2 метки переходов «да» и «нет» не указываются; для обозначения формы блоков условий на схеме для полной формы отображения схемы я использую термин «раздвинутый ромб»;

- тип цикла − цикл ДЛЯ (for), цикл ПОКА (while) или цикл ПОВТОРЯТЬ ДО (repeat … until) − определяется после размещения логической конструкции цикла на схеме путем задания конкретных операторов начала и конца цикла.

Конструкция «силуэт» реализуется в ДАЛВЯЗ 2 при помощи цикла сложного условия (цикла-силуэта), показанного на рис. 1.3.

Из рис. 1.3 видно, что форма блока действия, в котором задается номер следующей ветки сложного условия, совпадает с формой блока задания имени следующей ветки в языке ДРАКОН.

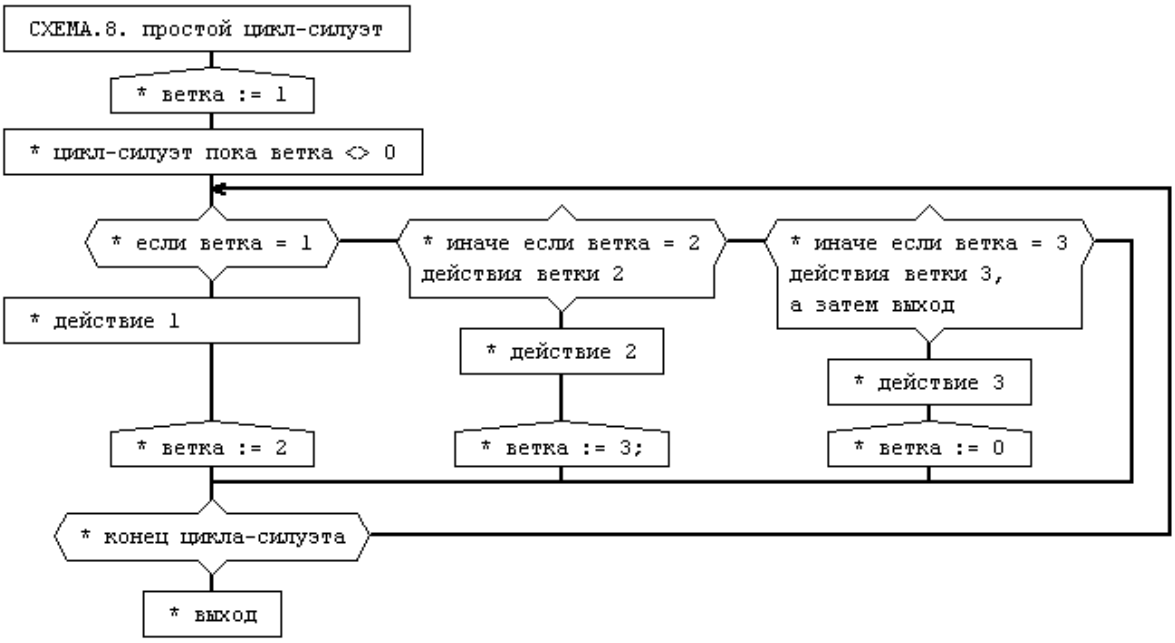

Рисунок 1.3 – Цикл сложного условия (цикл-силуэт)

Что касается термина «силуэт», то при программировании с циклом сложного условия он применяется в своем буквальном значении: обратная петля цикла сложного условия охватывает процедуру, тем самым формируя внешний контур (силуэт) схемы процедуры .

1.4. Формирование в ДАЛВЯЗ 2 блок-схемы на основе исходного кода процедуры

В ДАЛВЯЗ 2 блок-схема на основе исходного кода процедуры (или псевдокода, описывающего логику процедуры) создается при помощи следующих типов служебных комментариев, включаемых в текст исходного кода процедуры:

* СХЕМА.N. – заголовок схемы
* если – начало блока сложного условия;
* иначе если – дополнительное условие сложного условия;
* иначе – альтернатива сложного условия
* конец – конец сложного условия;
* цикл – начало цикла;
* конец цикла – окончание цикла;
* действие – начало действия;
* ветка = – задание заголовка ветки силуэта;
* ветка := – задание адреса перехода к следующей ветке
* выход – выход из процедуры, возможно с возвратом значения;
* КОНЕЦ СХЕМЫ – конец исходного кода (псевдокода), задающего визуальную схему.

В конфигурации служебные комментарии задаются как символьные строки. В некоторых случаях служебные комментарии могут совпадать с ключевыми словами языка программирования. Ниже приведены наборы служебных комментариев для языка программирования LUA и для русскоязычной версии языка программирования Компонентный паскаль.

Набор служебных комментариев для языка программирования LUA:

```
prg_otkr_kmnt      = "--"  // строчный комментарий
prg_zakr_kmnt      = " "   // закрывающий комментарий отсутствует
zagol_prg          = "-- i."   // заголовок процедуры
prg_t_IF           = "if"
prg_t_EI           = "elseif"
prg_t_EB           = "else"
prg_t_E            = "end;"
prg_t_CB           = "-- цикл"
prg_t_CE           = "-- конец цикла"
prg_t_A            = "--a"   // начало действия
prg_prow_vetki     = "vetka == "  // проверка ветки в условии
prg_k_vetke        = "vetka = "    // задание новой ветки
prg_t_RET          = "-- выход"
prg_STOP           = "-- конец процедуры"
```

Набор служебных комментариев для русскоязычной версии языка программирования Компонентный паскаль:

```
prg_otkr_kmnt      = "(*"  // открывающий комментарий
prg_zakr_kmnt      = "*)"  // закрывающий комментарий
zagol_prg          = "(* i."  // заголовок процедуры
prg_t_IF           = "ЕСЛИ"
prg_t_EI           = "ИНЕСЛИ"
prg_t_EB           = "ИНАЧЕ"
prg_t_E            = "КОНЕЦ;"
prg_t_CB           = "ЦИКЛ"
prg_t_CE           = "КОНЕЦ_ЦИКЛА"
prg_t_A            = "(**"   // начало действия
prg_prow_vetki     = "ветка = "  // проверка ветки в условии
prg_k_vetke        = "ветка := "    // задание новой ветки
prg_t_RET          = " (** ВЫХОД"
prg_STOP           = " (** КОНЕЦ ПРОЦЕДУРЫ"
```

Ниже приведен набор служебных комментариев для простой процедуры на языке Компонентный паскаль.

```
(* i.1 процедура *)
  (** Действие 1 *)
  ЕСЛИ  Условие 1
  ИНЕСЛИ  Условие 2
    (** Действие 2 *)
  ИНАЧЕ
    (** Действие 3 *)
  КОНЕЦ;
  (** ВЫХОД *)
(** КОНЕЦ ПРОЦЕДУРЫ *)
```

1.5. Программирование на ДАЛВЯЗ 2 с циклом сложного условия

При программировании с циклом сложного условия логика процедуры размещается в ветках сложного условия, находящегося внутри цикла.

Параметром условий цикла и веток сложного условия является переменная **vetka**. Цикл условия выполняется, пока переменная **vetka** не равна 0. Каждой ветке сложного условия соответствует свое значение переменной **vetka**. Для перехода к следующей ветке нужно задать переменной **vetka** соответствующее значение. Для выхода из цикла условия нужно задать значение переменной **vetka** равным 0.

Программирование с циклом сложного условия позволяет:

- разбивать логику процедуры на компактные и легко обозримые фрагменты, находящиеся в ветках условия;

- изменить компоновку элементов блок-схемы с преимущественно вертикальной на горизонтальную, что дает возможность просмотра схемы, умещающейся по высоте в один экран, при помощи только горизонтальной полосы прокрутки;

- организовывать веточные циклы, при которых выполняется переход из веток условия со старшими номерами к веткам условия с младшими номерами; для задания в условии признака веточного цикла нужно ввести в условие выражение следующего вида:

elseif
vetka == 500 -- vetka = 800
then
где vetka = 800 будет указывать на последнюю ветку веточного цикла;

- в процессе разработки схемы при необходимости проводить нормализацию номеров веток условия с шагом 10, а после окончания разработки схемы провести нормализацию номеров веток условий с шагом 1, чтобы все ветки условия для удобства просмотра были пронумерованы по порядку следования; в тексте процедуры будут подвергаться нормализации выражения вида «vetka == N» и «vetka = N» (пример для LUA и C/C++), так что в условии ветки начала веточного цикла номера веток начала и конца цикла будут нормализованы автоматически;

- переходить к нужной ветке путем наведения указателя мыши на блок, задающий переменной **vetka** значение, равное номеру нужной ветки, с последующим нажатием правой кнопки мыши.

Иногда даже при организации цикла сложного условия бывает трудно уместить схему процедуры по высоте в один экран. Для решения этой проблемы в конфигурацию введены параметры (далее приводятся примеры для Компонентного паскаля) «маскировать строку» "(*[-]*)", «маскировать все строки кроме 1-й» "|V|" (размещается в 1-й строке элемента), «маскировать ключевое слово» "(*--*)".

Приведем пример применения параметра «маскировать ключевое слово», который позволяет спрятать условие в блоке действия:

```
(**  блок действия 1 *)
…
ЕСЛИ  условие  ТО   (*--*)   (* маскируем начало условия *)
  …
КОНЕЦ;   (*--*)   (* маскируем конец условия *)
…
(**  блок действия 2 *)
```

При создании схемы будут созданы блоки действий 1 и 2, а ключевые слова условия будут проигнорированы.

