ИНФОРМАТИКА
(Основы программирования)

И Ф Н и Т

(первый курс, весенний семестр)

Лукша Олег Игоревич

ВИФШ

к. 306, II уч. корп.

https://phys-el.ru/programming.php

СОДЕРЖАНИЕ

Материалы лекций доступны по адресу:
https://phys-el.ru/programming.php

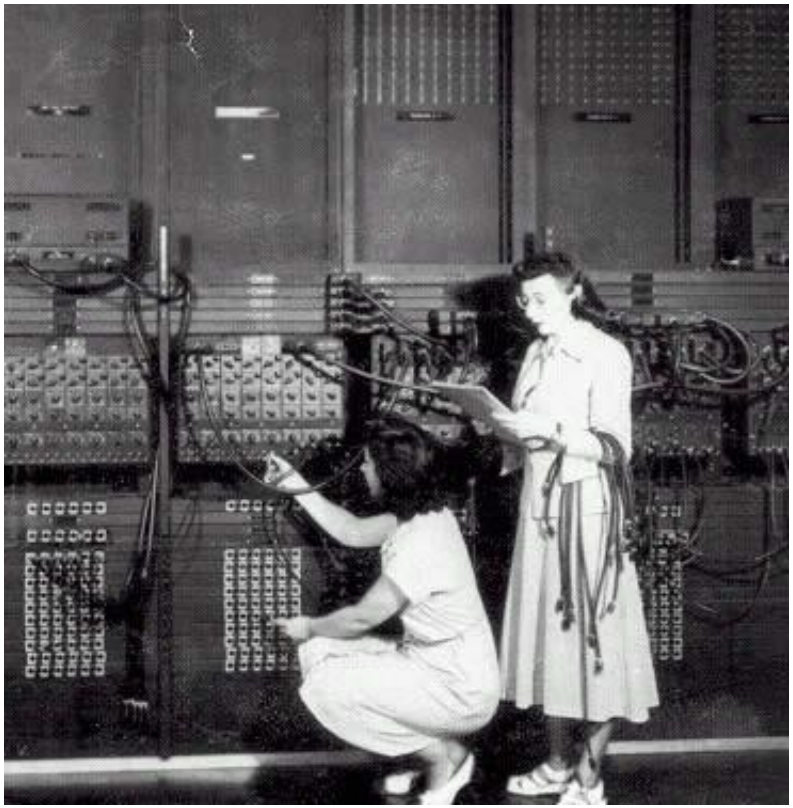
Л И Т Е Р А Т У Р А

Основная:

1. Иванова Г.С. *Основы программирования*, Изд. МГТУ, 2001.
2. Макарова Н.В., Волков В.Б. *Информатика: Учебник для вузов*. СПб.: Питер, 2011.
3. Свердлов С.З. *Языки программирования и методы трансляции : Учебное пособие*. СПб.: Питер, 2007.
4. Иванова Г.С. *Программирование* : учебник, М.: КНОРУС, 2013.
5. Фаронов В.В. *Система программирования Delphi*, СПб.: БХВ, 2006.

Дополнительная:

1. Себеста Р.У. *Основные концепции языков программирования*, Изд. дом «Вильямс», 2001.
2. Бусыгин Б.С., Коротенко Г.М., Коротенко Л.М. *Введение в современную информатику*, Днепропетровск, 2004.
3. Зелковиц М., Шоу А., Гэннон Дж. *Принципы разработки программного обеспечения*, Мир, 1982.
4. Румянцев Д.Г., Монастырский Л.Ф. *Путь программиста*, Изд. дом «Инфра-М», 2000.
5. Хьюз Дж., Мичтом Дж. *Структурный подход к программированию*, Мир, 1980.
6. Кнут Д. *Искусство программирования*, 1-3 тт., Изд. дом «Вильямс», 2000.
7. Вирт Н. *Алгоритмы и структуры данных*, Мир, 1989.



ENIAC (Electronic Numerical Integrator and Computer) считается первым универсальным электронным компьютером. Создан в 1945-1946 гг. в Высшем техническом училище *Пенсильванского университета* группой под рук. Д. Мочли (John Mauchly) и П. Эккерта (Presper Eckert). Предназначался для расчета баллистических таблиц для нужд артиллерии.

ENIAC состоял из 17468 электронных ламп и соединительных проводов, смонтированных на 40 панелях в комнате площадью 9x15 кв. метров (масса – 30 т., энергопотребление – 150 кВт).

Возможности ENIAC: тактовая частота – 100 кГц, время выполнения операции сложения – 0.2 мс, время выполнения операции умножения – 2.8 мс, емкость внутреннего запоминающего устройства – 20 десятизначных чисел.

Каждое изменение программы ENIAC требовало переключения сотен кабелей и установку в нужное положение приблизительно 6 тыс. переключателей, на что уходило два дня кропотливой ручной работы.

В новом компьютере **EDVAC** (**Electronic Discrete Variable Automatic Computer**, демонстрация – 1947 г.) Д. Мочли и П. Эккерт в качестве внутренней памяти предложили использовать ртутные линии задержки для увеличения объема памяти, а также ориентироваться на работу с двоичными числами, что позволяло упростить конструкцию арифметического устройства.

UNIVAC (Universal Automatic Computer) был разработан в период 1946-1951 гг. и первоначально предназначался для Национального бюро переписи населения США. Этот компьютер имел объем запоминающего устройства – 1000 72-битных слов, время сложения – 120 мкс, время умножения – 1800 мкс.

UNIVAC был оснащен программой-компоновщиком, который по заданному идентификатору осуществлял выборку нужной подпрограммы из специальной библиотеки (автор Г. Хоппер (Grace Hopper) назвала ее ***компилятором*** (compiler)).

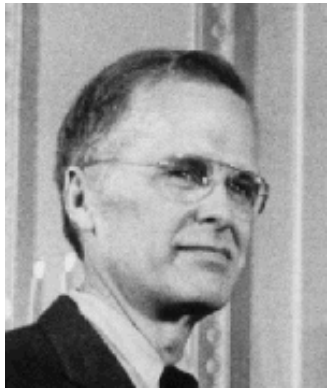
Коммерческий успех UNIVAC послужил толчком технологической революции, которая основывалась на прогрессе в разработке быстродействующей электроники и непрерывном совершенствовании языка общения человека с машиной.

Языки программирования - это тщательно составленные последовательности слов, букв, чисел и мнемонических сокращений, используемые для общения с компьютером.

Программы для ЭВМ первого поколения (ламповые машины 40-50 х гг.) создавались на языках машинных команд (*машинный код*).

Язык ассемблера - ориентированная на человека форма машинных команд. Языки программирования, имитирующие естественные языки и способные на основании одного предложения строить несколько команд компьютера, принято считать *языками высокого уровня*.

Первым широко используемым языком высокого уровня является **FORTRAN** (FORmula TRANslator), который разработала к 1957 г. группа инженеров фирмы *IBM* под руководством Джона Бекуса (John Backus) для программирования компьюера IBM 704.



FORTRAN был компилируемым языком и предназначался для работы с формулами, используемыми в математике и других научно-технических дисциплинах.

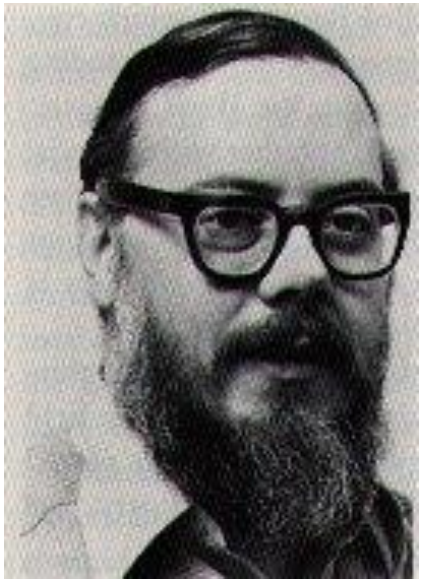
Язык **BASIC** (Beginner's All-purpose Symbolic Instruction Code) был разработан сотрудниками *Дартмутского колледжа* Т. Курцом (Thomas Kurtz) и Д. Кемени (John Kemeny) в 1964 г. Он создавался как язык, предназначенный главным образом для студентов, изучавших гуманитарные науки. Отличался простотой и ориентацией на интерактивное взаимодействие с компьютером посредством терминала. Получил широкое распространение в 70-80-х гг. как встроенный язык микрокомпьютеров.

ALGOL (ALGOrithmic Language) был создан на совещании в Цюрихе в мае 1958 г. как единый язык для научного программирования в США и в Европе. ALGOL многое унаследовал от языка FORTRAN, но в то же время основные понятия в нем были собраны в более логическую структуру. Для описания ALGOL 60 была впервые использована универсальная форма описания синтаксиса языков программирования – форма Бекуса-Наура (BNF – Backus-Naur form). Большинство императивных языков программирования прямо или косвенно являются потомками ALGOL 60. Он свыше 20 лет оставался единственным официальным средством представления алгоритмов в научной литературе.

Автор языка **Pascal** Никлаус Вирт (Niklaus Wirth) назвал его в честь французского философа и математика XVII века. Исходное описание языка Pascal было опубликовано в 1971 г. После своего появления Pascal стал очень популярным в сфере обучения программированию. Сильная структурированность делала Pascal весьма подходящим для создания больших программ. В исходной версии отсутствовали важные с практической точки зрения возможности. Pascal считается классическим языком для реализации принципов структурного программирования.

В работе "Заметки по структурному программированию" (1968 г.) Э. Дийкстра (E. Dijkstra) доказывал, что одна из основных причин "кризиса программного обеспечения" состоит в частом использовании в программах оператора безусловного перехода **GOTO**, который передает управление из одной точки в другую. Дийкстра предложил использовать три типа управляющих структур: следование, разветвление и цикл.

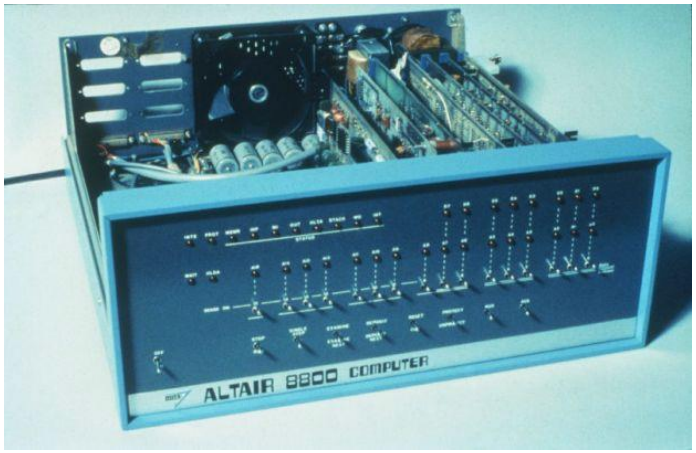
Язык **C** разработал в 1972 г. Деннис Ричи (Dennis Ritchie), специалист по системному программированию из фирмы *Bell Laboratories*. Язык С наиболее близок к языкам низкого уровня, поскольку обеспечивает непосредственный доступ к аппаратуре (на С было написано более 90 % всего кода центральной программы (ядра) операционной системы UNIX).



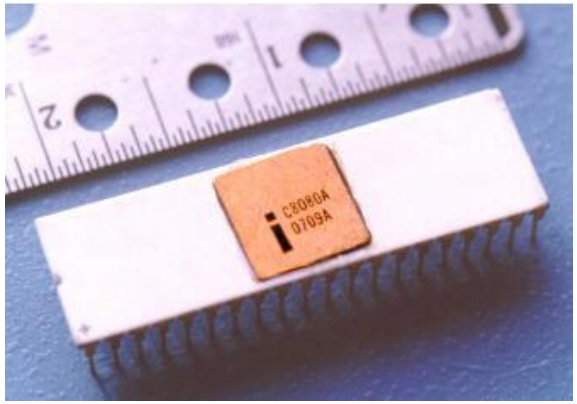
Успех С был неразрывно связан с тем, что в одном месте в одно и то же время появились сразу три грандиозных творения:
- язык программирования C,
- операционная система UNIX,
- мини-компьютер PDP-11 (в СССР аналоги СМ-4, СМ-1420, ДВК).

«С – это инструмент, острый как бритва: с его помощью можно создать и элегантную программу, и кровавое месиво»
(Брайан Керниган – соавтор книги Kernighan B.W., Ritchie D.M. “The C Programming Language” 1978).

В 1975 г. в журнале *Popular Electronics* был описан первый набор для сборки мини-компьютера **"ALTAIR - 8800"** (компания производитель - *Micro Instrumentation and Telemetry System*, *MITS*, цена - 397 долл.).



В компьютере "ALTAIR" использовался микропроцессор **Intel** (Integrated Electronics) **i8080** (n-MOS, 8-разрядный, тактовая частота 2 МГц, 6-мкм технология, 6 тыс. транзисторов). В первоначальном варианте пользователь самостоятельно создавал двоичную программу и вводил ее в машину с помощью переключателей на передней панели.



Среди первого программного обеспечения для "ALTAIR" был интерпретатор языка BASIC, который создали П. Аллен (Paul Allen) и Б. Гейтс (Bill Gates), основавшие вскоре фирму *Microsoft*.

Первые прикладные программы

- **текстовые процессоры**

  - Electric Pencil (Michael Schrayer, 1976).
  - WorldStar (John Barnaby, 1978).

- **электронные таблицы**

  - VisiCalc (Visible Calculator) (Daniel Bricklin, 1979).

  Изначально это программа была создана для персонального компьютера **Apple II** (Steve Wozniac, Steve Jobs, 1977), сыграв важную роль в успехе этой машины.

  - Lotus 1-2-3 (Mitch Kapor, 1983).

  Был намного проще в работе, чем VisiCalc, имел интегрированные возможности создания диаграмм, схем и баз данных. Благодаря Lotus 1-2-3 персональный компьютер быстро обрел статус настольной системы для организации бизнес-активности.

- **системы управления базами данных**

  - dBASE II (Wayne Ratliff, 1981).

Apple II

тактовая частота – 1 МГц;
ОЗУ – 48-64 Кб;
дисковод 5.25" – 140 Кб;
цветной монитор;
клавиатура, модем.

Первые операционные системы

- **CP/M (Control Program for Microcomputers)**

  8-разрядная ОС была разработана в 1974 г. Гэрри Килдолом (Gary Kildall) для компьютеров на базе процессора Intel 8080. Это была первая система, работающая на машинах разных производителей.

- **MS-DOS (Microsoft Disk Operating System)**

  16-разрядная ОС разработана фирмой *Microsoft* для IBM PC (1981 г.). Объем адресуемой памяти – 1 Мб. Текстовый режим экрана.

- **Mac OS (Macintosh Operating System)**

  Разработана в компании *Apple Computer Inc.* для компьютера Macintosh (1984 г.). Впервые применяется графический интерфейс (GUI – Graphical User Interface), который затем был использован в операционной системе Microsoft Windows 1.0 (1985 г.).

IBM PC XT

процессор Intel i8088 (29 тыс. транзисторов, 16 разрядов);

тактовая частота – 4.77 МГц;

8-разрядная шина;

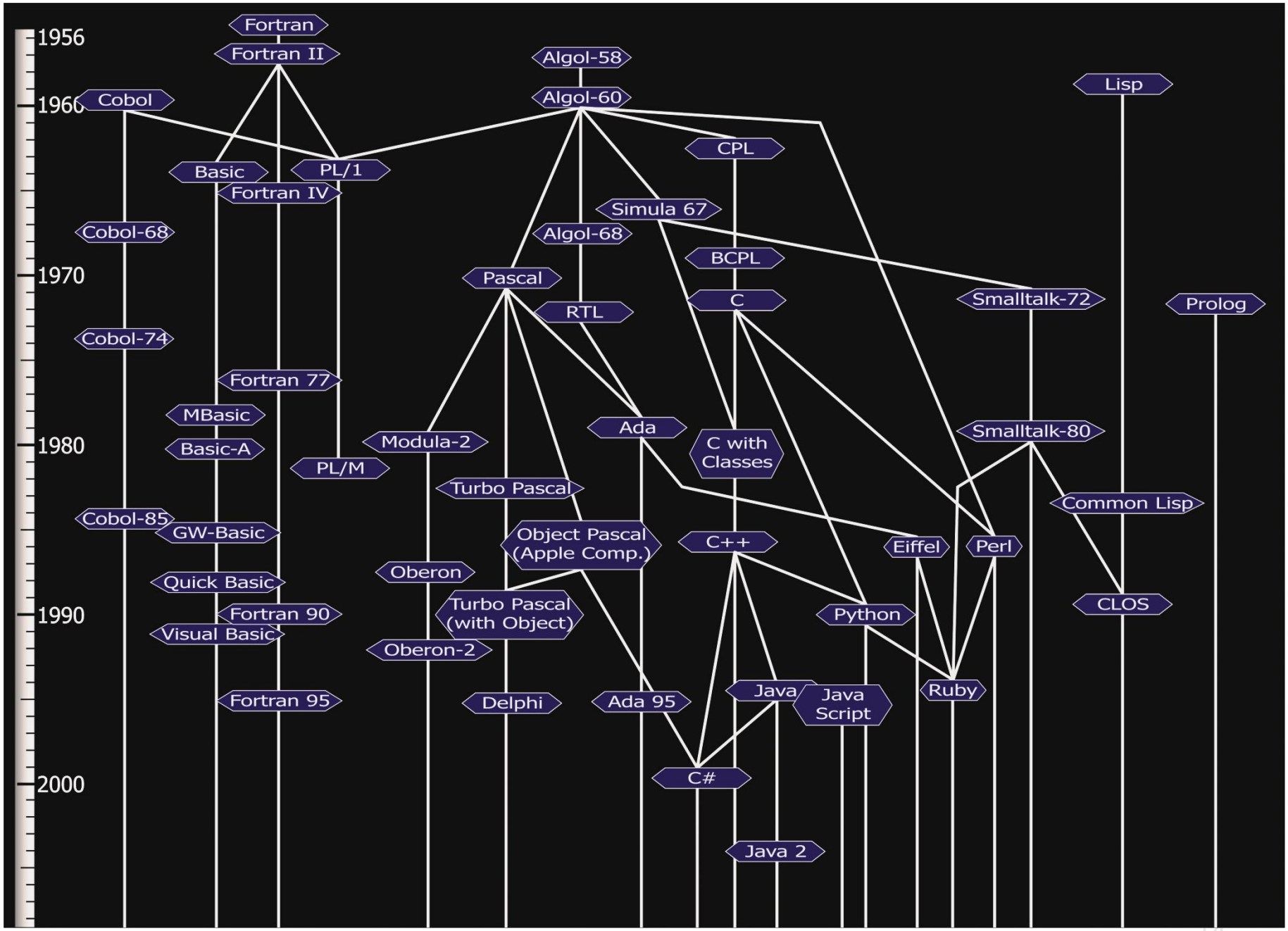
ОЗУ – 64 Кб;

дисковод 5.25" – 160 Кб;

Компилятор **Turbo Pascal** был создан в фирме *Borland International* (Philippe Kahn, 1983 г.). Разработчик языка Turbo Pascal А. Хейльсберг (Anders Hejlsberg) стал затем автором проекта Delphi, а после перехода в корпорацию Microsoft – ведущим архитектором языков .NET, а также автором языка C#. Среда Turbo Delphi – 2006 г.

В начале 80-х годов появилась разработанная сотрудником *Bell Laboratories* Б. Страуструпом (Bjarne Stroustrup) новая версия языка **C++**. В него были внесены ряд добавлений, главным из которых были средства объектно-ориентированного программирования, при сохранении ориентации на системное программирование. ***С-подобные языки сейчас являются признанными лидерами в области профессионального программирования***

Более простой по сравнению с C++ объектно-ориентированный язык **Java** был разработан в начале 90-х гг. Д. Гослингом (James Gosling) из компании *Sun Microsystems*. Язык Java поддерживает программирование для World Wide Web в форме платформо-независимых Java-апплетов.


1956
1960
1970
1980
1990
2000
Fortran
Fortran II
Algol-58
Lisp
Cobol
Algol-60
CPL
Basic
PL/1
Fortran IV
Simula 67
Cobol-68
Algol-68
BCPL
Pascal
C
Smalltalk-72
Prolog
RTL
Cobol-74
Fortran 77
MBasic
Ada
Smalltalk-80
Modula-2
C with Classes
Basic-A
PL/M
Turbo Pascal
Common Lisp
Cobol-85
GW-Basic
Object Pascal (Apple Comp.)
C++
Eiffel
Perl
Oberon
Quick Basic
Turbo Pascal (with Object)
CLOS
Fortran 90
Python
Visual Basic
Oberon-2
Java
Java Script
Ruby
Fortran 95
Delphi
Ada 95
C#
Java 2

Прогресс в программировании во многом связан с противостоянием языков высокого уровня:

- **FORTRAN** и **ALGOL 60** (70-е годы);
- **Pascal** и **C** (80-е годы);
- **C++** и **Java** (90-е годы).

Язык **C#** (июнь 2000 г.) был разработан в *Microsoft* как часть новой технологии **.NET**. В рамках этой технологии предусмотрена единая среда выполнения программ (Common Language Runtime, CLR), написанных на разных языках программирования. CLR основана на использовании промежуточного языка IL (Intermediate Language), исполняющего почти ту же роль, что и байт-код виртуальной машины языка Java. В составе первой версии Microsoft.NET были компиляторы в IL с C++, C#, Visual Basic.

Скриптовые (сценарные) языки широко распространены в последнее время, в частности для компьютерных сетей. Среди скриптовых языков можно выделить языки разметки – GML, TeX, SGML, XML и др., и универсальные языки – Tcl, Perl, PHP, Python, Ruby и др.

Глава 2. ПРОГРАММНОЕ УПРАВЛЕНИЕ КОМПЬЮТЕРОМ

- Программа и данные
- Поколения и классификация ЭВМ
- Устройство компьютера и особенности обмена информацией между его узлами
- Архитектура фон Неймана
- Машинный код, выполнение машинной команды
- Типы программ
- Системы счисления. Перевод чисел из одной системы счисления в другую
- Представление чисел в памяти компьютера
- Машинная математика

ПРОГРАММА И ДАННЫЕ

Программирование – это процесс создания программ.
Программа – набор инструкций, посланный вычислительной машине (компьютеру).

Носителем информации является *сообщение*.
Данные – это сообщения, закодированные в форму, пригодную для хранения и обработки их компьютером (на основе двоичного набора знаков – из-за простоты распознавания и хранения на физическом уровне сигналов, имеющих только два состояния: "включен – выключен").
Сейчас реализуется с помощью *CMOS-транзисторов*.

Для кодировки *сообщений* применяется двоичный набор, состоящий из двух знаков 0 и 1 (*binary digit*, сокращенно *bit*).

Порядок выполнения операций над данными строится на основе некоторого *алгоритма*.

Программу можно рассматривать как *алгоритм*, записанный на понятном для компьютера языке, и *данные*, которые компьютер будет обрабатывать в соответствии с этим алгоритмом.

ПОКОЛЕНИЯ ЭВМ

- **Первое** (электронные лампы – 1945-1955 гг.). Быстродействие – от неск. сотен до неск. тысяч операций в сек. Примеры: ENIAC, EDVAC, IBM 701; СССР – МЭСМ, БЭСМ.
- **Второе** (транзисторы – 1955-1965 гг.). Быстродействие – до сотен тыс. и миллионов операций в сек. Системное ПО, программирование на языках высокого уровня. Примеры: IBM 1620, PDP-1.
- **Третье** (интегральные схемы (ИС) – 1965-1980 гг.). Быстродействие – до дес. миллионов операций в сек. Массовое производство ИС. Примеры: IBM серии System/360, PDP-8; СССР – ЭВМ серии ЕС.
- **Четвертое** (сверхбольшие ИС – с 1980 г.). Микроэлектроника. Микропроцессор. Компьютерные сети. Примеры: ПК IBM PC, Apple Macintosh, суперкомпьютеры.
- **Пятое** (?). В настоящее время процесс продолжается в рамках парадигмы сверхбольшой ИС. Цель – "интеллектуализация" компьютера.

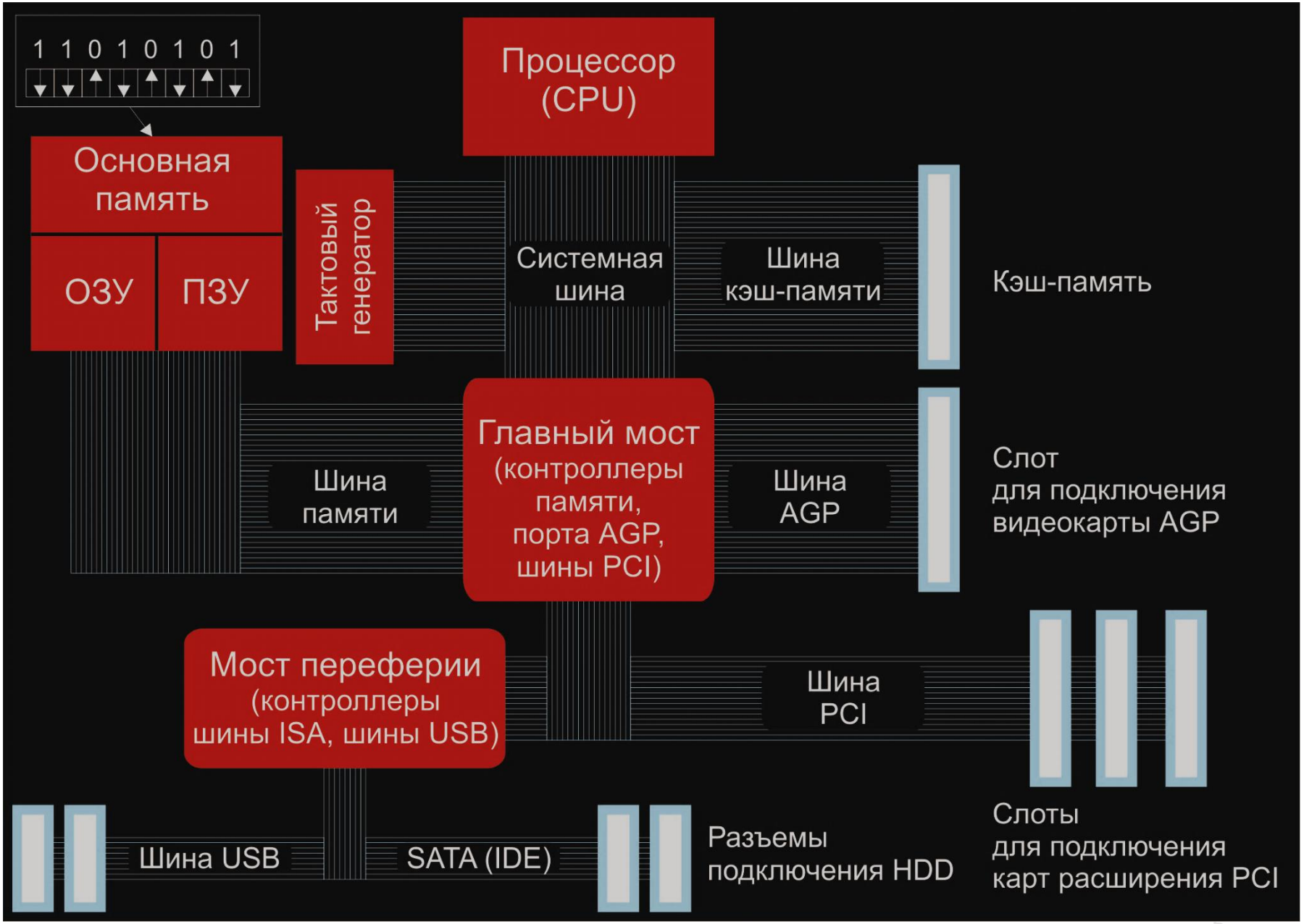
Переход к более высокому поколению – это принципиальное снижение размеров, потребляемой мощности, стоимости и увеличение быстродействия.

КЛАССИФИКАЦИЯ КОМПЬЮТЕРОВ

Мфлопс (MFLOPS, Million of FLoating point Operation Per Second)

МИПС (MIPS, Million Instruction Per Second)

АРХИТЕКТУРА КОМПЬЮТЕРА



АРХИТЕКТУРА КОМПЬЮТЕРА (ОСНОВНЫЕ УЗЛЫ)

Процессор. Характеристики – *разрядность*, *тактовая частота*, *набор команд*.

Основная память (ОЗУ + ПЗУ). Объем памяти измеряется в *байтах.* **Байт** – группа из восьми бит, обрабатываемая как единое целое. Емкость ячейки памяти – 1 байт. Производные: Кбайт (1024 (2^{10}) байт), Мбайт (1 048 576 (2^{20}) байт) , Гбайт (1 073 741 824 (2^{30}) байт).

Чипсет (Chipset). В состав входят *главный мост* и *мост периферии*.

Компьютерные шины – набор линий-проводников для передачи информации между узлами. Выделяют *системную шину*, *периферийные шины*. Архитектура любой из шин включает линии для обмена данными (*шина данных*), для адресации данных (*шина адресов*), для управления данными (*шина управления*). **Пропускная способность шины (Мбайт/с)** определяется ее *разрядностью*, умноженной на *тактовую частоту*.

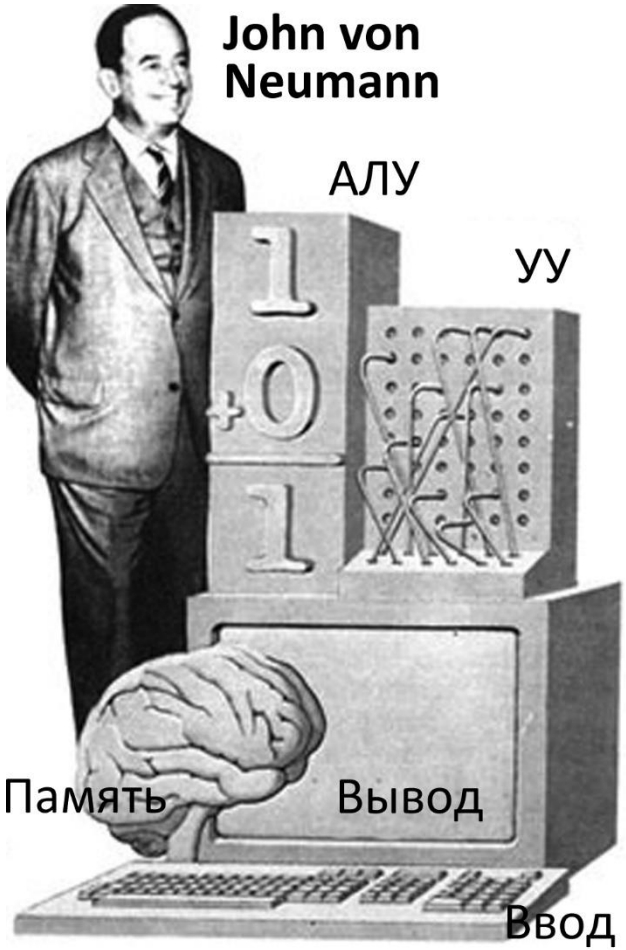
Кэш-память. Предназначена для временного хранения копий блоков данных тех областей ОЗУ, к которым выполнялись последние обращения и весьма вероятны обращения в ближайшие такты. Имеется кэш-память *1-го*, *2-го, 3-го* уровней.

Другие узлы: магнитные диски, флэш-память, клавиатура, мышь, монитор, и т.д.

АРХИТЕКТУРА ФОН НЕЙМАНА

Машина (архитектура) фон Неймана построена на следующих принципах:

- **наличие трех основных элементов** – *памяти* (для хранения информации), *арифметико-логического устройства, АЛУ* (выполняет команды) и *устройства управления, УУ* (указывает команды для выполнения);
- **однородности памяти** – команды и данные хранятся в одной и той же памяти; вид записи команд и данных одинаков;
- **адресности** – структурно память состоит из пронумерованных ячеек, процессору в произвольный момент времени доступна любая ячейка;
- **программного управления** – программа состоит из набора команд, которые выполняются процессором автоматически друг за другом в определенной последовательности;
- **наличие канала связи** между памятью и процессором.



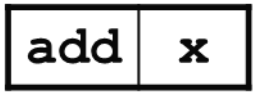
МАШИННЫЙ КОД

Машинная команда состоит из кода выполняемой операции (**оператор**) и адресной части (**операнды**).

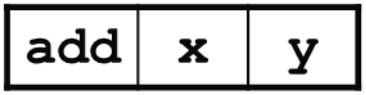
Машинный код – *закодированное* представление команды процессора

7C90104A 00648B0D

Регистры – поименованные ячейки памяти процессора (сумматор – регистр АЛУ).



