Языки программирования высокого уровня

Домашнее задание

- Подготовить конспект на тему «Краткая история языков программирования. Классификация ЯП (Парадигмы программирования)»
- http://www.inf1.info/book/export/html/216
- http://ru.wikipedia.org/wiki/Язык_программирования

Язык программирования

- **Язык программирования** - формализованный язык, предназначенный для описания программ и алгоритмов решения задач на ЭВМ.

Эволюция языков программирования

Поколения	Языки программирования	Характеристика
1	Машинные	Ориентированы на использование в конкретной ЭВМ, сложны в освоении, требуют хорошего знания архитектуры ЭВМ
2	Ассемблеры, Макроассемблеры	Более удобны для использования, но по-прежнему машинно-зависимы
3	Языки высокого уровня	Мобильные, человеко- ориентированные, проще в освоении
4	Непроцедурные, объектно-ориентированные, языки запросов, параллельные	Ориентированы на непрофессионального пользователя и на ЭВМ с параллельной архитектурой
5	Языки искусственного интеллекта, экспертных систем и баз знаний, естественные языки	Ориентированы на повышение интел лектуального уровня ЭВМ и интерфейса с языками

Классификация ЯП

Фактор	Характеристика	Группы	Примеры ЯП
Уровень ЯП	Степень близости ЯП к архитектуре компьютера	Низкий	Автокод, ассемблер
		Высокий	Fortran, Pascal, ADA, Basic, C и др.
Специализация ЯП	Потенциальная или реальная область применения	Общего назначения (универсальные)	Algol, PL/1, Simula, Basic, Pascal
		Специализированные	Fortran (инженерные рассчеты), Cobol (коммерческие задачи), Refal, Lisp (символьная обработка), Modula, Ada (программирование в реальном времени)
Алгоритмичность (процедурность)	Возможность абстрагироваться от деталей алгоритма решения задачи. Алгоритмичность тем выше, чем точнее приходится планировать порядок выполняемых действий	Процедурные	Ассемблер, Fortran, Basic, Pascal, Ada
		Непроцедурные	Prolog, Langin

Классификация ЯП

Система программирования

Система программирования - программная система, предназначенная для разработки программ на конкретном языке программирования.

Система программирования предоставляет пользователю специальные средства разработки программ:

- транслятор,
- (специальный) редактор текстов программ,
- библиотеки стандартных подпрограмм,
- программную документацию,
- отладчик и др,

обеспечивающих автоматизацию составления и отладки программ пользователя.

Транслятор

- **Транслятор** — программа или техническое средство, выполняющее *трансляцию программы*
- **Трансляция программы** — преобразование программы, представленной на одном из языков программирования, в программу на другом языке и, в определённом смысле, равносильную первой. Транслятор обычно выполняет также диагностику ошибок, формирует словари идентификаторов, выдаёт для печати тексты программы и т. д.
- Язык, на котором представлена входная программа, называется *исходным языком*, а сама программа — *исходным кодом*.
- Выходной язык называется *целевым языком* или *объектным кодом*.

Транслятор

- В настоящее время трансляторы разделяются на три основные группы: ассемблеры, компиляторы и интерпретаторы.
- **Ассемблер** — системная обслуживающая программа, которая преобразует символические конструкции в команды машинного языка.

Специфической чертой ассемблеров является то, что они осуществляют дословную трансляцию одной символической команды в одну машинную.

Компилятор

- **Компилятор** — это обслуживающая программа, выполняющая трансляцию на машинный язык программы, записанной на исходном языке программирования.
- Программа на компилируемом языке при помощи компилятора преобразуется (компилируется) в машинный код (набор инструкций) для данного типа процессора и далее собирается в исполнимый модуль, который может быть запущен на исполнение как отдельная программа.

Этапы компиляции

Процесс компиляции, как правило, состоит из нескольких этапов:

- лексический анализ;
- синтаксический анализ;
- семантический анализ;
- создание на основе результатов анализов промежуточного кода;
- оптимизация промежуточного кода;
- создание объектного кода, в данном случае машинного.

Компилятор

Достоинства:

- программа компилируется один раз и при каждом выполнении не требуется дополнительных преобразований,
- не требуется наличие компилятора на целевой машине, для которой компилируется программа.

Недостатки:

- отдельный этап компиляции замедляет написание и отладку и затрудняет исполнение небольших, несложных или разовых программ.
- при внесении изменений в исходный код, требуется повторная компиляция

Большинство современных языков предназначены для разработки сложных пакетов программ и рассчитаны на компиляцию.

Интерпретатор

- **Интерпретатор** – это программа, предназначенная для построчных трансляции и выполнения исходной программы. Такой процесс называется интерпретацией.

Достоинства:

- Интерпретатор сообщает о найденных им ошибках после трансляции каждой строки программы. Это облегчает процесс поиска и исправления ошибок в программе.

Недостатки:

- увеличивается время трансляции,
- необходимость наличия интерпретатора на устройстве, на котором планируется интерпретация программы

Иногда один и тот же язык может использовать и компилятор, и интерпретатор.

Этапы работы интерпретатора

Процесс работы интерпретатора, как правило, состоит из нескольких этапов:

- лексический анализ;
- синтаксический анализ;
- семантический анализ;
- создание промежуточного представления кода (при чистой интерпретации не выполняется);
- исполнение.

Классификация СП

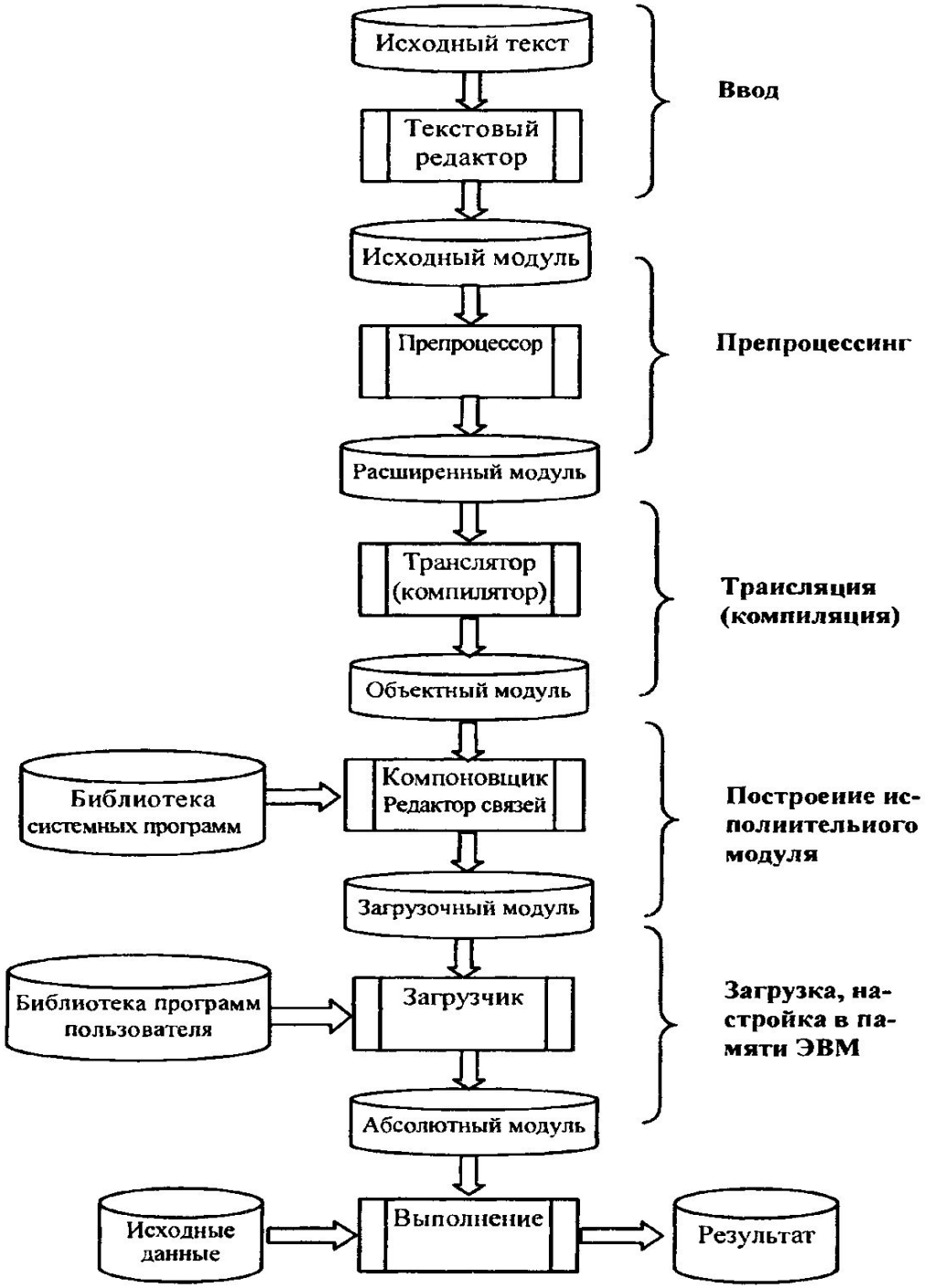
Признак классификации	Типы
Набор исходных языков	Одноязыковые
	Многоязыковые
Возможности расширения	Замкнутые
	Открытые
Трансляция	Компиляция
	Интерпретация

Классификация СП

- Во *многоязыковых* системах отдельные части (секции, модули или сегменты) программы могут быть подготовлены на различных языках и объединены во время или перед выполнением в единый модуль;
- в *открытую* систему можно ввести новый входной язык с транслятором, не требуя изменений в системе;
- в *интерпретирующей* системе осуществляется покомандная расшифровка и выполнение инструкций входного языка (в среде данной системы программирования); в *компилирующей* — подготовка результирующего модуля, который может выполняться на ЭВМ практически независимо от среды.

Структура СП

структура абстрактной многоязыковой, открытой, компилирующей системы программирования и процесс разработки приложений в данной среде



Процесс разработки приложений в СП

- *Ввод.* Программа на исходном языке (исходный модуль) готовится с помощью текстовых редакторов и в виде текстового файла или раздела библиотеки поступает на вход препроцессора .
- *Препроцессинг* — необязательная фаза, состоящая в анализе исходного текста, извлечения из него директив препроцессора и их выполнения.

  Директивы препроцессора представляют собой помеченные спецсимволами (обычно %, #, &) строки, содержащие аббревиатуры или другие символические обозначения конструкций, включаемых в состав исходной программы перед ее обработкой компилятором.

  Например, препроцессор включает содержимое одних файлов в другие, заменяет в тексте исходного кода имена констант на их значения, удаляет символы конца строки

Процесс разработки приложений в СП

- *Трансляция (компиляция)* — в общем случае многоступенчатый процесс, включающий следующие фазы:
  - синтаксический анализ — проверка правильности конструкций, использованных программистом при подготовке текста;
  - семантический анализ — выявление несоответствий типов и структур переменных, функций и процедур;
  - генерация объектного кода — завершающая фаза трансляции.
- *Объектный модуль* (итог этапа компиляции) представляет собой текст программы на машинном языке, включающий машинные инструкции, словари, служебную информацию.

