ЧАСТЬ 2. ПРАКТИКА

2.1. Файл конфигурации программы dal2.cfg

Файл конфигурации программы dal2.cfg позволяет задавать параметры форматирования и служебные комментарии для редактируемой процедуры.

Ниже приведен файл конфигурации для работы с русифицированной версией языка Компонентный паскаль.

```
;
;         ФАЙЛ  КОНФИГУРАЦИИ  ПРОГРАММЫ  dal_vjaz_2  v.087.4
;
;=============================================================================
; параметры конфигурации
;
;                        " " или "значение"
;
__шаг_уровня_структуры   "3"  ; число пробелов для шага табуляции
__отступ_слева           "3"    ; для 1-го действия после заголовка
;
__загол_прог_записи_исх_кода      "(* i."
;
__опред_рис_стиля        " РИС "   ; определитель рисунка и стиля
;
__прг_строка_док_IF        "ЕСЛИ"                ; начало условия
__прг_строка_док_EI        "ИНЕСЛИ"              ; дополнительное условие
__прг_строка_док_EB        "ИНАЧЕ"               ; альтернатива условиям
__прг_строка_док_END       "КОНЕЦ;"              ; конец условия
__прг_строка_док_CB        "ЦИКЛ"
__прг_строка_док_CE        "КОНЕЦ_ЦИКЛА"
__прг_строка_док_A         "(**"                 ; основное начало действия
__прг_строка_док_RET       "(** ВЫХОД"           ; оператор возврата,
;                                                ; возможно с параметром
__прг_строка_док_STOP      "(** КОНЕЦ ПРОЦЕДУРЫ"
__прг_проверка_ветки       "ветка = "            ; в условии
__прг_переход_к_ветке      "ветка := "           ; путем присваивания
;                                                ; нового значения
;
;
__альтернативные_начала_действия "команда(|данные(| := |УВЕЛИЧИТЬ(|УМЕНЬШИТЬ("  ;
;
__подавление_ключевого_слова_в_строке    "(*--*)"
;
__маскировать_строку_при_выводе          "(*[-]*)"
;
__маскировать_все_строки_кроме_1-й       "|V|"
;
__открывающий_коммент    "(*"
__закрывающий_коммент    "*)"
;
__коэф_выравн_блоков     "85"   ; коэффициент выравнивания блоков вертикали
;                                 по левой границе (0, от 65 до 100)
;
; отступы схемы от границ листа схемы, задают размеры рамку пустого
; пространства вокруг схемы на листе схемы
;
;
__отступ_схемы_слева     "50"
__отступ_схемы_справа    "50"
__отступ_схемы_сверху    "50"
__отступ_схемы_снизу     "50"
```

```
;
__диапазон_прокрутки     "300"  ; увеличение этого числа увеличивает
;                                 плавность прокрутки, а также сдвига
;                                 схемы колесиком мыши или нажатой
;                                 правой кнопкой мыши для больших схем
;
;-------------------------------------------------------------
.
```

Работа с основными параметрами файла конфигурации dal2.cfg уже была рассмотрена в Части 1.

Параметры «шаг уровня структуры» и «отступ слева» задают параметры форматирования текста отредактированной процедуры.

Параметр «коэф_выравн_блоков» управляет выравниванием по левой границе элементов типа «* действие», лежащих на одной вертикали. Если коэф_выравн_блоков = 0, то выравнивания блоков действий по левой границе не происходит. Если коэф_выравн_блоков = 100, то ширина всех блоков действий, лежащих на одной вертикали, является одинаковой.

Назначение параметров «отступ_схемы …» и «диапазон_прокрутки» описано в тексте файла конфигурации dal2.cfg.

2.2. Файл стилей отображения схемы styles.cfg

Файл стилей отображения схемы styles.cfg позволяет менять цвета отображения окон программы и по умолчанию выглядит следующим образом:

```
;
;  ФАЙЛ СТИЛЕЙ ПРОГРАММЫ  dal_vjaz_2  0.87.4
;
;=============================================================================================
;  СЕКЦИЯ "Цвета"
;
;  Определение наборов цветов для стилей
;  цвета задаются тройками цифр от 0 до 255 по порядку:
;  красный, зеленый, синий
;
;  Набор цветов "_Цвета_1" действует по умолчанию
;
;  пока возможно задание до 50 наборов цветов
;
;  обозначения:
;
;  ЦТБ    - цвет текста блока
;  ЦКБ    - цвет контура блока (сейчас совпадает с ЦСЛС)
;  ЦФБ    - цвет фона блока
;           служебные значения:
;           300 300 300  - выводятся только контур и текст блока, а фон прозрачный
;           400 400 400  - выводится только текст блока на фоне схемы или на фоне
;                          рисунка-подложки, если он задан для схемы
;
;  ЦНСБ   - цвет для начальной строки блока (сейчас не используется)
;  ЦФС    - цвет фона схемы
;  ЦСЛС   - цвет соединительных линий схемы
;
;        N       ЦТБ          ЦКБ          ЦФБ          ЦНСБ         ЦФС          ЦСЛС
;
;            *** *** ***  *** *** ***  *** *** ***  *** *** ***  *** *** ***  *** *** ***
;
__Цвета "1   000 000 000  064 000 000  200 200 200  000 000 000  000 128 000  064 000 000"
;
__Цвета "2   255 255 255  064 000 000  255 128 000  000 000 000  255 255 255  255 255 000"
;
; цвета 2 задают цвета текста и фона для условий и цвет контура выделенного блока (ЦФС)
```

```
;
__Цвета "3   000 000 000  064 000 000  200 200 200  000 000 000  000 128 000  064 000 000"
;
; цвета 3 задают цвета текста и фона для окна редактирования текста элемента схемы
;
__Цвета "4   000 000 000  000 100 000  000 200 200  000 000 000  200 200 200  000 100 000"
;
__Цвета "5   255 255 000  000 255 255  000 000 255  000 000 000  000 150 000  000 255 255"
;
__Цвета "6   255 255 000  000 255 255  000 000 255  000 000 000  000 150 000  000 255 255"
;
__Цвета "7   000 255 000  000 255 255  000 150 000  000 000 000  000 150 000  000 255 255"
;
__Цвета "8   000 255 000  000 255 255  000 150 000  000 000 000  000 000 255  000 255 255"
;
;=============================================================================================
;  СЕКЦИЯ "Фонты"
;
;  Фонт_1 действует по умолчанию
;  Фонт 2 используется для редактирования текста элемента схемы
;
;  пока возможно задание до 30 фонтов
;
;  для удобства считывания имя фонта заключается в одиночные кавычки
;
;  обозначения:
;
;   ЖКП     - жирный, курсив, подчеркивание:  каждый из параметров задается как 0 или 1
;  0 0 0    - это нормальный стиль
;
;        N       Имя                Стиль           Размер
;                                    ЖКП
;
__Фонт  "1   'Times New Roman'      1 1 0             14"
__Фонт  "2   'Times New Roman'      1 1 0             18"
__Фонт  "3   'Courier New'          0 0 0             9"
__Фонт  "4   'Book Antiqua'         1 1 0             14"
__Фонт  "5   'Courier New'          1 0 0             10"
;
;
;=============================================================================================
;  СЕКЦИЯ "Стили"
;
;  _Стиль_1 действует по умолчанию
;
;  пока возможно задание до 50 стилей
;
;  обозначения:
;
;  ННЦ    - номер набора цветов
;  НФТБ   - номер фонта текстов блоков схемы
;  РСТБ   - расстояние в пикселях между строками текста в блоках
;  НФНС   - номер фонта номеров строк блоков (сейчас не используется)
;  НРП    - трехзначный номер рисунка-подложки, например
;           000 - рисунок-подложка для схемы отсутствует
;  ТКБ    - толщина контуров блоков схемы (от 1 до 2), совпадает с ТСЛ
;  ТСЛ    - толщина соединительных линий (от 1 до 2)
;  НСЗС   - номер стиля для заголовка схемы
;
;  служебные стили:
;
;  1 - стиль по умолчанию для схемы, задает цвета блоков действий
;  2 - задает цвета блоков условий и цвет контура выделенного блока (ЦФС)
;  3 - задает стиль для окна редактирования элемента
;-------------------------------------------------------------
;
;        N     ННЦ   НФТБ  РСТБ  НФНС  НРП  ТКБ  ТСЛ  НСЗС
;
__Стиль "1      1     1     2     1   '001'  2    2    1"
;
```

```
__Стиль "2      2     1     0     1   '000'  1    2    5"
;
__Стиль "3      3     2     2     1   '000'  1    2    3"
;
; стиль 3 обращается к третьему набору цветов и использует для редактирования
; текста элемента более крупный шрифт, чем при выводе схемы
;
__Стиль "4      4     1     0     1   '001'  2    2    4"
;
__Стиль "5      5     3     0     1   '000'  1    2    5"
;
__Стиль "6      6     3     0     1   '000'  1    2    5"
;
__Стиль "7      7     3     0     1   '000'  1    2    5"
;
__Стиль "8      8     3     0     1   '000'  1    2    2"
;
;=============================================================================================
;
.
```

2.3. Возможные варианты служебных комментариев

Я долгое время не мог определиться в вопросе выбора наиболее предпочтительных, с точки зрения удобства их использования, служебных комментариев.

Первый вариант (ниже приведен первый вариант служебных комментариев для языка Паскаль) был предложен в материале «ДАЛВЯЗ 2 – описание» и, очевидно, он является не слишком удобным в использовании.

```
prg_otkr_kmnt       = "(*"
prg_zakr_kmnt       = "*) "
zagol_prg           = "(* i."
prg_t_IF            = "(*i*)  if "
prg_t_EI            = "(*i*)  end else if "
prg_t_EB            = "(*i*)  end else begin"
prg_t_E             = "(*i*)  end;"
prg_t_CB            = "(*i CB *)"        // начало цикла
prg_t_CE            = "(*i CE *)"        // конец цикла
prg_t_A             = "(*i*)"            // действие
prg_prow_vetki      = " if (wetka = "    // проверка ветки в условии
prg_k_vetke         = " wetka := "       // задание новой ветки
prg_t_RET           = "(*i ВЫХОД *)"
prg_STOP            = "(*i КОНЕЦ *)"
```

Толчком к следующему шагу послужило желание перелистать на досуге книгу С. Макконелла «Совершенный код». Глава 9 этой книги специально посвящена программированию с псевдокодом. С. Макконелл советует:

«* Избегайте синтаксических элементов языков программирования. Псевдокод позволяет проектировать на несколько более высоком уровне, чем код. Применяя конструкции языка программирования, вы мыслите на более низком уровне и теряете преимущества проектирования на высоком уровне, загружая себя ненужными синтаксическими ограничениями.

* Пишите псевдокод на уровне намерений. Описывайте назначение подхода, а не то, как этот подход нужно реализовать на выбранном языке программирования.

* Пишите псевдокод на достаточно низком уровне, так чтобы код из него генерировался практически автоматически.

[При описании метода] начните с основных моментов, а затем детализируйте их. Самая главная часть метода - заголовок-комментарий, описывающий действия метода, так что начните с краткой формулировки назначения метода. Написание этой формулировки поможет вам прояснить ваше понимание метода.

Написав общий комментарий, добавьте высокоуровневый псевдокод.

Написав псевдокод, вы окружаете его кодом, а псевдокод превращаете в комментарии программы. Если вы руководствуетесь перечисленными правилами создания псевдокода, комментарии в вашей программе будут полными и ясными.»

Ниже приведен второй вариант служебных комментариев для языка Паскаль, выбранный в соответствии с предложенной С. Макконеллом логикой написания процедуры: сначала пишем комментарии, а затем добавляем исходный код.

```
prg_otkr_kmnt      = "(*"
prg_zakr_kmnt      = "*) "
zagol_prg          = "(* i."
prg_t_IF           = " (** если"
prg_t_EI           = " (** иначе если"
prg_t_EB           = " (** иначе"
prg_t_E            = " (** конец"
prg_t_CB           = " (** цикл"            // начало цикла
prg_t_CE           = " (** конец цикла"    // конец цикла
prg_t_A            = " (**"                 // начало действия
prg_prow_vetki     = "wetka = "     // проверка ветки в условии
prg_k_vetke        = "wetka := "    // задание новой ветки
prg_t_RET          = " (** выход"
prg_STOP           = " (** КОНЕЦ СХЕМЫ"
```

Кстати, приводимые ниже схемы процедур b_sozdatx_shemuClick и podschet_max_koord формируются с использованием второго варианта служебных комментариев.

Но и такой вариант тоже не является совсем удобным, т.к. при его использовании при создании нового блока условия создаются служебные комментарии «* если», «* иначе» и «* конец», что, в свою очередь:

1) визуально перегружает текст процедуры;

2) требует ручной или автоматической подстановки для служебных комментариев «* иначе» и «* конец» соответствующих им ключевых слов языка программирования (для Паскаля это «end else begin» и «end;»); в текущей версии программы dal_vjaz_2 0.87.4 возможность автоматической подстановки ключевых слов языка программирования для служебных комментариев «* иначе» и «*конец» отсутствует, т.к. для автоматической подстановки при вставке в схему блока условия соответствующих служебным комментариям «* иначе» и «* конец» ключевых слов языка программирования нужно ввести в конфигурацию программы dal_vjaz_2 дополнительные параметры, а для версии 0.87.4 стояла обратная задача: оставить в файле конфигурации dal2.cfg как можно меньше параметров, задаваемых пользователем.

В связи с вышеуказанными недостатками второго варианта служебных комментариев я перешел к использованию третьего варианта: служебные комментарии для сложного условия совпадают с ключевыми словами сложного условия, а остальные служебные комментарии пишутся явно.

Ниже приведен третий вариант служебных комментариев для языка Паскаль:

```
prg_otkr_kmnt       = "(*"
prg_zakr_kmnt       = "*) "
zagol_prg           = "(* i."
prg_t_IF            = " if"
prg_t_EI            = " end else if"
prg_t_EB            = " end else begin"
prg_t_E             = " end;"
prg_t_CB            = " (** цикл"              // начало цикла
prg_t_CE            = " (** конец цикла"    // конец цикла
prg_t_A             = " (**"                   // начало действия
prg_prow_vetki      = "wetka = "      // проверка ветки в условии
prg_k_vetke         = "wetka := "     // задание новой ветки
prg_t_RET           = " (** выход"
prg_STOP            = " (** КОНЕЦ СХЕМЫ"
```

При использовании третьего варианта служебных комментариев возникает проблема, которой не было при использовании двух предыдущих вариантов: условия, размещаемые внутри действий в связи с тем, что они не имеют важного значения для общей логики процедуры, нужно маскировать, чтобы при чтении логической структуры процедуры эти условия не отображались на схеме.

Для решения этой проблемы в конфигурацию был введен параметр «маскировать ключевое слово» (его подробное описание приведено в пункте 1.5 настоящего материала).

Кстати, приведенные в пункте 1.4 служебные комментарии для русифицированной версии Компонентного паскаля отличаются тем, что для них не только служебные комментарии сложного условия, но и служебные комментарии начала и конца цикла совпадают с ключевыми словами языка программирования.