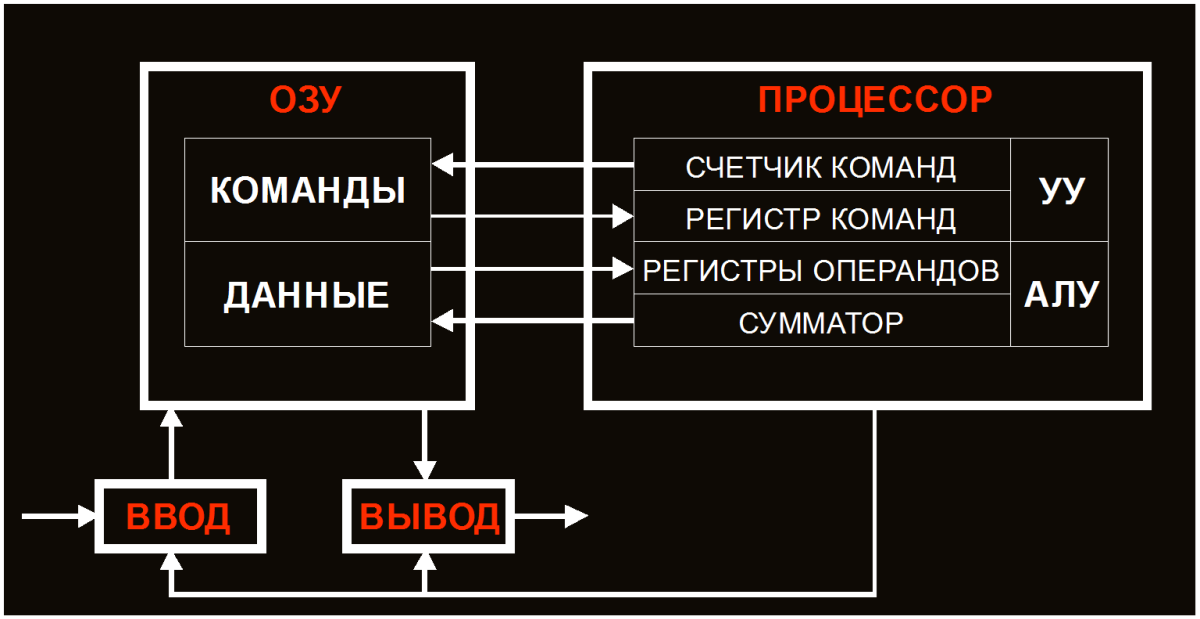
одноадресная команда – содержимое ячейки x ОЗУ сложить с содержимым сумматора, а результат оставить в сумматоре;



двухадресная команда – сложить содержимое ячеек x и y, а результат поместить в ячейку y;

трехадресная команда – содержимое ячейки x сложить с содержимым ячейки y, сумму поместить в ячейку z);

ВЫПОЛНЕНИЕ МАШИННОЙ КОМАНДЫ



1. из ячейки памяти, адрес которой хранится в счетчике команд, выбирается очередная команда; содержимое счетчика увеличивается на длину команды;
2. эта команда передается в УУ на регистр команд;
3. УУ расшифровывает адресное поле команды;
4. по сигналам УУ операнды считываются из памяти и записываются в АЛУ на регистры операндов;
5. УУ расшифровывает код операции и выдает в АЛУ сигнал выполнить соответствующую операцию над данными;
6. результат операции либо остается в процессоре, либо отправляется в память, если в команде был указан адрес результата.

ТИПЫ ПРОГРАММ

СИСТЕМНОЕ ПО

Операционные системы
файловые системы
драйверы устройств

Базовое ПО
BIOS,
Basic Input/Output System

Служебные программы
утилиты
антивирусные средства

Системы программирования
включают трансляторы, библиотеки, отладчики, среду разработки и т.д.

ПРИКЛАДНОЕ ПО

Обработчики текста
текстовые процессоры
издательские системы

Интегрированные офисные системы

СУБД

Графические системы
ðедакторы векторой графики
редакторы растровой графики

Переводчики, игры, и др.

СИСТЕМЫ СЧИСЛЕНИЯ

Система счисления – способ отображения чисел с помощью символов.
Совокупность символов – алфавит, число символов в алфавите – размерность.
В позиционной системе счисления вес символа (цифры) зависит от его позиции
$\mathbf{V = B^N}$, B – основание системы счисления, N – порядковый номер позиции.

Двоичная система счисления – позиционная (основание – 2).

Перевод числа из двоичной системы в десятичную:

$$(1101.1011)_2 = 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3} + 1*2^{-4} = 13 + 0.5 + 0.125 + 0.0625 = (13.6875)_{10}$$

Перевод *целого числа* из десятичной системы в двоичную осуществляется последовательным делением на 2. В качестве остатка от деления получается очередная цифра двоичного числа, начиная с младшей:

13/2 = 6 (остаток 1 - младшая цифра), 6/2 = 3 (0), 3/2 = 1 (1), 1/2 = 0 (1). Результат - $(1101)_2$

Перевод *дроби* из десятичной системы в двоичную осуществляется умножением на 2. Целая часть полученного числа – очередная цифра двоичного, начиная с первой цифры после запятой:

0.6875*2 = 1.375 (первая цифра - 1), 0.375*2 = 0.75 (0), 0.75*2 = 1.5 (1), 0.5*2 = 1.0 (1). Результат - $(0.1011)_2$

СИСТЕМЫ СЧИСЛЕНИЯ

Восьмеричная система счисления – позиционная (основание – 8, используются цифры 0 1 2 3 4 5 6 7).

Шестнадцатеричная система счисления – позиционная (основание – 16, используются цифры 0 1 2 3 4 5 6 7 8 9 и первые буквы латинского алфавита A B C D E F).

Перевод восьмеричных и шестнадцатеричных чисел в двоичную систему (и обратно) осуществляется заменой каждой цифры эквивалентной ей двоичной триадой (тройкой цифр) или тетрадой (четверкой цифр).

$$(537.1)_8 = \begin{matrix} 101 & 011 & 111. & 001 \\ 5 & 3 & 7. & 1 \end{matrix} = (101\ 011\ 111.001)_2$$

$$(1A3.F)_{16} = \begin{matrix} 0001 & 1010 & 0011. & 1111 \\ 1 & A & 3. & F \end{matrix} = (1\ 1010\ 0011.1111)_2$$

"10"	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
"2"	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111	10000
"8"	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17	20
"16"	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10

ПРЕДСТАВЛЕНИЕ ЧИСЕЛ (ЦЕЛЫХ) В ПАМЯТИ

Представление (вид) числа в памяти определяется тем **типом**, к которому оно принадлежит. В частности, тип числа задает количество двоичных разрядов, отводимых под хранение числа.

Для хранения числа в памяти отводится целое число ячеек емкостью 1 байт – 8 бит (разрядов). Число занимает всю ячейку, независимо от того, сколько двоичных разрядов требуется для его представления.

Целые беззнаковые типы (представление числа 5)

BYTE (1 байт, 0...255)	**WORD** (2 байта, 0...65535)
`0000 0101`	`0000 0000 0000 0101`

ПРЕДСТАВЛЕНИЕ ЧИСЕЛ (ЦЕЛЫХ) В ПАМЯТИ

Для хранения чисел со знаком старший разряд отводится под знак (0 – если число положительное, 1 – если отрицательное).

Целые знаковые типы (представление числа 5)

SHORTINT (1 байт, -128...127)

```
0 000 0101
```

SMALLINT (2 байта, -32768...32767)

```
0 000 0000 0000 0101
```

Отрицательные числа представлены в виде **дополнительного кода**.
Для получения дополнительного кода необходимо сначала поменять все разряды числа на обратные, а затем к полученному результату прибавить 1.

Целые знаковые типы (представление числа –5)

SHORTINT (1 байт, -128...127)

```
1 111 1011
```

SMALLINT (2 байта, -32768...32767)

```
1 111 1111 1111 1011
```

МАШИННАЯ МАТЕМАТИКА

Все математические операции с числами основаны на сложении.

Сложение осуществляется поразрядно с соблюдением правила: "Если в результате сложения двух соответствующих разрядов чисел получилось число большее или равное основанию системы счисления (2), то следует из полученного числа отнять основание системы счисления и записать в строке итога полученный результат. Кроме того, необходимо запомнить единицу с тем, чтобы добавить ее при сложении следующего разряда".

```
  0 1 0 0  1 0 0 1     (73)10
+ 0 0 1 0  0 1 0 1     (37)10
-----------------------------
  0 1 1 0  1 1 1 0    (110)10
```

Вычитание заменяется сложением чисел, одно из который берется с обратным знаком (используется дополнительный код).

```
  0  0 0 0  0 1 1 1     (7)10
+ 1  1 1 1  1 0 1 1    (-5)10
-----------------------------
  0  0 0 0  0 0 1 0     (2)10
```

МАШИННАЯ МАТЕМАТИКА

Умножение заменяется сложением чисел, сдвинутых на разное число двоичных разрядов подобно тому, как это делается при умножении чисел "в столбик".

```
    1 0 . 1      (2.5)10
x     1 . 1      (1.5)10
  ---------
    1 . 0 1
+ 1 0 . 1
  ---------------
  1 1 . 1 1    (3.75)10
```

Деление выполняется через вычитание (путем многократного прибавления к делимому дополнительного кода делителя).

```
  1 1 1 1 | 1 1          15 / 3 = 5
- 1 1     |-------
          | 1 0 1
  -------
      1 1
-     1 1
  -------
        0
```

Возведение в степень выполняется через умножение и т.д.

Имеется **математический сопроцессор** – специальное устройство, ускоряющее вычисления с плавающей точкой.

МАШИННАЯ МАТЕМАТИКА

Работа компьютера (в том числе выполнение арифметических операций) основана на **логических действиях.**

Например, возможны только четыре комбинации соответствующих разрядов при сложении двух чисел, представленных в двоичном виде.

Разряд I слагаемого	Разряд II слагаемого
0	0
1	0
0	1
1	1

Правила по установке соответствующего разряда итогового числа построено по законам логики:
"Если складываются разряды с равным состоянием (ноль с нулем или единица с единицей), то итоговый разряд устанавливается равным нулю. В противном случае, он устанавливается равным единице. Если складываются два разряда, равные единице, то вырабатывается сигнал переноса единицы в следующий разряд".

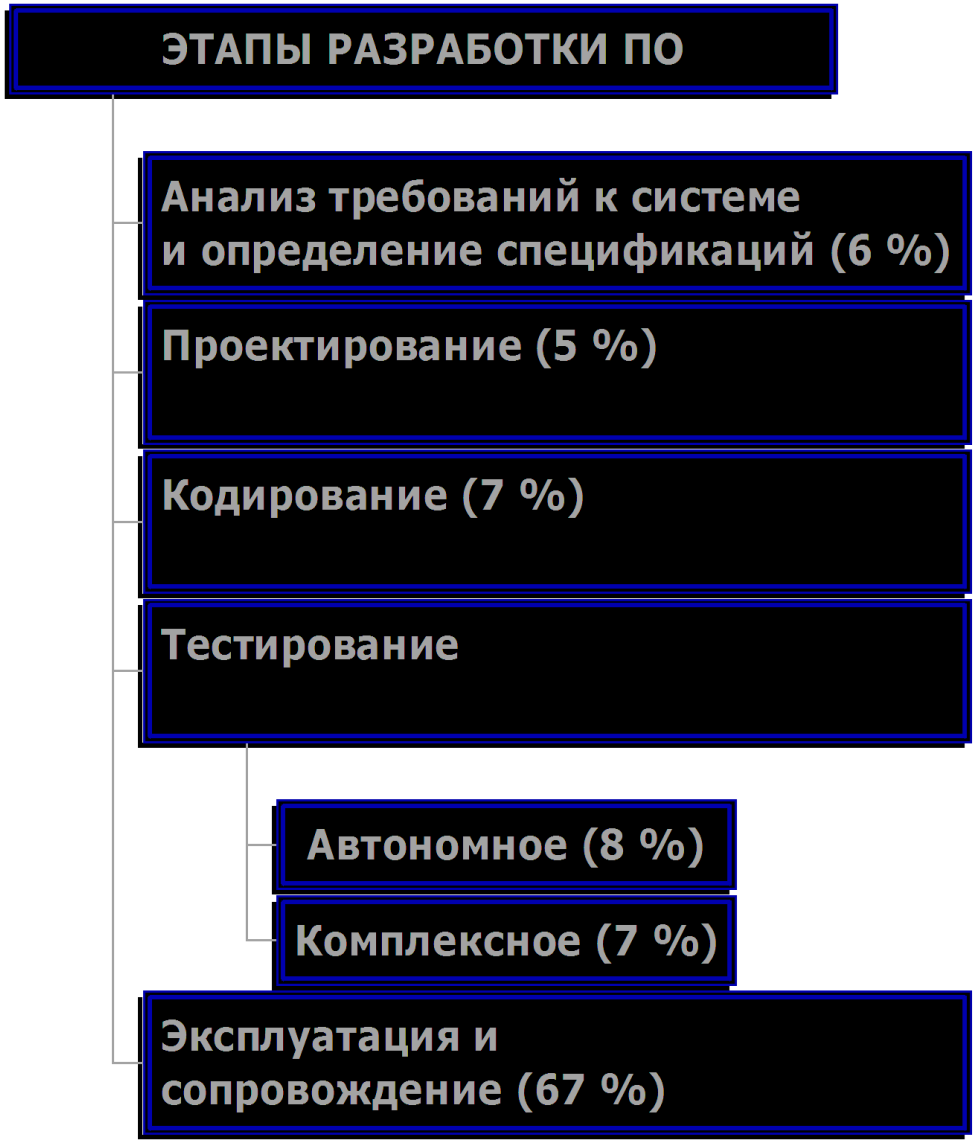
Логическая конструкция "Если условие обращается в истину, то выполнить некую последовательность действий" называется **импликацией**.
Более сложную конструкцию: "Если …, то…, иначе" реализуют с помощью оператора: **if – then – else**.

Глава 3. ЭТАПЫ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ И ЯЗЫКИ ПРОГРАММИРОВАНИЯ

- Программное обеспечение
- Этапы разработки программного обеспечения
- Метод пошаговой детализации
- Языки программирования низкого уровня и высокого уровня
- Императивные, объектно-ориентированные, функциональные, логические языки программирования
- Классификация языков программирования
- Типы трансляторов
- Схема компилятора
- Языки веб-программирования

ЭТАПЫ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Программное обеспечение (пакет программ) – группа взаимосвязанных и взаимодействующих программ, предназначенных для решения любой задачи из конкретной области (например, для моделирования технологических процессов, для выполнения графических работ). Программы какого-либо пакета рассчитаны на совместное использование в различных комбинациях друг с другом.



ЭТАПЫ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

I. Анализ требований к системе и определение спецификаций

Первичный документ – постановка задачи. Результат (после нескольких итераций) – **техническое задание**, в котором, как правило, содержатся название системы, цели создания, характеристика области применения, требования к системе в целом (интерфейс, особые требования к отдельным модулям, безопасность, и т.д.), информационная база (структура базы данных, с которой будет взаимодействовать программная система), программное обеспечение (обоснование выбора языка программирования), техническое обеспечение, описание данных для тестирования.

На основании технического задания формируются **спецификации** – описание количества и режимов работы модулей, их взаимодействия.

ЭТАПЫ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

II. Проектирование

На данном этапе разрабатываются **алгоритмы**, задаваемые спецификациями, и формируется общая структура будущей программы путем детальной проработки последовательности ее действий. Запись – с помощью блок-схем или псевдокода.

Применяются различные технологии – структурное программирование, объектно-ориентированное программирование и др. В рамках структурного программирования используется метод **пошаговой детализации**, при котором процесс преобразования исходных данных в результат вначале представляется в виде последовательности небольшого числа простых этапов (задач). На следующем шаге задачи разбиваются на последовательность подзадач нового уровня и т.д. Детализация заканчивается, когда каждый отдельный этап может быть простым способом записан на выбранном языке, или представляет собой известную задачу, для которой уже имеется готовая программа.

III. Кодирование

Кодирование представляет собой реализацию разработанных алгоритмов, составление по ним текстов программы с использованием конкретного языка программирования. Включает процесс трансляции – перевода программы в последовательность машинных команд (машинный код).

ЭТАПЫ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

IV. Тестирование

При **автономном тестировании** каждый модуль проверяется отдельно. При этом программная среда модуля имитируется с помощью программы управления тестированием, содержащей фиктивные программы вместо реальных подпрограмм, к которым имеется обращение из данного модуля.
При **комплексном тестировании** производится совместная проверка групп программных компонентов.

В процессе тестирования происходит оптимизация системы (разгрузка участков повторяемости – циклов, замена сложных операций на более простые, экономия памяти и т.д.).

V. Эксплуатация и сопровождение

На данный этап приходится основная часть расходов, затрачиваемых в течение жизненного цикла системы.
Причины выпуска новых версий (модификаций) ПО:

- необходимость исправления ошибок, выявленных в процессе эксплуатации;
- необходимость совершенствования, например, улучшения интерфейса или расширения состава, выполняемых функций;
- изменение среды (появление новых технических средств и/или программ).

ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Языки программирования – это тщательно составленные последовательности слов, букв, чисел и мнемонических сокращений, используемые для общения с компьютером.

Языки низкого уровня

Машинная команда состоит из кода операции и адресной части. Код операции указывает вид выполняемого действия (сложить, переместить и т.д.). Адресная часть содержит адреса (номера) ячеек памяти, в которых расположены операнды, и адрес ячейки, куда следует поместить результат.

Языки ассемблера отличаются от машинного кода тем, что коды операций заменены буквенными обозначениями (например, ADD – сложить, MOV – переслать данные) и вместо номеров ячеек используются символические адреса.

Ассемблер – программа, преобразующая мнемонику языка ассемблера непосредственно в двоичные представления машинных команд.

Языки ассемблера – машинно-зависимые языки низкого уровня, в которых одна команда соответствует одной машинной команде (возможны макрокоманды – вызовы макросов, объединяющих несколько машинных команд).

ЯЗЫКИ ВЫСОКОГО УРОВНЯ

Языки программирования, имитирующие естественные языки и способные на основании одного предложения строить несколько команд компьютера, принято считать **языками высокого уровня**.

Варианты языков – подмножества, расширения, диалекты.

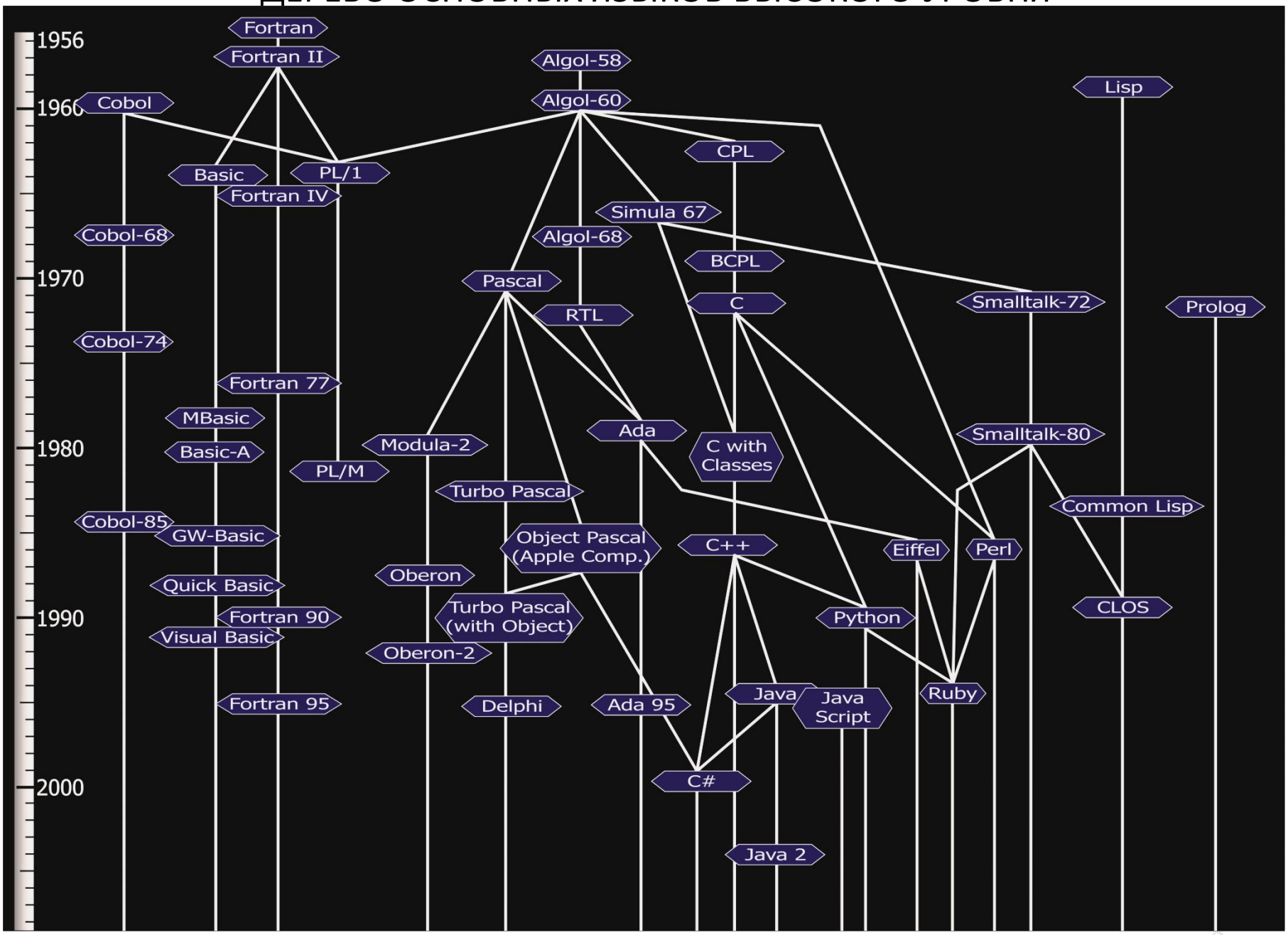
Критерии оценки (выбора) языка программирования:

- легкость чтения программ (особенно важна с точки зрения эксплуатации и сопровождения программ);
- легкость создания программ в выбранной области;
- надежность (т.е. насколько программа соответствует своему предназначению в любых условиях).

Основные характеристики языка:

- простота;
- структурированность;
- доступные типы и структуры данных;
- "естественность" синтаксиса;
- поддержка абстракции;
- выразительность;
- проверка совместимости типов;
- обработка исключительных ситуаций.

ДЕРЕВО ОСНОВНЫХ ЯЗЫКОВ ВЫСОКОГО УРОВНЯ



ИМПЕРАТИВНЫЕ ЯЗЫКИ

Основными элементами **императивных** языков программирования являются переменные (моделируют ячейки памяти), операторы присваивания (определяют и изменяют содержимое ячейки памяти), итеративная форма повторений (циклы). На использование этих языков ориентирована технология структурного программирования, базирующаяся на **процедурной декомпозиции**.

FORTRAN (FORmula TRANslator). Считается первым компилируемым языком высокого уровня. Ориентирован на создание программ, выполняющих естественно-научные и математические расчеты.

COBOL (COmmon Business Oriented Language). Структура и словарь близки к обычному английскому языку. Является основным языком в США для обработки данных в таких учреждениях как банки и страховые компании. Раздел данных – сильная сторона языка COBOL, тогда как раздел процедур – относительно слабая.

BASIC (Beginner's All-purpose Symbolic Instruction Code). Исходный BASIC был легок для изучения начинающими и не требователен к ресурсам компьютера. Популярный язык микрокомпьютеров в конце 70-х и начале 80-х годов. Важнейшим аспектом была его ориентация на использование удаленного доступа к компьютеру посредством терминала. Возрождение языка BASIC произошло в начале 90-х годов с выходом языка Visual BASIC (Microsoft, 1991), который является одним из основных языков в платформе Microsoft.NET.

ALGOL

ALGOL (ALGOrithmic Language). Язык ALGOL – результат попытки создания универсального языка. В этом языке была формализована концепция типов данных, реализована идея составных операторов. В диалекте ALGOL 60 была введена концепция блочной структуры, что позволяло программистам локализовать части программы, вводя разные области видимости. Имелась возможность передавать параметры подпрограммам по значению и по имени, а также создания рекурсивных процедур. Были доступны динамические массивы (ALGOL 68), т.е. массивы, диапазон индексов которых задавался переменной, которая могла менять свое значение во время работы программы. Свыше 20 лет ALGOL оставался единственным официальным средством представления алгоритмов в научной литературе.

ИМПЕРАТИВНЫЕ ЯЗЫКИ – ПОТОМКИ ЯЗЫКА ALGOL

Pascal. Обеспечивает возможность создания больших программ, поддерживая их строгую логическую структуру. Для коротких программ может оказаться слишком громоздким. Считается важнейшим инструментом для обучения методам структурного программирования.

C. Отличная замена ассемблеру для низкоуровневого программирования, с одной стороны, и применения принципов структурного программирования – с другой. Популярен благодаря многим решениям, сделавшим запись программы на С весьма компактной. В целом С – очень мощный и в то же время изящный язык. Одной из особенностей является отсутствие полной проверки типов, что порождает гибкость языка в сочетании с ненадежностью.

Ada. Происходит от Pascal, но заметно сложнее его. Содержит средства выделения в отдельные модули объектов данных, обеспечивает поддержку использования абстракции данных в структуре программы. Содержит средства обработки исключительных ситуаций. Программные блоки могут быть настраиваемыми. Обеспечивает параллельное выполнение программных блоков (заданий).

Modula-2. По сравнению с Pascal добавлена поддержка модулей. Имеются абстрактные типы данных, возможность использования процедур как типов, низкоуровневые средства системного программирования и сопрограммы.

ОБЪЕКТНО-ОРИЕНТИРОВАННЫЕ ЯЗЫКИ

Объектно-ориентированные языки программирования в наибольшей степени способны реализовать технологию объектно-ориентированного программирования, которая основана на **объектной декомпозиции**.

Smalltalk. Основные идеи происходят от первого ООЯ SIMULA 67. Программными модулями языка Smalltalk являются объекты – структуры, объединяющие локальные данные и набор операций (методы), которые доступны другим объектам. Метод определяет реакцию объекта на определенное сообщение, соответствующее данному методу. Абстракциями объектов являются классы. Экземпляры классов – объекты программы. Имеется иерархия классов. Подклассы наследуют функциональные возможности и переменные родительского класса, имея возможность добавлять новые функциональные возможности, а также изменять или скрывать унаследованные.

C++. Представляет собой надстройку над языком C, поддерживающую большинство возможностей, открытых языком Smalltalk (т.е. C++ – это объединение императивного и объектно-ориентированного языков). Поддерживается наследование и множественное наследование. Динамическое связывание обеспечивается функциями виртуального класса. Недостатки: объемность и сложность.

ОБЪЕКТНО-ОРИЕНТИРОВАННЫЕ ЯЗЫКИ

Смешанные языки, созданные путем введения в соответствующий императивный язык объектно-ориентированный поддержки: **Oberon, Delphi, Ada 95.**

Java. Является непосредственным наследником C++. Отличается от него отсутствием некоторых потенциально ненадежных механизмов, а также тем, что устранены любые, не относящиеся к объектно-ориентированному программированию, средства. Все объекты в этом языке – динамические. Особенность языка Java состоит также в использовании особого вида трансляции – динамической кодогенерации. Основные девизы Java – простота, объектная ориентированность, сетевые возможности, надежность, независимость от архитектуры, интерпретируемость.

C#. Разработан в Microsoft как часть новой технологии .NET. Устранены некоторые недостатки языка Java: присутствуют перечисления, статические структуры, многомерные массивы, передача параметров по ссылке. Предусмотрена единая среда выполнения программ (Common Language Runtime, CLR), написанных на разных языках. Трансляция основана на использовании промежуточного кода (Intermediate Language, IL). Компиляция увеличивает быстродействие.

ФУНКЦИОНАЛЬНЫЕ И ЛОГИЧЕСКИЕ ЯЗЫКИ

LISP (LISt Processing). Разрабатывался в связи с работами в области искусственного интеллекта. В чистом языке LISP (1958 г.) существовало только два типа структур данных: атомы и списки. Списки представляли собой совокупность узлов, каждый из которых представляет собой два указателя. Первый указывает на представление атома, т.е. на его символьное или числовое значение, а второй – на следующий элемент списка.
Все вычисления в этом языке, который разрабатывался как язык **функционального программирования**, производятся путем применения функций к аргументам. Итеративные процессы определяются с помощью рекурсивных функций.

Prolog (PROgramming LOGig) – язык **логического программирования**, основная идея которого состоит в использовании формальной логической записи для сообщения компьютеру вычислительных процессов. Программирование в нем является непроцедурным. Программы не устанавливают точно, как должен вычисляться результат, а только описывают его форму. Транслирующая система сама должна выбрать нужный порядок выполнения команд, который приведет к желаемому результату. Логическое программирование было использовано, главным образом, в системах управления реляционными базами данных, экспертных системах и обработке текстов на естественных языках.

КЛАССИФИКАЦИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

По парадигмам (система идей и понятий, определяющих стиль программы) – императивные, функциональные, логические, объектно-ориентированные.

По факту создания процесса – языки, создающие процесс (после запуска программы создается отдельный процесс выполнения этой программы, C, C++, Pascal, Delphi и др.), сценарные языки (сценарий, или скрипт – программа, которую выполняет другая программа, PHP, Python, Ruby и др.)

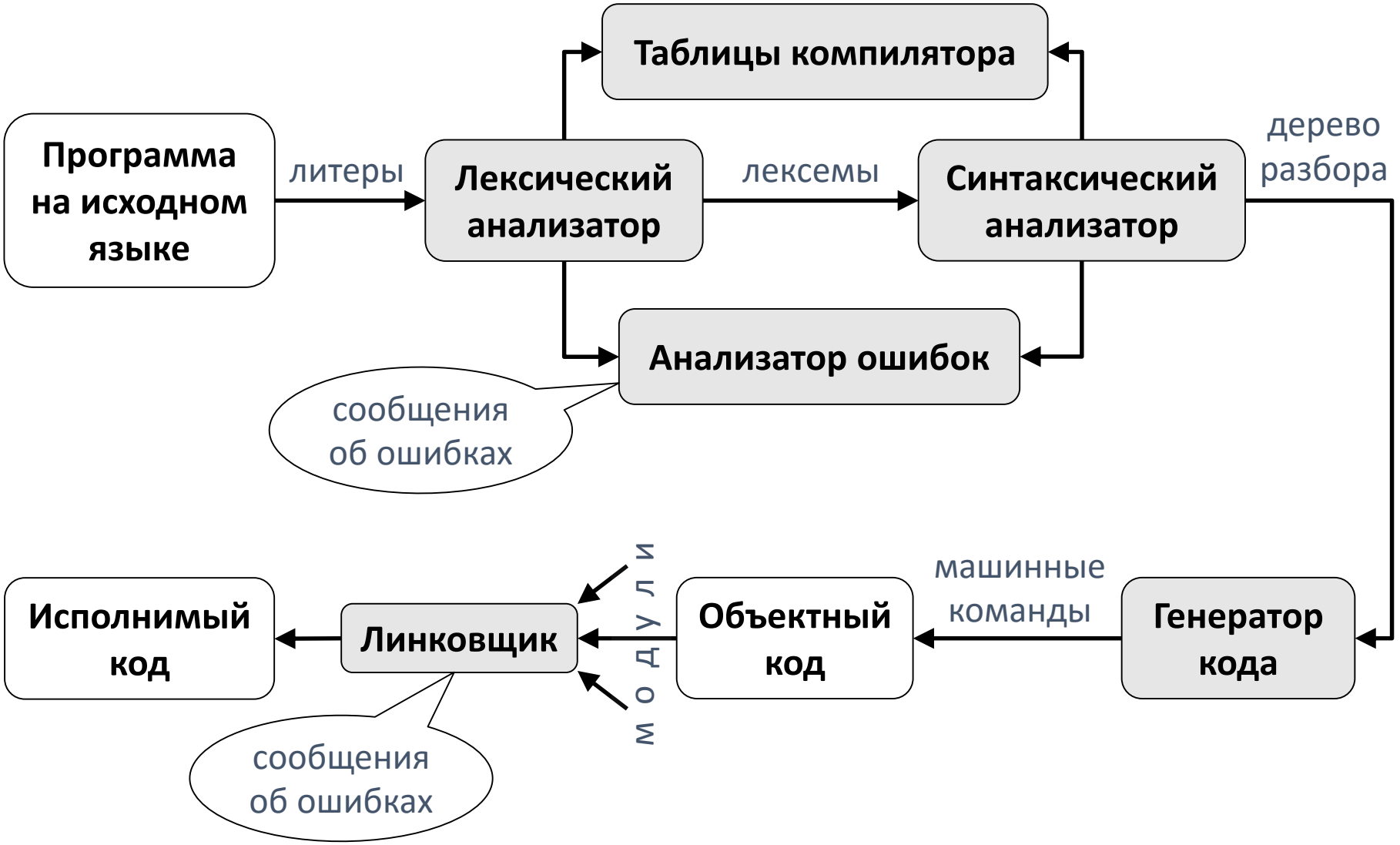
По поколениям (Generation Language, GL) –

- **1 GL** (время появления – 40-50 гг.). Языки машинных команд. Первый ассемблер (названия команд в символическом виде, десятичный и шестнадцатеричный формат чисел).
- **2GL** (конец 50-х – начало 60-х гг.). Символический ассемблер (включал понятие переменной, требовался компилятор).
- **3GL** (60-е годы). Универсальные языки высокого уровня (FORTRAN, COBOL, ALGOL, Pascal, C и др.) Современные системы программирования (Delphi, Visual BASIC, Java Development Kit и др.) включают поддержку объектно-ориентированной технологии и другие инструменты.
- **4 GL** (начало 70-х г.). Непроцедурные языки, ориентированные на конкретную область применения, в частности, на работу с базами данных (SQL, Structured Query Language и др.). Команды языков 4GL близки к обычному языку.
- **5 GL** (наст. время). По одной из точек зрения, декларативные языки.

ТРАНСЛЯТОРЫ

Трансляция – перевод программы, написанной на языке программирования, в последовательность машинных команд.