  Объектный модуль не работоспособен, поскольку содержит неразрешенные ссылки на вызываемые подпрограммы библиотеки транслятора, реализующие функции ввода-вывода, обработки числовых и строчных переменных, а также на другие программы пользователей или средства пакетов прикладных программ.

Процесс разработки приложений в СП

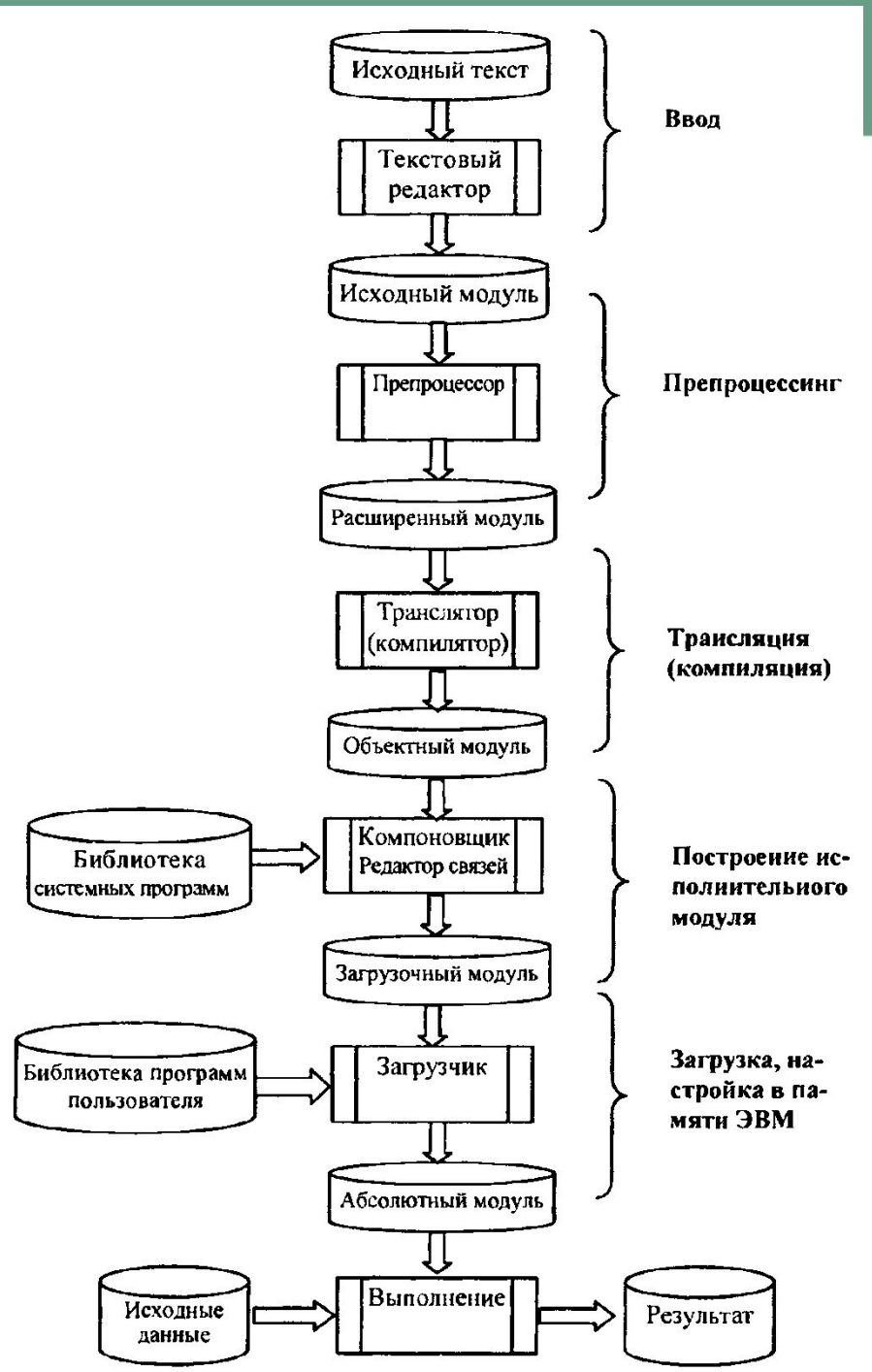
- *Построение исполнительного модуля.* Построение исполнительного (загрузочного) модуля осуществляется специальными программными средствами, основной функцией которых является объединение объектных и загрузочных мо дулей в единый загрузочный модуль с последующей записью в библиотеку или файл. Полученный модуль в дальнейшем может использоваться для сборки других программ и т.д., что создает возможность наращивания программного обеспечения.

- Загрузочный модуль – программный модуль, представленный в форме, пригодной для загрузки в основную память для выполнения

Процесс разработки пр

Загрузка программы. Загрузочный
мещается в качестве раздела в пользов
либо в качестве последовательного фа
диске (НМД). Выполнение модуля сост
память, настройке по месту в памяти и
загрузочного модуля в памяти назыв
скольку все команды ЭВМ здесь прио
получают абсолютные адреса в памя

После загрузки программы задаются исходные данные и получают результат.



ОБЩИЕ ПРИНЦИПЫ РАЗРАБОТКИ ПО

Частотный принцип

- Основан на выделении в алгоритмах и в об рабатываемых структурах действий и данных *по частоте использования*.
- Для действий, которые часто встречаются при работе ПО, обеспечиваются условия их быстрого выполнения. К данным, которым происходит частое обращение, обеспечивается наиболее быстрый доступ, а подобные операции стараются сделать более короткими.

Принцип модульности.

- Под *модулем* в общем случае понимают функциональный элемент рассматриваемой системы, имеющий оформление, законченное и выполненное в пределах требований системы, и средства сопряжения с подобными элементами или элементами более высокого уровня данной или другой системы.

Принцип функциональной избирательности

- В ПО выделяется некоторая часть важных модулей, которые постоянно должны быть в состоянии готовности для эффективной организации вычислительного процесса. Эту часть в ПО называют *ядром* или *монитором*.
- Программы, входящие в состав монитора, постоянно находятся в оперативной памяти. Остальные части ПО постоянно хранятся во внешних запоминающих устройствах и загружаются в оперативную память только при необходимости, иногда перекрывая друг друга.

Принцип генерируемости

- Данный принцип определяет такой способ исходного представления ПО, который бы позволял осуществлять *настройку* на конкретную конфигурацию технических средств, круг решаемых проблем, условия работы пользователя.

Принцип функциональной избыточности

Этот принцип учитывает возможность проведения одной и той же работы (функции) *различными средствами*. Особенно важен учет этого принципа при разработке пользовательского интерфейса для выдачи данных из-за психологических различий в восприятии информации.

Принцип «по умолчанию»

- Принцип основан на хранении в системе некоторых базовых описаний структур, модулей, конфигураций оборудования и данных, определяющих условия работы с ПО. Эту информацию ПО использует в качестве заданной, если пользователь забудет или сознательно не конкретизирует ее. В данном случае ПО само установит соответствующие значения.
- Применяется для облегчения организации связей с системой как на стадии генерации, так и при работе с уже готовым ПО.

ЖИЗНЕННЫЙ ЦИКЛ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Жизненный цикл ПО

Жизненный цикл ПО - процесс его создания и применения от начала до конца.

Этот процесс состоит из нескольких стадий:

1) определение требований и спецификаций,
2) проектирование,
3) программирование,
4) отладка
5) сопровождение.

1. Определение требований и спецификаций

- На ней устанавливаются общие требования к ПО: по надежности, технологичности, правильности, универсальности, эффективности, информационной согласованности; вырабатывается описание системы с точки зрения пользователя.
- Итогом выполнения этого этапа являются *эксплуатационные и функциональные спецификации*, содержащие конкретное описание ПО.

1. Определение требований и спецификаций

- Эксплуатационные спецификации содержат сведения о быстродействии ПО, затратах памяти, требуемых технических средствах, надежности и т.д.
- Функциональные спецификации определяют функции, которые должно выполнять ПО, т.е. в них определяется, *что* надо делать системе, а не то, *как* это делать.
- Спецификации должны быть полными, точными и ясными.

1. Определение требований и спецификаций

Значение спецификаций:

- Спецификации являются заданием на разработку ПО
- Спецификации используются для проверки готовности ПО.
- Спецификации являются неотъемлемой частью программной документации, облегчают сопровождение и модификацию ПО.

2. Проектирование ПО

На этом этапе:

- Формируется структура ПО и разрабатываются алгоритмы, задаваемые спецификациями.
- Устанавливается состав модулей с разделением их на иерархические уровни на основе изучения схем алгоритмов.
- Выбирается структура информационных массивов.
- Фиксируются межмодульные интерфейсы.

Цель этапа — иерархическое разбиение сложных задач создания ПО на подзадачи меньшей сложности. *Результатом* работы на этом этапе являются спецификации на отдельные модули, дальнейшая декомпозиция которых нецелесообразна.

3. Программирование

- На данном этапе производится программирование модулей.
- Этап менее сложен по сравнению со всеми остальными. Проектные решения, полученные на предыдущей стадии, реализуются в виде программ.

4. Отладка ПО

- Этап заключается в *проверке* выполнения всех требований, всех структурных элементов системы на таком количестве всевозможных комбинаций данных, какое только позволяют здравый смысл и бюджет.
- Этап предполагает *выявление* и исправление в программах *ошибок*, проверку работоспособности ПО, а также его *соответствие спецификациям*.

- Отладка = Тестирование + Поиск ошибок + Редактирование

5. Сопровождение

- На данном этапе происходит процесс исправления ошибок, координация всех элементов системы в соответствии с требова ниями пользователя, внесение всех необходимых ему исправлений и изменений.
- Он вызван, как минимум, двумя причинами: во-первых, в ПО остаются ошибки, не выявленные при отладке; во-вторых, пользователи в ходе эксплуатации ПО настаивают на внесении в него изменений и его дальнейшем совершенствовании.

Распределение временных и стоимостных затрат по стадиям жизненного цикла ПО

(сведения из разных источников)

Стадии жизненного цикла программного обеспечения		Сред-нее
Определение требо-ваний и спецификаций		8,6
Проектирование		10,2
Программирование		8,8
Отладка		16,2
Сопровождение		56,2
Всего		100

Основные параметры стадий жизненного цикла ПО

Стадии жизненного цикла программного обеспечения	Количество ошибок,%	Затраты на устранение ошибок, %
Определение требований и спецификаций		9
Проектирование	61–64	6
Программирование	36–39	10
Отладка		12
Сопровождение		63

Зависимость вероятности правильного исправления ошибок и стоимости исправления ошибок от этапа разработки

Проверочная

Вариант 1

1. Эволюция ЯП.
2. Опишите понятие транслятора, виды трансляторов, достоинства и недостатки каждого вида.
3. Охарактеризуйте стадии жизненного цикла ПО
4. Опишите среднее распределение временных и стоимостных затрат по стадиям жизненного цикла ПО
5. На каких стадиях жизненного цикла затраты на устранение ошибок максимальны и почему?

Вариант 2

1. Классификация ЯП.
2. Дайте определение СП
3. Опишите структуру абстрактной многоязыковой, открытой, компилирующей системы программирования и процесс разработки. приложений в данной среде
4. Охарактеризуйте общие принципы разработки ПО.
5. Опишите основные параметры стадий жизненного цикла ПО. Какие стадии ЖЦ требуют наибольших затрат и почему?