В русифицированной версии Компонентного паскаля определены четыре типа циклов:

- ЦИКЛ_ДЛЯ … КОНЕЦ_ЦИКЛА (FOR … END;) ;
- ЦИКЛ_ПОКА … КОНЕЦ_ЦИКЛА (WHILE … END;) ;
- ЦИКЛ_ДО … КОНЕЦ_ЦИКЛА_ДО (REPEAT … UNTIL) ;
- ЦИКЛ … КОНЕЦ_ЦИКЛА (LOOP … END;) .

Для всех этих типов циклов подходят определенные для русифицированной версии Компонентного паскаля служебные комментарии начала и конца цикла «ЦИКЛ» и «КОНЕЦ_ЦИКЛА».

По умолчанию файл конфигурации dal2.cfg настроен для работы с русифицированным Компонентным паскалем. В рабочем каталоге программы в подкаталоге \lang_cfg есть версии файлов dal2.cfg для следующих языков программирования:

- Компонентный паскаль: cpas.cfg;
- русифицированный Компонентный паскаль: cpasrus.cfg;

- Паскаль: pas.cfg;

- LUA: lua.cfg;

- C/C++ с закрывающим комментарием: cpp.cfg (файл для C/C++ со строчным комментарием «//» легко получить из файла для LUA).

Для работы с выбранным языком программирования нужно удалить или переименовать текущий файл dal2.cfg, а затем переименовать файл конфигурации для этого языка в файл dal2.cfg.

Из вышеприведенных вариантов служебных комментариев для различных языков программирования легко получить файлы конфигурации и для других языков программирования семейств Паскаль и C/C++.

2.4. Шаблоны редактирования процедуры

Для удобства можно завести файлы шаблонов редактируемых процедур с несколькими наиболее часто используемыми шаблонами редактирования процедуры.

Наиболее часто используемым шаблоном редактирования процедуры при программировании с циклом сложного условия является шаблон, приведенный на рис. 2.1.

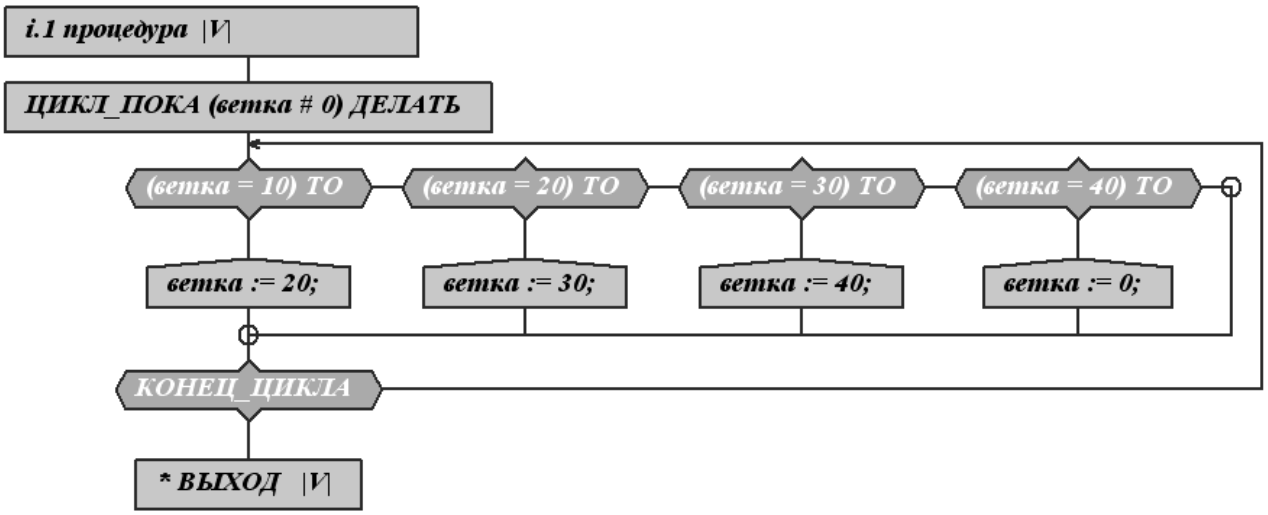

Рисунок 2.1 – Наиболее часто используемый шаблон процедуры при программирования с циклом сложного условия

Ниже приведен текст шаблона процедуры, показанного на рис. 2.1.

```
(* i.1 процедура  |V| *)
ПРОЦЕДУРА П1;
ПЕРЕМЕННЫЕ
ветка:ЦЕЛЫЙ;
НАЧАЛО
   ветка :=1;
   ЦИКЛ_ПОКА (ветка # 0) ДЕЛАТЬ
      ЕСЛИ (ветка = 10) ТО
         ветка := 20;
      ИНЕСЛИ (ветка = 20) ТО
         ветка := 30;
      ИНЕСЛИ (ветка = 30) ТО
         ветка := 40;
      ИНЕСЛИ (ветка = 40) ТО
         ветка := 0;
```

```
      ИНАЧЕ
      КОНЕЦ;
   КОНЕЦ_ЦИКЛА;
   (** ВЫХОД   |V| *)
   КОНЕЦ  П1;
(** КОНЕЦ ПРОЦЕДУРЫ *)
```

Для начала работы с этим шаблоном нужно нажать кнопку «Счит.» на панели редактирования программы dal_vjaz_2, после чего скопировать в открывшееся окно текст шаблона и нажать кнопку «Сохранить».

2.5. Редактирование структурных блок-схем произвольного вида

Для редактирования в программе dal_vjaz_2 структурных блок-схем произвольного вида следует использовать главным образом минимизированную форму отображения схемы.

Если размер минимизированной схемы по высоте начинает превышать три высоты экрана, то это явно говорит о том, что процедура слишком разрослась, и необходимо начать выделять из нее подчиненные процедуры для уменьшения размера основной процедуры. Можно, конечно, наращивать размер схемы по высоте и дальше, но редактировать такую схему будет уже очень неудобно.

Чтобы перейти к фрагменту полной формы отображения схемы, содержащему элемент минимизированной схемы, на который наведен указатель мыши, нужно нажать клавишу «F2» на клавиатуре компьютера.

2.6. Использование пиктограмм и рисунков при создании схем

Для работы с пиктограммами и рисунками при создании схем используется определитель рисунка и стиля, формат которого задается в файле dal2.cfg.

По умолчанию определитель рисунка и стиля имеет формат « РИС .N1 .N2»,

где N1 – двузначный номер/мнемоника пиктограммы или трехзначный номер рисунка;

N2 – номер одного из стилей блока (схемы), задаваемых в файле styles.cfg (может отсутствовать, по умолчанию используется стиль 1, а для блоков условий – стиль 2).

Пиктограммы находятся в подкаталоге \picts, а рисунки – в подкаталоге \images рабочего каталога программы dal_vjaz_2. И пиктограммы и рисунки должны быть jpg-файлами.

Пиктограммы используются для обозначения дополнительных маршрутных операторов (см. пункт 1.3 настоящего материала) или поясняют действия, выполняемые в блоке.

Номера и мнемоника для первых девяти пиктограмм, обозначающих дополнительные маршрутные операторы, являются предопределенными:

- c? – продолжение цикла по условию;
- b? – выход из цикла по условию;
- u? – переход goto по условию вверх;
- mu – метка перехода goto по условию вверх;
- d? – переход goto по условию вниз;
- md – метка перехода goto по условию вниз;
- e? – переход к обработке ошибки по условию;
- er – обработка ошибки;
- r? – выход из процедуры по условию.

Один из возможных вариантов пиктограмм приведен на рис. 2.2.

Номера рисунков начинаются с 001 и должны следовать друг за другом по порядку номеров – отсутствие следующего по порядку номера означает окончание чтения набора рисунков.

В текущей версии программы dal_vjaz_2 0.87.4 пиктограммы отображаются на схеме в масштабе 1/8, а рисунки – в масштабе 1/1.

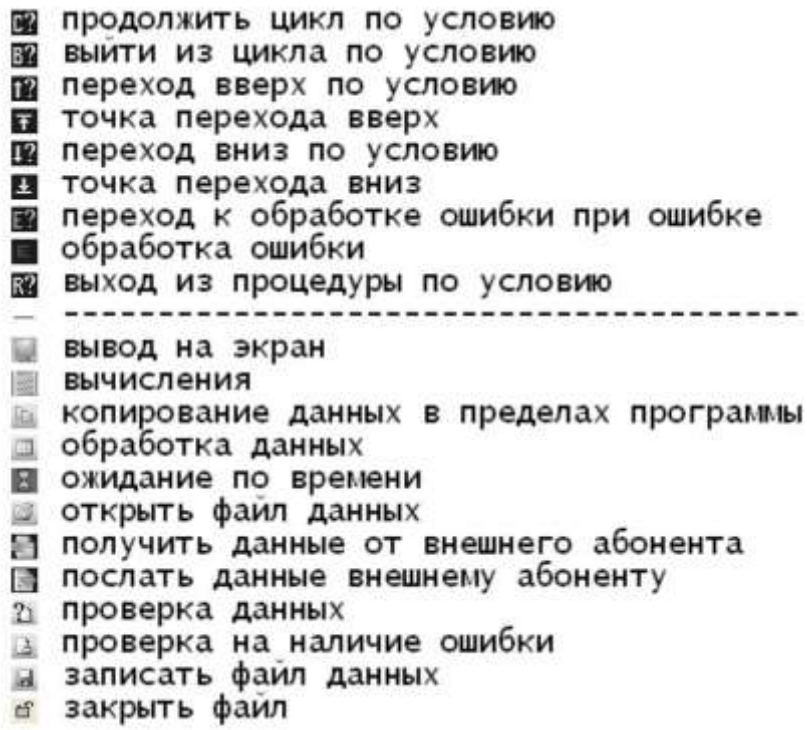

Рисунок 2.2 – Предлагаемый набор пиктограмм для визуальных элементов

2.7. Всплывающие окна для элементов схемы

Для минимизированной формы схемы при наведении указателя мыши на элемент схемы появляется всплывающее окно этого элемента, соответствующее виду этого элемента для полной формы отображения схемы. Чтобы отменить появление всплывающих окон элементов при случайном наведении указателя мыши на эти элементы при перемещении указателя мыши по схеме, нужно нажать и удерживать клавишу Shift на клавиатуре компьютера.

2.8. Создание схемы для псевдокода

В ранних версиях программы существовал псевдокод по умолчанию, служебные комментарии для которого приведены в пункте 1.4. Кстати, с использованием такого псевдокода была создана схема «Морозко» (рис. 2.3).

В текущей версии программы псевдокод ничем с точки зрения способа задания не отличается от других языков программирования – для него нужно сформировать файл dal2.cfg.

Схема «Морозко» является примером структурной блок-схемы произвольного вида, написанной на псевдокоде.

Ниже приведено начало текста для схемы «Морозко».

```
Схема.1. МОРОЗКО  [-] РИС .001 .4
МОРОЗКО
(схема сюжета сказки)
   *  [-]  РИС .002 .2

   Жил-был Иван, вдовий сын.
```

```
Ваня - парень хоть куда:
и рыбак, и охотник,
и пахарь, и плотник.
И воинским навыкам обучен.
* [-]  РИС .003

Однажды в лесу
повстречался Иван
с разбойниками.
* [-]  РИС .004

. . .
```

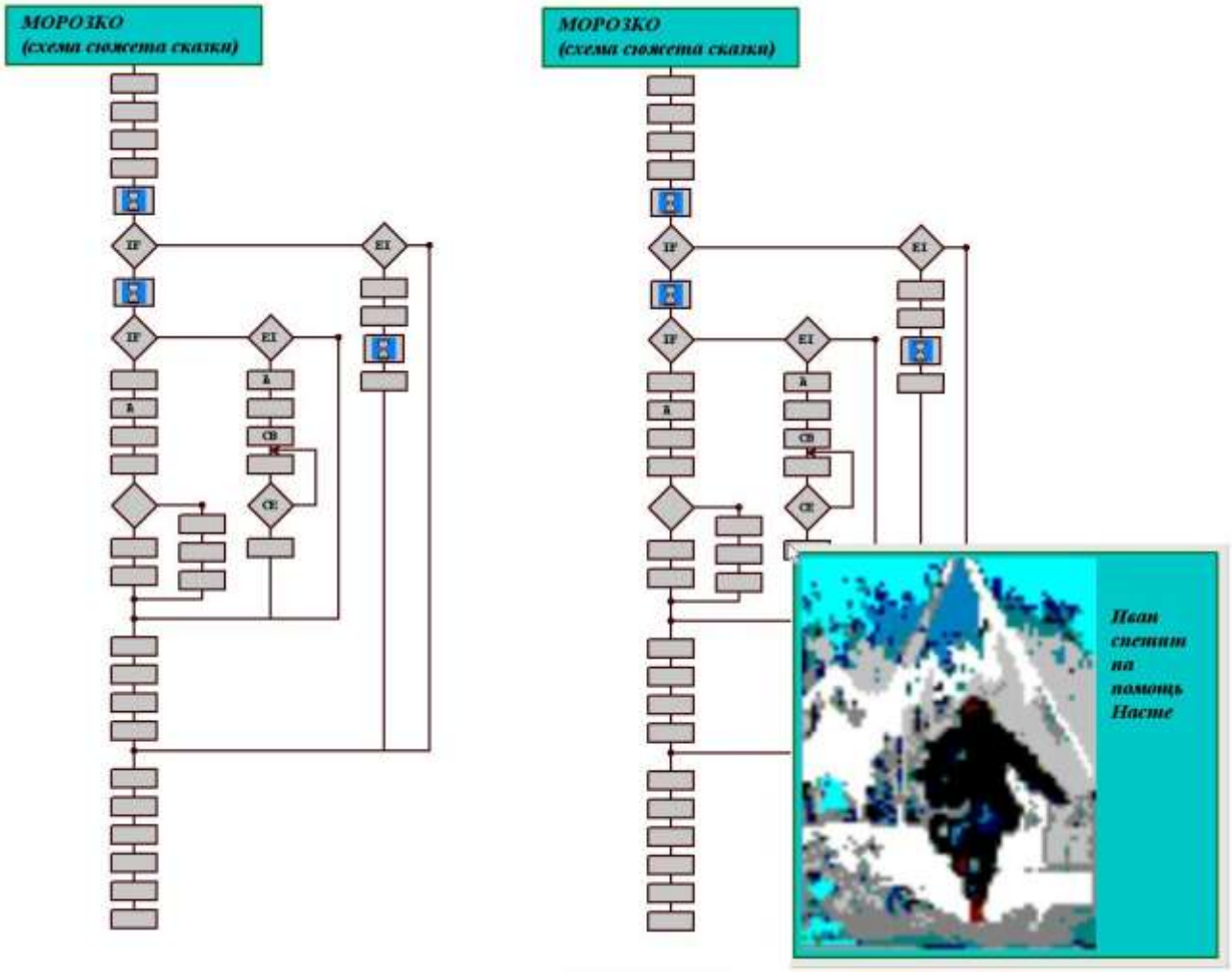

Рисунок 2.3 – Минимизированная форма отображения для схемы «Морозко»

Из приведенного фрагмента текста для схемы «Морозко» видно, что тексты элементов схемы начинаются с символов «*», что первые строки элементов маскированы и что каждому элементу присвоен свой рисунок. Для отображения схемы задан стиль 4.

В полной форме отображения схему «Морозко» на листе А4 разместить с приемлемым качеством не получится, поэтому на рис. 2.3 приведена минимизированная форма схемы без всплывающих окон и со всплывающим окном для элемента схемы, на который наведен указатель мыши.