СХЕМА КОМПИЛЯТОРА

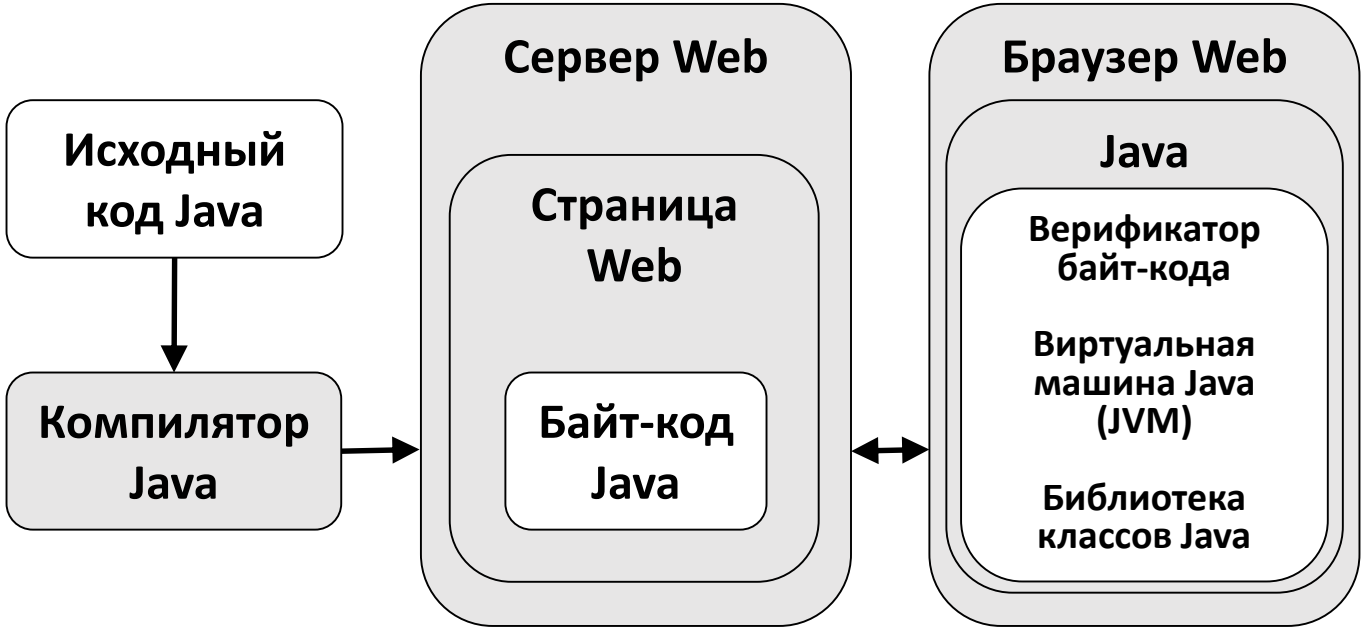


Лексема – минимальная значимая единица текста программы (идентификатор, специальный символ, константа, зарезервированное слово и др.).

ЯЗЫКИ ВЕБ-ПРОГРАММИРОВАНИЯ

Языки веб-программирования в основном предназначены для работы с интернет-технологиями. Делятся на две группы: клиентские (программы на клиентских языках обрабатываются на стороне пользователя, как правило их выполняет браузер – JavaScript, VBScript, Java и др.) и серверные (вызванная из браузера страница обрабатывается на сервере, то есть выполняются все программы, связанные со страницей, и потом возвращается к клиенту по сети в виде файла – PHP, Perl, Python, Ruby, любой .NET язык, Java и др.).

Выполнение Java-апплета



Java-апплет – прикладная программа, чаще всего написанная на языке программирования Java в форме байт-кода.

ЯЗЫКИ ВЕБ-ПРОГРАММИРОВАНИЯ

	Исполнение на сервере	*Исполнение у клиента*
Программа размещается внутри HTML-кода	Активные серверные страницы: **PHP**, **ASP**, **JSP**, …	Скрипты: **JavaScript**, **VB Script**, …
Программа размещается отдельно	CGI и сервлеты: **Perl**, **C**, **Pascal**, **Java**, …	Апплеты: **Java**, **Oberon-2**, …

Идея **WWW** (World Wide Web – всемирная паутина) базируется на языке описания гипертекста HTML (Hypertext Markup Language) и протоколе передачи гипертекста (HTTP). HTML следует считать языком внешнего представления Web-страниц (языком разметки), а не языком программирования.

Глава 4. АЛГОРИТМЫ И СТРУКТУРНОЕ ПРОГРАММИРОВАНИЕ

- Понятие и свойства алгоритма
- Язык блок-схем
- Структурное программирование
- Основные структуры алгоритмов
- Язык проектирования программ (псевдокод)
- Рекурсивный алгоритм
- Алгоритмы поиска
- Алгоритмы сортировки
- Объектно-ориентированное программирование (принципы, внутренность объекта, иерархия классов)

ПОНЯТИЕ И СВОЙСТВА АЛГОРИТМА

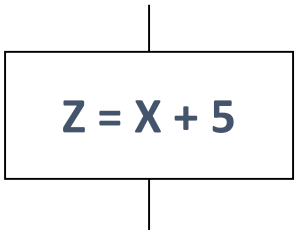
Алгоритм - формальное описание последовательности действий, которое необходимо выполнить для решения задачи.

Основные свойства алгоритма

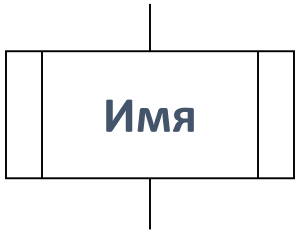
1. **Дискретность.** Алгоритм представляет процесс решения задачи как последовательность выполнения шагов-этапов. Для выполнения каждого этапа требуется определенное время, т.е. преобразование исходных данных в результат происходит дискретно во времени.
2. **Определенность (детерминированность).** Каждое правило алгоритма должно быть четким и однозначным. Отсюда выполнение алгоритма носит механический характер.
3. **Результативность (финитность, конечность).** Алгоритм должен приводить к решению задачи за конечное число шагов.
4. **Массовость.** Алгоритм решения задачи разрабатывается в общем виде, т.е. он должен быть применим для некоторого класса задач, различающихся исходными данными (область применимости алгоритма).

Программа – окончательный вариант алгоритма, ориентированный на исполнителя (вычислительную машину).

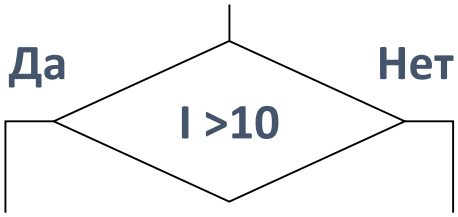
ЯЗЫК БЛОК-СХЕМ



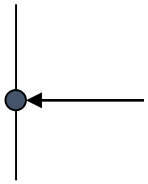
Процесс – обработка данных (вычисление, пересылка и т.п.)



Предопределенный процесс – подпрограмма



Решение – проверка условия

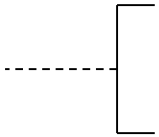


Соединительные линии и их объединение

Данные – ввод-вывод данных

Соединитель – точка связи

Терминатор – начало и конец вычислительного процесса



Комментарий

СТРУКТУРНОЕ ПРОГРАММИРОВАНИЕ

Основные (базовые) структуры алгоритмов – это ограниченный набор стандартных способов соединения отдельных блоков или структур блоков для выполнения типичных последовательностей действий.

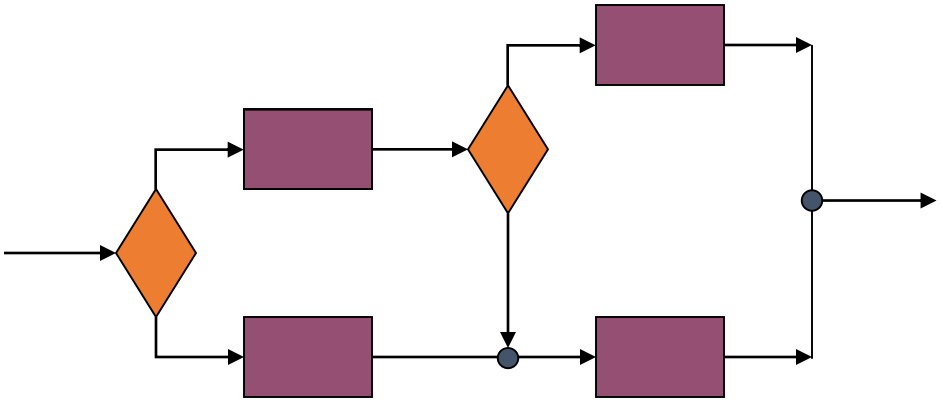
Доказано, что программу для любой простой логической задачи можно составить из структур следование, разветвление и повторение (цикл).

Технология **структурного программирования** – подход к программированию, в котором для передачи управления в программе используется три базовых структуры (конструкции): следование, разветвление, цикл. Эта технология разработки сложных программ рекомендует разбивать (декомпозировать) программу на подпрограммы (процедуры), решающие отдельные подзадачи, т.е. базируется на **процедурной декомпозиции**.

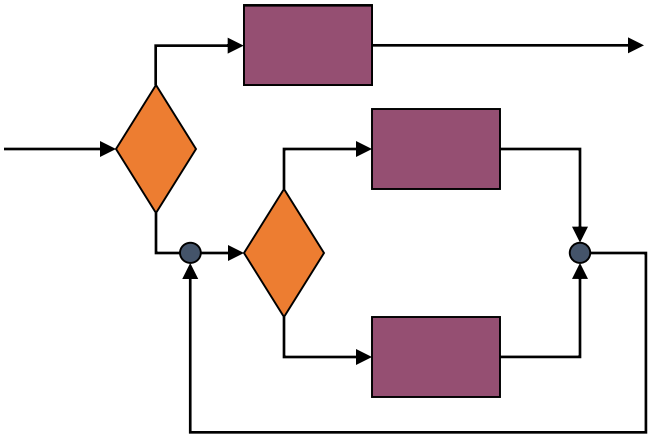
Простая программа – алгоритм, для которого:

- существует единственный вход и единственный выход;
- для каждого элемента алгоритма существует путь от входа к выходу через этот элемент (т.е. алгоритм не содержит бесконечных циклов и не содержит бесполезных (недостижимых) фрагментов).

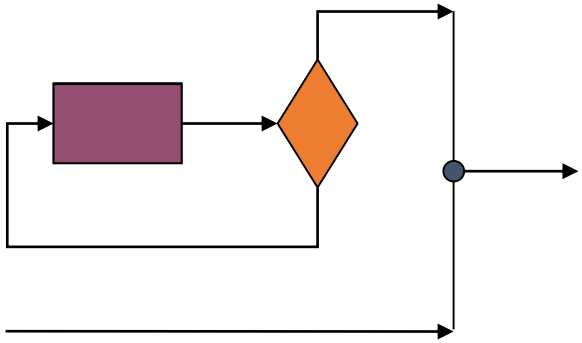
ПРИМЕРЫ ПРОСТОЙ И НЕПРОСТЫХ ПРОГРАММ



Простая программа



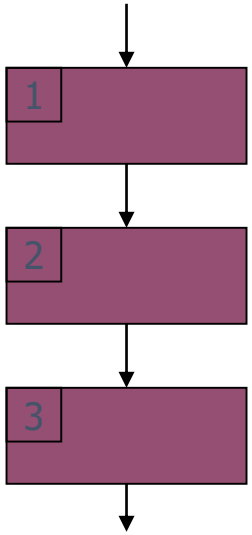
Бесконечный цикл



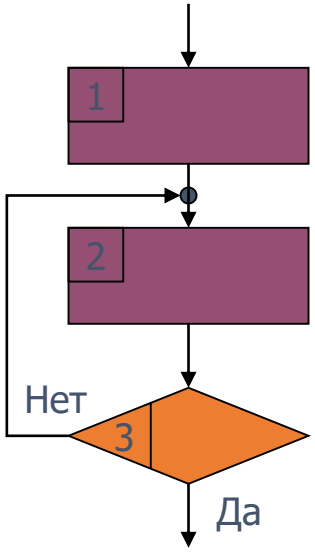
Недостижимый фрагмент

ОСНОВНЫЕ СТРУКТУРЫ АЛГОРИТМОВ И ИХ ПРОИЗВОДНЫЕ

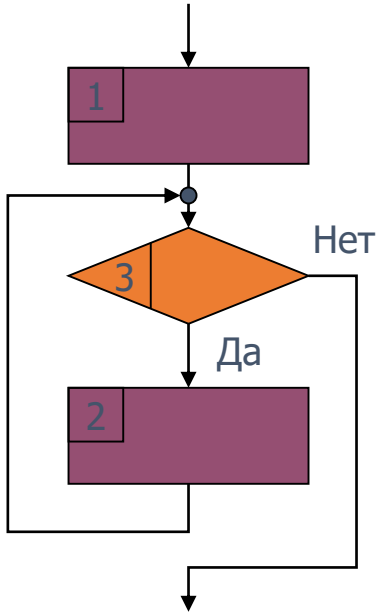
Следование – последовательное выполнение действий (блоков).



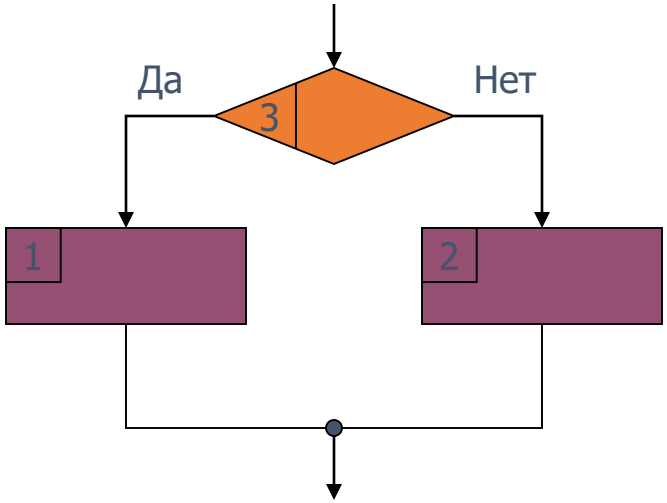
Цикл "До" (с постусловием) – тело цикла (блок 2) выполняется до тех пор, пока условие (блок 3) не станет истинным.



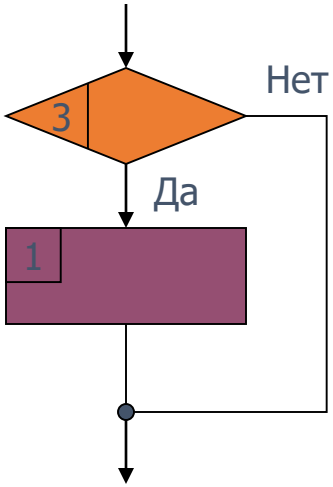
Цикл «Пока» (с предусловием) – пока не будет нарушено условие (блок 3), осуществляется повторение тела цикла (блок 2).



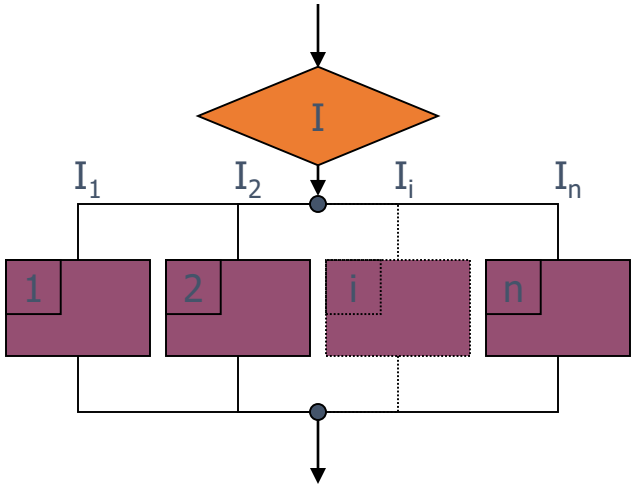
ОСНОВНЫЕ СТРУКТУРЫ АЛГОРИТМОВ И ИХ ПРОИЗВОДНЫЕ



Разветвление – применяется, когда в зависимости от условия требуется выполнить либо одно действие, либо другое.



Обход – частный случай разветвления, когда одна ветвь не содержит ни каких действий.



Множественный выбор – обобщение разветвления, когда в зависимости от значения переменной I выполняется одно из нескольких действий.

ПСЕВОДОКОД

Псевдокод (язык проектирования программ PDL, Process Design Language) – способ описания программы на этапе проектирования. Состоит из внешнего синтаксиса и внутреннего синтаксиса.

Внешний синтаксис – заданный набор операторов, построенных по образцу языков программирования и описывающий логику программы. Внешний синтаксис соответствует основным структурам алгоритмов. К внешнему синтаксису также относятся процедуры и модули. **Процедура** – это хранимая в памяти машины подпрограмма, которая может вызываться для выполнения из различных мест основной программы, либо из других процедур. Она вызывается и выполняется до завершения без сохранения внутренних данных. **Модуль** – это несколько процедур, организованных в систему для удобства работы пользователя. Модуль имеет доступ к общим данным, которые сохраняются между последовательными вызовами модуля.

Внутренний синтаксис – общий, обычно специально не определяемый синтаксис, пригодный для описания задач в данной области. Практически любое предложение, написанное на естественном языке, либо на специализированном языке (например, математические формулы) может быть использовано.

ОПЕРАТОРЫ ВНЕШНЕГО СИНТАКСИСА ПСЕВДОКОДА

Следование. Записываются последовательно операции одна под другой. Для отделения части последовательности операторов используются операторы do…end-do.

```
первая операция
вторая операция
do
    третья операция
end-do
```

Индексная последовательность (цикл по счетчику). Цикл с заранее определенным числом шагов (среднее между обычными последовательностью и классическим циклом).

```
for
    индексный список
do
    do-часть
end-do
```

Цикл-До. Операции структуры, включая модификацию until-теста, выполняются один или более раз до тех пор, пока until-тест не примет значение истина.

```
do
    do-часть
until
    until-тест
end-do
```

ОПЕРАТОРЫ ВНЕШНЕГО СИНТАКСИСА ПСЕВДОКОДА

Цикл-Пока. Do-часть выполняется, пока while-тест имеет значение истина. Do-часть модифицирует условие while-теста для того, чтобы окончить вычисления.

```
while
    while-тест
do
    do-часть
end-do
```

```
if
    if-тест
then
    then-часть
else
    else-часть
end-if
```

Разветвление.
Если…то…иначе.

```
case
список 1
    case-часть 1
список 2
    case-часть 2
…
список n
    case-часть n
else
    else-часть
end-case
```

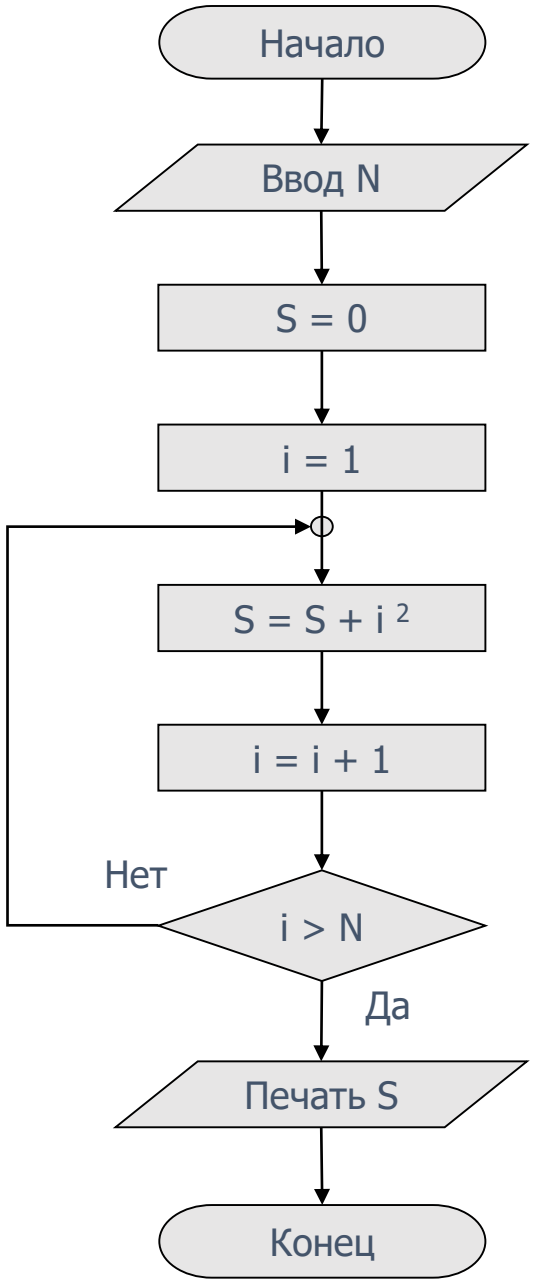
Множественный выбор.

ПРИМЕР АЛГОРИТМА

Вычисление суммы квадратов первых N целых чисел с использованием псевдокода и языка блок-схем

$$S = \sum_{i=1}^{N} i^2 = 1^2 + 2^2 + \cdots + N^2$$

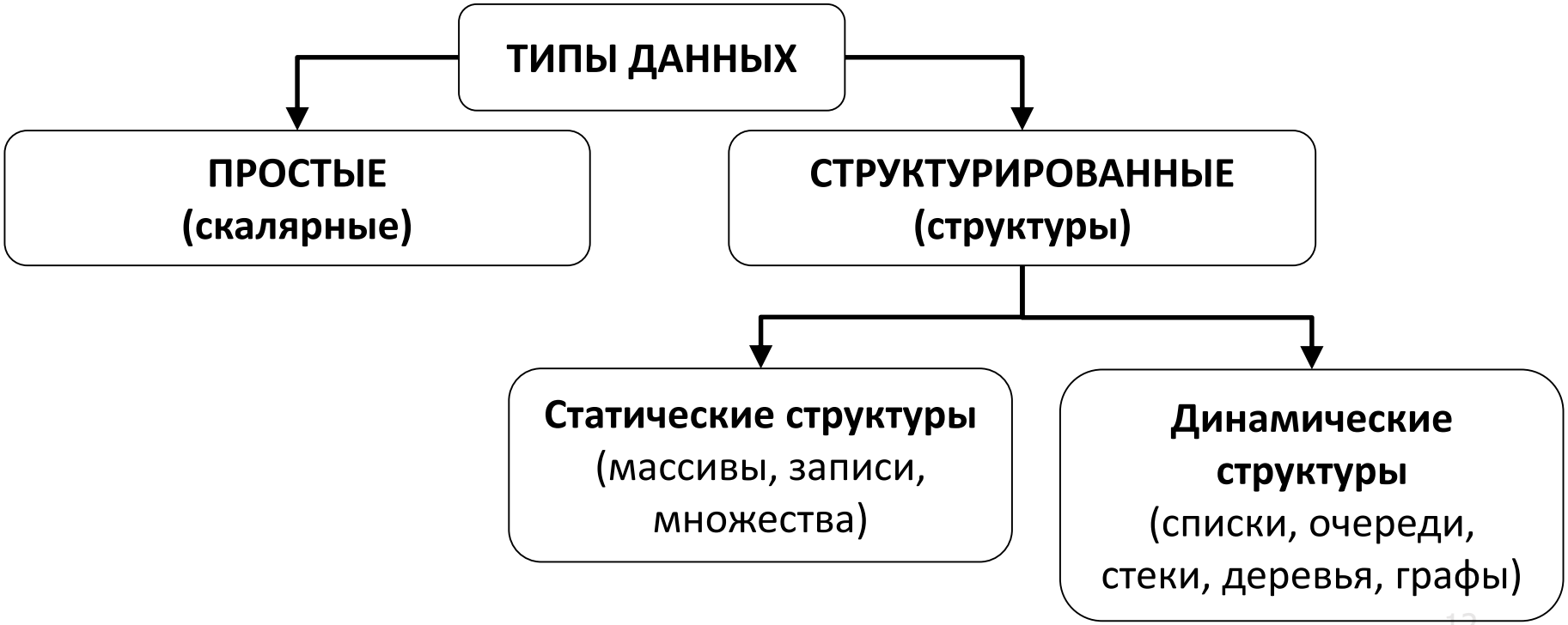
```
Ввести Число слагаемых
Сумма = 0
Номер = 1
do
    Сумма = Сумма + Номер²
    Увеличить Номер на единицу
until
    Номер > Число слагаемых
end-do
Напечатать Сумму
```



ТИПЫ ДАННЫХ

Помимо совокупности управляющих структур, важным аспектом структурного программирования является организация данных, участвующих в решении проблемы. Структура программы и строение данных неразрывно связаны.
"Программа – это конкретное, основанное на некотором реальном представлении и строении данных, воплощение абстрактного алгоритма" (Н. Вирт).

Тип данных – это характеристика данных, определяющая множество значений и операций, которые могут быть применены к этим данным, а также правила их выполнения.

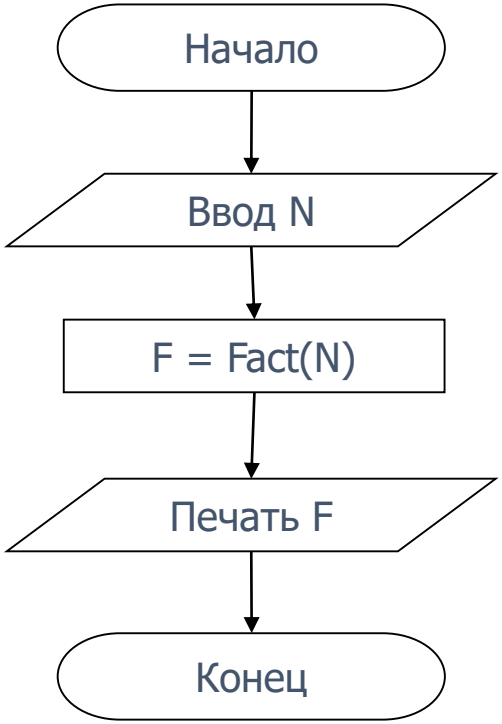


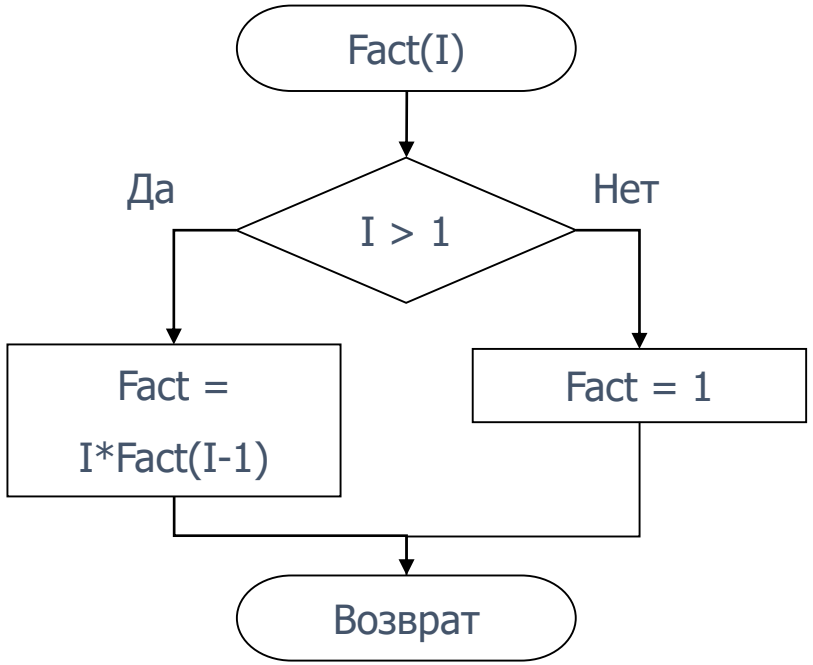
РЕКУРСИЯ

Задача имеет рекурсивное решение, если его возможно сформулировать как известное преобразование другого, более простого решения той же задачи, хотя само решение (более простое) может быть неизвестно. Многократное повторение такого преобразования должно сходится к базисному утверждению.

Функция F является **рекурсивной**, если
1. F(N) = G(N,F(N-1)), где G – известная функция;
2. F(1) – известно (базисное утверждение).

Вычисление факториала N с использованием рекурсивной функции





ПОИСК

Поиск - обнаружение нужного элемента в некотором наборе (структуре) данных. Элемент данных – это запись, состоящая из ключа (целое положительное число) и тела, содержащего некоторую информацию. Задача поиска состоит в том, чтобы обнаружить запись с нужным ключом.

Линейный поиск.

Элементы проверяются последовательно, по одному, до тех пор, пока нужный элемент не будет найден. Для массива из N элементов требуется, в среднем, (N+1)/2 сравнений (выч. сложность O(N)). Легко программируется, подходит для коротких массивов.

Двоичный (бинарный) поиск.

Применим, если массив заранее отсортирован (по возрастанию ключей). Ключ поиска сравнивается с ключом среднего элемента в массиве. Если значение ключа поиска больше, то та же самая операция повторяется для второй половины массива, если меньше – то для первой. Операция повторяется до нахождения нужного элемента. На каждом шаге диапазон элементов в поиске уменьшается вдвое. Требуется, в среднем, $(\log_2 N+1)/2$ сравнений (выч. сложность $O(\log_2 N)$). Применяется для поиска в больших массивах.

Функция G(x) – функция-оценка для функции F(x), F(x) = O(G(x)), если

$$\lim_{x\to\infty}(F(x)/G(x)) = const(x) \neq 0$$

СОРТИРОВКА

Сортировка (упорядочение) – переразмещение элементов данных в возрастающем или убывающем порядке. При выборе метода сортировки необходимо учитывать число сортируемых элементов (N) и до какой степени элементы уже отсортированы.

Критерии оценки метода сортировки:

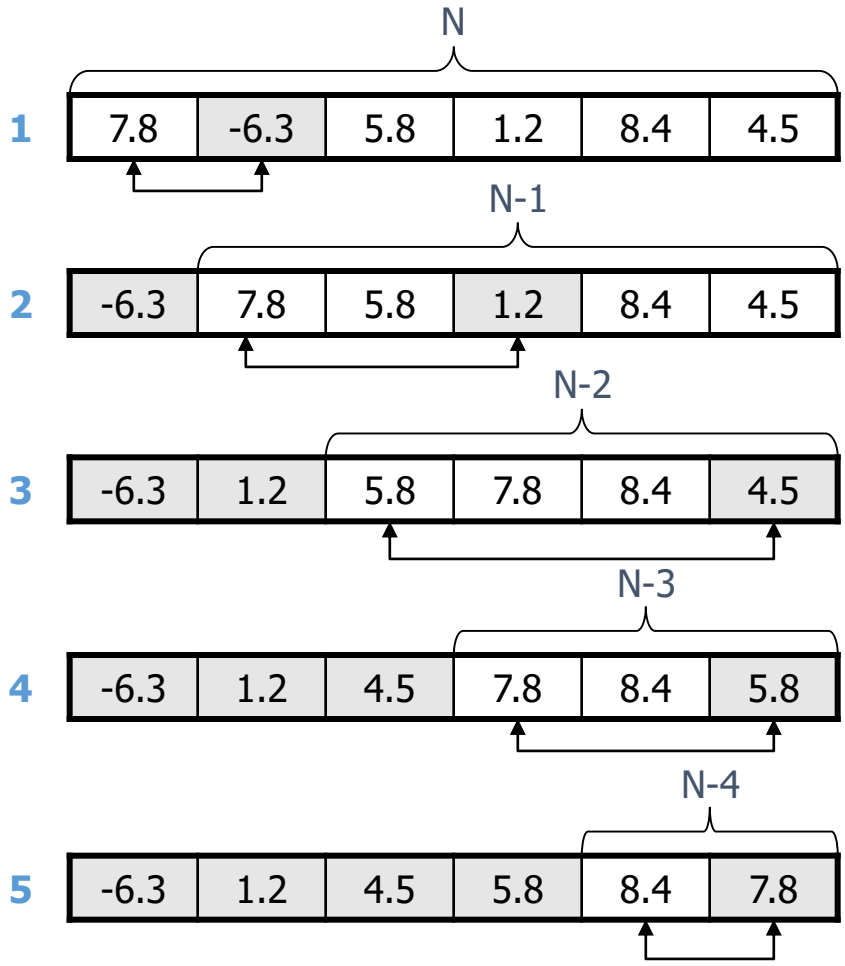
- количество необходимых операций сравнения в зависимости от числа элементов N, вычислительная сложность алгоритма характеризуется с помощью O-функции, аргументом которой может быть другая функция от N;
- эффективность использования памяти

$$f = \frac{S(N)}{S(N) + \Delta S(N)},$$

где S(N) – объем памяти, занимаемый элементами данных до сортировки, ΔS(N) – объем дополнительной памяти, требуемой в процессе сортировки.

