

1. Алгоритм и его свойства. Способы описания алгоритмов. Пример

Алгоритмом называется система правил, четко описывающая последовательность действий, которые необходимо выполнить для решения задачи.

Алгоритмизация - сведение задачи к последовательности этапов, выполняемых друг за другом так, что результаты предыдущих этапов используются при выполнении следующих.

Свойства правильно разработанного алгоритма:

- 1) **Дискретность** – Дискретность алгоритма означает, что в алгоритме могут быть выделены отдельные шаги, причём выполняться они должны последовательно, один за другим, и каждый следующий шаг должен использовать результаты, полученные на предыдущих шагах.

Лучший пример реализации недискретного алгоритма — водитель маршрутки, пытающийся одновременно вести транспортное средство, принимать оплату за проезд, разговаривать по телефону, курить и мысленно планировать выходные, причём используя в планах на выходные деньги, которые заработает только в следующем рейсе.

(Справедливости ради, в данном случае правильнее говорить о нескольких параллельно обрабатываемых алгоритмах.)

- 2) **Определенность(детерминированность)** – каждое правило алгоритма должно быть четким и однозначным.

«В понедельник и среду врач работает в первую смену, во вторник и четверг — во вторую». В повседневной жизни на случай пятницы, субботы и воскресенья обычно есть какое-то подразумевающееся решение: например, в эти дни у врача выходной. При разработке алгоритмов для программ создать ситуацию неопределённости намного проще: «Если $x > 0$, то оставить его без изменений, если $x < 0$, изменить знак на противоположный» — что делать, если $x = 0$?

- 3) **Результативность(конечность)** – алгоритм должен приводить к решению задачи за конечное число шагов.

Пример алгоритма, не являющегося результативным:

- 1) набрать воду в электрический чайник;
- 2) включить чайник;
- 3) посидеть на YouTube 2–3 часа за просмотром видео с котятками;

- 4) подойти к чайнику;
 - 5) если вода в чайнике холодная, перейти к пункту 2;
 - 6) заварить чай и выпить его.
- 4) Массовость – алгоритм позволяет решить поставленную задачу при всех возможных исходных данных, предусмотренных задачей.
- Например, если написанный по Вашему алгоритму калькулятор складывает только чётные числа, то о массовости речи идти не будет

Способы написания алгоритма:

1. Словесное - запись на естественном языке.
2. Графическое - запись алгоритма в виде блок-схемы.
3. Составление программы - запись на алгоритмическом языке.

Пример: $y=|a+b|$

а) *словесное написание:*

Прим1

1. Sum:=a+b;
2. Если Sum>=0, идти к 5;
3. y:= -Sum;
4. Идти к 6;
5. y:= Sum;
6. Остановка;

Недостаток словесного описания – малая наглядность

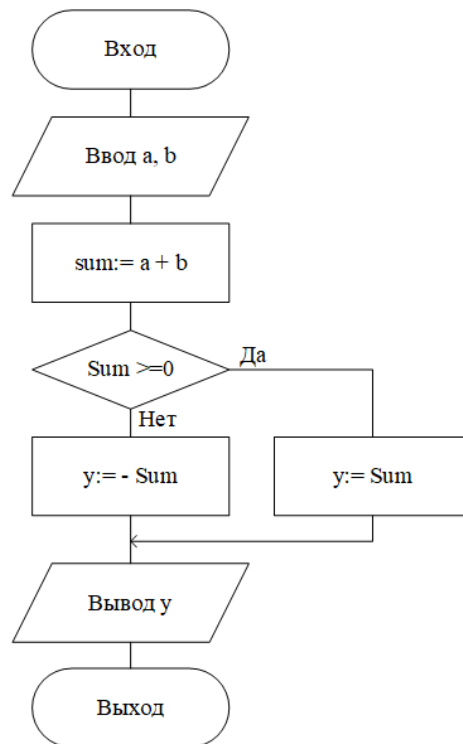
б) *графическое написание:*

-- ISO 5807-85

-- ГОСТ 19.701-90 – Единая система программной документации –

Схемы алгоритмов, программ, данных и систем – Условные обозначения и правила выполнения.

Прим2:



Прим3: ну на алгоритмическом, в своих там дельфинских, думаю напишете сами)))

2. Графическое представление алгоритмов по ГОСТ 19.701-90. Пример.

<http://vsegost.com/Catalog/28/28346.shtml>



Схема алгоритмов - графическое представление алгоритма, в котором этапы процесса обработки информации и носители информации представлены в виде геометрических символов, а последовательность процесса отображена направлением линий.

Виды схем

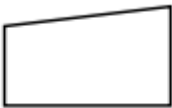

1. Схема данных
2. Схема программы
3. Схема работы систем
4. Схема взаимодействия программ
5. Схема ресурсов системы

1) Символы данных:

Основные:


	Данные		Используются для отображения ввода\вывода данных
	Запоминаемые данные		Символ отображает хранимые данные. Конкретный носитель данных не определен.

Специфические:



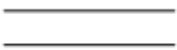



	Оперативное запоминающее устройство		Отображает данные, хранящиеся в оперативном запоминающем устройстве
	Запоминающее устройство с последовательным доступом		Этот символ используется для обозначения данных, которые находятся на запоминающем устройстве с последовательным доступом (магнитная лента)
	Запоминающее устройство с прямым доступом		Данные хранятся в прямом доступе (жесткий диск, флешка)
	Ручной ввод		Данные вводятся вручную во время обработки с устройства (с клавиатуры)
	Дисплей		Данные представляются в удобной форме для человека на отображаемом устройстве


2) Символы процесса:

Основные:

	Процесс		Отвечает за вычислительные операции
--	---------	---	-------------------------------------

Специфические:

	Предопределенный процесс		Отображает процесс, состоит из нескольких шагов программы, которые определены в другом месте (подпрограммы)
	Решение		Переключение типа, имеющий один вход и ряд выходов
	Параллельные действия		Несколько параллельный действий
	Граница цикла		Начало и конец цикла
	Ручная операция	 <small>Не путать с символом «Ручная стирка»!</small> 	обозначения этапов обработки данных, выполняемых человеком, т.е. самим пользователем программы или системы

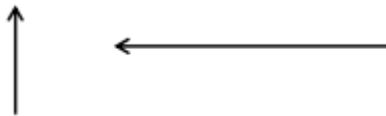
	Подготовка		Воздействие на последнюю функцию (для программ низкого уровня)
--	------------	---	--

3) Символы линий:

Основные:

3.1) Линия - символ отображает поток данных или управления.

При необходимости или для повышения удобочитаемости к линии могут быть добавлены стрелки

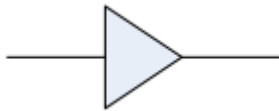


Специфические:

3.2) Пунктирная линия - используется для обведения выделяемого участка, а также как часть символа комментария.



3.3) Передача управления



3.4) Канал связи



4) Специальные символы

	Соединитель		Выход в другую часть схемы
	Терминал		Символ отображает выход во внешнюю среду и вход из внешней среды (в схемах алгоритмов – это начало или конец алгоритма)
	Комментарий		Используют для добавления комментариев (пояснительных записей)

	Пропуск	...	Отображения пропуска символа или группы символов
--	---------	-----	--

Правила применения символов:

- Символы должны быть расположены равномерно
 - Следует придерживаться минимального числа линий
 - Стандарт регламентирует форму символов
 - Размеры символов должны позволять включать текст внутрь символа
 - Не должны изменяться углы и другие параметры формы
 - По возможности символы должны быть одного размера
 - Символы могут быть вычерчены в вертикальной ориентации или зеркальном отображении (нежелательно)
 - Внутри символа следует помещать минимальное количество текста, необходимое для понимания его функции
 - Текст записывается слева направо, сверху вниз
 - Если текст не помещается внутрь символа, следует использовать комментарий
 - Линии показывают направление потока управления
 - Стандартное направление: сверху вниз, слева направо
 - Если управление потока отличается от стандартного, оно должно указываться стрелками
- Главная идея: Стремиться к наиболее понятному представлению алгоритма.

3. Разновидности структур алгоритмов. Виды циклов. Пример.

Структуры алгоритмов:

- линейные
- разветвляющиеся
- циклические

Линейный вычислительный процесс - процесс, в котором направление вычислений является единственным

$$\bullet \quad y = \sqrt[3]{\sin(x) + 2\cos(xz) + 3xz}$$

Начало

|

Ввод x, z

|

A := x * z

.....

|

Конец

Разветвляющийся вычислительный процесс - процесс, в котором направление вычислений определяется некоторыми условиями.

Циклический процесс - процесс, в котором отдельные участки вычислений выполняются многократно.

Цикл - участок схемы, многократно повторяемый в ходе вычислений.

- Простые
- Сложные (сложные разделяются на вложенные и внешние)

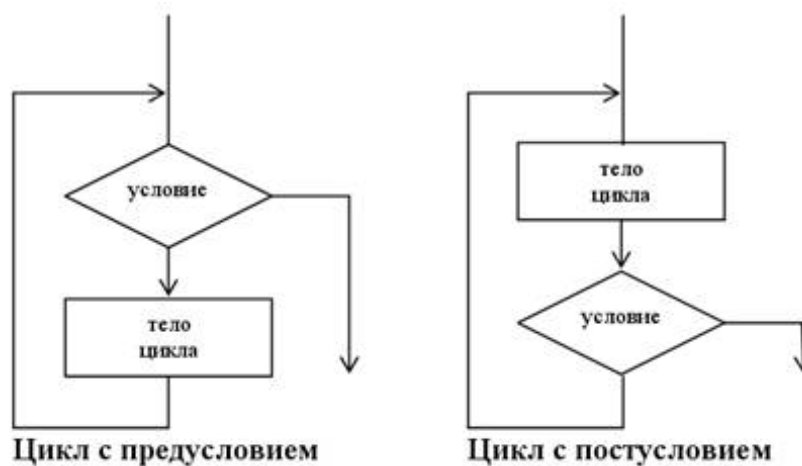
Циклы по местоположению условия выполнения цикла:

- с [предусловием](#)
- с [постусловием](#)

Циклы по виду условия выполнения цикла:

- циклы с параметром
- итерационные циклы

Схемы:



Цикл с параметром:

- цикл со счётчиком
- циклический процесс с известным количеством повторений

Пример: вычислить сумму

$$S = \sum_{i=1}^n x_i$$

Итерационный цикл - циклический процесс, в котором количество повторений заранее неизвестно и зависит от получающихся в ходе вычислений результатов.

Пример: вычисление синуса

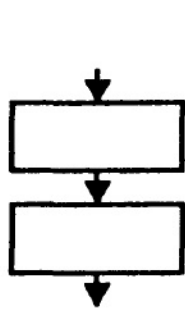
$$Y = x + \frac{x}{1!} + \frac{x^2}{2!} + \dots$$

В условии могут проверяться разность нового и старого значений либо значение прибавляемого элемента

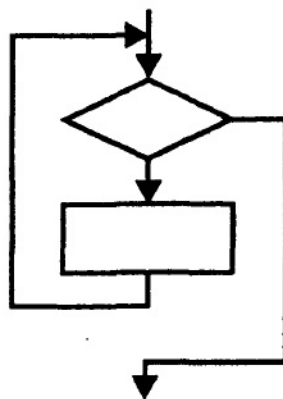
Простые алгоритмы могут быть использованы в качестве частей более сложных алгоритмов.

Пример:

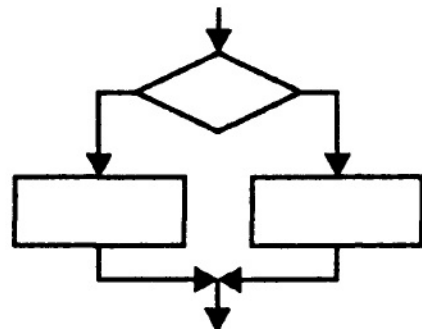
$$S = \sum_{i=1}^n \prod_{j=1}^m x_{ij}$$



Следование



Цикл



Ветвление

Базовые конструкции структурного программирования

4. Основные положения теории структурного программирования.

Пример.

Структурное программирование?

- подход к разработке программ
- возникло в 1970-х гг. (а ты до сих пор не умеешь) с появлением ЭВМ третьего поколения
- Структурное программирование — это одна из парадигм программирования, т.е. набор концепций, принципов разработки

программ. Основная идея структурного программирования заключается в следующем:

- Каждый модуль (участок) программы должен проектироваться с единственным входом и единственным выходом
- при этом программа состоит из вложенных модулей, удовлетворяющих этому требованию
- любая программа может быть разработана с использованием трёх базовых структур (принцип Бомы-Джакопини):
 - конструкция следования
 - конструкция двоичного решения (конструкция выбора)
 - конструкция обобщённого цикла

Каждый модуль (участок) программы должен проектироваться с единственным входом и единственным выходом

Вся программа при этом представлена вложенными друг в друга подобными модулями (частями), которые ещё называются функциональными блоками.

Основные концепции:

- отказ от использования оператора безусловного перехода goto
- применение фиксированного набора управляющих конструкций (3 базовых конструкции)
- использование метода нисходящего проектирования
- Обратные преобразования могут использоваться для проектирования по методу нисходящего программирования..
- Все операции в программе должны представлять собой:
 - исполняемые в линейном порядке выражения
 - вызовы подпрограмм (1 вход, 1 выход)
 - вложенные на произвольную глубину операторы if-then-else
 - циклические операторы (while)

Иногда допускаются расширения:

- дополнительные конструкции цикла:
 - цикл с параметром
 - цикл с постусловием
- оператор case как расширение if
- подпрограммы с несколькими входами и выходами
- оператор goto с жёсткими ограничениями

Достоинства:

- упрощение тестирования программ
- повышение производительности программистов
- повышение читаемости программ и их сопровождение
- повышение эффе
- +ивности объектного кода программ

5. Реализация теоретических основ структурного программирования в современных языках программирования. Достоинства структурного программирования. Пример.

Достоинства:

- упрощение тестирования программ
- повышение производительности программистов
- повышение читаемости программ - упрощается их сопровождение
- повышение эффективности объектного кода программ

Преобразования Бома-Джакопини

- Используются для доказательства:
 - правильности программы
 - структурированности программы
 - основано на принципе “чёрного ящика”

Реализация теоретических основ структурного программирования
базируется на следующих *правилах*.

Все операции в программе должны представлять собой либо непосредственно исполняемые в линейном порядке выражения, либо одну из следующих **управляющих конструкций**:

- 1) *вызовы подпрограмм* – любое допустимое на конкретном языке программирования обращение к замкнутой подпрограмме с одним входом и одним выходом;
- 2) вложенные на произвольную глубину операторы If-Then-Else;
- 3) *циклические операторы* (цикл с предусловием, называемый циклом «Пока»).

Этих средств достаточно для составления структурированных программ. Однако иногда допускаются их некоторые расширения:

1. дополнительные конструкции организации цикла:
 - 1.1. цикл с параметром как вариант цикла с предусловием;
 - 1.2. цикл с постусловием
2. подпрограммы с несколькими входами и несколькими выходами
 - 2.1. (например, один выход нормальный, второй – по ошибке);

3. применение оператора GoTo с жёсткими ограничениями
3.1. (например, передача управления не далее, чем на десять операторов, или только вперёд по программе);
4. использование оператора Case как расширения конструкции If-Then-Else

6. Преобразование неструктурированных программ в структурированные. Метод дублирования кодов.
Достоинства и недостатки метода. Пример.

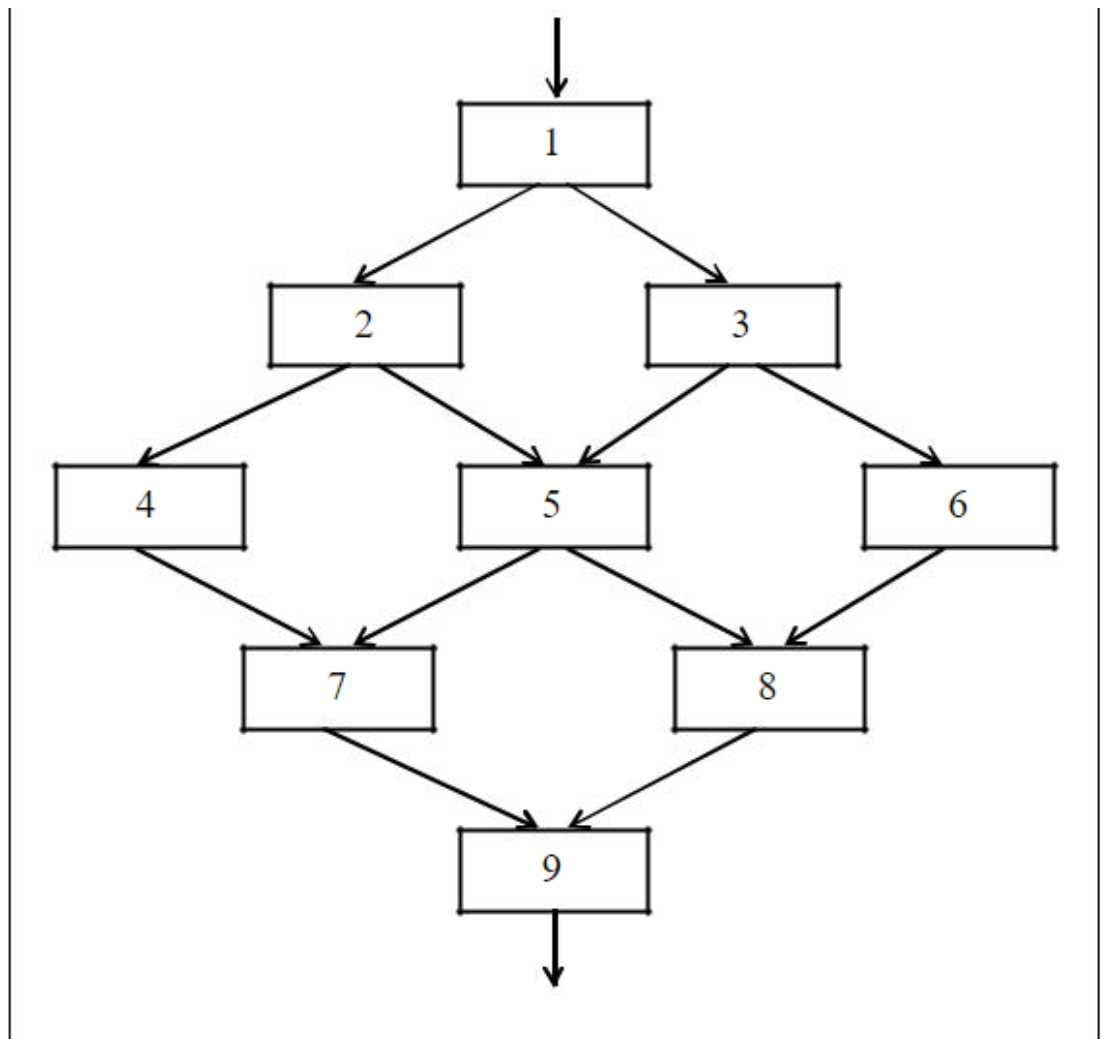
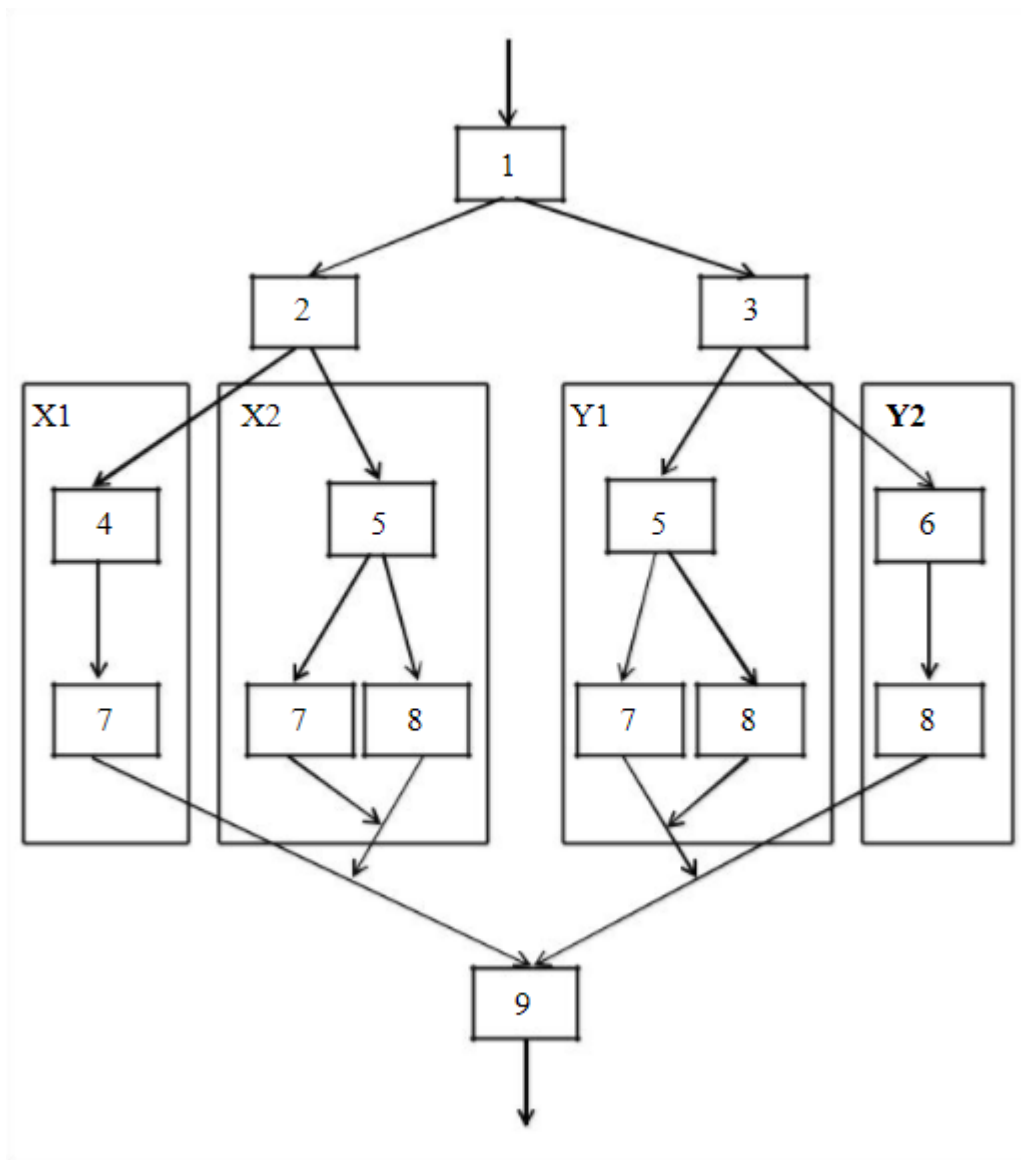


Рисунок 3.7 – Алгоритм неструктурированной программы типа «решетка»

Сущность метода дублирования кодов: дублируются те модули исходного алгоритма или программы, в которые можно войти из нескольких мест (кроме последнего блока).

