Языки программирования и методы трансляции

(конспект лекций для студентов 3-го курса ИВМиИТ КФУ, весна 2013)

Плещинский Н.Б.

Предисловие

Данный текст – конспект лекций по курсу «Языки программирования и методы трансляции», которые я читал в 2008-2013 гг. студентам факультета ВМК Казанского государственного университета и сейчас – Института вычислительной математики и информационных технологий Казанского федерального университета. При переходе на бакалавриат этот общий курс изъят из учебного плана. Трансляторы сейчас редко кто пишет, так что это ремесло (или искусство) вполне может быть переведено на самостоятельное изучение. Но понимание того, как исходный код превращается в машинные команды, остается полезным для современного программиста. Ну а умение программировать остается востребованным, языки программирования нужно изучать …

PNB

Введение

ЯПиМТ – дисциплина с шифром ОПД.Ф.05 из (старого) учебного плана подготовки по специальности «Прикладная математика и информатика». Государственный образовательный стандарт (ГОС) предусматривал изучение следующих разделов:

- *основные понятия языков программирования*
- *синтаксис, семантика, формальные способы описания языков программирования*
- *типы данных, способы и механизмы управления данными*
- *методы и основные этапы трансляции*
- *конструкции распределенного и параллельного программирования*

Курс построен на следующем принципе: обсуждать идеи и разбирать примеры.
Автор надеялся, что конспект лекций избавит слушателей от утомительной необходимости переписывать с экрана или с доски на бумажный носитель полезную информацию, а особенно фрагменты программ. Но напрасно надеялся … ☺

Рекомендуемая основная литература
(существенно использовалась при подготовке курса лекций)

[1] Опалева Э.А., Самойленко В.П. Языки программирования и методы трансляции. – СПб.: БХВ-Петербург, 2005.
[2] Карпов Ю.Г. Теория и технология программирования. Основы построения трансляторов. – СПб.: БХВ-Петербург, 2005.
[3] Вирт Н. Алгоритмы и структуры данных. – М.: Мир, 1989.
[4] Лавров С.С. Программирование. Математические основы, средства, теории. – СПб.: БХВ-Петербург, 2001.
[5] Хантер Р. Проектирование и конструирование компиляторов. – М.: Финансы и статистика, 1984.

1. Проблема трансляции

Процессор компьютера работает с двоичными кодами (выполняет машинные команды). Программировать в машинных кодах трудно. Обычно программы для компьютера (*исходный код*) пишут на языках программирования высокого уровня, более близких к предметной области решаемых задач. Специальные программы – *трансляторы* – должны переводить исходный код в *машинный код*.

При грубой классификации ЯП используют два понятия: *низкий уровень* – конструкции языка близки к машинному коду (командам компьютера) и *высокий уровень* – конструкции языка близки к понятиям исходной задачи. Термин *средний уровень* также имеет смысл.

Языком программирования (ЯП) можно называть любую систему обозначений и понятий для описания *структур данных* и *алгоритмов* [1]. Не случайно первое издание книги [3] имело название «Алгоритмы + структуры данных = программы».

Языки программирования, которые используются при написании *программ* для компьютеров (или, как говорили раньше, для ЭВМ – электронных вычислительных машин), – *формальные искусственные языки,* они имеют алфавит и синтаксис, а также семантику.
Алфавит – набор символов, которые разрешено использовать.
Синтаксис – система правил, по которым записываются конструкции языка.
Семантика – набор правил, на основе которых следует истолковывать эти конструкции.

Обсудим коротко, что представляет собой *компьютер*. Главное с точки зрения программиста: имеется *система команд*, при выполнении которых производятся *операции над данными* (или какие-либо иные действия).

Историческая справка ([+] *www.computer-museum.ru*)

В 1949-52 гг были построены независимо друг от друга три ЭВМ:

- ЭДСАК, Морис Уилкс (Великобритания, 1949)
- МЭСМ, Сергей Лебедев (СССР, 1951)
- ЭДВАК, Джон Моучли, Преспер Эккерт, Джон фон Нейман (США, 1952)

Работы по проектированию первой отечественной ЭВМ МЭСМ начались в 1947 г. в Институте электроники АН Украины под рук. С.А.Лебедева, в ноябре 1950 состоялся пробный пуск, 25 декабря 1951 года машина сдана в эксплуатацию. Система команд МЭСМ – трехадресная, всего 13 команд, 50 операций в секунду (тактовая частона 5 кГц), в составе АУ имелся сумматор и два регистра, машинное слово – 17 двоичных разрядов, ОЗУ: 31 число и 63 команды (столько же ДЗУ); позже были добавлены ВЗУ на магнитном барабане (5 тысяч слов), на магнитной ленте; ввод данных производился с перфокарт или с пульта, вывод – на печатающее устройство или на фотопленку; в конструкции ЭВМ имелось 2500 триодов и 1500 диодов (электронные лампы!), потребляемая мощность 25 квт, машина занимала площадь 60 кв.м.