Проверочная

Вариант 1

1. Эволюция ЯП.
2. Опишите понятие транслятора, виды трансляторов, достоинства и недостатки каждого вида.
3. Охарактеризуйте стадии жизненного цикла ПО

Вариант 2

1. Классификация ЯП.
2. Опишите структуру исходной программы на ЯП
3. Объясните понятия алфавита, синтаксиса и семантики ЯП

Вариант 3

1. Дайте определение СП
2. Опишите основные параметры стадий жизненного цикла ПО. Какие стадии ЖЦ требуют наибольших временных и стоимостных затрат и почему?
3. Опишите структуру ЯПВУ

Проверочная

Вариант 1

1. Эволюция ЯП.
2. Опишите понятие транслятора, виды трансляторов, достоинства и недостатки каждого вида.
3. Охарактеризуйте общие принципы разработки ПО.

Вариант 2

1. Классификация ЯП.
2. Дайте определение СП
3. Опишите основные стадии жизненного цикла ПО. Какие стадии ЖЦ требуют наибольших временных и стоимостных затрат и почему?

СТРУКТУРА И СПОСОБЫ ОПИСАНИЯ ЯПВУ

Структура ЯПВУ

Элементы ЯП

Любой язык программирования образуют три основные составляющие:

алфавит, синтаксис и семантика.

- **Алфавит** – фиксированный для данного языка набор основных символов, допускаемых для составления текста программы на этом языке.
- **Синтаксис** – система правил, определяющих допустимые конструкции языка программирования из букв алфавита.
- **Семантика** – система правил однозначного толкования отдельных языковых конструкций, позволяющих воспроизвести процесс обработки данных.

Синтаксис

Для описания синтаксиса языка программирования тоже нужен какой-то язык, предназначенном для описания других языков.

Наиболее распространенными метаязыками в литературе по программированию являются металингвистические формулы Бекуса – Наура (язык БНФ) и синтаксические диаграммы.

Язык БНФ

В БНФ всякое синтаксическое понятие описывается в виде формулы, состоящей из правой и левой части, соединенных знаком ::=, смысл которого эквивалентен словам «по определению есть».

Слева от знака ::= записывается имя определяемого понятия (метапеременная), которое заключается в угловые скобки < >, а в правой части записывается формула или диаграмма, определяющая все множество значений, которые может принимать метапеременная.

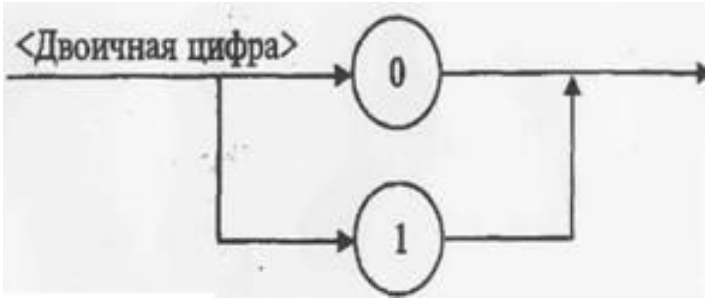
Пример. Формула БНФ, определяющая понятие «двоичная цифра», выглядит следующим образом:
<двоичная цифра>::=0|1

Значок | эквивалентен слову «или».

Синтаксические диаграммы

В диаграммах стрелки указывают на последовательность расположения элементов синтак сической конструкции; кружками обводятся символы, присутствующие в конструкции.

Пример: определение понятие «двоичная цифра» на языке синтаксических диаграмм:

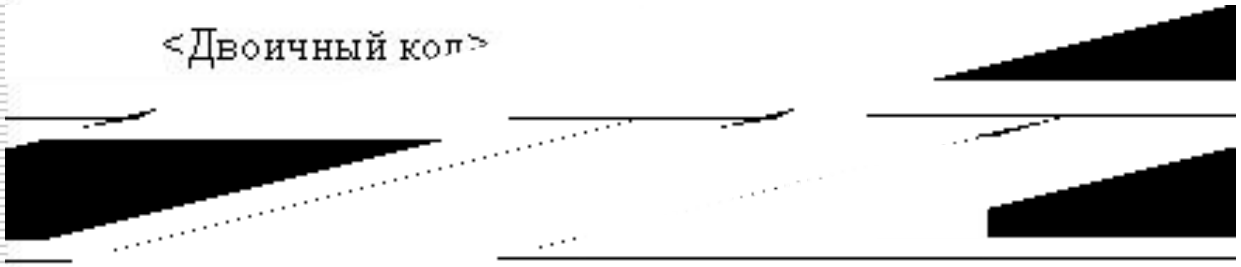


Примеры

Понятие «двоичный код» как непустую последовательность двоичных цифр БНФ описывает так:

<двоичный код>::=<двоичная цифра>|<двоичный код><двоичная цифра>

Синтаксическая диаграмма двоичного кода:



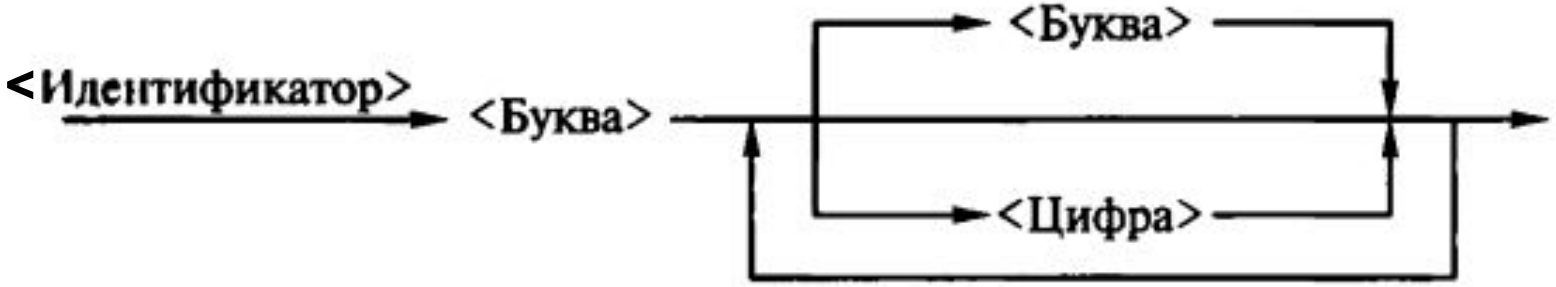
Возвратная стрелка обозначает возможность многократного повторения.

Определение, в котором некоторое понятие определяется само через себя, называется рекурсивным.

Примеры

<Знак операции> ::= +|-|/|*|^|=|>|>=|<|<=

<Идентификатор> ::= <Буква>|<Идентификатор><Буква>|
<Идентификатор><Цифра>



Структура исходной программы на ЯП

Исходная программа (source module), как правило, состоит из следующих частей:

- раздел идентификации – область, содержащая наименование программы, а также дополнительную информацию для программистов и/или пользователей;
- раздел связи – фрагмент текста, описывающий внешние переменные, передаваемые вызывающей программой (если таковая имеется), т.е. ту часть исходных данных, которая обязательно поступает на вход программы при ее запуске;
- раздел оборудования (среда) – описание типа ЭВМ, процессора, требований к оперативной и внешней памяти, существенных с точки зрения выполнимости программы;
- раздел данных – идентификация (декларация, объявление, описание) переменных, используемых в программе, и их типов.
- раздел процедур – собственно программная часть, содержащая описание процессов обработки данных. Элементами процедуры являются операторы и стандартные функции, входящие в состав соответствующего языка программирования.

Типы данных

Тип данных (тип) — множество значений и операций на этих значениях

Тип данных однозначно определяет:

- внутреннее представление данных
- диапазон их возможных значений;
- допустимые действия над данными (операции и функции).

ПАСКАЛЬ. ОСНОВНЫЕ ПОНЯТИЯ.

Структура программы

Схематически программа представляется в виде последовательности восьми разделов:

1) заголовок программы;
2) описание внешних модулей, процедур и функций;
3) описание меток;
4) описание констант;
5) описание типов переменных;
6) описание переменных;
7) описание функций и процедур;
8) раздел операторов.

Структура программы

Program <имя программы>;

Uses <раздел модулей>;

Label <раздел меток>;

Const <раздел констант>;

Type <раздел типов>;

Var <раздел переменных>;

Procedure (Function) <раздел подпрограмм>;

Begin

<раздел операторов>

End.

; ставится в конце заголовка программы, в конце каждого раздела, после каждого оператора. Не ставится перед else

Пример программы

```
program circle;
const
      pi=3.14159;
var
      r,s,l : real;
begin
      writeln (‘введите радиус');
      readln(r);
      s:=pi*r*r;
      l:=2*pi*r;
      writeln('площадь круга=',S:8
      writeln('длина окружности=
end.
```

```
1  program modul_search;
2  uses Crt, Search;
3  var
4  mas: array[1..5] of integer;
5  n, j: integer; str: string;
6  y: char;
7  begin
8  clrscr;
9  writeln('Enter the array elements');
10 for j:=1 to 5 do
11 readln(mas[j]);
12 write('Enter number search: '); readln(n);
13 write('This array is ordered? (y/n) '); readln(y);
14 if y='y' then binary_search(n, mas, str)
15 else line_search(n, mas, str);
16 write(str);
17 readkey;
18 end.
```

```
Program Sort_Array;
Var W_Array : Array [1..100] of Integer;
      I, J, min, I_min, N : Integer;
Begin
   Write('Введите количество элементов массива
         (N<=100): ');
   Readln(N);

   For I := 1 to N do  // Ввод элементов массива
         Write('Введите элемент массива №', I , ':');
         Readln(W_Array[I])
End;
For I := 1 to N do      // Внешний цикл прохода
                        // по элементам
   Begin I_min := I;    // Начальное задание индекса
                        // минимального элемента
         For J := I + 1 to N do
                        // Внутренний цикл поиска
                        // минимального элемента
                        // в пределах от I + 1 до N
         If W_Array[J] < W_Array[I_min] then I_min:= J
         min := W_Array[I_ min];
                        // Сохранение минимального
                        // значения
                        // Перестановка элементов: I-го
                        // и минимального
         W_Array[I_min]:= W_Array[I];
         W_Array[I]:= min;
   End;

      For I := 1 to N do
                              // Вывод отсортированного
                              // массива
            Writeln('Элемент массива №', I, '=',
                     W_Array[I]);
```

Алфавит

Алфавит Паскаля включает:

- прописные и строчные латинские буквы, знак подчеркивания _;
- цифры от 0 до 9;
- специальные символы, например, +, *, { и @;
- пробельные символы: пробел, табуляцию и переход на новую строку.

Алфавит

<Символ> ::= <Буква>|<Цифра>|<Ограничитель>

<Буква> ::= A|B|C|D|E|F|G|H|I|J|G|K|L|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|_|
a|b|c|d|e|f|g|h|i|j|g|k|l|n|o|p|q|r|s|t|u|v|w|x|y|z|

<Цифра> ::= 0|1|2|3|4|5|6|7|8|9

<Ограничитель> ::= <Служебный знак>|<Служебное слово>

<Служебный знак> ::= <Знак пунктуации>|<Знак операции>|
<Разделитель>

<Знак пунктуации> ::= {|}|(*|*)|(|)|[|]|;|:|'|#|@|$|.|,|..|=|:=

<Знак операции> ::= +|-|/|*|^|=|>|>=|<|<=

<Разделитель> ::= <Пробел>|<Управляющий символ (коды от 0 до 31)>

Лексемы

Символы из алфавита языка используются для построения базовых элементов - *лексем*.