СОРТИРОВКА ВЫБОРКОЙ

Принцип: *Из массива выбирается наименьший элемент и меняется местами с первым элементом массива, затем выбирается наименьший элемент из оставшихся и меняется местами со вторым элементом массива и т.д.*

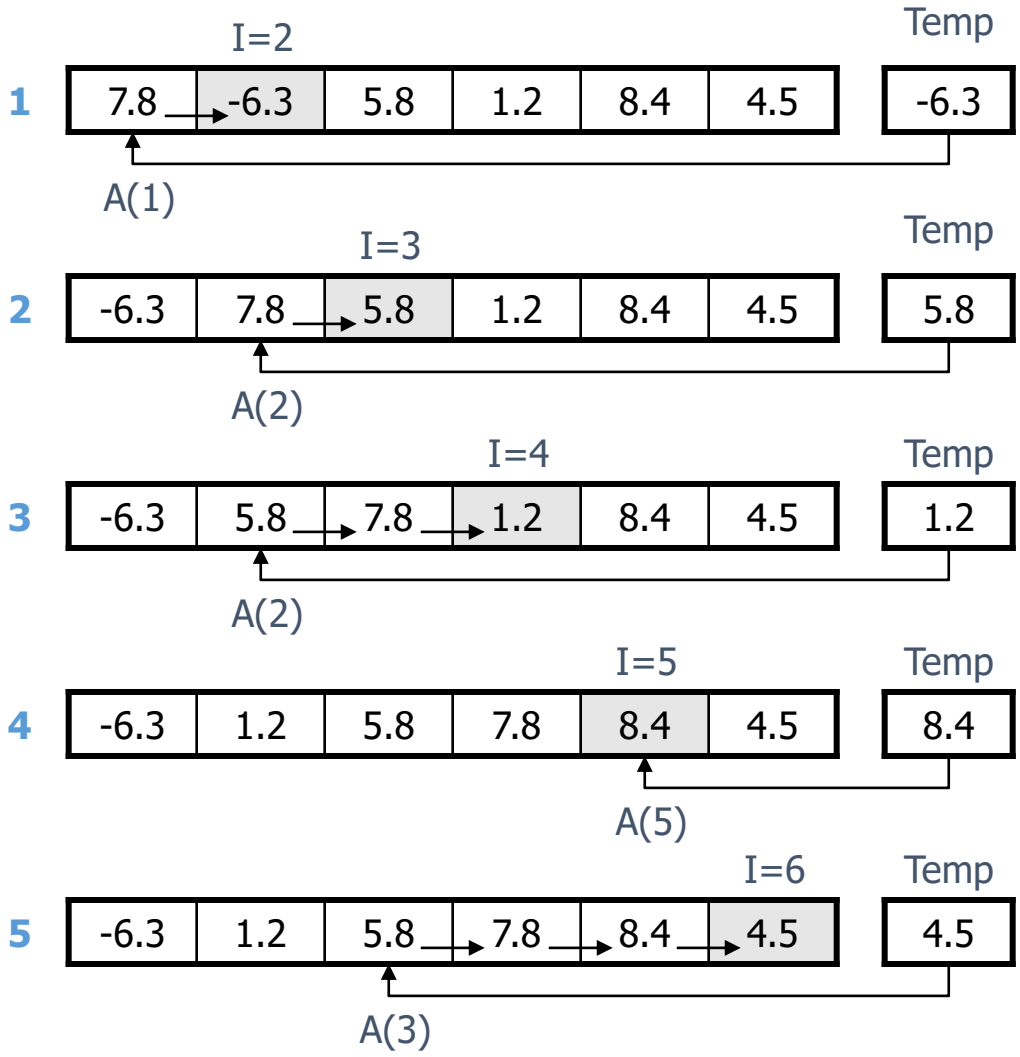



```
Ввести массив A(1..N)
for J = 1,N-1,1
do
    Мин.Эл. = A(J)
    Индекс Мин.Эл. = J
    for I = J+1,N,1
    do
        if A(I) < Мин.Эл.
        then
            Мин.Эл. = A(I)
            Индекс Мин.Эл. = I
        end-if
    end-do
    A(Индекс Мин.Эл.) = A(J)
    A(J) = Мин.Эл.
end-do
Вывести A(1..N)
```

Требуется, в среднем, $N(N+1)/2$ сравнений (выч. сложность $O(N^2)$, не зависит от начальной упорядоченности). Дополнительная память не нужна.

СОРТИРОВКА ВКЛЮЧЕНИЕМ

Принцип: *Элементы выбираются по очереди и помещаются в нужное место отсортированной части массива.*

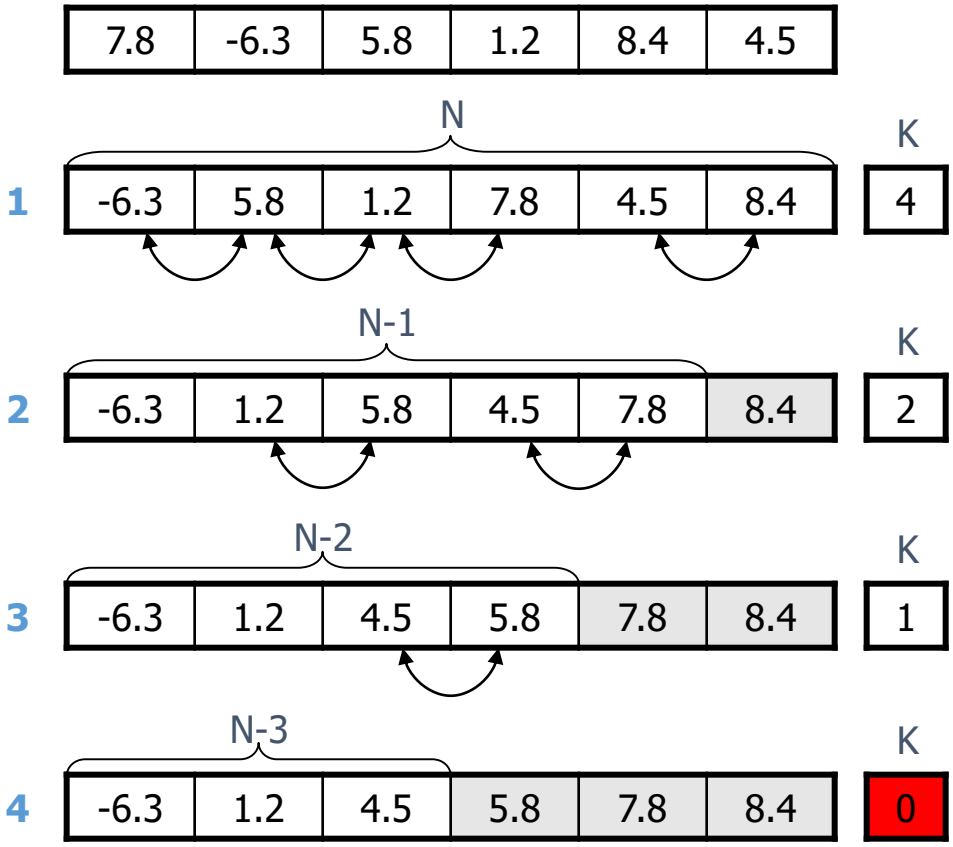



```
Ввести массив A(1..N)
for I = 2,N,1
do
    Temp = A(I)
    A(0) = Temp
    J = I-1
    while A(J) > Temp
    do
        A(J+1) = A(J)
        J = J-1
    end-do
    A(J+1) = Temp
end-do
Вывести A(1..N)
```

Требуется, в среднем, (N-1)(N/2+1)/2 сравнений (выч. сложность $O(N^2)$). Скорость данного метода зависит от начальной упорядоченности массива. Не требует дополнительной памяти.

СОРТИРОВКА ОБМЕНАМИ (ПУЗЫРЬКОВАЯ)

Принцип: *Выбираются два элемента, и если друг по отношению к другу они не находятся нужном порядке, то меняются местами. Процесс продолжается пока никакие два элемента не нужно менять местами.*



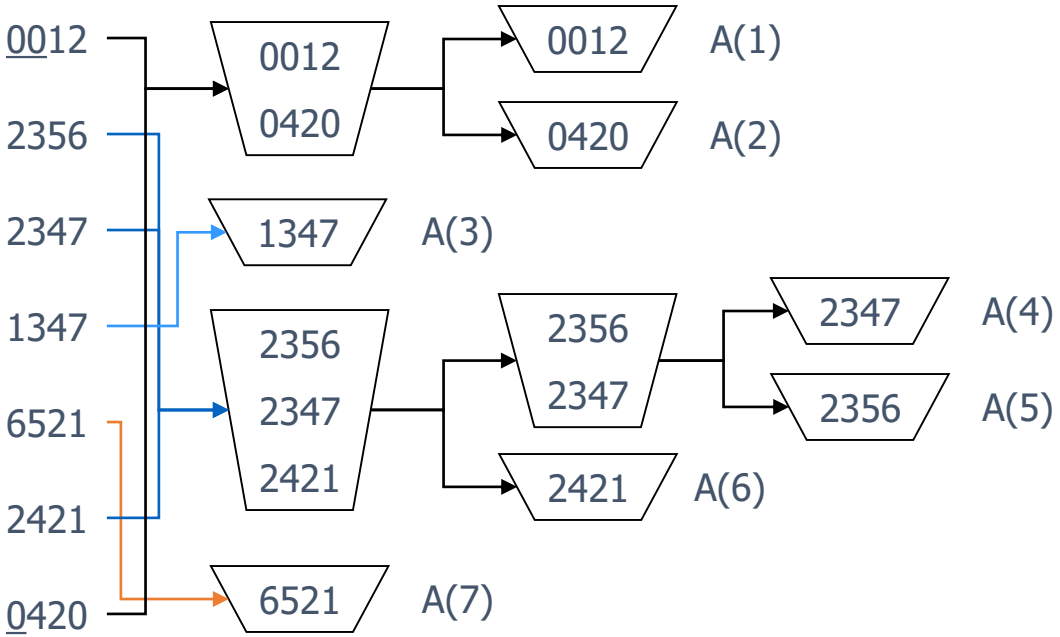

```
Ввести массив A(1..N)
K = 1, I = 1
while K <> 0
do
   K = 0
   for J = 1,N-I,1
   do
      if A(J) > A(J+1)
      then
         T = A(J),
         A(J) = A(J+1),
         A(J+1) = T, K = K+1
      end-if
   end-do
   I = I +1
end-do
Вывести A(1..N)
```

Вычислительная сложность метода сильно зависит от исходного расположения элементов. Минимальное число сравнений – N-1 в полностью отсортирован-ном массиве, максимальное – $(N^2-N)/2$ при начальной сортировке в обратном порядке. Средняя вычислительная сложность $O(N^2)$. Доп. память не требуется.

СОРТИРОВКА РАСПРЕДЕЛЕНИЕМ (МЕТОД КОРЗИН)

Принцип: *Элементы массива рассматриваются как совокупность цифр (символов), первый шаг - сортировка по значению старшей цифры, затем полученные подмножества (группы) сортируются по значению следующей цифры и т.д.*

Каждый элемент массива A (1..N) – совокупность цифр $C_1C_2C_3…C_m$, где m – количество цифр максимального элемента (если какой-то элемент содержит меньше цифр, то он слева дополняется нулями).



Средняя вычислительная сложность O(N $\log_2$N) и лучше, если m (число цифр) мало. Требуется дополнительный массив размером N, и еще массив размером 10, в котором подсчитывается число элементов с выделяемой цифрой 0,1,…9.

БЫСТРАЯ СОРТИРОВКА

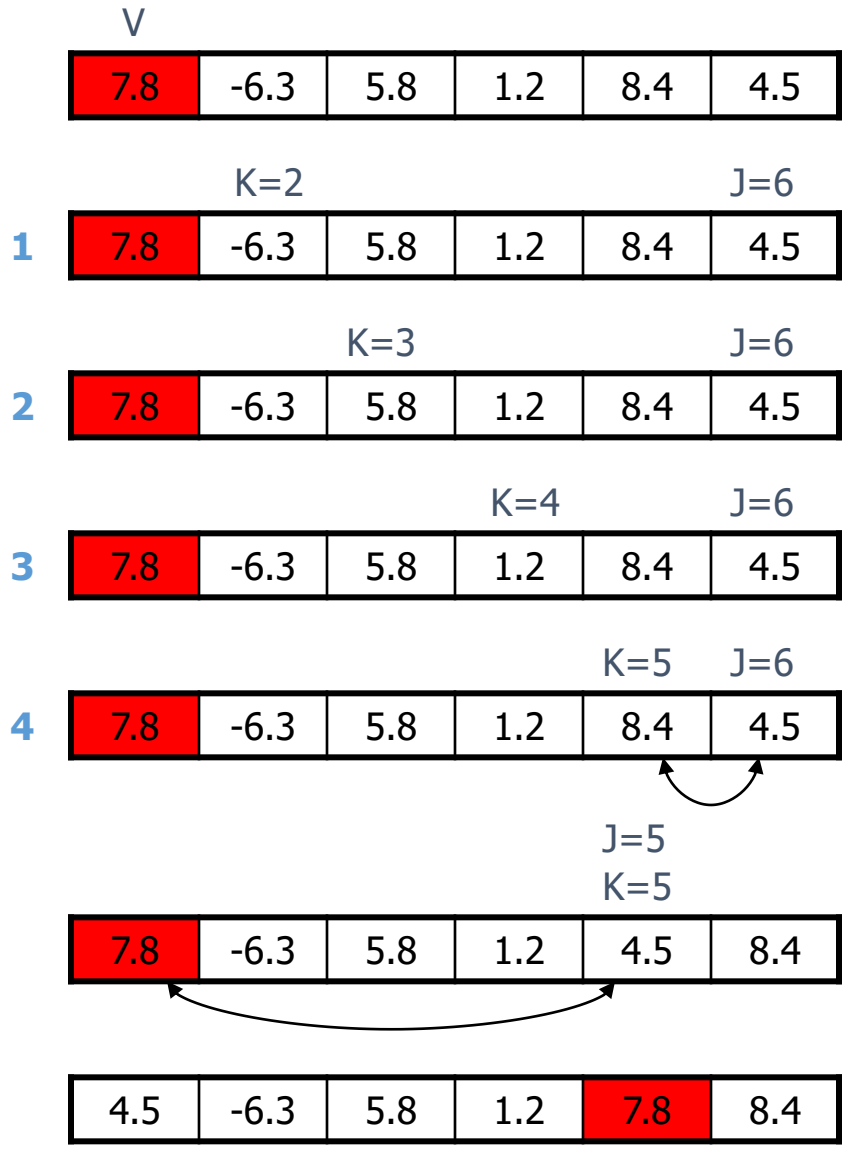
Принцип: *Определенным образом выделяется пороговый элемент. На первом этапе элементы обмениваются так, что новый массив оказывается разделенным пороговым элементом на две части: в левой все элементы меньше порогового, а в правой – больше или равны пороговому. Затем подобный способ используется для разделения каждого из новых массивов на две части и т.д.*

Алгоритм процедуры разбиения массива A(1..N) пороговым элементом, находящимся вначале на месте A(1).

```
V = A(1), K = 2, J = N
while K < J
do
   if A(K) < V
   then
      K = K + 1
   else
      if A(J) < V
      then
         Temp = A(K), A(K) = A(J)
         A(J) = Temp
      end-if
      J = J - 1
   end-if
end-do
if A(K) >= V
then
   K = K - 1
end-if
Temp=A(1),A(1)=A(K),A(K)=Temp
```

БЫСТРАЯ СОРТИРОВКА

Средняя вычислительная сложность – $O(N \log_2 N)$. Важное значение имеет выбор значения порогового элемента. В частности, если исходный массив близок к отсортированному, то при выборе пороговым элементом первого элемента (как в примере) вычислительная сложность алгоритма будет $O(N^2)$. Желательно, чтобы пороговый элемент в конечном итоге разделил массив приблизительно на две равные части.



СОРТИРОВКА СЛИЯНИЕМ

Принцип: *Два отсортированных массива соединяются в один массив таким образом, чтобы и он стал отсортированным.*

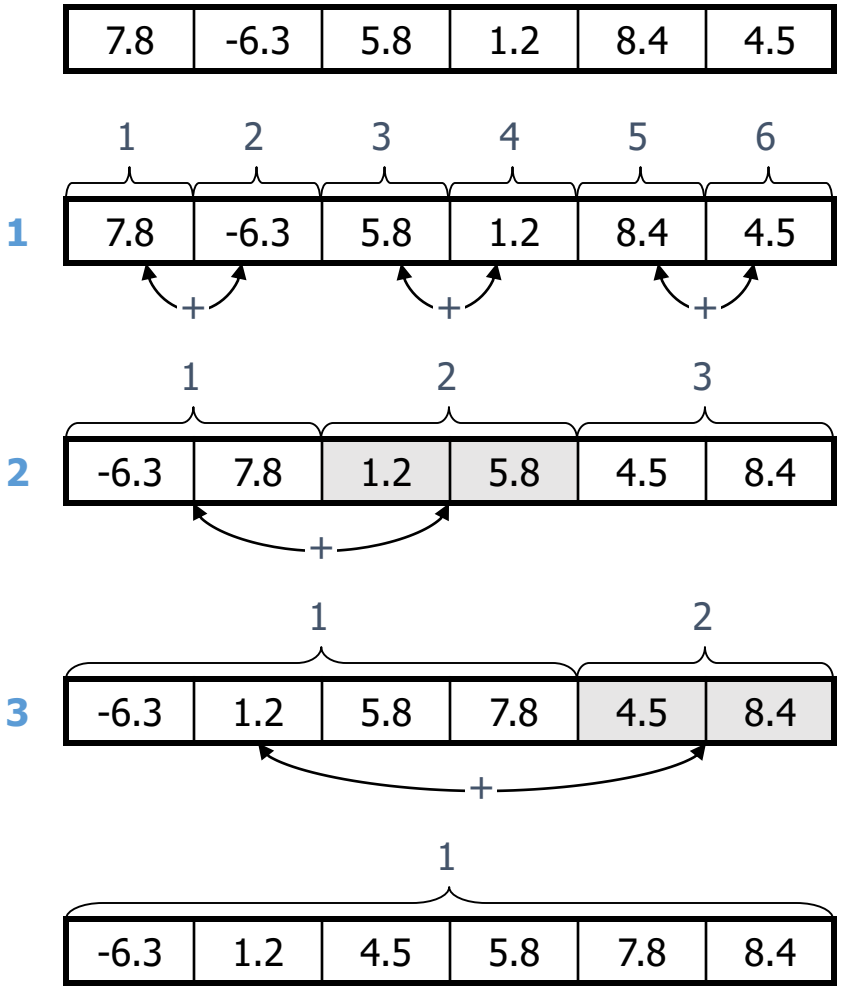
Алгоритм слияния отсортированных массивов B(1..M) и C(1..L) в массив A(1..M+L) заключается в следующем. В качестве A(1) выбираем наименьший из B(1) и C(1). Если это B(1), то в качестве A(2) – наименьший из B(2) и C(1) и т.д.

```
Ввести массивы B(1..M),C(1..L)
I = 1, J = 1
for K = 1,M+L,1
do
   if I > M
   then
      A(K) = C(J), J = J + 1
   else
      if J > L
      then
         A(K) = B(I), I = I + 1
      else
         if B(I) < C(J)
         then
            A(K) = B(I),I = I + 1
         else
            A(K) = C(J),J = J + 1
         end-if
      end-if
   end-if
end-do
Вывести A(1..K)
```

СОРТИРОВКА СЛИЯНИЕМ

Если имеется один неотсортированный массив A(1..N), то его можно рассматривать как совокупность N отсортированных массивов, каждый из которых состоит из одного элемента. Первый шаг – слияние массивов попарно, затем объединение пар в четверки и т.д.

Средняя вычислительная сложность алгоритма – $O(N \log_2 N)$. Требуется дополнительный массив, содержащий N элементов.



ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Структурное программирование приспособлено для описания действий, а объектно-ориентированное – для описания моделей.

Абстракция – это способность языка программирования отображать объекты внешнего мира в форме абстрактных структур в соответствии с решаемой задачей. Абстрактные структуры, при помощи которых реализуется этот принцип называются классами. **Класс** – это структура объединяющая переменные (поля), описывающие состояние объекта, и процедуры и функции (методы), описывающие его поведение. Классы представляют собой абстрактные описания структур данных, но сами данные они не содержат. Данные появляются тогда, когда по описаниям классов в памяти выделяется необходимое пространство и в нем создаются экземпляры класса (объекты).

Инкапсуляция – свойство языка программирования, позволяющее объединить данные и действия в единый объект и скрыть реализацию объекта от пользователя. Идея инкапсуляции, в частности, реализована в модуле при его разделении на секции интерфейса и реализации.

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Значительно упростить понимание сложных задач удается за счет введения иерархии классов. Связь между классами разных уровней достигается за счет наследования. **Наследование** – это расширение свойств наследника за счет принятия всех свойств предка.

Полиморфизм состоит в возможности переопределять методы класса-родителя. Он означает общий род действий, которые могут быть выполнены разными специфическими путями в зависимости от того, какие объекты выполняют эти действия. Неразрывно с полиморфизмом связано понятие **динамического связывания**, состоящее в возможности отложить подстановку реального адреса некоторой подпрограммы-метода с этапа компиляции (раннее связывание) до момента выполнения программы (позднее связывание).

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Объектный стиль программирования связан с воздействием на объекты (иначе, с передачей объекту сообщений). Проектирование базируется на том условии, что никакая подсистема данного уровня не должна зависеть от устройства любой другой подсистемы этого уровня (взаимодействуют, но не зависят).

Сообщение, посланное объекту, активизирует совокупность операций над объектом – метод. Задавая структуру обмена сообщениями между объектами, программист получает совокупность операций, которые и составляют программу.

Основные типы операций над объектами

Название	Содержание
Конструктор	Создает и инициализирует объект.
Деструктор	Освобождает объект, т.е. разрушает его.
Модификатор	Изменяет состояние объекта.
Селектор	Считывает состояние объекта без изменения этого состояния.
Итератор	Организует доступ ко всем частям объекта в строго определенной последовательности.

ВНУТРЕННОСТЬ ОБЪЕКТА

ОкружностьА

Поля

- **Позиция 15,20**
- **Размер 5**

Методы

- **Форма**
  - **Маркер** ПероВверх
  - **Маркер** СдвигК : Позиция
  - **Маркер** СдвигНа : Размер
  - **Маркер** ПероВниз
  - **Маркер** ЧертитьДугу : Размер,360
- **Высветить**
  - **Маркер** Черный
  - **Себе** Форма
- **Стереть**
  - **Маркер** Белый
  - **Себе** Форма

Маркер

…

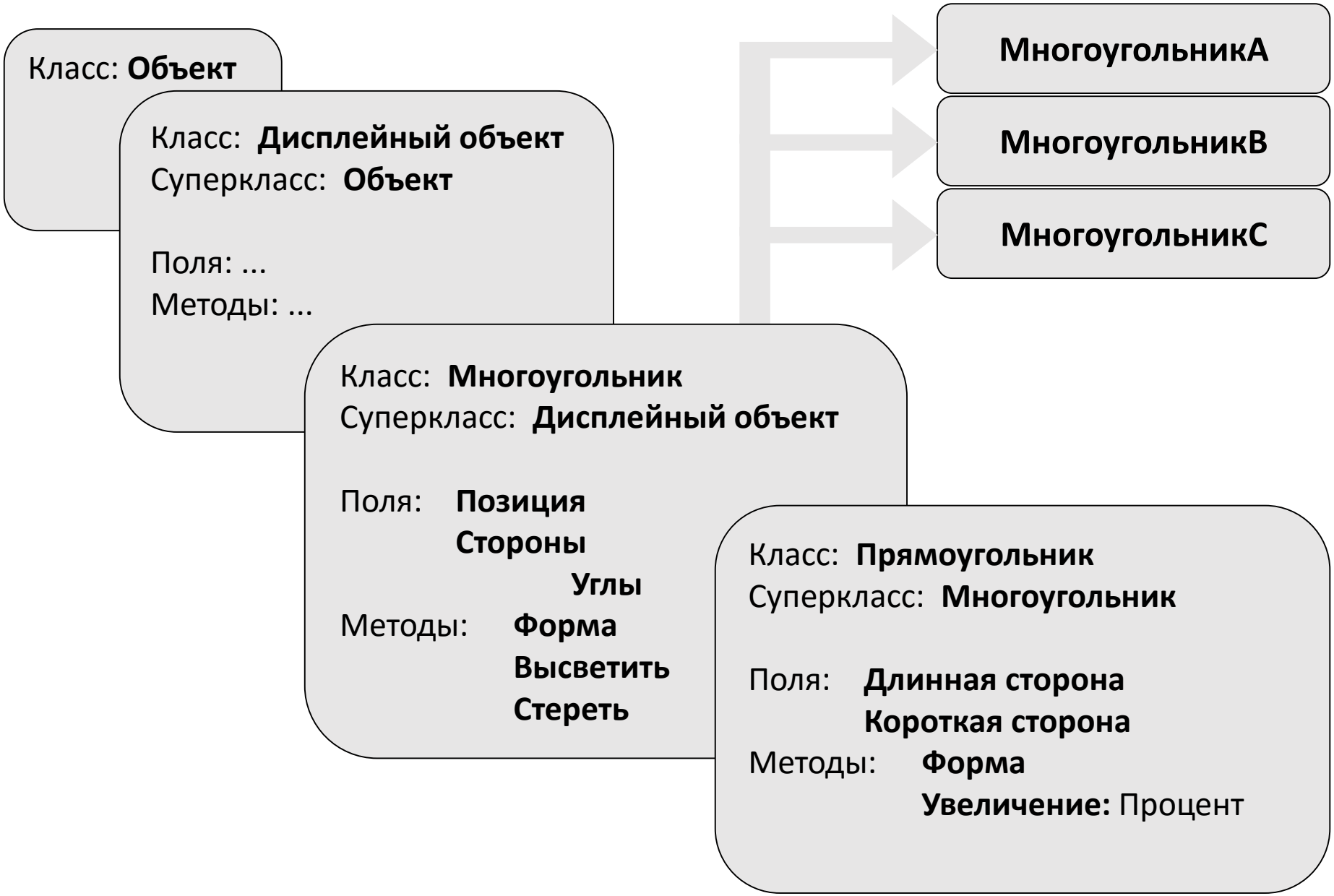
Методы

- **ПероВверх**
- **ПероВниз**
- **Черный**
- **Белый**
- **СдвигК :** X,Y
- **СдвигНа :** Z
- **ЧертитьДугу :** R,φ
- **…**

Сообщения (операции) объекту ОкружностьА:

```
ОкружностьА.Высветить
ОкружностьА.Стереть
```

ИЕРАРХИЯ КЛАССОВ



Глава 5. БАЗОВЫЕ ЭЛЕМЕНТЫ ЯЗЫКА ПРОГРАММИРОВАНИЯ (Object Pascal, Turbo Delphi)

- Синтаксис и семантика языка программирования
- Методы описания синтаксиса
- Идентификаторы
- Константы
- Адресация в оперативной памяти
- Переменные
- Типы
- Выражения и операции
- Скалярные типы данных (логический, целые, символьный, перечисляемый, тип-диапазон, вещественные)
- Арифметические и логические операции, операции отношения
- Приоритет операций
- Структура программы

СИНТАКСИС И СЕМАНТИКА

Синтаксис языка программирования – это форма, а **семантика** – смысл его выражений, операторов и программных единиц.

Формальное описание синтаксиса языка строится с использованием терминалов и нетерминалов. Конструкции языка, для которых требуются определения, называются нетерминальными символами или просто **нетерминалами** (например, <оператор>, <число>, <условный оператор> и т. д.). Число нетерминалов равно числу правил языка. Собственные символы (символы алфавита) и конструкции языка, для которых не требуется специальное описание, называются терминальными символами или **терминалами**.

Алфавит языка Delphi Pascal включает:
- прописные и строчные латинские буквы от **a** до **z** и от **A** до **Z** , а также знак подчеркивания
- арабские цифры от **0** до **9**
- специальные символы **+ - * / = , ‘ . : ; < > [ ] ( ) { } ^ @ $ # &** и пробел
- составные символы **:= <> .. <= >= (* *) (. .) //**

Терминалами также являются зарезервированные слова (**begin end array const if then else** и т.д.), в состав которых входят стандартные директивы (**absolute assembler external far forward near private virtual** и др.).

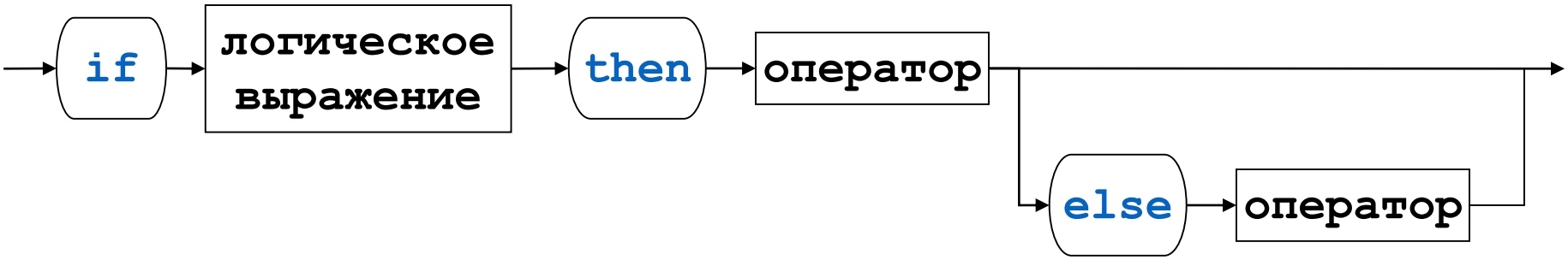
МЕТОДЫ ФОРМАЛЬНОГО ОПИСАНИЯ СИНТАКСИСА ЯЗЫКА

Форма Бэкуса-Наура (БНФ):

```
<цифра> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<условный оператор> → if <логическое выражение> then
                        <оператор>
                      | if <логическое выражение> then
                        <оператор>
                        else <оператор>
```

Синтаксическая диаграмма (условный оператор):



ИДЕНТИФИКАТОРЫ

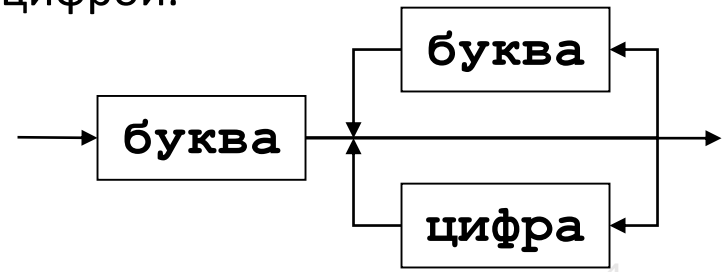
Текст программы состоит из **лексем**. Лексемы языка Delphi Pascal представлены такими категориями как специальные символы, идентификаторы, зарезервированные слова, метки, числовые константы и строковые константы. Две соседние лексемы должны отделяться друг от друга разделителями.

Идентификатор (имя) – это строка символов, используемая для идентификации некоторой сущности в программе (переменной, метки, подпрограммы, параметра и т.д.). Идентификаторы могут быть стандартные и пользовательские. Стандартные идентификаторы служат для обозначения заранее определенных разработчиками языка типов данных, констант, процедур и др.

Правила построения идентификаторов:

- идентификаторы могут иметь произвольную длину, но значащими будут только первые 255 символов;
- всегда начинается буквой, за которой могут следовать буквы и цифры (символ подчеркивания считается буквой, пробелы и специальные символы недопустимы);
- имена меток могут начинаться как буквой, так и цифрой.

Синтаксическая диаграмма конструкции <идентификатор>



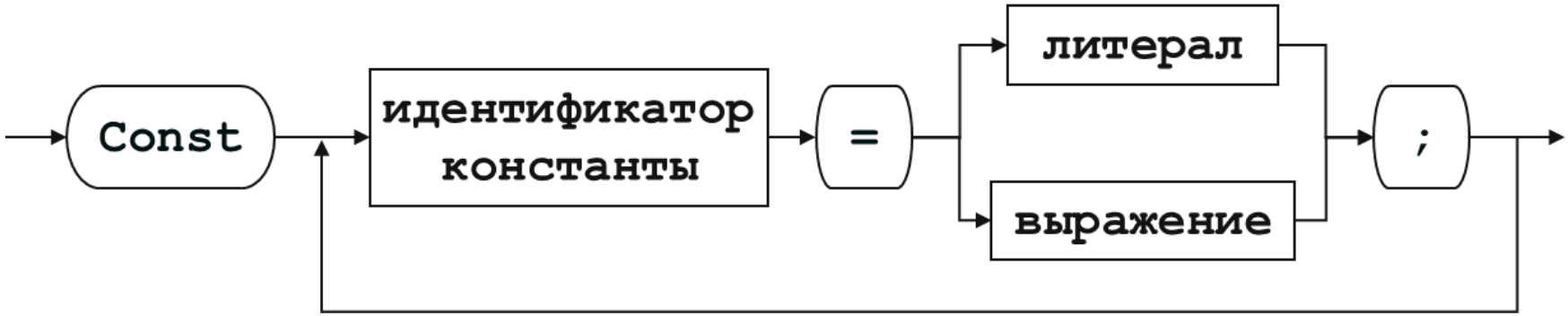
КОНСТАНТЫ

Языки программирования интерпретируют данные как константы или переменные.

Константа – элемент данных, значение которого не изменяется в процессе выполнения программы.

Литерал представляет собой значение константы, записанное непосредственно в программе (например, в выражении 2 + 5.1*x использованы два литерала "2" и "5.1").

Поименованная константа объявляется в инструкции секции описания констант Const.




```
Const
   Ln10 = 2.3026;
   Ln10R = 1 / Ln10;
   X = MaxInt div SizeOf(Real);
```

КОНСТАНТЫ

Значениями констант в языке Delphi Pascal могут быть:

- Целые числа (от -9 223 372 036 854 775 808 (-2^{63}) до 9 223 372 036 854 775 807 ($2^{63}-1$)).
- Вещественные числа. Может использоваться формат с фиксированной точкой или формат с плавающей точкой (a = 61.2; b = 3.14e5; c = -72E-3).
- Шестнадцатеричные числа (в диапазоне от $0000 0000 0000 0000 до $FFFF FFFF FFFF FFFF).
- Логические константы (True, False).
- Символы. Записываются в апострофах, кроме того допускается использование записи символа путем указания его внутреннего кода, которому предшествует символ # (Znak1 = 'd'; Znak2 = 'ф'; Znak3 = #90).
- Строки символов. Любая последовательность символов, заключенная в апострофы. Можно составить из кодов нужных символов с предшествующими каждому коду знаком #.
- Конструкторы множеств. Список элементов множества, обрамленный квадратными скобками (a = [1,2,4..7,12]; b = [red,blue,green]; c = [True]).
- Признак неопределенного указателя - NIL.

АДРЕСАЦИЯ

Оперативная память представляет собой последовательность ячеек емкостью 1 байт (8 бит). Доступ к любой ячейке осуществляется путем указания ее адреса.

MS-DOS. Память разбита на **сегменты** – непрерывные области памяти размером 64 К (65536 байт). Начало любого сегмента отстоит от начала предыдущего на 16 байт. **Смещение** – номер ячейки памяти, отсчитанный относительно той ячейки, с которой начинается сегмент.