1.6. Конкретный пример схемы

Возьмем в качестве примера текст реальной процедуры, написанной на русскоязычной версии Компонентного паскаля.

```
(* i.4.  вывод_схемы;  |V|  *)

001  ПРОЦЕДУРА  вывод_схемы;
002  ПЕРЕМЕННЫЕ ветка, ж, ии, и, смещение_эл_х, х1,х2,у1,у2,
003             вэкв, ээдв:ЦЕЛЫЙ;
004  НАЧАЛО
005     ветка := 10;
006     ЦИКЛ_ПОКА (ветка # 0)  ДЕЛАТЬ
007        ЕСЛИ (ветка = 10)  ТО
008           ветка := 20;
009           ЕСЛИ (запрос(З_СХЕМА_НАЧАЛО_ПРОГРАММЫ) = ОТВЕТ_ДА) ТО
010              (**) ВЫХОД;
011           КОНЕЦ;
012
```

```
013           команда(К_СХЕМА_ПОЛУЧИТЬ_РАЗМЕРЫ_КАНВЫ);
014           ЕСЛИ (запрос(З_СХЕМА_ТЕКСТОВАЯ_ЗАПИСЬ) = ОТВЕТ_ДА) ТО
015              команда(К_СХЕМА_ВЫВОД_ТЕКСТА);
016              команда  (К_СХЕМА_ОБНОВИТЬ_КАНВУ);
017              (**) ВЫХОД;
018           КОНЕЦ;
019
020        ИНЕСЛИ (ветка = 20) ТО
021           ветка := 30;
022           команда(К_СХЕМА_СТИЛЬ__СОЕДИНИТЕЛЬНЫЕ_ЛИНИИ);
023           команда  (К_СХЕМА_СТИЛЬ__ЦВЕТ_ФОНА);
024           команда  (К_СХЕМА_ФОН_СХЕМЫ);
025           ЕСЛИ (запрос(З_СХЕМА_ПОДСЧЕТ_МАКС_КООРДИНАТ) = ОТВЕТ_НЕТ) ТО
026              команда(К_СХЕМА_ОБНОВИТЬ_КАНВУ);
027              (**) ВЫХОД;
028           КОНЕЦ;
029
030        ИНЕСЛИ (ветка = 30) ТО
031           ветка := 40;
032           (** цикл вывода главной вертикали - задание начальных значений  |V|*)
033               ж :=0;
034               ии :=значение(В_СХЕМА_ИНД_1_ЭЛ_ВЕРТИКАЛИ, ж);
035               и :=0;
036           ЦИКЛ_ПОКА (и < значение(В_СХЕМА_КОЛ_ЭЛ_ВЕРТИКАЛИ, ж)) ДЕЛАТЬ
037              ЕСЛИ (значение(В_СХЕМА_ВЕРТИКАЛЬ_ЭЛЕМЕНТА, ии) = ж) ТО
038                 ЕСЛИ (значение(В_СХЕМА_ЭЛЕМЕНТ_ИСПОЛЬЗУЕТСЯ, ии) = ОТВЕТ_ДА) ТО
039                    (**  К_СХЕМА_ВЫВОД_ЭЛЕМЕНТА  |V| *)
040                    данные (Д_СХЕМА_ВЕРТИКАЛЬ, ж);
041                    данные (Д_СХЕМА_ИНДЕКС_ЭЛЕМЕНТА, ии);
042                    данные (Д_СХЕМА_ТЕК_ЭЛ_ВЕРТИКАЛИ, и);
043                    команда (К_СХЕМА_ВЫВОД_ЭЛЕМЕНТА);
044                    УВЕЛИЧИТЬ(и);
045                 КОНЕЦ;
046              КОНЕЦ;
047
048              УВЕЛИЧИТЬ(ии);
049           КОНЕЦ_ЦИКЛА;
050        ИНЕСЛИ (ветка = 40) ТО
051           (** начало веточного цикла для элементов доп. вертикалей *)
052           ж :=0;
053           ЕСЛИ (ж < запрос(З_СХЕМА_КОЛ_ДОП_ВЕРТИКАЛЕЙ)) ТО
054              ветка := 50;
055           ИНАЧЕ
056              команда(К_СХЕМА_ОБНОВИТЬ_КАНВУ);
057              ветка := 0;
058           КОНЕЦ;
059        ИНЕСЛИ (ветка = 50) ТО   (****  ветка := 120 *)
060           ветка := 60;
061           смещение_эл_х := 0;
062           ЕСЛИ ((значение(В_СХЕМА_РОДИТЕЛЬ_ВЕРТИКАЛИ, ж) # НЕ_ЗАДАН) &
063                 (значение(В_СХЕМА_ТИП_ДОП_ВЕРТИКАЛИ,ж)# ВЕРТИКАЛЬ_ЦИКЛА)) ТО
064              смещение_эл_х := значение(В_СХЕМА_ШИРИНА_РОДИТЕЛЯ_ВЕРТИКАЛИ,ж);
065           КОНЕЦ;
066
067           (** получаем  х1  |V| *)
068           х1 :=значение(В_СХЕМА_Х_ВЕРТИКАЛИ_РОДИТЕЛЯ, ж) +
069                 запрос(З_СХЕМА_СМЕЩЕНИЕ_СХЕМЫ_ПО_Х) +
070                 (смещение_эл_х ДЕЛИТЬ 2);
071           (** получаем  х2  |V| *)
072           х2 :=запрос(З_СХЕМА_СМЕЩЕНИЕ_СХЕМЫ_ПО_Х) +
073                 значение(В_СХЕМА_Х_ДОП_ВЕРТИКАЛИ,ж);
074           у1 := запрос(З_СХЕМА_СМЕЩЕНИЕ_СХЕМЫ_ПО_У) +
075                 значение(В_СХЕМА_У1_ДОП_ВЕРТИКАЛИ,ж);
```

```
076         ИНЕСЛИ (ветка = 60) ТО
077            ветка := 70;
078            ЕСЛИ (значение(В_СХЕМА_ТИП_ДОП_ВЕРТИКАЛИ,ж) = ВЕРТИКАЛЬ_ДОЧЬ) ТО
079               ЕСЛИ (значение(В_СХЕМА_ТИП_1_ЭЛ_ДОП_ВЕРТИКАЛИ,ж) = т_ИНЕСЛИ) ТО
080                  ЕСЛИ (значение(В_СХЕМА_ФЛАГ_ВЕТКИ_1_ЭЛ_ДОП_ВЕРТИКАЛИ,ж) =
081                                                                  ОТВЕТ_ДА) ТО
082                     смещение_эл_х :=
083                                 значение(В_СХЕМА_ШИРИНА_1_ЭЛ_ДОП_ВЕРТИКАЛИ,ж);
084                     х2 := х2 - (смещение_эл_х ДЕЛИТЬ 2);
085                  КОНЕЦ;
086               КОНЕЦ;
087
088            ИНАЧЕ
089               ветка := 90;
090            КОНЕЦ;
091         ИНЕСЛИ (ветка = 70) ТО
092            ветка := 80;
093            данные(Д_СХЕМА_КООРД_Х1, х1);    данные(Д_СХЕМА_КООРД_У1, у1);
094            данные (Д_СХЕМА_КООРД_Х2, х2);   данные (Д_СХЕМА_КООРД_У2, у1);
095            команда (К_СХЕМА_ЛИНИЯ);
096            вэкв := значение(В_СХЕМА_ВЕРТИКАЛЬ_ЭЛ_КОНЦА_ВЕРТИКАЛИ, ж);
097            (** получаем  х1  |V| *)
098            х1 :=запрос(З_СХЕМА_СМЕЩЕНИЕ_СХЕМЫ_ПО_Х) +
099                  значение(В_СХЕМА_Х_ВЕРТИКАЛИ,вэкв);
100            (** получаем  х2  |V| *)
101            х2 :=запрос(З_СХЕМА_СМЕЩЕНИЕ_СХЕМЫ_ПО_Х) +
102                  значение(В_СХЕМА_Х_ДОП_ВЕРТИКАЛИ, ж);
103            (** получаем  у2  |V| *)
104            у2 :=запрос(З_СХЕМА_СМЕЩЕНИЕ_СХЕМЫ_ПО_У) +
105                  значение(В_СХЕМА_У1_ДОП_ВЕРТИКАЛИ,ж) +
106                  значение(В_СХЕМА_ТЕК_У_ДОП_ВЕРТИКАЛИ,ж);
107            данные(Д_СХЕМА_КООРД_Х1, х1);    данные(Д_СХЕМА_КООРД_У1, у2);
108            данные (Д_СХЕМА_КООРД_Х2, х2);   данные (Д_СХЕМА_КООРД_У2, у2);
109            команда (К_СХЕМА_ЛИНИЯ);
110         ИНЕСЛИ (ветка = 80) ТО
111            ветка := 90;
112            ЕСЛИ (значение(В_СХЕМА_КОЛ_ЭЛ_ДОП_ВЕРТИКАЛИ, ж) # 0) ТО
113               ЕСЛИ (значение(В_СХЕМА_ТИП_1_ЭЛ_ДОП_ВЕРТИКАЛИ,ж) #
114                                                            т_ИНАЧЕ) ТО
115                  ЕСЛИ (значение(В_СХЕМА_ТИП_1_ЭЛ_ДОП_ВЕРТИКАЛИ,ж)
116                                                   # т_ИНЕСЛИ) ИЛИ
117                    (значение(В_СХЕМА_ФЛАГ_ВЕТКИ_1_ЭЛ_ДОП_ВЕРТИКАЛИ,ж)
118                                                   =ОТВЕТ_НЕТ) ТО
119                     (** получаем  у1  и  у2  |V| *)
120                     у1 :=запрос(З_СХЕМА_СМЕЩЕНИЕ_СХЕМЫ_ПО_У) +
121                           значение(В_СХЕМА_У1_ДОП_ВЕРТИКАЛИ,ж);
122
123                     у2 :=у1 + значение(В_СХЕМА_У1_1_ЭЛ_ДОП_ВЕРТИКАЛИ,ж);
124                     (**  К_СХЕМА_ЛИНИЯ  |V| *)
125                     данные (Д_СХЕМА_КООРД_Х1, х2);
126                     данные (Д_СХЕМА_КООРД_У1, у1);
127                     данные (Д_СХЕМА_КООРД_Х2,х2);
128                     данные (Д_СХЕМА_КООРД_У2,у2);
129                     команда (К_СХЕМА_ЛИНИЯ);
130                  КОНЕЦ;
131               КОНЕЦ;
132            КОНЕЦ;
133         ИНЕСЛИ (ветка = 90) ТО
134            ветка := 100;
135            ЕСЛИ (значение(В_СХЕМА_ТИП_ДОП_ВЕРТИКАЛИ,ж) = ВЕРТИКАЛЬ_ЦИКЛА) ТО
136               у1 := значение(В_СХЕМА_ВЫШЕ_ЗАГОЛ_ВЕТОК, у1);
137               (**  К_СХЕМА_ЛИНИЯ  |V| *)
138               данные (Д_СХЕМА_КООРД_Х1, х1);    данные (Д_СХЕМА_КООРД_У1, у1);
```

```
139             данные (Д_СХЕМА_КООРД_Х2, х2);   данные (Д_СХЕМА_КООРД_У2, у1);
140             команда (К_СХЕМА_ЛИНИЯ);
141             (**  К_СХЕМА_СТРЕЛКА_ЦИКЛА  |V| *)
142             данные (Д_СХЕМА_КООРД_Х1, х1);
143             данные (Д_СХЕМА_КООРД_У1, у1);
144             команда (К_СХЕМА_СТРЕЛКА_ЦИКЛА);
145             (**  получаем  у2  и  эздв  |V|  *)
146             у2 :=у1 + значение(В_СХЕМА_ТЕК_У_ДОП_ВЕРТИКАЛИ,ж) +
147                        (запрос(З_СХЕМА_ШАГ_ТЕКСТА_ПО_У) ДЕЛИТЬ 4);
148
149             эздв:= значение(В_СХЕМА_ЭЛ_ЗАКРЫВАЮЩИЙ_ДОП_ВЕРТИКАЛЬ,ж);
150                             (* КОНЕЦ/КОНЕЦ_ЦИКЛА *)
151             смещение_эл_х := значение(В_СХЕМА_ШИРИНА_ЭЛЕМЕНТА, эздв);
152             х1 := х1 + (смещение_эл_х ДЕЛИТЬ 2);
153             (**  К_СХЕМА_ЛИНИЯ  |V| *)
154             данные (Д_СХЕМА_КООРД_Х1, х1);    данные (Д_СХЕМА_КООРД_У1, у2);
155             данные (Д_СХЕМА_КООРД_Х2, х2);   данные (Д_СХЕМА_КООРД_У2, у2);
156             команда (К_СХЕМА_ЛИНИЯ);
157          КОНЕЦ;
158
159       ИНЕСЛИ (ветка = 100) ТО
160          ЕСЛИ (значение(В_СХЕМА_КОЛ_ЭЛ_ДОП_ВЕРТИКАЛИ, ж) = 0) ТО
161             ветка := 120;
162             у1 := запрос(З_СХЕМА_СМЕЩЕНИЕ_СХЕМЫ_ПО_У) +
163                   значение(В_СХЕМА_У1_ДОП_ВЕРТИКАЛИ, ж);
164             у2 := у1 + значение(В_СХЕМА_ТЕК_У_ДОП_ВЕРТИКАЛИ, ж);
165             ЕСЛИ (значение(В_СХЕМА_ТИП_ДОП_ВЕРТИКАЛИ, ж) =
166                                                   ВЕРТИКАЛЬ_ЦИКЛА) ТО
167                у1 := значение(В_СХЕМА_ВЫШЕ_ЗАГОЛ_ВЕТОК, у1);
168             КОНЕЦ;
169             (**   К_СХЕМА_ЛИНИЯ  |V| *)
170             данные (Д_СХЕМА_КООРД_Х1, х2);   данные (Д_СХЕМА_КООРД_У1, у1);
171             данные (Д_СХЕМА_КООРД_Х2, х2);   данные (Д_СХЕМА_КООРД_У2, у2);
172             команда (К_СХЕМА_ЛИНИЯ);
173          ИНАЧЕ
174             ветка := 110;
175          КОНЕЦ;
176       ИНЕСЛИ (ветка = 110) ТО
177          ветка := 120;
178          (** задание значений для цикла по всем эл-там доп. вертикали  |V| *)
179          ии :=значение(В_СХЕМА_ИНД_1_ЭЛ_ДОП_ВЕРТИКАЛИ, ж);
180          и :=0;
181          ЦИКЛ_ПОКА (и < значение(В_СХЕМА_КОЛ_ЭЛ_ДОП_ВЕРТИКАЛИ,ж)) ДЕЛАТЬ
182             ЕСЛИ (значение(В_СХЕМА_ВЕРТИКАЛЬ_ЭЛЕМЕНТА, ии) =
183                   значение(В_СХЕМА_ДОП_ВЕРТИКАЛЬ, ж)) ТО
184                ЕСЛИ (значение(В_СХЕМА_ЭЛЕМЕНТ_ИСПОЛЬЗУЕТСЯ, ии) =
185                                                        ОТВЕТ_ДА) ТО
186                   (** К_СХЕМА_ВЫВОД_ЭЛЕМЕНТА  и  УВЕЛИЧИТЬ (и);    |V| *)
187                   данные (Д_СХЕМА_ВЕРТИКАЛЬ,
188                          значение (В_СХЕМА_ДОП_ВЕРТИКАЛЬ, ж));
189                   данные (Д_СХЕМА_ИНДЕКС_ЭЛЕМЕНТА, ии);
190                   данные (Д_СХЕМА_ТЕК_ЭЛ_ВЕРТИКАЛИ, и);
191                   команда (К_СХЕМА_ВЫВОД_ЭЛЕМЕНТА);
192
193                   УВЕЛИЧИТЬ (и);
194                КОНЕЦ;
195             КОНЕЦ;
196             УВЕЛИЧИТЬ(ии);
197          КОНЕЦ_ЦИКЛА;
198       ИНЕСЛИ (ветка = 120) ТО
199          УВЕЛИЧИТЬ(ж);
200          ЕСЛИ (ж < запрос(З_СХЕМА_КОЛ_ДОП_ВЕРТИКАЛЕЙ)) ТО
201             ветка := 50;
```

```
202           ИНАЧЕ
203              команда(К_СХЕМА_ОБНОВИТЬ_КАНВУ);
204              ветка := 0;
205           КОНЕЦ;
206        ИНАЧЕ
207        КОНЕЦ;
208     КОНЕЦ_ЦИКЛА
209     (** ВЫХОД  *)
210     КОНЕЦ  вывод_схемы;     (*[-]*)
211  (** КОНЕЦ ПРОЦЕДУРЫ *)
```