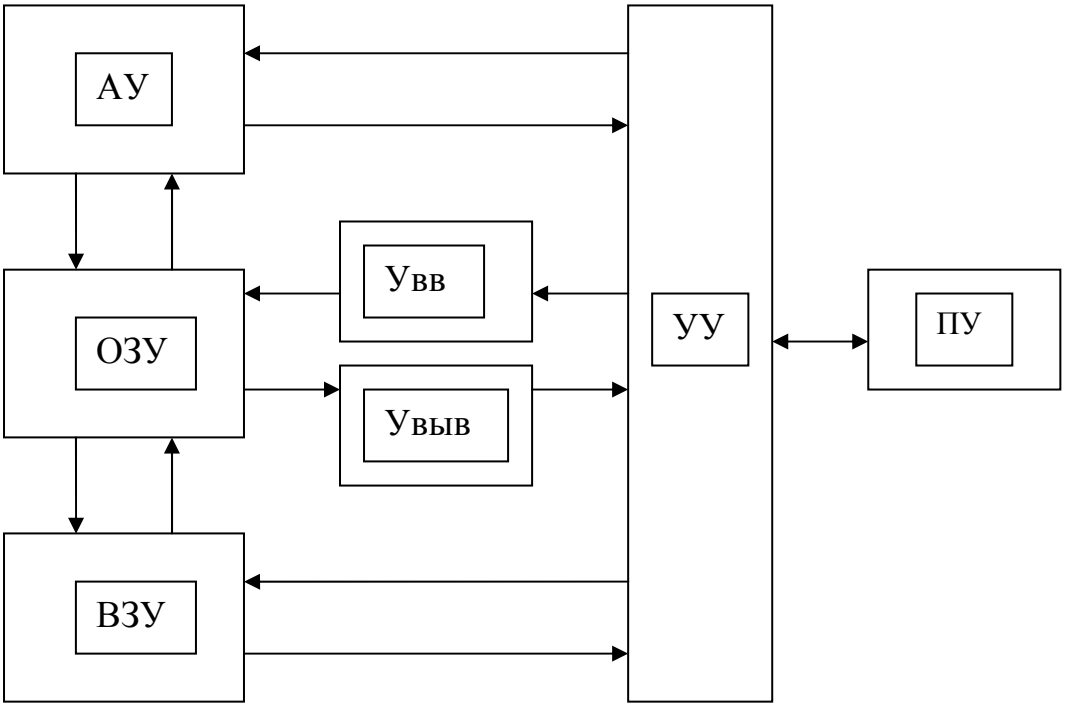
В Казанском университете первая ЭВМ «Урал-1» появилась в 1958 году, она была установлена в здании геологического факультета. Компьютер был способен выполнять 100 операций в секунду над 9-разрядными числами и имел память объемом 4 тысячи слов. В 1960 году Казанский завод ЭВМ выпустил первые две ЭВМ М-20 (20 тысяч операций в

секунду). Университет получил доступ к такой машине в 1964 году. Позднее в здании ВЦ КГУ на ул. Попова были установлены ее модификации.

ЭВМ М-20 была разработана под руководством С.А. Лебедева в ИТМиВТ АН СССР (совместно с СКБ-245, завод САМ). Казанский завод ЭВМ в 1960-64 гг. выпустил 63 таких машины. Обрабатывались числа с плавающей запятой, 45 двоичных разрядов. ОЗУ на ферритовых сердечниках, 4096 слов, 3 магнитных барабана, 4 накопителя на МЛ. Ввод и вывод с перфокарт, печатающее устройство 15 строк в секунду. В конструкции использовалось 4500 электронных ламп и 35000 полупроводниковых диодов.

Первые компьютеры были построены с использованием следующих идей:

- имеются арифметическое устройство, запоминающее устройство, устройство управления, устройства ввода и вывода,
- используется двоичная система счисления,
- выполняются арифметические и логические операции,
- память организована по иерархическому принципу,
- вычисления проводятся в соответствии с программой,
- программа хранится в памяти.



Принято говорить, что основанные на этих принципах компьютеры – реальные или виртуальные – имеют *архитектуру фон Неймана*. Принцип их работы следующий. Данные и команды хранятся в общей памяти. Устройство управления (УУ) анализирует очередную команду, переносит в арифметическое устройство (АУ) из памяти нужные значения и сообщает, какую операцию необходимо выполнить. Результат сохраняется в памяти (или, может быть, ожидает следующей операции в АУ).

