Книги: [www.labirint.ru/books/339798](http://www.labirint.ru/books/339798)

*Этапы постановки и решения задачи на компьютере*

1. Четкая формулировка задачи
   1. Исходные данные
   2. Результаты.
   3. Форматы данных
2. Формальная (математическая) постановка задачи.
3. Выбор метода решения.
4. Разработка алгоритма решения задачи.
5. Выбор структур данных.
6. Программирование
7. Тестирование и отладка.
8. Выполнение программы (решение задачи).

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

Алгоритм – система правил, четко описывающая последовательность действий, которые необходимо выполнить для решения задачи.

*Свойства правильного алгоритма*

1. Дискретность.
2. Определенность (детерминированность) (все варианты опций)
3. Результативность.
4. Массовость.

*Способы описания алгоритмов*

* Запись на естественном языке (словесное описание)
* Изображение в виде схемы (графическое описание).
* Запись на алгоритмическом языке (ЯП)

*Запись на естественном языке*

* Этап обработки (вычисления)

V := выражение

* Проверка условия

Если условие, идти к N

* Переход к этапу с номером N

Идти к N

* Конце вычислений

Останов.

y = |a+b|

1. Sum := a+b
2. Если Sum >= 0, идти к… (5)
3. y := -Sum
4. Идти к… (6)
5. y := Sum
6. Останов.

*Изображение в виде схемы*

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

ISO 5807-85

ГОСТ 19.701-90 – Единая система программной документации.

Виды схем:

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

Символы в ГОСТ 19.701-90

1. Символы данных.
2. Символы процесса.
3. Символы линий
4. Специальные символы

Символы групп 1-3:

* Основные символы;
* Специфические символы;

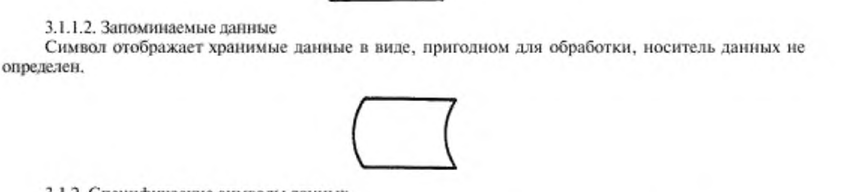
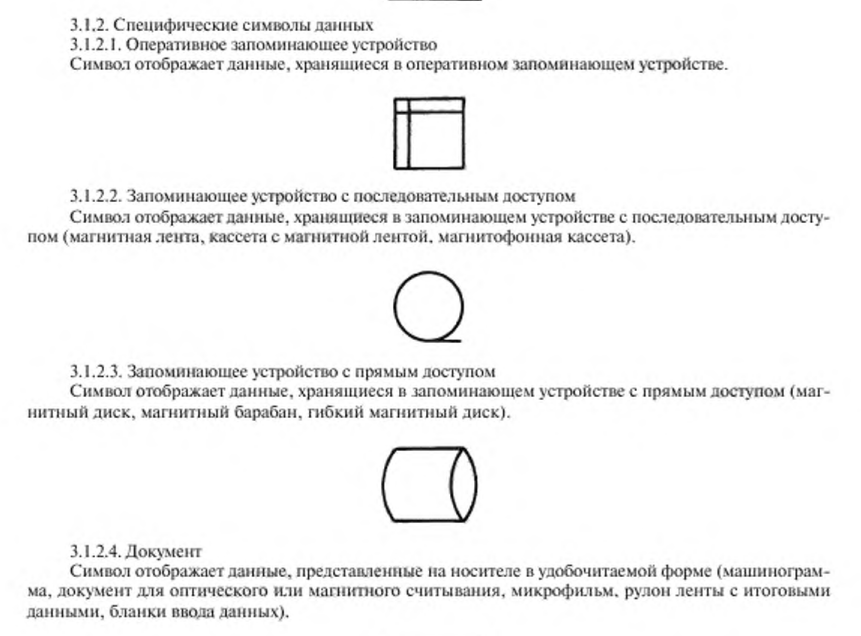
ДЗ:

1. ЭУМК (ЭРУД) по ОАиП (вк)
2. Изучить символы ГОСТ
3. Дополнить конспект

*Символы данных:*  
*Данные*

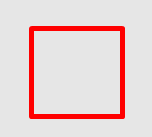


(Алгоритм only?)

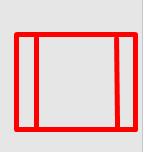


*Символы процесса:*

*Процесс*

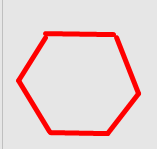


*Предопределенный процесс*



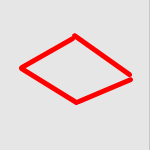
(Для детализации)

*Подготовка*

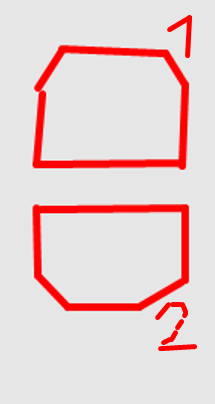


(Для языков низкого уровня)\

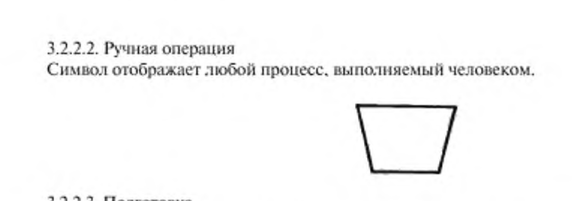
Решение

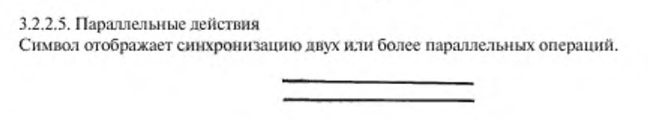


Граница цикла



Для циклов (1-начало, 2-конец





*Символы линий*

Линия

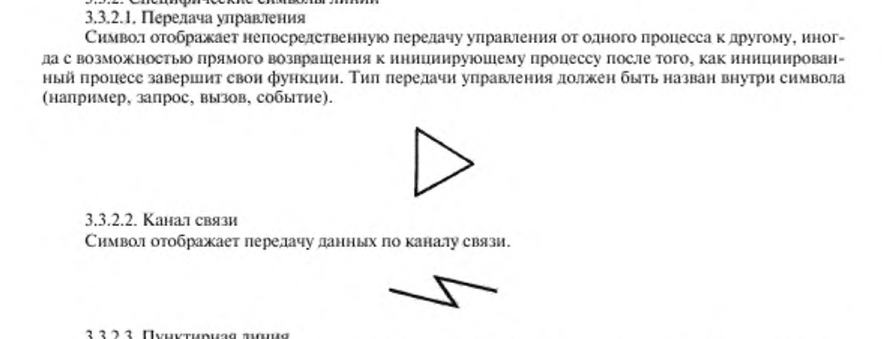


Или



Пунктирная линия

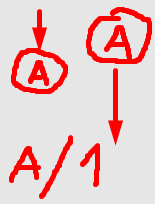


Специальные символы

Соединитель



Примеры:

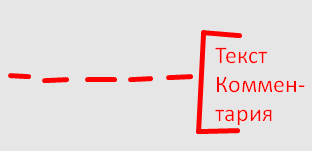


Терминатор



(Начало и конец алгоритма)

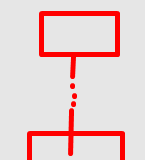
Комментарий



Пропуск



Пример:

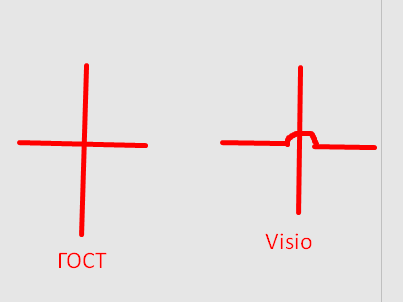


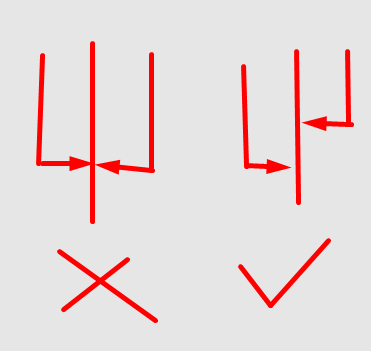
*Правила применения символов*

* 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 изображении (нежелательно)
* Внутри символа следует помещать минимальное количество текста необходимое для понимания его функции
* Текст записывается слева направо и сверху вниз.
* Если текст не помещается внутри символа, следует использовать комментарий.

Правила выполнения соединений

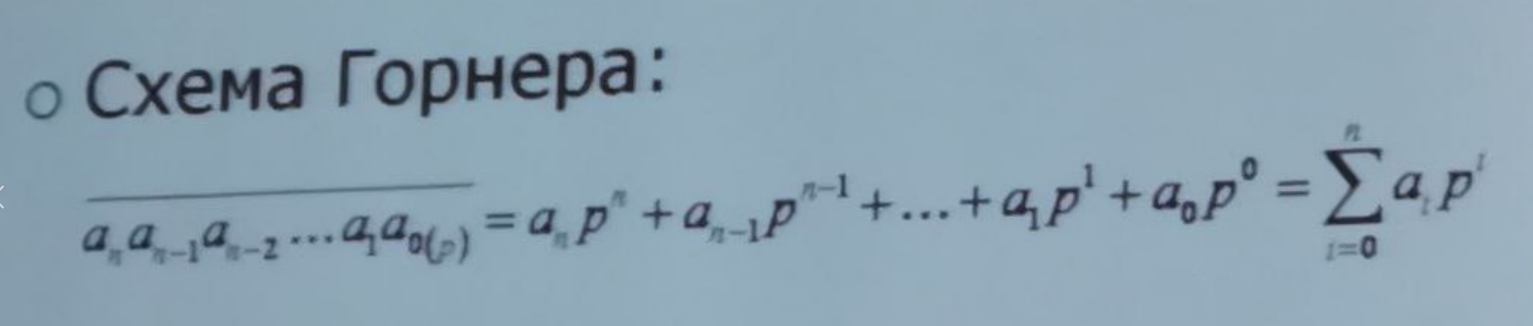
* Линии показывают направление потока управления
* Стандартные направления
  + Сверху вниз
  + Слева направо
* Если направление потока отличается от стандартного, оно должно указываться стрелками





* Линии входят слева или сверху.
* Линии выходят справа или снизу.
* Должны быть направлены к центру символа

***Системы счисления***



Перевод чисел между СС:

* Метод подбора степеней основания
* Метод деления/умножения на основание
* Метод использования особого соотношения оснований

Алгоритма преобразования числа N в р-ичную СС:

* Выбрать наибольшую степень р^n, которая будет меньше N
* Определить a.n – сколько раз p^n входит в N целиком
* Уменьшить N на a.n\*p^n
* Повторять, пока N не станет равно 0

163.375 = 101

35.375…

*Метод деления/умножения на основание*

Для перевода целой части:

* Делить на основание СС с остатком, пока делимое больше 0
* Выписать полученные остатки в обратном порядке.

Для перевода дробной части:

* Умножать на основание СС.
* Выписывать целые части

163/2 = 81.5

163/2 = 81 (1)

81/2 = 40 (1)

40 / 2 = 20 (0)

20 / 2 = 10 (0)

10 / 2 = 5 (0)

5 / 2 = 4 (1)

4/ 2 = 2 (0)

2 / 2 = (1) 0

10100011. …

0.375 \* 2 = (0).750

0.75 \* 2 = (1).5

0.5 \* 2 = (1)

0

10100011.011

*Метод использования особого соотношения оснований*

Если основания СС соотносятся как р1 = р2^к, то можно упростить перевод между ними.

Примеры:

2чная --- 8чная

2-чная --- 16чная

1010 0011

А 3

010 100 011

2 4 3

10010111  
01011010  
------------  
11110001

10010111  
01011010  
-------------  
00000000  
10010111

***Способы представления целых чисел***

Ширина/длина разрядной сетки (разрядность значения)

Прямой код

Обратный код

Дополнительный код

*Прямой код*

00101101 45(10)

*Обратный код*



Положительное число записывается, как в прямом коде

Отрицательное число записывается с инвертированием разрядов

01100111 103(10)

10011000 -103(10)

*Дополнительный код*



Положительное число записывается, как в прямом коде

Отрицательное число записывается с инвертированием разрядов

И прибавлением 1 к результату\

42 00101010 (2)

-42 11010110 (2)

00101010 (<>) – 11010101 (+1) – 11010110

00000000 0 (10)  
11111111 -1 (10)

10011011 -100

00110101 +53

11110000

*Способы представления вещественных чисел*

Два основных способа:

* С фиксированной точкой
* С плавающей точкой

Разряды делятся на две части:

* Старшие представляют целую часть
* Младшие - дробную

Сложение и вычитание чисел с ФТ производится так же, как и для целых

Умножение и деление требуют выполнения коррекции.