Ниже, на рисунках 1.4 – 1.16 приведена разбитая на ветки схема, соответствующая тексту процедуры `вывод_схемы` (в связи с тем, что pdf-файлы для 1-й и 2-й частей настоящего материала были сгенерированы с минимальным разрешением «публикация в сети Интернет», качество большинства рисунков 1.4 – 1.15 и 2.12 – 2.18 является неудовлетворительным с точки зрения разборчивости текста в блоках для полной формы отображения схемы, но этого и не требуется, так как вышеуказанные рисунки приведены для общего представления, так сказать «взгляд сквозь мутное стекло»).

Размер шрифта приведенного выше текста процедуры `вывод_схемы` примерно соответствует размеру шрифта текста в окне текстового редактора одной из современных сред программирования. Фрагмент окна ввода текста элемента схемы в графическом редакторе ДАЛВЯЗ 2 приведен на рис. 1.17. Из сравнения рисунка 1.17 и приведенного выше текста процедуры видно, что в окне ввода текста элемента схемы графического редактора ДАЛВЯЗ 2 шрифт крупнее, чем в окне текстового редактора среды программирования, а значит и глаза у программиста при вводе текста для элемента схемы при работе в графическом редакторе ДАЛВЯЗ 2 будут напрягаться меньше, чем при работе в текстовом редакторе среды программирования.