Полученная схема алгоритма является структурированной.

Чтобы доказать это, необходимо воспользоваться преобразованиями **Бома-Джакопини**, т.е. последовательно преобразовать схему к одному функциональному блоку с одним входом и одним выходом.



Преобразованная схема по методу дублирования кодов
Преимущества

- Можно использовать как промежуточный шаг при разборе запутанного кода.

Недостатки

- Неприменим для циклических алгоритмов
- Увеличение объема кода

7. Преобразование неструктурированных программ в структурированные. Метод введения переменной состояния. Достоинства и недостатки метода. Пример.

- 1)
 - Каждому блоку исходной схемы приписывается номер
 - Нумерация начинается с 1, последний блок - 0
- 2)
 - Вводится дополнительная целочисленная переменная - переменная состояния
- 3)
 - Функциональные блоки исходной схемы заменяются блоками, которые помимо основной задачи выполняют изменение переменной состояния
- 4)
 - Логические блоки преобразуются аналогично: в каждой ветви своё новое значение переменной состояния.
- 5) Исходная схема перестраивается:

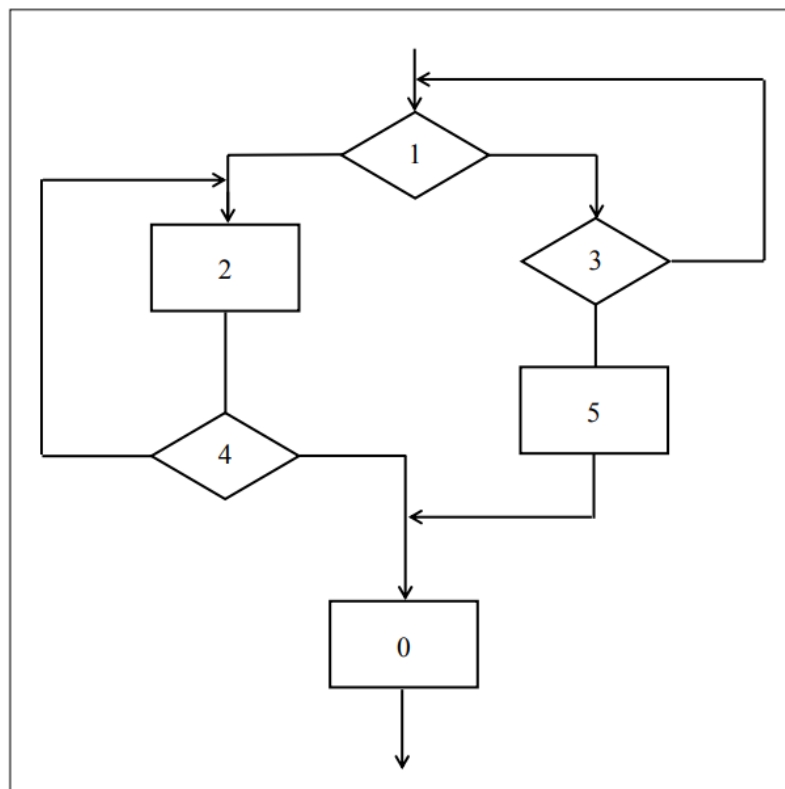
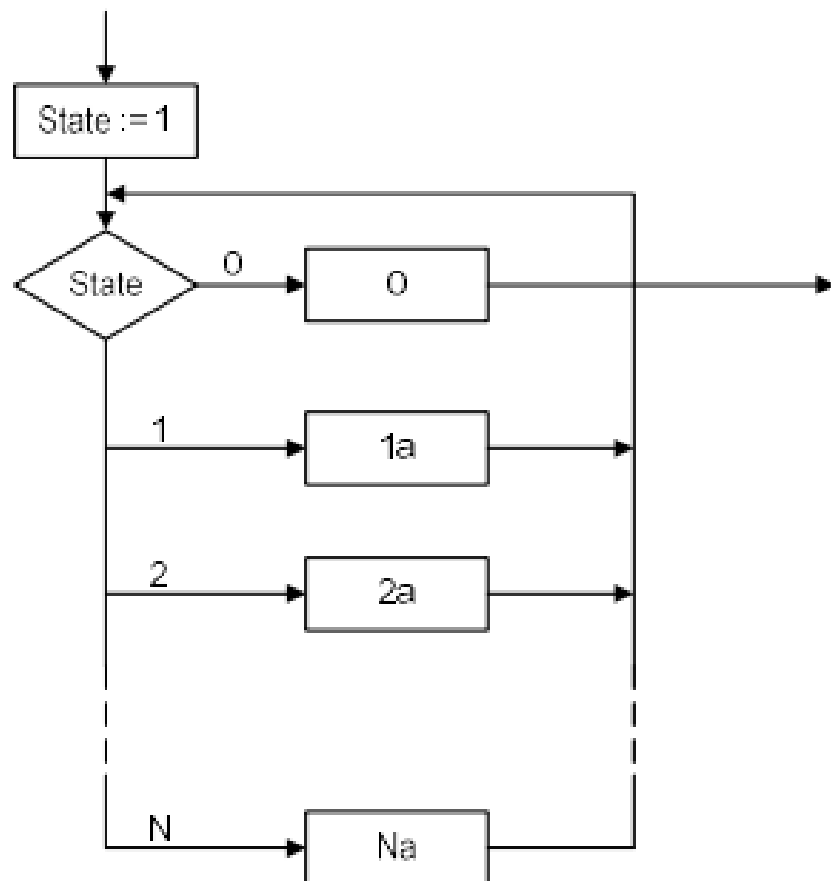


Рисунок 3.12 – Исходная схема неструктурированного алгоритма

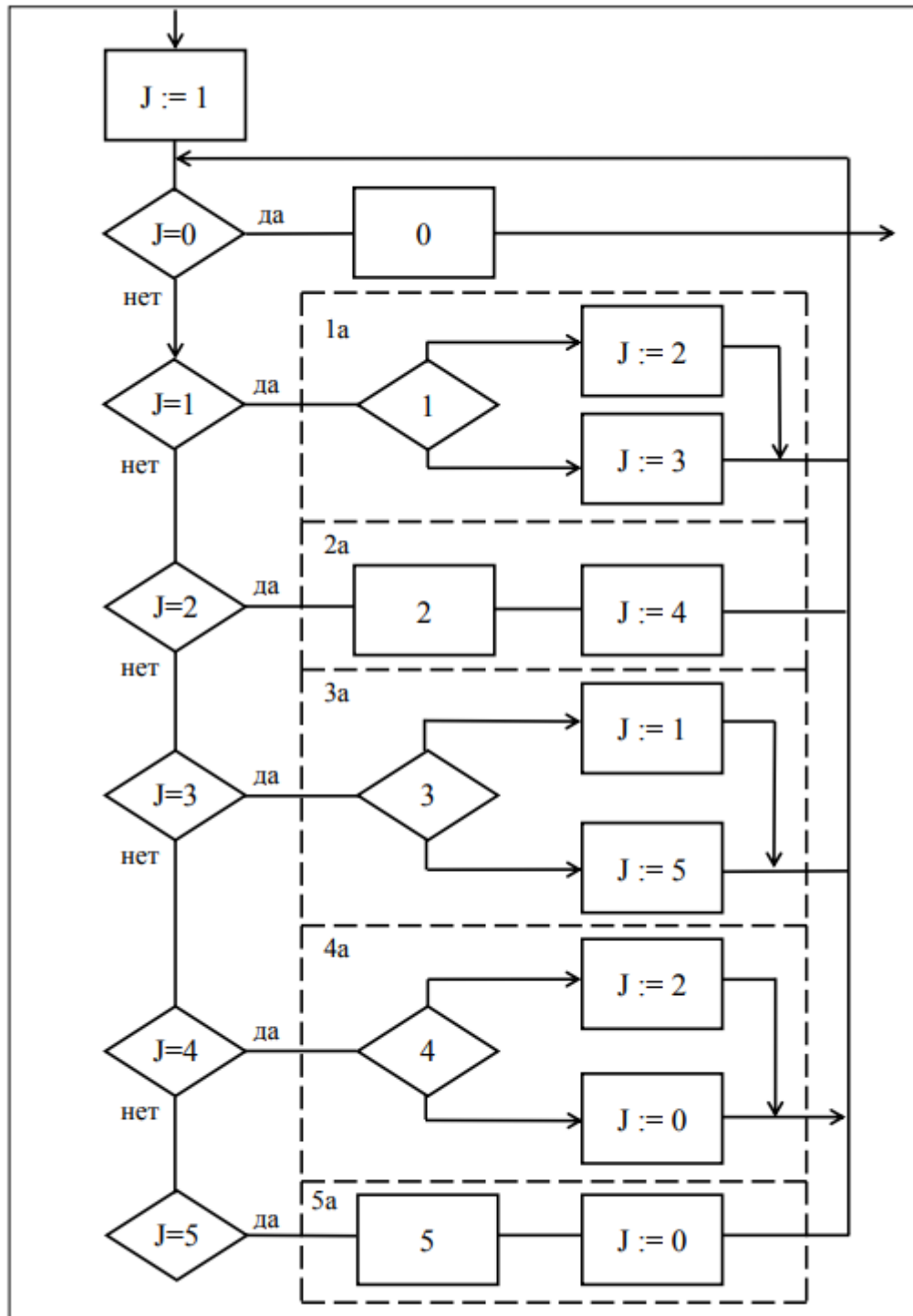


Рисунок 3.14 – Структурированная форма исходной схемы

Достоинства:

- Применим к алгоритмам любой структуры (в том числе и циклическим)
- Возможно автоматическое применение данного метода

Недостатки

- Топология схемы алгоритма сильно изменяется
- Затраты времени на проверку и изменение значения переменной состояния
- Громоздкость результирующей схемы алгоритма

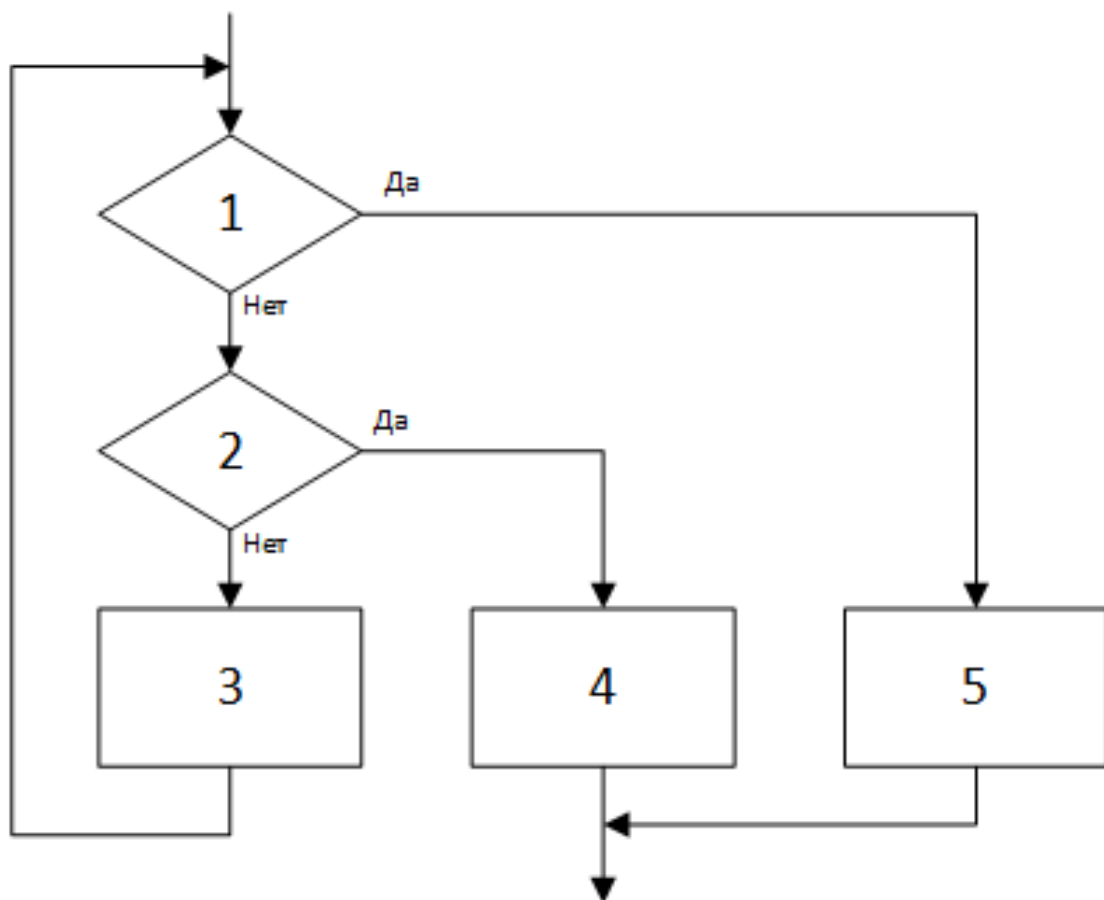
Вывод?

Метод введения переменной состояния можно рассматривать как разновидность автоматического программирования.

8. Преобразование неструктурированных программ в структурированные. Метод булевого признака. Достоинства и недостатки метода. Пример

Суть:

- в программе, содержащей циклы, вводится некоторый признак;(начальное значение признака задается до цикла)
- цикл выполняется, пока признак сохраняет исходное значение
- значение признака ИЗМЕНЯЕТСЯ при выполнении некоторых условий внутри цикла



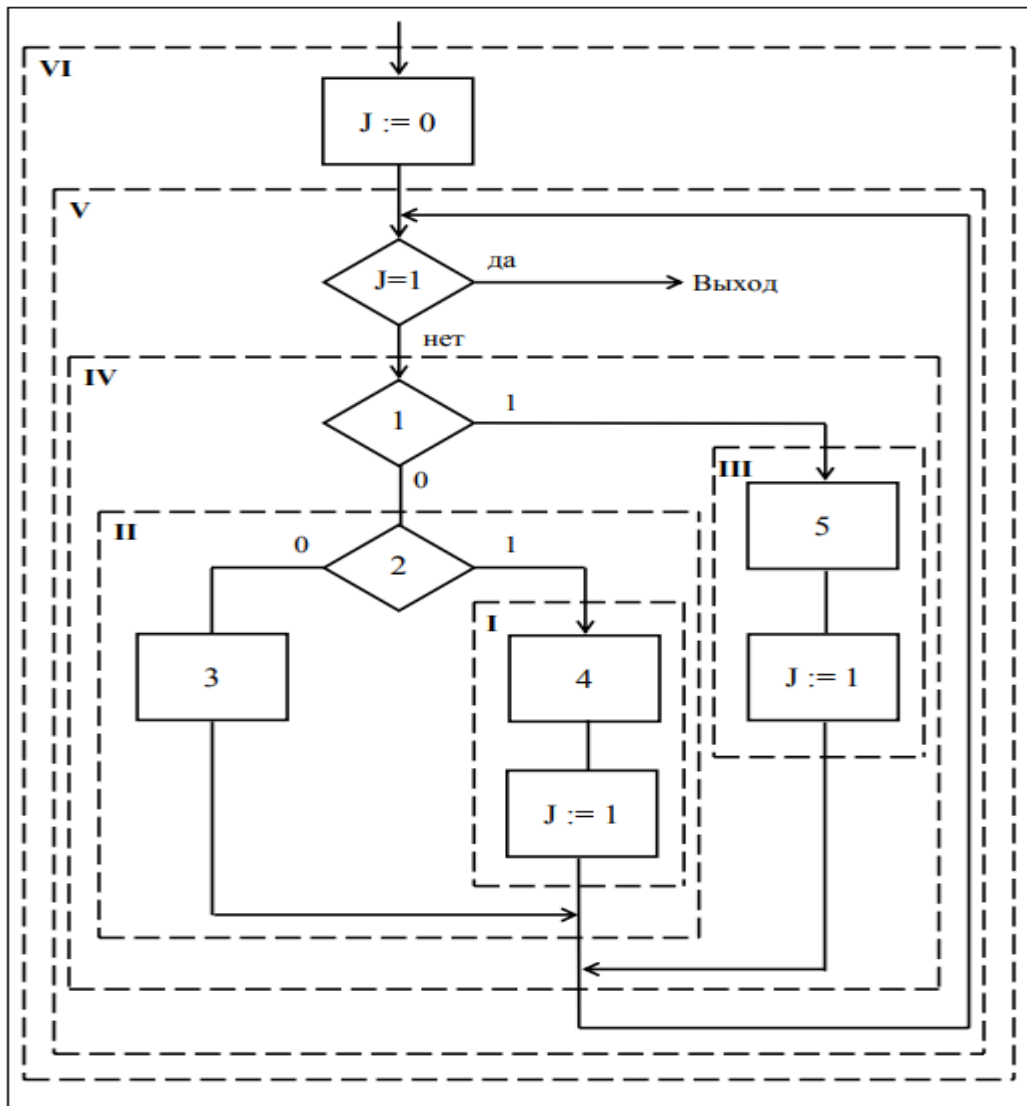


Рисунок 3.17 – Структурированная форма исходной схемы, преобразованная по методу булевого признака

Достоинства:

- Компактность, экономичность
- Топология схемы изменяется незначительно

Недостатки:

- Применяется только для алгоритмов с циклами.