Как видно из рисунка, минимизированная форма дает возможность поочередного рассматривания рисунков, привязанных к элементам схемы путем наведения указателя мыши на эти элементы.

В вышеприведенной схеме присутствуют все элементы, необходимые для построения структурной блок-схемы:

- прямоугольники действий;
- основное и дополнительное условия IF и EI сложных условий;
- блоки начала и конца цикла CB и CE.

Практикуясь с построением структурных схем для псевдокода можно, даже не имея твердых познаний в программировании, совершенствовать свое алгоритмическое мышление. А отсюда и до написания реальных процедур на каком-нибудь языке программирования уже недалеко.

В ДАЛВЯЗ 2 в схемах для псевдокода есть возможность определять одновременные (параллельные) действия, чего нельзя реализовать в схемах для языков программирования в связи с затруднительностью визуализации на одной схеме логики параллельного программирования для языков, с которыми работает ДАЛВЯЗ 2.

На рис. 2.4 показан фрагмент схемы «Морозко», изображающий события, происходящие одновременно с Настей и Иваном. Этому фрагменту схемы соответствуют следующие фрагменты текста:

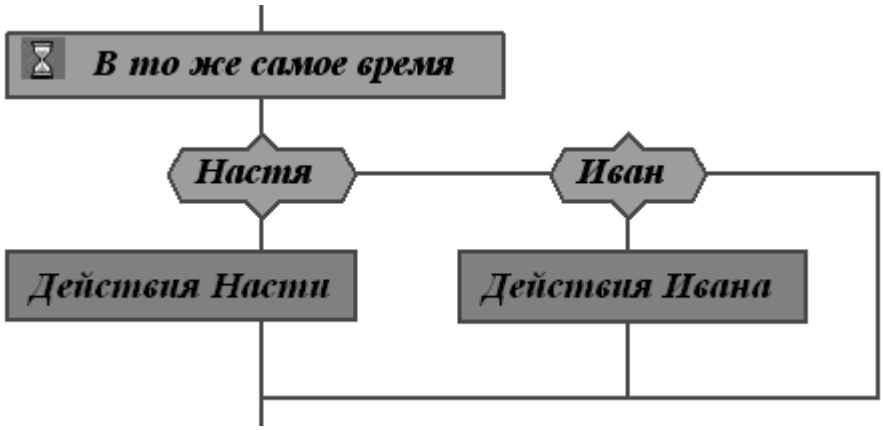

Рисунок 2.4 – Вариант изображения параллельных действий в ДАЛВЯЗ 2

- текст блока «в то же самое время»:

```
* [-]  РИС .14 .
В то же самое время
```

- первая строка текста маскирована, там же задается пиктограмма «часы»;

- текст блока «Настя»:

```
* если [-]
Настя
```

- блок «Настя» задает условие, первая строка которого маскирована;

- текст блока «Иван»:

```
* иначе если  [-]
Иван
```

- блок «Иван» задает дополнительное условие для блока «Настя», и первая строка блока «Иван» маскирована.

Таким образом, за счет маскирования первых строк блоков условий содержимое их текста в схеме для псевдокода может быть ассоциировано не с текстом условия, а с именем объекта, чьи действия выполняются в этой ветке условия, что и позволяет говорить о возможности отображения одновременных (параллельных) действий в ДАЛВЯЗ 2 в схемах для псевдокода.

2.9. Описание логики программы на над-процедурном уровне

Т.к. у меня сейчас нет готового ответа на вопрос, как должно выглядеть визуальное отображение структуры программы для над-процедурного уровня (на котором процедуры видны как атомы, без отображения их внутренней структуры), то в версии программы dal_vjaz_2 0.87.4 я исключил понятие редактируемого файла и возможность задания обрабатываемых программой ссылок на другие процедуры.

Но я по-прежнему считаю, что присвоение процедурам уникальных номеров в файле исходного кода упрощает документирование программы.

Ниже приведен фрагмент руководства программиста для программы dal_vjaz_2, где вместо ссылок на процедуры по их именам используются ссылки на процедуры по их номерам, что улучшает читабельность текста.

«Чтение из ListBox-а схемы в текстовом формате и запись элементов схемы в массив tek_alg.m_el : u_dalvj.29 и подчиненные процедуры u_dalvj.29.1 – u_dalvj.29.8; u_dalvj.29.8 определяет начало элемента, в том числе в зависимости от параметра конфигурации cfg_not_kword; в процедуре u_dalvj.29.6 вызывается процедура u_ls.33, в которой определяются уровни (отступы в пробелах) для каждого элемента схемы, а затем вызывается u_ls.9 – чтение логической структуры схемы.

Для считанной из текстового формата в массив элементов схемы проверяется логическая структура и создается массив вертикалей схемы с привязкой элементов схемы к вертикалям : u_ls.9 и подчиненные процедуры u_ls.10 – u_ls.12, u_ls.14 – u_ls.22, u_ls.25, u_ls.34 – u_ls.37, u_ls.42 – u_ls.45.»

Пока не придуман общепризнанный удобный способ отображения структуры программы для над-процедурного уровня, описание над-процедурной логики программы практичнее излагать в текстовом формате (с возможным использованием пояснительных схем и диаграмм) как последовательность абзацев, поясняющих логику взаимодействия процедур и обработки данных в документируемой программе.

2.10. Создание текущей версии процедуры вывод_схемы, начало

Приведенная в качестве примера в пункте 1.6 текущая версия процедуры `вывод_схемы` является версией №3 этой процедуры.

Версия №1 была написана на Delphi 4 и ее текст выглядит следующим образом:

```
(* i.21. вывод схемы *)

procedure Tform_dalvjaz.wywod_shemy(Sender: TObject);
var      i, j, x1, y1, x2, y2, y3, y4, len, ii, yw0, yw1,
             smxzl, smxzr, ipe, u, sz, cd, smx, smy, smxel:integer;
         stro, stro_u:s_dl_stro;  st:string[50];
         RectB, RectF:TRect;
label    m_end;
         (** начало *)
         begin

         if (cfg_dal2html[1] = '0') then begin
            Bitmap.Width := Pwidth;
            Bitmap.Height := PHeight;
         end;

         if ((fl_teksta = YES) or (fl_shemy = NO))  then  begin

            if (dalvjaz_beg = 0)  then  fajly.wywod_teksta;
            goto  m_end;
         end;

         stilx__soed_linii;
         stilx__cwet_fona;

         if (shema_min = true) then
            Bitmap.Canvas.Brush.Color:= RGB(200,200,200);

         Bitmap.Canvas.FillRect
                   (Rect(0,0,Bitmap.Width,Bitmap.Height));

         if (shema_min = false)  then
                                 stilx__oboi;

         (** проверка правильности схемы  РИС .e?
             и подсчет максимальных координат схемы *)
         smx := 0;  smy := 0;
         if (dalvjaz_penwidth = 2)  then  smx := -1;

         if ((tek_alg.kol_elem = 0) or
             (dalvjaz_shema_schitana = NO))  then begin

           goto  m_end;  // если элементов нет или не проверены все
                         // ветки схемы, то выход
         end;

         podschet_max_koord(Sender);

         (** вывод главной вертикали *)

         j := 0;
         ii := tek_alg.m_inew[j];

         i:=0;
         while (i < tek_alg.m_el_wert[j]) do begin

           if (tek_alg.m_el[ii].n_wert = j)  then begin

             if ((tek_alg.m_el[ii].used = YES) or
                 (j = (tek_alg.kol_wetok-1)))  then begin

               wywod_elem(Sender, j, ii, i);   inc(i);  end;
           end;
```

```
  inc(ii);
end;


(** цикл вывода элементов дополнительных вертикалей *)

for  j:=0  to  tek_alg.kol_dwert-1   do begin ////
   (**)
   smxel := 0;
   if ((tek_alg.m_wert[KOL_WETOK+j].parent_el <> -1) and
       (tek_alg.m_wert[KOL_WETOK+j]._type <> T_WERT_CIKLA)) then
      smxel := el_shirina(tek_alg.m_wert[KOL_WETOK+j].parent_el);

   x1 := tek_alg.m_wert[tek_alg.m_wert[KOL_WETOK+j].parent_w].x +
                          dalvjaz_smx + (smxel div 2);
   x2 := dalvjaz_smx + tek_alg.m_wert[KOL_WETOK+j].x;
   y1 := dalvjaz_smy + tek_alg.m_wert[KOL_WETOK+j].y1;

   (** если *)
   if (tek_alg.m_wert[KOL_WETOK+j]._type = T_WERT_DOCHX)  then begin
      (**)
      if (tek_alg.m_wert[KOL_WETOK+j].type_1_el = t_EI) then begin
        if (tek_alg.m_el[ tek_alg.m_inew[KOL_WETOK+j] ].fl_wetki = YES)
        then begin
          smxel := el_shirina(tek_alg.m_inew[KOL_WETOK+j]);
          x2 := x2 - (smxel div 2);
        end;
      end;

      dalvjaz_line(x1,y1,x2+smx,y1);

      ipe := tek_alg.m_wert[KOL_WETOK+j].parent_endw;

      x1 := dalvjaz_smx + tek_alg.m_wert[ipe].x;
      x2 := dalvjaz_smx + tek_alg.m_wert[KOL_WETOK+j].x;
      y1 := dalvjaz_smy + tek_alg.m_wert[KOL_WETOK+j].y1 +
                                  tek_alg.m_wert[KOL_WETOK+j].tyel;
      dalvjaz_line(x1,y1,x2+smx,y1);

      if (tek_alg.m_el_wert[KOL_WETOK+j] <> 0) then
      if (tek_alg.m_wert[KOL_WETOK+j].type_1_el <> t_EB)  then begin

        if ((tek_alg.m_wert[KOL_WETOK+j].type_1_el <> t_EI) or
           (tek_alg.m_el[ tek_alg.m_inew[KOL_WETOK+j] ].fl_wetki = NO))
        then begin

          y1 := dalvjaz_smy + tek_alg.m_wert[KOL_WETOK+j].y1;
          y2 := y1 + tek_alg.m_el[ tek_alg.m_inew[KOL_WETOK+j] ].y1;

          dalvjaz_line(x2,y1,x2,y2);
        end;
      end;
   (** конец *) end;
   (** если *)
   if (tek_alg.m_wert[KOL_WETOK+j]._type = T_WERT_CIKLA)  then begin
      (**)
      y1 := y1 - (dalvjaz_chdy div 4);  // чтобы не касаться меток
                                        // заголовков веток
      dalvjaz_line(x1,y1,x2+smx,y1);

      // стрелка цикла
```

```
            dalvjaz_line(x1,y1,x1+dalvjaz_chw,y1-(dalvjaz_chw div 2));
            y4 := y3 + (dalvjaz_chdy div 4);
            dalvjaz_line(x1,y1,x1+dalvjaz_chw,y1+(dalvjaz_chw div 2));


            y2 := y1 + tek_alg.m_wert[KOL_WETOK+j].tyel +(dalvjaz_chdy div 4);
            smxel := el_shirina(tek_alg.m_wert[KOL_WETOK+j].parent_end);
            x1 := x1 + (smxel div 2);
            dalvjaz_line(x1,y2,x2+smx,y2);
         (** конец *) end;

         (** если *)
         if (tek_alg.m_el_wert[KOL_WETOK+j] = 0)  then begin
            (**)
            y1 := dalvjaz_smy + tek_alg.m_wert[KOL_WETOK+j].y1;
            y2 := y1 + tek_alg.m_wert[KOL_WETOK+j].tyel;

            if (tek_alg.m_wert[KOL_WETOK+j]._type = T_WERT_CIKLA) then
               y1 := y1 - (dalvjaz_chdy div 4); // не касаться згл. веток
            dalvjaz_line(x2,y1,x2,y2);
         (** иначе *)
         end else begin ////
            (**)
            ii := tek_alg.m_inew[KOL_WETOK+j];

            i:=0;
            while (i < tek_alg.m_el_wert[KOL_WETOK+j]) do begin

              if (tek_alg.m_el[ii].n_wert = (KOL_WETOK+j))  then begin

                if (tek_alg.m_el[ii].used = YES)  then begin

                   wywod_elem(Sender, KOL_WETOK+j, ii, i);   inc(i);  end;
              end;

              inc(ii);
            end;
         (** конец *) end; ////
      (** конец цикла *)  end; ////

      (** выход *)
      m_end:

      if (cfg_dal2html[1] = '0') then begin

         RectF.Left := PLeft;
         RectF.Top  := PTop;
         RectF.Right := PLeft+PWidth-1;
         RectF.Bottom := PTop+PHeight-1;

         RectB.Left := 0;
         RectB.Top  := 0;
         RectB.Right := PWidth-1;
         RectB.Bottom := PHeight-1;

         Canvas.CopyRect(RectF, Bitmap.Canvas, RectB);
      end;

      end;
(** КОНЕЦ СХЕМЫ *)
```

Чтобы читатель не выпал из контекста под грузом предлагаемых листингов и блок-схем, нужно дать некоторые пояснения.

Раз уж здесь разговор идет о визуальном программировании, то тем, кто дочитал до этого места, возможно будет интересно взглянуть на вопрос создания схемы не только с точки зрения пользователя, но и с точки зрения программиста.