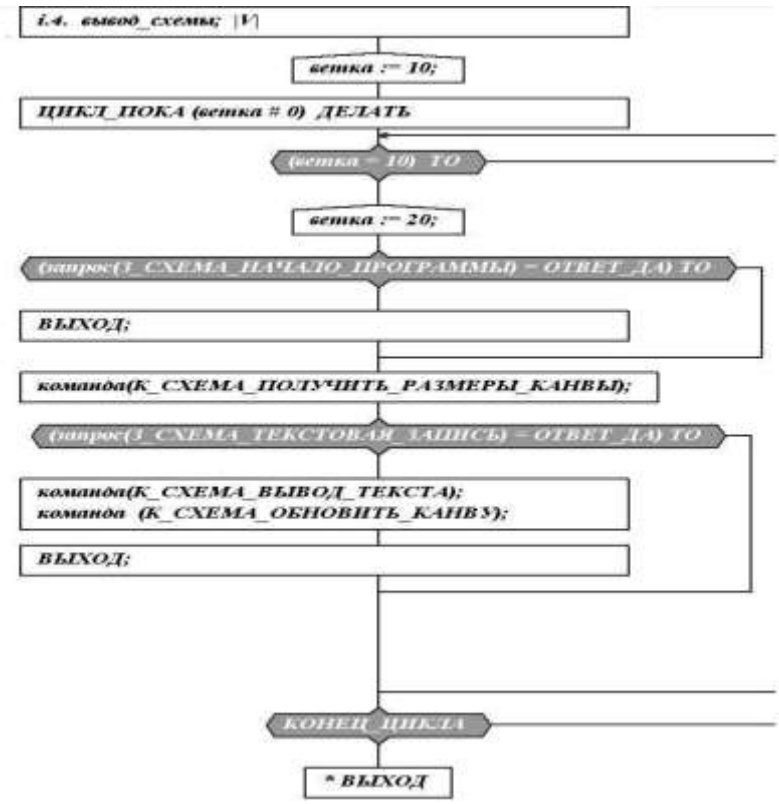

Рисунок 1.4 – Ветка 10 схемы

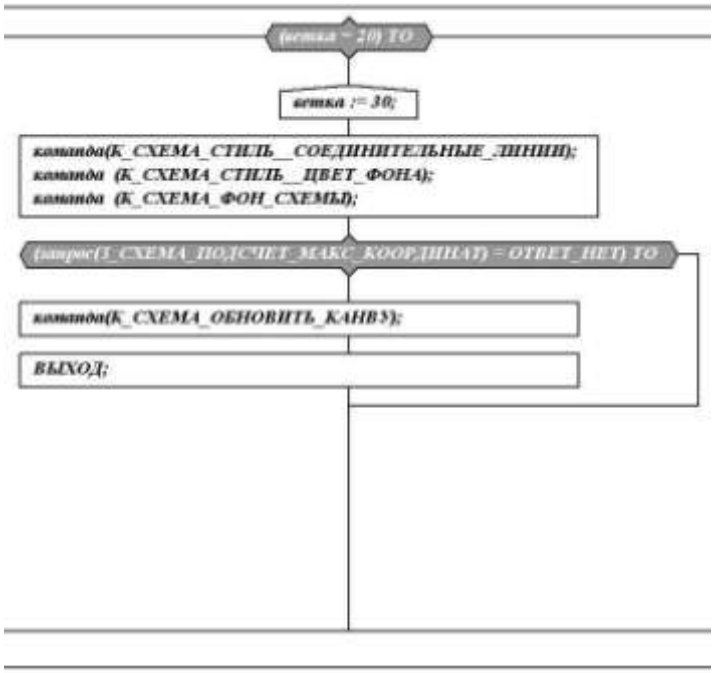

Рисунок 1.5 – Ветка 20 схемы

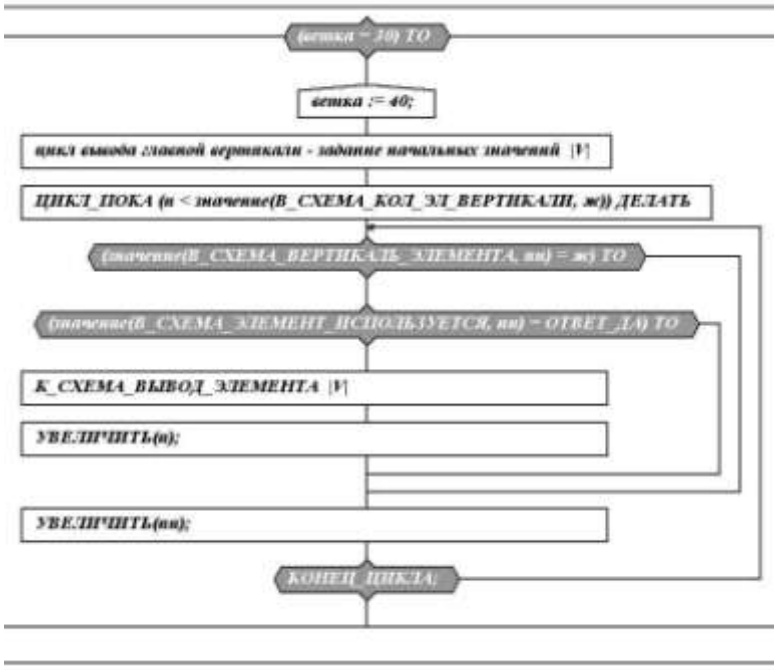

Рисунок 1.6 – Ветка 30 схемы

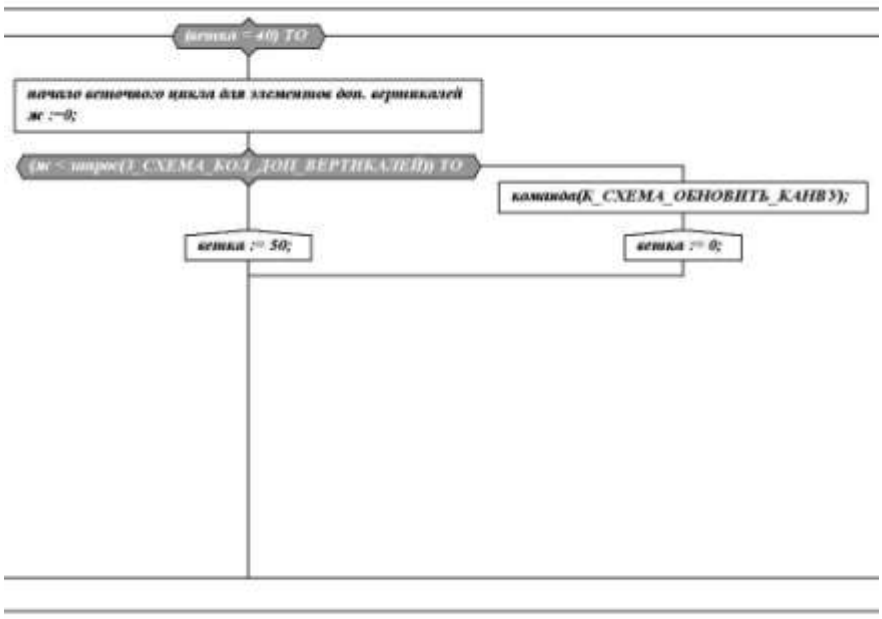

Рисунок 1.7 – Ветка 40 схемы

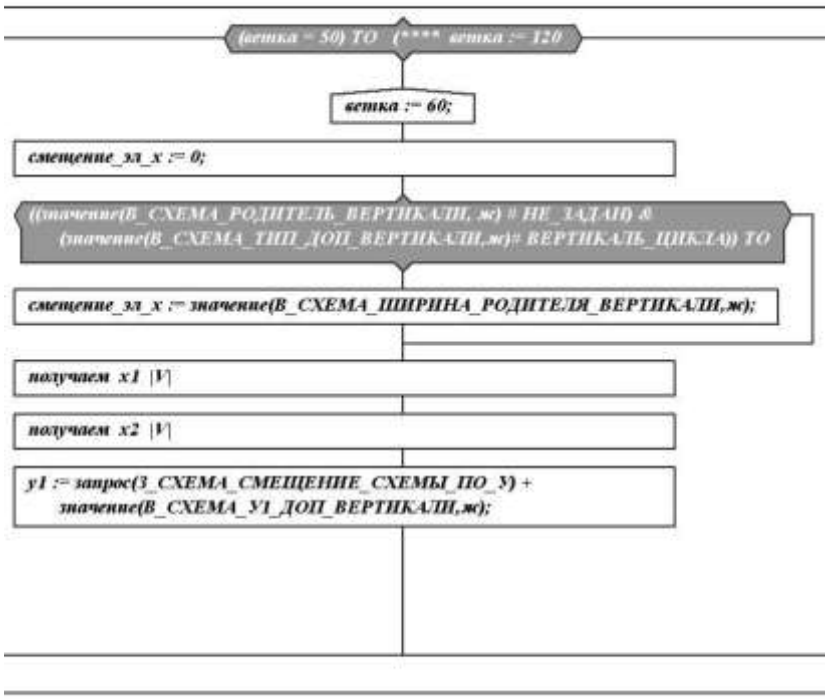

Рисунок 1.8 – Ветка 50 схемы

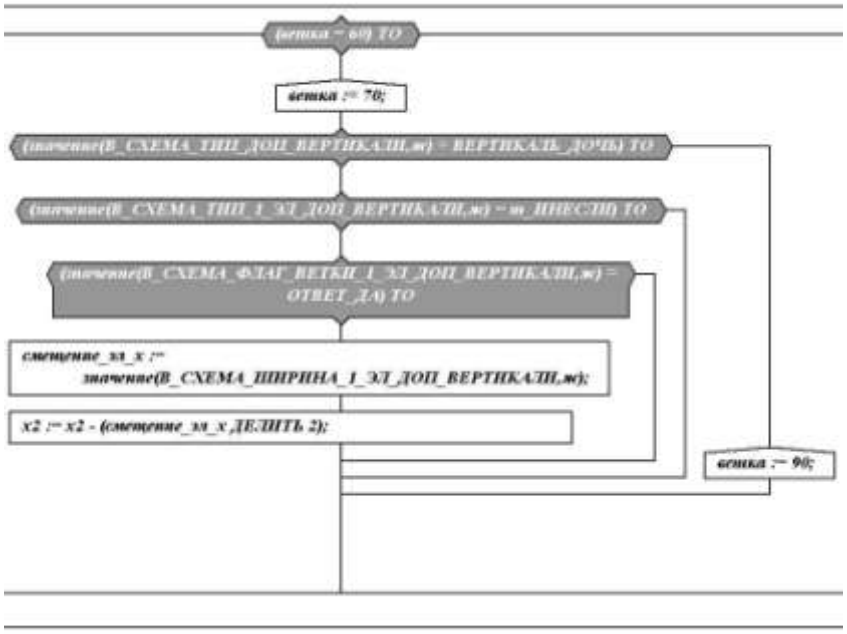

Рисунок 1.9 – Ветка 60 схемы

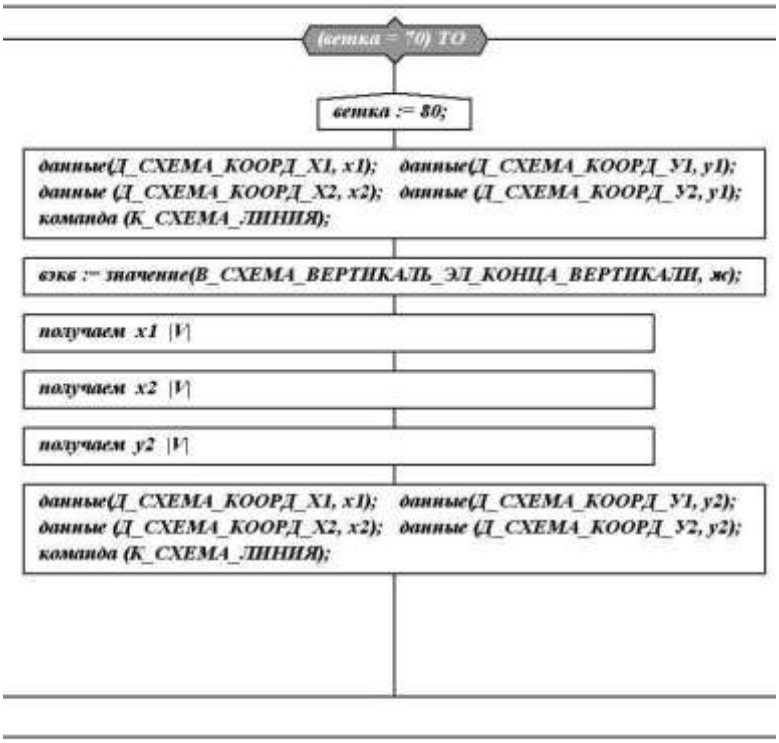

Рисунок 1.10 – Ветка 70 схемы

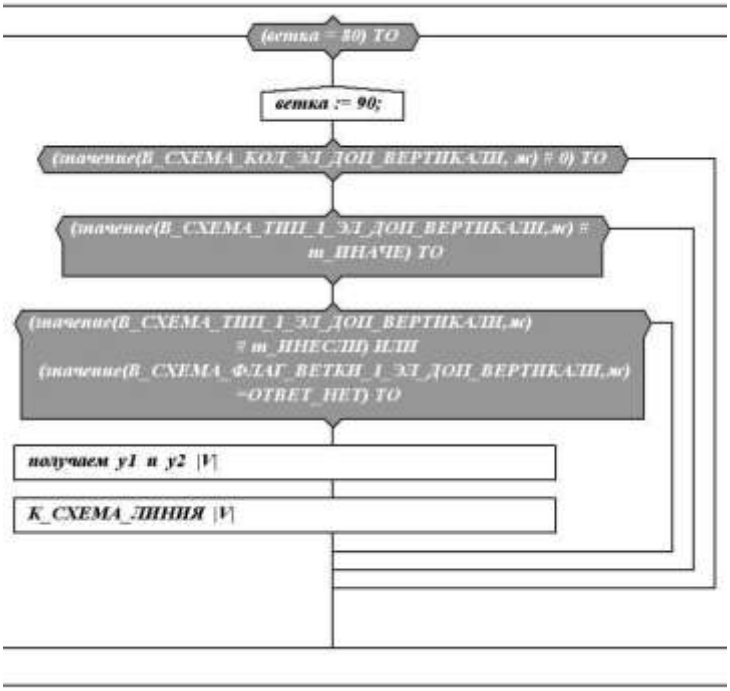

Рисунок 1.11 – Ветка 80 схемы

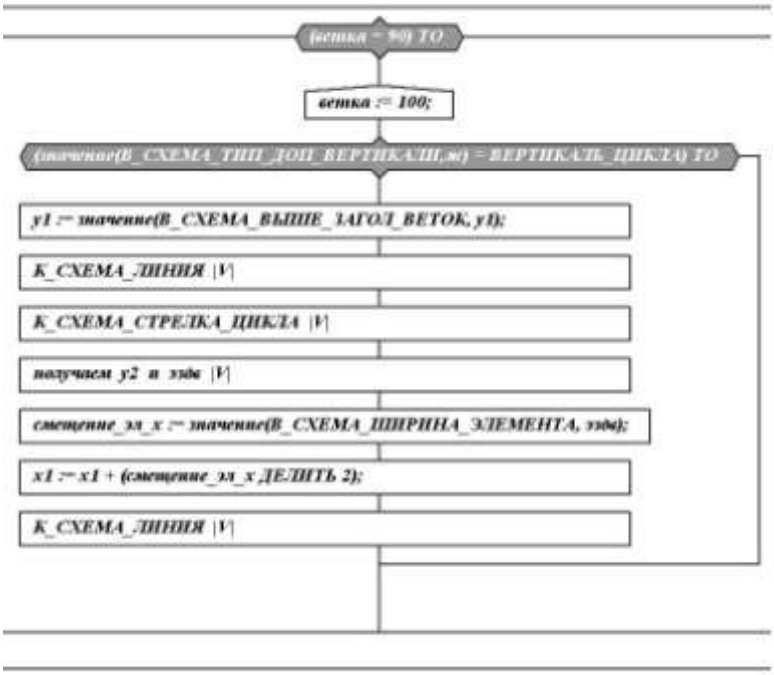