Верно ли, что компьютерная программа представляет собой последовательность команд? Действительно ли программа управляет работой компьютера?

Система команд **виртуальной трехадресной машины ВМ-1** построена по образцу машинных кодов для ЭВМ М-20. Каждая команда состоит из кода операции (в нашем примере – две десятичные цифры) и трех адресов ячеек памяти (по три цифры). В некоторых случаях не все адреса используются.

[+] Ляшенко В.Ф. Программирование для ЦВМ с системой команд типа М-20. – М.: Сов. радио, 1971.

00 A1 A2 A3	засылка числа вида A1.A2 в ячейку A3
01 A1 A2 A3	сложение: A3=A1+A2
02 A1 A2 A3	вычитание
03 A1 A2 A3	умножение
04 A1 A2 A3	деление
07 A1 000 A3	извлечение корня
10 A1 000 A3	ввод данных
20 A1 000 A3	вывод данных
77 000 000 000	конец программы

Разумеется, желательно иметь также команды условного и безусловного перехода, команды пересылки информации (обмен с внешними устройствами), и так далее.

Задача 1. Решить уравнение $ax^2 + bx + c = 0$. Составить программу в машинных кодах виртуальной машины.

Задача 2. Сформулировать принципы перевода на машинный язык исходного кода вида (0–b+sqrt(b*b–4*a*c))/(2*a)

Последовательность действий может быть, например, такова:
1) найти в исходном коде значения констант и имена переменных, заменить их адресами ячеек памяти (построить таблицы распределения памяти), внести в машинный код команды засылки значений констант;
2) блоки вида *адрес - знак операции - адрес* заменить адресами ячеек памяти (слева направо, с соблюдением приоритета, возможно, в несколько проходов) и внести в машинный код соответствующие команды; при этом нужно удалять парные скобки вокруг адресов.

Существенно то, что текст программы просматривается не один раз, а до тех пор, пока не будет полностью переработан.

Ясно, что можно написать программу короче и эффективнее, если не использовать эти правила. Одна простая идея: понять, что должно быть сделано, и написать программу заново, но на машинном языке. Анализ и синтез! Можно экономить память компьютера, а можно сокращать время работы программы …

Проблема трансляции состоит в следующем**:** какими должны быть языки программирования и какими должны быть правила преобразования программ на алгоритмическом языке в машинные команды?

При разработке конструкции другой отечественной ЭВМ – машины М-1 под руководством И.С. Брука (1950-51 гг) была использована двухадресная система команд (предложил это математик Ю. Шрейдер). Договорились, что в команде указываются адреса операндов, а результат сохраняется всегда, например, по первому адресу.

Интересно, что авторское свидетельство на конструкцию ЭВМ И.С. Брук и Б.И. Рамеев получили еще в 1948 году. Машина была построена на деталях с трофейных военных складов. Оперативная память – на электронно-лучевых трубках, всего 8 трубок, на каждой 32 строки по 25 точек (вторичная лучевая эмиссия!).

Если на основе этого принципы переделать виртуальную машину ВМ-1 в ВМ-2, то новая система укороченных команд

30 A1 A2	засылка целой части числа в A2
31 A1 A2	засылка дробной части числа в A2
01 A1 A2	сложение: A1=A1+A2
02 A1 A2	вычитание
03 A1 A2	умножение
04 A1 A2	деление
07 A1 000	извлечение корня
10 A1 A2	ввод данных
20 A1 A2	вывод данных
77 000 000	конец программы

В дальнейшем была высказан принцип: заменять истинные адреса ячеек памяти на условные обозначения и использовать символические имена для обозначения операций. Такую форму записи команд стали называть *мнемокодом*.

Писать программы на мнемокоде стало существенно легче и быстрее, чем в машинных командах. Но при этом необходимо иметь *Ассемблер* – специальную программу, переводящую программы из мнемокода в машинные коды.

Рассмотрим **виртуальную машину ВМ-3**. Предусмотрим в ее конструкции специальную ячейку памяти R – *регистр*. Содержимое регистра используется как первый операнд в арифметических командах, результат операций также сохраняется в регистре.