Лексема - минимальная единица языка, имеющая самостоятельный смысл.

Основные классы лексем:

- константы;
- имена (идентификаторы);
- ключевые слова;
- знаки операций;
- Разделители (скобки, точка, запятая, пробельные символы).

Примеры: число 128, имя Vasia, ключевое слово goto и знак операции сложения +.

Константы

- **Константа -** величина, не изменяющая свое значение в процессе работы программы.

Классификация констант:

Константы					
Целые		**Вещественные**		**Символьные**	**Строковые**
Десятичные	16-ричные	С плавающей точкой	С порядком		
2 5	$0101 $FFAA4	-0.26 .005	1.2E4 0.1E-5	‘k’ #186	‘лололо’ ‘I’’m fine’

Идентификаторы

Идентификатор – последовательность букв алфавита Pascal и цифр, начинающаяся с буквы. Значимыми являются только первые **63** символа.

Имена даются элементам программы, к которым требуется обращаться - переменным, константам, процедурам, функциям, меткам и так далее.

Правила:

- имя должно начинаться с буквы;
- имя должно содержать только буквы, знак подчеркивания и цифры;
- прописные и строчные буквы не различаются.

Примеры:	X	_Beta	программа	Alfa_Beta
	Y2.2M	5ABC	Figure/Fer	CH Gamma

Ключевые слова

Ключевые (зарезервированные) слова - это идентификаторы, которые имеют специальное значение для компилятора. Их можно использовать только в том смысле, в котором они определены.

Например, для оператора перехода определено ключевое слово *goto*, а для описания переменных - *var*.

Имена, создаваемые программистом, не должны совпадать с ключевыми словами.

Ключевые слова

absolute	div	goto	nil	repeat	var
and	do	if	not	set	virtual
array	downto	implementation	object	shl	while
asm	else	in	of	shr	with
assembler	end	inline	ot	string	xor
begin	external	interface	packed	then	
case	file		private	to	
const	for	interrupt	procedure	type	
constructor	forward	label	program	unit	
destructor	function	mod	record	uses	

Знаки операции

Знак операции - это один или более символов, определяющих действие над операндами. Внутри знака операции пробелы не допускаются.

Условное обозначение	Наименование операции
+	сложение
-	вычитание
*****	умножение
/	деление
div	деление целочисленное
mod	остаток от целочисленного деления
:=	присвоение
=	равно (сравнение)
<>	не равно (сравнение)
<	меньше (сравнение)
>	больше (сравнение)
<=	меньше или равно (сравнение)
>=	больше или равно (сравнение)
not	логическое НЕ
and	логическое И
or	логическое ИЛИ
xor	исключающее ИЛИ

Назначение знаков пунктуации

Знакн	Назначение
(* *) { }	Скобки комментария. Текст, заключенный между скобками, поясняет алгоритм и не является его частью
[ ]	Задание индексов массива, размера строки, элементов множества
( )	Выделение части выражения, задание списков параметров
;	Отделение одного предложения программы от другого, разделение параметров (в части объявления)
:	Отделение переменной или константы от типа (в части объявлений), отделение метки от оператора, следующего за ней
,	Разделение элементов списка, параметров процедуры и функции при вызове
@	Обозначение адреса (переменной, константы, процедуры, функции, метода)
$	Признак числа в шестнадцатеричной системе, обозначение директивы компилятора
#	Обозначение символа по его коду
..	Разделение границ диапазона в типе-диапазоне
:=	Знак оператора присваивания
=	Отделение идентификатора типа (константы) от его описания (значения)
'	Апостроф — признак символа или строковой константы.

ТИПЫ ДАННЫХ

Типы данных

Тип данных однозначно определяет:

- внутреннее представление данных
- диапазон их возможных значений;
- допустимые действия над данными (операции и функции).

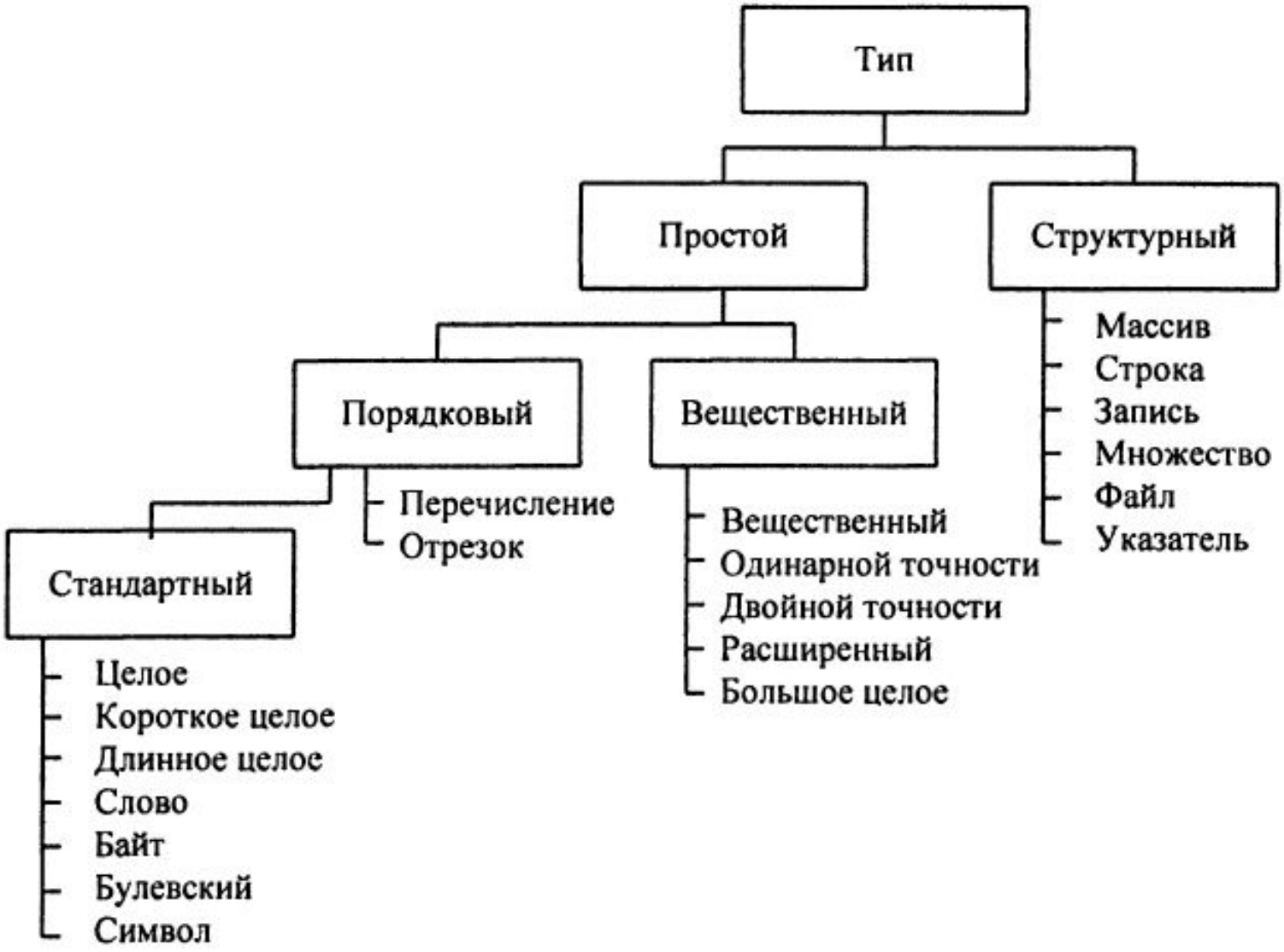
Классификация типов 1

Стандартные	Определяемые программистом	
	Простые	**Структурированные**
логические целые вещественные символьный строковый адресный файловые	перечисляемый интервальный адресные	массивы строки записи множества
		файлы процедурные типы объекты

Стандартные типы не требуют предварительного определения. Для каждого типа существует ключевое слово, которое используется при описании переменных, констант и т.д.

Если же программист определяет собственный тип данных, он описывает его характеристики и сам дает ему имя, которое затем применяется точно так же, как имена стандартных типов.

Классификация типов 2



Классификация типов данных (Паскаль)

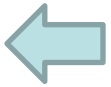
- Типы данных
  - Простые
    - Порядковые
      - Целые
      - Логические
      - Символьные
      - Перечисляемые
      - Интервальные
    - Вещественные
  - Структурированные
    - Массивы
    - Строки
    - Множества
    - Записи
    - Файлы

ПРОСТЫЕ ТИПЫ ДАННЫХ

Логический тип - **boolean**

Тип	Длина (байт)	Диапазон значений	Операции
boolean	1	true, false	Not, And, Or, Xor, >=, <=, =, <>, <, >

Результат операций имеет логический тип.



Целые типы

Внутреннее представление

Тип	Название	Размер	Знак	Диапазон
integer	целое	2 байта	есть	-32768..32767($-2^{15}..2^{15}-1$)
shortint	короткое целое	1 байта	есть	-128..127($-2^{7}..2^{7}-1$)
byte	байт	1 байт	нет	0..255($0..2^{8}-1$)
word	слово	2 байта	нет	0..65535($0..2^{16}-1$)
longint	длинное целое	4 байта	есть	-2147483648..2147483647($-2^{31}..2^{31}-1$)

Операции

Операции	Результат
+, -, /, *, Div, Mod	Целый (при делении дробная часть отбрасывается)
>=, <=, =, <>, <, >	**boolean**
Not, And, Or, Xor	целый

```
85 div 10 = 8
85 mod 10 = 5
```

Целые типы

Стандартные функции и процедуры

Имя	Описание	Результат	Пояснения
		Функции	
abs	модуль	целый	|x| записывается abs(x)
arctan	арктангенс угла	вещественный	arctg x записывается arctan(x)
cos	косинус угла	вещественный	cos x записывается cos(x)
exp	экспонента	вещественный	e^x записывается exp(x)
ln	Натур-ый логарифм	вещественный	$\log_e x$ записывается ln(x)
odd	проверка на четность	логический	odd(3) даст в результате true
pred	предыдущее значение	целый	pred(3) даст в результате 2
sin	синус угла	вещественный	sin x записывается sin(x)
sqr	квадрат	целый	x^2 записывается sqr(x)
sqrt	квадратный корень	вещественный	записывается sqrt(x)
succ	следующее значение	целый	succ(3) даст в результате 4
		Процедуры	
inc	инкремент		inc(x) - увеличить x на 1 inc(x, 3) - увеличить x на 3
dec	декремент		dec(x) - уменьшить x на 1 dec(x, 3) - уменьшить x на 3

Символьный тип - char

Служит для представления любого символа из набора допустимых символов. Каждый символ строковой величины занимает 1 байт памяти и имеет числовой код в соответствии с таблицей кодов ASCII.