Рисунок 1.12 – Ветка 90 схемы

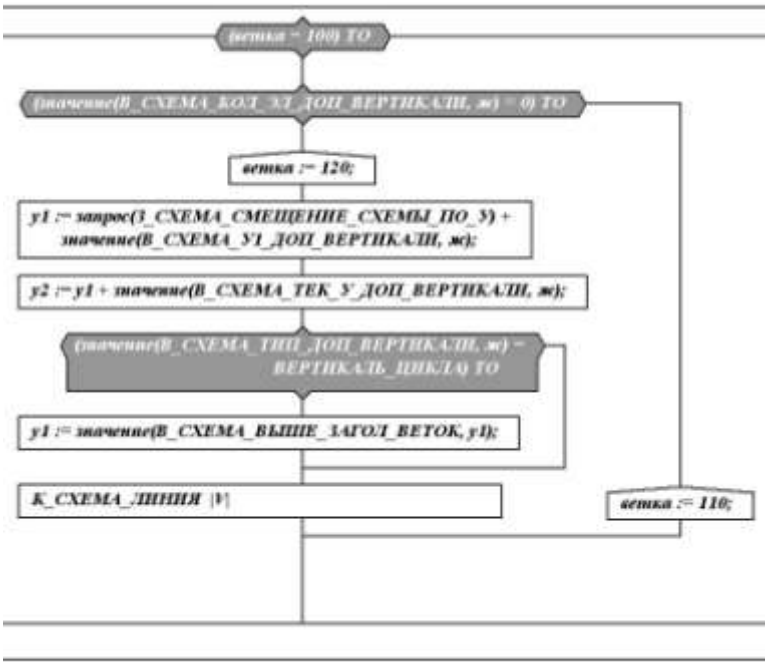

Рисунок 1.13 – Ветка 100 схемы

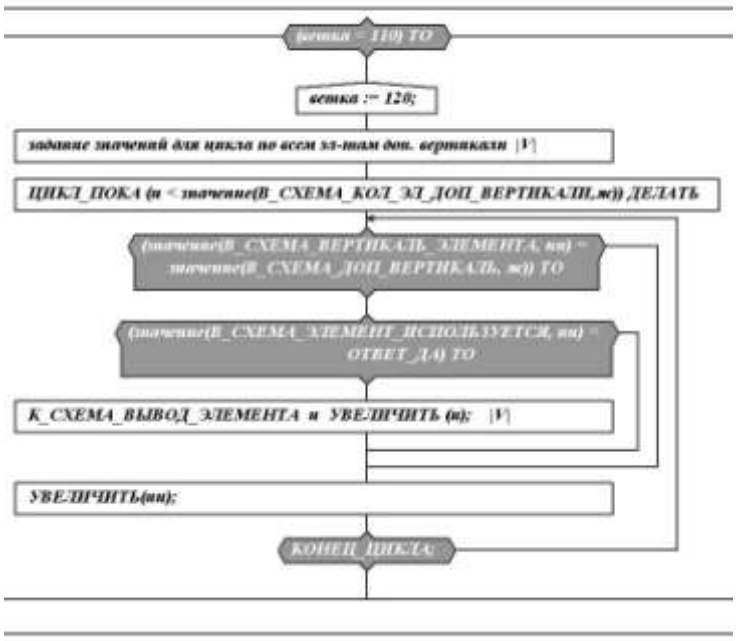

Рисунок 1.14 – Ветка 110 схемы

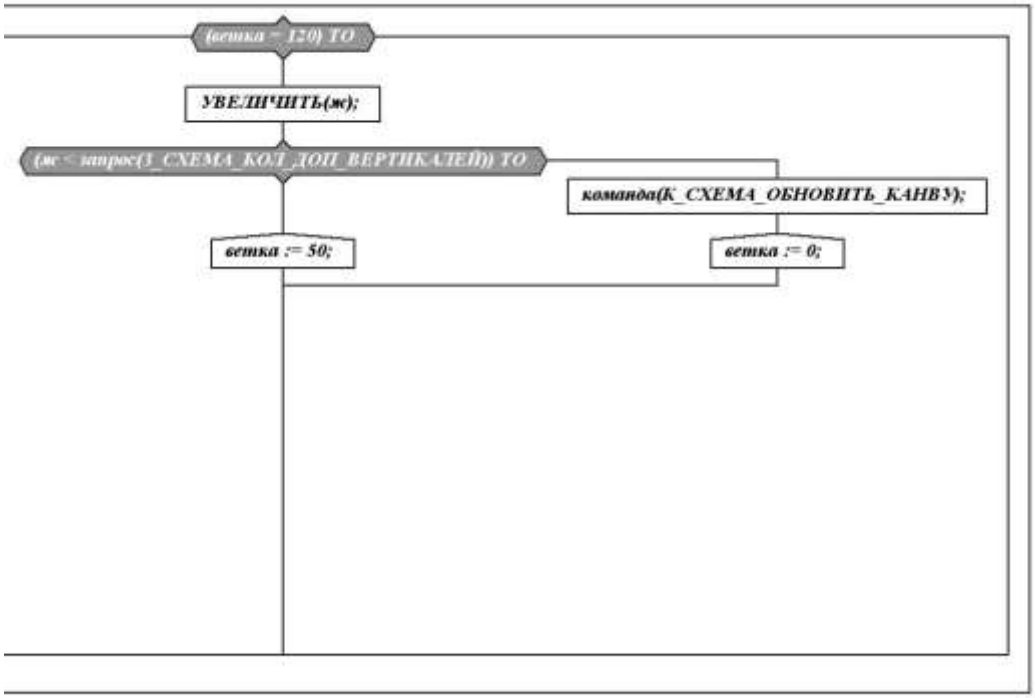

Рисунок 1.15 – Ветка 120 схемы

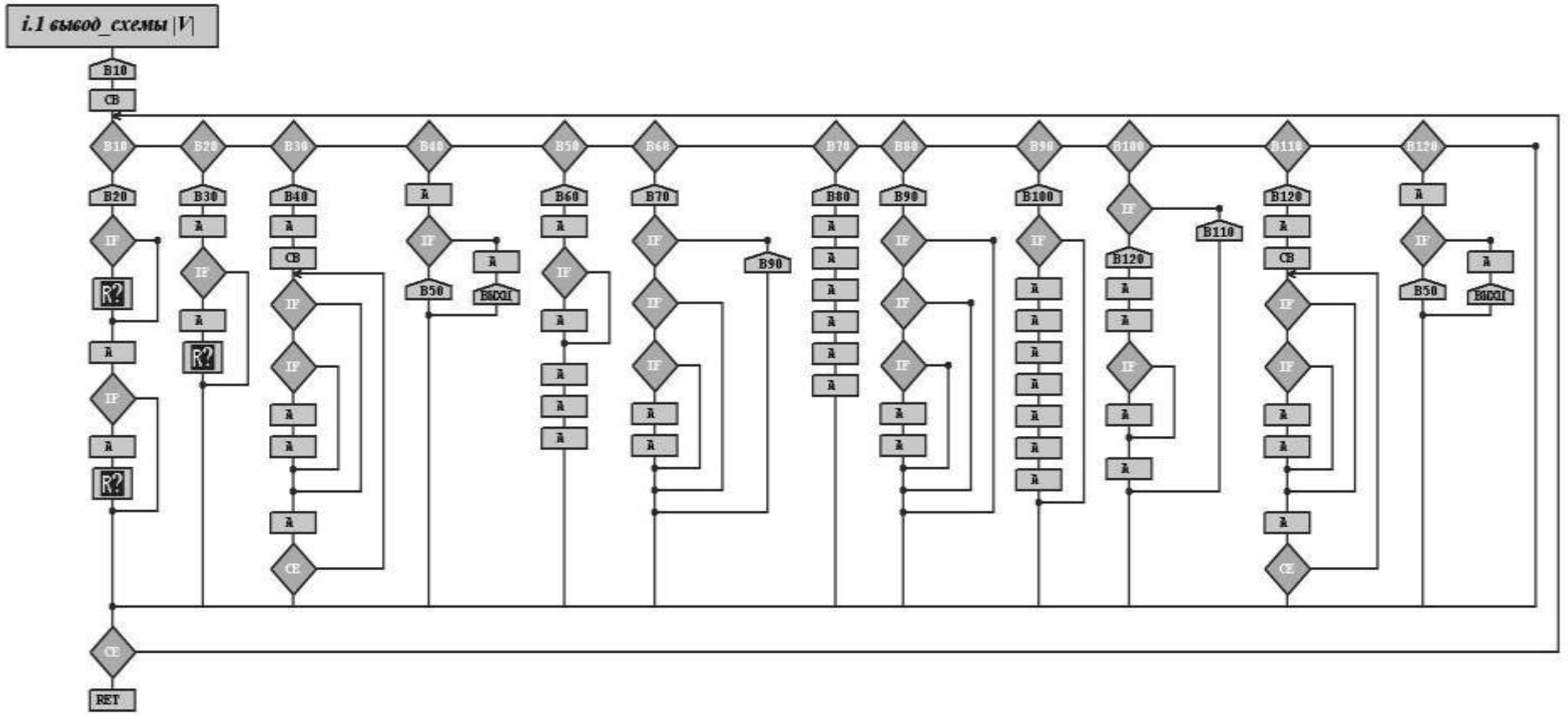

Рисунок 1.16 – Минимизированная схема для процедуры `вывод_схемы`

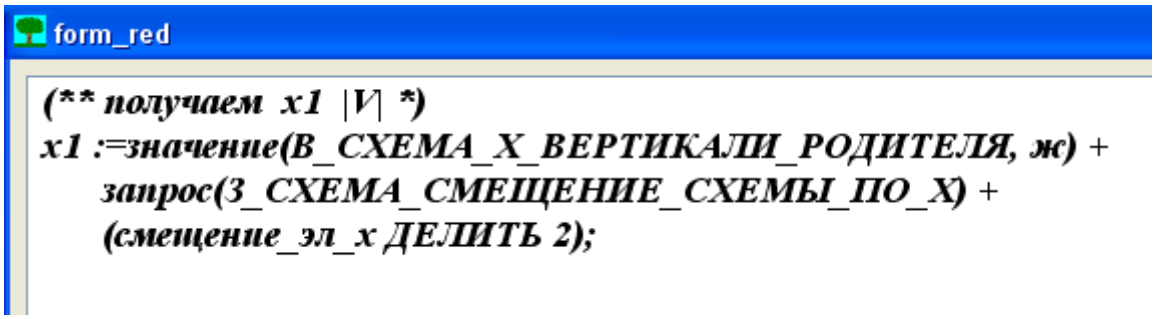

Рисунок 1.17 – Фрагмент окна ввода текста элемента схемы

Можно сказать, что любая программа-редактор блок-схем выступает в роли своеобразной экранной лупы по сравнению с текстовым редактором, т.к. попытка увеличить размер шрифта текстового редактора приводит к тому, что на экране отображается меньший фрагмент процедуры, а это ведет к тому, что программисту становится труднее анализировать всю процедуру в целом. Поэтому применяемые в настоящее время в средах программирования размеры шрифтов являются неким компромиссом между возможностью видеть на экране достаточно большой фрагмент процедуры и при этом не слишком напрягать глаза. Компромисс, надо сказать, не слишком полезный для глаз – как минимум у половины моих знакомых программистов, да и у меня тоже, имеется близорукость.