(т.к. по сути каждый из множителей имеет вид N.p^k)

*Плавающая точка*

Разряды делятся на три части:

* Знак
* Порядок
* Мантисса

Запись числа с плавающей точкой по смыслу эквивалентна экспоненциальной форме записи чисел

IEEE 754\

Знак (1) Порядок Мантисса

Single 8 23

Double 11 52

Extended 15 (1) 64

Нормализованная мантисса – мантисса, имеющая значение в диапазоне [1; p)

NaN – (qNaN/sNaN)

Сравнение вещественных чисел на точное равенство выполнять **нельзя!**

0.3(10) = 0.0100110011001…(2)яы

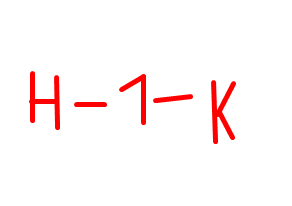
***Разновидности структур алгоритмов***

Структуры:

* Линейные
* Разветвляющиеся
* Циклические

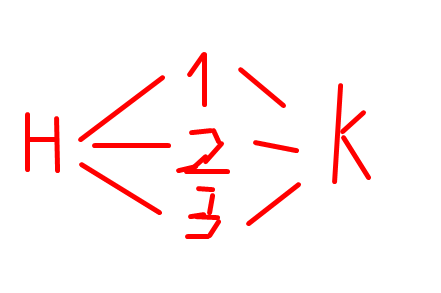
*Линейные вычислительные процессы*

Процесс, в котором направление вычислений является единственным



*Разветвляющийся вычислительный процесс*

Процесс, в котором направление вычислений определяется некоторыми условиями



*Циклический вычислительный процесс*

Процесс, в котором отдельные участки вычислений **могут** выполнятся многократно

Цикл – участок схемы, многократно повторяемый в ходе вычислений

Классификация циклов

По взаимному расположению

* Простые
* Сложные
* Вложенные
* Внешние

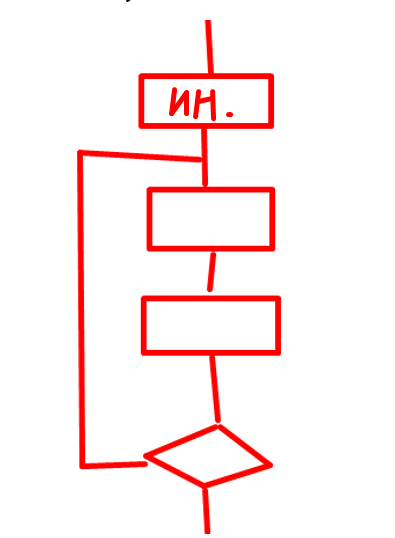
По местоположению условия выполнения цикла

* С предусловием
* С постусловием

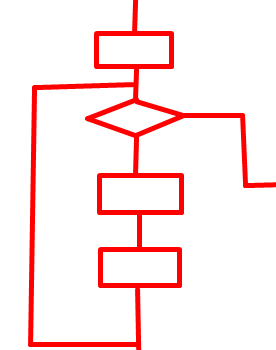
По виду условия выполнения цикла:

* Циклы с параметром
* Итерационные циклы

*Цикл с постусловием*



*Цикл с предусловием*

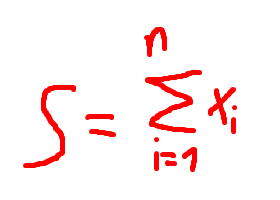


Другие названия

* Цикл со счетчиком
* Циклический процесс с известным количеством повторений

Пример:

* Вычислить сумму



*Итерационный цикл*

Циклический процесс, в котором количество повторений заранее неизвестно и зависит от получающихся в ходе вычислений результатов

Y = 1 + x/1! + x^2/2! + x^3/3! + …

До начала цикла:

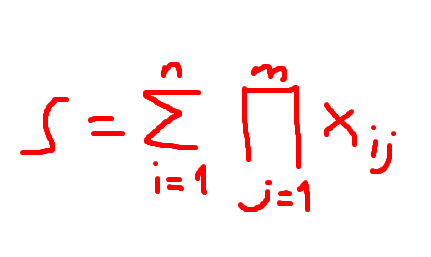
* Ycurr := начальное\_значение

На каждой итерации:

* Yprev := Ycurr
* Вычислить новое значение Ycurr
* Найти разность |Ycurr - Yprev|
* Если разность больше eps, повторить

***Сложные алгоритмы***

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



***Восходящее проектирование***

* Разработка начинается с простых алгоритмов
* Простые алгоритмы объединяются в более сложные
* Объединение продолжается, пока не будет получен алгоритм решения всей задачи

***Нисходящее проектирование***

* Разрабатывается алгоритм решения всей задачи в общем виде, без детализации
* Этапы алгоритма уточняются, пока не будет достигнут требуемый уровень детализации

***Язык программирования Delphi/Pascal***

Принцип программного управления

* Сигналы управления работой отдельных частей компьютера в процессе вычислений.
* Источником информации о требуемых типах сигналов на каждом шаге вычислений является код команды, считываемый из памяти компьютера.

Процессор постоянно:

1. Считывает очередную команду
2. Выполняет её

Команда представляет собой последовательность байтов

Любую последовательность байтов процессор попытается выполнить как команду (или последовательность команд)

Программа может записываться сразу в виде команд процессора (в машинном коде)

10110100 00001001

10111010 00001101 00000001

11001101 00100001

10110100 00001000

11001101 00100001

11000011

01001000 01100101 01101100 01101100

01101111 00101100 00100000 01110111

01101111 01110010 01101100 01100100

00100001 00100100

Для того, чтобы упростить написание программ, используют языки программирования (ЯП)

* Языки низкого уровня
* Языки высокого уровня

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

**ДЗ:**

* Дополнить конспект классификацией языков программирования

Языки программирования могут быть также классифицированы по различным критериям, но наиболее распространенные из них включают:

* По абстракции: низкоуровневые (ассемблер), среднеуровневые (C, C++), высокоуровневые (Python, Java).
* По показателю создания: интерпретируемые (Python, JavaScript), компилируемые (C++, Java).
* По сфере применения: научные (Matlab), веб-разработка (HTML, CSS, JavaScript), системные (С, C++), игровые (C#).
* По типу данных: динамически типизированные (Python, JavaScript), статически типизированные (C, Java).
* Императивные. Применяются для создания инструкций.
* Декларативные. Применяются не для решения проблемы, а для ее определения.
* Объектно-ориентированные. Применяются для обозначения объектов и классов, которые определяют данные и инструкции для управления данными.

*Инструменты программиста*

* Текстовый редактор (чтобы набирать тексты программ)
* Транслятор (чтобы преобразовывать программы из исходных кодов в машинный код) Бывают:
  + компиляторы,
  + интерпретаторы,
  + и др.
* Отладчик (дебаггер) (чтобы искать ошибки в программах)
* Документация (самый важный инструмент!)
* Вспомогательные библиотеки (чтобы не изобретать велосипед)
* Редактор связей (linker) (чтобы собирать программу из частей)
* И т.д.

*Среды программирования*

* Использовать инструменты по отдельности неудобно!
* Среда программирования – программный комплекс, включающий в себя инструменты для разработки программ
* Интегрированная СП (ide) – СП, в которой инструменты обмениваются информацией между собой.

**Грамматическое описание** любого языка программирования включает:

* Алфавит языка
* Синтаксис – правила построения фраз
* Семантика – смысловое значение фраз языка

*Алфавит языка Delphi/Pascal*

Буквы:

* Заглавные буквы латинского алфавита: A, B, …, Z
* Строчные буквы латинского алфавита: a, b, …, z
* Символ подчеркивания: \_

В программах на языке Delphi/Pascal прописные и строчные буквы эквивалентны (кроме строковых литералов)

Цифры:

* Десятичные арабские цифры: 0, 1, …, 9

Специальные символы (1 группа):

* Простые:
  + Символы-ограничители:
    - Знаки арифмет. операций:  
      + - \* /
    - Знаки операций сравнения:  
      < > =
    - Знаки-разделители:
    - . , : ; ‘
    - Знаки скобок:
    - ( ) [ ] { }
  + Другие символы:  
    @ # $ ^ **пробел**
* Составные специальные символы:
  + Присваивание  
    :=
  + Не равно  
    <>
  + Диапазон значений  
    ..
  + Знаки скобок:  
    (\* \*) (. .)
  + Однострочный комментарий  
    //

**ДЗ:**

1. Дополнить конспект описанием составных специальных символов

Специальные символы (2 группа):

* Служебные (зарезервированные) слова:

*Элементы программ*

Текст программы состоит из:

* Лексем
* Комментариев
* Пробелов

Лексема – неделимая последовательность знаков алфавита, имеющая в программе определенный символ («слова» ЯП) (нельзя разрывать)

* Специальные символы
* Идентификаторы
* Литералы (строковые и символьные константы)
* Числовые константы
* Метки

Идентификатор – это имя, которое позволяет однозначно выбрать один объект из множества объектов.

В Делфи/Паскаль идентификатор – любая последовательность **букв, цифр, знаков подчеркивания**

Начинающая с **буквы** или **знака подчеркивания** (б = \_)

Предопределенные (стандартные):

*Integer Real Sin …*

Определенные программистом:

*Name X ALPHA N\_18 …*

*Комментарии*

Используются для внесения пояснений в программу (в текст программы).

> ~~~~~~~~~~~~~~~~~~~~~~~~~~

*Способы описания синтаксиса*

Синтаксис – набор правил и соглашений, описывающих правильные предложения языка *(правила)*

Метаязык – формализованная система обозначений, применяемая для записи правил синтаксиса *(способ их записи)*

*Метаязыки*

Расширенная форма Бэкуса-Наура (РБНФ)

Синтаксические диаграммы

*РБНФ: основные понятия*

* Метаконстанта
* Метапеременная
* Метасимвол
* Синтаксическая единица

*РБНФ: метаконстанта*

* Используется для обозначения лексем языка программирования
* В программе метаконстанте соответствует она сама

“end” “+” “begin” “for”

*РБНФ: метапеременная*

Используется для обозначения конструкций языка

*<Шестнадцатеричная\_цифра>  
<Оператор\_while>  
<Объявление\_переменных>*

*РБНФ: синтаксическая единица*

Строка, описывающая состав и порядок следования элементов конструкций ЯП.

*РБНФ: метасимволы*

|  |  |
| --- | --- |
| Метасимвол | Описание |
| : := или = | «Определяется как», «по определению есть» |
| . | Конец определения |
| | | «либо», «или» (выбор альтернатива) |
| {} | Повторение (0,1,2,…) |
| [] | Необязательная часть (0 или 1) |
| (|) | Альтернативы |

*РБНФ: примеры*

<Индетификатор> ::=

(<Буква> | <Подч>) {<Буква> | <Цифра> | <Подч>}

<Буква> ::=

“a” | “b” | “c” | … | “z” | “A” | “B” | … | “Z”.

<Цифра> ::=

“0” | “1” | “2” | … | “9”.

<Подч> ::=

“\_”.

<Комментарий> ::=

<Однострочный\_к.> | <Многострочный\_к.>.

<Однострачный\_к.> ::=

“//” {<Всё\_кроме\_перевода\_строки>} <Перевод\_строки>.

…

*Синтаксические диаграммы*

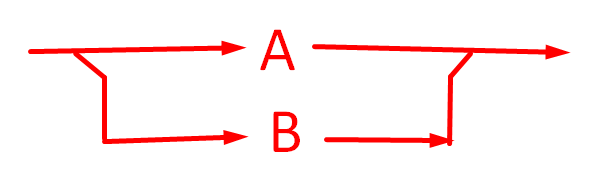
Синтаксическая диаграмма – ориентированный граф с размеченными рёбрами, используемый для описания синтаксической конструкции языка

* Рёбра помечены метапеременными и метаконстантами.
* Метасимволы не используются
* Метаконстанты записываются без кавычек

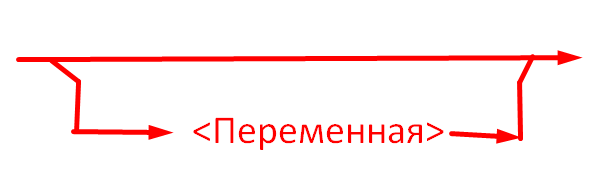
Выбор, альтернатива (метасимвол |)

<Переменная> ::= “A” | “B”.