2.11. Создание схемы с точки зрения программиста

Для работы со схемой задается переменная tek_alg типа t_alg (несущественные для дальнейшего пояснения поля структур схемы, элемента и вертикали здесь не приводятся):

```
type  t_alg = record

  m_el :array[0..KOL_ELEM] of  t_elem;  // массив элементов схемы

  kol_elem :integer;         // число элементов схемы

  kol_wetok :integer;        // число веток
  // сейчас  KOL_WETOK = 1, т.к. только 1 главная вертикаль

  m_wert :array[0..KOL_WERT] of  t_wert;  // массив вертикалей схемы
                             // первые KOL_WETOK вертикалей отводятся для
                             // хранения главных вертикалей
                             // дополнительные вертикали хранятся
                             // начиная со смещения KOL_WETOK
  kol_dwert :integer;        // число дополнительных вертикалей
  m_el_wert :array[0..KOL_WERT] of integer;  // число элементов на вертикали

  m_inew :array[0..KOL_WERT] of integer;     // массив индексов начальных
                                             // элементов вертикалей в m_el
end;
```

Для элемента задается структура t_elem:

```
type t_elem = record
    _type :integer;          // тип элемента
    n_wert :integer;         // номер вертикали элемента
    n_na_wert :integer;      // номер элемента на вертикали (с 1)
    n_dwert :integer;        // номер дочерней вертикали элемента (для условия)

    wetka2 :integer;         // по умолчанию = -1;  = 1, если t_A задает
                             // переход к другой ветке цикла-силуэта
    n_wetki :integer;        // номер ветки, задаваемый t_IF, t_EI, t_A
    fl_usl :byte;            // YES/NO, флаг условия для IF, EI, CE

    x1,x2,y1,y2 :integer;  // координаты в системе координат вертикали

    m_d :array[0..KOL_STR_DEJSTW] of  str210;  // массив строк действий
    kol_s_d :integer;        // число всех строк действия (элемента)
    kol_sl_str :byte;        // число служебных строк действия, они маскированы
                             // и не выводятся
    n_img :integer;          // -1: рисунок или значок к элементу не привязаны
                             // от 0 до KOL_IMG-1: рисунок
                             // >= KOL_IMG: значок (пиктограмма)
    wydel: integer;          // 0: элемент не выделен на схеме
                             // 1: элемент выделен на схеме как старший
                             // 2: элемент выделен на схеме как подчиненный

    urowenx :integer;        // уровень в ЛСП (главный =1), чем больше в тексте
                             // процедуры отступ текста элемента слева,
```

```
                          // тем больше уровень элемента в ЛСП

    n_str :integer;       // номер элемента (индекс) в логической структуре
                          // процедуры (ЛСП) и в массиве tek_alg.m_el  (с 0)
    n_strz :integer;      // номер строки элемента в тексте процедуры  (с 1)

    stilx :string[dl_name_el];// стиль элемента, задаваемый в параметре РИС
    n_stilq :integer;     // № стиля для элемента

    used :byte;           // YES/NO  сейчас всегда YES после создания эл-та
end;
```

Для вертикали задается структура t_wert:

```
type  t_wert = record

  _type :integer;      // главная, дочерняя, вертикаль цикла
  parent_w :integer;   // индекс родительской вертикали или -1
  parent_el :integer;  // индекс родительского элемента или -1
  parent_endw :integer;// индекс родительской вертикали для элемента конца
  parent_end :integer; // индекс элемента конца, закрывающего дочернюю
                       // вертикаль или -1
  x :integer;          // координаты вертикали в системе координат схемы
  y1 :integer;
  y2 :integer;         // координата нижнего конца вертикали в системе
                       // координат схемы, вычисляем как y1 + tyel
  tyel :integer;       // координата У следующего эл. в системе координат
                       // вертикали в процессе создания вертикали
                       // после создания вертикали - нижняя точка вертикали в
                       // системе координат вертикали
  n_sprawa :integer;   // номер справа от главной вертикали
                       // или 0 для главной вертикали
  dx :integer;         // ширина вертикали по x
  type_1_el :integer;  // тип 1-го элемента для вертикали
                       // если t_EB, то линия до 1-го элемента на вертикали
                       // рисуется сама, иначе нужно рисовать вручную

  kol_el_wyrawn :integer;  // кол-во эл-тов на вертикали, которые нужно
                       // выравнивать

  used :byte;         // IS/NO
end;
```

Сначала текст процедуры считывается в массив tek_alg.m_el с присвоением типа каждому элементу.

Для схемы определены следующие типы элементов:

```
const    t_HD =   7; // заголовок процедуры
const    t_IF =   1; // если (условие), то начать блок
const    t_EI =   2; // конец блока иначе если (условие), то начать блок
const    t_EB =   3; // иначе начать блок
const    t_E  =   4; // конец блока
const    t_CB =   5; // начало цикла
const    t_CE =   6; // конец цикла
const    t_A  =   9; // действие (action)
const    t_RET =  10;  // конец, возврат из процедуры (RETURN)
const    t_STOP = 100; // конец текста схемы, псевдотип
```

Если считывание текста схемы выполнено успешно, то выполняется чтение логической структуры процедуры, задание вертикалей схемы и привязка элементов схемы к вертикалям. Эти действия выполняются процедурой Tform_ls.b_sozdatx_shemuClick и вызываемыми из нее

подчиненными процедурами. Ниже, на рис. 2.5 – 2.10 приведены ветки процедуры b_sozdatx_shemuClick.