Операции

Операции	Результат	Пояснения
>=, <=, =, <>, <, >	**boolean**	сравниваются коды символов. Меньшим окажется символ, код которого меньше.

'A' < 'B' < 'C' < ... < 'X' < 'Y' < 'Z' '0' < '1' < '2' < ... < '7' < '8' < '9'

Символьный тип - char

Фрагмент таблицы ASCII

32 пробел	48 0	64 @	80 P	96 `	112 p
33 !	49 1	65 A	81 Q	97 a	113 q
34 "	50 2	66 B	82 R	98 b	114 r
35 #	51 3	67 C	83 S	99 c	115 s
36 $	52 4	68 D	84 T	100 d	116 t
37%	53 5	69 E	85 U	101 e	117 u
38 &	54 6	70 F	86 V	102 f	118 v
39 ‘	55 7	71 G	87 W	103 g	119 w
40 (	56 8	72 H	88 X	104 h	120 x
41)	57 9	73 I	89 Y	105 i	121 y
42 *	58 :	74 J	90 Z	106 j	122 z
43 +	59 ;	75 K	91 [	107 k	123 {
44 ,	60 <	76 L	92 \	108 l	124 |
45 -	61 =	77 M	93]	109 m	125 }
46 ,	62 >	78 N	94 ^	110 n	126 ~
47 /	63 ?	79 O	95 _	111 o	127

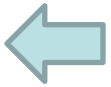
Символьный тип - char

Стандартные функции

Имя	Описание	Результат	Пояснения
ord	порядковый номер символа	целый	ord('b') даст в результате 98 ord('ю') даст в результате 238
chr	преобразование в символ	char	chr(98) даст в результате 'b' chr(238) даст в результате 'ю'
pred	предыдущий символ	char	pred('b') даст в результате 'a'
succ	последующий символ	char	succ('a') даст в результате 'b'

При отсутствии предыдущего или последующего символов значение со- ответствующих функций не определено.

Для литер из интервала 'a'..'z' применима функция UpCase(C), кото- рая переводит эти литеры в верхний регистр 'A'..'Z'. UpCase('z')=‘Z'



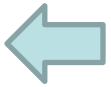
Типы данных, определяемые программистом

Для адекватного представления информации которую требуется обрабатывать в программе, используются типы данных, определяемые самим программистом в разделе описания типов type. При описании типу дается произвольное имя.

```
type имя_типа = описание_типа
...
var имя_переменной: имя_типа
```

Применяется и задание типа непосредственно при описании переменных.

```
var имя_переменной: описание_типа
```



Перечисляемый тип данных

Перечисляемый тип представляет собой ограниченную упорядоченную последовательность констант, составляющих данный тип.

type имя_типа = (список имен констант)

Пример.

```
type Menu = (READ, WRITE, EDIT, QUIT);
var m: Menu;
```

Или

```
var m: (READ, WRITE, EDIT, QUIT);
```

Перечисляемый тип данных

Каждое значение является константой своего типа и может принадлежать только одному из перечисляемых типов, заданных в программе.

Например,

```
type  Traffic_Light= (RED, YELLOW, GREEN);
type  Rainbow = (RED, ORANGE, YELLOW,
          GREEN, LIGHT_BLUE, BLUE, VIOLET);
```

тип Traffic_Light не может быть определен в одной программе с типом Rainbow, так как оба типа содержат одинаковые константы.

Перечисляемый тип данных

Перечисляемый тип относится к порядковым типам данных. К перечисляемым переменным и константам могут быть применены операции отношения и стандартные функции Pred, Succ, Ord (значения нумеруются, начиная с 0, в порядке их перечисления в определении типа).

Пример:

type Traffic_Light= (RED, YELLOW, GREEN);

succ(YELLOW)= GREEN

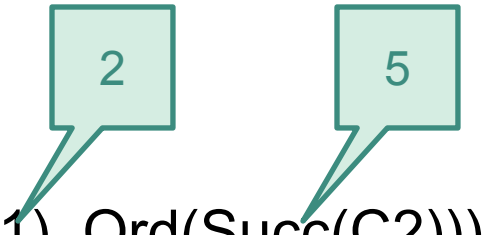
pred(YELLOW)=RED,

ord (GREEN)=2

Перечисляемый тип данных

Пример.

```
Program T1;
Type Colors = (Black, Blue, Green, Cyan, Red, Magenta,
               Brown, Yellow, White);
Var C1,C2 : Colors;
Begin
    C1:=Green;
    C2:=Red;
    Writeln(Ord(C1), Ord(Succ(C2)))
End.
```



Интервальный тип данных

С помощью интервального типа задается диапазон значений какого-либо типа:

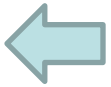
type имя_типа = константа_1 .. константа_2

Пример.

```
type Hour = 0 .. 23;
     Range = -100 .. 100;
     Letters = 'a' .. 'z';
```

Или

```
var r : -100 .. 100;
```



Вещественные типы

Внутреннее представление

Внутреннее представление вещественного числа состоит из двух частей - мантиссы и порядка, и каждая часть имеет знак.

Пример: 0,087 $0{,}87*10^{-1}$

в памяти хранится мантисса 87 и порядок -1

Тип	Название	Размер	Значащих цифр	Диапазон значений
real	вещественный	6 байт	11-12	2.9e-39..1.7e+38
single	одинарной точности	4 байта	7-8	1.5e-45..3.4e+38
double	двойной точности	8 байт	15-16	5.0e-324..1.7e+308
extended	расширенный	10 байт	19-20	3.4e-4932..1.1e+4923

" e + n " означает "умножить на 10 в степени n"

Операции

Операции	Результат
+, -, /, *	вещественный
>=, <=, =, <>, <, >	**boolean**

Вещественные типы

Стандартные функции

Имя	Описание	Результат	Пояснения
abs	модуль	вещественный	|x| записывается abs(x)
arctan	арктангенс угла	вещественный	arctg x записывается arctan(x)
cos	косинус угла	вещественный	cos x записывается cos(x)
exp	экспонента	вещественный	e^x записывается exp(x)
frac	дробная часть аргумента	вещественный	frac(3.1) даст в результате 0.1
int	целая часть аргумента	вещественный	int(3.1) даст в результате 3.0
ln	натуральный логарифм	вещественный	$\log_e x$ записывается ln(x)
pi	значение числа π	вещественный	3.1415926536
round	округление до целого	целый	round(3.1) даст в результате 3 round(3.8) даст в результате 4
trunc	Ближайшее целое, не превышающее x по модулю	целый	
sin	синус угла	вещественный	sin x записывается sin(x)
sqr	квадрат	вещественный	x^2 записывается sqr(x)
sqrt	квадратный корень	вещественный	записывается sqrt(x)

Пример

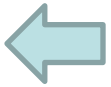
- **Задача**. Для введенного числа определить, является ли оно полным квадратом.

```
program Ya;
var a,b,c:real;
begin
  write('Введите число a: ');
  readln(a);
  if a>=0 then
    begin
       b := sqrt(a);
       c := frac(b);
       if c = 0 then writeln('a является полным квадратом.')
       else writeln('a не является полным квадратом.');
    end
  else writeln('Вы ввели отрицательное число. Невозможно найти корень.');
  readln;
end.
```

Структурные типы данных

Для представления и обработки однотипных данных: таблиц, текстов, множеств и т.д. используют структурные типы данных. В Паскале определены следующие структурные типы данных:

- ***массивы*** - для представления однотипных или табличных данных;
- ***строки*** - для представления символьной (текстовой) информации;
- ***множества*** - для представления абстрактных математических множеств;
- ***записи*** - для представления таблиц с данными различных типов.



Строковый тип - String

Тип String предназначен для хранения строковых величин до 255 символов длиною.

Var <имя_переменной>: string[<максимальная длина строки>]

Например:

Var s1: string[10];

s2: string[20];

smax: string;

можно записать не более 10 символов, и в памяти она займет 11 байт

дается максимальная размерность - 255 символов, что потребует 256 байт памяти для хранения значения.

В памяти строка занимает MAX+1 байт, где MAX - объявленное максимальное количество символов в строке.

Строковый тип - String

Операции

Операции	Название	Результат	Пояснения
+	конкантенация	string	Для сцепления нескольких строк в одну
=, <>, >, <, >=, <=	отношения	**boolean**	посимвольное сравнение двух строк слева направо до первого несовпадающего символа. выражение 'MS-DOS'<'MS-Dos' имеет значение True

Пример:

```
a := 'Turbo';
b := 'Pascal';
c := a + b;
```

c ='TurboPascal’

Строковый тип - String

Правила сравнения строк

- Строки считаются равными, если они совпадают по длине и содержат одни и те же символы на соответствующих местах в строке.
- Большей считается та строка, в которой первый несовпадающий символ имеет больший номер в стандартной таблице обмена информацией.
- Если строки имеют различную длину, но в их общей части символы совпадают, считается, что более короткая строка меньше, чем более длинная.
- Если значение переменной после выполнения оператора присваивания превышает по длине максимально допустимую при описании величину, то все лишние символы справа отбрасываются.

Строковый тип - String

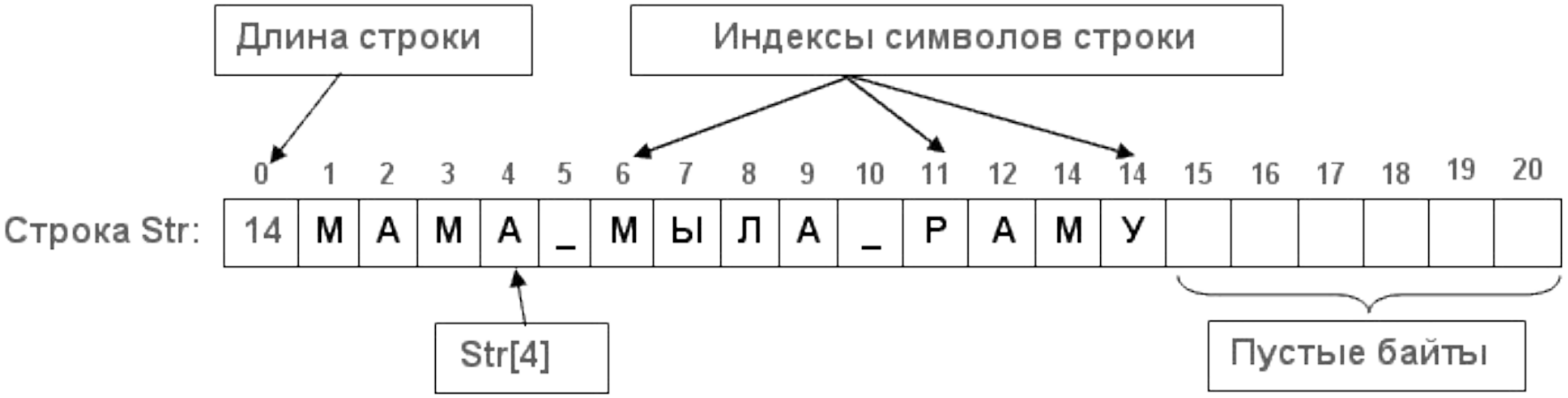
- ‘строка’<>’строки’ (верно, т.к. не совпадают последние символы);
- ‘Abc’<’abc’ (отношение истинно, т.к. код символа ‘A’ равен 65 в десятичной системе счисления, а код символа ‘a’ – 97);
- ‘год’>’век’ (отношение верно, т.к. буква ‘г’ в алфавите стоит после буквы ‘в’, а, следовательно, имеет больший код).