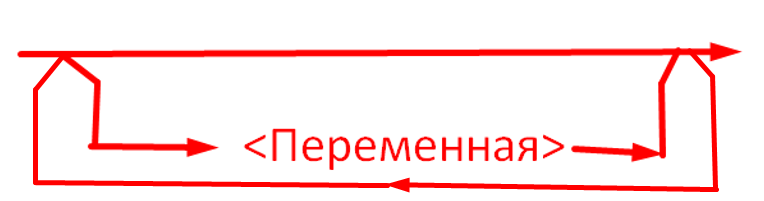
<Переменная> ::=



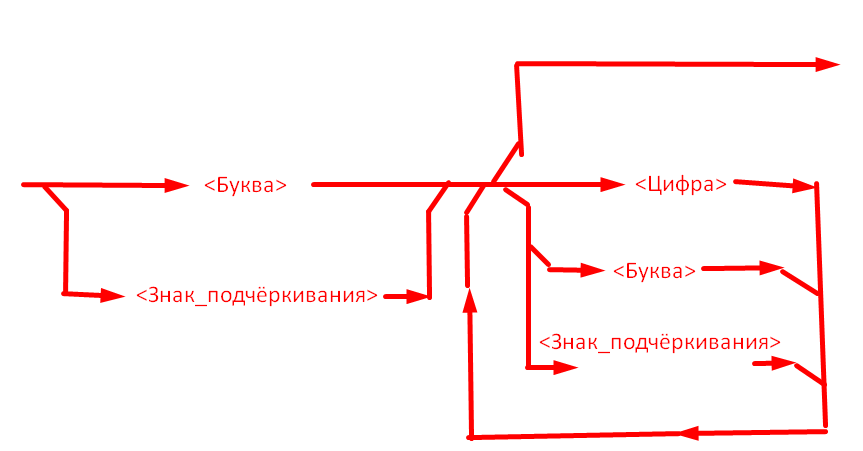
Необязательная часть (метасимвол [])



Повторение (метасимвол {})



<Идентификатор> ::=



**Данные в языке Delphi**

Константа – элемент данных, имеющий фиксированное значение.

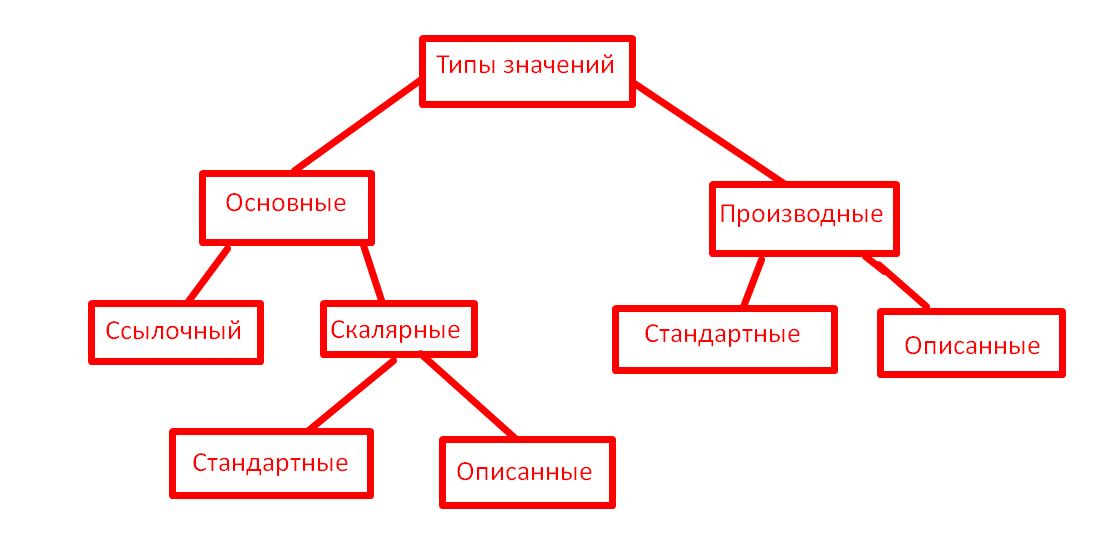
* Обозначается:
  + Своим значением: 42, 55, 897452
  + Именем: Pi, MaxSize.

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

* Обозначается:
  + Именем: x, ItemCount, BufSize.

Тип дынных (значений):

* Множество различных значений, которые могут принимать элементы данного типа
* Свойства данных значений
* Операции, которые можно выполнять над этими значениями



*Целочисленные типы*

|  |  |  |
| --- | --- | --- |
| Тип | Диапазон | Формат |
| ShortInt | -2^7 / 2^7-1 | 1 байт, зн. |
| SmallInt | -2^15 / 2^15-1 | 4 байта, зн. |
| LongInt | -2^31 / 2^31-1 | 4 байта, зн. |
| Byte | 0 / 2^8-1 | 1 байт, беззн. |
| Word | 0 / 2^16-1 | 2 байта, беззн. |
| LongWord | 0 / 2^32-1 | 4 байта, беззн. |
| Integer | -2^31 / 2^31-1 | 4 байта, зн.\* |

*Целочисленные константы*

Множество целочисленных значений является **перенумерованным.**

Целочисленные литералы:

Десятичные: 42, 875, -542  
 16-чные: $42, $C001C0DE

Тип целочисленного литерала – тип с наименьшим диапазоном, включающим значение литерала

*Операции над целочисленными данными*

|  |  |  |  |
| --- | --- | --- | --- |
| Операция | Вид | Описание | Тип результата |
| + | Одноместная | Сохранение знака | Целый |
| - | Одноместная | Отрицание знака | Целый |
| + | Двухместная | Сложение | Целый |
| - | Двухместная | Вычитание | Целый |
| \* | Двухместная | Умножение | Целый |
| / | Двухместная | Деление | Вещественный |
| div | Двухместная | Целочисленное деление | Целый |
| mod | Двухместная | Остаток целочисленного деления | Целый |

Поразрядные:

|  |  |  |  |
| --- | --- | --- | --- |
| Операция | Вид | Описание | Тип результата |
| not | Одноместная | Поразрядное дополнение целого | Целый |
| and | Двуместная | Поразрядное логическое умножение (И) | Целый |
| or | Двуместная | Поразрядное логическое сложение (ИЛИ) | Целый |
| xor | Двуместная | Поразрядное логическое исключающее ИЛИ | Целый |

Логические и поразрядные операции

Логические значчения:

* Истина (1)
* Ложь (0)

Логические операции:

* Операнды – логические значения.
* Логические операции могут быть заданы таблицами истинности

Поразрядные операции – выполняются так же, как и логические операции, но для отдельных разрядов операндов.

Операция and

|  |  |  |
| --- | --- | --- |
| X1 | X2 | Y |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

12 and 5 = 4

12(10) = 0000 1100(2)

5(10) = 0000 0101(2)

-------------------------

0000 0100(2) = 4(10)

Операция or

|  |  |  |
| --- | --- | --- |
| X1 | X2 | Y |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

12 or 5 = 13

12(10) = 0000 1100(2)

5(10) = 0000 0101(2)

-------------------------

0000 1101(2) = 13(10)

Операция xor

|  |  |  |
| --- | --- | --- |
| X1 | X2 | Y |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

12 xor 5 = 9

12(10) = 0000 1100(2)

5(10) = 0000 0101(2)

-------------------------

0000 1001(2) = 9(10)

Операция not

|  |  |
| --- | --- |
| X | Y |
| 0 | 1 |
| 1 | 0 |

Not 12 = 243

12(10) = 0000 1100(2)

-----------------------------

1111 0011(2) = 243(10)

Или -13, если используется знаковый тип (дополнительный код)

Опервации над целочислеными данными

Операции сдвига

|  |  |  |  |
| --- | --- | --- | --- |
| Операция | Вид | Описание | Тип результата |
| shl | Двухместная | i shl j – свдиг влево значения i на j бит | Тип i |
| shr | Двухместная | i shr j – свдиг вправо значения i на j бит | Тип i |

7 shl 1 = 14

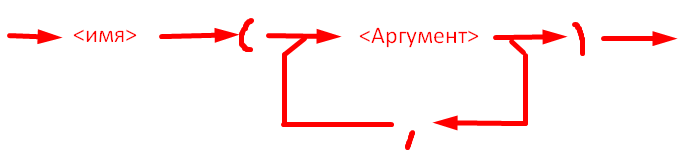
9 shl 2 = 36

24 shr 2 = 6

19 shr 1 = 9

|  |  |  |  |
| --- | --- | --- | --- |
| Операция | Вид | Описание | Тип результата |
| =  <>  <  >  <= >= | Двухместная | Равно  Не равно  Меньше  Больше  Меньше или равно  Больше или равно | Логический |

*Встроенные процедуры и функции*



|  |  |  |  |
| --- | --- | --- | --- |
| Название | Вид | Описание | Тип результата |
| Chr(x) | Функция |  | Char |
| Ord(x) | Функция |  | Integer |
| Abs(x) | Функция |  | Тип x |
| Sqr(x) | Функция |  | Тип x |
| Dec(x[, n]) | Процедура |  | - |
| Inc(x[, n]) | Процедура |  | - |
| Odd(x) | Функция |  | Boolean |
| Pred(x) | Функция | Возвращает предыдущий элемент из цепочки?? На 1 меньше  (Полезные не в интеджире) | Тип x |
| Succ(x) | Функция | Следующий соотв. На 1 больше | Тип x |
| Hi(x) | Функция | Возвращает 8 айт из числа левые ?? | Byte |
| Lo(x) | Функция |  | Byte |
| Swap(x) | Функция | Меняет страший и младший байт местами | Тип x\* |
| Random(n) | Функция |  | Integer\*\* |
| SizeOf(x) | Функция | Возвращает название переменной в байтах | Integer |

\* Параметр x (и результат) могут быть только типов SmallInt и Word. Функция оставлена для обратной совместимости.

\*\* Тип возвращаемого значения определяется компилятором

513 = $0201

Little-Endian (intel/amd)

|…|01|02|…| (bmp)

Big-Endian (Motorola, Network-order)

|…|02|01|…| (png)

*Вещественные типы*

Вещественные константы:

* Форма записи с фиксированной точкой:

0.25 -2.48 +31.0

* Форма записи с плавающей точкой:

14.3E5 681E-2 -5.16E-3

Представление вещественных чисел в памяти компьютера описывается стандартом IEEE 754

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Размер, байт | Диапазон | Точность, цифр |
| Real | 6 | 10^ +- 38 | 11/12 |
| Single | 4 | 10^ +- 38 | 7/8 |
| Double | 8 | 10^ +- 308 | 15/16 |
| Extended | 10 | 10^ +- 4932 | 19/20 |

*Операции над вещественными данными*

<> и = нельзя, но можно если аккуратно

Встроенные функции (вещественные типы)

|  |  |  |
| --- | --- | --- |
| Название | Описание | Тип результата |
| Round(x) | Округление к ближайшему целому | Integer\* |
| Trunc(x) | Целая часть x | Integer\* |
| Int(x) | Целая часть x | Extended |
| Frac(x) | Дробная часть x | Extended |
| Abs(x) | Модуль x | Extended\*\* |
| ArcTan(x) | Арктангенс x | Extended |
| Cos(x) | Косинус x | Extended |
| Sin(x) | Синус x | Extended |
| Exp(x) | Экспонента (e в степени x) | Real |
| Ln(x) | Натуральный логарифм x | Real |
| Sqr(x) | Возведение в квадрат | Extended |
| Sqrt(x) | Квадратный корень из x | Extended |
| SizeOf(x) | Размер переменной, байт | Integer |

*Вещественные константы*

Pi = 3.14… (extended)

*Символьный тип Char*

Размер – 1 байт

Способ упорядочения – в соответствии с таблицей символов

ASCII/ANSI:

* Управляющие и специальные символы
* Десятичные цифры
* Заглавные латинские буквы
* Строчные латинские буквы
* …

Константы типа Char:

‘g’ ‘A’ ‘z’ ‘8’ ‘Б’

#13 #42 #89 #$0D #$FE

Над значениями типа Char определены только операции сравнения

|  |  |  |
| --- | --- | --- |
| Название | Описание | Тип результата |
| Ord(x) | Порядковый номер символа | Integer |
| Pred(x) | Символ, порядковый номер которого на 1 меньше, чем у символа x | Char |
| Succ(x) | Символ, порядковый номер которого на 1 больше, чем у символа x | Char |
| UpCase(x) | Возвращает заглавную латинсую букву, если x – строчная латинсая буква, иначе x | Char |
| SizeOf(x) | Размер переменной, байт | Integer |