!Иногда можно использовать без введения дополнительной переменной

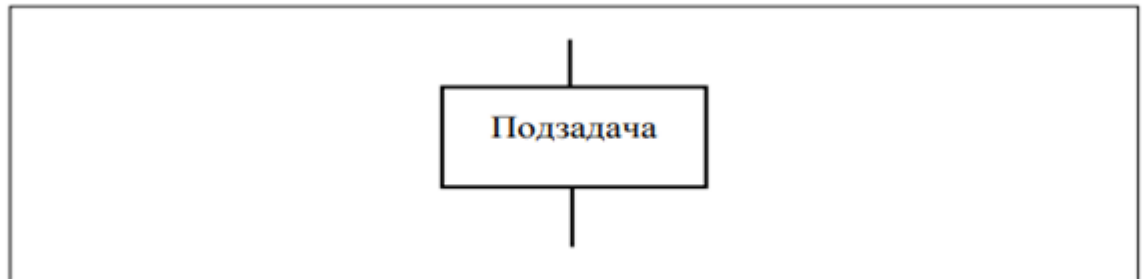
9. Метод Дамке

Три базовых конструкции :

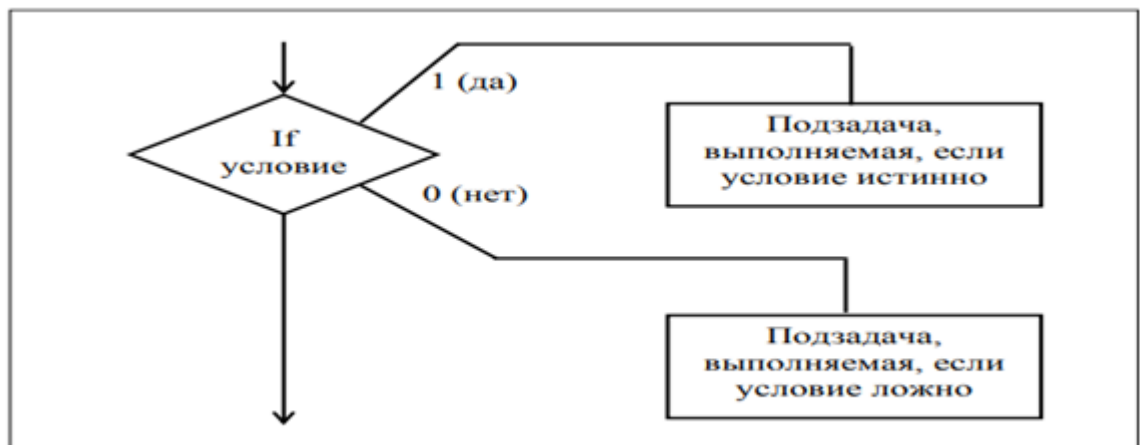
1. Функциональный блок

2. Конструкция if-then-else
3. Конструкция цикла с предусловием
- Допускаются дополнительные конструкции(цикл с постусловием, с параметром, case)

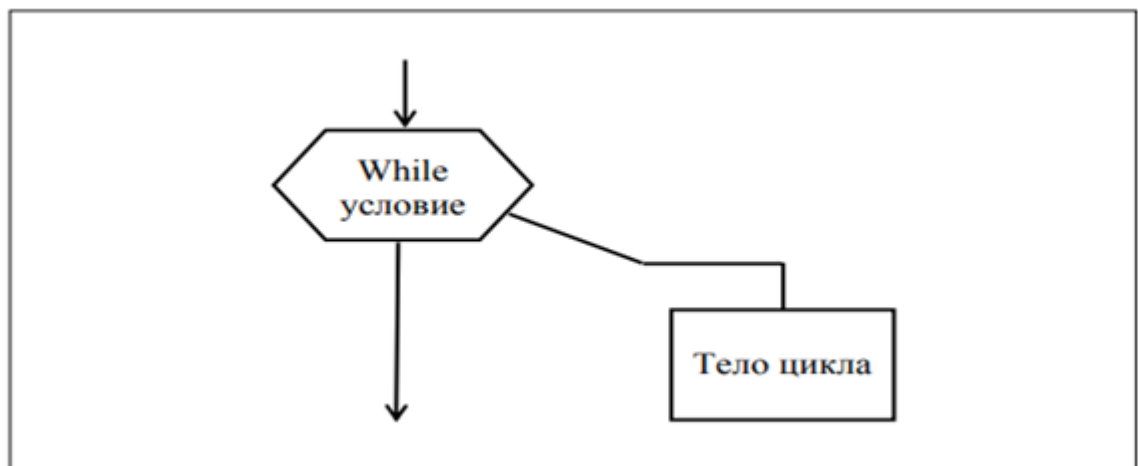
Функциональный блок



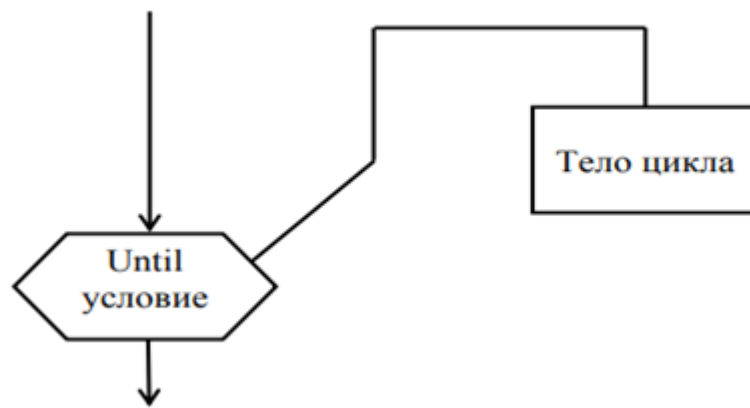
Конструкция if-then-else



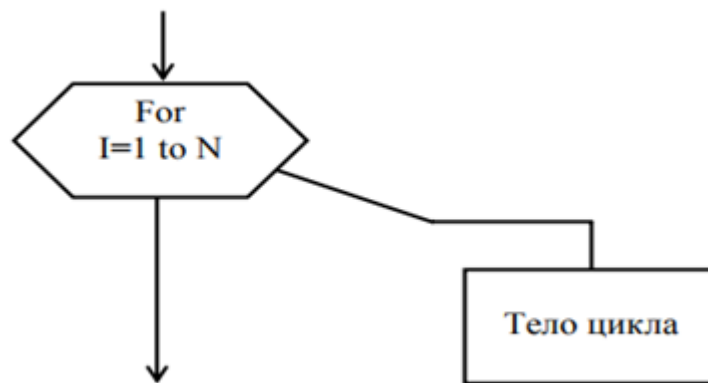
Конструкция цикла с предусловием



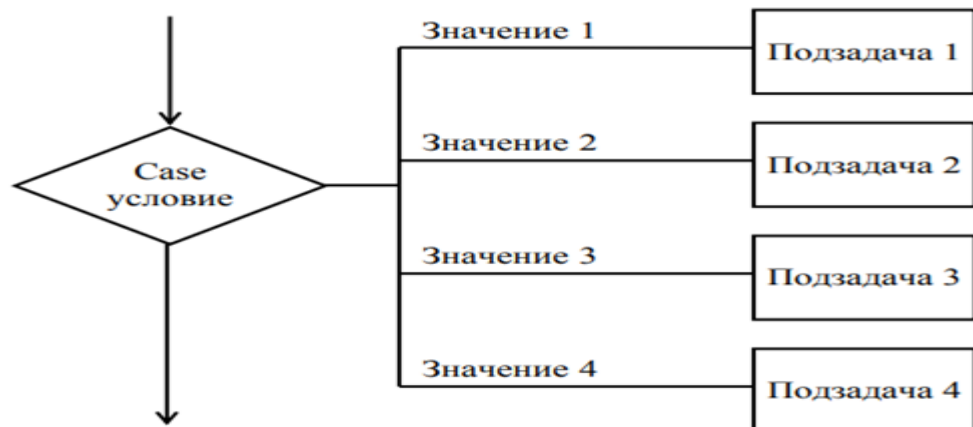
конструкции Repeat-Until



конструкция цикла с параметром



конструкция Case

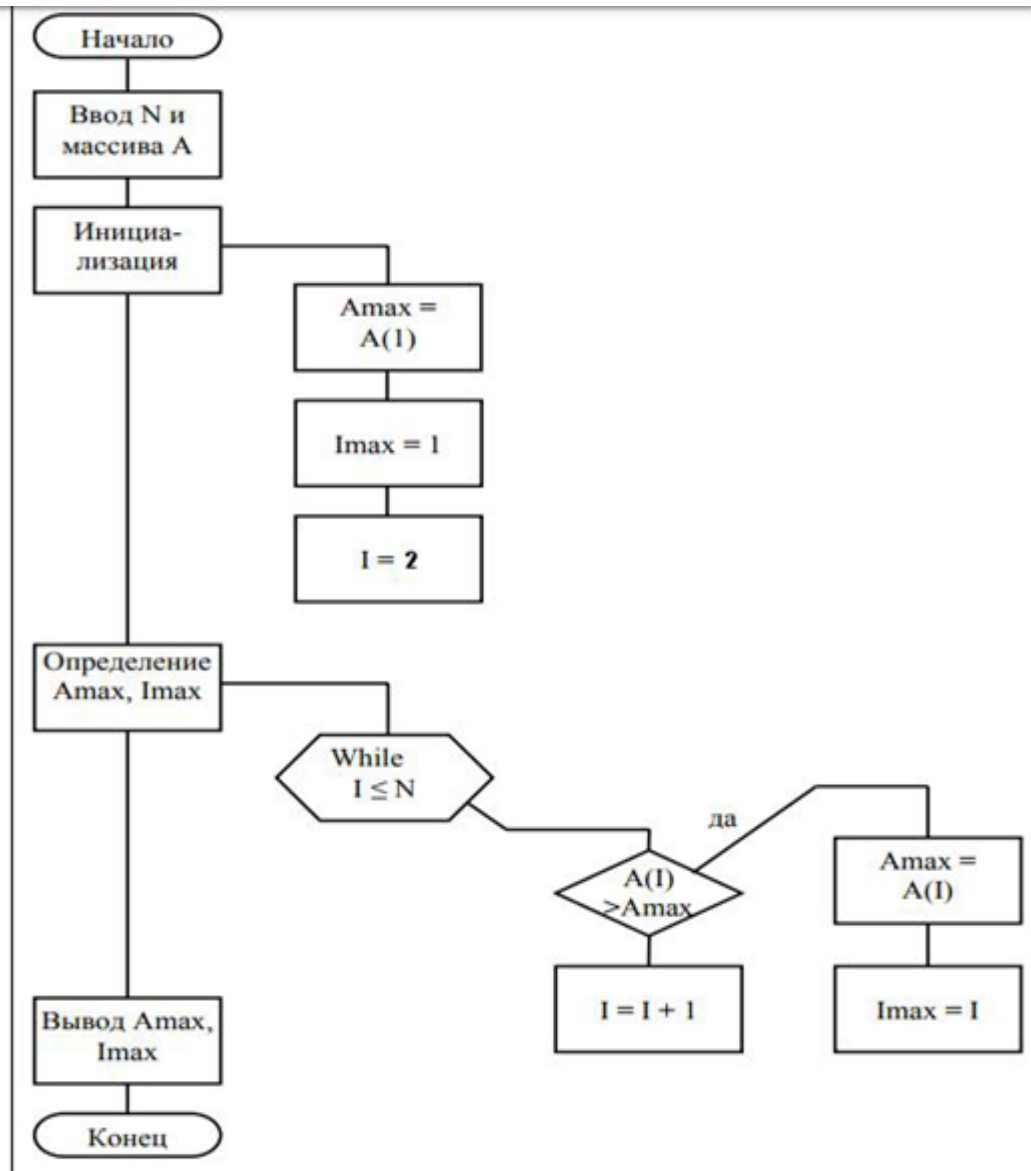


- o Основной принцип построения схемы алгоритма по методу дамке – принцип **декомпозиции**
- Элементы расположенные левее представляют укрупненную структуру алгоритма
- ±:
- o Не позволяет разработать схему неструктурированного алгоритма
- o Удобно использовать для нисходящего проектирования

- о Наглядность для большинства программ
- о Удобство коллективной работы
- ± Схемы могут быть более громоздкими

Пример

Дан массив A, состоящий из N элементов. Найти наибольший из элементов массива (Amax) и его номер (Imax).



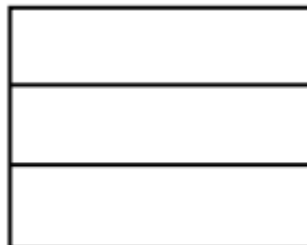
10. Схемы Насси-Шнейдермана. Пример.

Другое название - структурограмма

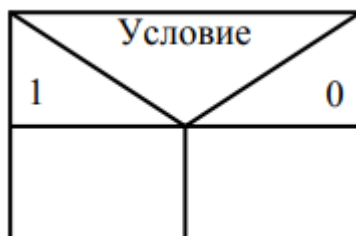
- иллюстрируют структуру передач управления с помощью вложенных блоков
- Отсутствуют линии, передачу управления

Функциональный блок

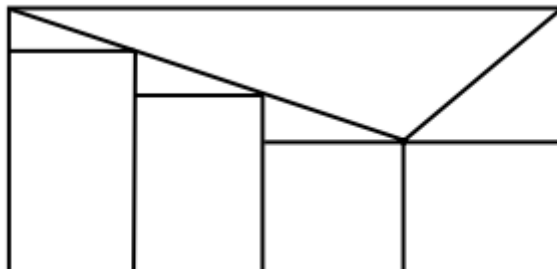
Блок следования



Блок решения



Блок Case



Цикл “Пока”



Цикл “До”



Дан массив A , состоящий из N элементов. Найти наибольший из элементов массива (A_{\max}) и его номер (I_{\max}).

6

Укрупненная схема
Насси-Шнейдермана

Ввод N и массива A
Инициализация
Определение A_{\max} , I_{\max}
Вывод A_{\max} , I_{\max}

Подробная схема
Насси-Шнейдермана

Ввод N и массива A			
$A_{\max} = A(1)$			
$I_{\max} = 1$			
$I = 2$			
$I \leq N$			
<table> <tr> <td>$A(I) > A_{\max}$</td></tr> <tr> <td>да</td></tr> <tr> <td>нет</td></tr> </table>	$A(I) > A_{\max}$	да	нет
$A(I) > A_{\max}$			
да			
нет			
$A_{\max} = A(I)$			
$I_{\max} = I$			
$I = I + 1$			
Вывод A_{\max} , I_{\max}			

11. Этапы постановки и решения задачи на компьютере. Методы автоматизации программирования. Назначение и классификация языков программирования.

- Чёткая формулировки задачи
 - исходные данные + формат данных
 - результат
- Математическая(формальная) постановка задачи
 - Выбор метода решений
- Разработка алгоритма решения задачи
- Выбор структур данных
- Программирование
- Тестирование и отладка
- Выполнение программы(решение задачи)

К **основным методам автоматизации программирования** можно отнести следующие.

- 1) Использование языков высокого уровня, близких к естественному человеческому языку, позволяющих автоматически однозначно преобразовывать написанную на них программу в программу на языке машины, т.е. в машинные коды.
- 2) Создание и использование библиотек стандартных программ и подпрограмм, предназначенных для реализации часто используемых задач.
- 3) Использование современных технологий программирования.
- 4) Использование Case-средств, предназначенных для автоматизации процесса разработки программ

Языки программирования позволяют записывать программы как обычный текст (по определенным правилам), а затем преобразовать в машинный код

Классификация ЯП:

- Низкого уровня
- Высокого уровня
 - *императивные* (процедурные)
 - *функциональные*
 - *логические*
 - *объектно-ориентированные*

12. Структура программного обеспечения. Системы программирования. Назначение, состав.

Под программным обеспечением (Software – в буквальном переводе – гибкое или мягкое) понимается совокупность программ, выполняемых вычислительной системой.

ПО, можно разделить на 3 категории:

- системное ПО (программы общего пользования), выполняющие различные вспомогательные функции, например создание копий используемой информации, выдачу справочной информации о компьютере, проверку работоспособности устройств компьютера и т.д.;

К системному программному обеспечению относят следующие программы:

1. Операционная система. Программа (точнее набор программ), которая управляет работой компьютера, ведет диалог с пользователем, запускает другие программы.
2. Драйверы. С помощью драйверов возможно подключение к компьютеру новых или нестандартное использование имеющихся устройств.
3. Программы-оболочки. Оболочки и среды представляют пользователю более удобный и приятный диалог, заменяя ввод команд операционной системе с клавиатуры нажатием функциональных клавиш или использованием манипуляторов типа мышь.
4. Программы-упаковщики (архиваторы). Применяя специальные методы упаковки информации, сжимают информацию из одного или нескольких файлов в новый архивный файл меньшего размера.
5. Антивирусные программы. Защищают компьютер от заражения компьютерными вирусами: осуществляют профилактику, диагностику, лечение.
6. Программы диагностики компьютера. Проверяют конфигурацию компьютера, параметры устройств, их использование и работу.
7. Программы обслуживания или утилиты. Утилиты – программы, предназначенные для внесения квалифицированных изменений в работу операционной системы и облегчающие поддержание компьютера в работоспособном состоянии.

- прикладное ПО, обеспечивающее выполнение необходимых работ на ПК: редактирование текстовых документов, создание рисунков или картинок, обработка информационных массивов и т.д.;

Примеры: текстовые процессоры; электронные таблицы; системы управления базами данных (СУБД); настольные издательские системы; электронные

словари; программы машинного перевода; программы распознавания текста и т.д.

- инструментальное ПО (системы программирования), обеспечивающее разработку новых программ для компьютера на языке программирования.

Системы программирования = инструменты программиста:

1) Текстовый редактор (text editor)

Чтобы набирать текст программы

2) Транслятор (translator)

Преобразовать программы из исходного кода в машинный. Выделяют два особых вида трансляторов: компиляторы и интерпретаторы.

- Компилятор - сразу переводит весь код, создает исполнительный файл
- Интерпретатор - переводит построчно, напрямую взаимодействуя с ОС

3) Отладчик (debugger)

Чтобы искать ошибки в программе

4) Документация

Самый важный инструмент!

5) Редактор связей (linker)

чтобы собирать программу из частей

6) И другие ...

13. Общая характеристика языка Delphi. Достоинства языка Delphi.

Команда - последовательность байтов.

Процессор постоянно:

- считывает очередную команду;
- выполняет ее.

Программа может быть написана сразу в виде команд процессора (машинный код).

Поэтому для упрощения написания кода используются языки программирования:

- низкого уровня;
- высокого уровня.

Языки программирования позволяют записывать программы как обычный текст (по определенным правилам), а затем преобразовывать его в машинный код.

Классификация языков программирования:

- машинно-ориентированный (низкого уровня, т.к. отношение машинных команд к количеству операндов языка, необходимых для написания программы близка к 1);

- процедурно-ориентированный(в основе - процедуры; C, Pascal, Basic);
- объектно-ориентированный(в основе лежат понятия класс и объект; C#, C++);
- проблемно-ориентированный(ориентированы на узкий класс однотипных задач; RPG - report program generator - генератор отчетов);
- языки 4 поколения(автоматически генерируют текст целиком, реализуют технологии визуального программирования; среды программирования Borland Delphi, C++ Builder).

Вторая - пятая группы относятся к **машинно-независимым языкам**, программа на них может быть выполнена только, если на компьютере есть транслятор.

Язык Паскаль относится к процедурно-ориентированным языкам высокого уровня. Разработан американским ученым Николас Виртом в 1971 г. в качестве языка для обучения программированию. Базой при разработке явился язык Алгол.

Достоинства языка Паскаль:

- 1) относительная простота (т.к. разрабатывался с целью обучения программированию);
- 2) идеология языка Паскаль близка к современным методикам и технологиям программирования, в частности, к структурному программированию и нисходящему проектированию (методу пошаговой детализации) программ. Паскаль может использоваться для записи программы на различных уровнях ее детализации, не прибегая к помощи схем алгоритмов;
- 3) гибкие возможности в отношении используемых структур данных;
- 4) высокая эффективность программ;
- 5) наличие средств повышения надежности программ, включающих контроль правильности использования данных различных типов и программных элементов на этапах трансляции, редактирования и выполнения.

Delphi — объектно-ориентированный язык программирования со строгой статической типизацией переменных.

Преимущества Delphi по сравнению с аналогичными программными продуктами.

- быстрота разработки приложения (RAD(rapid application development));
- высокая производительность разработанного приложения;
- низкие требования разработанного приложения к ресурсам компьютера;
- наращиваемость за счет встраивания новых компонентов и инструментов в среду Delphi;
- возможность разработки
- удачная проработка иерархии объектов.

14. Алфавит языка Delphi. Классификация символов.

Пример.

Алфавит языка делится на три группы:

14.1) Буквы.

В данную группу входят прописные (заглавные) буквы латинского алфавита от **A** до **Z**, строчные буквы латинского алфавита от **a** до **z** и символ подчеркивания (**_**). В программах строчные латинские буквы эквивалентны прописным везде, за исключением литералов (строковых констант).

Например, неразличимы записи

cost COST Cost,

но литералы

'cost' 'COST' 'Cost'

различны.

14.2) Цифры.

В данную группу входят десять десятичных цифр от 0 до 9. При записи программы на бумаге принято перечеркивать цифру 0, чтобы отличить ее от буквы O, которая не перечеркивается.

14.3) Специальные символы.

Специальные символы делятся на две подгруппы.

Первую подгруппу составляют простые и составные специальные символы.

К простым относятся следующие простые символы.