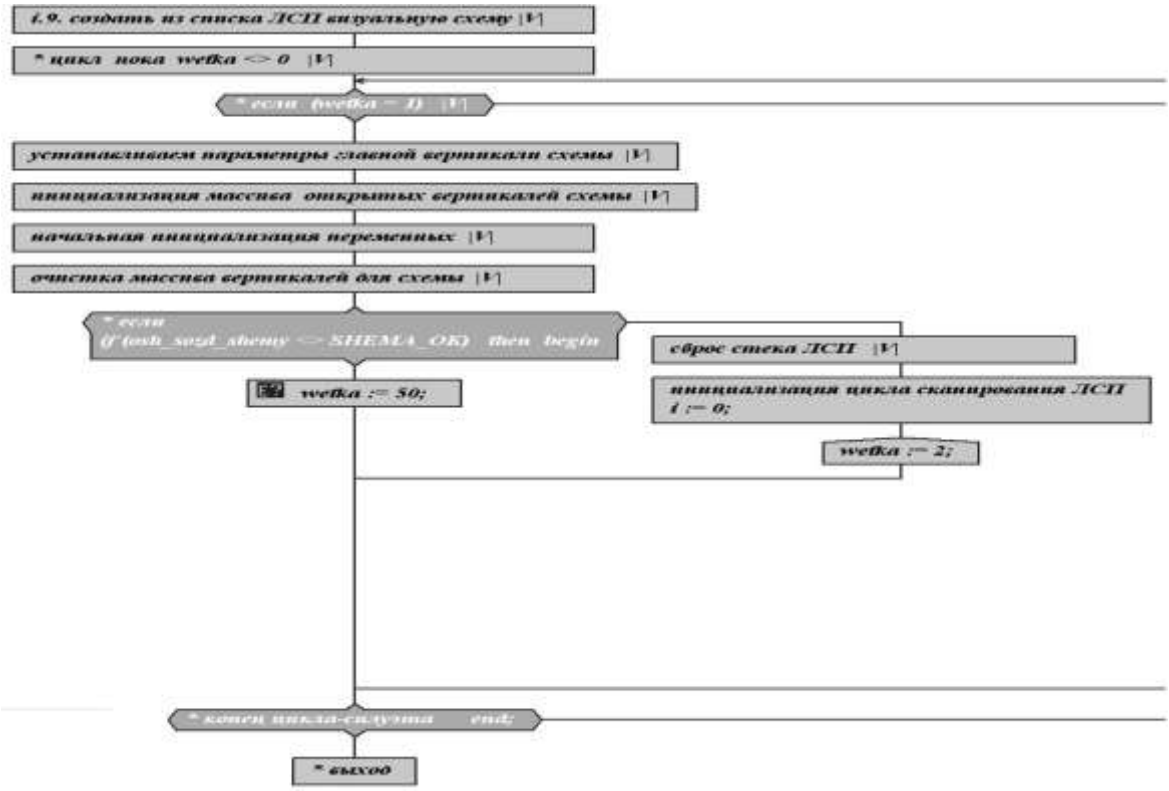

Рисунок 2.5 – Ветка 1 процедуры b_sozdatx_shemuClick

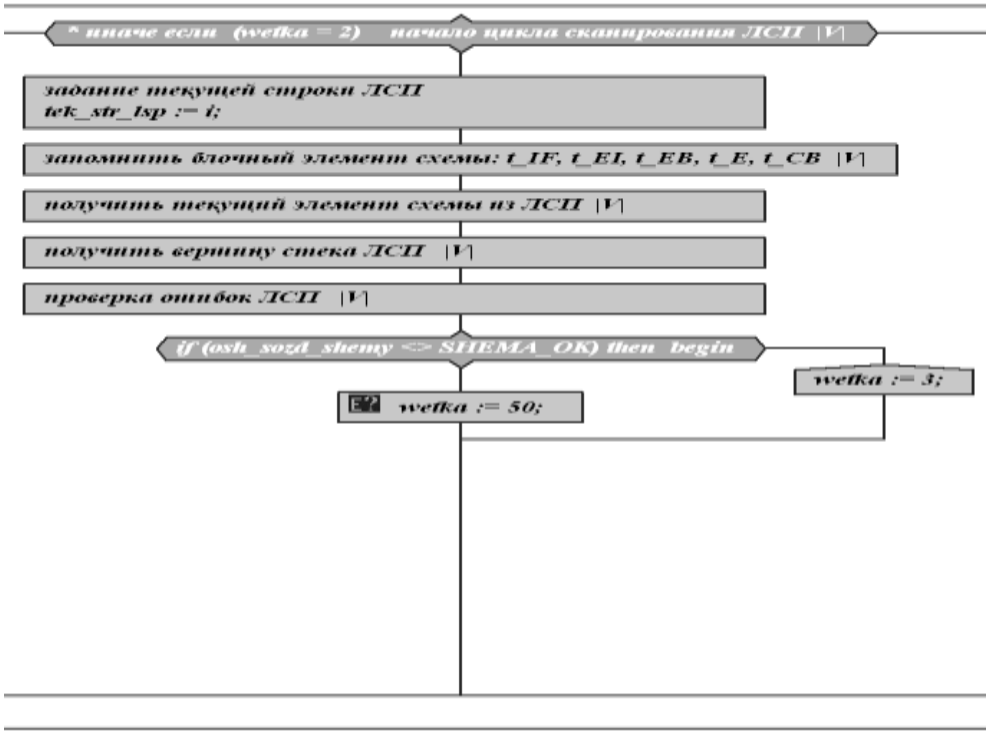

Рисунок 2.6 – Ветка 2 процедуры b_sozdatx_shemuClick

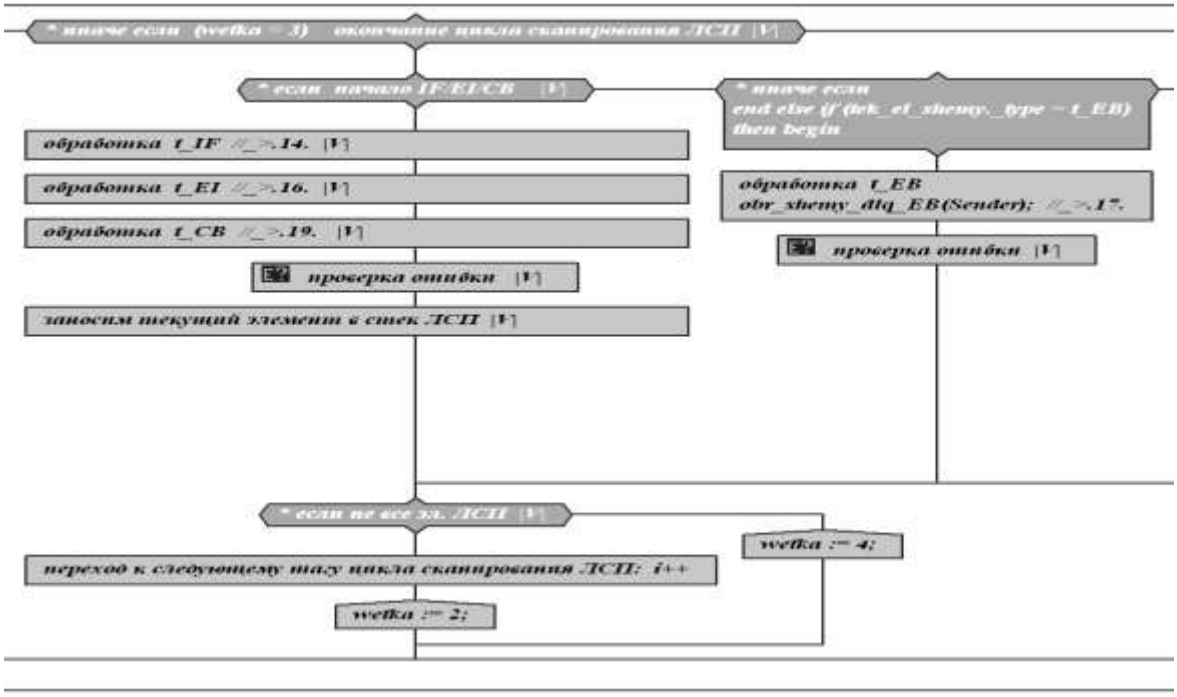

Рисунок 2.7 – Начало ветки 3 процедуры b_sozdatx_shemuClick

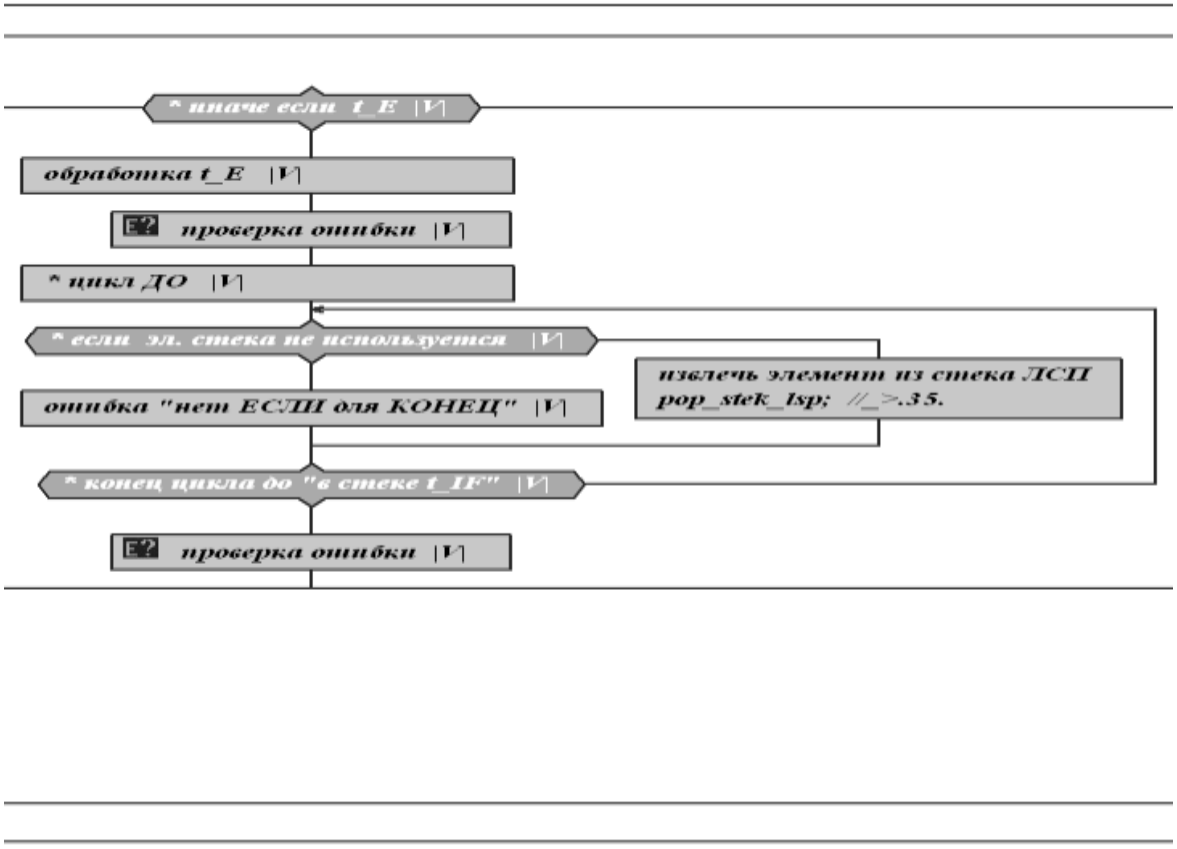

Рисунок 2.8 – Продолжение ветки 3 процедуры b_sozdatx_shemuClick

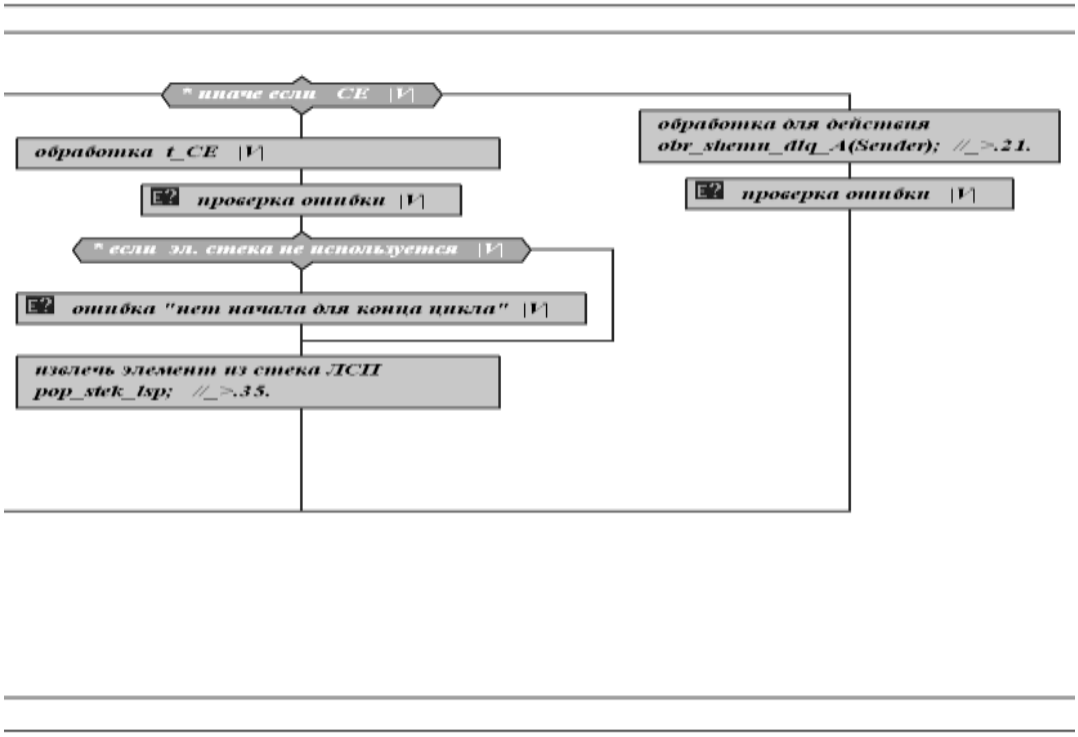

Рисунок 2.9 – Окончание ветки 3 процедуры b_sozdatx_shemuClick

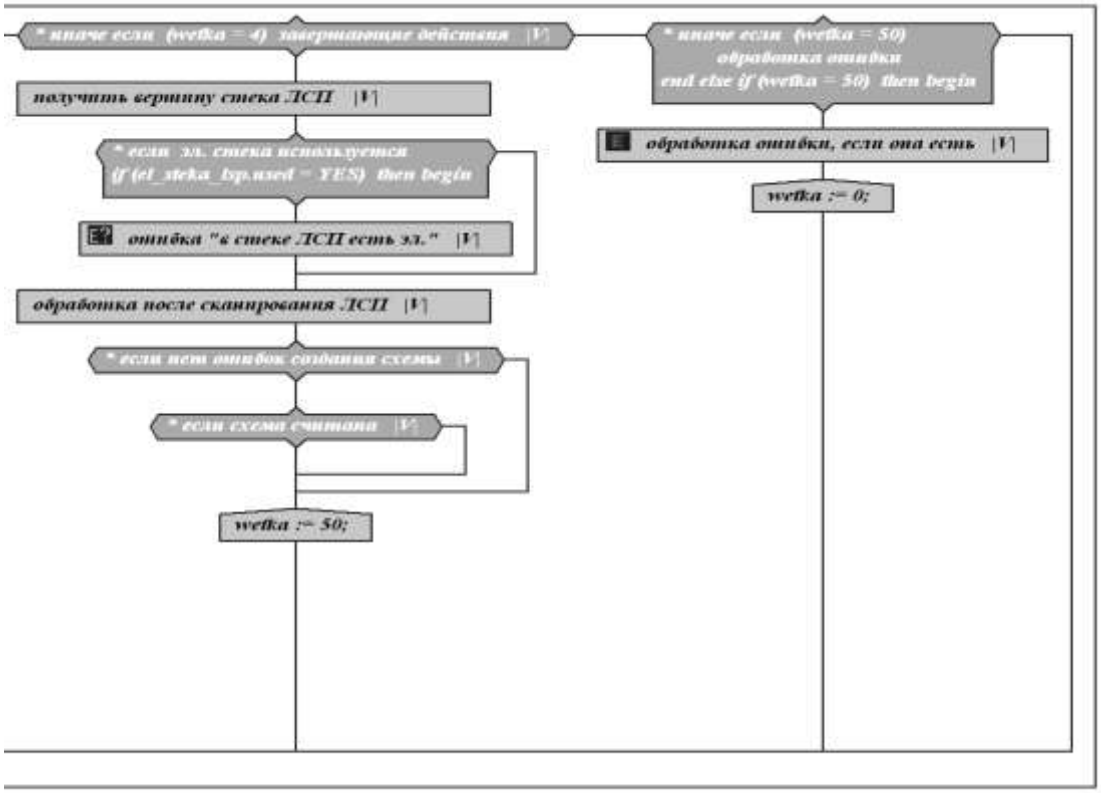

Рисунок 2.10 – Ветки 4 и 50 процедуры b_sozdatx_shemuClick

Обработка ошибок в процедуре b_sozdatx_shemuClick выполняется следующим образом:

```
if (osh_sozd_shemy <> SHEMA_OK)  then begin

   wetka := 50;   continue;
end;
```

При выполнении процедуры b_sozdatx_shemuClick формируется размер схемы по вертикали, а горизонтальные размеры элементов в этой процедуре для простоты принимаются одинаковыми и равными некоторому условному значению. В результате для каждой вертикали формируется значение переменной n_sprawa, равное условному расстоянию этой вертикали от главной (первой слева) вертикали схемы.

Вычисление размера схемы по горизонтали производится в процедуре Tform_dalvjaz.podschet_max_koord, на рис. 2.11 представлена минимизированная форма отображения этой процедуры, а ее ветки приведены ниже на рис. 2.12 – 2.18.

Т.к. процедура podschet_max_koord помещается в один экран по высоте только в своей минимизированной форме, то для наглядности отображения схемы процедуры некоторые из ее веток пришлось поделить на несколько частей.

В текущей версии программы dal_vjaz_2 действия для ветки 5 процедуры podschet_max_koord замаскированы, поэтому на рисунках 2.12 – 2.18 приведены только ветки 1 – 4.