Логический адрес конкретной ячейки: **`сегмент : смещение`**

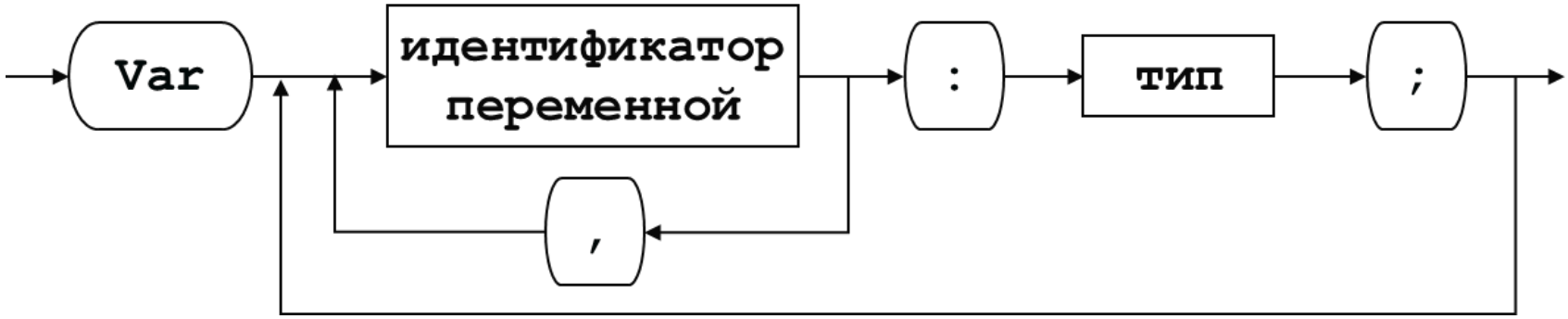
`физический адрес = сегмент*10h + смещение`

Например, ячейка с физическим адресом `00012h` может иметь логический адрес `0001h : 0002h` либо `0000h : 0012h`.

Windows. Приложения Windows имеют доступ ко всей памяти компьютера. Используется "страничная" модель памяти. Под адресом будем понимать целое число размером 32 бита (4 байта). Например, `1111FFFFh`

ПЕРЕМЕННЫЕ

Переменной называется поименованная область в памяти компьютера, выделяемая для хранения конкретных данных, значение которой может изменяться в ходе выполнения программы. Переменные интерпретируются однозначным способом – с помощью явного указания ее типа в специальном разделе описаний. Тип соответствует количеству байт, которые должны быть выделены под эту переменную в соответствии с теми значениями, которые сможет принимать переменная.



Возможна инициализация переменных в их объявлениях:

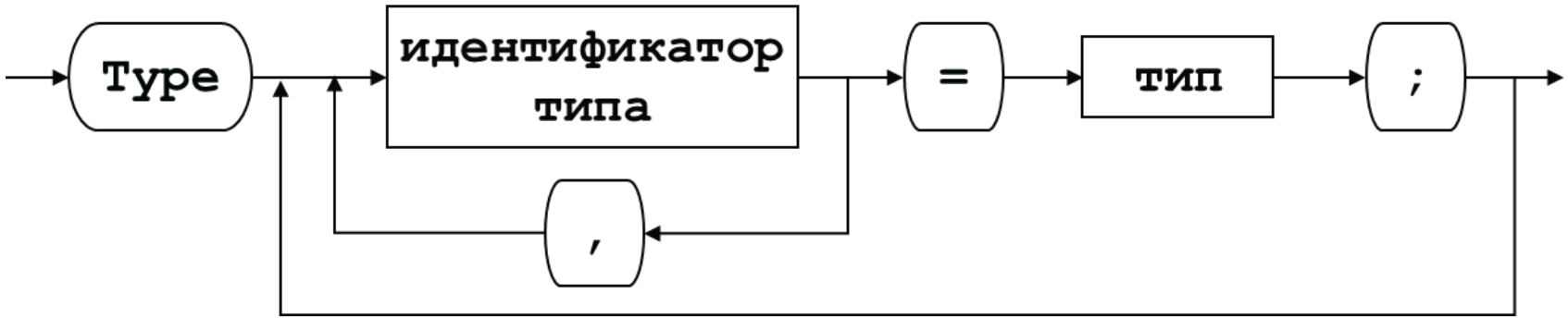
```
Var
   S : string = 'Start';
```

Типизированная константа – переменная с заранее определенным значением.

```
Const
   A : Real = 5.6;
```

ТИПЫ

Типы с уникальными именами описываются в специальной секции раздела описаний.




```
Type
   LightType = (red, yellow, green);
Var
   A, B, StreetLight : LightType;
```

Если значение не структурное, то есть не распадается на компоненты, то оно называется **скаляром**, а сам тип, описывающий такие значения – **скалярный тип**.

Параметры-атрибуты переменной: имя, адрес, значение, тип, время жизни, область видимости.

ВЫРАЖЕНИЯ, ОПЕРАЦИИ

Выражение задает порядок выполнения действий над элементами данных и состоит из операндов (константы, переменные, обращения к функциям), знаков операций и круглых скобок. **Операции** определяют действия, которые надо выполнить над операндами. Операция – атрибут типа.

```
(X + sin(Y) - 10)*4
```

`X,sin(Y),10,4` – операнды, `"+","-","*"` – операции.

Унарные операции относятся к одному операнду и всегда записываются перед ним (not, @, +, -).

Бинарные операции выражают отношение между двумя операндами и записываются между ними (X + Y).

ЛОГИЧЕСКИЙ ТИП

Значениями переменных логических (булевских) типов могут быть логические константы True, False.

```
Var
   X : Boolean;
```

Логические типы: **Boolean** (1 байт), ByteBool (1 байт), WordBool (2 байта), LongBool (4 байта).
Логические переменные могут выступать операндами в логических операциях.

Операнд 1	Операнд 2	and (конъюнкция)	or (дизъюнкция)	xor (исключающее ИЛИ)	not (отрицание)
False	False	False	False	False	-
True	False	False	True	True	-
False	True	False	True	True	-
True	True	True	True	False	-
True	-	-	-	-	False
False	-	-	-	-	True

ЦЕЛОЧИСЛЕННЫЕ ТИПЫ

Название	Длина, байт	Диапазон значений
Byte	1	0 .. 255
ShortInt	1	-128 .. +127
Word	2	0 .. 65535
SmallInt	2	-32768 .. +32767
Cardinal (LongWord)	4	0 .. 4 294 967 295
Integer (LongInt)	4	-2 147 483 648 .. +2 147 483 647
Int64	8	-2^{63} .. $+2^{63} - 1$

К операндам целых типов могут быть применены арифметические операции.

Результат операций сложения "+", вычитания "-", умножения "*", целочисленного деления **div**, возвращения остатка от целочисленного деления **mod** двух переменных X и Y относится к целому типу **Integer** (либо к Int64, если X или Y имеет тип Int64). Результат операции деления "/" относится к наиболее мощному вещественному типу Extended.

```
Var X : Byte; Y : ShortInt;
... Z := X + Y; {Z → Integer}
```

ЦЕЛОЧИСЛЕННЫЕ ТИПЫ

Арифметические операции **сдвига** K shl N и K shr N оперируют с двоичным (битовым) представлением целых чисел. Они восстанавливают в качестве результата значение, полученное путем сдвига на N позиций влево или вправо числа K, представленного в двоичном виде. Тип результата – Integer.

$2\ \text{shl}\ 7 = 256;\quad (2)_{10} = (10)_2;\quad (100000000)_2 = (256)_{10}.$

$161\ \text{shr}\ 2 = 40;\quad (161)_{10} = (10100001)_2;\quad (101000)_2 = (40)_{10}.$

При использовании логических операций применительно к целым числам осуществляется побитное (поразрядное) сравнение операндов. Тип результата – наименьший целый, включающий типы операндов. Двоичные цифры результата образуются по следующим правилам:

Операнд 1	Операнд 2	and	or	xor	not
0	0	0	0	0	-
1	0	0	1	1	-
0	1	0	1	1	-
1	1	1	1	0	-
1	-	-	-	-	0
0	-	-	-	-	1

$12\ \text{or}\ 22 = 30;\quad (12)_{10} = (01100)_2;\quad (22)_{10} = (10110)_2;\quad (11110)_2 = (30)_{10}.$

$X : \text{Byte} = 5;\quad (X)_2 = 00000101;\quad X := \text{not}\ X = (11111010)_2 = \underline{(250)_{10}}.$

$X : \text{ShortInt} = 5;\quad (X)_2 = 0\ 0000101;\quad X := \text{not}\ X = (1\ 1111010)_2 = \underline{(-6)_{10}}.$

СИМВОЛЬНЫЕ ТИПЫ

Значениями переменных символьного типа является множество всех символов компьютера. Каждому символу приписывается целое число (код).
В основе кодировки лежит 7-битный код **ASCII** (American Standard Code of Information Interchange). Символы с кодами 128..255 отводятся под национальный алфавит – 1-байтовые кодировки CP866 (DOS), KOI8-R, ANSI CP1251 (Win)). Unicode – 2 байта.

```
Var
   X : Char;
...
X := 'ф';
X := #244;      {CP1251}
```

Символьные типы – **Char**, ANSIChar (1 байт); WideChar (2 байта).

Применимы встроенные функции:

CHR(X) – преобразует выражение X : Byte в символ и возвращает этот символ;

UPCASE(X) – возвращает прописную букву, если X – строчная латинская буква, в противном случае (например, если X – русская буква) возвращает сам символ X.

ПЕРЕЧИСЛЯЕМЫЙ ТИП

Перечисляемый тип задается перечислением тех значений, которые он может получать. Значения перечисляемого типа должны иметь синтаксис идентификаторов. Во внутреннем представлении значения перечисляемого типа кодируются целыми числами, начиная от нуля.

```
Type
   Colors = (red, white, blue);
Var
   Color1, Color2 : Colors;
Var
   Color3 : (red, white, blue);
...
   Color3 := red;
```

ТИП-ДИАПАЗОН

Тип-диапазон – это подмножество базового типа, в качестве которого может выступать любой целый, логический, символьный или перечисляемый тип. Тип-диапазон задается границами своих значений внутри базового типа.

```
Type
   Digit1 = '1'..'9'; {базовый тип – символьный}
   Digit2 = 1..9; {базовый тип – целый}
Var
   LatCht : 'A'..'Z'; {базовый тип – символьный}
```

Стандартные функции:

HIGH(X) – возвращает максимальное значение типа-диапазона.

LOW(X) – возвращает минимальное значение типа-диапазона.

ПОРЯДКОВЫЕ ТИПЫ

Рассмотренные скалярные типы (целые, символьные, логические, перечисляемый, тип-диапазон) относятся к классу **порядковых типов**. Все значения порядкового типа можно упорядочить, с каждым из них можно сопоставить целое число – порядковый номер значения.

К любому из порядковых типов применима функция **ORD(X)**, которая возвращает порядковый номер значения X.
Целые типы: ORD(X) = X (кроме Int64).
Логический тип Boolean: ORD(False) = 0, ORD(True) = 1.
Символьные типы: ORD(X) = код символа X.
Перечисляемый тип: ORD(X) = целое число 0..65535 в соответствии с положением X в списке.
Тип-диапазон: ORD(X) определяется свойствами базового типа.

Стандартные функции:
PRED(X) – возвращает предыдущее значение порядкового типа;
ORD(PRED(X)) = ORD(X) - 1.
SUCC(X) – возвращает следующее значение порядкового типа;
ORD(SUCC(X)) = ORD(X) + 1;

```
Var C,D : Char;
begin C := 'f'; D := PRED(C); end.
```

ВЕЩЕСТВЕННЫЕ ТИПЫ

Обработка вещественного числа выполняется с некоторой конечной точностью, зависящей от его внутреннего формата (типа).

Название	Длина, байт	Количество значащих цифр	Диапазон значений
Real	**8**	**15...16**	**$\pm 5.0 \cdot 10^{-324}$ … $\pm 1.7 \cdot 10^{308}$**
Real48	6	11…12	$\pm 2.9 \cdot 10^{-39}$ … $\pm 1.7 \cdot 10^{38}$
Single	4	7...8	$\pm 1.5 \cdot 10^{-45}$ … $\pm 3.4 \cdot 10^{38}$
Double	8	15...16	$\pm 5.0 \cdot 10^{-324}$ … $\pm 1.7 \cdot 10^{308}$
Extended	10	19...20	$\pm 3.6 \cdot 10^{-4932}$ … $\pm 1.1 \cdot 10^{4392}$
Comp	8	19...20	-2^{63} … 2^{62}
Currency	8	19…20	–922 337 203 685 477.5808 … 922 337 203 685 477.5807

ПРЕДСТАВЛЕНИЕ ВЕЩЕСТВЕННЫХ ЧИСЕЛ В ПАМЯТИ

Структура вещественных чисел в памяти компьютера соответствует представлению в формате с плавающей точкой:

s	e	m
знак (1 бит)	порядок(d бит)	мантисса (r бит)

1 + d + r = 8N, где N – число байт, отводимых под переменную данного типа (Single – d = 8 бит, r = 23 бита; Extended – d = 15 бит; r = 64 бита).

s = 0, если знак числа "+"; s = 1, если знак "-";
e – задает истинный порядок числа $t = e - (2^{d-1} - 1)$;
m – задает мантиссу (дробную часть) $m_1 = m \cdot 2^{-r}$ $(0 \le m_1 < 1)$.

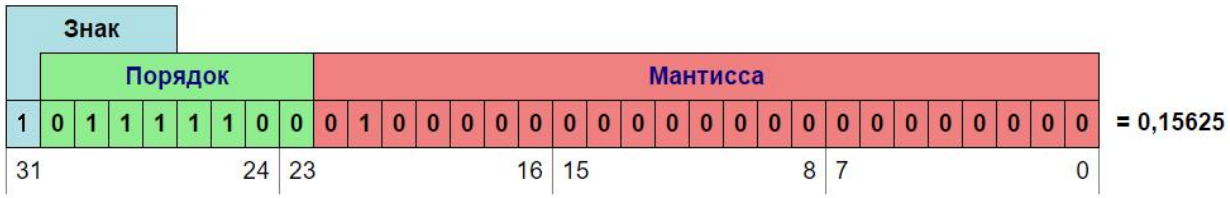
Записанное число может быть определено по формуле:

$$(-1)^s \cdot (1+m_1) \cdot 2^t \leftarrow \textbf{нормализованная запись}$$

Пример: $(-0.15625)_{10}$ как тип Single – N = 2 байта (s = 1, d = 8, r = 23).

$(-0.15625)_{10} = (-0.00101)_2 = (-1.01)_2 \cdot 2^{-3}$, т.е. $t = (-3)_{10}$, $m_1 = (0.01)_2$ => s = 1,

e = $-3 + (2^7 - 1) = (124)_{10} = (0111\ 1100)_2$, m = $m_1 \cdot 2^{23} = (10\ 0000\ 0000\ 0000\ 0000\ 0000)_2$



ВЕЩЕСТВЕННЫЕ ТИПЫ

Вещественные типы введены в расчете на **арифметический сопроцессор** – устройство, которое подключается непосредственно к центральному процессору (или интегрировано в ЦП) и предназначено для выполнения операций над числами в формате с плавающей точкой и длинными целыми числами. Сопроцессор всегда обрабатывает числа в формате типа Extended (остальные вещественные типы получаются усечением результатов до нужных размеров).

```
Var
  E1,E2 : Extended; E3 : Double; Result : Single;
  ...
  Result := E1*E2/E3;
```

Тип **Currency** используется для представления денежных единиц. В памяти он хранится как масштабированное в 10 000 раз 8-байтовое целое (минимизируются ошибки округления). Совместим с другими вещественными типами.

Значения типа **Comp** – вещественные представления "больших" целых чисел. Оставлен для обратной совместимости с ранними версиями языка.

Возможные ситуации, когда результат арифметических операций не удовлетворяет ограничениям на диапазон (или форму) представляемых чисел:

– переполнение (`Var X,Y:SmallInt;...X:=20000; Y:=15000; X:=X+Y;`)
– "исчезновение порядка".

ОПЕРАЦИИ ОТНОШЕНИЯ

Операции отношения выполняют сравнение двух совместимых операндов и определяют, истинно значение выражения или ложно. Определены для скалярных типов, строк, множеств и др. Результат – True или False.

Операция	Название	Выражение	Результат
=	Равно	A = B	*True*, если A равно B
<>	Не равно	A <> B	*True*, если A не равно B
>	Больше	A > B	*True*, если A больше B
<	Меньше	A < B	*True*, если A меньше B
>=	Больше или равно	A >= B	*True*, если A больше или равно B
<=	Меньше или равно	A <= B	*True*, если A меньше или равно B
in	Принадлежность	A in M	*True*, если A принадлежит множеству M

ПРИОРИТЕТ ОПЕРАЦИЙ

Операции	Приоритет
@, not	1 (высший)
*, /, div, mod, and, shr, shl	2
+, -, or, xor	3
>, <, <>, =, <=, >=, in	4 (низший)

Правила для определения старшинства операций:

- операнд, находящийся между двумя операциями с различными приоритетами, связывается с операцией, имеющей более высокий приоритет;
- операнд, находящийся между двумя операциями с равными приоритетами, связывается с операцией, которая находится слева;
- выражение, заключенной в скобки, перед выполнением вычисляется, как отдельный операнд.

Результат операции @ – адрес операнда.

СТРУКТУРА ПРОГРАММЫ

```
Program Example;
{$APPTYPE CONSOLE} {директива консольного приложения}
Uses SysUtils;     //подключение модулей (библиотек)
Label
   1,m1,Stop;
Const
   MaxN: Word = 100;
   Kod = $20;  {Шестнадцатеричная константа}
Type
   Matrix = array [1..Kod] of Real;
Var
   A,B : Integer; C : Integer = 5;
   Result : Matrix;
Procedure <имя>;
   begin <тело процедуры> end;
Function <имя> : <тип>;
   begin <тело функции> end;
BEGIN
   <операторы>;
END.
```

СТРУКТУРА ПРОГРАММЫ

Программа состоит из заголовка, раздела описаний и раздела операторов. В строке может быть несколько операторов (объявлений). Разделитель – символ ";".

Любой подраздел раздела описаний может отсутствовать, если в нем нет необходимости. Подразделы **`Label, Type, Const, Var`** могут следовать друг за другом в любом порядке и встречаться в разделе описаний сколько угодно раз. При этом обязательно соблюдение правила: все, что в программе используется, должно быть перед этим описано.

Комментарии – это произвольный поясняющий текст в любом месте программы, заключенный в фигурные скобки { }, или между двойными символами (* *), или после символа //.

Директивы компилятора используются программистом для управления режимами компиляции (включать или выключать контроль ошибок, изменять распределение памяти и т.д.). Они заключаются в фигурные скобки и имеют отличительный признак $. При использовании директив-переключателей указывается буква-ключ с последующим знаком "+" (включить режим) или "-" (выключить). Например, {$R-} – отключить проверку принадлежности заданному диапазону. Директивы-параметры устанавливают режимы и параметры компиляции. Например, {$APPTYPE CONSOLE} – директива консольного приложения.

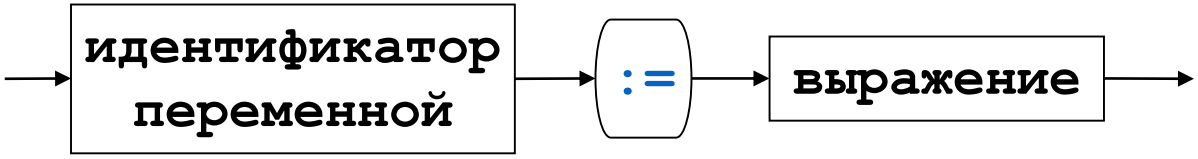
Глава 6. УПРАВЛЯЮЩИЕ СТРУКТУРЫ

- **Оператор присваивания**
- **Простой и составной операторы**
- **Условный оператор**
- **Оператор множественного выбора**
- **Оператор цикла с предусловием**
- **Оператор цикла с постусловием**
- **Оператор цикла с параметром**
- **Оператор и процедуры безусловного перехода**

ОПЕРАТОР ПРИСВАИВАНИЯ

Оператором называется конструкция языка программирования, служащая для задания какого-либо действия или последовательности действий над данными в программе. Совокупность операторов программы реализует заложенный в ней алгоритм.

Процесс «засылки» значения в переменную называется **присваиванием** (первое присваивание называется **инициализацией**). Присваивание осуществляется с помощью специальной конструкции – **оператора присваивания**:




```
Var W,H : Integer;
Begin
   W := 23;
   H := 17;
   W := W * H
End.
```

ПРОСТОЙ И СОСТАВНОЙ ОПЕРАТОРЫ

Оператор в программе – это единое неделимое предложение, выполняющее какое-либо действие. **Простой оператор** не содержит в себе других операторов (оператор присваивания, оператор безусловного перехода,…).

Два последовательных оператора должны разделяться точкой с запятой (имеет смысл конца оператора):

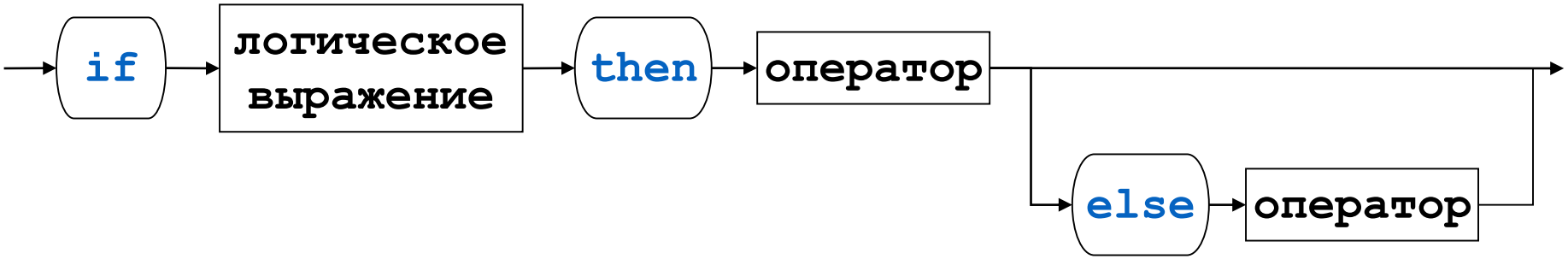
```
a := 11;  b := a * a;  Write(a,b);
```

Составной оператор – это последовательность операторов, рассматриваемых как единый. Оформляется с помощью зарезервированных слов begin и end (операторные скобки)

```
begin
   a := 11;
   b := a * a;
   Write(a,b)
end;
```

УСЛОВНЫЙ ОПЕРАТОР

Условный оператор используется для программирования ветвлений, т.е. ситуаций, когда возникает необходимость при определенных условиях выполнять различные действия. Условный оператор имеет структуру:



В каждой ветви допускается запись только одного оператора.

```
if K > 5 then
   begin
      X := X + 5; Y := 1
   end
else
   Y := -1;
```

ВЛОЖЕННЫЕ УСЛОВНЫЕ ОПЕРАТОРЫ

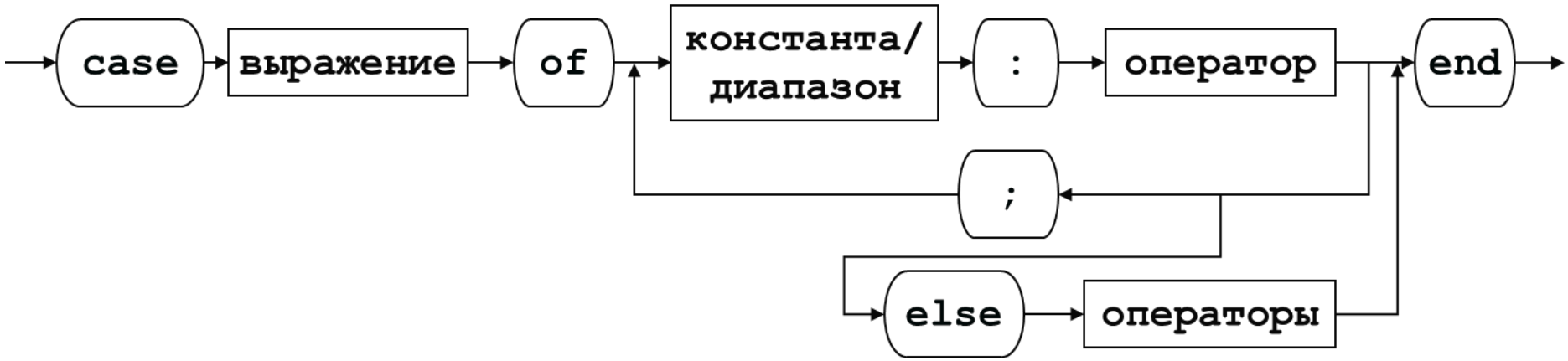
Альтернатива else считается принадлежащей ближайшему условному оператору if, не имеющему своей ветви else.

```
if <условие 1> then
   if <условие 2> then
      <оператор А>
else
   <оператор Б>;
```

```
if <условие 1> then
   begin
      if <условие 2> then
         <оператор А>
   end
else
   <оператор Б>;
```

ОПЕРАТОР МНОЖЕСТВЕННОГО ВЫБОРА

Оператор выбора используется для реализации нескольких альтернативных вариантов действий, каждый из которых соответствует своим значениям некоторого параметра.



Значение <выражения>, а также <константа/диапазон> должны относиться к одному из порядковых типов.

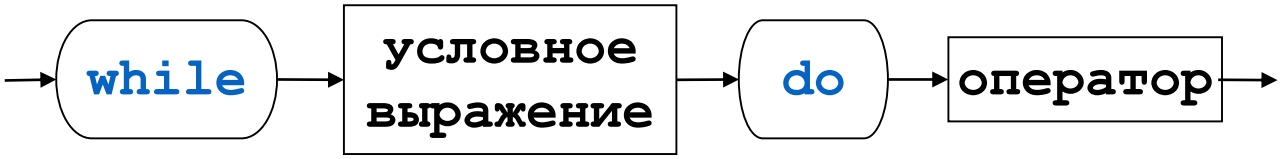
В зависимости от значения <выражения> выбирается тот оператор, которому предшествует константа выбора, равная вычисленному значению (альтернатива – операторы после else).

Значения констант должны быть уникальными в каждом наборе, т.е. они могут появиться только в одном варианте.

ОПЕРАТОР МНОЖЕСТВЕННОГО ВЫБОРА

```
case I of                                {I : Word}
   1      :  X := X +1;
   2,3    :  X := X +2;
   4..9   :  begin
                Write(X);
                X := X + 3               {м.б. ";"}
             end                         {м.б. ";"}
   else
      X := X * X;
      Writeln(X)                         {м.б. ";"}
end;
```

ОПЕРАТОР ЦИКЛА "ПОКА" (С ПРЕДУСЛОВИЕМ)

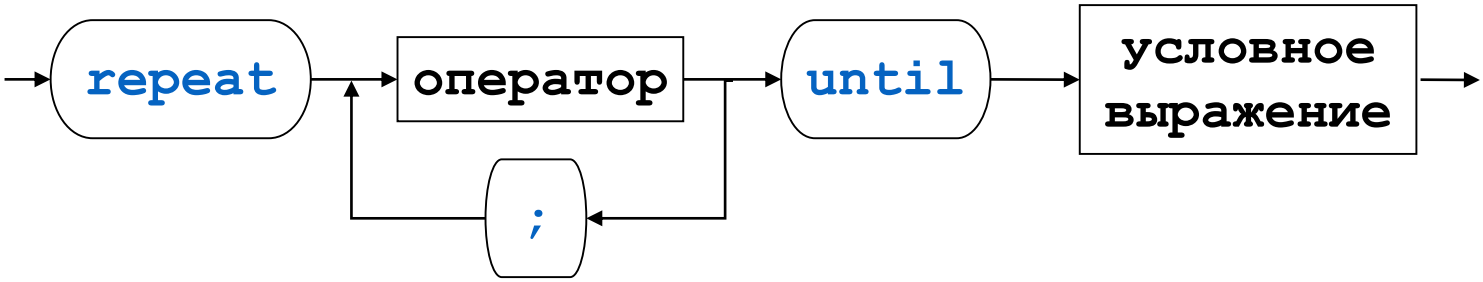


<Оператор> (**тело цикла**), стоящий после служебного слова do, будет выполняться циклически до тех пор, пока выполняется логическое условие, т.е. пока значение <условного выражения> равно True.

Чтобы цикл имел шанс когда-либо завершиться, содержимое его тела должно влиять на условие цикла. Условие должно состоять из корректных выражений и значений, определенных еще до первого выполнения тела цикла.

```
Var   F,N : Int64;                    {вычисление 10!}
Begin
   F := 1; N := 1;
   while N <= 10 do
      begin
         F := F * N; Inc(N)           {N := N + 1}
      end;
   Writeln(F)
End.
```

ОПЕРАТОР ЦИКЛА "ДО" (С ПОСТУСЛОВИЕМ)



Операторы между словами repeat и until образуют **тело** цикла.

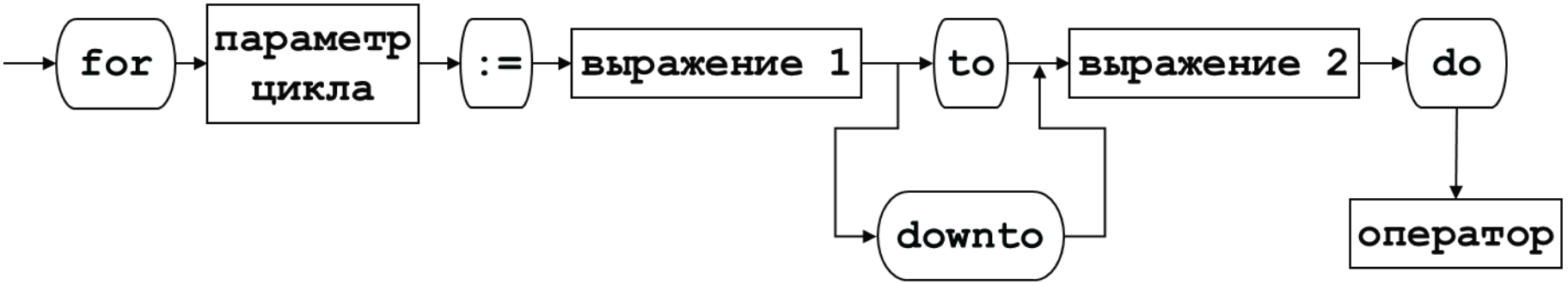
Если <условное выражение> имеет значение True, то цикл завершается.

Цикл "Пока" – "Пока условие истинно, выполнять операторы тела".

Цикл "До" – "Выполнять тело цикла до тех пор, пока не станет истинным условие";

ОПЕРАТОР ЦИКЛА С ПАРАМЕТРОМ (ЦИКЛ ПО СЧЕТЧИКУ)

Используется для организации "строгих" циклов, которые должны быть проделаны заданное число раз.



<Параметр цикла> – переменная порядкового типа, к этому же типу должны относиться значения <выражения 1> и <выражения 2>.

Значение <параметра цикла> меняется в возрастающем (при использовании зарезервированного слова to) или убывающем (downto) порядке от значения <выражения 1> до значения <выражения 2> с постоянным шагом, равным интервалу между двумя ближайшими значениями в типе, к которому относится <параметр цикла> (для целочисленных типов – это 1, для символьного – от одного символа к другому при увеличении кода на 1, и т.д.).

Циклы for допускают вложенность, если никакой из вложенный циклов не использует и не модифицирует переменные – параметры внешних циклов.

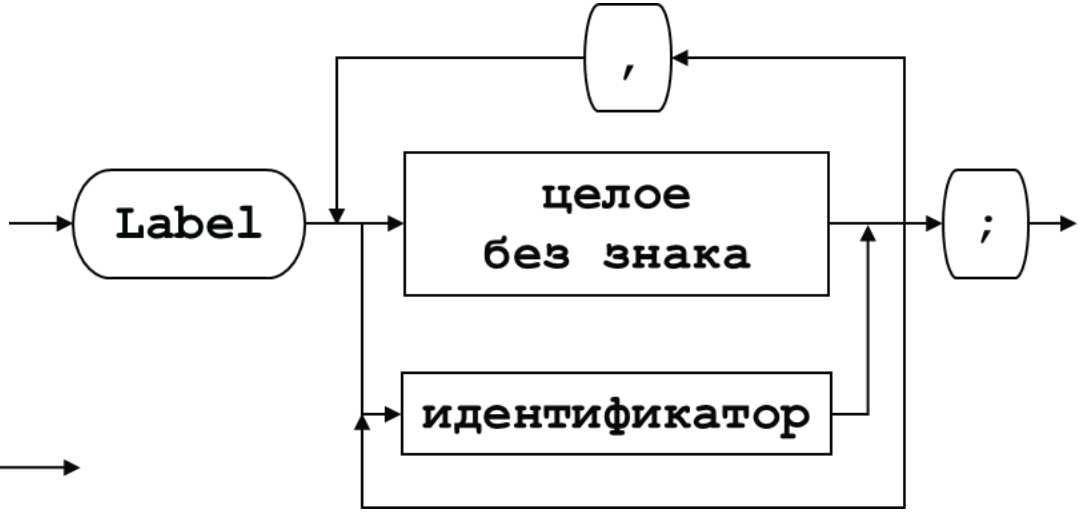
ОПЕРАТОР ЦИКЛА С ПАРАМЕТРОМ (ЦИКЛ ПО СЧЕТЧИКУ)

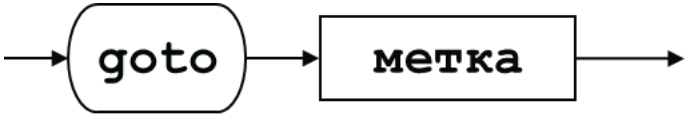
```
Var
   I : Integer;
   C : Char;
   B : Boolean;
   E : (E1,E2,E3,E4);
Begin
   for I := -10 to 10 do Write(I);
   for C := 'a' to 'z' do Write(C);
   for B := False to True do Write(B);
   for E := E1 to E4 do
      begin                    {составной оператор}
         I := ORD(E);
         Write(I)
      end
End.
```

ОПЕРАТОР БЕЗУСЛОВНОГО ПЕРЕХОДА

Оператор безусловного перехода передает управление выполнением в указанное с помощью метки место программы (является "лишним" с точки зрения теории структурного программирования).