а) Символы-ограничители, содержащие

знаки арифметических операций:

+	плюс (сложение, объединение множеств, сцепление строк);
—	минус (вычитание, разность множеств);
*	звездочка (умножение, пересечение множеств);
/	наклонная черта, слэш (знак деления, результат всегда имеет вещественный тип);

знаки операций сравнения:

<	меньше;
>	больше;
=	равно;

знаки-разделители:

.	точка (десятичная точка в вещественных константах, разделитель полей при обращении к записи, признак конца модуля (блока));
,	запятая (разделитель в перечислениях)

: двоеточие (отделяет объявляемый элемент от его определения и метку от оператора);
; точка с запятой (разделитель операторов программы);
' апостроф (ограничитель строковых констант);

знаки скобок:

() левая и правая скобки (для выделения подвыражений);
[] левая и правая квадратные скобки (для выделения индексов массивов);
{ } левая и правая фигурные скобки (ограничители комментариев);

б) Символы, не входящие в подгруппу символов-ограничителей:

@ коммерческое 'at' (операция взятия адреса элемента);
номер (номер элемента в коде ASCII);
\$ знак доллара (признак шестнадцатеричных констант);
^ знак карата (обозначение указателей и динамических переменных);
пробел (символ, не имеющий обозначения).

В языке имеются определенные комбинации символов, называемые **составными специальными символами**, которые имеют специальное значение. Данные символы также относятся к символам-ограничителям:

<= меньше или равно;
>= больше или равно;
<> не равно;
:= операция присваивания;
.. обозначение диапазона.

Кроме того, имеются составные символы, являющиеся аналогами специальных символов скобок:

(* *) аналоги { };
(. .) аналоги [].

В языке Паскаль используются также синтаксическое понятие «шестнадцатеричная цифра». Шестнадцатеричными цифрами являются десятичные цифры и буквы A, B, C, D, E, F (или a, b, c, d, e, f). Обозначения шестнадцатеричных констант начинаются со знака \$. Например, \$F – это число 15 в десятичной системе счисления. Шестнадцатеричные цифры относятся к **составным символам**.

Вторая подгруппа специальных символов состоит из служебных слов. **Служебные слова** – это определенные комбинации латинских букв, имеющие в языке строго фиксированный смысл. Эти слова нельзя применять для других целей. Поэтому их еще называют **зарезервированными словами**. В языке Паскаль имеется большое количество служебных слов. К ним относятся:

Absolute And Asm Array Begin Break Case Const Constructor Continue
Destructor Div Do Downto Else End Exit External File For Forward Function Goto
If Implementation In Inline Interface Interrupt Label Mod Nil Not Object Of Or
Packed Procedure Program Record Repeat Set Shl Shr String Then To Type Unit
Until Uses Var Virtual While With Xor.

15. Основные понятия языка Delphi. Лексемы и их типы. Идентификаторы. Комментарии. Понятие оператора. Типы операторов. Пример.

Язык Делфи - структурированный объектно-ориентированный язык высокого уровня со строгой статической типизацией переменных

Грамматическое описание любого языка программирования включает в себя:

- 1) алфавит – набор основных символов языка, используемых при записи текста программы;
- 2) синтаксис – правила построения фраз языка;
- 3) семантика – смысловое значение фраз языка.

Текст программы, написанной на языке Delphi, состоит **из лексем, комментариев и пробелов.**

В программировании лексемой (или лексической единицей) называют неделимую последовательность знаков алфавита, имеющую в программе определенный смысл. В Delphi имеются следующие типы лексем:

- 1) простые и составные специальные символы;
- 2) идентификаторы;
- 3) литералы (строковые и символьные константы);
- 4) числовые константы;
- 5) метки;

I. Идентификаторы

Идентификаторы - это имя, которое позволяет однозначно выбрать один объект из множества объектов

В делфи идентификатор - любая последовательность

- букв (в т.ч. знаков подчеркивания)
- цифр

Идентификатор начинается с буквы(в т.ч. знака подчеркивания)

Идентификаторы делятся на две группы:

- **Предопределенные(стандартные)** - идентификатор, имеющий стандартный смысл и входящий в описание языка.

Предопределенные идентификаторы **не являются служебными словами**. Поэтому программист при желании может изменить их смысл, используя соответствующее описание. Однако это нежелательно, чтобы избежать лишних ошибок.

Примеры:

Real – атрибут действительных чисел.

Integer – атрибут целых чисел.

Sin встроенная функция синус.

- **Определенные программистом** – это идентификатор, смысл которого определен непосредственно в программе.

Такие идентификаторы обычно задают имена некоторых элементов программы.

Например, X – имя переменной.

Примеры правильно составленных идентификаторов:

Name **X** **ALFA** **N_18**

II. Комментарии

Комментарии используются для внесения пояснения текста программы

Может включать в себя любая последовательность символов, кроме символов-ограничителей комментариев

В языке Delphi комментарий ограничен специальными символами:

{ Блэт, комэнтарий }

или

(* Блэт, комэнтарий *)

// А это - только в одну строку _(ツ)_/

Вложенность комментариев не допускается. Но комментарии разного вида могут быть вложены друг в друга: *(*...{...}...*)* или *{...(* ...*)...}*

Комментарии *игнорируются транслятором* и не оказывают влияния на решение задачи. При выводе текста программы на печать комментарии выводятся вместе с текстом.

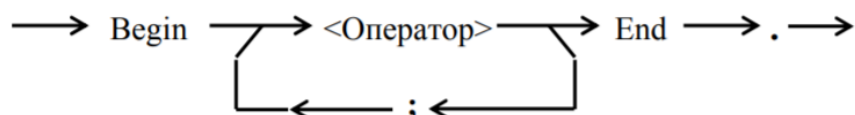
III. Оператор

Оператор - законченное предложение, написанное на языке программирования (ЯП)

- Состоит из лексем (“слов” ЯП)
- Представляет собой законченное описание какого-либо действия

Операторы записываются в разделе операторов

<Раздел_операторов> ::=



Операторы в языке Delphi могут быть разделены на две группы:

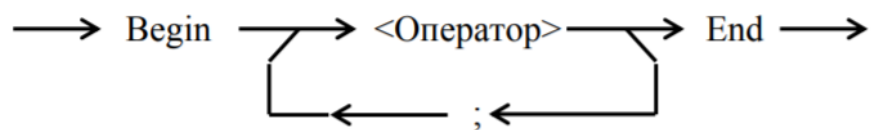
- **Основные операторы** – операторы, не содержащие в своем составе других операторов. Например, оператор присваивания $X := X + Y$
- **Производные операторы** – операторы, в состав которых входят другие операторы, например, операторы цикла(repeat/until, for, while), оператор условия (if, case) составной оператор(begin/end). Признаком конца оператора в последовательности операторов является точка с запятой

Составной оператор служит для объединения нескольких операторов, которые должны выполняться последовательно, путем их заключения в операторные скобки.

Символ ; (пустой оператор) используется для разделения операторов, входящих в состав составного. В составной оператор могут входить как основные, так и производные операторы, в том числе составные.

Компилятор воспринимает составной оператор как один оператор. Поэтому его можно использовать везде, где можно использовать основной оператор.

<Составной_оператор> ::=



16. Способы описания синтаксиса языков программирования. РБНФ. Пример.

Синтаксис - набор правил и соглашений, описывающие правильные предложения языка (правила)

Метаязык - формализованная система обозначений, применяемая для записи правил синтаксиса (способ записи правил)

Метаязыки:

- Расширенная форма Бэкуса-Наура (РБНФ)
- Синтаксическая диаграмма
- И ДРУГИЕ (Метаязык Хомского, Метаязык Хомского-Щутценберже, Бэкуса-Наура формы)

Основные понятия РБНФ:

- **Метаконстанты**

- Используется для обозначения лексем языка
- В программе метаконстанте соответствует она сама

Пример:

“end”, “+”, “begin”, “for”

- **Метапеременные**

- Используется для обозначения конструкций языка
- Метапеременные при записи заключаются в угловые скобки.

Примеры:

<Шестнадцатиричная_цифра>

<Оператор_while>

<Объявление_переменных>

- **Метасимволы**

- специальные символы, используемые в метаязыках для описания синтаксиса
- в программирования.

Метасимвол	Описание
::= (или =)	“определяется как” “по определению есть”
.	Конец определения
	“Либо”, “или”
{ }	Повторение (0,1,2, ...)
[]	Необязательная часть (0 или 1 раз)
()	Альтернатива

Примеры:

<Идентификатор> ::=
(<Буква>|<Подч.>) {<Буква> | <Цифра> | <Подч.> }.

<Буква> ::=
“a” | “b” | ... | “z” | “A” | “B” | ... | “Z”.

<Цифра> ::=
“0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”.

<Подч.> ::=
“ _ ”.

- **Синтаксическая единица**

- это строка, описывающая состав и порядок следования элементов конструкций языка программирования. Синтаксическая единица состоит из метапеременных, метаконстант и метасимволов.

17. Способы описания синтаксиса языков программирования. Синтаксические диаграммы. Пример.

Синтаксические диаграммы

Синтаксическая диаграмма - ориентированный граф с размеченными ребрами, используемый для описания синтаксической конструкции языка.

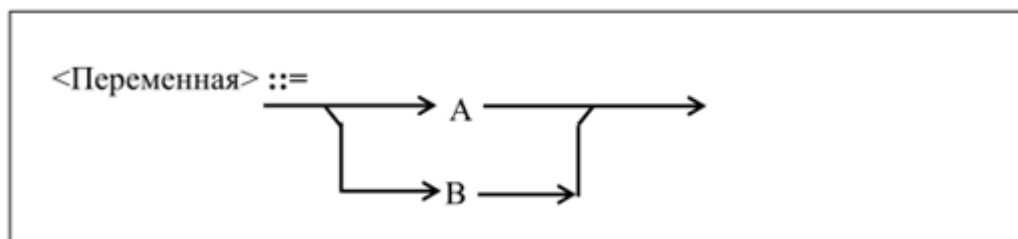
Ø Ребра помечены метапеременными и метаконстантами

Ø Метасимволы не используются

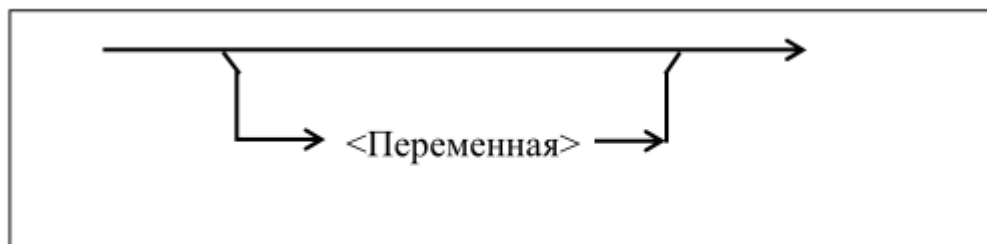
Ø Метаконстанты записываются без кавычек

Ниже рассмотрено представление в виде ориентированных графов некоторых из метасимволов языка РБНФ.

- а) **Выбору, альтернативе** (метасимволу | (Или)) соответствует разветвление в синтаксической диаграмме с последующим объединением. Например, переменная может принимать значение **А** или **В** (**А, В** – это лексемы языка Паскаль, т.е. *метаконстанты*).



- б) **Необязательная часть** (`[]`)



- с) **Повторение** (`{ }`)



Пример:

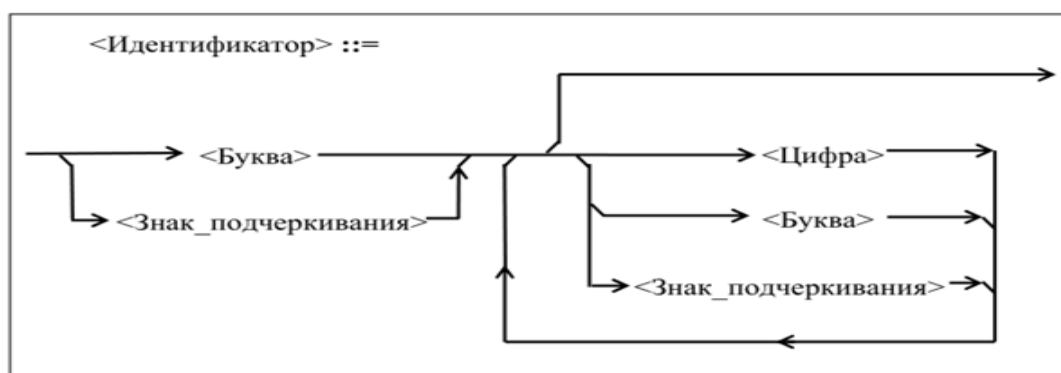


Рисунок 4.5 – Синтаксическая диаграмма определения «Идентификатор»

Если сравнивать между собой язык РБНФ и синтаксические диаграммы, то можно сделать следующие выводы. Язык РБНФ более строг и точен, более удобен для представления синтаксиса в памяти машины, более компактен. Синтаксические диаграммы более наглядны и просты для понимания, но более громоздки.

18. Классификация данных в языке Delphi. Пример.

Константа – это элемент данных, имеющий фиксированное значение.

Переменная – элемент данных, который может изменять своё значение в процессе выполнения программы.

Константа обозначается:

- Своим значением (42, 33);
- Именем (Pr, maxsize).

Переменная обозначается:

- Именем (x, BufSize).

Пример: $X := X + 5;$

X – переменная, 5 – константа

Тип данных (значений):

– множество различных значений, которые могут принимать элементы данного типа.

- свойства данных значений.
- операции, которые можно выполнять над этими значениями.

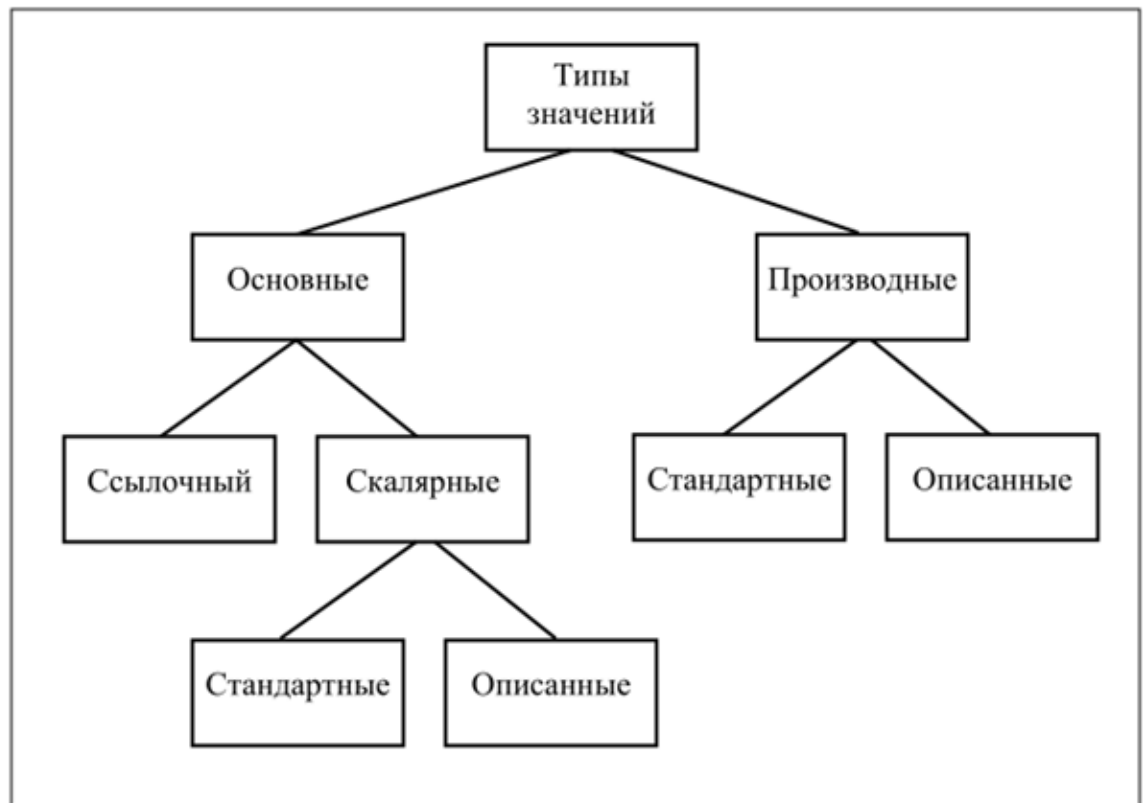


Рисунок 5.1 – Классификация типов языка Паскаль

В зависимости от типа для хранения значений этого типа выделяется определённый объём памяти. Значения записываются с помощью соответствующего количества нулей и единиц.

19. Целочисленные типы данных. Форматы. Диапазоны представления данных. Представление в памяти. Пример.

Основными целочисленными типами данных в Borland Pascal были следующие:

Название типа	Диапазон значений	Размер
Беззнаковые		
Byte	$0..2^8 - 1$	1 байт
Word	$0..2^{16} - 1$	2 байта
LongWord	$0..2^{32} - 1$	4 байта
Знаковые		
ShortInt	$-2^7..2^7 - 1$	1 байт
Integer	$-2^{15}..2^{15} - 1$	2 байта
LongInt	$-2^{31}..2^{31} - 1$	4 байта

Для 32-битных версий Delphi основными целочисленными типами данных стали:

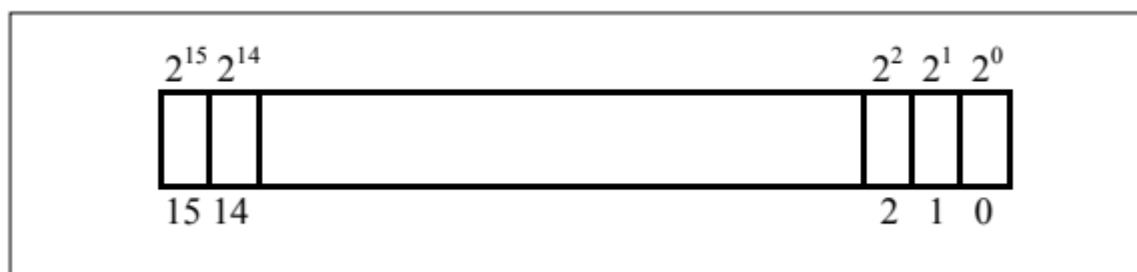
Название типа	Диапазон значений	Размер
Беззнаковые		
Byte	$0..2^8 - 1$	1 байт
Word	$0..2^{16} - 1$	2 байта
LongWord	$0..2^{32} - 1$	4 байта
Cardinal	$0..2^{32} - 1$	4 байта (*)
Знаковые		
ShortInt	$-2^7..2^7 - 1$	1 байт
SmallInt	$-2^{15}..2^{15} - 1$	2 байта
LongInt	$-2^{31}..2^{31} - 1$	4 байта
Int64	$-2^{63}..2^{63} - 1$	8 байт
Integer	$-2^{31}..2^{31} - 1$	4 байта (*)

Внутреннее представление целых чисел.

Рассмотрим внутреннее представление целых чисел в персональных компьютерах на базе, например, 16-разрядных микропроцессоров Intel.

Разрядная сетка таких компьютеров может быть представлена так, как показывает рисунок.

.На данном рисунке под каждым разрядом указан его номер (принято нумеровать разряды со стороны младших разрядов, начиная с нуля), а над каждым разрядом – его вес.



Минимальное двоичное число, которое может быть записано в данной разрядной сетке –

$$000...000_2 = 0_{10}$$

Максимальное число, которое может быть записано в шестнадцати разрядах –

$$11...11_2 = 2^{15} + 2^{14} + 2^{13} + \dots + 2^1 + 2^0 = 2^{16} - 1 = 65535_{10}$$

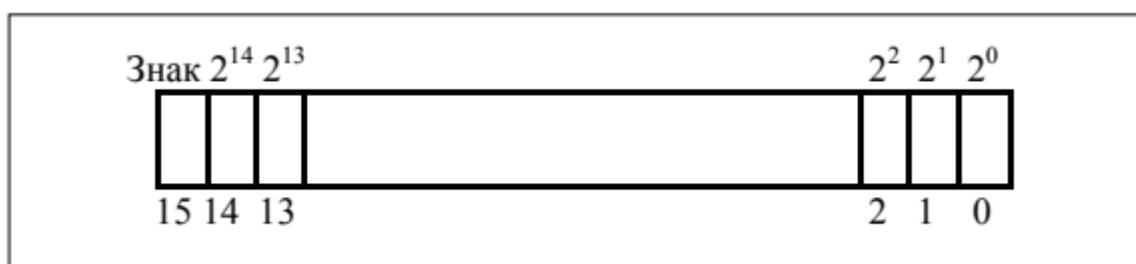
Этот диапазон (0 ÷ 65535) соответствует формату Word Паскаля.

Для типа Byte используется один байт без знака.

В данном случае диапазон представления чисел –

$$00...00_2 \div 11..11_2 = 0 \div 2^8 - 1 = 255_{10}$$

Для чисел со знаком старший бит соответствующего поля памяти, отведенного для хранения числа, считается знаковым. Если в нем 0 – число положительное (знак +), 1 – отрицательное (знак –).