*Логический тип (Boolean)*

Логический тип – скалярный тип, имеющей 2 возможных значения:

* True
* False

1 байт

Not and or xor = <>

|  |  |  |
| --- | --- | --- |
| Название | Описание | Тип результата |
| Ord(x) | Порядковый номер значения x (0 для False, 1 для True) | Integer |
| Pred(x) |  | Char |
| Succ(x) |  | Char |
| SizeOf(x) | Размер переменной, байт | Integer |

Выражение

Выражение – формула для вычисления некоторого значения, состоящая из:

* Операндов
* Знаков операций
* Круглых скобок

Операнды – величины, участвующие в вычислениях:

* Константы
* Переменные
* Вызовы функций
* Выражения

*Приоритет операций*

1. Not, @
2. \* / div mod and shl shr
3. + - or xor
4. In = <> > >= < <=

*Тип выражений*

Каждое выражение имеет определённый тип

Тип выражения определяется типом результата его вычисления

Выражения скалярных типов

* Арифметические
* Логические
* Символьные

**Основные операторы языка Delphi**

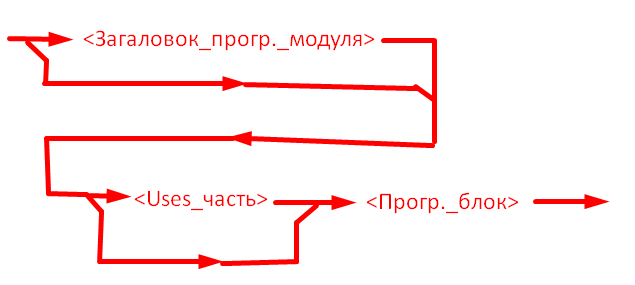
*Структура программы*

Программа состоим из одного или нескольких модулей

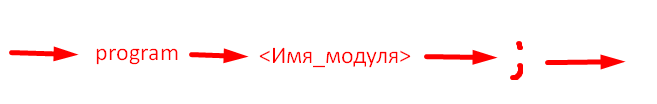
Модули:

* Программный модуль (основной)
* Модули unit (вспомогательные)

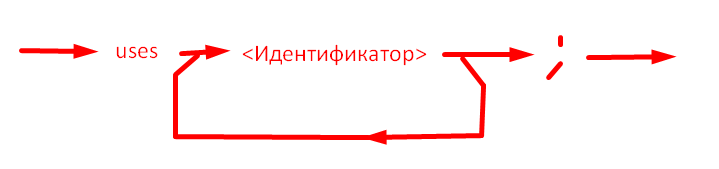
*Программный модуль*



<Загаловок\_прогр.модуля> ::=



<Uses\_часть> ::=



<Программный\_блок> ::=

---> <Раздел\_описаний> ---> <Раздел\_операторов> --->

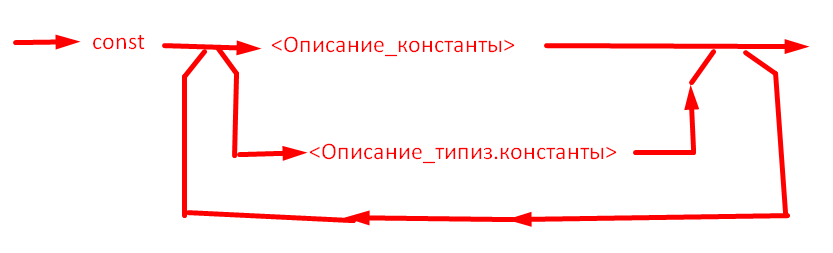
Раздел описаний:

* Объявление меток
* Объявление констант
* Объявление типов
* Объявление переменных
* Объявление процедур и функций

<Раздел\_меток> ::=



<Раздел\_констант> ::=



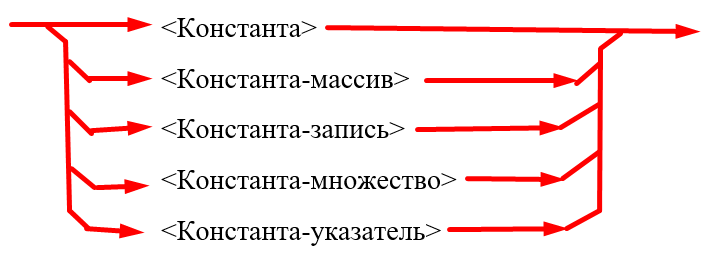
<Описание\_константы> ::=

**🡪** <Ид.> **🡪** = **🡪** <Константное\_выражение> **🡪**

<Описание\_типизированной\_константы> ::=

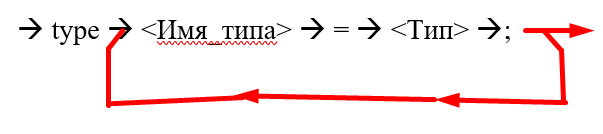
🡪 <Ид.> 🡪 : 🡪 <Тип> 🡪 = 🡪 <Типиз.\_константа> 🡪

<Типизованная\_константа> ::=

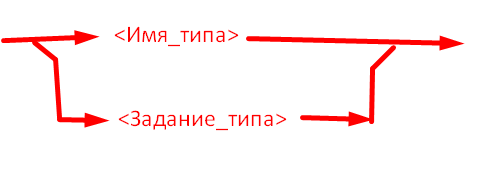


e: Real = 2.71;