Синтаксические диаграммы <объявления меток> и <оператора безусловного перехода>





Метка может стоять в программе в любом месте между операторами и отделяется от второго оператора двоеточием ":".

Область действия операторов безусловного перехода строго локализована. Запрещены переходы по оператору goto между процедурами, а также между основным блоком и процедурой.

```
Label L1, L2;
Begin
...
   goto L1;
...
   L1 : goto L2;
...
   L2 : End.
```

ПРОЦЕДУРЫ БЕЗУСЛОВНОГО ПЕРЕХОДА

Процедуры Exit и Halt специально предназначены для выхода из программных блоков (процедур, функций, основного программного блока).

Halt (<код завершения>) осуществляет выход из программы, возвращая операционной системе заданный код завершения.

Exit осуществляет безусловный выход из подпрограммы. Если процедура использована в основном блоке, то она выполняется аналогично Halt.

Процедуры неструктурной передачи управления при работе с циклическими структурами:

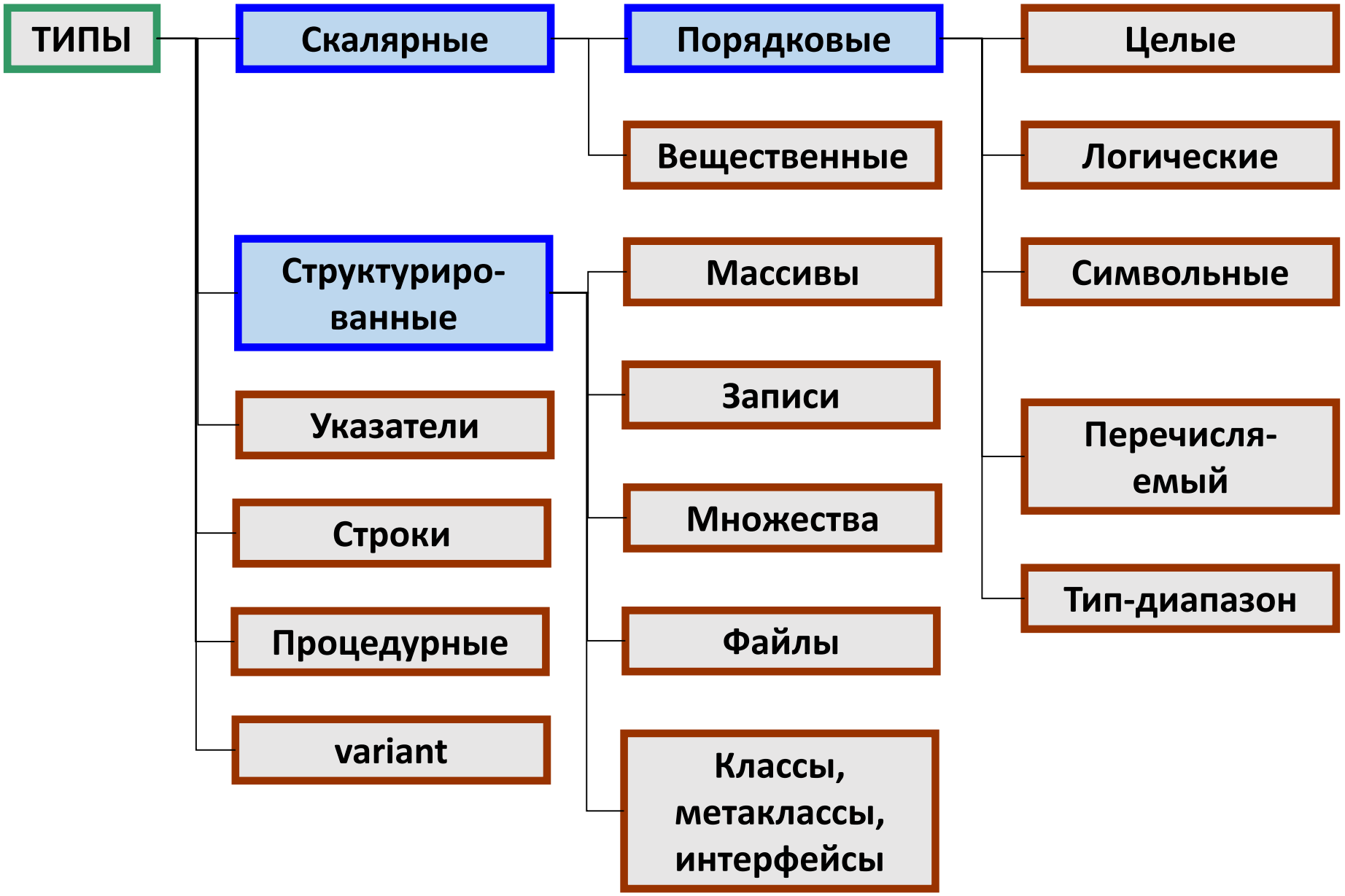
Break – реализует выход из цикла любого типа;

Continue – осуществляет переход на следующую итерацию цикла, игнорируя оставшиеся до конца тела цикла операторы.

Глава 7. СТРУКТУРИРОВАННЫЕ ТИПЫ ДАННЫХ

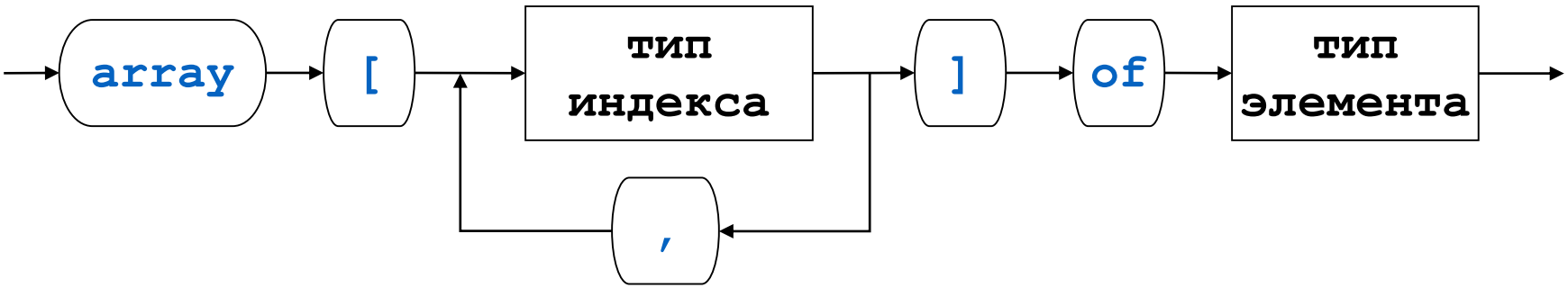
- Организация типов данных
- Массивы
- Записи, оператор присоединения
- Множества, операции над множествами
- Строки, стандартные процедуры и функции, работающие со строками
- Совместимость типов
- Явное и неявное преобразование типов

ТИПЫ ДАННЫХ ЯЗЫКА DELPHI PASCAL



МАССИВЫ

Массив – упорядоченная совокупность однотипных данных.



<тип индекса> – любой порядковый тип, размер которого не превышает 2 Гбайт

```
Type
   Vector = array [1..3] of Real;
                    {тип индекса – тип-диапазон}
Var
   R,V : Vector;
```

<u>или</u>

```
Var
   R,V : array [1..3] of Real;
   T : array [1..5] of Byte = (0,1,2,3,4);
```

МАССИВЫ

<тип элемента> массива – любой допустимый в Delphi Pascal тип кроме файла (в том числе и другой массив).

Многомерные массивы:

```
Type
   Matrix = array [0..5] of array [-2..2] of
            array [Char] of Real;
```

или

```
Type
   Matrix = array [0..5,-2..2,Char] of Real;
```

Доступ к элементам массива:

```
Var
   m : Matrix;
   N : Byte;
Begin
... m[1,0,'d'] := 5.2;
   N := 2;
   m[N-1][0]['n'] := 6.3; ...
End.
```

МАССИВЫ

Присваивание массивов:

```
Var
   a,b : array [1..5] of Real;
Begin
   ...
   a := b;
   ...
End.
```

При большом числе элементов массива возможно ограничение, связанное с максимальным объемом памяти, который отводится под глобальную переменную – 2 Гбайт.

```
Var
  Mas3D = array [1..1000,1..1000,1..1000] of Integer;
                                    {4 000 000 000 байт}
```

При компиляции в режиме, задаваемом ключом {$R+}, будет проверяться принадлежность значения индекса объявленному диапазону, и в случае нарушения границ будет выдано сообщение об ошибке.

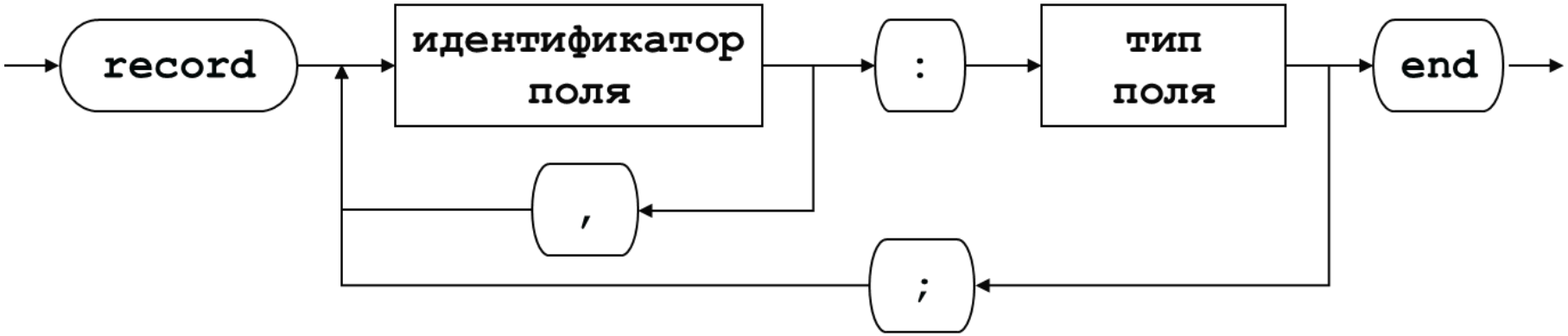
ДИНАМИЧЕСКИЕ МАССИВЫ

```
Var
   a : array of Real;
   N,i : Byte;
Begin
   N := 5;                          {a → 0}
   SetLength(a,N);                  {a → (0,0,0,0,0)}
   for i := 0 to N-1 do
      a[i] := i*2;                  {a → (0,2,4,6,8)}
   SetLength(a,N-2);                {a → (0,2,4)}
   SetLength(a,N+1);                {a → (0,2,4,0,0,0)}
End.
```

Переменная типа динамического массива – это указатель (значение – адрес).

ЗАПИСИ

Запись – структура данных, состоящая из фиксированного числа разнотипных компонент, называемых **полями записи**.



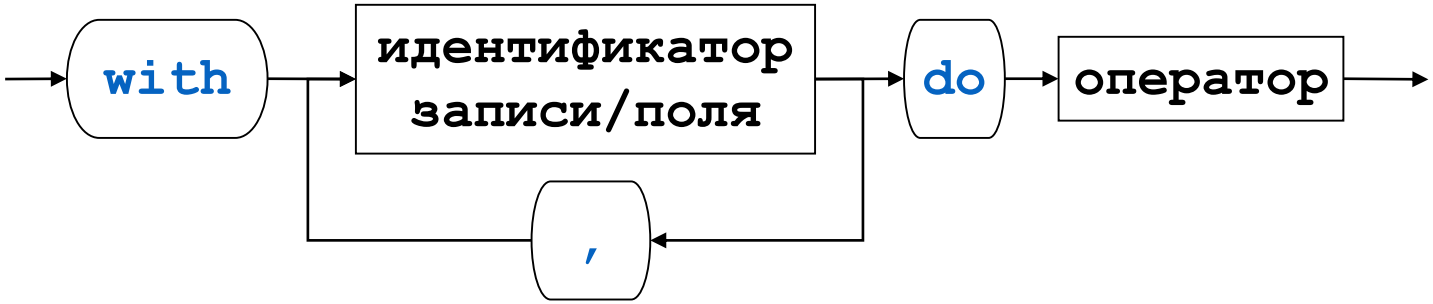

```
Type
   Data = record
          X,Y : Integer;
          Z   : Char
          end;
Var    D1,D2 : Data;
Begin
...    D1.X := 10;
...    D2.Z := 'n';
...    D2 := D1;          {присваивание записей}
End.
```

ЗАПИСИ

Полем записи может быть другая запись (вложенные структуры):

```
Var
   D : record
         X : Integer;
         R : record
               RX : Integer;
               RZ : Char
               end
         end;
Begin
   ...
   D.R.RX := 2;
   ...
End.
```

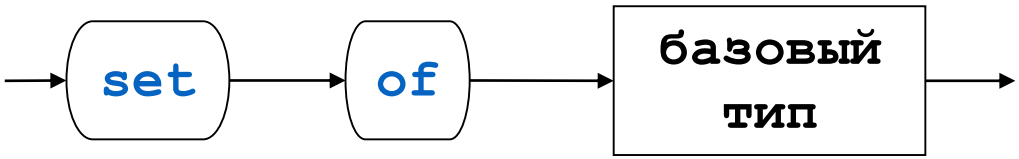
ОПЕРАТОР ПРИСОЕДИНЕНИЯ




```
with D do
   begin
      R.RX := 2;          {D.R.RX}
      with R do
         RZ := 'f';       {D.R.RZ}
   end;
```

МНОЖЕСТВА

Множество – это структурированный тип данных, представляющий собой неупорядоченную совокупность неповторяющихся элементов. Количество элементов, входящих во множество, может меняться в пределах от 0 до 256 (множество может быть пустым).



<базовый тип> – любой порядковый тип с числом элементов, не превышающим 256 (с кодами в диапазоне 0..255).

```
Type
   TypeSet1 = set of Char;
   TypeSet2 = set of 0..9;

   VideoType = (Hercules,CGA,EGA,VGA,SVGA);
   TypeSet3 = set of VideoType;
```

МНОЖЕСТВА

Значением переменной множественного типа является множество, которое определяется с помощью конструктора множества, представляющего собой перечисление элементов базового типа в квадратных скобках.

```
Var
   Set1, Set2 : set of Byte;
   Set3 : set of 'a'..'f';
   X : Integer;
Begin
   ...
   Set1 := [3..10,12];
   Set3 := ['a','d'];
   X := 5;
   Set1 := [X+2,4,9];
   Set3 := [];
   Set2 := [9,7,9,4];
   ...
End.
```

ОПЕРАЦИИ НАД МНОЖЕСТВАМИ

Set1 = [0..3,6] Set2 = [3..9]

* – **пересечение множеств**, результат содержит элементы, общие для обоих множеств (Set1 * Set2 = [3,6]);

+ – **объединение множеств**, результат содержит элементы первого множества, дополненные недостающими элементами второго (Set1 + Set2 = [0..9]);

- – **разность множеств**, результат содержит элементы первого множества, которые не принадлежат второму (Set1 - Set2 = [0,1,2]);

= – **проверка эквивалентности**, возвращает True, если оба множества эквивалентны;

<> – **проверка неэквивалентности**, возвращает True, если множества неэквивалентны;

<= – **проверка вхождения**, возвращает True, если первое множество включено во второе;

>= – **проверка вхождения**, возвращает True, если второе множество включено в первое;

in – **проверка принадлежности** (E in S), возвращает True, если значение E входит в множество S и принадлежит базовому типу этого множества (3 in Set1 = True, 2*2 in Set1 = False).

МНОЖЕСТВА

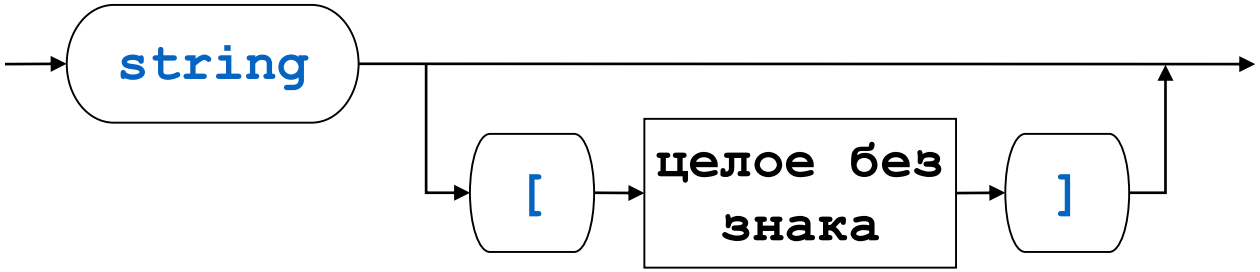
Процедуры, параметром которых является множество:

INCLUDE (S,I) – включает новый элемент I в множество S (включаемый элемент должен принадлежать базовому типу множества S).
EXCLUDE (S,I) – исключает элемент I из множества S.

```
Var
   Set1 : set of 1..10;
   I : Byte;
Begin
   ...
   Set1 := [2,3,4];
   Include(Set1,2*3);
   for I := 1 to 10 do
      if I in Set1 then Writeln(I);
   Writeln(SizeOf(Set1));
   Set1 := Set1 + [12];
   ...
End.
```

СТРОКИ

Тип String используются для обработки текстов и трактуется как цепочка символов. **Строка** – динамический (переменной длины) массив, состоящий из символов. Максимально возможная длина строки 255 символов. Тип объявляется как String[N], где N - максимальное число символов в строке.




```
Var
   S32 : String[32];
   S255 : String[255]; {String}
Begin
   S32 := 'Это строка';
   {ORD(S32[0]) = 10}


   S32[3] := 'a';
   S32 := S32 + '!!!';
End.
```

ПРОЦЕДУРЫ И ФУНКЦИИ ДЛЯ РАБОТЫ СО СТРОКАМИ

LENGTH (S : String) : Integer – возвращает длину строки (функция);

CONCAT (S1, S2,…,Sn : String) : String – возвращает конкатенацию (слияние) строк S1,…,Sn (функция);

COPY (S : String; Start, Len : Integer) : String – возвращает подстроку длиной Len, начинающуюся с позиции Start строки S (функция);

DELETE (Var S : String; Start, Len : Integer) – удаляет из S подстроку длиной Len, начинающуюся с позиции Start строки S (процедура);

INSERT (SubS : String; Var S : String; Start : Integer) – вставляет в S подстроку SubS, начиная с позиции Start (процедура);

POS (SubS, S : String) : Integer – ищет вхождение подстроки SubS в строке S и возвращает номер первого символа SubS в S или 0, если SubS нет в S (функция);

ПРОЦЕДУРЫ ПРЕОБРАЗОВАНИЯ

STR (X; Var S : String) – преобразует числовое значение X в строковое S, возможно задание формата для X (Str(X:F:n,S), где F – общая ширина поля, n – количество символов в дробной части для вещественных чисел);

VAL (S : String; Var X; Var Code : Integer) – преобразует строковое значение S (строку цифр) в значение числовой переменной (X – целое или вещественное, параметр Code содержит ноль, если преобразование прошло успешно, в противном случае он содержит номер позиции в строке, где обнаружен ошибочный символ, при этом X не меняется).

Операции отношения (=, <>, >, <, >=, <=):
Результат - логическая константа (*True, False).* Сравнение строк выполняется последовательно слева направо с учетом внутренней кодировки символов до первого несовпадающего символа.

'aBcd' = 'ab' (результат *False*);
'aBcd' > 'ab' (результат *False*);
'aBcd' < 'ab' (результат *True*).

Строковые типы Delphi Pascal: ShortString, ANSIString, String, WideString, PChar.

СОВМЕСТИМОСТЬ ТИПОВ

Delphi Pascal требует соблюдения правил совместимости типов в ряде случаев: при использовании оператора присваивания, при выполнении операций отношения, при подстановке переменных или значений в вызовы процедур и функций и т.д.

Два типа совместимы, если они тождественны (идентичны). Типы считаются **тождественными**, если:

1. Они описаны вместе, либо одним и тем же идентификатором типа:

```
Type
      T1 = Boolean;
      T2 = Boolean;
      T3,T4 = array [1..2] of Real;
```

2. Типы описаны как эквивалентные

```
Type
      T1 = array [1..2] of Real;
      T2 = T1;
      T3 = T2;
```

СОВМЕСТИМОСТЬ ТИПОВ

Типы совместимы (гарантирует работу операций отношения, подстановку значений или переменных в параметры функций и процедур), если:

- оба типа являются тождественными;
- оба типа являются вещественными;
- оба типа являются целыми;
- один тип является поддиапзоном другого;
- оба типа являются поддиапазонами одного и того же базового типа;
- оба типа являются множествами, составленными из одного и того же базового типа;
- один тип является строковым, а другой символьным или строковым;
- один тип является указателем, а другой указателем или ссылкой;
- оба типа являются классами, ссылками на класс или интерфейсами, причем один тип унаследован от другого;
- ...

СОВМЕСТИМОСТЬ ПО ПРИСВАИВАНИЮ

Переменной *X* (тип *T1*) может быть присвоено значение *Y* (тип *T2*) (т.е., X := Y) если:

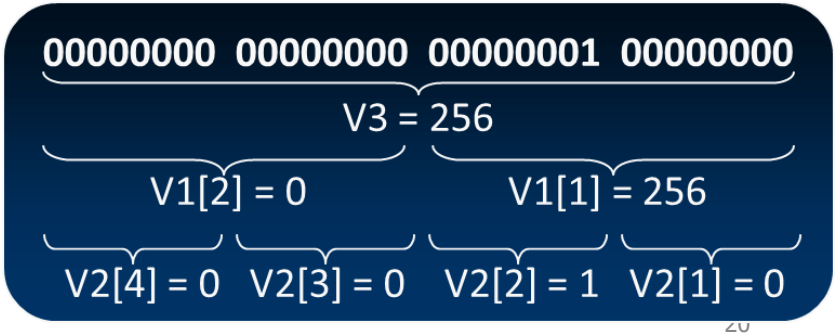
- *T1* и *T2* – тождественные типы, и не один не является файловым типом (или структурным типом, содержащим компонент с файловым типом);
- *T1* и *T2* – совместимые типы (в смысле, рассмотренном ранее), относящиеся к порядковым, и значения типа *T2* попадают в диапазон возможных значений *T1*;
- *T1* и *T2* – вещественные типы и значения типа *T2* попадают в диапазон возможных значений *T1*;
- *T1* – вещественный тип, *T2* – целочисленный тип;
- *T1* и *T2* – строковые типы;
- *T1* – строковый тип, *T2* – символьный тип;
- *T1* и *T2* совместимые множества и все члены значения множества типа *T2* попадают в диапазон возможных значений *T1*;
- *T1* и *T2* совместимые адресные типы;
- *T1* и *T2* оба являются классами, ссылками на классы или интерфейсами, при этом *T2* унаследован от *T1*;
- ...

ЯВНОЕ ПРЕОБРАЗОВАНИЕ ТИПОВ

1. Может быть реализовано посредством использования специальных функций:
TRUNC(x) – преобразует значение вещественного типа в значение целого типа, отбрасывая дробную часть;
ROUND(x) – преобразует значение вещественного типа в значение целого типа, округляя его до ближайшего целого;
ORD(x) – преобразует значение порядкового типа в его номер;
CHR(x) – преобразует код символа в сам символ и др.

2. При приведении типа переменной используется функция, которая совпадает с именем типа, к которому должна быть приведена переменная. Необходимо равенство длин внутреннего представления в памяти обоих типов.

```
Type  M2Word = array [1..2] of Word;
      M4Byte = array [1..4] of Byte;
Var   V1 : M2Word;      V2 : M4Byte;
      V3 : LongInt;     V4 : SmallInt;
Begin
      V3 := 256;
      V1 := M2Word(V3);
      V2 := M4Byte(V3);
V4 := SmallInt(V1[1]); End.
```



НЕЯВНОЕ ПРЕОБРАЗОВАНИЕ ТИПОВ

1. При несовпадении типов правой и левой частей оператора присваивания для совместимых типов автоматически выполняется неявное преобразование результата выражения к типу переменной, указанной в правой части (например, деление целых чисел → Extended).

2. Происходит, если одна и та же область памяти попеременно трактуется как содержащая данные то одного, то другого типа (совмещение в памяти данных разного типа).

Совмещение данных в памяти, в частности, возможно при размещении данных разного типа по одному и тому же абсолютному адресу. Для размещения переменной по нужному абсолютному адресу она описывается с последующей стандартной директивой Absolute, за которой помещается имя ранее определенной переменной.

```
Var
      x : Real;
      y : array [1..2] of Integer absolute x;
```

Глава 8. ПОДПРОГРАММЫ И МОДУЛИ

- Процедуры и функции
- Область видимости переменных
- Формальные и фактические параметры
- Параметры-значения и параметры-переменные
- Особенности функций
- Параметры структурированных типов
- Рекурсия
- Процедурный тип
- Параметры-функции и параметры-процедуры
- Стандартные математические функции и процедуры
- Модуль, структура модуля
- Стандартные модули

ПРОЦЕДУРЫ И ФУНКЦИИ

Подпрограммы (процедуры или функции) представляют собой относительно самостоятельные фрагменты программы, оформленные особым способом и снабженные именем. Упоминание этого имени в тексте программы называется **вызовом** процедуры (функции).

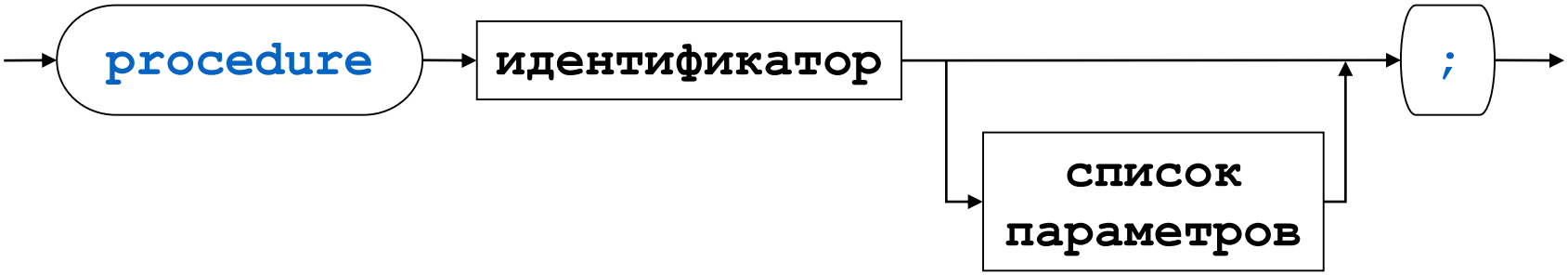
Подпрограммы представляют собой инструмент, с помощью которого любая программа может быть разбита на ряд относительно независимых частей, что имеет смысл по двум причинам.

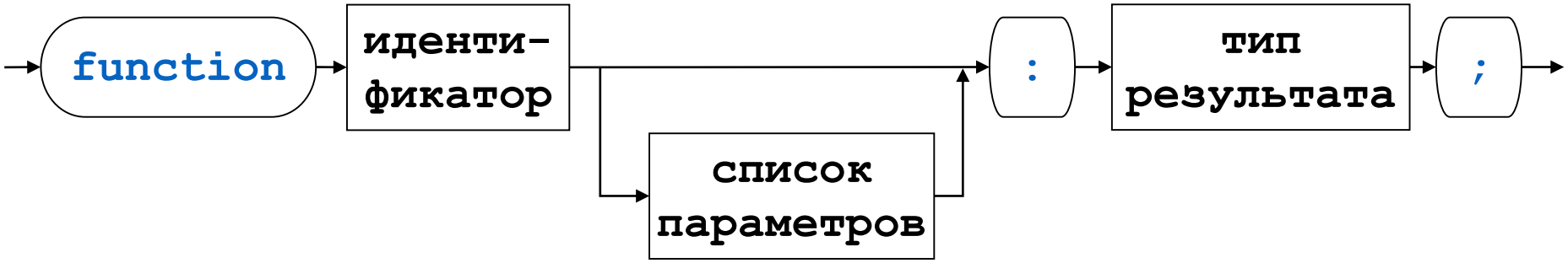
Во-первых, это средство экономии памяти и других ресурсов: каждая подпрограмма существует в программе в единственном экземпляре, в то время как обращаться к ней можно многократно из разных точек программы.

Во-вторых, при таком разбиении максимально реализуется принцип нисходящего проектирования. В этом случае алгоритм представляется в виде последовательности относительно крупных подпрограмм, реализующих более или менее самостоятельные смысловые части алгоритма.

```
Program Calculator;
   Begin
      Initialize;
      DrawScreen;
      Calculation;
      RestScreen;
   End.
```

СТРУКТУРА ПОДПРОГРАММЫ, ЗАГОЛОВКИ ПРОЦЕДУРЫ И ФУНКЦИИ

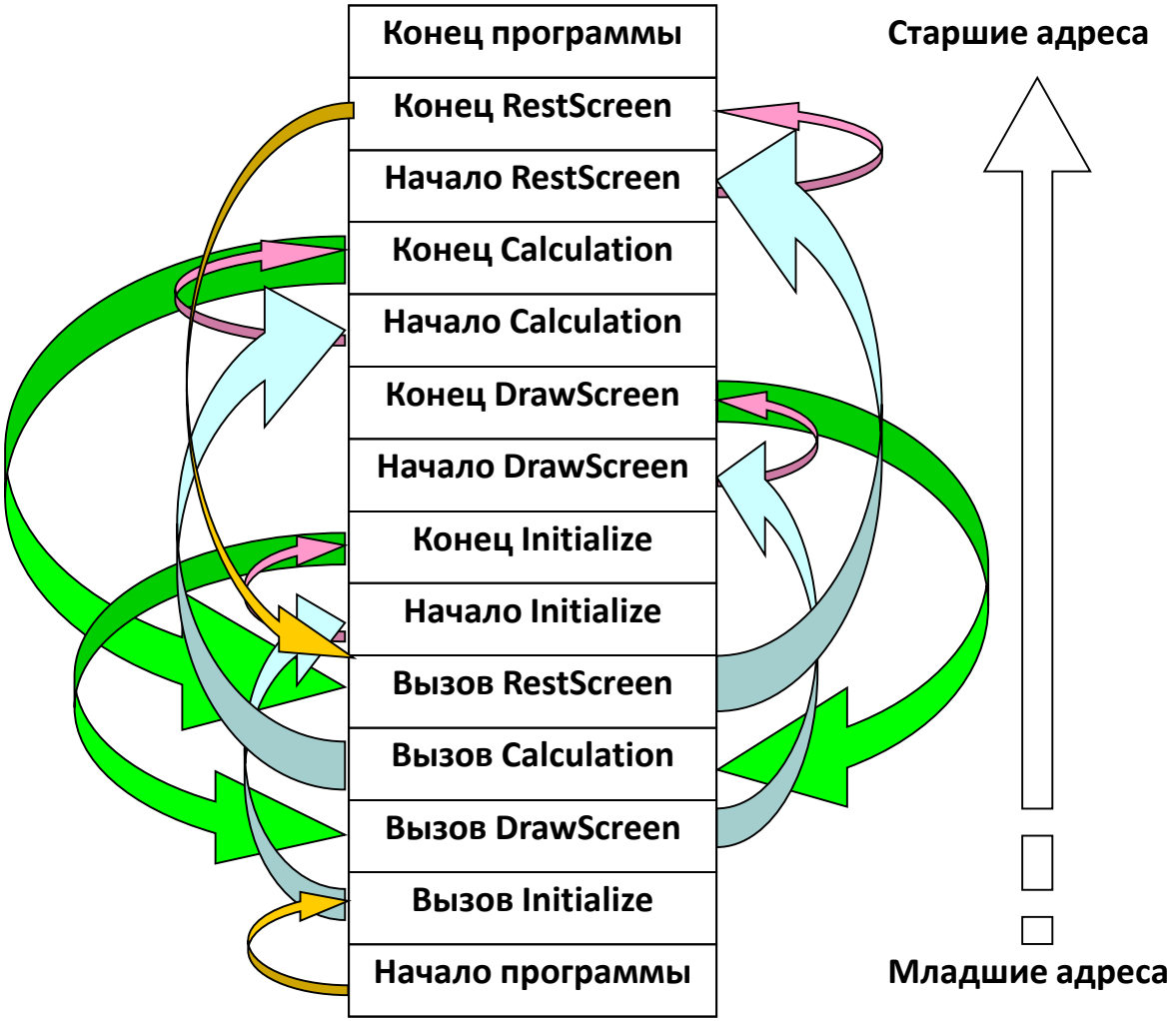




ВЫЗОВ ПОДПРОГРАММ

Машинные команды, из которых состоит любая программа, хранятся в ОЗУ. Выполнение программы происходит путем последовательного чтения команд, начиная с первой; следующей выполняется команда, расположенная "выше" только что выполненной ("естественный" порядок выполнения команд"). Адрес байта, где находится текущая команда, называется **активной точкой программы**.

Естественный порядок выполнения команд нарушается при обращении к подпрограммам.



СТЕК В ПАМЯТИ

Адрес точки возврата (адрес машинной команды в момент вызова подпрограммы) хранится в ячейках специальной области ОЗУ, которая называется **стеком**. Кроме того, стек в памяти используется для хранения значений локальных переменных подпрограммы во время выполнения операторов, входящих в ее тело. Принцип его работы совпадает с организацией динамической структуры данных, имеющей такое же имя – стек. Стек – это LIFO-устройство (Last In First Out). Стек в памяти заполняется сверху вниз.