Строковый тип - String

- Каждый символ строки имеет порядковый номер, начиная с первого.
- Имеется возможность обратиться к любому элементу строки, указав его номер

Пример: Str[2] - второй символ в строке Str,

при обращении к символу строки можно поменять значение символа (Str[2]:='r'), вывести на экран это значение или присвоить его другой переменной.



Строковый тип - String

Пример: сформировать строку из 26 символов, содержимым которой является последовательность заглавных букв латинского алфавита.

```
Program stringElements3;
var
   Str : string[26]; {длина строки = 26}
   i : char;
begin
   Str:='';
   for i := 'A' to 'Z' do  Str := Str + i;
   writeln(Str);
end.
```

Строковый тип - String

Стандартные функции

Имя	Описание	Рез-т	Пояснения
Copy(S,p,n)	выделяет из строки S, начиная с позиции p, подстроку из n символов. p,n - целые	String	S:='строка символов' Copy (S,3,3) даст в результате 'рок'
Concat (s1, s2,...,sn)	выполняет слияние строк s1, s2,...,sn в одну строку	String	Concat('язык', ' ', 'Pascal') даст в результате 'язык Pascal'
Length(S)	определяет текущую длину строки S	целый	S:='(a+в)*c' Length(S) даст в результате 7
Pos(subS,S)	определяет позицию первого вхождения подстроки subS в строку S.	целое	S:='строка символов' Pos('рок', S) даст в результате 3 Pos('e', S) даст в результате 0

Строковый тип - String

Стандартные процедуры

Имя	Описание	Результат	Пояснения
Delete (S,p,n)	Удаляет из строки S, начиная с позиции p, подстроку из n символов. p,n - целые	String	S:='строка символов' Delete (S,3,3) даст в результате 'ста символов'
Insert (subS, S, p)	вставляет в строку S, начиная с позиции p, подстроку subS.	String	S:='рис. 2' Insert('№', S, 6) даст в результате 'рис. №2'
Str(x, S)	преобразует число x в строковый формат.	String	Str (3456, S) даст в результате '3456' Str (sin(1):6:4, S) даст в результате '0.0175'
Val(S, x, k)	преобразует строку символов S в число x. k –переменная типа integer, которая равна номеру позиции в строке S, начиная с которой произошла ошибка	целое	x: real Val('12.34', x, k)→ x=12.34, k=0 x:integer Val('12.34', x, k)→ x=12, k=3

Примеры работы со строками

```
Program DemoFunctionLength;
 var  word : string;
 begin
    write('введите слово :');
    readln(word);
    writeln('это слово состоит из ',length(word),' букв');
 end.
```

Определяет длину строки, введенной с клавиатуры.

Примеры работы со строками

```
Program DemoFunctionUpcase;
var
   word: string;
   i: byte;
begin
   word:='фирма Microsoft';
   for i:= 1 to length(word) do
      word[i]:=upcase(word[i]);
   writeln(word);     {выводится текст 'фирма MICROSOFT'}
end.
```

Преобразовывает символ любой литеры из строчного в прописной. Т. к. эта функция рассчитана на обработку отдельного символа, то для обработки строки символов с помощью этой функции приходится организовывать цикл. Русские литеры не могут обрабатываться этой функцией.

Примеры работы со строками

Русские литеры не могут обрабатываться функцией upcase.

Для того, чтобы преобразовать в заглавные строчные буквы русского алфавита, применяют оператор выбора Case: . . . .

```
  . . .
 case Word[i] of
   'а' : Word[i]:= 'А';
   'б' : Word[i]:= 'Б';
   'в' : Word[i]:= 'В';
   . . .
end;
. . .
```

Примеры работы со строками

Копирует фрагмент строки из переменной word в переменную word1

```
Program DemoFunctionCopy;
var
  word : string;
  word1 : string[20];
begin
  word := 'фирма Microsoft';
  writeln(word);      {выводится текст 'фирма Microsoft'}
  word1 := copy(word,1,5);
  writeln(word1);    {выводится текст 'фирма'}
end.
```

Если начальная или конечная позиции копируемого текста находятся вне пределов исходной строки символов, то сообщение об ошибке не выдается. Результатом выполнения операции в первом случае будет строка нулевой длины, во втором - фрагмент от начальной позиции копирования до конца исходной строки.

Примеры работы со строками

```
 Program DemoFunctionPos;
var
  word : string;
  word1 : string[20];
  p: byte;
begin
 writeln ('введите исходный текст ');
 readln (word);
 writeln ('введите искомый текст ');
 readln (word1);
 p := pos(word1, word);
 if p<> 0    then
    write ('фрагмент ‘,word1,'содержится в строке ',word, 'с позиции ',p);
 else
    writeln('фрагмент ‘,word1,’ не содержится в строке ',word);
end.
```

Примеры работы со строками

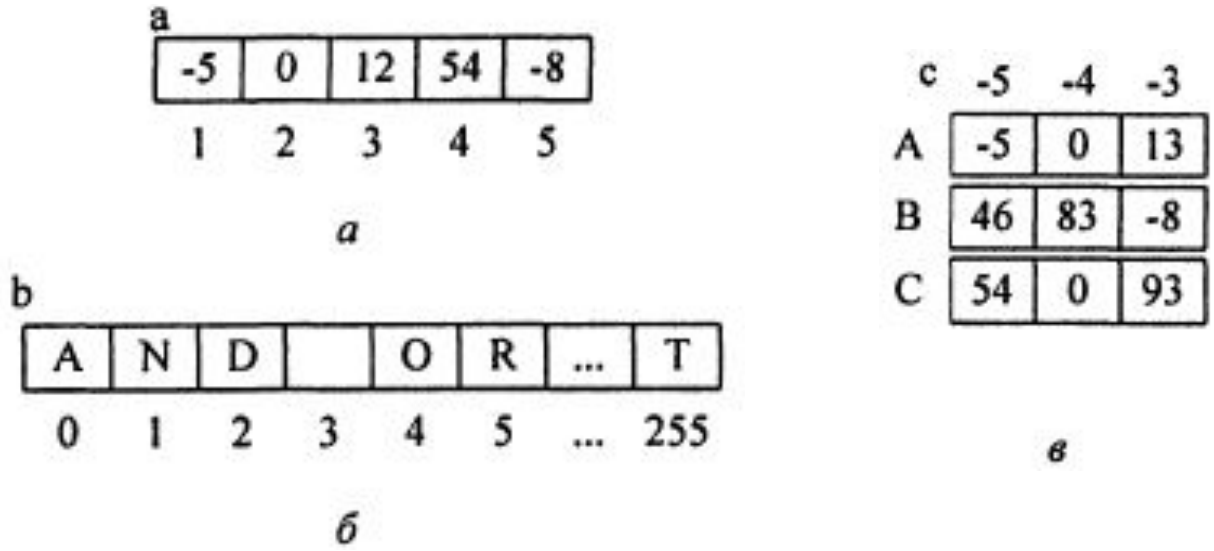
Преобразует значение строки Stroka в величину целочисленного типа и помещает результат в Chislo. Значение строковой переменной Stroka не должно содержать пробелов в начале и в конце.

```
 Program DemoProcedureVal;
var
  word: string;
  chislo, code: integer;
begin
  writeln(‘Введите строку цифр ');
  readln(word);
  val(word, chislo, code);                {преобразование строки в число}
  if code <> 0
    then
      writeln(‘Ошибка! В позиции ',code,' не цифра!');
end.
```

Описание типа «массив»

- **Массив** – это структурированный тип данных, состоящий из фиксированного числа элементов, имеющих один и тот же тип.
- Тип элементов массива называется ***базовым***. Это любой допустимый в Borland Pascal тип (в том числе и массив), кроме файла.

Пример.



а

б

в

а) Одномерный массив из 5 целых чисел; *б*) одномерный массив из 256 символов ; ***в***) двумерный массив (матрица) из 9 чисел

Индекс

Каждому элементу массива соответствует один или несколько индексов, определяющих положение элемента в массиве.

Тип индекса определяет его допустимые значения. В качестве типа индекса может быть указан любой порядковый тип (boolean, char, integer, перечисляемый тип, а также диапазоны этих типов), кроме типа longint и его производных.

В зависимости от количества типов индексов различают: одномерные, двумерные и n-мерные массивы. Двумерные массивы обычно называют *матрицами, считая первый индекс - номером строки, а второй* - номером столбца.

Общий вид описания массива

Объявление переменных типа массив выполняется двумя способами:

- в операторе объявления переменных,

var <имя массива>**: array** [<тип индекса>] **of** <тип элементов>;

- с предварительным объявлением типа

type <имя массива>**= array** [<тип индекса>] **of** <тип элементов>;

Примеры описания массивов

- с предварительным объявлением типа,

```
Type mas =array[1..10] of integer;  {объявляем тип}
Var a:mas;                          {объявляем переменную}
```

- в операторе объявления переменных, например:

```
Var a:array[1..5] of integer; {массив из 5 целых чисел}
    b: array[byte] of char; {массив из 256 символов, индекс элемента массива изменяется от 0 до 255}
    c:array['A'..'C',-5..-3] of byte; {матрица из 9 чисел}
    d:array['A'.. 'C'] of array[-5..-3] of byte; {матрица из 9 чисел, по структуре эквивалентная предыдущей}
```

Инициализация массивов

Можно присвоить значения элементам массива до начала выполнения программы. Это делается в разделе описания констант:

```
type intmas = array [1 .. 6] of integer;
const a : intmas = (0, 5, -7, 100, 15, 1);
```

Количество констант должно точно соответствовать числу элементов массива.