Максимальное число, которое может быть записано в данной разрядной

$$11...11_2 = 2^{15} - 1 = 32767_{10}.$$

сетке -

Минимальное - 0 .

Отрицательные числа в памяти машины хранятся в дополнительном коде.

Дополнительный код (ДК) числа i образуется путем инвертирования

(замены значения на противоположное) всех значащих разрядов прямого кода

числа и прибавления 1 к самому младшему разряду. Аналогично производится

преобразование из дополнительного кода в прямой код.

Например, пусть $i = -1$.

000...01 прямой код числа 1

111...10 обратный код числа -1

+1

111...11 дополнительный код числа -1.

Дополнительный код числа i может также быть определен по формуле

$$i_{don} = 2^n - |i|$$

где n – разрядность представления операндов (включая знак), i – модуль i .

20. Целочисленные константы. Операции над целочисленными данными. Пример.

Множество целочисленных значений является **перенумерованным**, порядковым номером каждого значения целочисленного типа является само это значение.

Например, порядковый номер числа 5 равен 5, числа (-5) – (-5), порядковый номер нуля равен нулю.

Переменные целочисленных типов объявляются в разделе описания переменных программы с помощью указания идентификатора типа.

Например,

Var

a, b: integer;

c: shortint;

d: word;

Целые константы делятся на два типа – десятичные и шестнадцатеричные:

Признак шестнадцатеричной константы – знак \$ перед числом.

Примеры записи десятичных констант: +16, 25, -48

Примеры записи шестнадцатеричных констант: \$F, \$9A0

Целой константе присваивается целочисленный тип с наименьшим диапазоном, включающим значение этой константы.

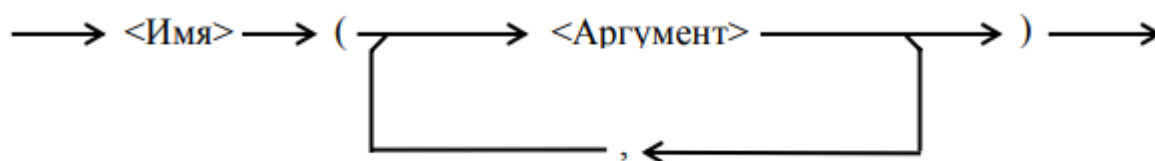
Таблица 5.4 – Операции над целочисленными данными

Группа операций	Операция	Вид	Описание	Тип результата
Арифметические операции	+	Одноместная	Сохранение знака	Целый
	–	Одноместная	Отрицание знака	Целый
	+	Двухместная	Сложение	Целый
	–	Двухместная	Вычитание	Целый
	*	Двухместная	Умножение	Целый
	/	Двухместная	Деление	Вещественный
	<i>div</i>	Двухместная	Целочисленное деление	Целый
	<i>mod</i>	Двухместная	Остаток целочисленного деления	Целый
Логические операции	<i>not</i>	Одноместная	Поразрядное дополнение целого	Целый
	<i>and</i>	Двухместная	Поразрядное логическое умножение (И)	
	<i>or</i>	Двухместная	Поразрядное логическое сложение (ИЛИ)	
	<i>xor</i>	Двухместная	Поразрядное логическое исключающее ИЛИ	
Операции сдвига	<i>shl</i>	Двухместная	<i>i shl j</i> – сдвиг влево значения <i>i</i> на <i>j</i> битов	Тип <i>i</i>
	<i>shr</i>	Двухместная	<i>i shr j</i> – сдвиг вправо значения <i>i</i> на <i>j</i> битов	Тип <i>i</i>
Операции сравнения	=	Двухместная	Равно	Логический
	<>	Двухместная	Не равно	
	<	Двухместная	Меньше	
	>	Двухместная	Больше	
	<=	Двухместная	Меньше или равно	
	>=	Двухместная	Больше или равно	

21. Встроенные процедуры и функции, определенные над целочисленными данными. Пример.

Встроенные процедуры и функции – это процедуры и функции, которые определены в компиляторе языка программирования. Имена встроенных процедур и функций являются предопределенными идентификаторами.

Синтаксическая диаграмма обращения к подпрограмме:



Процедуры и функции, определенные над целочисленными данными:

Название	Вид	Описание	Тип результата
chr(x)	функция	Возвращает символ кода ASCII с заданным номером (x: byte)	char
ord(x)	функция	Возвращает порядковый номер скалярного аргумента x (Ord(5) = 5; Ord(0) = 0; Ord(-5) = -5)	integer
abs(x)	функция	Абсолютное значение x (модуль)	тип x
sqr(x)	функция	Возвращает квадрат числа	тип x
dec(x, [n])*	процедура	Уменьшает значение x на величину n (по умолчанию на единицу; n:longint)	-
inc(x, [n])*	процедура	Увеличивает значение x на величину n (по умолчанию на единицу; n:longint)	-
odd(x)	функция	Возвращает True (истину), если аргумент нечетный, False (ложь) – если четный (x:longint)	boolean
pred(x)	функция	Возвращает предшествующий элемент в типе аргумента (для целочисленных данных возвращает x - 1)	тип x
succ(x) (вторая)	функция	Возвращает следующий элемент в типе аргумента (для целочисленных данных возвращает x + 1)	тип x
hi(x)	функция	Возвращает старший байт своего аргумента (x: integer или x: word)	byte
lo(x)	функция	Возвращает младший байт своего аргумента (x: integer или x: word)	byte
swa	функция	Возвращает значение,	тип x**

p(x)		образованное сменой младшего и старшего байта своего аргумента (x: integer или x: word)	
size of(x)	функция	Возвращает число байт, занимаемых своим аргументом (аргумент может быть именем переменной или именем типа)	integer***
random(n)	функция	Возвращает случайное число из диапазона 0..N – 1 (N:word)	integer

Примечания: * - в квадратные скобки заключена необязательная часть конструкции

** - параметр x (и результат) могут быть только типом smallint и word. Функция оставлена для обратной совместимости.

*** - тип возвращаемого значения определяется компилятором.

22. Вещественные типы данных. Форматы. Диапазоны представления данных. Представление в памяти. Вещественные константы и способы их записи в программе. Пример.

Значениями вещественных типов являются числа с плавающей точкой (ПТ), представленные в виде мантиссы и порядка.

Например, $0,125 = 0,125 * 10^0 = 1,25 * 10^{-1} = 125 * 10^{-3} = 0,00125 * 10^2$.
В общем случае числа с ПТ представляются неточно, операции над ними выполняются по правилам действий над приближенными числами.

Вещественные константы.

Вещественные константы в программе могут быть записаны в форме числа с фиксированной точкой (ФТ) или числа с плавающей точкой (ПТ).

Примеры записи вещественных чисел в форме с ФТ: 0.25; -2.48; +31.0

Примеры записи вещественных чисел в форме с ПТ: 14.3E5; 681E-2; -5.16E-3

Вещественные типы и их представление в компьютере.

Вещественные числа в памяти компьютера описываются стандартом IEEE754.

Существует пять вещественных типов для представления переменных:

- Real (вещественный);
- Single (с одинарной точностью);
- Double (с двойной точностью);
- Extended (с повышенной (расширенной) точностью);
- Comp (сложный тип).

Диапазоны и точности представления вещественных данных:

Тип	Диапазон	Точность (разрядность мантиссы)		Разрядность порядка (бит)	Формат (байт)
		бит	десятичных цифр		
Real	$10^{\pm 38}$	40 *	11 ÷ 12	8	6
Single	$10^{\pm 38}$	24 *	7 ÷ 8	8	4
Double	$10^{\pm 308}$	53 *	15 ÷ 16	11	8
Extended	$10^{\pm 4932}$	64	19 ÷ 20	15	10

Нормализованным числом называется число, мантисса которого попадает в диапазон

$(1/q) \leq m < 1$, где q – основание системы счисления.

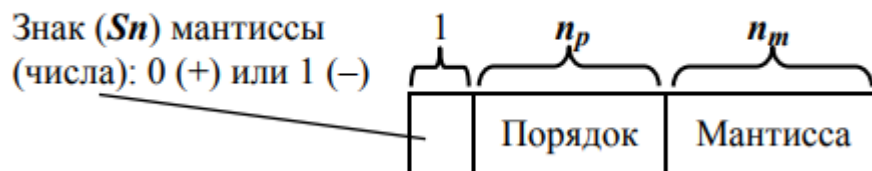
Для 2СС

$(1/2) \leq m < 1$,

то есть нормализованная мантисса представляет собой правильную дробь с единицей в старшем двоичном разряде.

Например, нормализованные мантиссы могут принимать значения 0.1101; 0.1000; 0.1111.

Представление вещественных чисел в памяти компьютера^



23. Операции над вещественными данными.

Встроенные функции, определенные над вещественными данными. Пример.

Операции над вещественными данными.

Для значений вещественных типов определены следующие операции.

1) Арифметические операции:

а) одноместные

+ (сохранение знака);

– (изменение знака);

б) двухместные

+ (сложение);
 – (вычитание);
 * (умножение);
 / (деление).

Все арифметические операции дают тип результата Real (при отсутствии сопроцессора), Extended (при наличии сопроцессора).

2) Операции сравнения:

= (равно);
 <> (не равно);
 >= (больше или равно);
 > (больше);
 <= (меньше или равно);
 < (меньше).

Встроенные функции, определенные над вещественными данными.

Функция	Описание	Тип результата
<i>Round(x)</i>	Округление, результат – ближайшее к <i>x</i> целое	Integer
<i>Trunc(x)</i>	Целая часть числа <i>x</i> независимо от знака	Integer
<i>Int(x)</i>	Возвращает целую часть <i>x</i>	Extended
<i>Frac(x)</i>	Возвращает дробную часть <i>x</i>	Extended
<i>Abs(x)</i>	Абсолютная величина <i>x</i>	Extended
<i>Arctan(x)</i>	<i>arctg(x)</i> . Результат в радианах.	Extended
<i>Cos(x)</i>	<i>cos(x)</i> . <i>x</i> – в радианах	Extended
<i>Sin(x)</i>	<i>sin(x)</i> . <i>x</i> – в радианах	Extended
<i>Exp(x)</i>	e^x (экспоненту; <i>e</i> в степени <i>x</i>)	Real

<i>Ln(x)</i>	<i>ln(x)</i>	Real
<i>Sqr(x)</i>	x^2	Extended
<i>Sqrt(x)</i>	\sqrt{x} (квадратный корень из x)	Extended
<i>Sizeof(x)</i>	Количество байтов для представления вещественного значения	Integer

24. Символьный тип данных. Способ упорядоченности. Представление в памяти. Операции и встроенные функции, определенные над символьными данными. Пример.

Символьный тип Char – это скалярный тип. Значениями этого типа являются элементы расширенного набора символов (литер) кода (американский стандартный код обмена информацией). Для представления значений типа Char отводится **один байт памяти**.

Элементы множества значений типа Char считаются перенумерованными (упорядоченными), т.е. каждому значению типа Char поставлен в соответствие свой порядковый номер. Порядковый номер символа равен его коду ASCII.]

Способ упорядочения определяется в соответствии с кодом ASCII (перечисление идет по возрастанию порядковых номеров):

- 1) Управляющие символы и специальные символы
- 2) десятичные цифры (они упорядочены по возрастанию);
- 3) заглавные латинские буквы (они упорядочены по алфавиту);
- 4) строчные латинские буквы (по алфавиту);

...

Переменные типа Char объявляются в разделе описания переменных.

Например,

Var

a, b, c, x: char;

Константой типа Char является один из допустимых символов, взятый в апострофы. Если значением константы является сам апостроф, то он записывается дважды. Примеры записи констант типа Char:

Над значениями типа Char определены только **операции сравнения**:

= (равно);

<> (не равно);

>= (больше или равно);

> (больше);

<= (меньше или равно);
 < (меньше).

'g' 'A' 'z' '8' 'Б' '''
 └──────────┘
 константа
 «апостроф»

Встроенные функции, определенные над символьными данными

Функция	Описание	Тип результата
<i>Ord(x)</i>	Преобразует x к целочисленному типу (возвращает порядковый номер символа x во множестве значений типа Char в коде ASCII)	Integer
<i>Pred(x)</i>	Возвращает символ, порядковый номер которого на единицу меньше порядкового номера x в коде ASCII	Char
<i>Succ(x)</i>	Возвращает символ, порядковый номер которого на единицу больше порядкового номера x в коде ASCII	Char
<i>Uppcase(x)</i>	Возвращает большую латинскую букву, если x – маленькая латинская буква, иначе возвращает x	Char
<i>Sizeof(x)</i>	Указывает количество байтов, требуемое для представления значения типа Char (значение	Integer

**25. Логический тип данных. Способ упорядоченности.
Представление в памяти. Операции и встроенные
функции, определенные над логическими данными.
Пример.**

Логический тип определяется как скалярный тип, множество значений которого состоит всего из двух значений:

False (ложь) и True (истина).

Значения логического типа упорядочены: значение False имеет порядковый номер 0, значение True имеет порядковый номер 1.

Значения типа Boolean занимают **один байт** памяти.

Логические переменные объявляются в разделе описания переменных.

Например,

Var

X, Y, Z: Boolean;

Логическими константами являются предопределенные в языке Паскаль идентификаторы

True и False.

Например, можно записать

X := True;

Y := False;

В этом случае переменным X, Y логического типа присваиваются значения констант True и False.

Таблица 5.11 – Операции, определенные над логическими данными

Группа операций	Операция	Описание	Тип результата
<i>Логические операции</i>	<i>Not</i>	Одноместная операция (НЕ), результат равен <i>True</i> , если значение операнда <i>False</i> , в противном случае – <i>False</i> .	Boolean
	<i>And</i>	Двухместная операция (И), результат равен <i>True</i> , если значение обоих операндов <i>True</i> , в противном случае – <i>False</i>	Boolean
	<i>Or</i>	Двухместная операция (ИЛИ) результат равен <i>True</i> , если хотя бы один из операндов равен <i>True</i> , в противном случае – <i>False</i>	Boolean
	<i>Xor</i>	Двухместная операция (исключающее ИЛИ) результат равен <i>True</i> , если только один операнд имеет значение <i>True</i> , в противном случае – <i>False</i>	Boolean
<i>Операции сравнения</i>	<div> <div>=</div> <div><></div> <div>></div> <div>>=</div> <div><</div> <div><=</div> </div>	<div> <div>Равно</div> <div>Не равно</div> <div>Больше</div> <div>Больше или равно</div> <div>Меньше</div> <div>Меньше или равно</div> </div>	Boolean

Таблица 5.13 – Встроенные функции, определенные над логическими данными

Функция	Описание	Тип результата
<i>Ord(x)</i>	Возвращает ноль, если $x = False$, единицу, если $x = True$	LongInt
<i>Pred(X)</i>	Для $x = True$ возвращает <i>False</i> , иначе не определено	Boolean
<i>Succ(X)</i>	Для $x = False$ возвращает <i>True</i> , иначе не определено	Boolean
<i>Sizeof(X)</i>	Указывает количество байтов для представления значения типа <i>Boolean</i>	Word

26. Выражения и их типы. Правила написания и вычисления выражений. Приоритет операций. Пример.

Выражение – формула для вычисления некоторого значения, состоящая из операндов, знаков операций и круглых скобок.

Операндами могут быть константы, переменные, указатели функций (вызовы функций, функции) и другие выражения.

Приоритет операций.

В порядке убывания приоритета операции в Паскале делятся на четыре группы.

1. операции Not, @ (взятие адреса объекта)
2. операции группы умножения:
*, /, Div, Mod, And, Shl, Shr
3. операции группы сложения
+, -, Or, Xor
4. операции группы сравнения
in, =, <>, >, >=, <, <=

Общие правила написания и вычисления выражений.

Ниже даны общие правила написания и вычисления выражений.

1. Первая группа операций имеет самый высокий приоритет, т.е. операции данной группы выполняются в первую очередь.
2. В каждой из групп операции имеют одинаковый приоритет
3. Операции одного приоритета обычно вычисляются в порядке их следования в выражении (слева направо). Иногда для оптимизации вычислений транслятор может переупорядочить их.
4. Операнд, находящийся между двумя операциями с различными приоритетами, связывается с операцией, имеющей более высокий приоритет.
5. Операнд, находящийся между двумя операциями с равными приоритетами, связывается с той операцией, которая находится слева от него.
6. Для уточнения последовательности действий применяются круглые скобки. Действия в скобках выполняются в первую очередь.

7. Последовательная запись знаков двух операций запрещена.
Исключение – операция **Not**.
Например, нельзя писать:
 $X^* - Y$.
Нужно написать:
 $X^*(-Y)$.
Однако можно написать:
 $X^*\text{Not } Y$.
8. Многоэтажные формулы в выражении записываются в одну строку.
9. Знак умножения * опускать нельзя.

ТИПЫ ВЫРАЖЕНИЙ

Скалярные стандартные выражения делятся на три типа:

- арифметические;
- логические;
- символьные.

Тип выражения определяется типом результата его вычисления.

27. Оператор присваивания и его типы. Составной оператор. Пустой оператор. Назначение. Формат. Пример.

Оператор присваивания

В общем случае **оператор присваивания** имеет формат, который представляет рисунок 5.11. Здесь <Ид._функции> -идентификатор (имя функции).

Оператор присваивания предписывает вычислить значение выражения, записанного в его правой части, и присвоить его переменной, имя которой записано в левой части. К моменту вычисления выражения все входящие в него переменные уже должны быть определены (иметь некоторые значения).

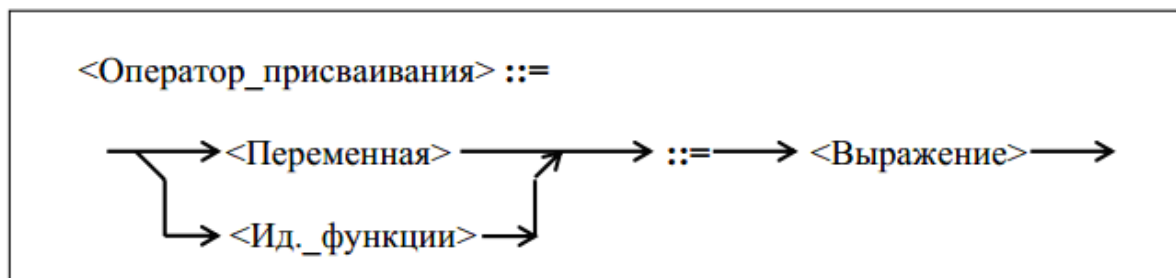


Рисунок 5.11 –Синтаксическая диаграмма оператора присваивания

Тип переменной в левой части оператора присваивания и тип выражения должны быть совместимыми по присваиванию. Поэтому, с учетом классификации скалярных стандартных выражений, существует **три типа скалярных стандартных операторов присваивания: арифметический, логический, символьный.**

Арифметический оператор присваивания.

Служит для присваивания значения переменной арифметического типа (вещественного или целочисленного). В правой части оператора должно быть

записано арифметическое выражение.

Примеры арифметических операторов присваивания:

$X := 0;$

$Y := 2 * a / b;$

$Z := \sin(c * 2 + a * a);$

Все переменные должны иметь арифметический тип.

Логический оператор присваивания.

Это оператор присваивания, в левой части которого указана переменная

типа Boolean. В правой части оператора должно быть логическое выражение.

Примеры логических операторов присваивания:

$A := \text{False};$

$B := G > L;$

$C := (E <> F) \text{ Or } \text{Odd}(X);$

$D := Y = Z;$

Здесь: **A, B, C, D** – переменные логического типа, **X** – переменная целого типа, **E, F, G, L, Y, Z** – переменные любых скалярных типов (совместимых между собой).

Символьный оператор присваивания.

Это оператор присваивания, в левой части которого указана переменная

типа Char. В правой части оператора должно быть задано символьное выражение.

Примеры литерных операторов присваивания:

$A := 'A';$

$B := C;$

$D := \text{Pred}(B);$

Здесь **A, B, C, D** – переменные типа Char.

Составной оператор

Структуру составного оператора представляет рисунок 7.1.

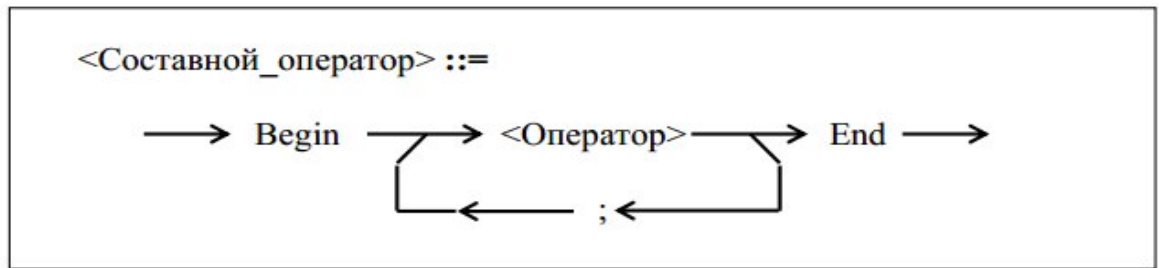


Рисунок 7.1 – Синтаксическая диаграмма составного оператора

Составной оператор служит для объединения нескольких операторов, которые должны выполняться последовательно.