После выполнения всех операторов подпрограммы стек освобождается для хранения данных следующей вызванной подпрограммы. В последнюю очередь считывается адрес команды, которая стоит после вызова выполненной подпрограммы в основном блоке (адрес точки возврата).

ОБЛАСТЬ ВИДИМОСТИ ПЕРЕМЕННЫХ

Глобальные константы, типы, переменные – это те, которые объявлены в программе вне процедур или функций. Они доступны из любого места программы.

Локальные – это константы, типы, переменные, существующие только внутри процедур или функций, и объявленные либо в соответствующих разделах *Const*, *Type*, *Var* внутри данной процедуры или функции, либо в списке формальных параметров (как параметры-значения).
Глобальные переменные хранятся в области данных, локальные – в стеке.

Область видимости переменной (константы, типа) – это ряд операторов, в которых переменная "видна". Все имена, описанные внутри подпрограммы (локальные данные), локализуются в ней, т.е. они как бы "невидимы" снаружи подпрограммы. При наличии вложенных подпрограмм переменные данного уровня являются "видимыми", т.е. доступными для всех нижних уровней описания процедур. Подпрограммы, наряду со своими локальными данными, могут использовать и модифицировать и глобальные. Для этого лишь нужно, чтобы описание процедуры (функции) стояло в тексте программы ниже, чем описание соответствующих глобальных типов, констант и переменных.

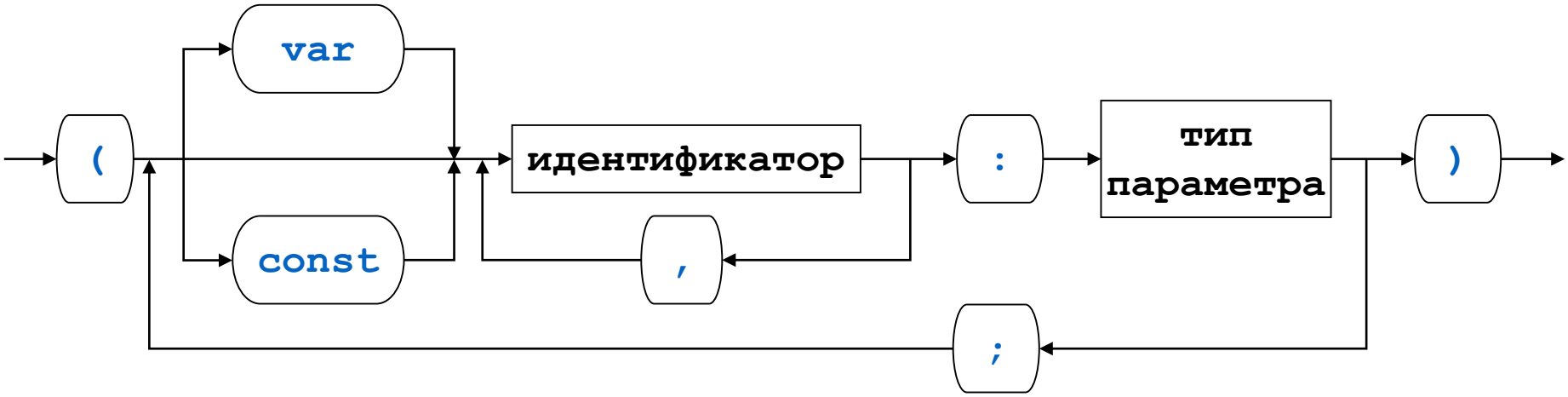
Время жизни переменной – это время, в течение которого переменная связана с определенной ячейкой памяти.

ГЛОБАЛЬНЫЕ И ЛОКАЛЬНЫЕ ПЕРЕМЕННЫЕ

```
Program Main;
Var
   Xmain, Ymain : Real; {глобальные переменные}
Procedure P1;                {описание процедуры}
   Var
      Res : Real;            {локальная переменная}
   begin
      Res := 0.5;
      Ymain := Res + Xmain*Ymain;
      Xmain := Xmain + 1
   end;
...
Var          {глобальные переменные, недоступные в P1)
   Zmain : Extended;
...
Begin
   ...
   P1;                          {вызов процедуры}
   ...
End.
```

ПАРАМЕТРЫ

Параметры обеспечивают обмен значениями между вызывающей частью программы и вызываемой подпрограммой. Могут быть параметры-значения, параметры-переменные или параметры-константы.



Описываемые в заголовке подпрограммы параметры называются **формальными**:

```
Procedure P1 (A : Integer; Var B : Real);
```

а те, которые подставляются на их место при вызове – **фактическими**, т. к. они при выполнении программы замещают все вхождения в подпрограмму своих формальных "двойников":

```
P1(A_fact,B_fact);
```

ПАРАМЕТРЫ

Если параметр описан как **параметр-значение**, то в подпрограмму передается копия значения этого фактического параметра, и никакие изменения этой копии не возвращаются в вызывающий блок. Параметр-значение – это локальная переменная подпрограммы, стартовое значение которых задается при вызове подпрограммы из внешних блоков.

Если параметр описан как **параметр-переменная** (служебное слово Var в заголовке) или как **параметр-константа** (слово Const), то в подпрограмму передаются адреса фактических параметров. Все изменения параметра-переменной в подпрограмме происходят с переменной, переданной в качестве фактического параметра. Параметр-константа не может изменяться в подпрограмме.

На место фактического параметра-значения можно подставлять литерал, выражение, идентификатор (переменную или константу), а на место параметра-переменной и параметра-константы – только идентификатор.

ПАРАМЕТРЫ

```
{$J+}
Const
   A : Integer = 5; {типизированные константы}
   B : Integer = 7;
Procedure Plus (Var C : Integer; B : Integer);
   begin
      C := C + C;
      B := B + B;
      Writeln(C,B)
   end;
Begin
   Writeln(A,B);
   Plus(A,B);
   Writeln(A,B)
End.
```

ФУНКЦИИ

Кроме обмена значениями через параметры функция возвращает в вызываемый блок свое значение в виде скаляра, строки или указателя. Для присвоения функции значения ее имя должно появиться хотя бы однажды в левой части оператора присваивания в теле самой функции (альтернатива – *Result*). Функция имеет побочный эффект, если она производит другие действия, не связанные с вычислениями своего значения.

```
Var                                          {функция вычисления X^Y}
   X,Y : Real;
Function Power (Arg, P : Real) : Real;
   begin
      if Arg <> 0 then
         Power := exp(P * ln(abs(Arg)))      {Result :=...}
      else
         if P = 0 then
            Power := 1                       {Result :=...}
         else   Power := 0                   {Result :=...}
   end;
Begin
   Readln(X,Y);   Writeln(Power(X,Y))
End.
```

ПАРАМЕТРЫ СТРУКТУРИРОВАННЫХ ТИПОВ

Чтобы передать в подпрограмму массив (строку, запись,...), нужно сначала описать его тип. Следствие – возможные затраты памяти.

```
Type
      vector = array [1..100] of Real;
Procedure Sum (A : vector; Var B : vector);
      begin...end;
```

Открытые массивы:

```
Procedure Sum (A : array of Real; Var B : array of Real);
      Var i : Word;
      begin
            for i := 0 to High(A) do B[i] := A[i]+B[i]
      end;
Var A1,A2 : array [1..100] of Real;
Begin
      {инициализация массивов A1 и A2}
      Sum(A1,A2);
      {вывод A2}
End.
```

РЕКУРСИЯ

Применительно к практическому программированию под **рекурсией** понимается вызов функции (процедуры) из тела этой же самой функции (процедуры). Необходимым условием работоспособности рекурсивных подпрограмм является наличие условия окончания рекурсивных вызовов.

```
Var                                  {вычисление n!}
   N : Word;
Function Fact (m : Word) : Int64;
   begin
      if m = 0 then
         Fact := 1
      else
         Fact := m*Fact(m-1)
   end;
Begin
   Readln(N);
   Writeln('n! = ',Fact(N))
End.
```

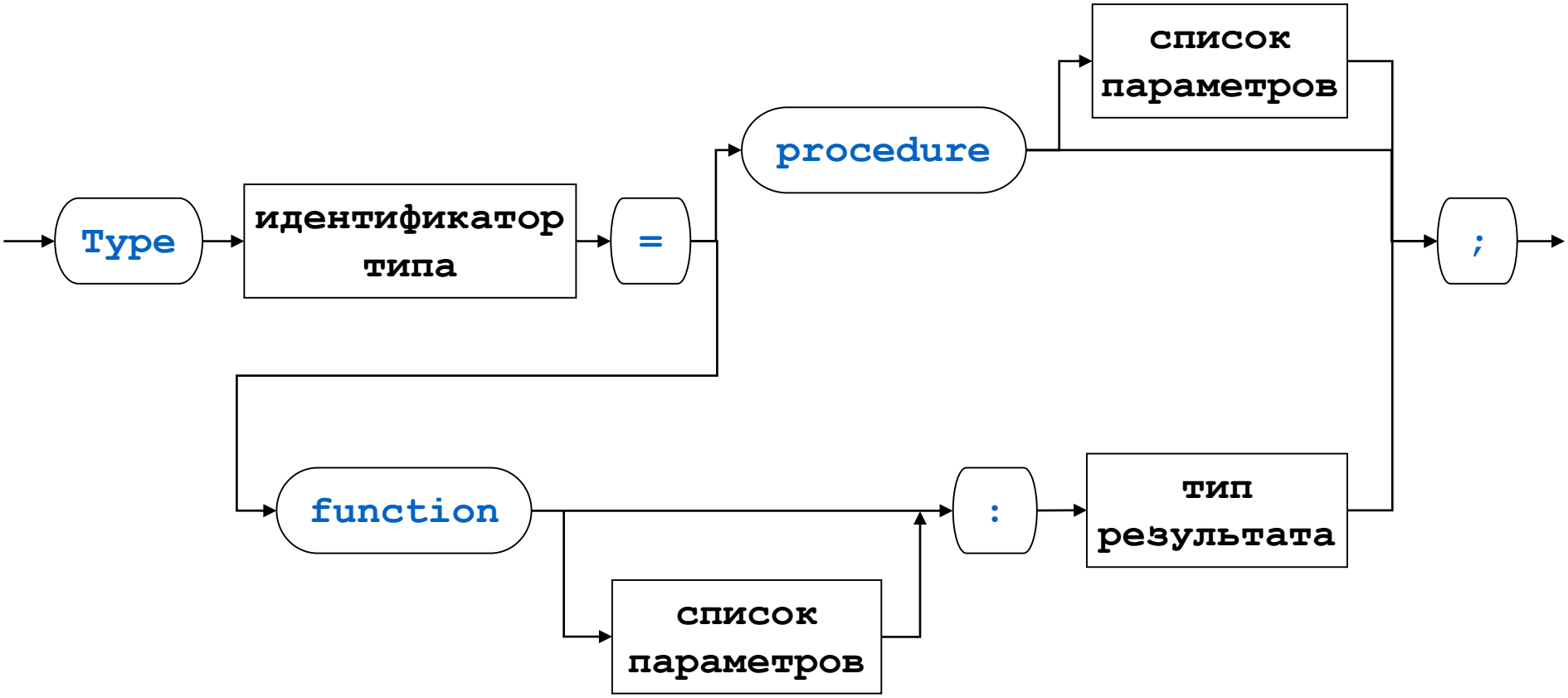
РЕКУРСИЯ

Реализация рекурсивных алгоритмов при большом числе итераций ограничена доступным объемом памяти; каждый отложенный вызов процедуры или функции – это свой набор значений всех локальных переменных этой подпрограммы (фрейм активации), размещенных в стеке.

```
{вычисление n! итерационным методом}
Function Fact (m : Word) : Int64;
   Var
      I : Word; P : Int64;
   begin
      P := 1;
      if m = 0 then
         Fact := 1
      else
         begin
            for I := 1 to m do P:=P*I;
            Fact := P
         end
   end;
```

ПРОЦЕДУРНЫЙ ТИП

Процедурный тип позволяет интерпретировать процедуры и функции как данные-объекты, которые можно использовать, в частности, при передаче параметров. Значением переменной процедурного типа является конкретная процедура или функция.



ПАРАМЕТРЫ-ФУНКЦИИ И ПАРАМЕТРЫ-ПРОЦЕДУРЫ

```
{вычисление интеграла функции}
Type
   FuncType = Function (X : Real) : Real;
Procedure Integral (LowerLimit, UpperLimit : Real;
                    Var Result : Real;
                    Funct : FuncType);
   begin ... end;
Function SinExp (Arg : Real) : Real;
   begin
      SinExp := Sin(Arg) + Exp(Arg)
   end;
Var
   Res : Real;
...
Begin
   ...
   Integral (0, 1, Res, SinExp);
   ...
End.
```

ПРОЦЕДУРА – ПОЛЕ ЗАПИСИ

```
Type
   ProcType = Procedure;
   RecType = record
             X,Y : Integer;
             P : ProcType;
             end;
Var
   Rec1,Rec2 : RecType;
Procedure P1;
   begin
   ...
   end;
Begin
   ...
   with Rec1 do P := P1;
   ...
End.
```

Процедурные переменные совместимы с переменными типа *Pointer* (их значения – адреса). Процедуры и функции – это адреса.

СТАНДАРТНЫЕ МАТЕМАТИЧЕСКИЕ ФУНКЦИИ И ПРОЦЕДУРЫ

ABS(X) – возвращает абсолютное значение аргумента X (X – целое/вещественное, результат – как у аргумента);
SQR(X) – возвращает квадрат X (X – целое/вещественное, результат – как у аргумента);

SIN(X), **COS(X)**, **ArcTan(X)** – возвращают значения синуса, косинуса и арктангенса (X – целое/вещественное, результат – *Extended*);
SQRT(X) – возвращает квадратный корень из X (X > 0 – целое/вещественное, результат – *Extended*);
EXP(X), **LN(X)** – возвращает экспоненту или натуральный логарифм X (X – целое/вещественное, результат – *Extended*);
FRAC(X) – дробная часть числа X (X – целое/вещественное, результат – *Extended*);

INT(X) – целая часть числа X (X – целое/вещественное, результат – *Extended*);
TRUNC(X) – целая часть числа X (X – целое/вещественное, результат – *Int64*);
ROUND(X) – округление X до ближайшего целого (X – целое/вещественное, результат – *Int64*);

СТАНДАРТНЫЕ МАТЕМАТИЧЕСКИЕ ФУНКЦИИ И ПРОЦЕДУРЫ

RANDOM(X) – возвращает случайное целое число из диапазона от 0 до X (X и результат – *Integer*);
RANDOM – возвращает случайное число от 0 до 1 (результат – *Extended*);
RANDOMIZE – процедура гарантирует несовпадение последовательностей случайных чисел, выдаваемых функцией Random;

ODD(X) – возвращает *True*, если X – нечетное число, и *False*, если X – четное (X – целое, результат – *Boolean*);

INC(Var X : <целое>), **DEC(Var X : <целое>)** – увеличивает или уменьшает значение X на 1 (процедуры);
INC(Var X : <целое>; N : <целое>), **DEC(Var X : <целое>; N : <целое>)** – увеличивает или уменьшает значение X на N (процедуры);

```
Var
   X : Integer;
Begin
   X := 2;
   INC(X,4)          {X → 6}
End.
```

МОДУЛИ

Модуль – это автономно компилируемая программная единица, включающая в себя различные компоненты раздела описаний (типы, константы, переменные, процедуры, функции) и, возможно, некоторые исполняемые операторы, которая предназначена для использования другими модулями и программами.

```
Unit <имя модуля>;
        Interface
           <секция интерфейса>
        Implementation
           <секция реализации>
        Initialization
           <секция инициализации>
        Finalization
           <секция завершения>
        End.
```

Имя модуля (например, Unit_1) должно совпадать с именем дискового физического файла, в который помещается исходный текст модуля (unit_1.pas). Для подключения ресурсов модуля к другим программным единицам (программа, модуль) необходимо указать его имя в спецификации Uses раздела описаний этой программной единицы (Uses Unit_1;).

```
Unit Unit_1;
Interface
   Type
      Complex = record
                Re, Im : Real
                end;
   Var
      Zmain : Complex;
   Procedure P_1 (X,Y : Complex; Var Z : Complex);
Implementation
   Procedure P_1;
      begin
         Z.Re := X.Re + Y.Re;
         Z.Im := X.Im + Y.Im
      end;
Initialization
   Zmain.Re := 1; Zmain.Im := -1
End.
```

СЕКЦИИ МОДУЛЯ

В **интерфейсной секции** содержатся объявления всех глобальных элементов модуля (типов, констант, переменных, функций, процедур), которые должны стать доступными основной программе и другим модулям, к которым подключен данный модуль. При объявлении глобальных подпрограмм указывается только их заголовок. Если при объявлении типов, данных и подпрограмм используются элементы, введенные в других модулях, то они должны быть указаны в разделе Uses сразу после слова Interface.

Секция реализации содержит описание подпрограмм, объявленных в интерфейсной части, и описание внутренних ресурсов модуля (локальных переменных, типов, подпрограмм). Обращение к этим ресурсам возможно только из подпрограмм, описанных в этом же модуле.

Секция инициализации (может отсутствовать) включает программные действия, которые будут произведены перед выполнением основной программы, к которой подключен данный модуль. Обычно в разделе инициализации происходит заполнение стартовыми значениями библиотечных переменных, а также одноразовые действия, которые должны выполниться в начале программы (например, открываться нужные файлы и т.д.).

Секция завершения (может отсутствовать) содержит операторы, которые будут выполняться при завершении работы приложения (основной программы).

СТАНДАРТНЫЕ МОДУЛИ DELPHI PASCAL

- **Системные модули:**

- **SYSTEM** – включает все стандартные математические процедуры и функции, обеспечивает работу с файлами, с динамической памятью, с другими модулями и т.д.; подключается автоматически к каждой программе.
- **SYSUTILS** – содержит дополнительные процедуры и функции для работы с файлами, дисками, строками, по обработке исключительных ситуаций и др.
- **MATH** – содержит множество дополнительных математических функций и процедур.

- **Модули визуальных компонентов** (VCL – Visual Component Library) используются для визуальной разработки полнофункциональных GUI-приложений - приложений с графическим пользовательским интерфейсом (Graphical User Interface).

Глава 9. ВВОД-ВЫВОД ДАННЫХ И ФАЙЛОВАЯ СТРУКТУРА

- **Логический и физический файлы**
- **Типы файловой переменной**
- **Общие процедуры работы с файлами**
- **Текстовые файлы**
- **Обмен данными с консолью**
- **Типизированные файлы**
- **Нетипизированные файлы**

ЛОГИЧЕСКИЙ И ФИЗИЧЕСКИЙ ФАЙЛ

Файловая система, реализуемая в Delphi Pascal, состоит из двух уровней: логических файлов и физических файлов.

Логический файл – это переменная одного из файловых типов, определенных в Delphi Pascal. Введенная файловая переменная может быть использована как средство общения с любым физическим файлом. Специальной процедурой AssignFile устанавливается, что объявленный логический файл будет служить средством доступа к конкретному физическому файлу.

```
Var
   F : TextFile; {файловая переменная – лог. файл}
Begin
   AssignFile(F,'A:\text.txt');
   ...   {связывается физический файл text.txt
          на диске A: с логическим файлом F}

End.
```

Физический файл – это:

- именованная область на устройствах внешней памяти (адресная часть в виде строковой константы оформляется в соответствии с правилами Windows – 'C:\PASCAL\program.pas');
- логическое устройство.

ЛОГИЧЕСКИЕ УСТРОЙСТВА, БУФЕРИЗАЦИЯ

Логические устройства используются для обмена информацией с основными устройствами ввода-вывода, такими как дисплей, клавиатура, и т.п. Они имеют стандартные имена (также записываются в виде строковой константы):

'CON' – консоль (клавиатура и экран);

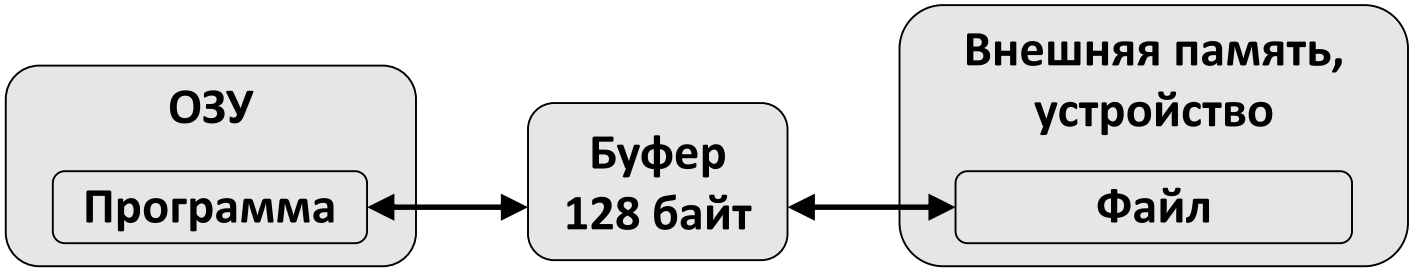
'LPT1', 'LPT2'… – параллельные порты (типа Centronix);

'PRN' – принтер (синоним имени LPT1);

'COM1', 'COM2'... – последовательные порты;

'NUL' – фиктивное устройство (пустой файл), используется для отладки.

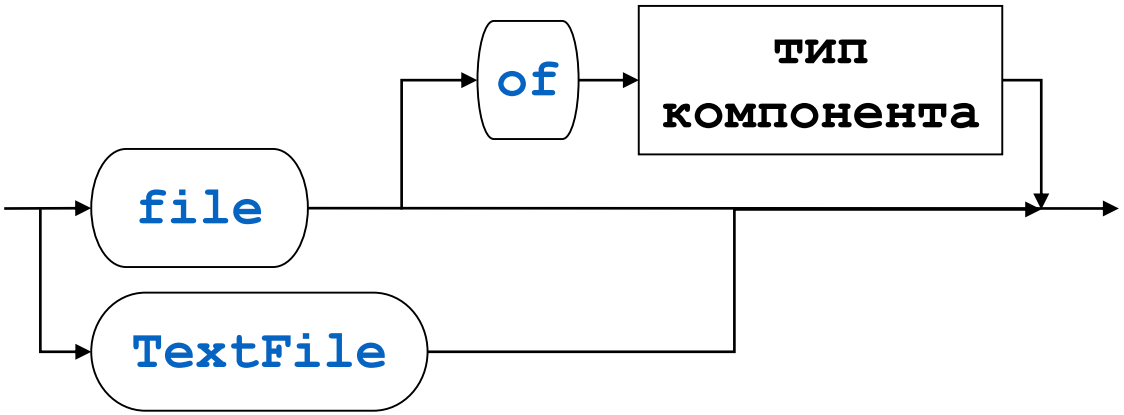
Физически операции ввода-вывода с файлами выполняются с использованием специального буфера. Стандартный объем буфера 128 байт. Физическая запись на устройство происходит только тогда, когда информацией будет занят последний байт буфера. Этим достигается компромисс между количеством и длительностью обращений к диску. **Буферизация** – накопление данных с целью обмена ими между программой и внешними устройствами.



ФАЙЛОВЫЕ ТИПЫ

Turbo Pascal поддерживает три файловых типа:

- текстовые файлы – `Var F1 : TextFile;`
- типизированные файлы – `Var F2 : file of Real;`
- нетипизированные файлы – `Var F3 : file;`



ОБЩИЕ ПРОЦЕДУРЫ РАБОТЫ С ФАЙЛОВЫМИ ПЕРЕМЕННЫМИ

- **AssignFile (Var F : <файловый тип>, St : String)**. Инициализирует файловую переменную F, связывая ее с физическим файлом, определенным строкой St.
- **Reset (Var F : <файловый тип>)**. Открывает файл, определенный файловой переменной F, для чтения. Для корректного выполнения процедуры необходимо иметь файл с соответствующим именем на диске.
- **Rewrite (Var F : <файловый тип>)**. Открывает файл, определенный файловой переменной F , для записи. При открытии для записи существующего файла старый файл уничтожается.
- **AppEnd (Var F : TextFile)**. Открывает текстовый файл, определенный файловой переменной F, для добавления строк.
- **Read (F,...), Readln(F,...), Write(F,...), Writeln(F,...)**. Выполняют операции чтения/записи (кроме нетипизированных файлов).
- **CloseFile (Var F : <файловый тип>)**. Закрывает файл, открытый для записи или чтения. Связь файла с файловой переменной сохраняется.

ТЕКСТОВЫЕ ФАЙЛЫ

Физический текстовый файл трактуется в Delphi Pascal как совокупность строк переменной длины, состоящих из символов кодовой таблицы. В конце каждой строки ставится маркер конца строки EOLN (End Of Line – последовательность кодов ASCII #13 и #10), а в конце всего файла – маркер конца файла EOF (End Of File – код ASCII #26).

Физические текстовые файлы связываются с файловыми переменными, принадлежащими типу TextFile.

Для ввода и вывода информации в текстовые файлы используют процедуры Read (ReadLn) и Write (WriteLn), первым параметром которых является имя логического файла, после чего следует список переменных, относящихся к символьному (Char) или строковому (String) типу, а также к любому целому или вещественному.

```
WriteLn(Var F : TextFile; X1, X2,…, Xn);
```

ЧТЕНИЕ ИНФОРМАЦИИ ИЗ ТЕКСТОВОГО ФАЙЛА (READ И READLN)

- Если переменные ввода относятся к символьному типу Char, то символы считываются поочередно в соответствующие переменные (без разделителей).
- Для строкового типа String[N] количество "заносимых" символов в строковую переменную зависит от ее объявленной длины. Но если попался символ конца строки #13, то чтение строки прекращается.
- При введении числовых значений два числа считаются разделенными, если между ними есть пробел, символ табуляции (#9) или символ конца строки (#13). Если нарушен формат, то фиксируется ошибка ввода-вывода.

Процедура ReadLn считывает значения в текущей строке, и затем переводит позицию в начало следующей строки, даже если в текущей строке остались непрочитанные данные.

```
Var
   X1, X2 : Real;
   F : TextFile;      {F → 12.3 13.4 14.4 15.6}
Begin
   ...
   ReadLn(F,X1,X2);   {X1 → 12.3; X2 → 13.4}
End.
```

ЗАПИСЬ ИНФОРМАЦИИ В ТЕКСТОВЫЙ ФАЙЛ (WRITE И WRITELN)

При записи информации в текстовый файл список вывода может включать выражения типа Char, String, Boolean, а также целого или вещественного типов. Процедура Write выводит данные в текущую строку и не закрывает ее, т.е. следующие данные запишутся в ту же строку. Процедура WriteLn выводит строку данных и закрывает ее, приписывая символы #13 и #10 в ее конец. Имеется возможность управлять форматом вывода чисел. Список вывода может содержать не только переменные, но и константы, выражения, вызовы функций.

```
Write(F,RealVar,'номер',RealVar + Cos(5*5));
```

Обмен данными с консолью

В модуле System определены две переменные Input и Output типа TextFile, которые заранее связаны с физическим файлом-устройством **'CON'** – консолью, которое объединяет клавиатуру и дисплей.
Если в процедурах ввода Read (ReadLn) опущено имя файла, то считается, что ввод идет из системного файла Input, что физически означает набор на клавиатуре. Аналогично, если в процедурах вывода данных Write (WriteLn) опущено имя файла, то они выводятся через файловую переменную Output на экран.

```
Write(Output, 23) = Write(23)
```

ТИПИЗИРОВАННЫЕ ФАЙЛЫ

Типизированный файл – это файл, все компоненты которого одного типа, заданного при объявлении файловой переменной. Тип данных может быть любым, кроме файлов, объектов и структурированных компонентов (массивов, записей, и т.д.), содержащих файлы или объекты. Компоненты файла хранятся на диске во внутреннем (двоичном) формате.

Так как каждый компонент занимает в файле одинаковый объем (равный размеру его типа), имеется возможность организовать прямой доступ к каждому из них. Процедура позиционирования

```
Seek(Var F : <файловый тип>; <номер компонента>)
```

смещает указатель файла к требуемому компоненту.
Ввод-вывод данных осуществляется стандартными процедурами Read, Write, параметрами которых кроме файловой переменной являются переменные такого же типа, как и тип файла.

НЕТИПИЗИРОВАННЫЕ ФАЙЛЫ

Нетипизированные файлы также состоят из машинных представлений данных, которые объявлены без указания типа его компонентов. Нетипизированный файл – это высокоскоростной низкоуровневый канал ввода-вывода для доступа к любым файлам с любым типом. С его помощью можно записывать на диск произвольные участки рабочей памяти компьютера и считывать их в память.

Операции чтения и записи осуществляется блоками с помощью процедур BlockRead и BlockWrite.

Глава 10. УКАЗАТЕЛИ И ДИНАМИЧЕСКАЯ ПАМЯТЬ

- **Динамическая память**
- **Типизированные и нетипизированные указатели**
- **Процедуры работы с указателями**
- **Использование динамической памяти для размещения данных большого объема**
- **Динамические структуры**
- **Создание и работа со списком**

ДИНАМИЧЕСКАЯ ПАМЯТЬ, УКАЗАТЕЛИ

В Delphi Pascal есть средства, которые позволяют использовать **динамическую память** (heap-область, куча) под размещение данных большой размерности, при организации динамических структур и для других целей. При этом происходит **динамическое размещение** данных, что означает использование области динамической памяти непосредственно во время работы (при динамическом размещении заранее не известен ни тип, ни количество размещаемых данных). Средство управления динамической памятью – **указатели**.

Указатель – это переменная, которая в качестве своего значения содержит адрес байта памяти. Размер указателя равен 4 байтам.

Типизированный указатель (ссылка) связывается с некоторым типом данных. Для объявления типизированного указателя используется значок ^, который помещается перед соответствующим типом:

```
Var   PInt : ^Integer;  PReal : ^Real;
```

Нетипизированные указатели (указатели) хранят просто адреса, которые не связаны с данными конкретных типов:

```
Var   P : Pointer;
```

Nil – «нулевой» адрес.

РАБОТА С ДИНАМИЧЕСКОЙ ПАМЯТЬЮ

Динамическое распределение памяти происходит следующим образом:

- прикладная программа запрашивает у системы фрагмент памяти определенного размера;
- если в вычислительной машине есть свободная память требуемого объема, то система выделяет этот фрагмент;
- система передает прикладной программе указатель на эту непрерывную область в памяти;
- прикладная программа получает указатель;
- прикладная программа размещает данные в выделенной области, начиная с указанного адреса;
- по окончании работы с этими данными прикладная программа обязана сообщить системе о том, что выделенная область памяти может быть снова получена системой, иначе говоря, освободить выделенный фрагмент.

РАБОТА С ДИНАМИЧЕСКОЙ ПАМЯТЬЮ

Память под динамически размещаемую переменную во время работы программы выделяется процедурой **NEW**. Процедура

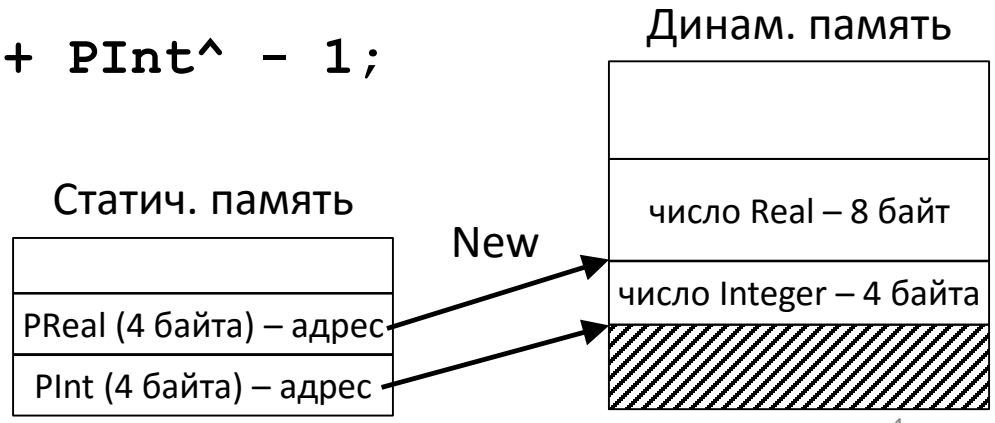
New(Var <типизированный указатель>);

возвращает адрес выделенного участка памяти через параметр-переменную. Размер участка памяти определяется базовым типом указателя.

```
Var
   PInt : ^Integer;
   PReal : ^Real;
Begin
   New(PInt);
   New(PReal);
   PInt^ := 2;
   PReal^ := 2*pi;
   PReal^ := Sqr(PReal^) + PInt^ - 1;
End.
```

```
{New – функция}
 Type TyPInt : ^Integer;
 Var  PInt : TyPInt;…
 PInt := New(TyPInt);…
```

После процедуры New в куче можно разместить значение соответствующего типа. Используется значок ^ – **разыменование указателя**



РАБОТА С ДИНАМИЧЕСКОЙ ПАМЯТЬЮ

Передавать значения между указателями можно в том случае, если они связаны с одним и тем же типом данных (либо один из них – нетипизированный указатель или "нулевой адрес" Nil) .

```
Type
      TType = (red, green, blue);
      PType = ^TType;
Var
      P1,P2 : PType;
      PInt : ^Integer; P : Pointer;
Begin
      New(P1); New(P2);
      P1^ := red; P2^ := green;
      P1 := P2;
      ...
      PInt := P; P := P1; P2 := Nil;
      Writeln(Cardinal(P1));
End.
```

При выполнении операции присваивания теряется участок памяти, на который ссылался указатель, стоящий слева от оператора присваивания. Это типичный случай создания "мусора" (garbage) в динамической памяти.