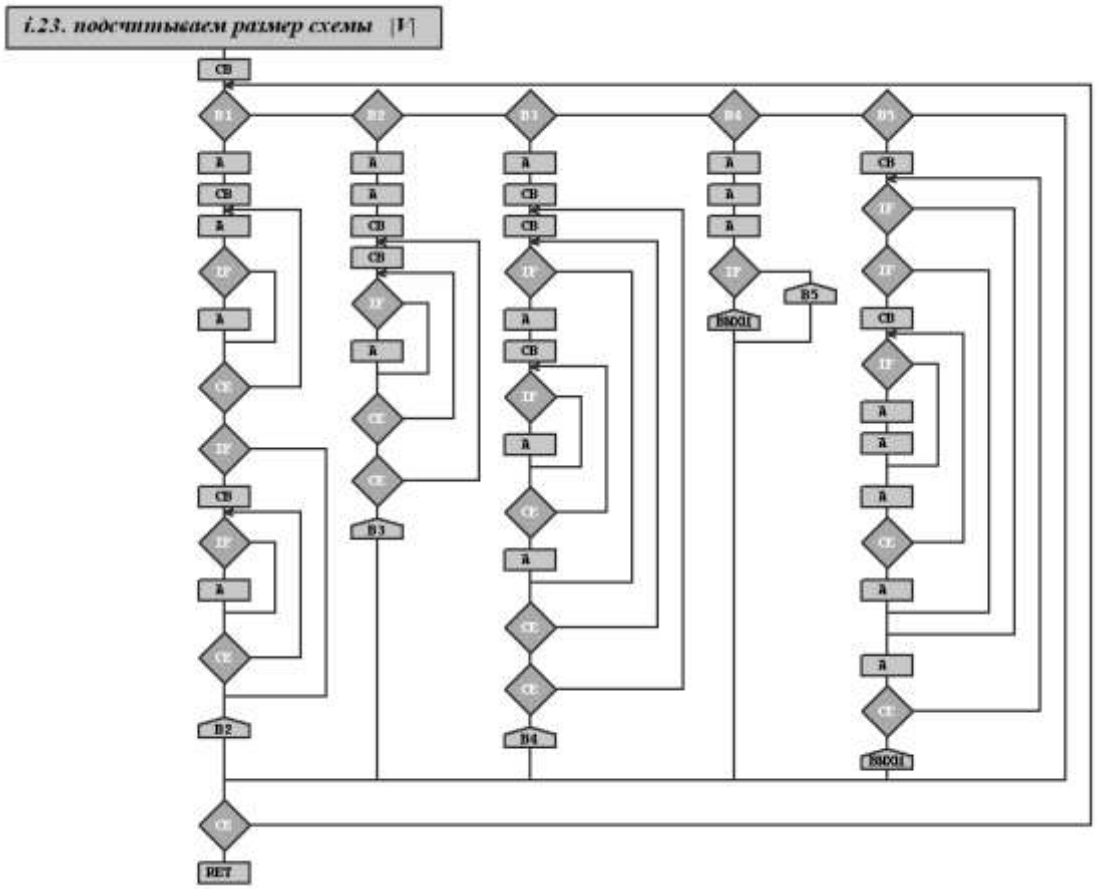

Рисунок 2.11 – Минимизированная форма процедуры podschet_max_koord

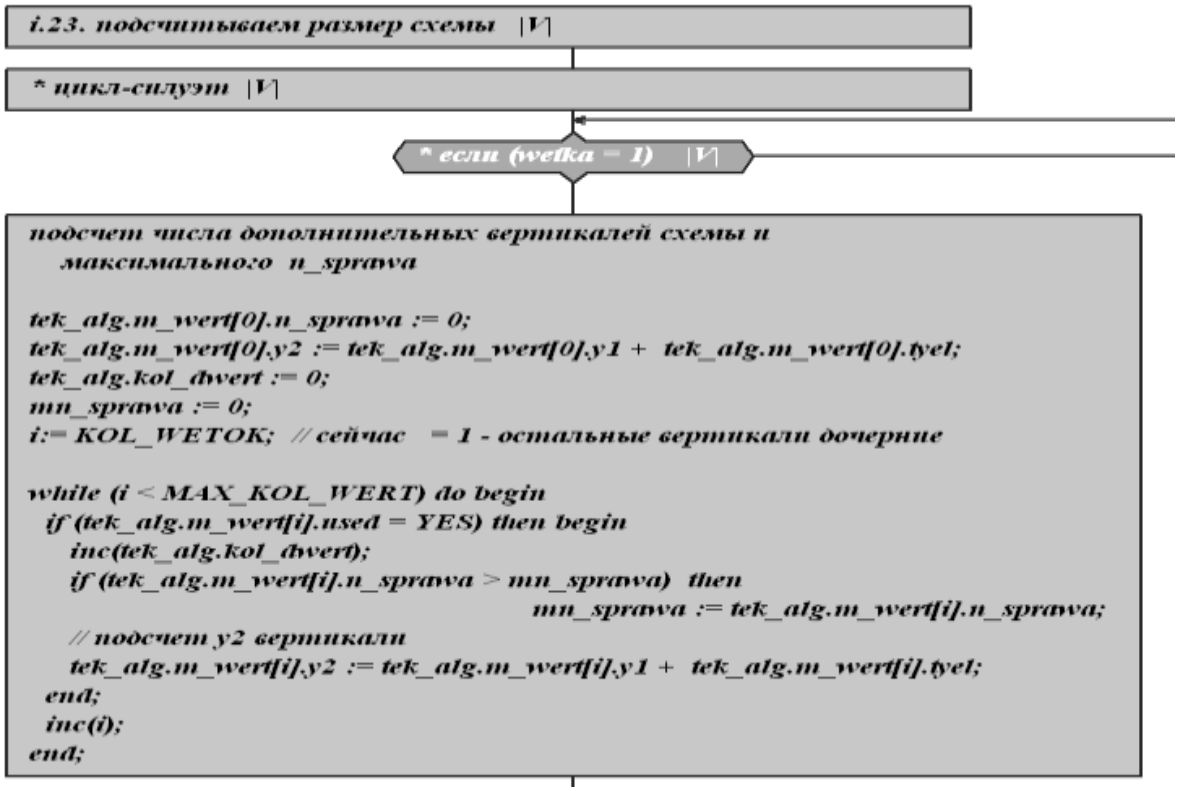

Рисунок 2.12 – Начало ветки 1 процедуры podschet_max_koord

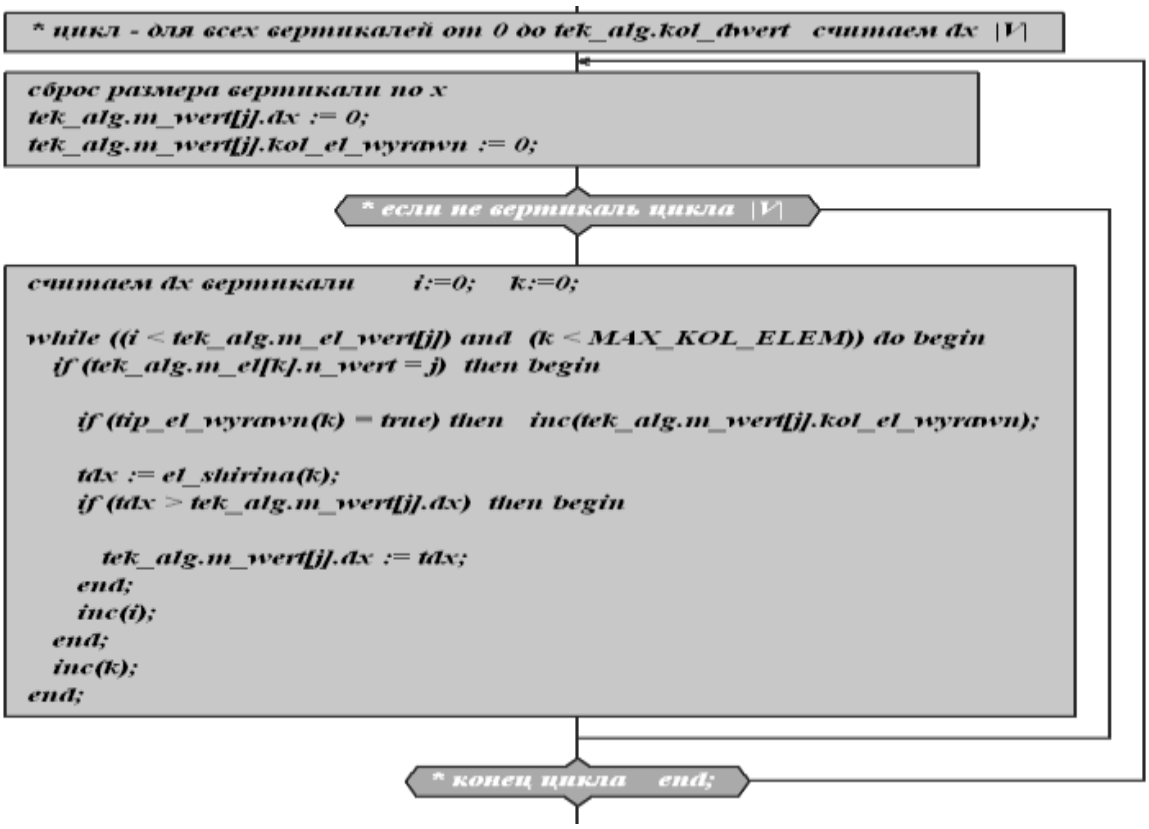

Рисунок 2.13 – Продолжение ветки 1 процедуры podschet_max_koord

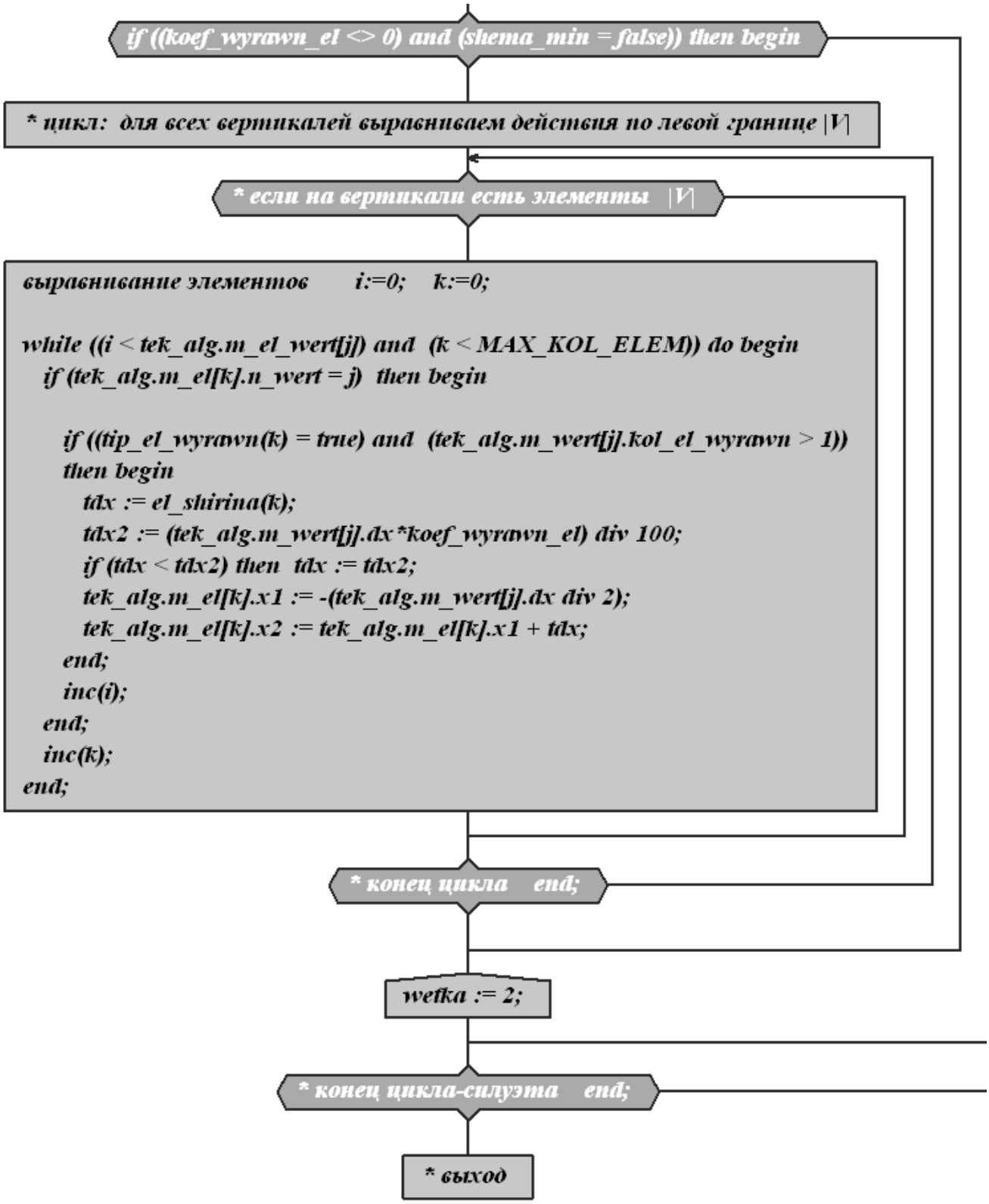

Рисунок 2.14 – Окончание ветки 1 процедуры podschet_max_koord

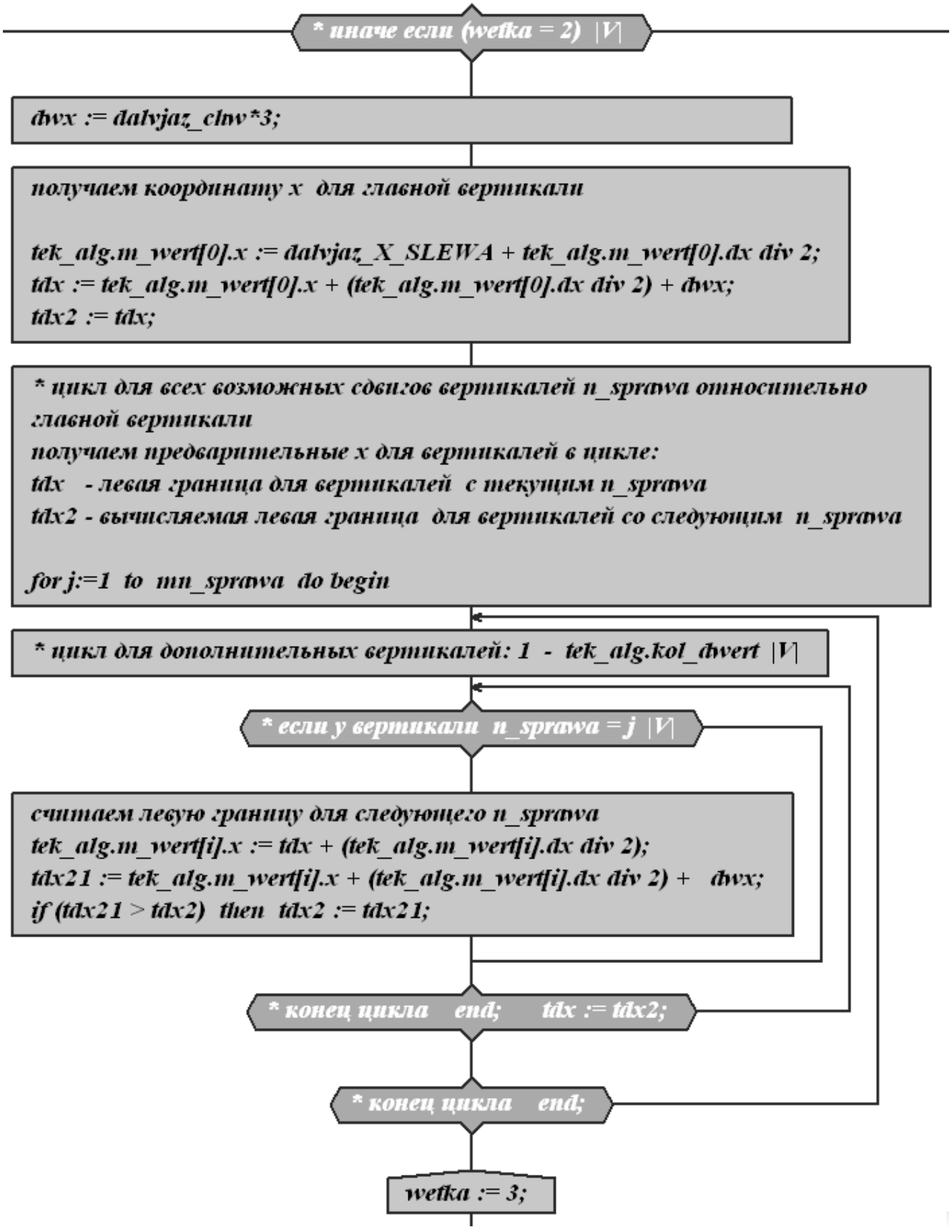

Рисунок 2.15 – Ветка 2 процедуры podschet_max_koord

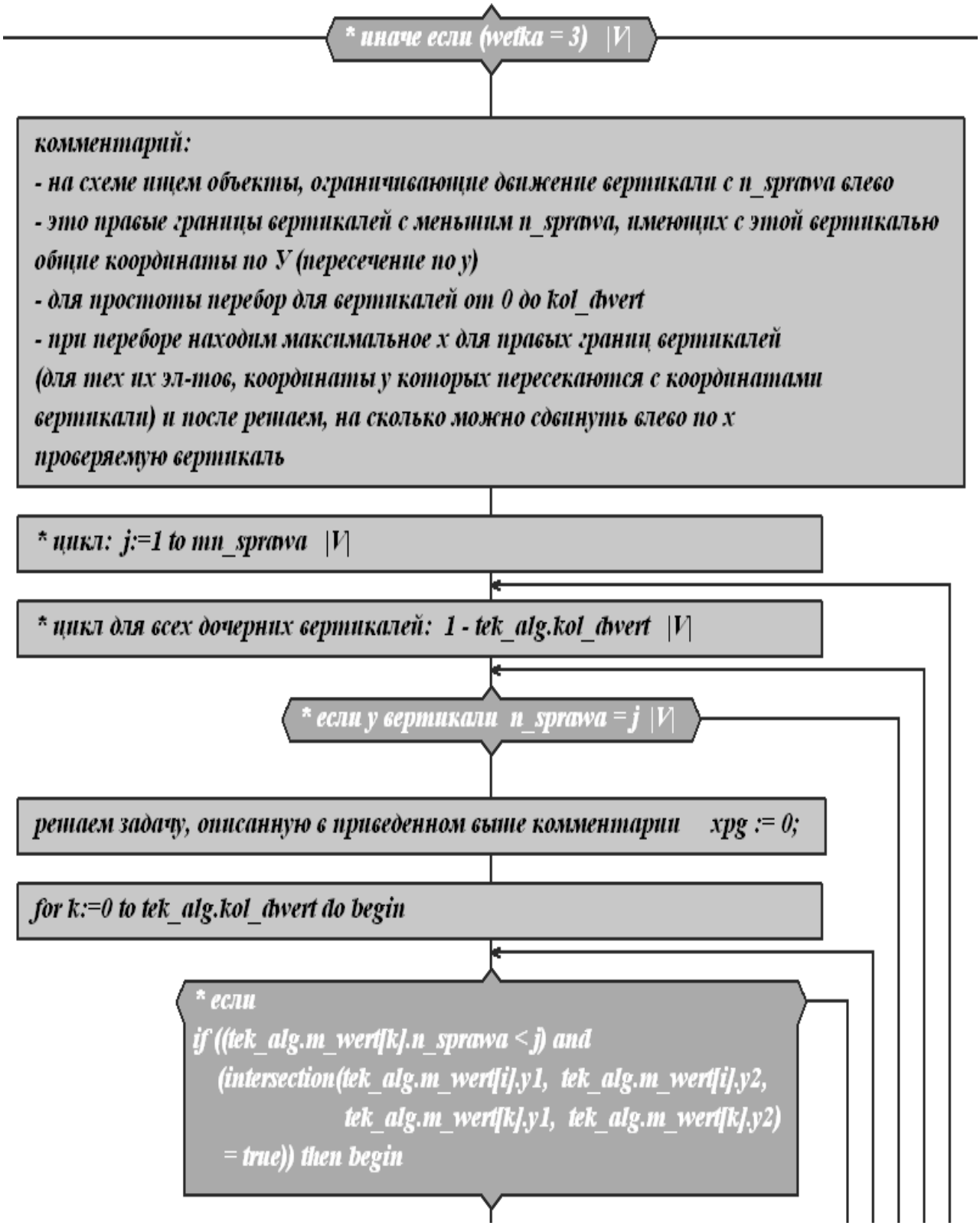

Рисунок 2.16 – Начало ветки 3 процедуры podschet_max_koord

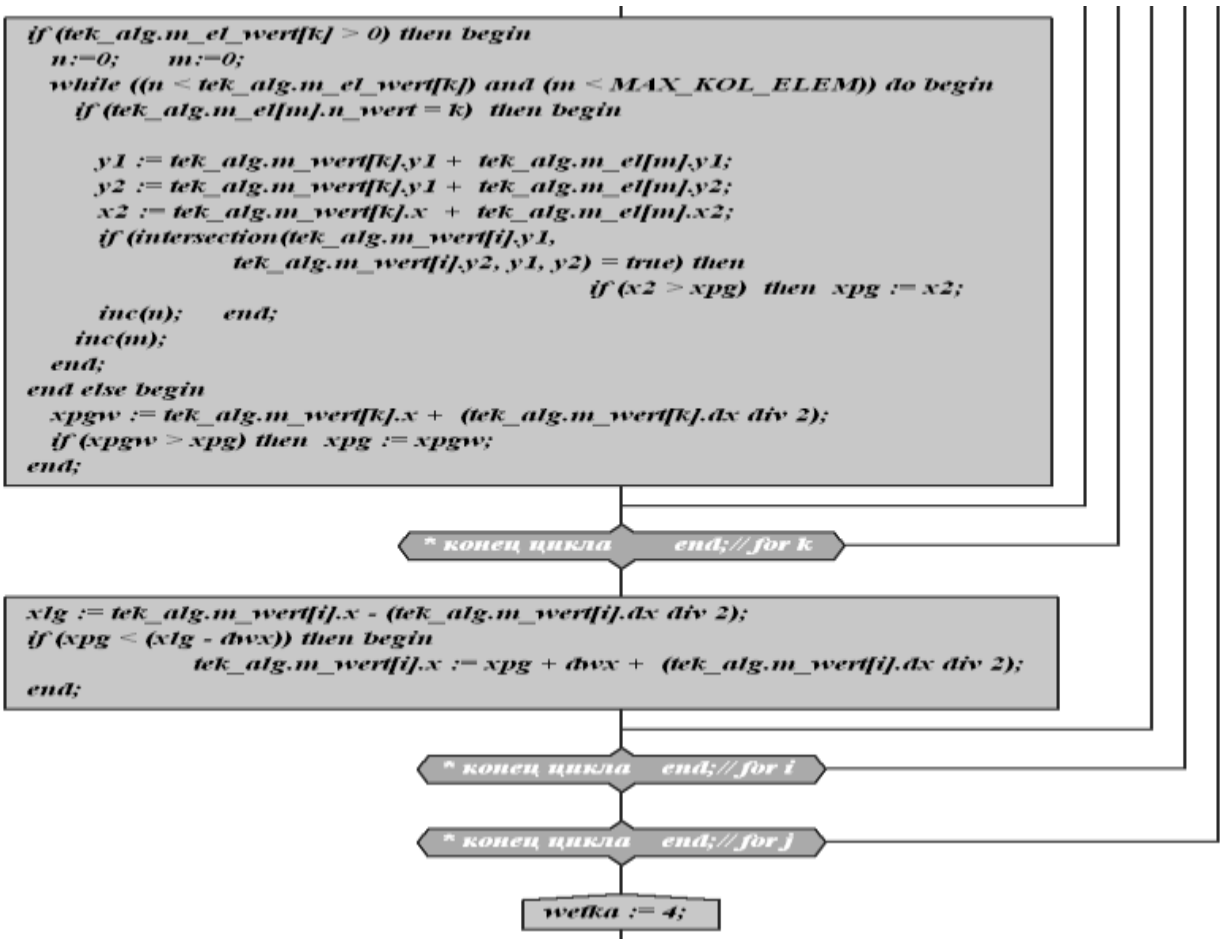

Рисунок 2.17 – Окончание ветки 3 процедуры podschet_max_koord

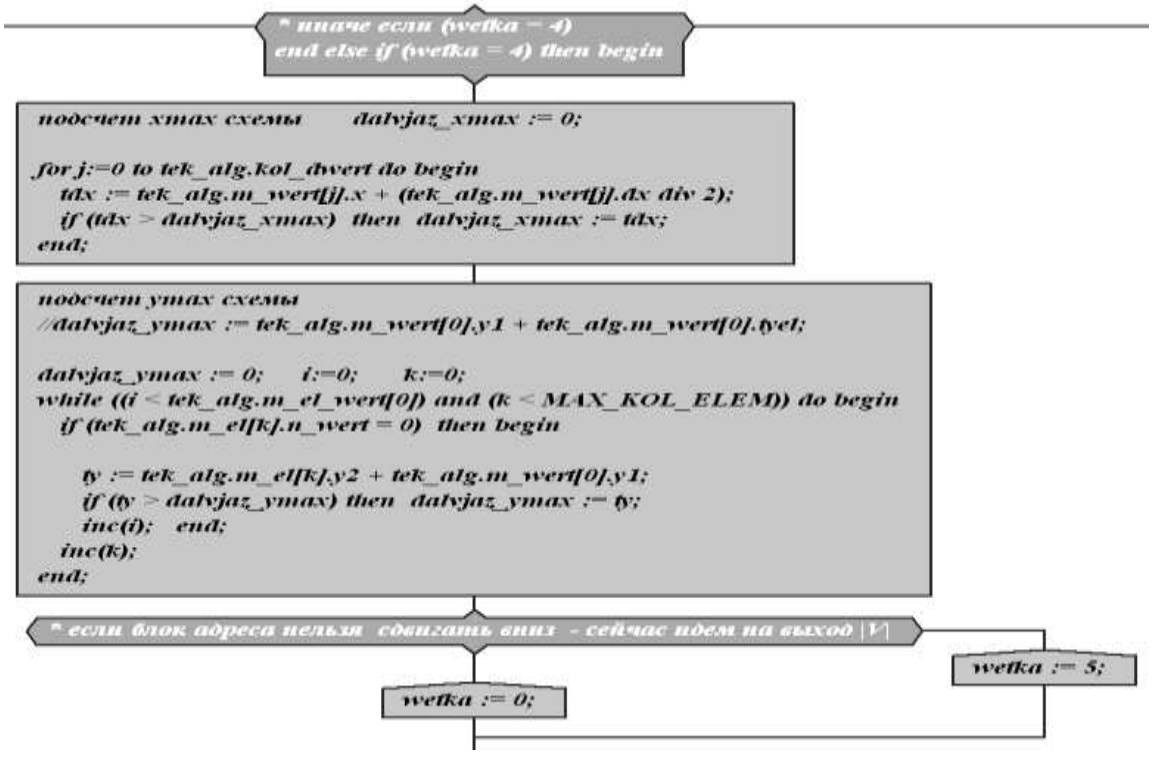

Рисунок 2.18 – Ветка 4 процедуры podschet_max_koord

Выполнение процедуры podschet_max_koord завершает формирование координат схемы, после чего ее можно выводить на экран.

2.12. Создание текущей версии процедуры вывод_схемы, окончание

Путем превращения процедуры вывод_схемы в процедуру ЛВУ и переноса ее на Компонентный паскаль была получена версия №2:

```
(* i.1 вывод_схемы  |V|*)

001  ПРОЦЕДУРА  вывод_схемы;
002  ПЕРЕМЕННЫЕ ж, ии, и, смещение_эл_х, х1,х2,у1,у2,
003             вэкв, эздв:ЦЕЛЫЙ;
004  НАЧАЛО
005     ЕСЛИ (запрос(З_СХЕМА_НАЧАЛО_ПРОГРАММЫ) = ОТВЕТ_ДА) ТО
006        (**) ВЫХОД;    (*[-]*)   (* РИС .r? *)
007     КОНЕЦ;
008
009     команда(К_СХЕМА_ПОЛУЧИТЬ_РАЗМЕРЫ_КАНВЫ);
010     ЕСЛИ (запрос(З_СХЕМА_ТЕКСТОВАЯ_ЗАПИСЬ) = ОТВЕТ_ДА) ТО
011        команда(К_СХЕМА_ВЫВОД_ТЕКСТА);
012        команда (К_СХЕМА_ОБНОВИТЬ_КАНВУ);
013        (**) ВЫХОД;    (*[-]*)   (* РИС .r? *)
014     КОНЕЦ;
015
016     команда(К_СХЕМА_СТИЛЬ__СОЕДИНИТЕЛЬНЫЕ_ЛИНИИ);
017     команда (К_СХЕМА_СТИЛЬ__ЦВЕТ_ФОНА);
018     команда (К_СХЕМА_ФОН_СХЕМЫ);
019     ЕСЛИ (запрос(З_СХЕМА_ПОДСЧЕТ_МАКС_КООРДИНАТ) = ОТВЕТ_НЕТ) ТО
020        команда(К_СХЕМА_ОБНОВИТЬ_КАНВУ);
021        (**) ВЫХОД;    (*[-]*)   (* РИС .r? *)
022     КОНЕЦ;
023
024     (** вывод главной вертикали *)
025         ж :=0;
026         ии :=значение(В_СХЕМА_ИНД_1_ЭЛ_ВЕРТИКАЛИ, ж);
027         и :=0;
028     ЦИКЛ_ПОКА (и < значение(В_СХЕМА_КОЛ_ЭЛ_ВЕРТИКАЛИ, ж)) ДЕЛАТЬ
029        ЕСЛИ (значение(В_СХЕМА_ВЕРТИКАЛЬ_ЭЛЕМЕНТА, ии) = ж) ТО
030           ЕСЛИ (значение(В_СХЕМА_ЭЛЕМЕНТ_ИСПОЛЬЗУЕТСЯ, ии) = ОТВЕТ_ДА) ТО
031              данные(Д_СХЕМА_ВЕРТИКАЛЬ, ж);
032              данные (Д_СХЕМА_ИНДЕКС_ЭЛЕМЕНТА, ии);
033              данные (Д_СХЕМА_ТЕК_ЭЛ_ВЕРТИКАЛИ, и);
034              команда (К_СХЕМА_ВЫВОД_ЭЛЕМЕНТА);
035              УВЕЛИЧИТЬ(и);
036           КОНЕЦ;
037        КОНЕЦ;
038
039        УВЕЛИЧИТЬ(ии);
040     КОНЕЦ_ЦИКЛА;
041     ЦИКЛ_ДЛЯ  ж := 0 ДО запрос(З_СХЕМА_КОЛ_ДОП_ВЕРТИКАЛЕЙ)-1  ДЕЛАТЬ
042     (* цикл вывода элементов дополнительных вертикалей *)
043        смещение_эл_х := 0;
044        ЕСЛИ ((значение(В_СХЕМА_РОДИТЕЛЬ_ВЕРТИКАЛИ, ж) # НЕ_ЗАДАН) &
045              (значение(В_СХЕМА_ТИП_ДОП_ВЕРТИКАЛИ, ж) # ВЕРТИКАЛЬ_ЦИКЛА)) ТО
046           смещение_эл_х := значение(В_СХЕМА_ШИРИНА_РОДИТЕЛЯ_ВЕРТИКАЛИ, ж);
047        КОНЕЦ;
048
049        х1 := значение(В_СХЕМА_Х_ВЕРТИКАЛИ_РОДИТЕЛЯ, ж) +
050              запрос(З_СХЕМА_СМЕЩЕНИЕ_СХЕМЫ_ПО_Х) + (смещение_эл_х ДЕЛИТЬ 2);
051        х2 := запрос(З_СХЕМА_СМЕЩЕНИЕ_СХЕМЫ_ПО_Х) +
052                                    значение(В_СХЕМА_Х_ДОП_ВЕРТИКАЛИ,ж);
053        у1 := запрос(З_СХЕМА_СМЕЩЕНИЕ_СХЕМЫ_ПО_У) +
054                                    значение(В_СХЕМА_У1_ДОП_ВЕРТИКАЛИ,ж);
055        ЕСЛИ (значение(В_СХЕМА_ТИП_ДОП_ВЕРТИКАЛИ,ж) = ВЕРТИКАЛЬ_ДОЧЬ) ТО
056           ЕСЛИ (значение(В_СХЕМА_ТИП_1_ЭЛ_ДОП_ВЕРТИКАЛИ,ж) = т_ИНЕСЛИ) ТО
057              ЕСЛИ (значение(В_СХЕМА_ФЛАГ_ВЕТКИ_1_ЭЛ_ДОП_ВЕРТИКАЛИ,ж)=ОТВЕТ_ДА) ТО
058                 смещение_эл_х := значение(В_СХЕМА_ШИРИНА_1_ЭЛ_ДОП_ВЕРТИКАЛИ,ж);
```

```
059                x2 := x2 - (смещение_эл_х ДЕЛИТЬ 2);
060             КОНЕЦ;
061          КОНЕЦ;
062
063          данные(Д_СХЕМА_КООРД_Х1, х1);   данные (Д_СХЕМА_КООРД_У1, у1);
064          данные (Д_СХЕМА_КООРД_Х2, х2);   данные (Д_СХЕМА_КООРД_У2, у1);