1.7. Панель инструментов для работы со схемой

Панель инструментов для работы со схемой приведена на рис. 1.18 и включает в себя следующие элементы:

Рисунок 1.18 – Панель редактирования схемы ДАЛВЯЗ 2

- кнопка «Счит.» – открывает форму с полем ввода текста схемы, куда надо методом копирования в вставки перенести из текстового редактора среды программирования редактируемую процедуру, а затем нажать кнопку «Сохранить» формы с полем ввода текста схемы. Если при нажатии кнопки «Сохранить» поле ввода текста схемы останется пустым, то будет начато редактирование новой схемы;

- кнопка «Сохр.» – при ее нажатии текст отредактированной схемы автоматически (форма с отредактированным текстом процедуры при этом не открывается) копируется в клипборд Windows, откуда этот текст можно вставить (путем нажатия клавиш Ctrl+V или Shift+Insert) в текстовый редактор среды программирования; кроме того, текст отредактированной схемы копируется в файл proc.txt, находящийся в рабочем каталоге графического редактора ДАЛВЯЗ 2;

- флажок «схема» – переключение между графическим и текстовым форматами отображения процедуры;

- флажок «Кружки» – отображение на схеме кружками элементов «* иначе» и «* конец»; если флажок «Кружки» сброшен, то элементы «* иначе» и «* конец» на схеме в явном виде не отображаются;

- флажок «мин» – переключение между минимальной и полной формами отображения схемы;

- флажок «цвет» – переключение между цветным и черно-белым отображением схемы;

- флажок «прозр» – включение/отключение прозрачности блоков; если прозрачность блоков включена, то текст блоков будет отображаться на фоне схемы или рисунка-подложки;

- флажок «рис. фона» – включение/отключение рисунка-подложки.

Переключение флажков «цвет» – «рис. фона» для минимизированной формы отображения схемы приводит к результатам, показанным на рис. 1.19. Переключение вышеуказанных флажков для полной формы отображения схемы в смысле изменения стиля отображения схемы выглядит аналогично.

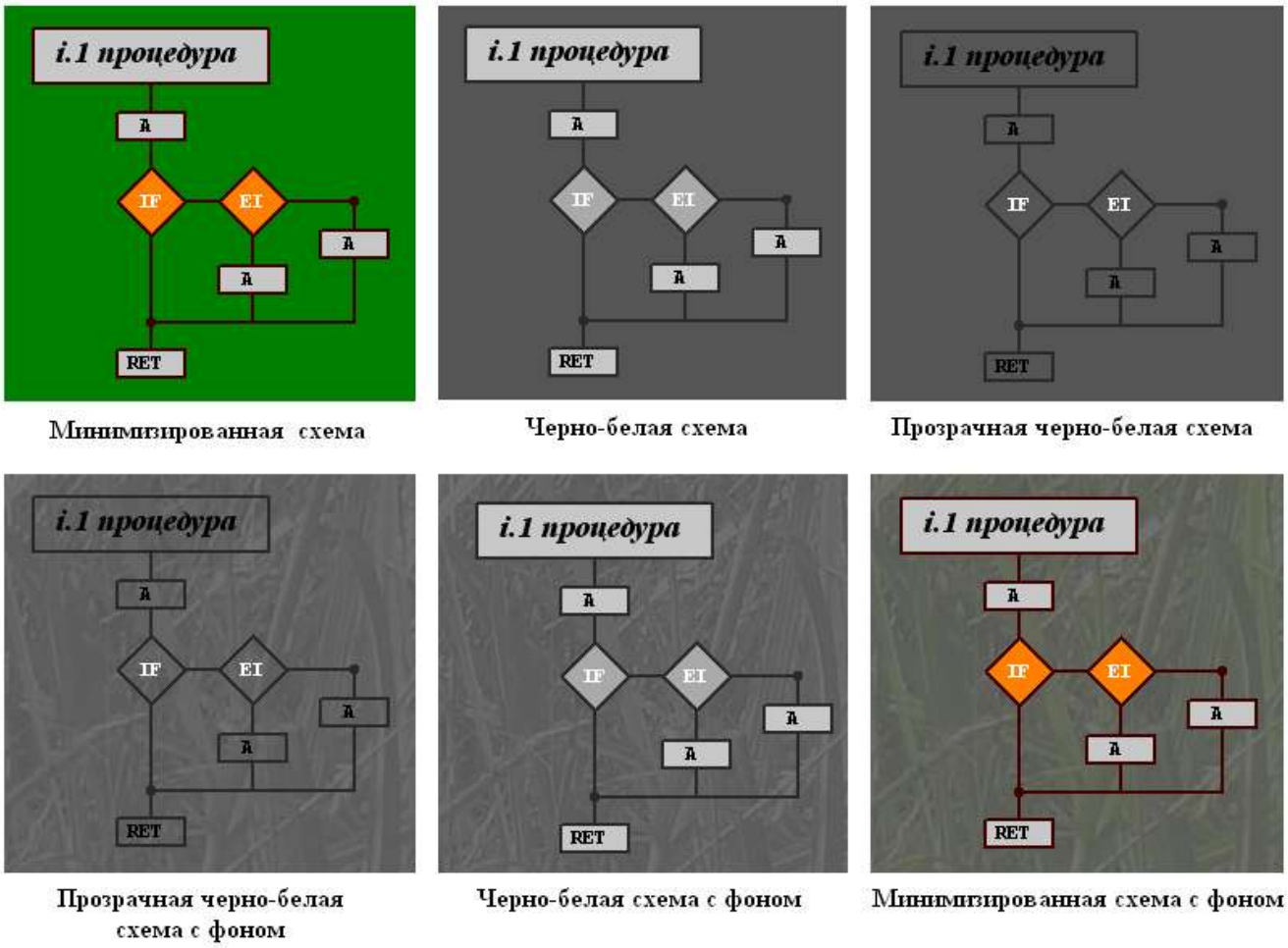

Рисунок 1.19 – Виды отображения минимизированной схемы в зависимости от состояния флажков «цвет», «прозр», «рис. фона».

Изменение время от времени стиля отображения схемы путем переключения состояний флажков «цвет», «прозр», «рис. фона» на панели инструментов графического редактора ДАЛВЯЗ 2 помогает бороться с монотонностью работы по вводу текста исходного кода программы в графическом редакторе ДАЛВЯЗ 2.

Описание инструментов редактирования схемы приводится в следующем пункте.

1.8. Редактирование схемы в ДАЛВЯЗ 2

Ну ладно, пусть редактор блок-схем позволяет вводить текст более крупным шрифтом, чем текстовый редактор среды программирования, и при этом меньше напрягаются глаза. Но это преимущество напрочь перечеркивается неудобством редактирования схемы. Надо двигать блоки элементов, чтобы получше их разместить на схеме, надо устанавливать связи между элементами – в общем сплошная морока, и настоящий программист этим заниматься не будет.

Рассмотрим, как вопрос редактирования схемы решается в ДАЛВЯЗ 2.

В ДАЛВЯЗ 2 определены следующие инструменты редактирования схемы:

- кнопка  – начать вставку элемента «* действие»; чтобы вставить в схему элемент «* действие», нужно после нажатия кнопки подвести указатель мыши к нужному элементу на схеме и нажать левую кнопку мыши, после чего элемент «* действие» будет вставлен в схему ниже выбранного элемента схемы;

- кнопка  – начать вставку элемента «* если»; чтобы вставить в схему элемент «* если», нужно после нажатия кнопки подвести указатель мыши к нужному элементу на схеме и нажать левую кнопку мыши, после чего элемент «* если» будет вставлен в схему ниже выбранного элемента схемы;

- кнопка  – начать вставку элемента «* иначе если»; чтобы вставить в схему элемент «* иначе если», нужно после нажатия кнопки подвести указатель мыши к нужному элементу «* если» или «* иначе если» на схеме и нажать левую кнопку мыши, после чего элемент «* иначе если» будет вставлен в схему правее выбранного элемента схемы;

- кнопка  – начать вставку элемента «* иначе»; чтобы вставить в схему элемент «* иначе», нужно после нажатия кнопки подвести указатель мыши к нужному элементу «* если» или «* иначе если» на схеме и нажать левую кнопку мыши, после чего элемент «* иначе» будет вставлен в схему правее выбранного элемента схемы;

- кнопка  – начать вставку элементов «* цикл» и «* конец цикла»; чтобы вставить в схему элементы «* цикл» и «* конец цикла», нужно после нажатия кнопки подвести указатель мыши к нужному элементу на схеме и нажать левую кнопку мыши, после чего элементы «* цикл» и «* конец цикла» будут вставлены в схему ниже выбранного элемента схемы;

- кнопка  – вырезать выделенный фрагмент схемы в буфер хранения; при нажатой клавише Ctrl движение указателя мыши над элементом вызывает выделение этого элемента; выделение элементов прекращается при отпускании клавиши Ctrl и является структурным: если был выделен блок начала условия, то будут выделены все элементы этого условия; если был выделен блок начала ветки условия, то будут выделены все элементы ветки условия; если был выделен блок начала цикла, то будут выделены все элементы этого цикла; элементы «* иначе», «* конец» и «* конец цикла» выделяются только после отпускания клавиши Ctrl;

- кнопка  – копировать выделенный фрагмент схемы в буфер копирования;

- кнопка  – начать вставку из буфера хранения; чтобы вставить в схему содержимое буфера хранения, нужно после нажатия кнопки подвести указатель мыши к нужному элементу на

схеме и нажать левую кнопку мыши, после чего содержимое буфера хранения будет вставлено в схему ниже (правее, если вставляется ветка условия) выбранного элемента схемы;

- кнопка  – начать вставку из буфера копирования; чтобы вставить в схему содержимое буфера копирования, нужно после нажатия кнопки подвести указатель мыши к нужному элементу на схеме и нажать левую кнопку мыши, после чего содержимое буфера копирования будет вставлено в схему ниже (правее, если вставляется ветка условия) выбранного элемента схемы;

- кнопка  – отмена последнего внесенного в схему изменения;

- кнопка «шаг 1» – нормализация номеров веток условия с шагом 1;

- кнопка «шаг 10» – нормализация номеров веток условия с шагом 10.