Символ (;) используется для разделения операторов, входящих в состав составного.

В составной оператор могут входить как основные, так и производные операторы, в том числе составные.

Раздел операторов является, по существу, составным оператором.

Компилятор воспринимает составной оператор как один оператор.

Поэтому его можно использовать везде, где можно использовать основной оператор.

Примеры составных операторов:

1) *Begin X := 0 End*

Здесь составной оператор содержит один основной оператор.

2) *Begin X := A / B; Y := 2 * X End*

Здесь составной оператор содержит два основных оператора.

3) *Begin*

A := Sin(X);

Begin I := 0; J := 0 End

End

Здесь составной оператор содержит два оператора: основной и составной в составе составного.

Пустой оператор

Формат пустого оператора иллюстрирует рисунок 7.6.

Пустой оператор не задает никаких действий и

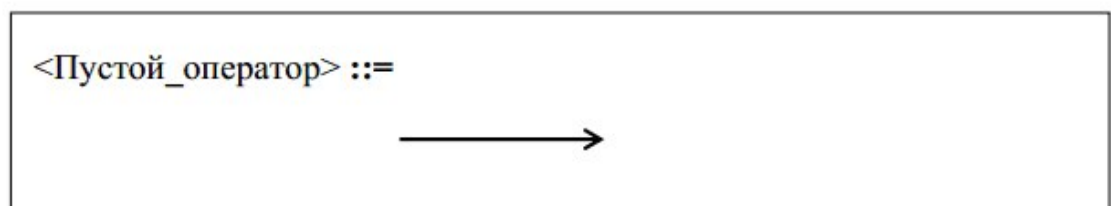


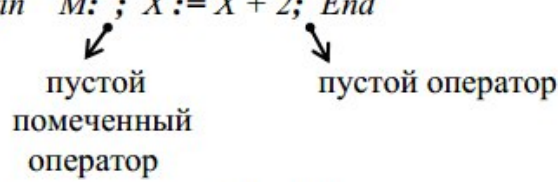
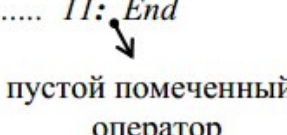
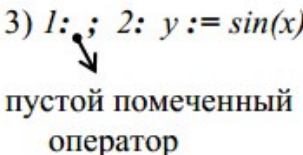
Рисунок 7.6 – Формат пустого оператора

не оказывает никакого влияния на ход выполнения программы. Он

может быть использован везде, где по синтаксису используется понятие <Оператор>.

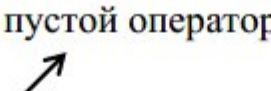
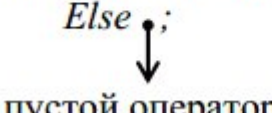
Пустой оператор часто используется для размещения метки в программе, не соотнося ее с некоторым оператором; для организации возможности обращения к одному оператору по нескольким меткам.

Примеры использования пустого оператора.

- 1) *Begin M: ; X := X + 2; End*

- 2) *Begin 11: End*

- 3) *1: ; 2: y := sin(x) – на данный оператор можно сослаться по любой из меток.*


Пустой оператор может быть использован в операторе *If* для устранения двусмысленности, связанной с использованием его вложенной сокращенной формы.

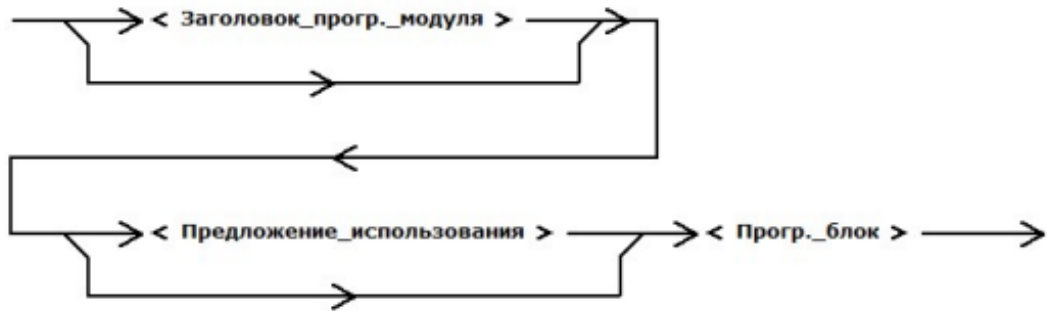
Примеры использования пустого оператора в операторе *If*.

- 1) *If A > B Then*

*Else A := 2 * A;*
- 2) *If X < 0 Then X := Abs(X)*
Else ;


Но, пожалуй, одно из основных назначений пустого оператора – упростить работу программиста с синтаксисом языка Паскаль, поскольку он позволяет более свободно оперировать с разделителем операторов (;).

28. Структура программы на языке Delphi.Синтаксис. Назначение стандартных модулей UNIT. Виды объявлений.

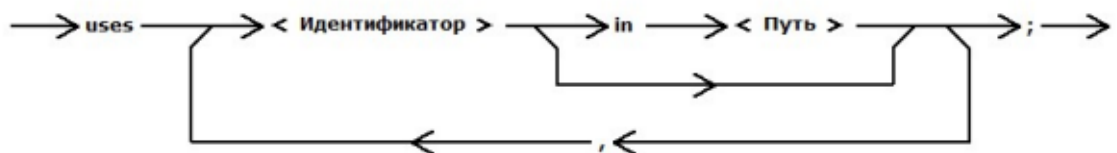
< Программный_модуль > ::=



< Заголовок_прогр._модуля > ::=



< Предложение_использования > ::=



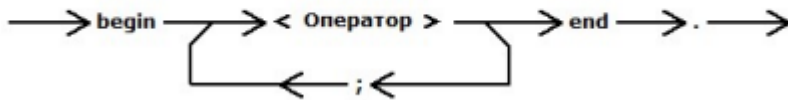
< Прогр._блок > ::=



Раздел описаний (виды объявлений)

- меток;
- констант;
- типов;
- переменных;
- процедур и функций (подпрограмм)

< Раздел_операторов > ::=



Модули Unit могут быть стандартными и модулями, созданными программистом. Существует восемь стандартных модулей Unit: System, Dos, Crt, Printer, Graph, Overlay, Turbo3, Graph3

29. Разделы меток, типов, переменных. Назначение, синтаксис. Пример.

Метка — это идентификатор или целочисленный литерал, используемый для обозначения определённого места в программе и последующего к нему обращения в нарушение последовательного порядка выполнения операторов. Переход к помеченному оператору осуществляется с помощью оператора goto.

< Раздел_меток > ::=



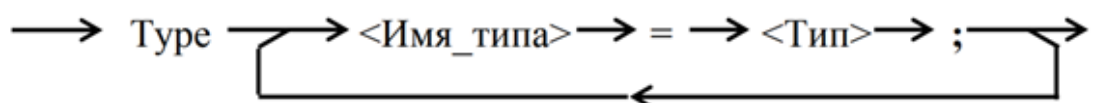
Пример:

```
Labelname1: X:=X+Y;
...
anotherlabel: Z:=sin(X);
...    Goto labelname1;
...
Goto anotherlabel;
...
```

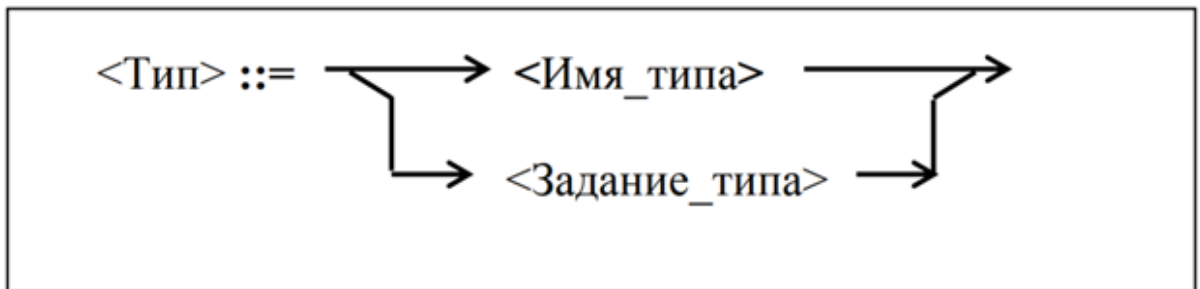
Раздел типов:

Позволяет задавать пользовательские типы.

<Раздел_типов> ::=



Описание типов:



type

```
Bool = Boolean;
```

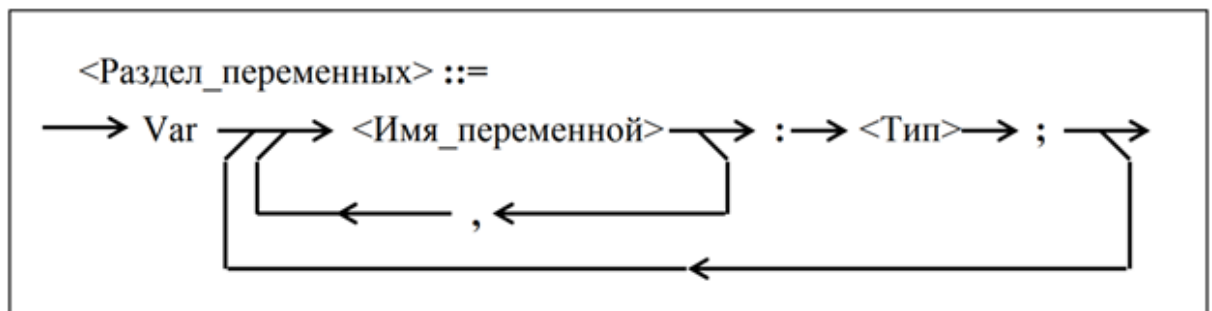
```
Int = Integer;
```

```
TWeek = (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
```

(При вычислениях переменные типа TWeek смогут принимать одно из перечисленных в скобках значений. Введенные в разделе Type типы могут использоваться в дальнейшем при описании переменных.)

Раздел переменных:

Раздел начинается словом Var:



Тип переменной можно задать двумя способами.

1-ый способ: Тип описан в разделе типов или является стандартным

Var

```
X: Bool;
```

```
I: Int;
```

```
Day: TWeek; //Явный задание типа
```

```
A, B, C: Real;
```

Здесь Bool, Int, TWeek – описанные в разделе Type типы, Real – стандартный тип.

2-ой способ. В описании переменной используется **неявное задание типа**.

При таком способе описывается переменная и одновременно задается новый тип без имени.

```
Var
```

```
Day: (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
```

Каждая переменная должна быть описана обязательно и ровно один раз.

30. Раздел констант. Типизованные и нетипизованные константы. Назначение, синтаксис. Типизованные скалярные константы. Пример.

Присвоение константе имени выполняется в разделе описания констант.

Раздел начинается служебным словом `const`:

```
Eps1 = 0.0001;  
Eps2 = Eps1 / 100.0;  
Pi1 = -2 * Pi;
```

Описание *нетипизированной константы* имеет следующий синтаксис:

< Описание_константы > ::=

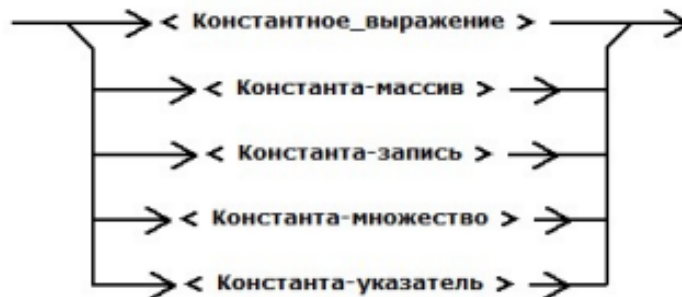
→ < Идентификатор > → = → < Константное_выражение > → ; →

Объявление типизованной константы:

< Описание_типиз._константы > ::=

→ < Идентификатор > → : → < Тип > → = → < Типиз._константа > →

< Типиз._константа > ::=



Типизированные константы во многих реализациях языка Pascal, а также ранних версиях Delphi вели себя, как обычные переменные, но имеющие начальное значение. В более поздних версиях Delphi это поведение настраивается: можно запретить изменение типизированных констант, сделав их полноценными константами.

Типизованные константы инициализируются только один раз – в начале выполнения программы или подпрограммы. При каждом новом входе в подпрограмму типизованные константы заново не инициализируются.

Const

```
MaxI: Integer = 10000;  
MinI: Integer = 0;  
Eps: Real = 0.0001;  
Pi2: Real = 4 * Pi;
```

31. Оператор GOTO. Формат. Назначение. Ограничения использования. Недостатки. Пример.

<Оператор_goto>::=

→ goto → < Метка > →

Используется для безусловной передачи управления оператору, помеченному меткой

Ограничения:

- 1) с помощью Goto нельзя переходить внутрь производных операторов, не содержащих данный оператор Goto (составного оператора, операторов For, Repeat, While, If, Case, With);
- 2) с помощью оператора Goto запрещен переход из одной альтернативы в другую в выбирающих операторах (If, Case);
- 3) с помощью оператора Goto нельзя входить в подпрограмму или выходить из нее

Label 1,2 ;

...

i:=1;

1: if i > 10 then

goto 2;

writeln (i : 7);

inc(i);

goto 1 ;

2: writeln ('Mission complete') ;

Недостатки:

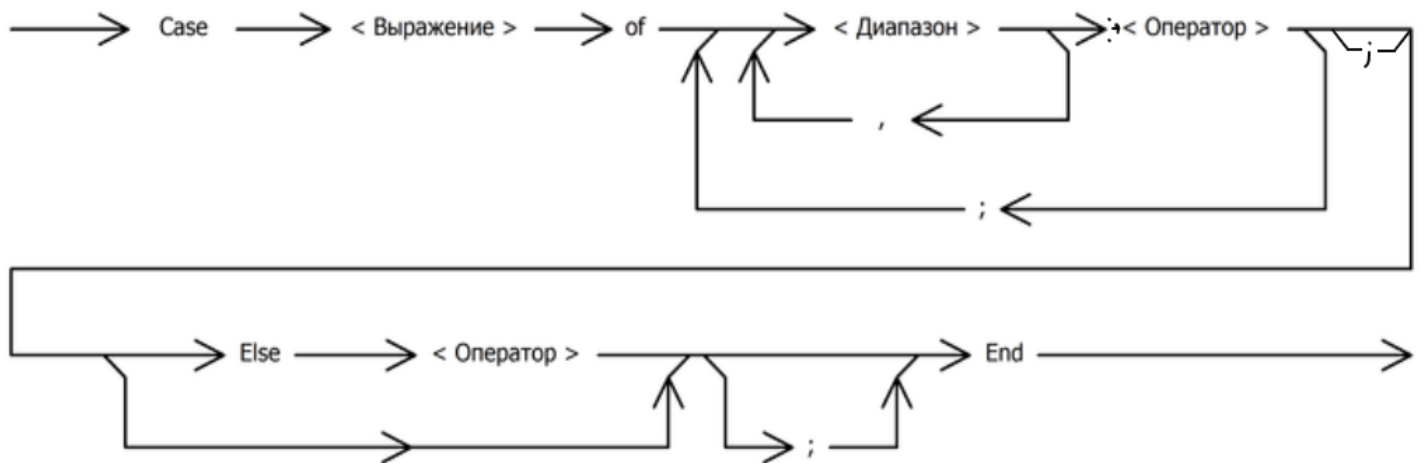
1. Наличие оператора Goto делает программу ненаглядной, трудно читаемой, трудно отлаживаемой.
2. Программа с Goto не является структурированной.

Примечание: см. [вопрос №4 "Структурное программирование"](#)

32. Оператор IF. Формат. Назначение. Полная и сокращенная форма. Пример.

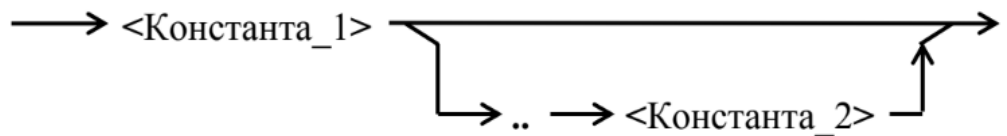
Оператор If является производным оператором. Относится к группе выбирающих операторов. Используется в разветвляющихся программах для выбора того или иного участка вычислений в зависимости от выполнения некоторого условия. Поэтому его еще называют оператором условного перехода.

< Оператор_Case > ::=



Здесь диапазон(синтаксическая диаграмма типа диапазон:)

<Диапазон> ::=



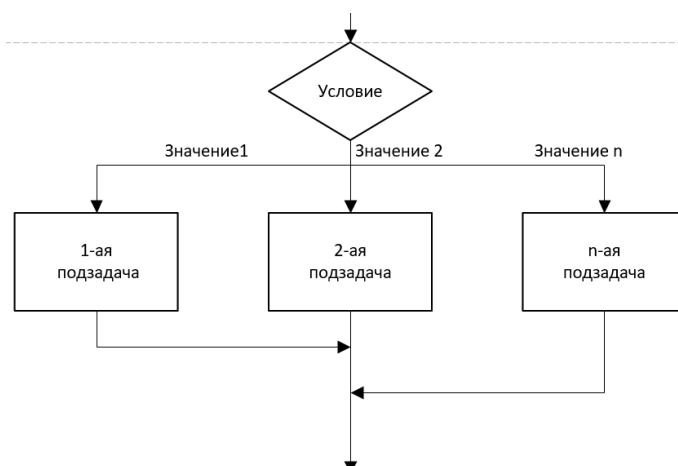
Оператор выбора Case это как If в общем случае

Выражение, находящееся после “Case” называется **селектором** (переключателем)

Селектор может иметь любой перенумерованный тип

Выполнение:

- 1) вычисление селектора
- 2) выбор из всех операторов того, у которого среди значений констант выбора имеется значение из пункта 1)
- 3) если 2) нет, то выполняется ветвь, помеченная Else
- 4) если и 3) нет, то выполняется оператор следующий, за Case



- это какая-то схемка из Сашиного конспекта 😊

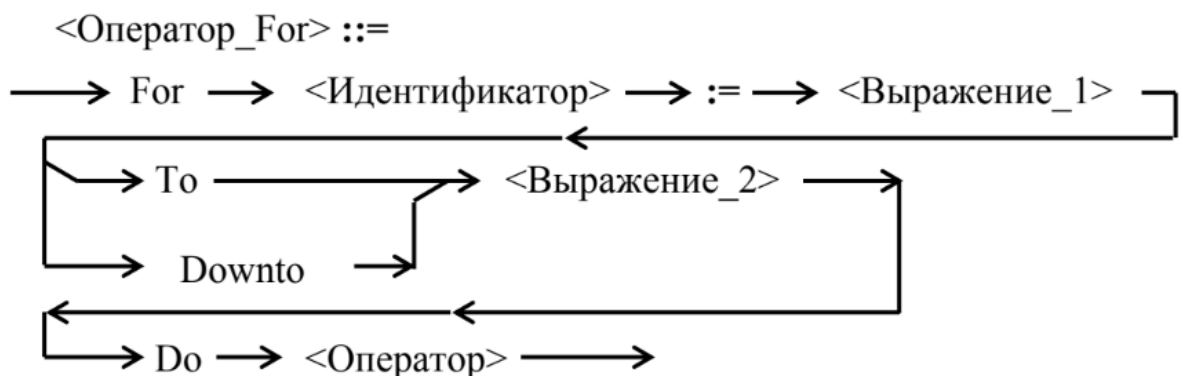
Пример:
Case Symbol of
'A'..'Z','a'..'z': writeln('Letter');
'0'..'9': writeln('Digit')
else writeln('Other character')
end

34. Оператор цикла с параметром. Формат. Назначение. Правила выполнения. Пример.

Используется в циклах с известным числом итераций

Параметр цикла - переменная, служащая для проверки условия выполнения цикла

ПОСЛЕ ЦИКЛА FOR ПАРАМЕТР ЦИКЛА ИМЕЕТ НЕОПРЕДЕЛЕННОЕ ЗНАЧЕНИЕ!!!!!!



<Идентификатор> – это имя параметра цикла. В качестве параметра может быть использована переменная любого скалярного типа, кроме вещественного (т. е переменная перенумерованного типа).

<Выражения_1 и 2> – это выражения того же типа, что и параметр цикла.

<Оператор>- это тело цикла. Он может быть только один (можно составной)

Выполнение

Перед каждым выполнением тела цикла происходит сравнение текущего значения параметра с его предельным значением, определяемым <Выражением_2>. Если текущее значение меньше

либо равно (в случае To) предельного значения или больше либо равно (в случае Downto) предельного значения, тело цикла выполняется.