065          команда (К_СХЕМА_ЛИНИЯ);
066          вэкв := значение(В_СХЕМА_ВЕРТИКАЛЬ_ЭЛ_КОНЦА_ВЕРТИКАЛИ, ж);
067          х1 := запрос(З_СХЕМА_СМЕЩЕНИЕ_СХЕМЫ_ПО_Х) +
068                                    значение(В_СХЕМА_Х_ВЕРТИКАЛИ,вэкв);
069          х2 := запрос(З_СХЕМА_СМЕЩЕНИЕ_СХЕМЫ_ПО_Х) +
070                                    значение(В_СХЕМА_Х_ДОП_ВЕРТИКАЛИ, ж);
071          у2 := запрос(З_СХЕМА_СМЕЩЕНИЕ_СХЕМЫ_ПО_У) +
072                значение(В_СХЕМА_У1_ДОП_ВЕРТИКАЛИ,ж) +
073                значение(В_СХЕМА_ТЕК_У_ДОП_ВЕРТИКАЛИ,ж);
074          данные(Д_СХЕМА_КООРД_Х1, х1);   данные (Д_СХЕМА_КООРД_У1, у2);
075          данные(Д_СХЕМА_КООРД_Х2, х2);   данные (Д_СХЕМА_КООРД_У2, у2);
076          команда (К_СХЕМА_ЛИНИЯ);
077          ЕСЛИ (значение(В_СХЕМА_КОЛ_ЭЛ_ДОП_ВЕРТИКАЛИ, ж) # 0) ТО
078             ЕСЛИ (значение(В_СХЕМА_ТИП_1_ЭЛ_ДОП_ВЕРТИКАЛИ,ж) # т_ИНАЧЕ) ТО
079                ЕСЛИ (значение(В_СХЕМА_ТИП_1_ЭЛ_ДОП_ВЕРТИКАЛИ,ж) # т_ИНЕСЛИ) ИЛИ
080                 (значение(В_СХЕМА_ФЛАГ_ВЕТКИ_1_ЭЛ_ДОП_ВЕРТИКАЛИ,ж)=ОТВЕТ_НЕТ) ТО
081                   у1 := запрос(З_СХЕМА_СМЕЩЕНИЕ_СХЕМЫ_ПО_У) +
082                         значение(В_СХЕМА_У1_ДОП_ВЕРТИКАЛИ,ж);
083                   у2 := у1 + значение(В_СХЕМА_У1_1_ЭЛ_ДОП_ВЕРТИКАЛИ,ж);
084                   данные(Д_СХЕМА_КООРД_Х1, х2);  данные (Д_СХЕМА_КООРД_У1, у1);
085                   данные (Д_СХЕМА_КООРД_Х2, х2);  данные (Д_СХЕМА_КООРД_У2, у2);
086                   команда (К_СХЕМА_ЛИНИЯ);
087                ЕСЛИ;
088             КОНЕЦ;
089          КОНЕЦ;
090       КОНЕЦ;
091
092       ЕСЛИ (значение(В_СХЕМА_ТИП_ДОП_ВЕРТИКАЛИ,ж) = ВЕРТИКАЛЬ_ЦИКЛА) ТО
093          у1 := значение(В_СХЕМА_ВЫШЕ_ЗАГОЛ_ВЕТОК, у1);
094          данные(Д_СХЕМА_КООРД_Х1, х1);   данные (Д_СХЕМА_КООРД_У1, у1);
095          данные (Д_СХЕМА_КООРД_Х2, х2);   данные (Д_СХЕМА_КООРД_У2, у1);
096          команда (К_СХЕМА_ЛИНИЯ);
097          данные(Д_СХЕМА_КООРД_Х1, х1);
098          данные (Д_СХЕМА_КООРД_У1, у1);
099          команда (К_СХЕМА_СТРЕЛКА_ЦИКЛА);
100          у2 := у1 + значение(В_СХЕМА_ТЕК_У_ДОП_ВЕРТИКАЛИ,ж) +
101                     (запрос(З_СХЕМА_ШАГ_ТЕКСТА_ПО_У) ДЕЛИТЬ 4);
102
103          ээдв:= значение(В_СХЕМА_ЭЛ_ЗАКРЫВАЮЩИЙ_ДОП_ВЕРТИКАЛЬ,ж);
104                           (* КОНЕЦ/КОНЕЦ_ЦИКЛА *)
105          смещение_эл_х := значение(В_СХЕМА_ШИРИНА_ЭЛЕМЕНТА, ээдв);
106          х1 := х1 + (смещение_эл_х ДЕЛИТЬ 2);
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       КОНЕЦ;
111
112       ЕСЛИ (значение(В_СХЕМА_КОЛ_ЭЛ_ДОП_ВЕРТИКАЛИ, ж) = 0) ТО
113          у1 := запрос(З_СХЕМА_СМЕЩЕНИЕ_СХЕМЫ_ПО_У) +
114                значение(В_СХЕМА_У1_ДОП_ВЕРТИКАЛИ, ж);
115          у2 := у1 + значение(В_СХЕМА_ТЕК_У_ДОП_ВЕРТИКАЛИ, ж);
116          ЕСЛИ (значение(В_СХЕМА_ТИП_ДОП_ВЕРТИКАЛИ, ж)= ВЕРТИКАЛЬ_ЦИКЛА) ТО
117             у1 := значение(В_СХЕМА_ВЫШЕ_ЗАГОЛ_ВЕТОК, у1);
118          КОНЕЦ;
119          данные(Д_СХЕМА_КООРД_Х1, х2);   данные (Д_СХЕМА_КООРД_У1, у1);
120          данные (Д_СХЕМА_КООРД_Х2, х2);   данные (Д_СХЕМА_КООРД_У2, у2);
121          команда (К_СХЕМА_ЛИНИЯ);
122       ИНАЧЕ
123          ии := значение(В_СХЕМА_ИНД_1_ЭЛ_ДОП_ВЕРТИКАЛИ, ж);
124          и := 0;
125          ЦИКЛ_ПОКА (и < значение(В_СХЕМА_КОЛ_ЭЛ_ДОП_ВЕРТИКАЛИ, ж)) ДЕЛАТЬ
126             ЕСЛИ (значение(В_СХЕМА_ВЕРТИКАЛЬ_ЭЛЕМЕНТА, ии) =
127                   значение(В_СХЕМА_ДОП_ВЕРТИКАЛЬ, ж)) ТО
128                ЕСЛИ (значение(В_СХЕМА_ЭЛЕМЕНТ_ИСПОЛЬЗУЕТСЯ, ии) = ОТВЕТ_ДА) ТО
```

```
129                    данные(Д_СХЕМА_ВЕРТИКАЛЬ, значение(В_СХЕМА_ДОП_ВЕРТИКАЛЬ, ж));
130                    данные (Д_СХЕМА_ИНДЕКС_ЭЛЕМЕНТА, ии);
131                    данные (Д_СХЕМА_ТЕК_ЭЛ_ВЕРТИКАЛИ, и);
132                    команда (К_СХЕМА_ВЫВОД_ЭЛЕМЕНТА);
133                    УВЕЛИЧИТЬ(и);
134                 КОНЕЦ;
135              КОНЕЦ;
136
137              УВЕЛИЧИТЬ(ии);
138           КОНЕЦ_ЦИКЛА;
139        КОНЕЦ;
140     КОНЕЦ_ЦИКЛА_ДЛЯ;
141     команда(К_СХЕМА_ОБНОВИТЬ_КАНВУ);
142     (** ВЫХОД *)
143     КОНЕЦ  вывод_схемы;
144  (** КОНЕЦ ПРОЦЕДУРЫ *)
```