Из описания приведенных выше инструментов редактирования схемы видно, что в ДАЛВЯЗ 2 при редактировании схемы нет необходимости вручную задавать взаимное расположение и размеры элементов схемы – после добавления или удаления элементов взаимное расположение и размеры элементов схемы пересчитываются автоматически.

1.9. Работа с русифицированной программной логикой верхнего уровня в ДАЛВЯЗ 2

Процедуры программы с точки зрения решаемых ими задач можно разделить на две группы:

- процедуры логики нижнего уровня, управляющие каким-либо оборудованием или обрабатывающие внешнюю для программы информацию;

- процедуры логики верхнего уровня (ЛВУ), вызывающие процедуры логики нижнего уровня в зависимости от текущего состояния программы, сформированного путем опроса процедур логики нижнего уровня.

Процедура, текст которой приведен в пункте 1.6, является процедурой логики верхнего уровня, причем написанной целиком по-русски. В ней используются только две латинские буквы: i при задании заголовка процедуры "(* i." и V в параметре |V| маскирования всех строк элемента, кроме первой.

В смысле реализации полной русскоязычности эта процедура является достаточно нетипичной, потому что даже такие языки программирования, как Компонентный паскаль, дающие программисту возможность русификации ключевых слов языка программирования (а большинство современных сред программирования дают программистам возможность по-русски писать только идентификаторы, а ключевые слова остаются английскими) не могут обойтись без вызова системных модулей, исходный код которых написан по-английски.

Чтобы обойти этот барьер на пути к полной русификации программной логики верхнего уровня, пришлось воспользоваться подстановочными процедурными вызовами «команда», «данные», «запрос», «значение», для которых в качестве параметров задавались русскоязычные константы и переменные. Такой набор подстановочных процедур был выбран потому, что он оказался достаточным для реализации логики рассматриваемой процедуры ЛВУ, обрабатывающей данные одномерных массивов.

Исходный код подстановочных процедур и вызываемого из них обработчика логики нижнего уровня для языка Delphi 2010 выглядит следующим образом:

```
procedure  команда(k:integer);  // посылка команды
var ot:integer;
begin
   ot := obrabotka(k,0);
end;

procedure  данные(k1, k2:integer); // посылка данных
var ot:integer;
begin
   ot := obrabotka(k1,k2);
end;

function   запрос(n:integer):integer;  // получение запроса
var ot:integer;
begin
   ot := obrabotka(n,0);
   запрос := ot;
end;

function   значение(n1,n2:integer):integer;  // получении запроса о состоянии
                                             // элемента некоторого массива
var ot:integer;
begin
   ot := obrabotka(n1,n2);
   значение := ot;
end;


function  obrabotka(n1, n2:integer):integer;
var ot:integer;
begin
   ot := -1;

   if (n1 >= 10001) and (n1 < 15000) then begin
      (* обработка_вывода_схемы *)          ot := obrabotka_СХЕМА(n1,n2);
   end else
   if (n1 >= 15001) and (n1 < 20000) then begin
      (* обработка логической структуры *)  ot := obrabotka_ЛОГ_СТРУКТ(n1,n2);
   end;
   obrabotka := ot;
end;


function  obrabotka_СХЕМА(n1, n2:integer):integer;
```

```
var ot:integer;
   ...
begin
   ot := -1;

   // обработка команд
   if (n1= К_СХЕМА_ПОЛУЧИТЬ_РАЗМЕРЫ_КАНВЫ) then begin
      Bitmap.Width := form_dalvjaz.Pwidth;
      Bitmap.Height := form_dalvjaz.PHeight;
   end else if (n1= К_СХЕМА_ВЫВОД_ТЕКСТА) then begin
      ...

   // обработка данных
   end else if (n1= Д_СХЕМА_ВЕРТИКАЛЬ) then begin
      dannye_wertikalx := n2;
   end else if (n1= Д_СХЕМА_ИНДЕКС_ЭЛЕМЕНТА) then begin
      ...

   // обработка запросов
   end else if (n1= З_СХЕМА_НАЧАЛО_ПРОГРАММЫ) then begin
      if (dalvjaz_beg = 1) then  ot := YES
      else                       ot := NO;
   end else if (n1= З_СХЕМА_ТЕКСТОВАЯ_ЗАПИСЬ) then begin
      ...

   // обработка значений
   end else if (n1= В_СХЕМА_ИНД_1_ЭЛ_ВЕРТИКАЛИ) then begin
      ot := tek_alg.m_inew[n2];
   end else if (n1= В_СХЕМА_КОЛ_ЭЛ_ВЕРТИКАЛИ) then begin
      ...

   end;

   obrabotka_СХЕМА := ot;
end;
```

Каждому обработчику, например obrabotka_СХЕМА, могут соответствовать несколько процедур ЛВУ.

Каждая ветка сложного условия обработчика может содержать либо вызов процедуры логики нижнего уровня, либо анализ или обработку каких-либо данных с получением соответствующего выходного значения для передачи его в процедуру ЛВУ.

Ветви сложного условия обработчика сгруппированы по типам обрабатываемых процедурных вызовов в следующей последовательности:

- обработка команд;
- обработка данных;
- обработка запросов;
- обработка значений.

Русскоязычные константы, передаваемые в подстановочные процедуры, состоят из трех частей:

- префикс: К_ (команда) / Д_ (данные) / З_ (запрос) / В_ (значение);
- идентификатор обработчика (например СХЕМА_);
- идентификатор выполняемого действия.

Пример константы, передаваемой в подстановочную процедуру:
З_СХЕМА_НАЧАЛО_ПРОГРАММЫ – запрос признака начала программы.

Для работы с процедурами ЛВУ в ДАЛВЯЗ 2 введен параметр «альтернативные начала действия», который для приведенной в пункте 1.6 процедуры, написанной на Компонентном паскале, имеет следующее значение:

"команда(|данные(| := |УВЕЛИЧИТЬ(|УМЕНЬШИТЬ("

В приведенной строке признаки альтернативных начал действия разделены символом «|».

Альтернативное начало действия должно присутствовать в строке, но не обязано находиться в ее начале в отличие от основных служебных комментариев ДАЛВЯЗ 2.

Чтобы несколько строк с присваиваниями воспринимались как одно действие, нужно начиная со второй строки действия выполнять присваивание без пробела между переменной и значением:

```
Переменная1 := значение1;   (* начало действия *)

Переменная2 :=значение2;  (* эта строка не считается альтернативным началом
                             действия *)
```

Аналогично и для других альтернативных начал действия, например для «данные(»:

```
данные(….     (* начало действия *)
данные  (…    (* эта строка не считается альтернативным началом действия *)
```

Использование вышеприведенного формата русифицированной ЛВУ с применением вызовов подстановочных процедур позволяет решать следующие задачи:

1) Обеспечение самодокументирования процедур ЛВУ

Многие пренебрежительно относятся к документированию своего кода. А между тем в англоязычных странах, где ПО пишется на родном языке, программистам легче, чем нам, разбираться в чужом коде и подхватывать начатые другими программистами программные проекты. Отсюда меньше переписываемых с нуля проектов, а значит и материальные издержки на разработку ПО тоже меньше.

2) Обеспечение механизма сокрытия данных путем разбиения логики программы на маршрутную (процедура ЛВУ) и функциональную (обработчик процедуры ЛВУ)

Механизм сокрытия данных на уровне ЛВУ обеспечивает гибкость разработки логики процедур ЛВУ, сопоставимую с гибкостью разработки программ, обеспечиваемой механизмом

сокрытия данных через интерфейсы модулей и классов. Например, при замене типов данных, используемых в обработчике процедуры ЛВУ, для самой процедуры ЛВУ ничего не изменится.

3) Разбиение функциональной логики программы на мелкие фрагменты с полностью прозрачной логикой, находящиеся в ветках сложных условий обработчиков процедур ЛВУ.

4) Формирование унифицированной трехуровневой логики разработки программы

При программировании с использованием ЛВУ логика программы разделяется на следующие логические уровни:

- процедуры ЛВУ для обработки возникающих событий;
- обработчики процедур ЛВУ;
- процедуры логики нижнего уровня.

Введение унифицированной логики разработки программы (шаблона разработки ПО) позволит упростить передачу сопровождения фрагментов программы другим программистам за счет упрощения ознакомления с унифицированной русифицированной ЛВУ того или иного фрагмента программы по сравнению с общепринятой логикой написания ПО с минимально возможным документированием исходного кода программы и размытой от постоянного внесения новых изменений логической структурой программы.