НЕЛЬЗЯ ИЗМЕНЯТЬ ЗНАЧЕНИЕ ПАРАМЕТРА ЦИКЛА ВНУТРИ ЦИКЛА!!!!

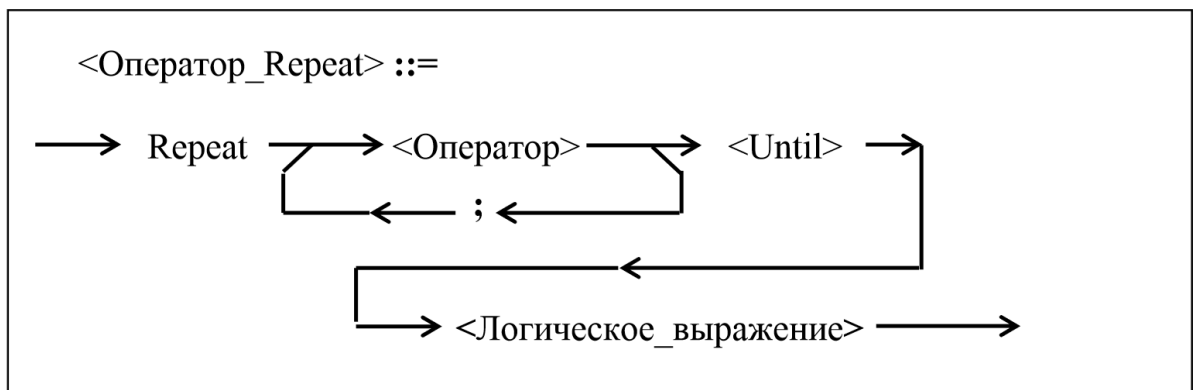
Пример	//Если внутри цикла больше чем одно
действие,	
s := 0;	//то нужно использовать begin-end
for i := 1 to n do	// Даже ежику понятно
s := s + i*i;	
WriteLn(s);	

35. Оператор цикла с постусловием. Формат.

Назначение. Правила выполнения. Пример.

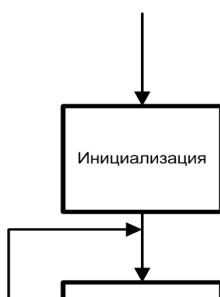
Цикл с постусловием — цикл, в котором условие проверяется **после** выполнения тела цикла. Отсюда следует, что тело **всегда выполняется** хотя бы один раз.

Формат оператора Repeat:



Пояснение к схеме: Тело цикла выполняется до тех пор, пока <Логическое_выражение> принимает значение False. Как только после очередного выполнения цикла <Логическое_выражение> станет равным True, выполнение цикла прекращается.

Схема цикла с постусловием:



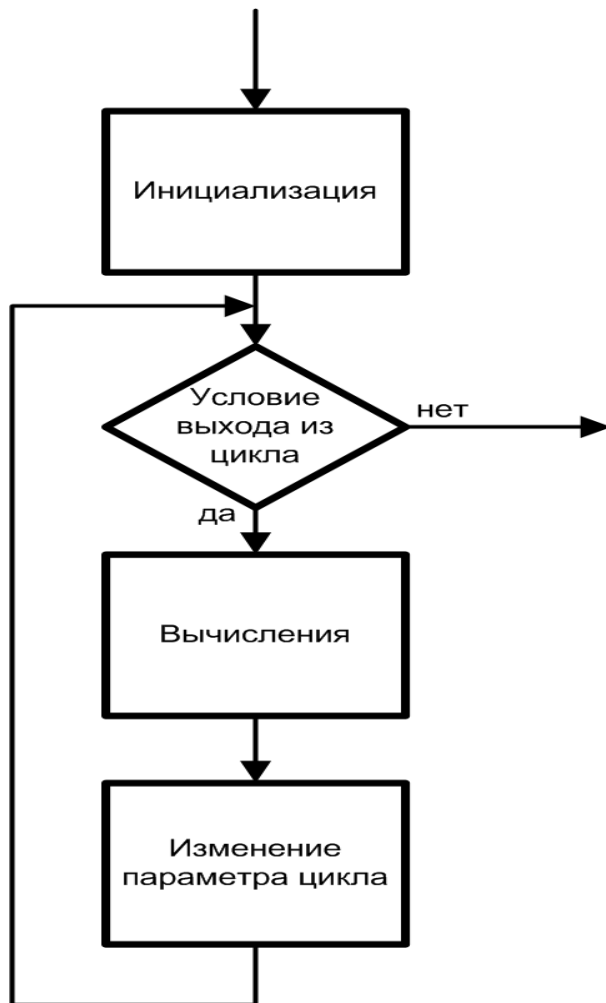
Пример:

```
S := 0;  
I := 1;                                {Установка начального значения I}  
Repeat  
    S := S + I * I;  
    I := I + 1                          {Самостоятельное изменение параметра I}  
Until I > N;                          {Выход из цикла по выполнению условия}  
Writeln (S);
```

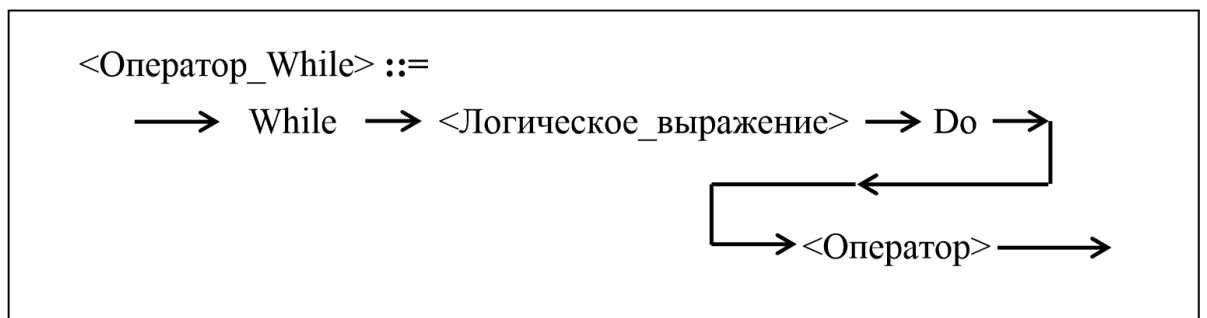
36. Оператор цикла с предусловием. Формат.

Назначение. Правила выполнения. Пример.

Используется для программирования цикла с заранее **неизвестным числом повторений** в тех случаях, когда тело цикла при некоторых условиях может **не выполняться ни разу**.



Формат оператора While:



Пояснение к схеме: <Оператор> определяет тело цикла. **Перед** каждым его выполнением вычисляется значение <Логического_выражения>. Если оно равно True, то тело цикла выполняется. Выполнение цикла прекращается, когда <Логическое_выражение> впервые станет равным False. Если к началу

выполнения цикла значение < Логического выражения> равно False, то тело цикла не выполняется **ни разу**. Если в теле цикла необходимо выполнить несколько операторов, используется составной оператор.

Схема цикла с **предусловием**:

Пример:

```
S := 0;
I := 1;                                {Установка начального значения I}
While I <= N Do                       {Вход в цикл по выполнению
условия}
Begin
    S := S + I * I;
    I := I + 1                          {Самостоятельное изменение параметра
I}
End;
Writeln (S);
```

37. Сравнительная характеристика операторов цикла. Пример.

Цикл с параметром for отлично выполняет свои функции, когда число повторений действий заранее известно. Однако такая ситуация встречается в программировании далеко не всегда. Часто приходится решать задачи, когда число повторений действий в теле цикла неизвестно и определяется в ходе решения задачи. В этом случае применяют цикл с условием. В языке программирования Pascal имеется две разновидности цикла с условием:

цикл с предварительным условием - условие цикла проверяется перед выполнением тела цикла;

цикл с последующим условием - условие цикла проверяется после выполнения тела цикла.

Остановимся на цикле с предварительным условием (кратко, с предусловием). Цикл с предусловием – это цикл, который повторяется до тех пор, пока условие выполняется (истинно). Блок-схема цикла с предусловием представлена на рисунке

Для реализации цикла с предусловием используется оператор While. В общем виде оператор While на языке программирования Паскаль может быть представлен так:

```
While <условие> do           заголовок цикла
<оператор>;                  тело цикла
```

Если тело цикла состоит из нескольких операторов, их нужно заключить в операторные скобки begin...end.

Значение выражения <условие>, записанное после слова While, проверяется перед каждым выполнением оператора (операторов) тела цикла.

Если <условие> верно (истинно), выполняется тело цикла, и снова вычисляется и проверяется выражение <условия>. Если результат проверки <условия> неверен (ложный), происходит выход из цикла.

Важно помнить:

Если условие сразу оказывается ложным, цикл с предусловием не выполнится ни разу!

В теле цикла должны быть операторы, которые в какой-то момент изменят значение условия, сделав его ложным. Если этого не случится, цикл будет бесконечным, то есть программа «зациклится».

Зациклившуюся программу следует остановить с помощью команды

Программа -> Завершить, иначе она будет выполняться бесконечно (точнее, до выключения компьютера).

В операторах for и while точка с запятой не ставится ни перед словом do, ни после него!

<i>Цикл с предусловием while (пока условие истинно)</i>	<i>Цикл с постусловием repeat (до истинности условия)</i>
1. До начала цикла должны быть сделаны начальные установки переменных, управляющих условием цикла, для корректного входа в цикл.	
2. В теле цикла должны присутствовать операторы, изменяющие переменные условия так, чтобы цикл через некоторое число итераций завершился.	
3. Цикл работает пока условие истинно (пока True).	3. Цикл работает пока условие ложно (пока False).
4. Цикл завершается, когда условие становится ложным (до False).	4. Цикл завершается, когда условие становится истинным (до True).
5. Цикл может не выполниться ни разу, если исходное значение условия при входе в цикл равно False .	5. Цикл обязательно выполняется как минимум один раз.
6. Если в теле цикла требуется выполнить более одного оператора, то необходимо использовать составной оператор.	6. Независимо от количества операторов в теле цикла использование составного оператора не требуется.
<i>Цикл со счетчиком for</i>	
1. Начальная установка переменной счетчика циклов до заголовка не требуется.	

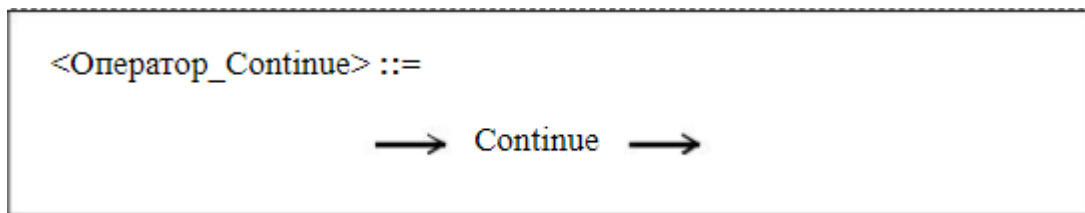
2. Изменение в теле цикла значений переменных, стоящих в заголовке цикла, не допускается.
3. Количество итераций цикла неизменно и точно определяется значениями нижней и верхней границ и шага цикла.
4. Нормальный ход работы цикла может быть нарушен оператором goto или процедурами Break и Continue .
5. Цикл может не выполниться ни разу, если шаг цикла будет изменять значение счетчика от нижней границы в направлении, противоположном верхней границе.

38. Операторы Continue и Break. Пример.

Данные операторы предназначены для гибкого управления операторами циклов For, While, Repeat.

Оператор Continue осуществляет передачу управления на конец тела цикла.

Формат оператора Continue

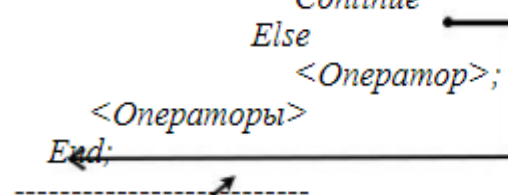


Данный оператор используется, если при некоторых условиях тело цикла или его часть выполнять не нужно. Осуществляет переход на конец тела цикла, после чего выполняется анализ условия дальнейшего выполнения цикла.

Пример 7.18.

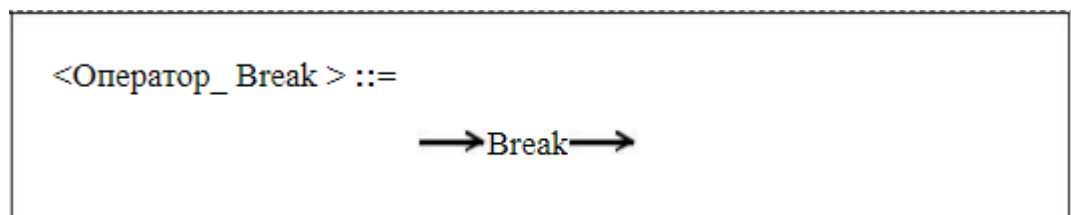
Фрагмент программы, использующей оператор *Continue*.

```
-----  
For I := 1 To N Do  
  Begin  
    <Операторы>;  
    If X > 0 Then  
      Continue  
    Else  
      <Оператор>;  
    <Операторы>  
  End;  
-----
```



По этой ветви осуществляется выход на конец тела цикла, затем изменяется значение параметра цикла *I* и выполнение тела цикла при следующем значении данного параметра.

Оператор Break служит для безусловного выхода из операторов For, Repeat и While.



Формат оператора Break:

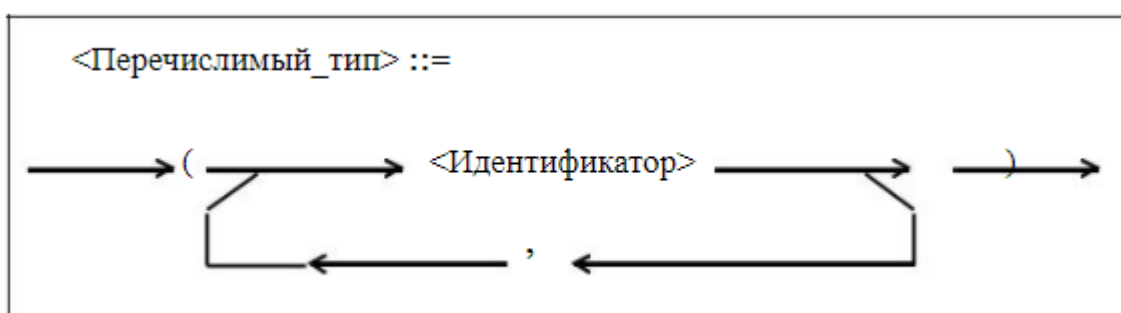
Если в примере 7.18 вместо оператора Continue записать оператор Break, то по отмеченной ветви осуществится выход из цикла: цикл при следующих значениях параметра выполняться не будет, осуществляется передача управления оператору, следующему за оператором цикла.

Использование операторов Continue и Break позволяет практически во всех программах обходиться без оператора безусловного перехода Goto.

39. Перечислимый тип данных. Формат задания. Способ упорядоченности. Представление в памяти. Операции и встроенные функции, определенные над данными перечислимого типа. Пример.

Перечислимый скалярный тип не является стандартным для языка Паскаль. Он должен быть описан программистом, поэтому относится к описанным скалярным типам.

Перечислимый скалярный тип определяется в программе с помощью задания типа, формат которого содержит рисунок 9.1.



Перечислимый тип задается указанием всех значений, которые может принимать переменная этого типа. Каждое значение является идентификатором и считается самоопределенной константой (константой определяемого типа). Константы в списке перечисления считаются перенумерованными, начиная с нуля, в порядке их перечисления: 0, 1, 2, 3.... (т.е. считаются упорядоченными так, что первое значение меньше второго, второе меньше третьего и т.д.).

Два определения перечислимых скалярных типов не должны содержать констант с одинаковыми именами в одной и той же области действия.

В памяти компьютера переменная перечислимого типа занимает один байт.

Пример перечислимого типа – тип Ned (Неделя), рассмотренный ранее при изучении структуры раздела описания типов (см. пример 6.4 в подразд.6.4). Данный тип может быть задан, например, следующим образом:

(Pn, Vt, Sr, Ch, Pt, Sb, Vs)

тип данных может быть задан двумя способами: явно в разделе описания типов Type или неявно в разделе описания переменных Var

Типы двух переменных называются идентичными, если в объявлении этих переменных используется один и тот же идентификатор типа или

используются различные идентификаторы T1 и T2, для которых имеется объявление вида:

Type

T1 = T2;

Идентичные типы являются совместимыми, т.е. переменные идентичных типов могут участвовать в одном выражении.

Если идентичные типы не являются файловыми типами, то они совместимы по присваиванию, т.е. значение одного типа может быть присвоено переменной другого типа.

Пример 9.1.

Раздел типов (явное задание типов).

Type

Ned = (*Pn*, *Vt*, *Sr*, *Ch*, *Pt*, *Sb*, *Vs*);

God = (*Yan*, *Fev*, *Mar*, *Apr*, *May*, *Iyn*, *Iyl*, *Avg*, *Sen*, *Oct*, *Nob*, *Dec*);

Citr = (*Mandarin*, *Apelsin*, *Limon*);

Fruct = *Citr*;

Здесь *Fruct* и *Citr* – идентичные типы.

Одно и то же имя константы не может использоваться в качестве значений в разных типах. Поэтому в примере 9.1 нельзя было бы написать:

Fruct = (*Mandarin*, *Apelsin*, *Limon*);

В данном случае типы *Fruct* и *Citr* трактуются как разные типы, а в них нельзя использовать одни и те же имена констант.

В примере 9.1 нельзя объявить, например, и такой тип:

Desert = (*Ananas*, *Vinograd*, *Grusa*, *Sliva*, *Apelsin*);

При объявлении типа *Desert* используется константа *Apelsin*. Однако она уже применена в другом типе (*Citr*).

Напомним, что при явном задании типа в разделе Type переменные данного типа должны быть объявлены в разделе Var с помощью указания имени типа.

Пример 9.2.

Задание переменных при явном задании типов. Типы заданы в примере 9.1.

```
Var
  Den, Den1: Ned;
  Mes, Mes1: God;
  C1, C2: Citr; F1,
  F2: Fruct;
```

Пример 9.3.

Неявное задание типов. Данные типы в разделе Type не задаются.

```
Var
  Den, Den1: (Pn, Vt, Sr, Ch, Pt, Sb, Vs);
  Mes, Mes1: (Yan, Fev, Mar, Apr, May, Iyn, Iyl, Avg, Sen, Oct, Nob,
  Dec); C1, C2, F1, F2: (Mandarin, Apelsin, Limon);
```

Над данными перечислимого типа определены следующие **встроенные функции**:

- Succ* (*x*) – следующее за *x* значение;
- Pred* (*x*) – предыдущее перед *x* значение;
- Ord* (*x*) – порядковый номер *x*;
- Sizeof* (*x*) – размер памяти, занимаемый аргументом; для перечислимого типа результат равен 1;

Пример 9.5.

Ввод и вывод значений перечислимого типа (с учетом объявлений, сделанных в примерах 9.1, 9.2).

```
Var
  i: 1..7;

Begin Read
  (i);
  Case i Of
    1: Den:= Pn;
    2: Den:= Vt;
    3: Den:= Sr;
    4: Den:= Ch;
    5: Den:= Pt;
    6: Den:= Sb;
    7: Den:= Vs
```

**40. Тип данных диапазон. Формат задания.
Базовый тип. Способ упорядоченности.
Операции и встроенные функции, определенные
над данными типа диапазон. Пример.**

Этот тип данных называют еще *ограниченным типом* или *интервальным типом*.

Тип диапазон задается **путем накладывания ограничений на базовый тип**. В качестве базового типа могут быть использованы скалярные типы, обладающие свойством перенумерованности (т.е. все скалярные типы, кроме вещественных).

Тип диапазон определяется в программе так, как представляет рисунок 9.2.

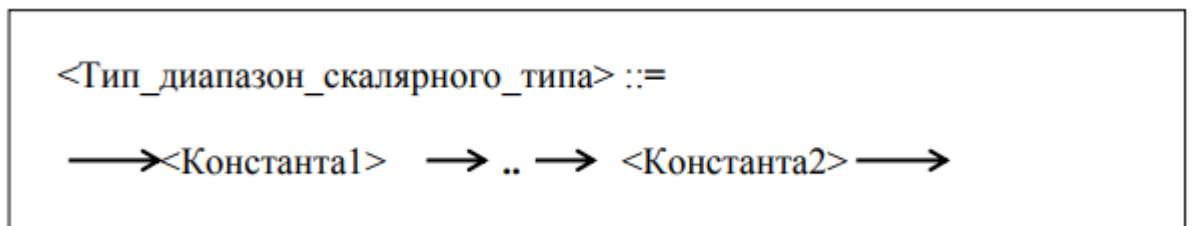


Рисунок 9.2 – Синтаксическая диаграмма
задания типа диапазон

Здесь <Константа1>, <Константа2>– это константы того базового типа, на основании которого вводится данный ограниченный тип. <Константа1> задает минимальное значение типа диапазон. <Константа2> задает максимальное значение типа диапазон.