<Раздел\_типов> ::=



<Тип> ::=

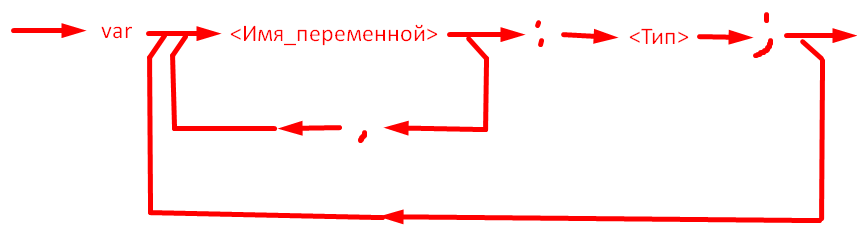


~~~~

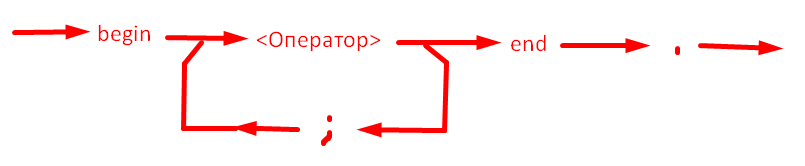
TMonth = (Jan, Feb, Mar, …);

TMatrix = array [1..N, 1..N] of Real’

<Раздел\_переменных> ::=



<Раздел\_операторов> ::=



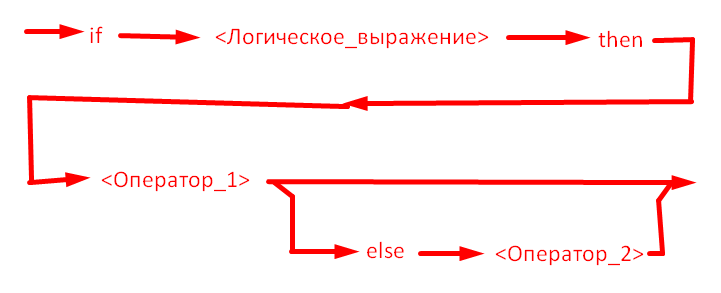
<Составной\_оператор> ::=



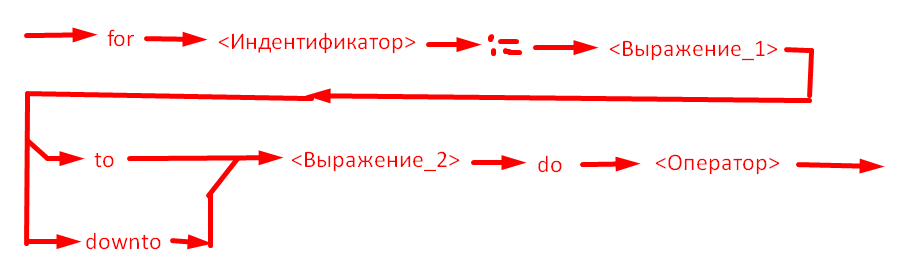
<Пустой\_оператор> ::=



<Оператор\_if> ::=



<Оператор\_for> ::=



*Оператор цикла с параметром for*

Перебирает значения переменной цикла из заданного диапазона.

Для каждого из значений выполняет тело цикла (оператор)

Следующее значение переменной цикла получает:

* Увеличением на 1, если to
* Уменьшением на 1, если downto

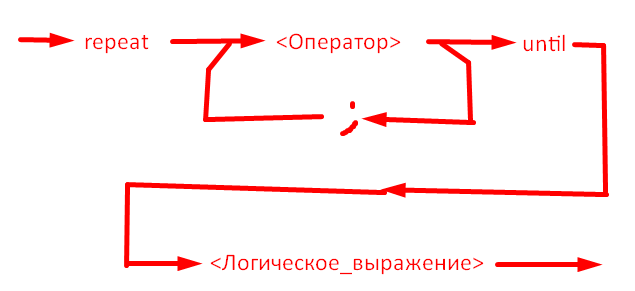
Изменения параметра цикла внутри тела for не допускается!

Значение параметра цикла после окончания for не определено.

<Оператор\_while> ::=

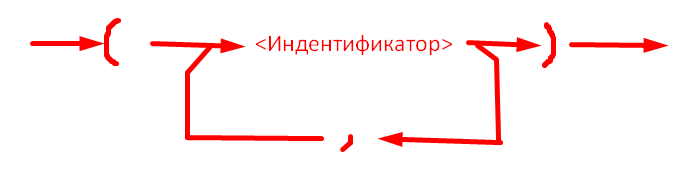


<Оператор\_repeat> ::=



**Описанные скалярные типы**

<Перечислимый\_тип> ::=



Задается перечислением всех своих значений.

* Каждый идентификатор считается константой этого типа.
* Константы считаются перенумерованными в порядке объявления (начиная с 0)

type

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

Одно и то же имя константы не может использоваться в нескольких перечислимых типах.

Имя константы перечислимого типа не может совпадать с именами констант, переменных или меток.

*Совместимость типов*

Типы двух переменных называются **идентичными**, если:

* При объявлении переменных использован один и тот же идентификатор типа;
* Существует объявление вида:  
  type  
   T1 = T2;

Переменные идентичных типов совместимы, т.е. могут учувствовать в одном выражении.

Идентичные типы (кроме файлового) совместимы по присваиванию:

* Т.е. значение одного из типов может быть присвоено переменной другого типа

Явное задание типа:

Type

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

Var

FDOW: TDayOfWeek;

Не явное задание типа:

Var

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

При неявном задании типа нельзя задать идентичные типы, т.к. у типа нет имени.

* Т.е. на этот тип нельзя сослаться в другом месте программы

*Перечислимый тип*

Определены только операции сравнения.

* Сравниваются порядковые номера (начиная с 0).

Значения перечислимого типа могут использоваться с оператором присваивания.

Определены встроенные функции:

Succ(x) Pred(x) Ord(x) SizeOf(x)

*Явное задание нумерации*

Type

TSize = ( Small = 5, Medium = 10, Large = Small + Medium);

*Тип диапазон*

Другие названия:

* Ограниченный тип
* Интервальный тип

Создается путем накладывания ограничений на другой тип.

* Такой тип называется базовым
* Базовым может быть любой скалярный перенумерованный тип.

<Тип\_диапазон> ::=

Тип диапазон – перенумерованный.

* Порядковый номер значения совпадает с порядковым номером этого значения в базовом типе

Type

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

TWorkingDay = Mon..Fri;

TDayOff = Sat..Sun;

Применимы все операции, которые применены к базовому типу.

Совместимы с базовым типом

* В том числе по присваиванию, но присваиваемое значение должно попадать в диапазон (быть допустимым в типе диапазон)

**Массивы и строки**

Типы значений – производные – описанные

**Массив** – упорядоченная совокупность однотипных элементов, имеющих общее имя

* Такое имя называется полной переменной
* Тип элементов – базовый тип массива

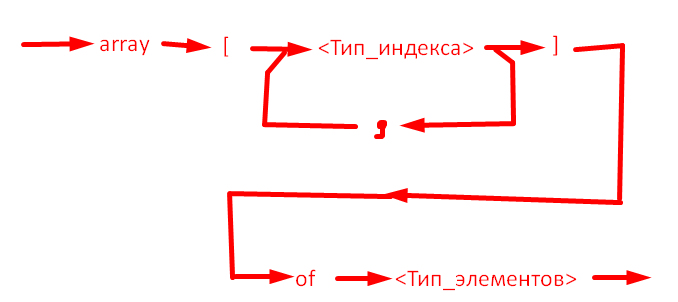
Обращение к элементам массивов происходит по **индексам**

<Обращение\_к\_элементу\_массива> ::=



Индексное выражение – выражение скалярного перенумерованного типа.

<Тип\_массива> ::=



Тип индекса:

* Любой перенумерованный тип
* Количество элементов массива определяется количеством возможных значений этого типа

Полученный массив не должен быть слишком большим

type

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

TSomeArray = array [Boolean] of Byte;

TOtherArray = array [Char] of Word;

TMyType = array [‘a’ .. ‘z’] of Real;

TBadType = array [Integer] of Int64 // Не даст объявить, не достяж. об. пам.

В качестве индекса может указываться любое выражение, имеющее тот же тип, что и индекс

В памяти массив представляет собой последовательно расположенные друг за другом переменные базового типа.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| А |  |  |  |  |  |  |  |  |  |  |
| 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 (C, C++, …) приводит к неопределенному поведению
* Часто такая ошибка долго не проявляется и обнаруживается при внесении значительных изменений в программу

Элементы массива могут быть массивы

Две формы записи:

* Сокращенная
* Полная

Type // const N = 10; M = 20;

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

TArray = array [1..N] of array [1..M] of Byte // Полная

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

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

*Многомерные массивы*

Обращение к многомерному массиву:

* A[i, j, k] // Сокращенная
* A[i][j][k] // Полная

Оба способа полностью взаимозаменяемы

* A[i][j, k]

Типы индексов могут быть различными

Размерность массива языком не ограничивается

* Ограничения могут накладывать:
  + Адресное пространство
  + Конкретная реализация компилятора
  + И т.д.

var

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| А |  |  |  |  |  |  |
| A[1,1] | A[1,2] | A[1,3] | A[2,1] | A[2,2] | A[2,3] |

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

**Индексированная переменная** – элемент массива.

* Ведёт себя, как обычная переменная базового типа

**Полная переменная** – сам массив целиком

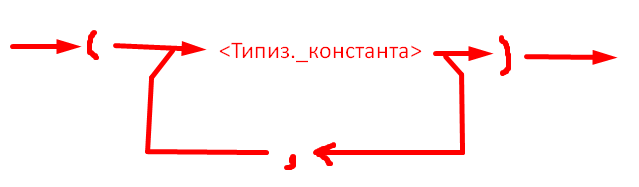
*Операции над массивами*

Над полными переменными не определено никаких операций.