РАБОТА С ДИНАМИЧЕСКОЙ ПАМЯТЬЮ

Процедура

```
Dispose(<типизированный указатель>);
```

освобождает память по адресу, хранящемуся в указателе. При этом не изменяется значение указателя – текущей границы незанятой динамической памяти. Поэтому чередование обращений к процедурам New и Dispose обычно приводит к **фрагментации** памяти – память разбивается на небольшие фрагменты с чередованием свободных и занятых участков.

При работе с нетипизированными указателями, в основном, используются другие процедуры:

```
GetMem(Var P : Pointer, Size : Word)
                        {резервирование памяти}
FreeMem(P : Pointer, Size : Word)
                        {освобождение памяти}
```

где P – нетипизированный указатель, Size – размер в байтах требуемой или освобождаемой части кучи.

Использование процедур GetMem/FreeMem (как и вообще вся работа с динамической памятью) требует особой осторожности и соблюдения правила: освобождать нужно ровно столько памяти, сколько ее было зарезервировано, и именно с того адреса, с которого она была зарезервирована.

РАЗМЕЩЕНИЕ ДАННЫХ БОЛЬШОГО ОБЪЕМА В ДИНАМИЧЕСКОЙ ПАМЯТИ

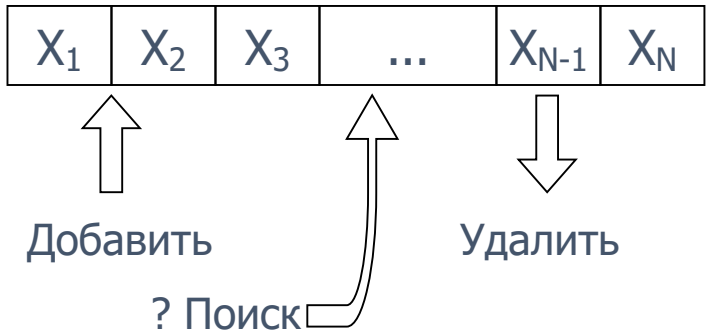
```
Type  Dim100x200 = array [1..100,1..200] of Real;
```

Целесообразно «большие» массивы размещать в динамической памяти.

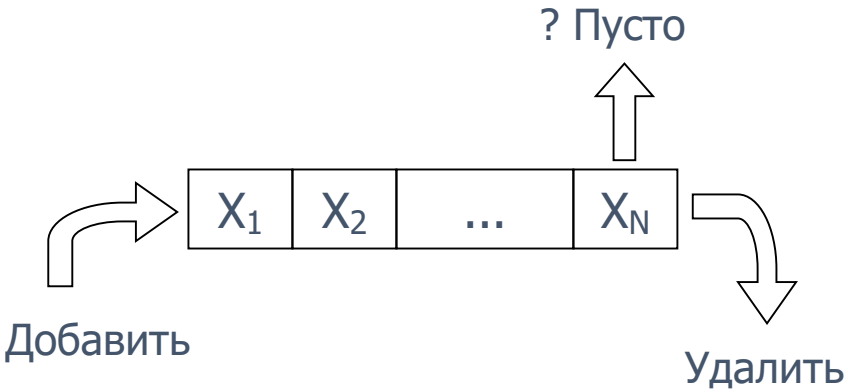
```
Type
   Dim100 = array [1..100] of Real;
            {строка-массив размером 800 байт}
   Dim100Ptr = ^Dim100;
            {указатель на строку размером 4 байта}
   Dim100x200 = array [1..200] of Dim100Ptr
            {массив указателей размером 800 байт}
Var
   D : Dim100x200;    I,J : Byte;
Begin
   for I := 1 to 200 do New(D[i]);
…  D[I]^[J] := 0.5;     {I = 1..200, J = 1..100}
End.
```

ДИНАМИЧЕСКИЕ СТРУКТУРЫ

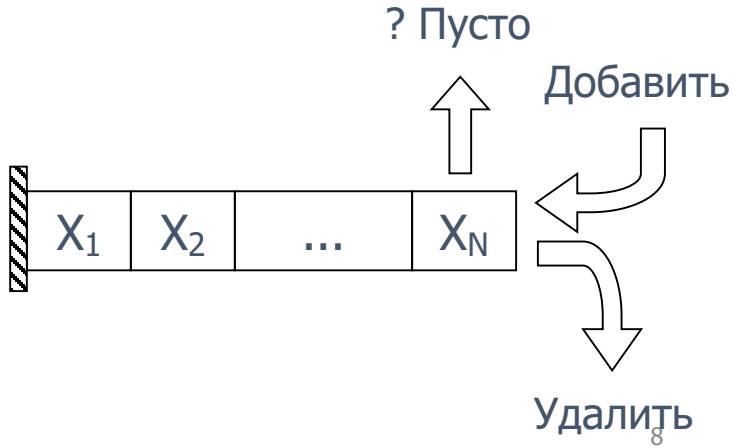
- **Список** – упорядоченный набор элементов одного типа. Размер списка может изменяться. Элемент списка (любой динамической структуры) состоит из двух частей: информационной, содержащей данные, и адресной, где хранятся указатели на соседние элементы.



- **Очередь** – список, в один конец которого элементы добавляются, а из другого изымаются. Очередь – это устройство FIFO (First In, First Out).

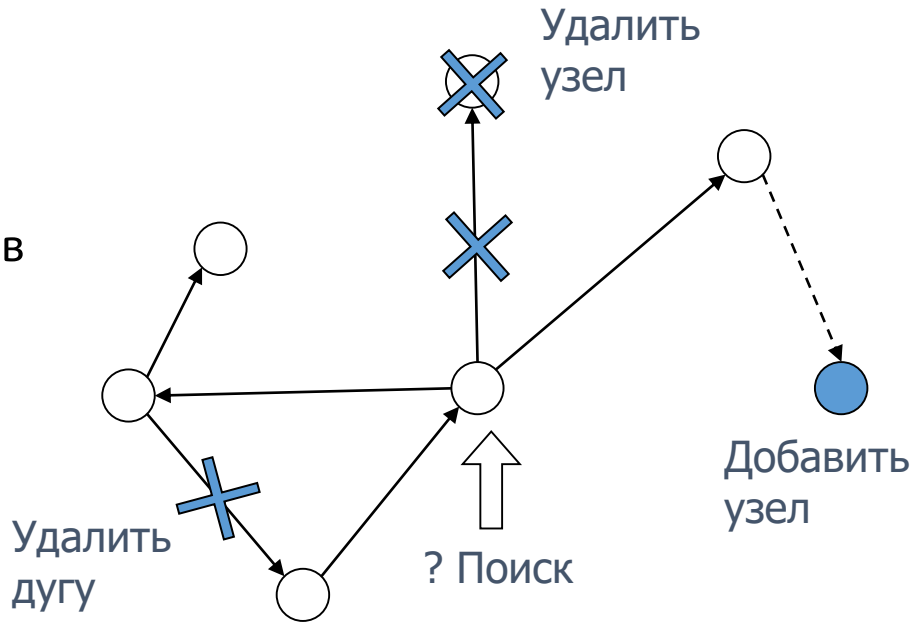


- **Стек** – список, в один конец которого элементы добавляются, и из него же изымаются. Стек – это устройство LIFO (Last In, First Out).

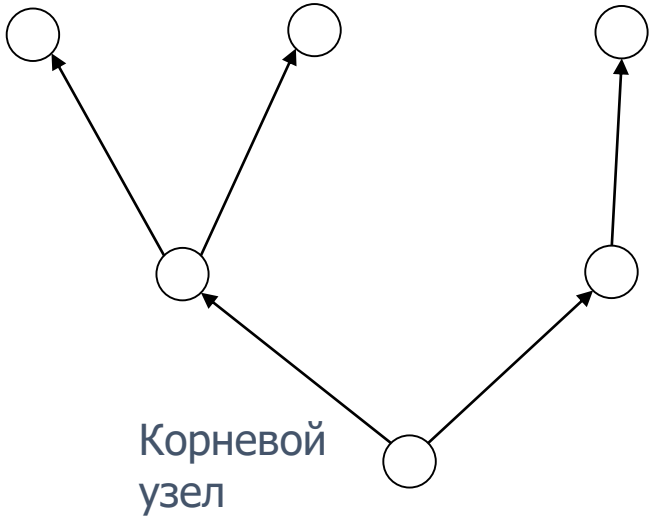


ДИНАМИЧЕСКИЕ СТРУКТУРЫ

- **Граф** – структура, состоящая из узлов и дуг, каждая дуга направлена от одного узла к другому.



- **Дерево** – направленный граф, у которого имеется корневой узел, не имеющий дуг, входящих в него; в каждый узел входит одна дуга; в каждый узел можно попасть из корневого за несколько шагов.



СОЗДАНИЕ ЛИНЕЙНОГО ОДНОСВЯЗНОГО СПИСКА

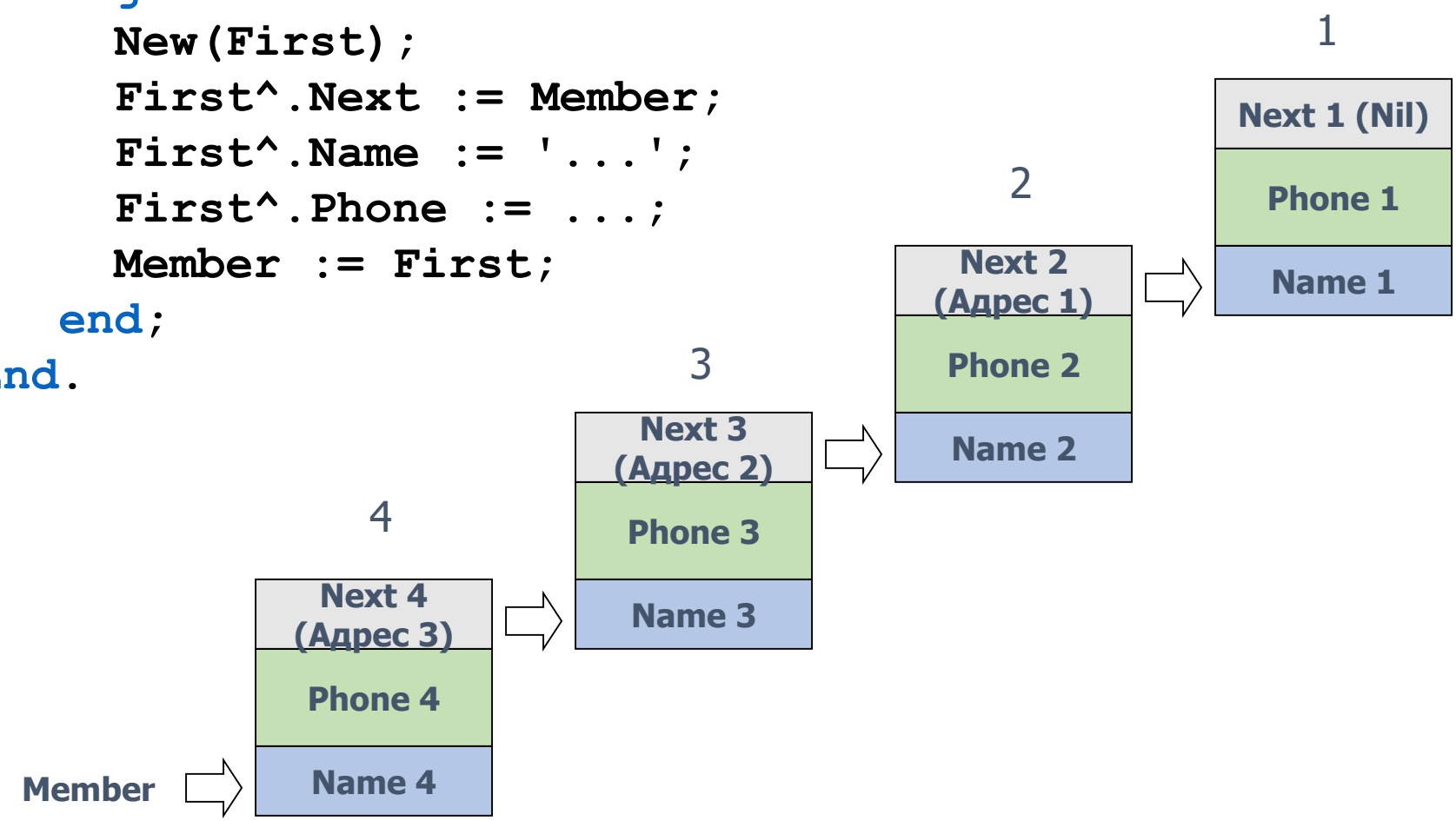
```
Type
    PMemberType = ^MemberType;
    MemberType = record
                    Name : String;
                    Phone : Word;
                    Next : PMemberType
                    end;
Var
    First, Member : PMemberType;
```

Указатель на тип MemberType описан до того, как описан сам тип MemberType. В Delphi Pascal такое предварительное использование идентификатора типа разрешено только при создании указателя на этот тип.

В запись MemberType включено поле Next, которое представляет собой указатель на запись MemberType. Это есть ссылка на соседний элемент списка.

СОЗДАНИЕ ЛИНЕЙНОГО ОДНОСВЯЗНОГО СПИСКА

```
Begin
    Member := Nil; {неопределенный указатель –
                    показывает, что память для Member не выделена}
    for I := 1 to 4 do
    begin
      New(First);
      First^.Next := Member;
      First^.Name := '...';
      First^.Phone := ...;
      Member := First;
    end;
End.
```



УНИЧТОЖЕНИЕ СПИСКА

```
while Member <> Nil do
   begin
      Element := Member^.Next;
      Dispose(Member);
      Member := Element
   end;
```

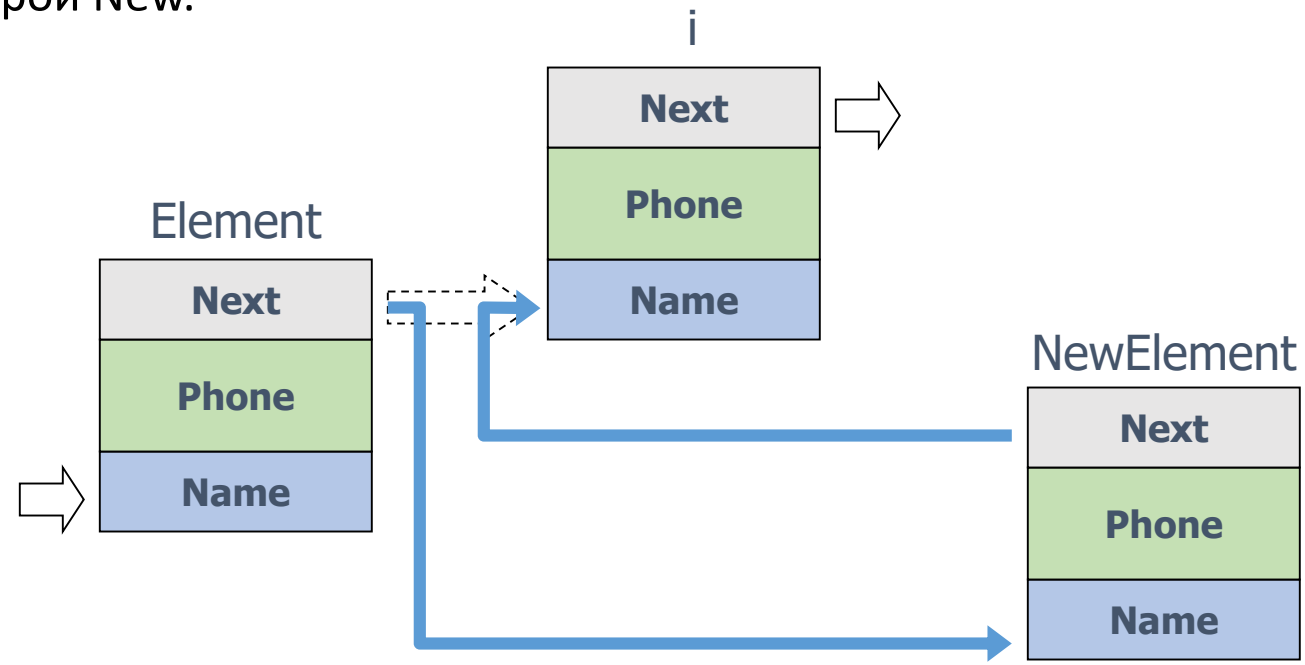
Перед удалением необходимо запомнить ссылку на следующий элемент (в буфер Element), т.к. в противном случае вся информация об оставшемся куске списка будет утеряна. Следует использовать цикл while…do, т.к. заранее не известно число повторений, и кроме того список может быть пустым.

ДОБАВЛЕНИЕ НОВОГО ЭЛЕМЕНТА В СПИСОК

после элемента Element, предварительно найденного по некоторому признаку:

```
NewElement^.Next := Element^.Next;
NewElement^.Name := '...';
NewElement^.Phone := ...;
Element^.Next := NewElement;
```

Предполагается, что NewElement имеет тип PMemberType, и был заранее создан процедурой New.



Для удобства поиска нужного элемента целесообразно формировать список в отсортированном (по некоторому полю) виде.

Глава 11. ОБЪЕКТЫ

- Описание классов, создание объектов
- Ограничение доступа к полям и методам
- Создание библиотек классов
- Реализация принципа наследования
- Полиморфизм
- Раннее связывание
- Позднее связывание, виртуальные методы

ОПИСАНИЕ КЛАССА

Класс – это структурный тип данных, который включает описание полей данных, а также процедур и функций (методы), работающих с этими полями.

```
Type
   PMouse = ^TMouse;
   TMouse = object          {class}
      Visible : Boolean;    {признак включен/выключен}
      Keys : Byte;          {количество кнопок}
      constructor Init;     {конструктор}
      destructor Done;      {деструктор}
      procedure On;         {включить указатель мыши}
      procedure Off;        {выключить указатель мыши}
   end;;
```

Метод Init создает и инициализирует объект – конструктор (описание начинается со слова constructor). Любой класс должен иметь как минимум один конструктор (если у класса есть виртуальные методы).

Обратным конструктору является метод уничтожения объекта Done – деструктор (описывается словом destructor). Класс не может иметь более одного деструктора.

Если объект динамический, то конструктор выделяет память для него в куче, а деструктор – освобождает память.

СОЗДАНИЕ ОБЪЕКТА

Объект – это экземпляр класса (переменная типа object или class). Подобно записи, объект используется с префиксной частью в виде имени этого объекта.

Статический объект:

```
Var
   Mouse : TMouse;
Begin
   Mouse.Init;      {инициализация объекта}
   Mouse.On;        {включить указатель мыши}
   ...
End.
```

Динамический объект:

```
Var
   Mouse : PMouse;      {тип – указатель на TMouse}
Begin
   New(Mouse,Init);       {инициализация объекта}
   Mouse^.On;             {включить указатель мыши}
   ...
   Dispose(Mouse,Done); {уничтожить объект}
End.
```

ОГРАНИЧЕНИЕ ДОСТУПА К ПОЛЯМ И МЕТОДАМ

```
if Mouse^.Keys > 0 then
   begin
      Mouse^.On;
      ...
      Mouse^.Off;
   end;
```

Фрагмент ошибочен с точки зрения объектного подхода, поскольку нарушен принцип **инкапсуляции**. Получать информацию об объекте следует только при помощи операции селектора, т.е. специальным методом.

ОГРАНИЧЕНИЕ ДОСТУПА К ПОЛЯМ И МЕТОДАМ

```
Type
   PMouse = ^TMouse;
   TMouse = object
      constructor Init;
      destructor Done;
      procedure On;
      procedure Off;
      function GetKeys : Byte; {число кнопок у мыши}
      function IsVisible : Boolean; {видна мышь или нет}
   private
      Visible : Boolean;
      Keys : Byte;
   end;
```

Введены два метода-селектора GetKeys, IsVisible для определения состояния объекта.

Сами поля Visible и Keys убраны в специальную частную секцию private. Эта секция делает недоступными для непосредственного использования в программе поля и методы, расположенные в ней.

БИБЛИОТЕКИ КЛАССОВ

При создании библиотек классов они описываются в отдельных модулях, которые могут подключаться к основной программе для использования этих классов. Для соблюдения принципа инкапсуляции, класс описывают в разделе Interface, а его методы – в разделе Implementation, что позволяет "скрыть" внутреннее содержание методов.

```
Unit UMouse;
Interface                    {секция интерфейса}
   Type
      PMouse = ^TMouse;
      TMouse = object
         constructor Init;
         destructor Done;
         procedure On;
         procedure Off;
         function GetKeys : Byte;
         function IsVisible : Boolean;
      private
         Visible : Boolean;
         Keys : Byte;
      end;
```

```
Implementation          {секция реализации}
   constructor TMouse.Init;
      begin
      ...
      end;
   ...
   function TMouse.IsVisible : Boolean;
      begin
      ...
      end;
...
End.
```

```
   Program Example;
      Uses UMouse;    {подключение модуля с классом}
      Var Mouse : PMouse;
      Begin
      ...
      End.
```

НАСЛЕДОВАНИЕ

При наследовании объекты класса-потомка получают возможность использования ("наследуют") полей и методов класса-родителя, что позволяет повторно не определять эти компоненты класса.

```
Program NewMouseDemo;
  Uses UMouse;  {подключаем созданный ранее модуль}
   Type
      PNewMouse = ^TNewMouse;
      TNewMouse = object(TMouse)    {предок - TMouse}
         function GetClick(Var X,Y : Word) : Byte;
      end;
  Var
      Mouse : PNewMouse;
      A,B : Word;
      Button : Byte;
  Function TNewMouse.GetClick(Var X,Y : Word): Byte;
      begin                {функция возвращает номер нажатой
      ...                  кнопки в своем значении и текущие
                           координаты мыши в параметрах-
      end;                 переменных};
```

```
Begin
   New(Mouse,Init);
   if Mouse^.GetKeys > 0 then
      begin                    {проверка, подключена ли мышь}
         Mouse^.On;
         repeat
            Button := Mouse^.GetClick(A,B);
            if Button > 0 then
               Writeln(Button,A,B);
         until Button = 3;
         Mouse^.Off;
      end;
   Dispose(Mouse,Done);
End.
```

Новый класс TMouseNew унаследовал все свойства класса TMouse, т.к. является его потомком. Соответственно, класс TMouse для класса TMouseNew является предком, или родителем. В новом классе описан только один новый метод GetClick для получения информации о нажатой кнопке. При этом все методы класса TMouse становятся доступными для экземпляров класса TMouseNew.

```
Var M1 : PMouse; M2 : PNewMouse; ... M1 := M2 {допустимо}
```

ПОЛИМОРФИЗМ

В объектно-ориентированном программировании полиморфизм проявляется в возможности переопределять методы класса-предка, т.е. расширять свойства объекта путем "перестраивания" его методов.

```
Type
   PPoint = ^TPoint;          {объект – точка на экране}
   TPoint = object
      X, Y : Word;
      Color : Byte;
      constructor Init (InitX, InitY : Word;
                        InitColor : Byte);
      destructor Done;
      procedure Draw;
   end;
   PCircle = ^TCircle;  {объект - окружность на экране}
   TCircle = object(TPoint)  {производный от TPoint}
      Radius : Word;
      constructor Init (InitX,InitY,InitRadius:Word;
                        InitColor : Byte);
      procedure Draw;
   end;
```

ПОЛИМОРФИЗМ

```
Constructor TPoint.Init;    {при создании объекта
   begin                    задаются координаты точки и
      X := InitX;           ее цвет через параметры
      Y := InitY;           конструктора}
      Color := InitColor;
   end;
Destructor TPoint.Done;     {нет операторов, объект
   begin end;          уничтожается с помощью Dispose}
Procedure TPoint.Draw;
   begin
     Writeln('Рисуем точку x,y=',x,y,'Цвет=',Color);
   end;
Constructor TCircle.Init;
   begin
      Radius := InitRadius;
      TPoint.Init(InitX,InitY,InitColor);
   end;
Procedure TCircle.Draw;
   begin
     Writeln('Рисуем окружность x,y,R=',x,y,Radius,
                                        'Цвет=',Color);
   end;
```

ПОЛИМОРФИЗМ

В программе описаны два класса: TCircle и его потомок TPoint. Деструктор Done наследуется, а методы Init и Draw на основе полиморфизма перекрываются.

В конструкторе класса TCircle не только инициализируется новое поле Radius, но и вызывается в явном виде конструктор "родителя". Это делается для того, чтобы проинициализировать те поля, которые "перешли по наследству" этому классу.

Метод Draw класса TCircle описан заново – рисование окружности отличается от рисования точки.

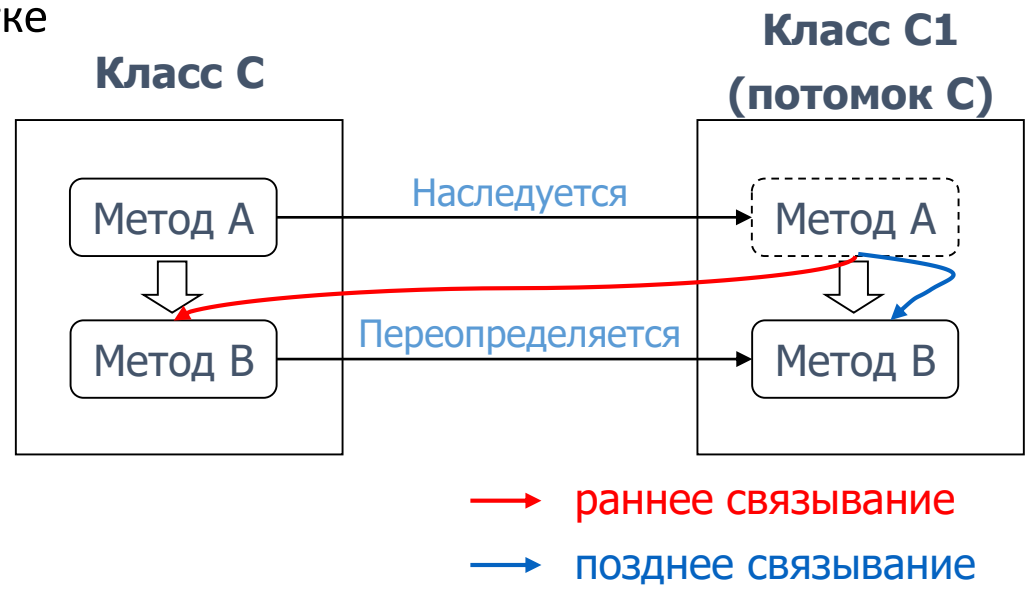
Приведенный пример относится к случаю **простого полиморфизма**, когда при вызове переопределенного метода тип объекта (класс), для которого вызывается данный метод, точно известен. При этом адреса методов-процедур, которые ассоциируются с именем Draw, и где размещаются команды этих методов, подставляются в вызовы данных методов на этапе компиляции. Такой процесс называется **ранним связыванием** – т.е. включением в машинный код точных адресов на те точки программы, с которых начинается размещение кода метода (такой метод называется **статическим**).

```
Var
   Point : PPoint;  Circle : PCircle;
Begin
... Point^.Draw; Circle^.Draw; ...
End.
```

ПОЛИМОРФИЗМ

Действия компилятора при обработке статических методов объектов, связанных иерархически:

1. При вызове метода компилятор устанавливает тип объекта, вызывающего метод.
2. Установив тип, компилятор ищет метод в пределах типа объекта. Найдя его, компилятор назначает вызов этого метода.



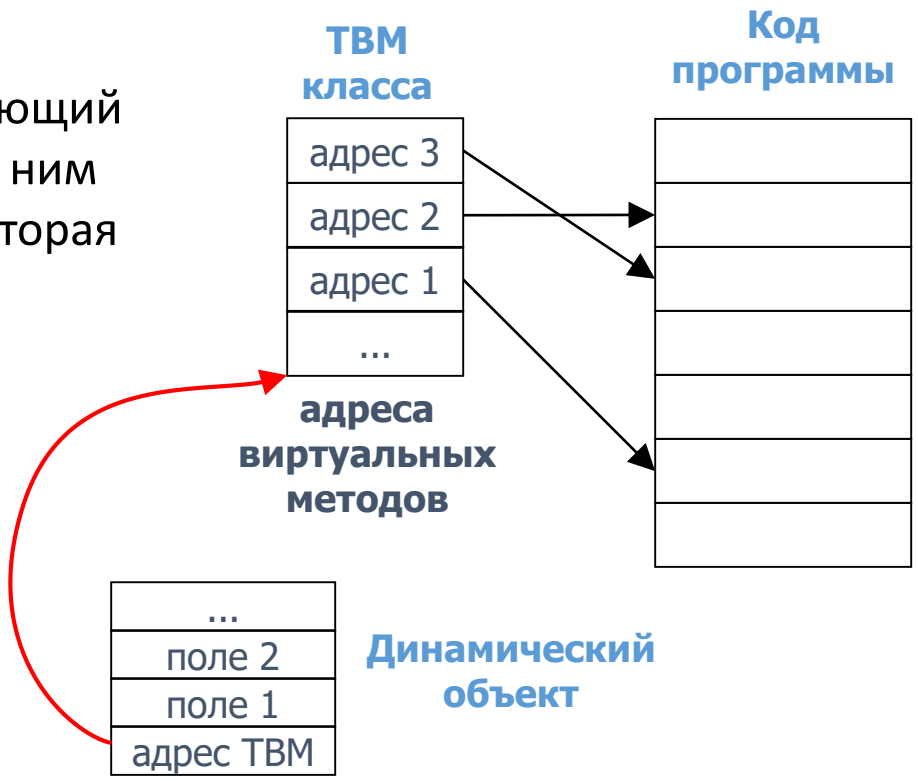
3. Если метод не найден, то компилятор начинает рассматривать тип непосредственного прародителя и ищет метод, имя которого вызвано, в пределах родительского типа. Если он найден, то вызов заменяется на вызов метода родителя. Если нет, то компилятор "поднимается" еще на один уровень и т.д. При этом, если метод родителя вызывает другие методы, то последние также будут методами родителя, даже если потомки имеют свои собственные методы.

Для стандартизации работы с объектами, имеющими полиморфные методы, введено понятие **позднего связывания**, когда компилятор формирует косвенное обращение к адресу ТВМ. Реальный адрес нужного метода не известен до момента выполнения программы. Этот метод подключается в процессе выполнения программы, когда однозначно определен тип объекта (класс).

ПОЛИМОРФИЗМ

Замещаемые (полиморфные) методы объектов, для которых необходимо реализовать механизм позднего связывания, называют **виртуальными** и отмечают в описании объекта стандартной директивой virtual. Одни и те же виртуальные методы должны иметь одинаковые заголовки у всех объектов данной иерархии. Метод, объявленный виртуальным в некотором классе, обязан быть виртуальным и во всех его наследниках. Использование механизмов позднего связывания возможно как для статических, так и динамических объектов (размещенных в динамической памяти).

Каждый класс, содержащий или наследующий виртуальные методы, имеет связанную с ним **таблицу виртуальных методов** (ТВМ), которая размещается в статической памяти.
В этой таблице содержится, в частности, информация об явных адресах всех виртуальных методов данного класса.
При создании объекта (после выполнения метода-конструктора) в него добавляется специальное поле – указатель на ТВМ данного класса.

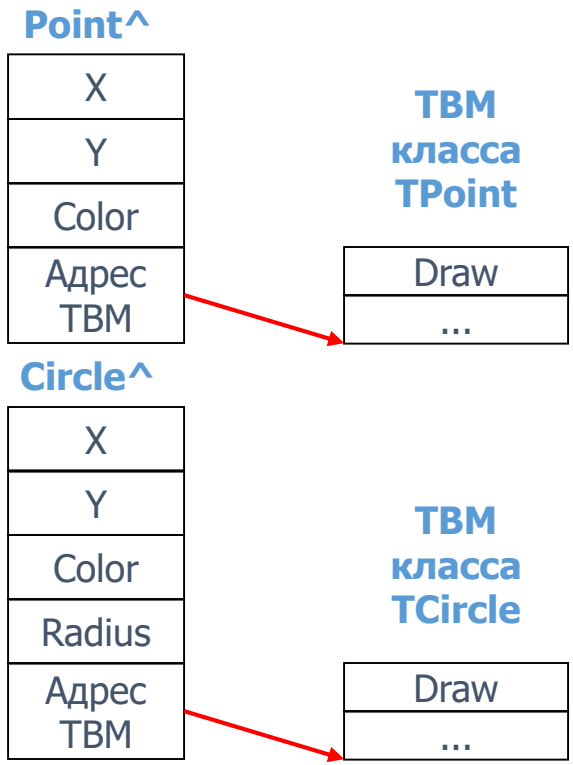


ВИРТУАЛЬНЫЕ МЕТОДЫ

```
Type
   PPoint = ^TPoint;
   TPoint = object
      X, Y : Word;
      Color : Byte;
      constructor Init (InitX, InitY : Word;
                        InitColor : Byte);
      destructor Done;
      procedure Draw; virtual;
   end;
   PCircle = ^TCircle;
   TCircle = object(TPoint)
      Radius : Word;
      constructor Init (InitX,InitY,InitRadius:Word;
                        InitColor : Byte);
      procedure Draw; virtual;
   end;
   ...
   {описания методов - прежние}
   ...
```

ВИРТУАЛЬНЫЕ МЕТОДЫ

```
Var
    Point : PPoint;
    Circle : PCircle;
    P : array[1..2] of PPoint;
    i : Byte;
Begin
    New(Point,Init(…));
    New(Circle,Init(…));
    P[1] := Point;
    P[2] := Circle;
    for i := 1 to 2 do P[I]^.Draw;
End.
```



Тип обеих переменных Point и Circle – указатель, поэтому можно ввести массив указателей. Присваивание P[2] := Circle корректно с точки зрения совместимости типов – все объекты-потомки совместимы со своими предками (обратное не верно).

Если бы метод Draw был статическим, то для рисования окружности потребовался бы, например, вызов: PCircle(P[2])^.Draw

Удобство использования виртуальных методов становится более заметно при увеличении числа и усложнении иерархии объектов.