Все действия по превращению версии №1 в версию №2 выполнялись в текстовом редакторе и к силуэтному программированию отношения не имеют, в отличие от последующих действий по превращению версии №2 в версию №3.

После загрузки версии №2 в программу dal_vjaz_2 была построена схема этой процедуры, минимизированная форма которой для удобства восприятия была разделена на четыре фрагмента, размещенные на рис. 2.19 слева направо.

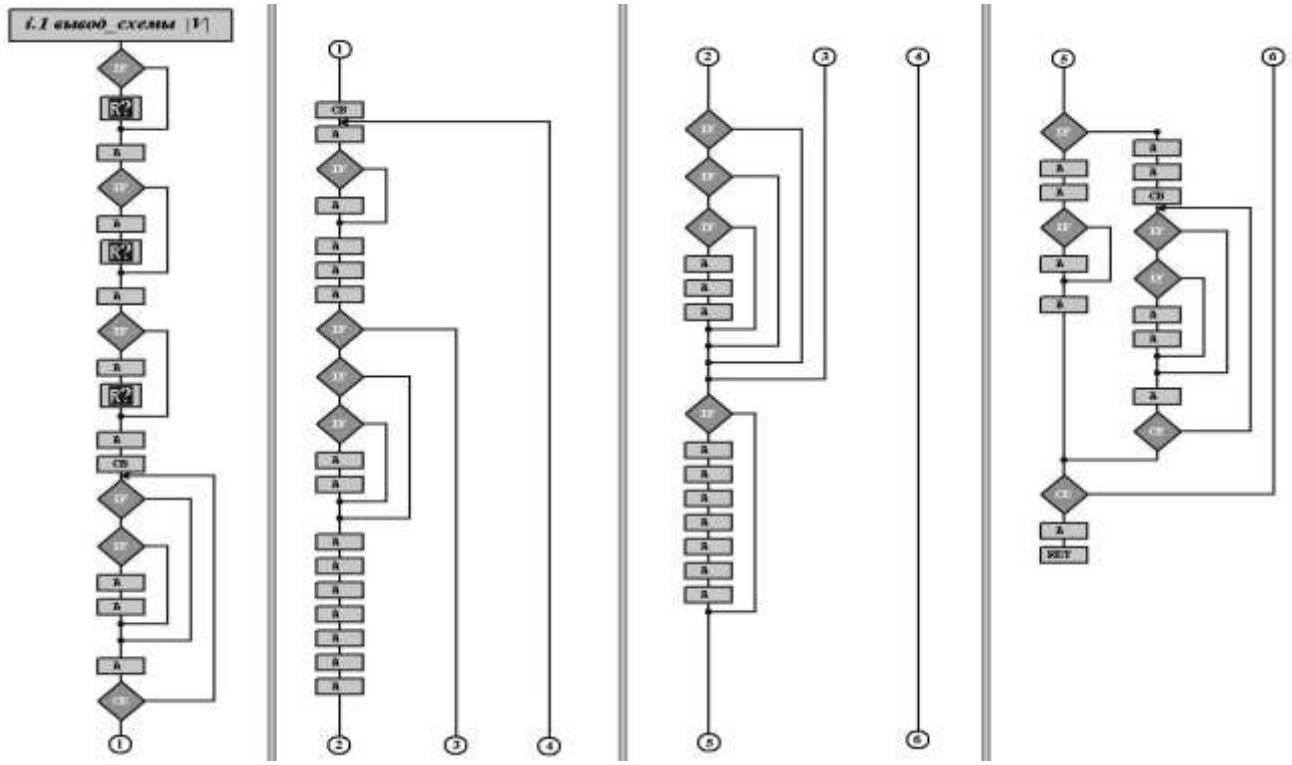

Рисунок 2.19 – Разбитая на 4 фрагмента минимизированная форма версии №2 процедуры вывод_схемы

Нужно превратить схему, показанную на рис. 2.19, в схему на рис. 1.16. Что для этого нужно сделать ?

Все просто – для поучения версии №3 процедуры `вывод_схемы` нужно создать сразу после заголовка процедуры, показанной на рис. 2.19, цикл сложного условия с несколькими ветками, используя кнопки  ,  и  , а затем начать перетаскивать туда путем вырезания  и вставки  логику процедуры. Главный цикл процедуры, обратная петля которого на рис. 2.19 обозначена кружками 4 и 6, нужно превратить в веточный цикл, а альтернативные ветки некоторых условий нужно распределить между различными ветками цикла сложного условия.

В результате должна получиться версия №3 процедуры `вывод_схемы`, показанная на рисунках 1.4 – 1.16.

А теперь еще раз и более подробно.

Итак, для того, чтобы преобразовать версию №2 процедуры `вывод_схемы` в версию №3, нужно выполнить следующую последовательность действий:

1) Скопировать в пустой текстовый файл текст процедуры `вывод_схемы`, приведенной в начале пункта 2.12.

2) Удалить в этом текстовом файле номера строк и выровнять отступы для всех строк (кроме 1-й строки «(* i.1 вывод_схемы |V|*)», которая должна начинаться с первой позиции), чтобы все они начинались с одной позиции.

3) Скопировать содержимое этого текстового файла в клипборд Windows (Ctrl+A, Ctrl+C).

4) Запустить программу dal_vjaz_2.

5) Выбрать кнопку «Счит.» на панели редактирования схемы.

6) В открывшееся окно ввода скопировать текст процедуры и нажать кнопку «Сохранить».

7) На панели редактирования схемы установить флажок «мин» и убедиться, что минимизированная форма схемы совпадает с приведенной на рис. 2.19.

8) Скопировать в отдельный графический файл или распечатать на принтере рис. 1.16, на котором приведена минимизированная форма схемы для версии №3 процедуры `вывод_схемы`.

9) Нажать кнопку  , после чего щелкнуть левой кнопкой мыши по заголовку схемы – в результате сразу после заголовка схемы будет вставлен пустой цикл.

10) Вставляя в этот цикл сложные условия и действия, а также присваивая в нужных действиях новые значения переменной `ветка` добиться, чтобы цикл стал выглядеть аналогично циклу сложного условия на рис. 1.16.

Сначала нужно вставить в цикл «* если», а в «*если» вставить 11 веток «* иначе если», после чего последовательно слева направо заполнить все созданные ветки нужными элементами, как на рис. 1.16.

Чтобы вставить элемент после конца условия, нужно щелкнуть левой кнопкой мыши по кружку конца условия.

Чтобы вставить элемент в ветку «* иначе» условия, нужно щелкнуть левой кнопкой мыши по кружку начала вертикали «* иначе».

При вводе новых номеров веток можно скопировать выражение «ветка := XX;», а затем вставлять его в нужные действия, меняя после вставки задаваемый номер ветки.

11) После того, как цикл сложного условия примет вид как на рис. 1.16, нужно перейти к заполнению условий в начале веток сложного условия.

Для первой ветки нужно ввести «ЕСЛИ (ветка = 10) ТО», а для остальных веток ввести «ИНЕСЛИ (ветка = XX) ТО», где XX равен 20, 30, …, 120.

Сразу после заголовка схемы, перед циклом сложного условия нужно вставить действие «ветка := 10;».

12) Теперь нужно заполнить цикл содержимым, которое сейчас как гирлянда висит ниже блока конца цикла сложного условия. Тут возможны несколько вариантов действий:

- поблочно копировать тексты для группы блоков из гирлянды в сложное условие, а затем блоки гирлянды, откуда данные уже были скопированы, удалять;

- переносить блоки путем вырезания и вставки в сложное условие, а их аналоги в сложном условии удалять;

- копировать и вставлять блоки в сложное условие, не удаляя их из гирлянды, а их аналоги в сложном условии удалять; но тут нужно быть внимательным, чтобы не скопировать еще раз из гирлянды уже скопированные блоки; впрочем, можно удалять сразу несколько групп уже скопированных блоков;

- заносить в блоки сложного условия соответствующий этим блокам текст из процедуры, приведенной в пункте 1.6 – ее структура соответствует сложному условию, поэтому разобраться, в какой блок что нужно копировать, будет несложно; для этого нужно скопировать в пустой текстовый файл текст процедуры `вывод_схемы`, приведенной в начале пункта 1.6, после чего, ориентируясь по номерам строк, начать переносить текст из текстового файла в нужные блоки (впрочем, можно копировать текст в блоки прямо из pdf-файла).

При копировании текста в блоки сложного условия можно по очереди попробовать все вышеуказанные варианты.

К моменту начала копирования заголовок схемы уже почти заполнен – нужно еще добавить туда определение переменной `ветка`.

Нужно быть внимательным при переносе в ветку 70 последнего из относящихся к нему действий из гирлянды – в гирлянде оно разбито на 2 действия за счет строки 075 процедуры `вывод_схемы` из пункта 2.12. Поэтому строки 075 и 076 надо вставить в предыдущий блок ветки 70, а сам блок при этом не копировать.

Для ветки 80 (3 условия, в 3-м условии 2 действия) в гирлянде есть 3 действия – два первых действия из гирлянды нужно объединить в одно: строки 081, 082 и 083 процедуры из пункта 2.12, да в начало нужно еще добавить строку «(** получаем y1 и y2 |V| *)». Да и для второго действия … - короче говоря, тексты этих действий лучше скопировать из строк 119 – 123 и 124 – 129 процедуры из пункта 1.6

В ветке 120 в первое действие нужно вставить строку 199, в условие – строку 200, а в 1-е действие ветки «* иначе» - строку 203 процедуры из пункта 1.6

13) После окончания заполнения блоков сложного условия текстом остатки гирлянды удаляем.

14) Нажимаем кнопку «Сохр.» на панели редактирования схемы.

15) Копируем в пустой текстовый файл (по Ctrl+V или по Shift+Ins) результаты редактирования схемы.

16) Выходим из программы dal_vjaz_2.

17) Сверяем полученный текст процедуры с текстом процедуры из пункта 1.6. При сверке можно обратить внимание, что в сгенерированном тексте кое-где присутствуют пустые ветки «* иначе», но это не имеет значения для правильности логики процедуры.

Кое-где могут отличаться комментарии в начале действий, т.к. были скопированы комментарии из предыдущей версии – в этом случае комментарии в полученном текстовом файле нужно исправить. Если во вставленном комментарии есть признак |V|, то из второй строки текста блока нужно исключить признак альтернативного начала действия, если этот признак там есть.

Конечно, при сравнении текстов могут обнаружится и ошибки: например, если вместо текста одного условия был скопирован текст другого, но такие ошибки относятся к области опечаток.

18) Запоминаем изменения, внесенные в полученный текстовый файл (я сохранил этот отредактированный текстовый файл под именем proc3.txt – он находится в рабочем каталоге программы dal_vjaz_2), после чего копируем полученное содержимое текстового файла в клипборд Windows.

19) Запускаем программу dal_vjaz_2 и копируем в нее содержимое отредактированного текстового файла.

Теперь при полной форме отображения схемы, если разрешение экрана монитора по высоте равно 1050 или больше (а если отступ схемы сверху в файле конфигурации dal2.cfg задать равным 20 вместо 50, то хватит и разрешения экрана по высоте 1024), в самом низу окна программы dal_vjaz_2 отображается блок конца цикла сложного условия, а это означает, что для полной формы отображения схемы сформированная из версии №3 процедуры `вывод_схемы` схема по высоте умещается в один экран, что и требуется для просмотра логики этой процедуры с использованием только горизонтальной полосы прокрутки (или, что функционально одно и то же, путем горизонтального перетаскивания схемы в окне программы dal_vjaz_2 мышью при нажатой правой кнопке мыши).

Заключение

Чтобы сделать процесс ознакомления с языком ДАЛВЯЗ 2 более наглядным, вместе с материалом «Алгоритмический язык ДАЛВЯЗ 2. Силуэтное программирование» выкладываю и версию программы dal_vjaz_2 0.87.4, для которой и были получены скриншоты схем процедур, представленные в данном материале.

Программа dal_vjaz_2 0.87.4 является свободно распространяемым ПО, создана для иллюстрации предлагаемого вниманию читателей материала и выкладывается «как есть», с возможностью дальнейшего развития и совершенствования всеми желающими.

Версия программы dal_vjaz_2 0.87.4 написана на Delphi 4, за исключением процедуры `вывод_схемы`, которая написана на русифицированной версии Компонентного паскаля и оттранслирована в dll-библиотеку v0874.dll, находящуюся в рабочем каталоге программы dal_vjaz_2. Вопросы взаимодействия КП и Delphi поясняются в теме http://forum.oberoncore.ru > Визуальное программирование > КП + Delphi.

ЛИТЕРАТУРА И ССЫЛКИ В СЕТИ ИНТЕРНЕТ

1. В.Д. Пароджанов «ЯЗЫК ДРАКОН краткое описание»
DrakonDescription.pdf

2. С. Макконнелл Совершенный код. Мастер-класс / Пер. с англ. -
М. : Издательско-торговый дом <Русская Редакция> ; СПб.: Питер, 2005.

3. Д. Барановский «ДАЛВЯЗ 2 – описание»
dalvjaz2_description.pdf

4. http://forum.oberoncore.ru