Значение <Константа1> должно быть не более значения <Константа2>.

Тип диапазон является *перенумерованным*. Порядковый номер элемента совпадает с порядковым номером этого элемента в базовом типе.

Тип диапазон может быть определен либо в разделе типов, либо с помощью описания переменных в разделе *Var*.

Пример 9.6.

Объявление типа диапазон (применительно к примеру 9.1 предыдущего подраздела).

Type

Ned = (*Pn*, *Vt*, *Sr*, *Ch*, *Pt*, *Sb*, *Vs*);

God = (*Yan*, *Fev*, *Mar*, *Apr*, *May*, *Iyn*, *Iyl*, *Avg*, *Sen*, *Oct*, *Nob*, *Dec*);

Rabdn = *Pon* .. *Pyat*; {базовый *mun Ned*}

Vesna = *Mar* .. *May*; {базовый *mun God*}

Leto = *Iyn* .. *Avg*; {базовый *mun God*}

Int = -100 .. 50; {базовый *mun ShortInt*}

Ch = 'A' .. 'Z'; {базовый *mun Char*}

Здесь базовый тип определяется видом записи и величиной константы.

Var

Den: *Ned*;

Vix: *Sub* .. *Vos*;

Trud: *Rabdn*;

Vs: *Vesna*;

Lt: *Leto*;

I: *Int*;

Simvol: *Ch*;

J: 1 .. 10;

K: *Integer*;

Объявления переменных с помощью явного задания типа с именем в разделе *Type*

Объявления переменных с помощью неявного введения типа без имени

В данном примере типы *Rabdn*, *Vesna*, *Leto*, *Int*, *Ch* являются типами диапазон. Переменные *Vix*, *Trud*, *Vs*, *Lt*, *I*, *Simvol*, *J*, *K* имеют тип диапазон.

Таким образом, множество значений типа диапазон принадлежит множеству значений базового типа. К значениям типа диапазон применимы все операции и функции, определенные над значениями базового типа.

Тип диапазон совместим со своим базовым типом, т.е. значения типа диапазон могут использоваться везде, где могут использоваться значения базового типа.

В одном и том же выражении могут быть использованы переменные как типа диапазон, так и соответствующего ему базового типа.

Тип диапазон совместим с базовым типом и по присваиванию, но лишь при условии, что значение, присваиваемое переменной ограниченного типа, принадлежит соответствующему диапазону.

Пример 9.7.

Совместимость типа диапазон с базовым типом (применительно к предыдущему примеру).

```
Den := Sub;  
Trud := Pred (Den); {Trud станет равным Pyat, это входит в диапазон  
Rabdn}  
Vix := Sub;  
K := Ord (Vix); {K станет равным пяти (как в базовом типе)}
```

Но если в данном примере исходное значение $Den := Vos$, то оператор

$Trud := Pred (Den)$

вызовет ошибку при выполнении программы.

Если $Trud := Pyat$, то оператор

$Den := Succ (Trud)$

установит значение Den в Sub , что входит в диапазон значений переменной Den .

Достоинства типа диапазон:

- 1) позволяет транслятору экономнее использовать память при представлении значений переменных;
- 2) обеспечивает возможность контроля как на этапе трансляции, так и во время выполнения программы за

корректностью присваиваний, что помогает исправлять ошибки в программе;

3) обеспечивает наглядную форму представления решаемой задачи.

Тип диапазон широко применяется в комплексе с производными типами, в частности, в задачах обработки массивов.

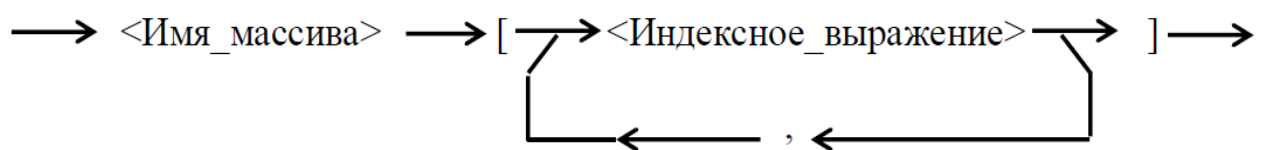
41. Массивы. Формат задания. Представление в памяти. Типы индекса. Полная и сокращенная формы задания многомерных массивов. Подмассивы. Пример.

Массив - упорядоченная совокупность однотипных элементов , имеющих общее имя.

- 1) Такое имя называют полной переменной;
- 2) Тип элементов - базовый тип;

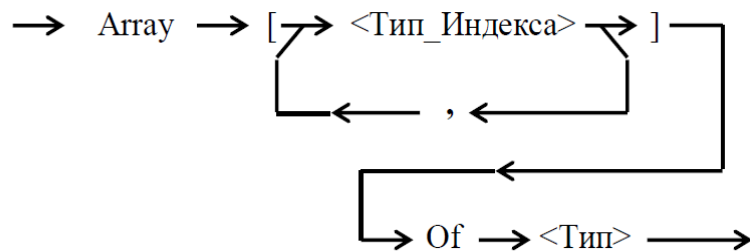
Обращение к элементам массива происходит по индексам.

<обращение_к_элементу_массива>::=



Индексное выражение – это выражение скалярного перенумерованного типа. Количество индексных выражений определяется количеством измерений массива.

<Тип_Array> ::=



Тип индекса:

- 1) Любой перенумерованный тип;
- 2) Количество элементов массива определяется количеством возможных значений этого типа;

//типа пример

type

 TMyArray = array[1..10] of integer;

 TSomeArray = array[Boolean] of Byte;

 Totherarray = array[Char] of Word;

//

//еще пример

const

 N=1000;

type

 TMyArray = array[1..N] of 18..70;

//

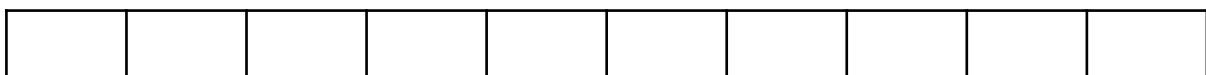
- В качестве индекса может быть указано любое выражение , имеющее тот же тип , что и индекс.
- В памяти массив представляет собой последовательно расположенные друг за другом переменные базового типа.

//

var

 a:array[1..10] of integer;

//



a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9] a[10]

Выход за границы массива - грубейшая ошибка!!!

- Может приводить к повреждению данных .
- В некоторых языках (например ,C/C++) приводит к неопределенному поведению.
- Часто такая ошибка долго не проявляется и обнаруживается при внесении значительных изменений в программу.

Элементами массива могут быть массивы.

Существует две формы записи:

- сокращенная ;
- полная.

//

const

 N=10;

 M=20;

type

 TArray = array[1..N,1..M] of Byte;

 TMyArray = array [1..N] of array [1..M] of Byte;

 TRow = array [1..M] of Byte;

 TArray = array[1..N] of TRow;

Обращение к многомерному массиву:

- a[i,j,k]
- a[i][j][k]

Оба способа взаимозаменяемы :

Пример: a[i][j,k]

Типы индексов могут быть различными.

Размерность массива языком не ограничивается.

Ограничения могут накладываться:

- адресное пространство.
- конкретная реализация компилятора .

Многомерный массив

var

 a:array[1..2,1..3] of Real;

a[1,1]	a[1,2]	a[1,3]	a[2,1]	a[2,2]	a[2,3]

[i..k,j..m]

↑ ↑

Строки Столбцы

42. Действия над массивами и над их элементами. Одномерная и многомерная константа-массив. Формат задания. Назначение. Пример.

Над элементами массивов определены те же операции, что и над их базовыми типами .

Индексированная переменная - элемент массива.

- Ведет себя , как и обычная переменная базового типа.

Над полными переменными(полная переменная - сам массив) не определено никаких операций.

Полные переменные могут использоваться в операторе присваивания.Условия:

- Массивы должны быть одного и того же типа.
- При этом происходит копирование всех элементов.

Для инициализации массива в языке Паскаль могут быть использованы типизованные константы типа массив.

```
//
type
    TArray = array [1..5] of Byte;
const
    A:Tarray = (12,6,73,8,2);
//
```

```
//
type
    TMatrix = array [1..2,1..2] of integer;
const
    M:TMatrix = ( (11,2),(55,-9));
//
```

Инициализация массива

Инициализация - задание начальных значений всем его элементам.

43. Динамические массивы. Синтаксис объявления. Представление в памяти. Основные операции над динамическими массивами. Подсчёт ссылок.

Динамический массив — массив, размер которого может изменяться во время выполнения программы.

Синтаксис представление.

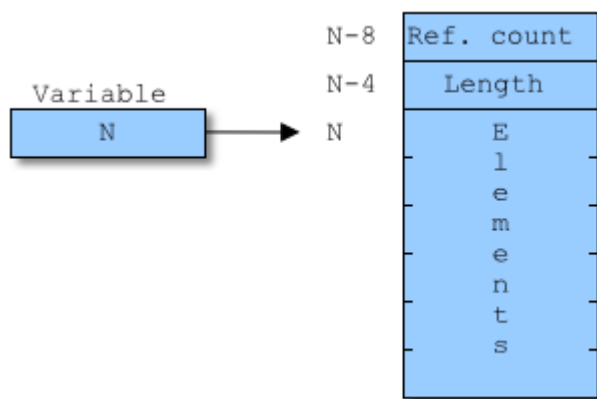
< Тип_динамического_массива >::=
→ array → of → < Тип_элементов > →

Пример:

```
var da_MyArray : array of integer;
```

Представление в памяти.

В участке памяти **ниже** адреса, на который указывает ссылка динамического массива, располагаются служебные данные массива: два поля - число выделенных элементов и *счётчик ссылок*.



Если, как на диаграмме выше, N - это адрес в переменной динамического массива, то *счётчик ссылок* массива лежит по адресу $N - 8$, а число выделенных элементов (*указатель длины*) лежит по адресу $N - 4$. Первый элемент массива (сами данные) лежит по адресу N .

Для каждой добавляемой ссылки (т.е. при присваивании, передаче как параметр в подпрограмму и т.п.) увеличивается счётчик ссылок, а для каждой удаляемой ссылки (т.е. когда переменная выходит из области видимости или при переприсваивании или присваивании `nil`) счётчик уменьшается.

Для выделения памяти для динамического массива в Delphi используется процедура **SetLength**:

```
SetLength(da_MyArray,20);
```

Операции над динамическими массивами.

Как только динамический массив был распределен, вы можете передавать массив стандартным функциям **Length**, **High**, **Low** и **SizeOf**. Функция **Length** возвращает число элементов в динамическом массиве, **High** возвращает самый высокий индекс массива (то есть **Length** - 1), **Low** возвращает 0. В случае с массивом нулевой длины наблюдается интересная ситуация: **High** возвращает -1, а **Low** - 0, получается, что **High** меньше **Low**. :) Функция **SizeOf** всегда возвращает 4 - длина в байтах указателя на динамический массив.

Для копирования необходимо использовать функцию **Copy**.

Пример: `da_A:=Copy (da_B);`

44. Строковые константы. Правила их записи в программе. Строковые переменные постоянной длины. Формат задания. Представление в памяти. Операции, определенные над строковыми данными постоянной длины. Пример.

Строковая константа — это последовательность любых символов, допускаемых для представления в компьютере, заключенная в апострофы.

Правила записи строковых констант в программе:

- 1) Если в строке необходимо поместить апостроф, то его повторяют дважды. При подсчете длины строки два рядом стоящих апострофа считаются одним символом.
- 2) При подсчете длины строки учитываются пробелы.
- 3) Допускаются пустые символьные константы, т.е. константы, не содержащие ни одного символа.
“ — это пустая строка нулевой длины.
- 4) Паскаль разрешает вставлять в строку символов управляющие символы. Символ # с целой константой без знака (от 0 до 255) обозначает соответствующий этому значению символ в коде ASCII. Между # и целой константой не должно быть никаких разделителей. Если несколько управляющих символов входят в строковую константу, то между ними не должно быть разделителей. Например: ‘TURBO’#13#10’TEKST’.

Строковая переменная постоянной длины определяется как одномерный массив символов:

```
Array [1 .. N] Of Char  
<Тип_индекса>
```

<Тип_индекса> может быть задан только с помощью типа диапазон, где N – длина строки ($N \geq 1$), определяемая как целое число без знака. При таком способе объявления строка обладает всеми свойствами массивов.

Особенности строковых переменных по сравнению с массивами:

1) Строковым переменным могут быть присвоены значения строковых констант (литералов), если длина строки равна длине литерала.

Пример:

```
Stroka1 := Literal;  
Stranitsa[I] := Literal;  
Stroka1 := 'ПРОГРАММА';
```

2) Над значениями строковых переменных можно выполнять операции сравнения (=, <>, >, <, >=, <=). Сравнение производится посимвольно, начиная с левой стороны до первого несовпадающего символа. Считается большей та строка, в которой первый несовпадающий символ имеет больший номер в коде обмена информацией (ASCII).

Доступ к символам строки осуществляется по индексам (нумерация начинается с 1).

Пример:

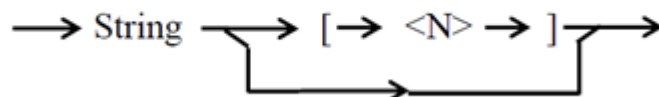
```

Type
  {Одномерный массив символов (строка)}
  Stroka = Array [1 .. 9] Of Char;
  {Двумерный массив символов (одномерный массив строк)}
  Stranitca = Array [1 .. 30] Of Stroka;
Var
  Stroka1, Stroka2: Stroka;
  Stranitca1, Stranitca2: Stranitca;
  I, K, J: 1..30;
  X, Y: Boolean;
Const
  Literal = 'Программа';
Begin
  -----
  {К этому моменту Stroka2 должна быть определена;}
  Stroka1 := Stroka2;
  {Одной строке может быть присвоено значение другой строки той же
  длины. Здесь K-ой строке страницы присваивается значение строки;}
  Stranitca1[K] := Stroka2;
  {Обращение к отдельным символам строковой переменной;}
  Stroka1[I] := 'A';
  Stroka1[J] := Stroka2[I];
  {J-ому символу I-ой строки страницы присваивается значение K-ого
  символа строки;}
  Stranitca1[I, J] := Stroka1[K];

```

45. Строки переменной длины. Формат задания. Представление в памяти. Операции, определенные над строковыми данными

<Задание_типа_String> ::=



и

переменной длины. Пример.

Строки переменной длины задаются при помощи типа String

N – максимальная длина строки. N должно быть ≤ 255. Если не указать N, то считается, что длина строки максимальная – 255. Текущая длина строки определяется длиной последнего занесенного в нее значения. Если длина присваиваемого значения больше указанной длины, лишние символы отсекаются. В памяти выделяется на 1 байт больше, чем длина строки. В нулевом байте хранится текущая длина строки



Над данными типа String определены операции сравнения и конкатенации(сцепления). Операция сцепления имеет больший приоритет. При сравнении строк большей считается та строка у которой первый несовпадающий элемент имеет больший код в таблице ASCII. Если символы строк одинаковые, но длины разные, большей считается та строка, которая длинней. Сцепление обозначается символом +. При ее выполнении две строки объединяются в одну. Результирующая длина не должна превышать 255 символов.

46. Встроенные процедуры и функции, определенные над строками переменной длины. Пример.

Встроенные функции:

Copy (S, from, count)

Копирует count символов из строки S, начиная с from.

```
S := 'ЧЕРТЁЖИК';
S1 := Copy(S, 5, 4);           // Теперь S1 содержит строку 'ЁЖИК'
```

Если параметры выбраны так, что требуется выделение большего количества символов, чем есть в исходной строке, выделяется только та часть, которая имеется в наличии:

```
S := 'ПАНТОГРАФ';
S1 := Copy(S, 6, 10);         // Теперь S1 содержит строку 'ГРАФ'
```

Concat (s1, s2, ..., sn)

Создает строку конкатенацией строк, переданные в качестве параметров.

Length (s)

Возвращает длину строки s

Pos (substr, s)

Возвращает позицию первого вхождения Substr в s.i

```
S := 'ПРОКРАСТИНАЦИЯ';
WriteLn(Pos('НАЦИЯ', S));     // Выведет 10
```

Но что эта функция должна вернуть, если заданная подстрока не встречается в заданной строке?

```
WriteLn(Pos('Н', S));         // Выведет 0
```

Встроенные Процедуры:

Delete (s, from, count)

Удаляет count символов, начиная с позиции from из строки s

```
S := 'КОРЗИНА';
Delete(S, 4, 3);    // Теперь в S записана строка 'КОРА'
```

Если позиция, с которой следует начинать удаление, выходит за границы строки, строка остаётся неизменной:

```
S := 'КАРТОНКА';
Delete(S, -1, 4);   // В переменной S по-прежнему строка 'КАРТОНКА'
Delete(S, 9, 2);    // И сейчас тоже ничего не изменилось
```

Insert (substr, s, from)

Вставляет substr в строку s так, чтобы первый вставленный символ оказался на позиции From.

```
S := 'ЧАС';
Insert('К НОРРИ', S, 3);    // Теперь в переменной S строка 'ЧАК НОРРИС'
```

Если позиция вставки меньше 1, она считается равной 1, т.е. подстрока будет вставлена в самое начало строки:

```
S := 'ВЕС';
Insert('НА', S, 0);         // Теперь в переменной S строка 'НАВЕС'
```

Если позиция вставки больше длины принимающей строки, вставляемая подстрока добавляется в конец строки:

```
S := 'ВЕС';
Insert('НА', S, 10);        // Теперь в переменной S строка 'ВЕСНА'
```

Str (Value, s)

Преобразует числовое значение величины Value и помещает результат в строку s. Величина Value должна иметь целочисленный или вещественный тип.

```
Str(Pi:0:4, S);    // Теперь в S строка '3.1416'
```

Val (s, value, errorcode)

Помещает в переменную value числовое значение строки s. В errorcode индекс первого недоступного символа(если таких нет, то errorcode = 0).

```
Val('123.45', X, Code);    // Теперь Code = 0, X = 123.45
```

Если же преобразование не удалось, в третий параметр будет записан индекс проблемного символа:

```
Val('38 попугаев', X, Code); // Теперь Code = 3
```

SetLength (S, NewLength)

Процедура SetLength, применённая к короткой строке, позволяет принудительно задать ей длину. Разумеется, новая длина не должна превышать максимальной длины, заданной при объявлении переменной.

SetString(S, Data, Len)

Ещё одна процедура, добавленная в Delphi, — SetString. Она позволяет произвольный участок памяти превратить в значение строковой переменной,

по сути — скопировать в строковую переменную произвольную последовательность байтов в памяти.

47. Динамические строки (Delphi-строки). Синтаксис объявления. Представление в памяти. Отличия в реализации от динамических массивов. Подсчёт ссылок.

Динамическая строка - последовательность символов неограниченной длины (теоретическое ограничение - 2гб). В зависимости от присваиваемого значения строка увеличивается и сокращается динамически.

Delphi-строки - управляемый тип данных. Он совмещает в себе лучшие качества Pascal-(реальная длина строки перед самой строкой в памяти) и C-(наличие символа конца строки - '\0') строк.

Это дает им следующие **преимущества**:

- *быстрота* выполнения операций над ними (не нужно всякий раз высчитывать длину строки);
- *Совместимость* с C-строками (Позволяет передавать Delphi-строки операционной системе, не выполняя дополнительных преобразований);
- *Экономия* памяти (подсчёт ссылок).

Объявление динамической строки:

```
var StringName: String;
```

(в зависимости от настроек компилятора данная запись может означать и объявление Pascal-строки максимальной длины - 255)

Переменная *StringName* хранит в себе **ссылку** на первый символ строки (отмечено розовым на схеме).

При инициализации delphi-строки она равна **nil** (пустая строка, == ' ').

Индексация символов начинается с **1**.

Строки **представляются в памяти** следующим образом:

(пример строки 'Hello!')

4 байта	4 байта	1	2	3	4	5	6	7
refCnt	6	'H'	'e'	'l'	'l'	'o'	'!'	0

Перед самой строкой располагается несколько битов (8) с дополнительной информацией о данной строке:

- **счетчик ссылок** (refCnt) - количество переменных, ссылающихся на эту строку;
- **длина строки** - реальная длина строки.

Когда счетчик ссылок становится равным нулю - память, занимаемой строкой становится "свободной"

У строковых констант счетчик ссылок равен -1 и никогда не изменяется.