Система команд ВМ-3 в мнемокоде может быть такой:

INP p.q	R=p.q
MOV R, A	R=A
MOV A, R	A=R
ADD A	R=R+A
SUB A	R=R-A
MUL A	R=R*A
DIV A	R=R/A
SQRT	R=√R
OUT A	вывод данных
END	останов

Задача 3. Предложить алгоритм перевода в мнемокод операторов присваивания типа x1=(-b+sqrt(b^2-4*a*c))/(2*a)

Еще одна идея: завести специальный регистр для хранения адреса команды, которая должна выполняться следующей. Тогда команды условного и безусловного перехода должны всего лишь изменять содержимое этого регистра.

Но и при программировании на языке Ассемблера производительность труда программиста не слишком высокая. В языках программирования высокого уровня при кодировании алгоритмов используются операторы языка – конструкции, записанные в более близком к естественному языку виде, которые заменяют, как правило, достаточно большие последовательности машинных команд. Преимущество алгоритмических языков очевидно – легко понять по исходному коду, какая задача решается.

Как уже было сказано, возможность программировать на языках высокого уровня также поддерживается специальными программами – *трансляторами*, которые преобразуют текст на алгоритмическом языке в последовательность команд компьютера или, может быть, в программу на другом языке более низкого уровня. Трансляторы пишут одновременно с разработкой языка программирования высокого уровня (возможно, в нескольких вариантах). При этом, разумеется, учитываются возможности оборудования.

Трансляторы подразделяются на *компиляторы* и *интерпретаторы*. В первом случае исходный код целиком переводится в машинные команды. Во втором – транслируются его фрагменты (отдельные операторы) и сразу передаются на выполнение. Возможен более сложный процесс. Например, перевод исходного текста на промежуточный язык для платформы .NET . Используются термины: *исходный код*, **промежуточный код,** *объектный код*, *машинный код*.

Требования, предъявляемые к транслятору (требования противоречивые):

- любой правильный исходный код должен преобразовываться в эквивалентный машинный код;
- ошибки в программе обнаруживаются, комментируются (и даже, возможно, исправляются);
- транслятор имеет ограниченный объем и работает быстро при скромных системных ресурсах;
- машинный код имеет ограниченный объем и работает быстро.

Современные компиляторы, как правило, *оптимизирующие*.

Хорошая идея: написать компилятор для языка программирования на этом же языке.

2. Язык Ассемблера

Программирование на языке Ассемблера широко использовалось в эпоху ЭВМ третьего поколения. Наиболее распространенными тогда были машины семейства IBM 360/370 (выпускались с 1965 г. в различной комплектации) и совместимые с ними.

Настольная книга программиста прошлых лет

[+] Джермейн К. Программирование на IBM / 360. – М.: Мир, 1978.

Многие принципы работы этих машин дошли до наших дней. Отметим, что именно тогда элементарной ячейкой памяти компьютера стал *байт*, а для кодирования символов был принят стандарт ASCII. (Правда, у айбиэмовских машин использовалась и уникальное кодирование EBCDIC.) Длина машинного слова – 4 байта.

В составе арифметического устройства

- шестнадцать регистров 0..F общего назначения длиной в слово,
- четыре регистра 0,2,4,6 с плавающей точкой длиной в два слова,
- различные регистры управления.

Предусмотрены команды, относящиеся к четырем группам: ввод-вывод, обращение к памяти, арифметические операции и команды управления. Оперативная память до 16 Мб была построена на ферритовых сердечниках, а регистры – на быстродействующих электронных схемах.

Команды системы IBM 360/370 – двухадресные. Код операции – 1 байт, всего 143 различные команды. Хотя бы один операнд – регистр. При адресации ячеек памяти используется сложная система *индексирования*: истинный адрес вычисляется по формуле E=(X)+(B)+D – складываются числа из индексного регистра X, из базового регистра B и непосредственно указанное в команде D. Длина короткой команды вида КОП-регистр-регистр составляет два байта (четыре шестнадцатеричные цифры, а длинной команды вида КОП-регистр-XBDDD – четыре байта.

Приведем конкретные примеры.
18 A5 – пересылка из регистра 5 в регистр A (мнемокод L A,5).
58 A3 B0 07 – пересылка из памяти начиная с байта (3)+(И)+007 в регистр A (мнемокод L A,3,B,007)
1A 37 – сложение содержимого регистров 3 и 7, результат в регистре 3 (мнемокод A 3,7)

Структура команды перехода: BC M X B D, код операции 47.
Параметр M определяет условие перехода. Если M=1111, то имеем команду безусловного перехода. В общем случае переход зависит от того, как завершилась предыдущая операция. Например, для сложения в специальном регистре «признак результата» хранится код: 0 – результат был равен нулю, 1 – результат отрицательный, 2 – результат положительный, 3 – произошло переполнение… Если этот код совпадает со значением M, то управление передается по указанному в команде адресу. В ином случае выполняется следующая команда.