Полные переменные могут использоваться в операторе присваивания.

* Массивы должны быть одного и того же типа.
* При этом выполняется копирование всех значений всех элементов
* Можно использовать и для подмассивов

<Типиз.\_константа\_массив> ::=



Type

TArray = array [1..5] of Byte

Const

A: TArray = (12, 6, 73, 8, 2);

*Типизированные константы-массивы*

Type

TMatrix = array [1..2, 1..2, 1..2] of Real;

Const

M: TMatrix = (((0,1), (2,3)), ((4,5), (6,7)));

*Инициализация массива*

Инициализация – задание начального значения переменной

Инициализация массива – задание начальных значений всем его элементам

Строки

**Строка** – последовательность символов.

Строковые данные:

* Константы
* Переменные

Строковый литерал – последовательность символов, заключенная в апострофы.

* Если в строке должен быть символ апострофа – он удваивается
* При подсчете длины строки учитываются пробелы

*Строковые константы*

Пустая строка – ‘’

Символ с определенным кодом можно вставить так:

‘First line’#13#10’Second line’

3 вида строковых переменных

* Строки постоянной длины
* Строки переменной длины

Array [1..N] of Char

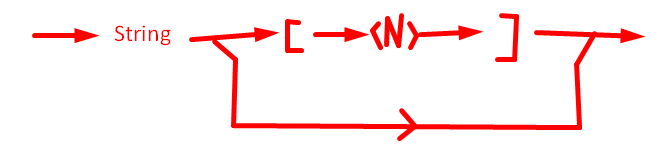
Такая строка обладает всеми свойствами массива

* Можно присвоить значение другой строковой переменной той же длины
* Можно обращаться к отдельным символам по индексам

В отличие от массивов:

* Можно присваивать строковые литералы, если их длина равна количеству элементов в массиве;
* Можно сравнивать строки одинаковой длины:
  + Сравнение производится посимвольно слева направо
  + Больше та строка, у которой первые несовпадающий символ имеет больший код

<Задание\_типа\_String> ::=



Строки переменной длины.

Необязательный параметр N задает максимальную длину.  
1 <= N <= 255

По умолчанию (если не задана) максимальная длина составляет 255 символов.

* Т.е. String эквивалентно String[255].
* Справедливо для классического Pascal.
* В Delphi всё несколько иначе

Var

S: String[9]

…

S := ‘Test’

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S | 4 | ‘T’ | ‘e’ | ‘s’ | ‘t’ | ? | ? | ? | ? | ? |
| Текущая длина строки | Символы строки | | | | Неиспользуемые байты | | | | |

Доступ к символам строки осуществляется по индексам.

* Нумерация начинается с 1.
* Обращение к символу с индексом не из диапазона 1..L аналогично выходу за границы массива (L – длина строки).

S[5] := #$30;

Операции (функции) над типом String:

* Операция конкатенации
* Операции сравнения

Copy(S, From, Count)

* Создает строку копированием **Count** символов начиная с позиции **From** из строк **S**

Concat(S1, S2, …, Sn)

* Создает строку конкатенацией строк переданных в качестве параметров

Length(S)

* Возвращают длину строки **S** в символах.

Pos(Substr, S)

* Возвращает позицию первого вхождения подстроки **Substr** в строку **S**.

Процедуры

Delete(S, From, Count)

* Удаляет из **Count** символов начиная с позиции **From** из строки **S**

Insert(Substr, S, From)

* Вставляет строку **Substr** в строку **S** начиная с позиции **From**.

Str(Value, S)

* Помещает в S строковое представление числа Value

Val(S, Value, ErrorCode)

* Помещает в переменную value числовое значение строки S в ErrorCode помещает недопустимого символа (0 при успешном преобразовании).

**Динамические массивы и строки**

*Способы представления строковых данных*

Существуют различные способы представления строк в памяти.

Для описания строки необходимо указать:

* Положение в памяти начала строки
* Положение в памяти конца строки

Основные способы представления строковых данных:

* Pascal – строки
* C – строки

Pascal – строки:

* Length-prefixed strings;
* P-strings.

C-строки

* Null-terminated strings;
* ASCIIZ strings.

*Pascal-строки*

| 6 | ‘H’ ‘e’ ‘l’ ‘l’ ‘o’ ‘!’ ? … ?

*Си-строки*

‘H’ ‘e’ ‘l’ ‘l’ ‘o’ ‘!’ | 0 | ? … ?

Символы строки | 0 – признак конца строки | Неиспользуемая часть

*Delphi-строки*

| RefCnt | | 6 | ‘H’ ‘e’ ‘l’ ‘l’ ‘o’ ‘!’ | 0

|-8| |-4|

*Счетчик ссылок*

Подсчет ссылок широко используется для решения автоматического управления памятью

Значение счётчика ссылок – количество переменных, ссылающихся на объект (строку, массив).

* Когда счетчик ссылок становится равным 0, память, занятая объектом (строкой, массивом), освобождается.

*Delphi-строки*

Тип String может соответствовать различным видам строк.

* String[N] – всегда Pascal – строки.
* String – Pascal- или Delphi-строки в зависимости от настроек компилятора.

Переменная типа “Delphi-строки” на самом деле является указателем.

* Присваивание nil позволяет удалить ссылку на строку.
* Значение nil эквивалентно пустой строке

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

На самом деле скрытых полей еще больше.

* Подробности в справке по Delphi.

При попытке изменения строки:

* Если счётчик ссылок равен 1, изменение происходит «по месту»
* Иначе создается копия строки в дальнейшем работа ведется с этой копией.
  + При этом счетчик ссылок уменьшается на 1 (кроме случая, когда он равен -1)

Delphi-строки – управляемый тип данных.

* Инициализируются компилятором в значение nil
* Управление занятой памятью и размерами осуществляется компилятором.

*Преимущества Delphi-строк*

Быстрое выполнение операций над строками.

* Не нужно каждый раз определять длины строк.

Совместимость с C-строками.

* Позволяет передавать Delphi-строки операционной системе, не выполняя дополнительных преобразований.

Экономия памяти.

* При присваивании строк не выполняется их копирование (в памяти остается один экземпляр).

*Динамические массивы*

Динамические массивы – массивы, размер которых может измениться во время выполнения программы.

<Тип\_динамического\_массива> ::=

🡪 Array 🡪 of 🡪<Тип\_элементов> 🡪

Индексация элементов всегда начинается с 0

MyArray: array of Integer;

N: Integer;

Readln(N);

SetLength(MyArrray, N);