Примеры инициализированных массивов

Примеры:

```
Const a: array[1..5] of real = (0,-3.6,7.8,3.789,5.0);

Const b: array[boolean,1..5] of real = ((0,-3.6,7.8,3.789,5.0),
                                        (6.1,0,-4.56,8.9,3.0));
```

{массив будет инициализирован следующим образом:

$b_{false,1}$ =0, $b_{false,2}$=-3.6, $b_{false,3}$ = 7.8, ..., b $t_{rue,1}$ = 6.1, и т.д.}

Примеры описания массивов

1) **Var** b: **array** [0..5] **of** real;

r: **array** [1..34] **of** char;

n: **array** [‘A’..’Z’] **of** integer;

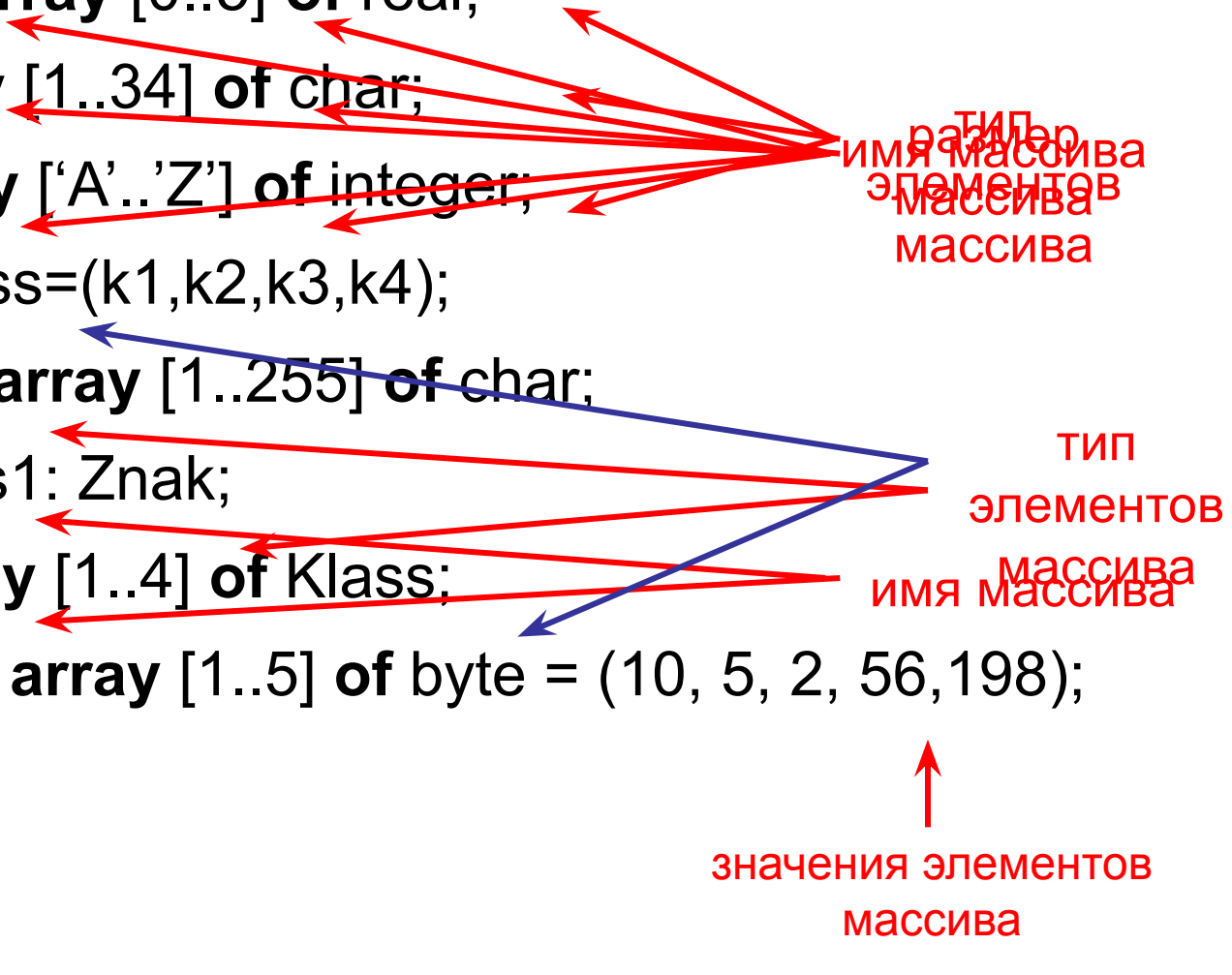
2) **Type** Klass=(k1,k2,k3,k4);

Znak= **array** [1..255] **of** char;

Var Mas1: Znak;

M: **array** [1..4] **of** Klass;

3) **Const** M: **array** [1..5] **of** byte = (10, 5, 2, 56,198);



Обращении к элементам массива

- Чтобы получить доступ к конкретному элементу массива, в качестве индекса можно использовать не только целое число, соответствующее порядковому номеру этого элемента в массиве, но и выражение, значение которого равно упомянутому целому числу.
- Например, при обращении к элементам некоторого массива "A: Array[1..100] of Real", в качестве индекса можно использовать любое арифметическое выражение, значением которого будет целое число из диапазона 1..100:

- A[56]; A[i+7]; A[i div j] .

Операции над массивами

- Над массивом в целом определена единственная операция - операция присваивания.
- **Присваивание** массивов заключается в копировании элементов одного массива в другой. Эту операцию можно выполнять только над массивами одного типа.
- Массивы считаются совпадающими по типу, если они объявлены через запятую в одной строке

Пример 1.

```
Var a, b: array [boolean] of real;
... a:=b;...
```

Операции над массивами

Пример 2.

```
Type mas =array[boolean] of real;
Const a:mas=(3.6, -5.1);
Var b: mas;
... b:=a;...
```

Ввод массива

Пример фрагмента программы ввода одномерного массива вручную (с клавиатуры)

```
var
   a : array [1..10] of integer ;
   i : byte ; {переменная i вводится как индекс массива}
begin
   for i:=1 to 10 do
      readln (a[i]); { ввод i- го элемента производится с клавиатуры }
 …
 end.
```

Ввод массива

Пример фрагмента программы заполнения массива Паскаля случайными числами

```
var
   a: array [1..10] of integer;
   i : byte ; //переменная i вводится как индекс массива
begin
randomize;
 for i :=1 to 10 do
      a [ i ]:= random (10);
// i -му элементу присваивается «случайное» целое число от 0 до 9
…
end.
```

Генератор случайных чисел

оператор:= random (max-min+1)+min;

- оператор - любая переменная
- min - целое число , не превышающее max
- max - целое число, большее min

Здесь min и max являются диапазонами генератора случайных чисел.

Вывод массива

Пример фрагмента программы вывода одномерного массива на экран

```
var
   a : array [1..10] of integer ;
   i : byte ; //переменная i вводится как индекс массива
begin
   for i:=1 to 10 do
    begin
      readln (a[i]); // ввод с клавиатуры
      write (a[i]); // вывод i- го элемента на экран
    end;
 end.
```

Операторы языка Паскаль

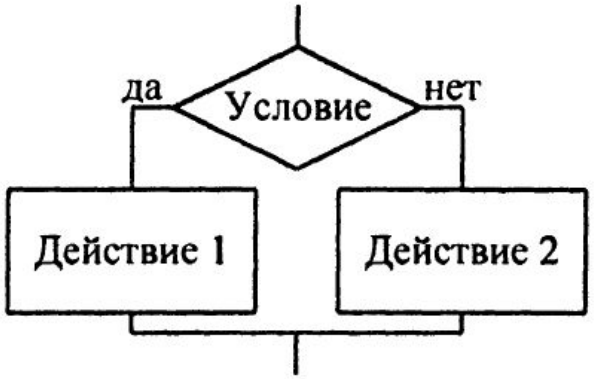
Оператор условия ЕСЛИ (if)

Формы оператора if

Оператор if может принимать одну из форм:

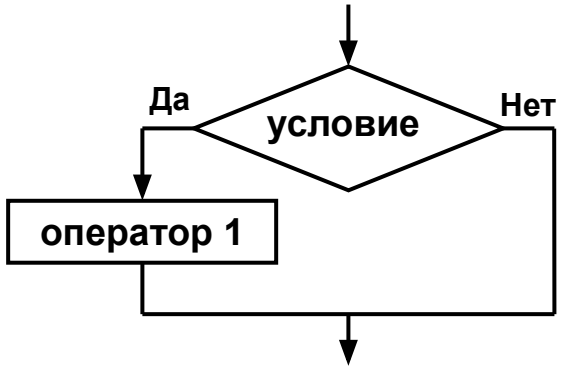
If <условие> then <оператор1>

else <оператор2>;



или

If <условие> then <оператор>;



Выполнение оператора if

If <условие> then <оператор1>

else <оператор2>;

Оператор выполняется следующим образом:

Сначала вычисляется выражение, записанное в условии.

В результате его вычисление получается значение логического (булевского) типа.

Если это значение – «истина», то выполняется оператор1.

Если же в результате имеем «ложь», то выполняется оператор2.

Выполнение оператора if

В случае, если вместо оператора1 или оператора2 следует серия операторов, то эту серию операторов необходимо заключить в операторные скобки begin…end ; .

Обратить внимание, что перед словом else точка с запятой не ставится.

```
If <условие> then
  begin
    <оператор1>;
    <оператор2>;
    <оператор3>
  end
else <операторN>;
```

Пример использования оператора if

Пример :

Даны два числа. Меньшее из этих чисел заменить суммой данных чисел, большее - произведением.

```
Program sh;
Var  x, y,s,p: integer;
Begin
  Write('Введите 2 числа');
  Readln(x, y);
  s:=x+y;
  p:=x*y;
  If x>=y then
    begin
      y:=s;
      x:=p;
    end
  else
    begin
      x:=s;
      y:=p;
    end;
```

Оператор условия ВЫБОР (case)

Форма оператора case

Оператор Case организует переход на один из нескольких вариантов действий в зависимости от значения выражения, называемого селектором.

Общий вид:

```
Case k of
.. : <оператор1>;
.. : <оператор2>;
      …..
.. : <операторN>
else <операторN+1>
end;
```

Здесь k – выражение-селектор, которое может иметь только простой порядковый тип (целый, символьный, логический). ,

… - константы того же типа, что и селектор.

Выполнение оператора case

При использовании оператора Case должны выполняться следующие правила:

1. Выражение-селектор может иметь только простой порядковый тип (целый, символьный, логический).
2. Все константы, которые предшествуют операторам альтернатив, должны иметь тот же тип, что и селектор.
3. Все константы в альтернативах должны быть уникальны в пределах оператора выбора.

Пример использования оператора Case

Пример :

Составить программу, которая по введенному номеру месяца выводит на экран название времени года.

```
Program m;
Var
  k:byte;
Begin
   Write('Введите номер месяца');
   Readln(k);
   Case k of
      1, 2, 12: writeln('Зима');
      3, 4, 5: writeln('Весна');
      6, 7, 8: writeln('Лето');
      9, 10, 11: writeln('Осень')
            else writeln ('Такого месяца нет');
   end;
end.
```

ОПЕРАТОРЫ ЦИКЛА

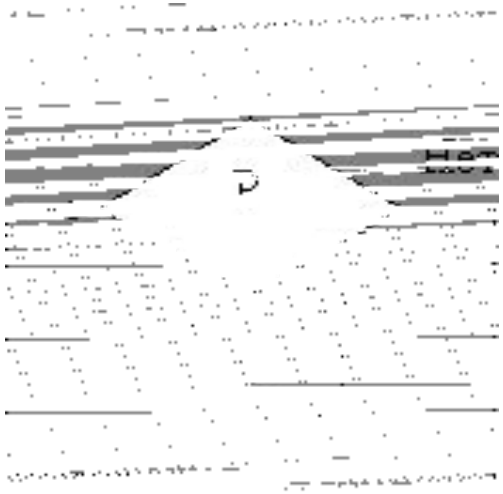
ЦИКЛ С ПРЕДУСЛОВИЕМ

While

Форма цикла с предусловием

Цикл с предусловием: проверка условия перед каждым выполнением «оператора»:

While <условие> do <оператор>;



В цикле While "оператор" выполняется если условие верно (True), если условие ложно (False), то цикл заканчивается, т. е. цикл While повторяется пока выполняется условие.
Цикл While начинается проверкой условия, поэтому, если начальное условие ложно, то "оператор" не выполняется ни разу.