Динамические массивы устроены так же, как Delphi-строки:

* Перед началом данных хранятся
  + Текущее количество элементов
  + Счетчик ссылок
* Значение переменной такого типа – указатель
* Значение nil эквивалентно пустому динамическому массиву.

Отличие от строк:

* В строках был в конце 0, в дин. массивах – нет.

SetLength(Arr, NewLen)

* Устанавливает размер динамического массива Arr в NewLen элементов.
* Если новый размер меньше старого, обрезаются последние элементы.

Length(Arr)

* Возвращает текущее количество элементов в динамическом массиве.

Var

A, B: array of Integer;

// Заполнение массива A

B := A; // Копия не создается

B := Copy(A) // А вось так создается

Динамические массивы могут быть многомерными:

var

Arr: array of array of TMyType;

SetLength(Arr, 5, 10);

Динамические массивы могут занимать много памяти.

* Если много больших элементов

Для удаления ссылки на динамический массив можно присвоить полной переменной nil

**Структурное программирование**

<Оператор\_goto> ::=

🡪 goto 🡪 <Метка> 🡪

Используется для безусловной передачи управления оператору, помеченному меткой

I := 1;

1:

If I > 10 then

Goto 2;

Writeln(I:7);

Inc(I)

Goto 1;

2:

Writeln(‘Good’);

Ограничения:

* Нельзя переходить внутрь производных операторов, не содержащих данный оператор goto:
  + Составной оператор, for, while, repeat…until, if, case, with и т.д.
* Нельзя переходить из одной альтернативы в другую в выбирающих операторах (if, case).
* Нельзя входить в подпрограмму или выходить из нее.

В рамках парадигмы структурного программирования использование оператора goto считается нежелательным

Подход к разработке программ.

Возник в 1970ых годах с появлением ЭВМ 3 поколения.

В основу положено требование:

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

При этом программа состоит из вложенных модулей, удовлетворяющих этому требованию.

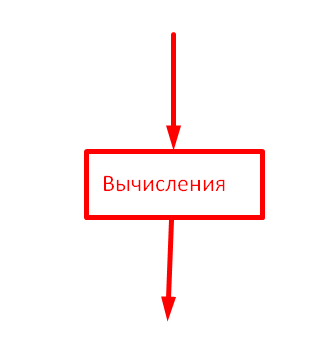
*Принцип Бома-Джакопини*

Любая программа может быть разработана с использованием **трех базовых структур:**

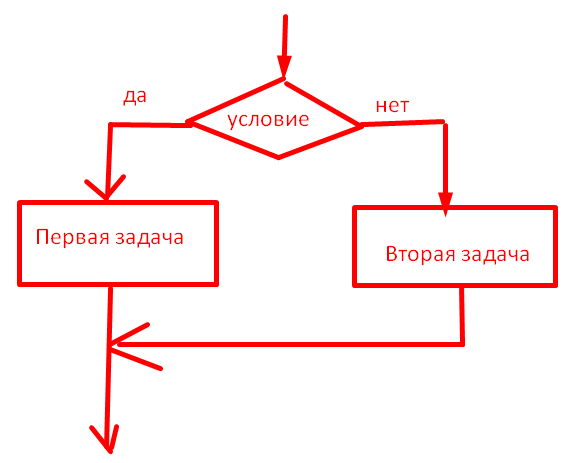
1. Функционального блока
2. Конструкции принятия двоичного (дихотомического) решения;
3. Конструкции обобщенного цикла

*Функциональный блок*

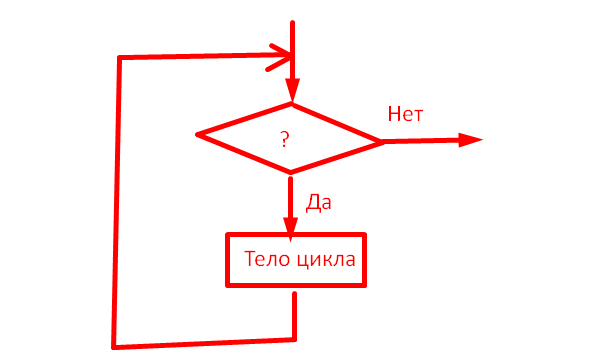
Отдельный оператор или последовательность операторов с одним входом и одним выходом



Конструкция принятия двоичного решения



Конструкция обобщенного цикла



Основные концепции:

* Отказ от использования оператора безусловного перехода goto
* Применение фиксированного набора управляющих конструкций
* Использование метода нисходящего проектирования

Достоинства:

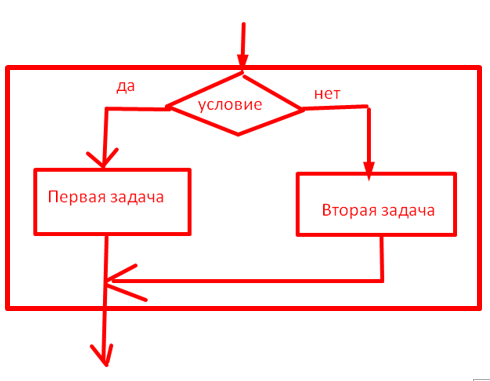
* Упрощение тестирования программ
* Повышение производительности программистов
* Повышение читаемости программ – упрощается их сопровождение
* Повышение эффективности объектного кода программ

*Преобразования Бома-Джакопини*

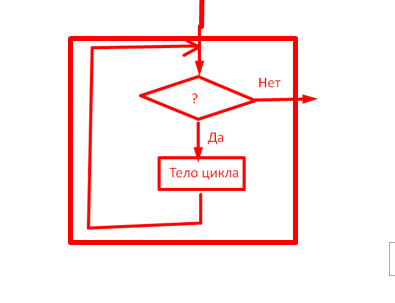
Используются для доказательства

* Правильности программы
* Структурированности программы

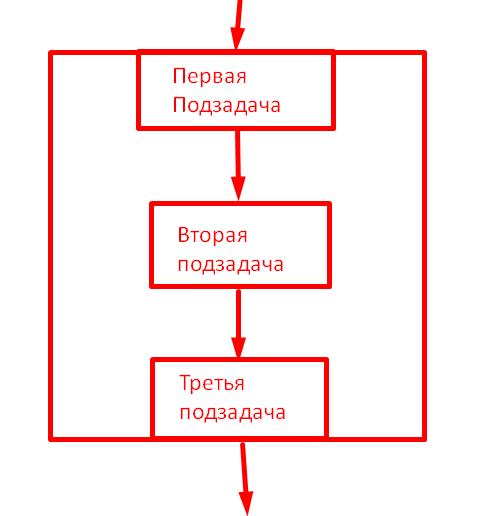
Первое преобразование:



Второе:



Конструкция следования



*Преобразования Бома-Джакопини*

Используются для доказательства

* Правильности программы
* Структурированности программы

Основаны на принципе «черного ящика».

Обратные преобразования могут использоваться для проектирования по методу нисходящего проектирования.

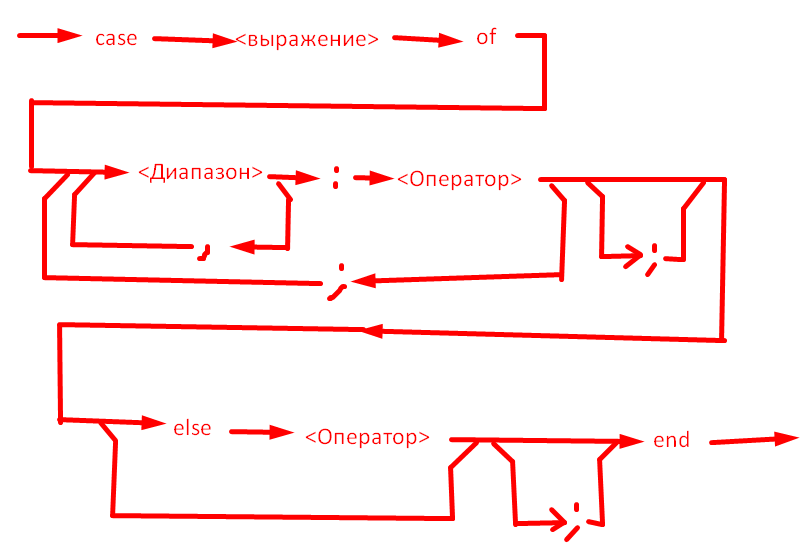
Все операции в программе должны представлять собой:

* Исполняемые в линейном порядке выражения
* Вызовы подпрограмм (обращение к замкнутому участку кода с 1 входом и 1 выходом)
* Вложенные на произвольную глубину операторы if-then-else
* Циклические операторы (while)

Иногда допускаются **расширения**:

* Дополнительные конструкции цикла:
  + Цикл с параметром
  + Цикл с постусловием
* Подпрограммы с несколькими входами или выходами\*
* Оператор goto с жёсткими ограничениями
* Оператор case как расширение if

<Оператор\_case> ::=



case symbol of

‘A’..’Z’, ‘a’..’z’:

Writeln(‘Letter’);

‘0’..’9’:

Writeln(‘Digit’);

Else

Writeln(‘Other character’);

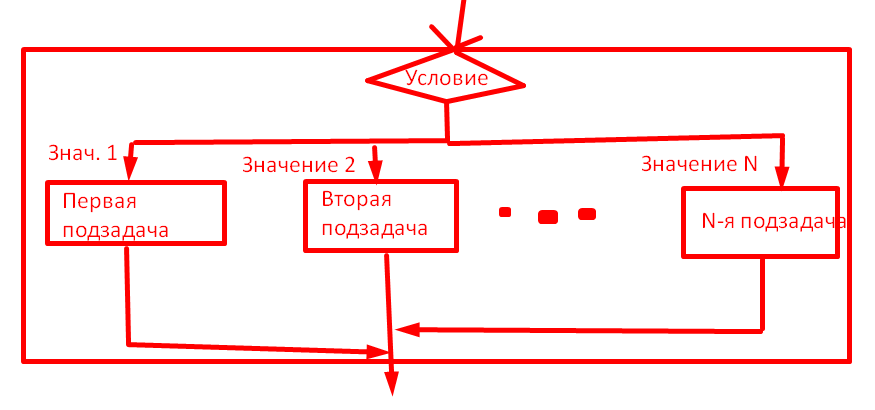
end

Выполняется только одна ветвь – та, которая соответствует значению селектора (выражения).

* Выражение-селектор должно быть перенумерованного типа.

Если подходящей ветви нет, выполняется ветвь else

* Если нет и её, не выполняется ни одна из ветвей



<Оператор\_break> ::= 🡪 break 🡪

Осуществляет досрочный выходи из цикла (не по условию цикла)

Является по сути оператором goto с ограничениями

<Оператор\_continue> ::= 🡪 continue 🡪

Осуществляется переход к концу тела цикла (т.е. к месту после последнего оператора тела цикла).

Является по сути оператором goto с ограничениями

Операторы break и continue нарушают принципы структурного программирования:

* Break позволяет создать цикл с несколькими выходами
* Continue позволяет создать конструкции с несколькими выходами внутри тела цикла.

Тем не менее, их использование **иногда** допускается

Применение оператора goto также иногда может быть оправдано.

* Выход из нескольких вложенных циклов.
* Сложная обработка ошибок
* Автоматически сгенерированный код

**Преобразование неструктурированных программ  
в структурированные**

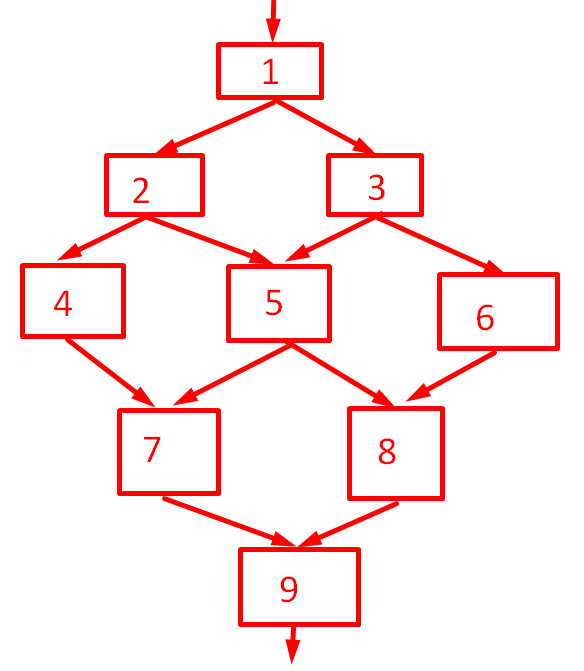
Структурированность – это свойство программы/алгоритма.

Методы преобразования неструктурированных программ в структурированные:

* Метод дублирования кодов
* Метод введения переменной состояния
* Метод булевого признака
* И др.

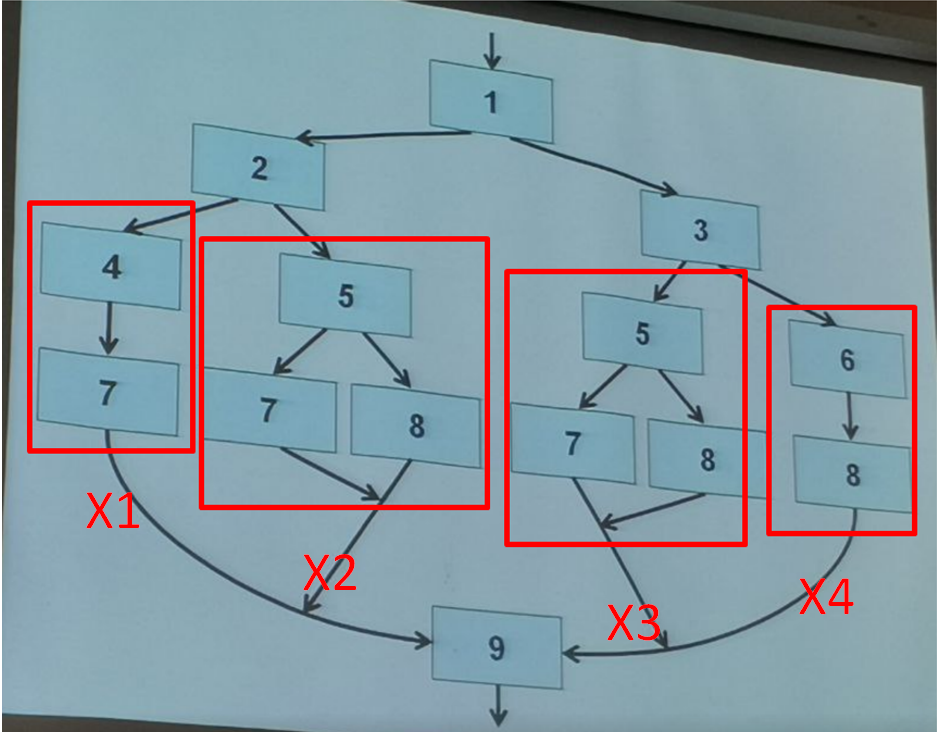
*Метод дублирования кодов*

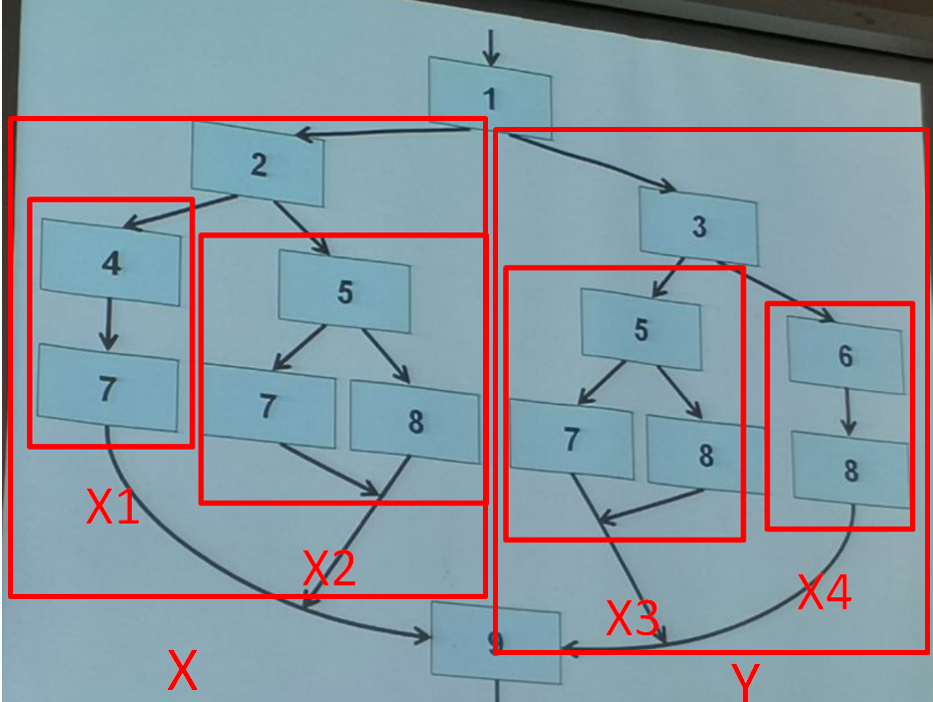
Неструктурированный алгоритм:

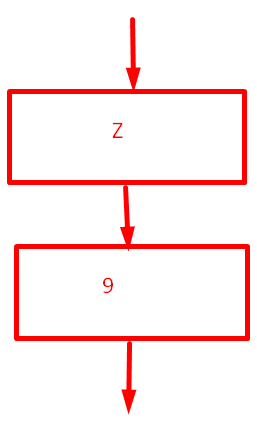


Сущность метода:

* Дублируются модули исходного алгоритма/программы, имеющие несколько входов (кроме последнего модуля).







Преимущества:

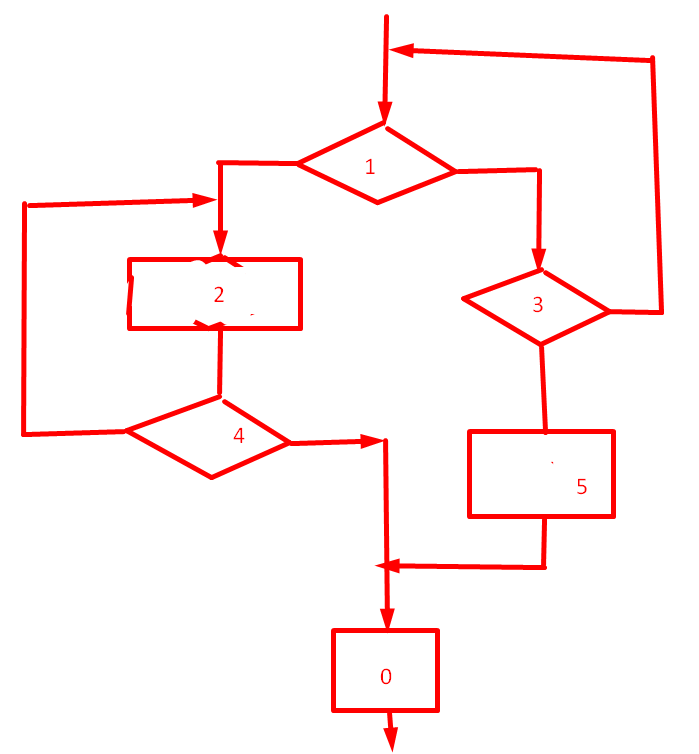
* Можно использовать как промежуточный шаг при разборе запутанного кода

Недостатки:

* Неприменим для циклических алгоритмов
* Увеличение объема кода (громоздкие схемы алгоритмов)

*Метод введения переменной состояния*

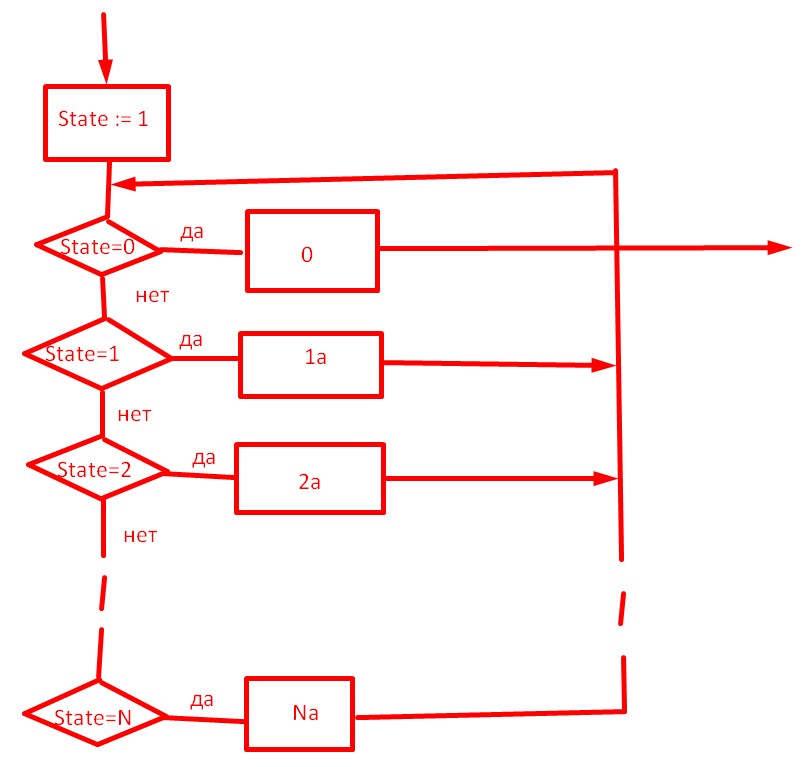
Предложен Ашкрофтом и Манной

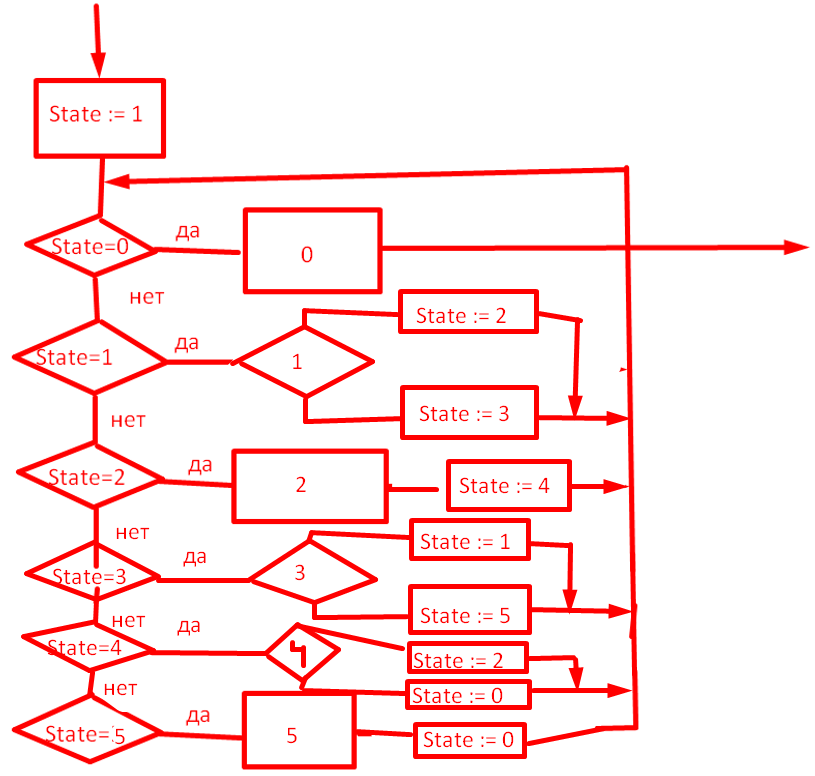


1. Каждому блоку исходной схемы приписывается номер:

* Обычно нумерация начинается с 1, последний блок получает номер 0.

1. Вводится дополнительная целочисленная переменная – **переменная состояния**
2. Функциональные блоки исходной схемы заменяются блоками, которые помимо основной задачи выполняют изменение переменной состояния.
3. Логические блоки преобразуются аналогично:
   * В каждой ветви своё новое значение переменной состояния
4. Исходная схема перестраивается к виду, предложенному Ашкрофтом и Манной.





Достоинства:

* применим к алгоритмам любой структуры (в т.ч. и циклическим)
* возможно автоматическое применение данного метода

Недостатки:

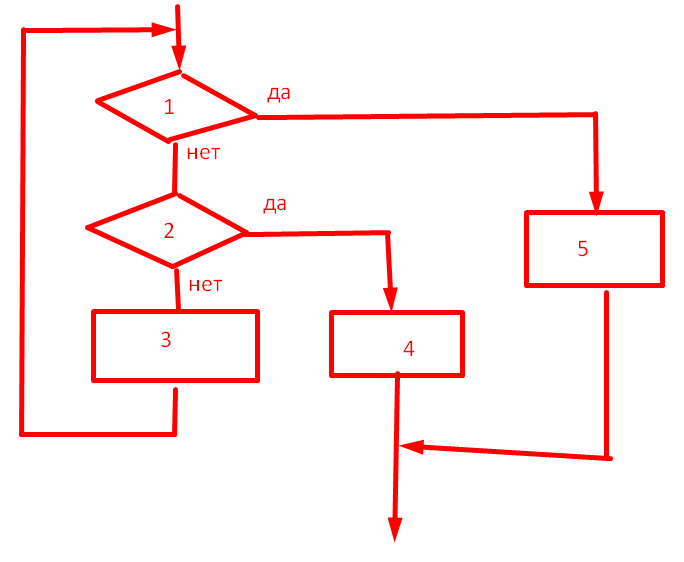
* топология схема алгоритма сильно изменяется
* затраты времени на проверку и изменение значения переменной состояния
* громоздкость результирующей схемы алгоритма

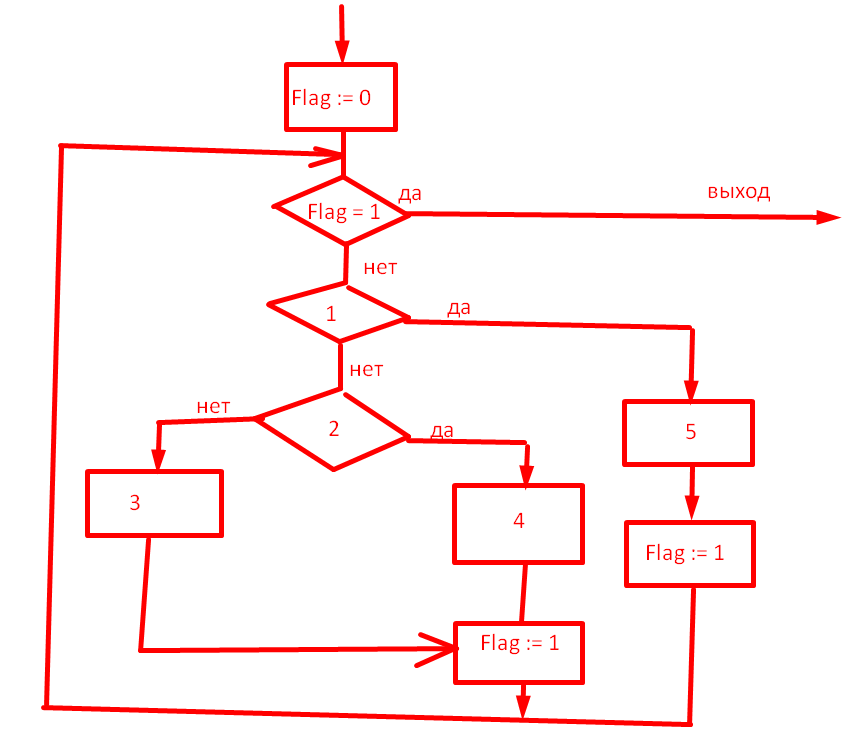
Можно рассматривать как разновидность автоматного программирования.

*Метод булевого признака*

Суть метода:

* в программу, содержащую 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 циклов.





Достоинства:

* компактность, экономичность
* топология схемы изменяется незначительно

Недостатки

* применим только для алгоритмов с циклами.

Иногда можно обойтись без введения дополнительной

**Графическое представление структурированных алгоритмов**

ГОСТ позволяет построить схему алгоритма, не являющегося структурированным.

Другие способы:

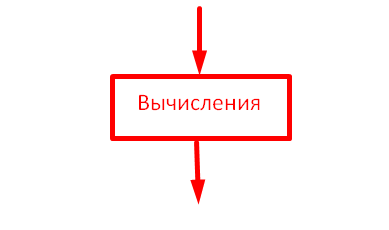
* метод Дамке
* схемы Насси-Шнейдермана

*Метод Дамке*

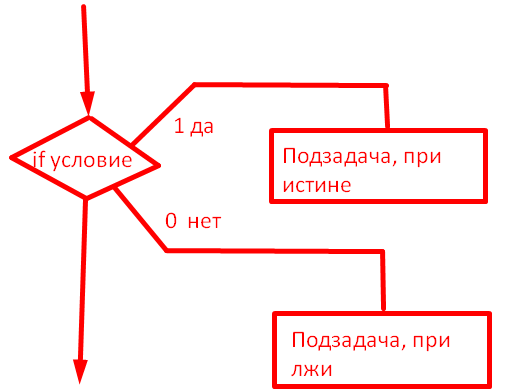
Три базовых конструкции:

* функциональный блок
* конструкция if then else
* конструкция цикла с предусловием

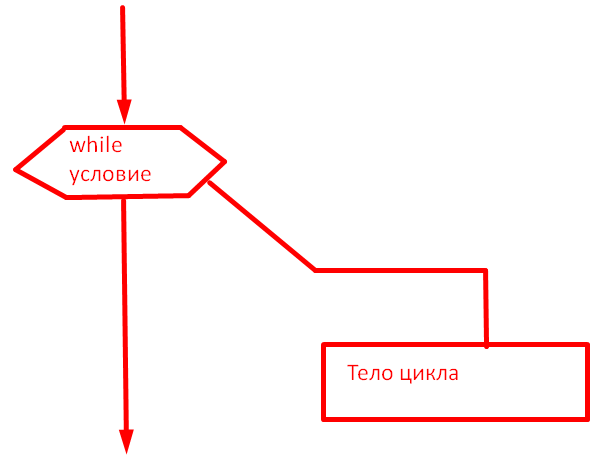
Функциональный блок:



Конструкция if then else:



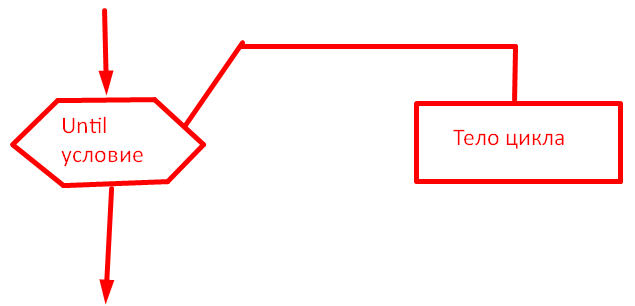
Конструкция цикла с предусловием



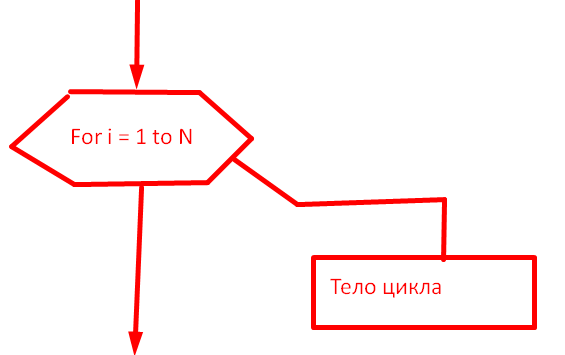
Допускаются дополнительные конструкции:

* конструкция цикла с постусловием
* конструкция цикла с параметром
* конструкция Case

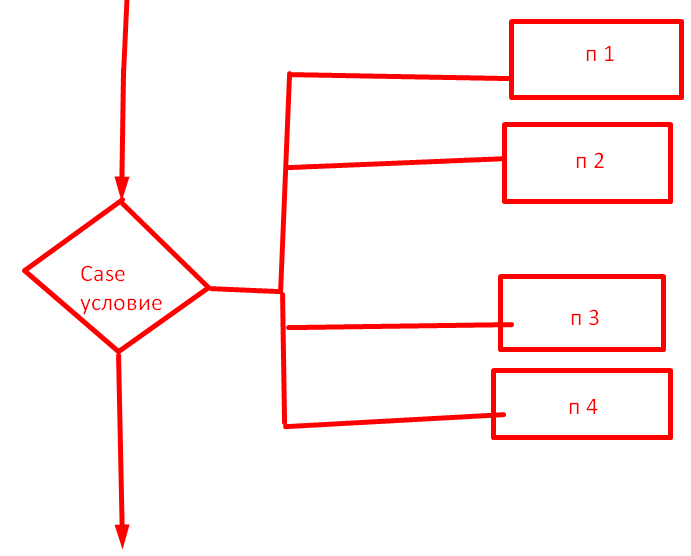
Конструкция цикла с постусловием:



Конструкция цикла с параметром:

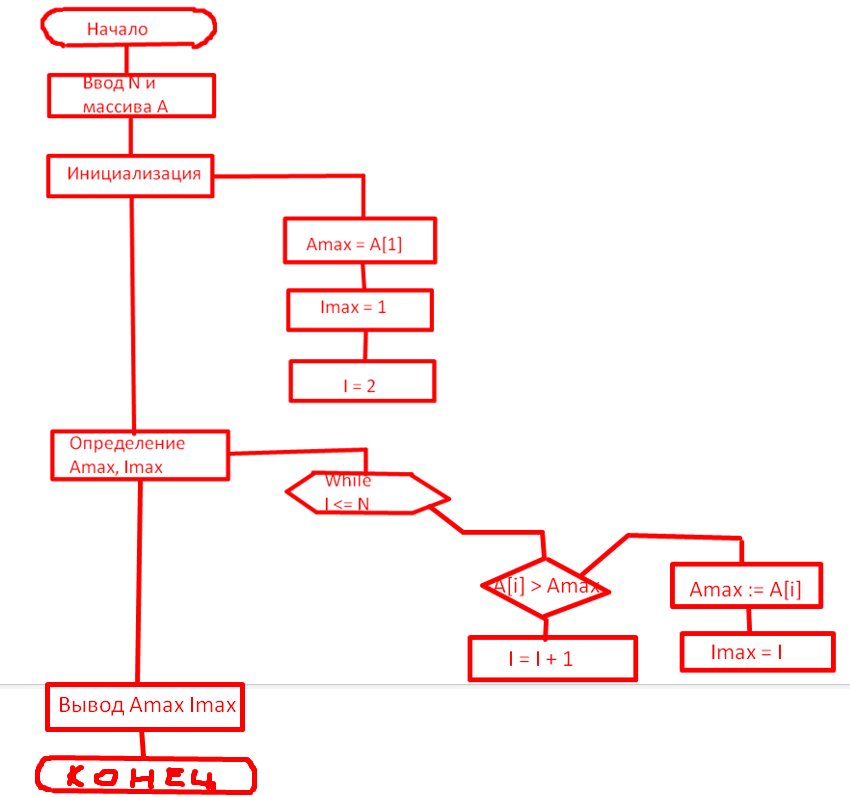


Конструкция Case



Основной принцип построения схемы алгоритма по методу Дамке – принцип декомпозиции.

* Элементы, расположенные левее, представляют укрупненную структуру алгоритма.



Преимущества:

* Не позволяет разработать схему неструктурированного алгоритма
* Удобно использовать для нисходящего проектирования
* Наглядность для больших программ
* Удобство коллективной разработки

Недостатки:

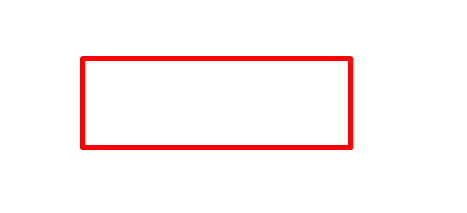
* Схемы могут быть более громоздкими

*Схемы Насси-Шнейдермана*

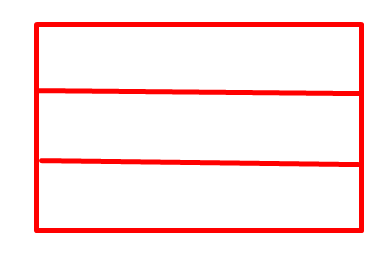
Другое название – структурограммы.

* Иллюстрируют структуру передачи управления с помощью вложенных блоков.
* Отсутствуют линии, указывающие передачу управления

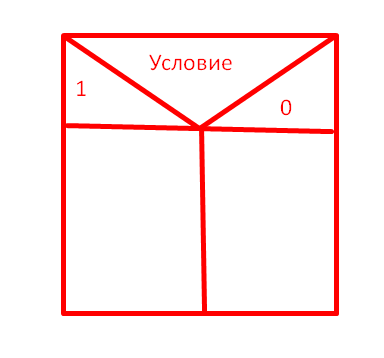
Функциональный блок



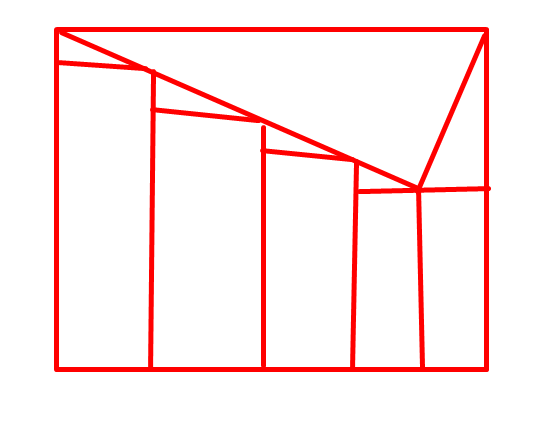
Блок следования



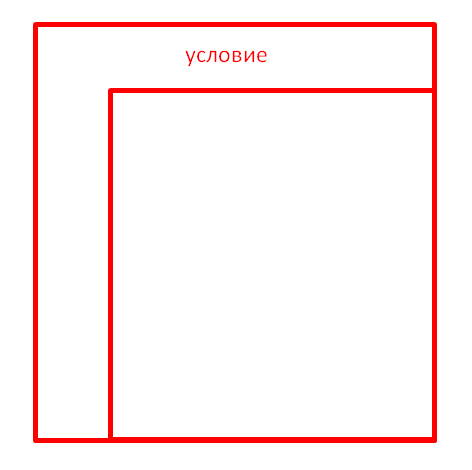
Блок решения



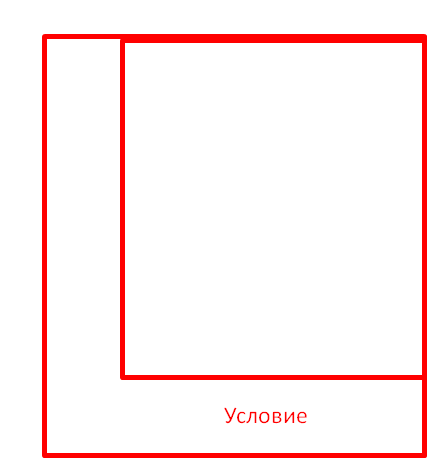
Блок Case



Цикл «Пока»



Цикл «До»



Сложность алгоритмов

Одна и та же задача может быть решена разными способами

* Т. е. может быть разные алгоритмы решения одной и той же задачи

Алгоритмы могут быть оценены:

* По количеству выполняемых операций
* По объему потребляемой памяти
* По использованию других ресурсов

Оценка может быть:

* Минимальный (лучший случай)
* Средней
* Максимальной (худший случай)

Вычисление суммы элементов (N элементов) (плохая реализация..?)

Sum := 0;

For I := 1 to N do

Sum := Sum + A[I]

N операций

* чтения элемента массива
* чтения переменной Sum
* сложения
* записи переменной Sum
* инкремента переменной I
* сравнения I < N

1 операция записи переменной Sum (инициализация)

Сложность = 6\*N + 1 ( O(N) )

Алгоритм нахождения суммы максимальных значений строк матрицы NxM

…

Сложность = O(NxM)

Алгоритм линейного поиска элемента в массиве

Index := -1;

For I := 1 to N do

If A[I] = value then

Begin

Index := I;

Break;

End;

Алгоритм бинарного поиска в отсортированном массиве:

* Выбрать средний элемент
* Если равен искомому значению – поиск завершен
* Если искомое значение меньше, повторить для «левой» половины
* Если искомое значение больше, повторить для «правой» половины

Сложность = O(log2(N))

В спецификации требований прописываются в т.ч. требования к производительности программы.

* Иногда эти требования могут подразумеваться

Зная оценку сложности алгоритма, можно определить, сколько времени ему потребуется для решения задачи заданного объема

Пример:

* Есть 2 алгоритма
  + Со сложностью O(N log10 N)
  + Со сложность O(N^2)
* Предполагаемый объем данных – 1 млрд элементов
* Ограничение по времени – 1 с.

Для 1000 элементов алгоритмы отработали за:

* 0.003 с
* 0.001 с

Сложность алгоритма по объему потребляемой памяти оценивается аналогично

Как правила, алгоритм можно ускорить, увеличив расход памяти и наоборот

На практике не все операции обходятся одинаково дорого.

* Выделение динамической памяти может само иметь сложность O(N)
* Присваивание может занимать много времени для больших типов данных (сложность – O(S))
* Характер обращений к памяти может влиять на использование кэша.
* Ветвления могут замедлять работу программ