Пример цикла с предусловием

Пример: Подсчет суммы цифр натурального числа (цикл с предусловием).

```
var a,x:integer;
    i,s:integer;
begin
     writeln('введите натуральное число');
     readln(a);
     x:=a;
     s:=0;
     while (x<>0) do
     begin
       s := s + (x mod 10);
       x := x div 10;
     end;
     writeln( 'Сумма цифр числа ',a,' = ', s );
     readln
end.
```

ЦИКЛ С ПОСТУСЛОВИЕМ

repeat

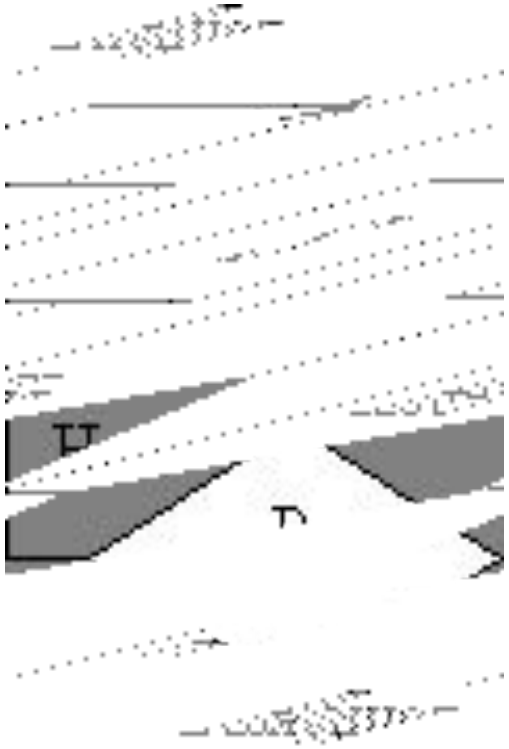
Цикл с постусловием

Цикл с постусловием: проверка условия после каждого выполнения "операторов".

repeat "операторы" until "условие";

Цикл Repeat повторяется, если условие ложно (False), и заканчивается, если условие верно (True), т. е. цикл Repeat повторяется до выполнения условия.

Цикл Repeat заканчивается проверкой условия, поэтому "операторы" выполняются не менее одного раза.



Пример цикла с постусловием

Подсчёт суммы положительных элементов последовательности до первого нулевого элемента. Последовательность чисел вводится поэлементно с клавиатуры.

```
var
 s,x:integer;
Begin
   writeln('введите числа x'); s:=0;
   repeat
     readln(x);
     if x>0 then s:=s+x;
   until x=0;
   writeln('summa= ',s);
   readln
End.
```

Подсчёт суммы положительных элементов последовательности до первого нулевого элемента. Последовательность чисел вводится поэлементно с клавиатуры. (использовать цикл с предусловием)

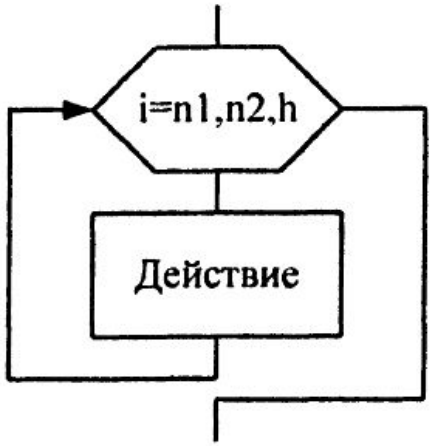
Цикл с параметром (счетный цикл)

for

Форма цикла со счетчиком

for *<переменная-счетчик> := <начальное значение>* ***to*** *<конечное значение>* ***do***

где <***переменная-счетчик***> — переменная целоисчисленного типа (byte, integer);
<***начальное значение***> — целое число, которое будет начальным значением переменной-счетчика;
<***конечное значение***> — целое число, которое должно быть больше <начального значения>.



В данном цикле переменная счетчик будет увеличиваться на единицу каждый раз при выполнении тела цикла, пока не достигнет конечного значения включительно. Тело цикла – оператор после служебного слова do. Если необходимо выполнить несколько операторов, то их замыкают между begin и end;

Форма цикла со счетчиком

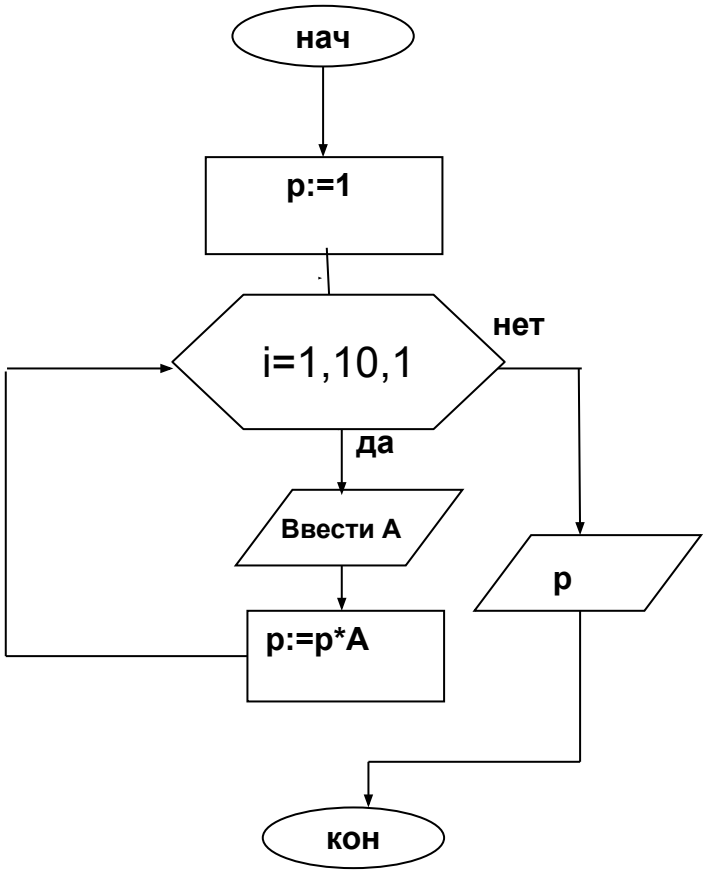
for <переменная-счетчик> := <начальное значение> ***downto*** *<конечное значение> do*

Действие этого цикла равнозначно предыдущему за одним исключением: параметр **downto** дает команду процессору уменьшать значение переменной-счетчика на единицу при каждом проходе тела цикла (а не увеличивать его, как в случае с параметром **to**). То есть начальное значение всегда должно быть выше конечного значения.

Пример цикла со счетчиком

Подсчитать произведение 10 чисел, введенных с клавиатуры.

Блок-схема



Решение

```
program my;
var i,p,a: integer;
 begin
    p:=1;
    for i:=1 to 10 do
        begin
                writeln ( ‘введите число’);
                readln (a);
                p:=p*a;
        end;
                   writeln (p);
 end.
```

Пример цикла со счетчиком

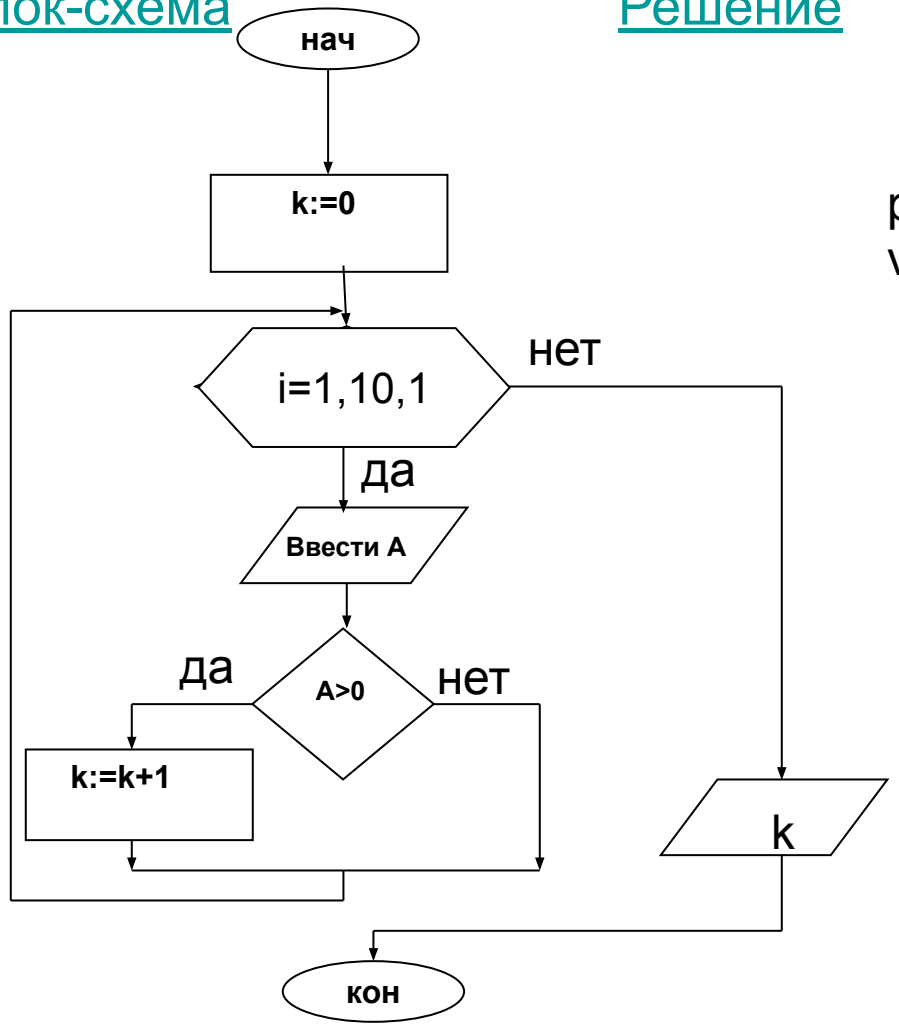
Подсчитать сумму положительных чисел среди 10 чисел введенных с клавиатуры.

```
program my;
var i,s,a: integer;
 begin
    s:=0;
    for i:=1 to 10 do
        begin
                writeln ( ‘введите число’);
                readln (a);
                if a>0 then s:=s+a;
        end;
                   writeln (s);
 end.
```

Пример цикла со счетчиком

Подсчитать количество положительных чисел среди 10 чисел введенных с клавиатуры.

Блок-схема



Решение

```
program my;
var i,k,a: integer;
 begin
    k:=0;
    for i:=1 to 10 do
        begin
                writeln ( ‘введите число’);
                readln (a);
                if a>0 then k:=k+1;
        end;
     writeln (k);
end.
```

```
program my;
var i,s: integer;
 begin
    s:=0;
    for i:=1 to 20 do
         begin
                      s:=s+i;
         end;
                    writeln (s);
 end.
```

Подсчитать сумму 20 слагаемых
1+2+3…+20

```
program my;
var i,p: integer; s: real;
 begin
    s:=0; p:=1;
    for i:=1 to 20 do
         begin
                        s:=s+p/i;
                        p:=-p;
         end;
                    writeln (s);
 end.
```

Подсчитать сумму 20 слагаемых
1-1/2+1/3…-1/20

Генератор случайных чисел

пример генератора случайных чисел от 2 до 5 .

```
var x:integer;
begin
  randomize;
  x :=random (4)+2;
end;
```

Здесь 4 - это результат выражения 5-2+1.

Сортировка массивов

- См. презентацию «Сортировка массивов»