На машинах системы IBM 360/370 использовались также алгоритмические языки программирования Fortran, Cobol, PL/1, Algol-60 (см. следующий раздел). В программном обеспечении фирма поставляла трансляторы разного уровня оптимизации.

В СССР было решено (правильно или нет – есть разные мнения) массово производить ЭВМ, программно совместимые с семейством IBM 360/370.
Первая машина ЕС-1020 (20 тыс. операций в секунду, главный конструктор Пржиялковский) выпущена в 1971 г. в Минске. С 1973 г. Казанский завод ЭВМ выпускал ЕС-1030 (100 тыс. операций в секунду). В Москве и Пензе было налажено производство ЭВМ ЕС-1050. Машина ЕС-1033 четвертого поколения выпускалась на Казанском заводе (главный конструктор Гусев).

В 1970-е годы на смену *мэйнфреймам* (IBM 360/370, ЕС ЭВМ и так далее) пришли *персональные компьютеры*. Первый персональный компьютер IBM PC появился в августе 1981 г., без жесткого диска, с процессором 8088 (упрощенная 8-битная версия 8086). Операционная система MS DOS была заказана у фирмы Microsoft. Все следующие модели персоналок изготавливались как программно совместимые с первой моделью.
В 1984 г. на основе процессора 80286 была выпущена модель IBM AT, столь же мощная, как «большие» компьютеры. Процессор поддерживал два режима: реальный и защищенный. Стала реальной *многозадачность*.

С 1989 г. выпускаются персоналки на 32-битный процессорах 486-й серии с адресацией памяти до 4 Гб. Первый Pentium появился в 1993 году.
В СССР микропроцессоры делали, но под большим секретом:
КР580ИК80 – копия 8080,
К1810ВМ86 – совместимый с 8086,
К1801 – в те годы не имел аналогов за рубежом …

В Интернете можно найти массу информации по микропроцессорам и микропроцессорной технике. Например,
[+] Скробов А.Л. Микропроцессоры 1970-х – 1990-х годов: архитектура и эволюция (реферат по информатике).
Хороший справочник по системе машинных команд
[+] Irvin K.P. Assembly language for Intel-based computers. 2003. IA-32 Processor Architecture.

Первый процессор 4004 фирмы Intel (1970 г.) был 4-х битным, в 1972 г. появился 8-ми битный процессор 8008, а в 1978 г. был выпущен 16-разрядный процессор 8086, его функции унаследовали все следующие поколения процессоров фирмы Intel.
Адресная шина процессора 8086 имеет 20 разрядов, поэтому возможна прямая адресация памяти до 1 Мбайта. Процессор выполняет 98 команд, среди них 19 для передачи данных, 38 для обработки данных, 24 команды перехода и 17 команд управления. Каждая команда имеет не более двух операндов, хотя бы один из них – регистр. Современные микропроцессоры интеловского семейства (и 32-х разрядные, и 64-х разрядные, и многоядерные…) выполняют порядка четырех тысяч команд.

Регистры процессора 8086

AX (AH AL)	аккумулятор
BX (BH BL)	базовый регистр
CX (CH CL)	счетчик
DX (DH DL)	регистр данных
SP	указатель стека
BP	указатель базы
SI	индекс источника
DI	индекс приемника
CS	сегмент команд
DS	сегмент данных
SS	сегмент стека
ES	дополнит. сегмент
IP	указатель команд
flags	флаги

Регистры EAX, EBX,… микропроцессора 486 стали 32-разрядными. Но аппаратно поддерживаются и их младшие части – регистры AX, BX,…Добавлены также новые регистры.

Команды процессора пишутся в шестнадцатеричном коде, их длины – байт, два или три. Мнемокод языка Ассемблера читается существенно проще. Для обозначения команд используются такие сокращения:

арифметические команды
ADD SUB MUL DIV MOD SHL SHR
логические команды
AND OR XOR NOT
операции отношения
EQ NE LT GT LE GE
команды, возвращающие значения
SEG OFFSET
команды присваивания атрибута
PTR HIGH LOW

Самые нетерпеливые могут в системе программирования Turbo Pascal прогнать следующую программу, которая находит максимальный элемент в массиве. Тело первой функции написано в машинных кодах (каждая строка – одна команда), а вторая – на языке встроенного Ассемблера.

```
program AssMax;
const
  N = 7;
  Arr: array[1..N] of Integer = (1,2,3,2,17,7,2);

function Max1(var Arr; N: Integer): Integer;
  inline(
  $59/
  $5E/
  $1F/
  $33/$C0/
  $BB/$8001/
  $3B/$C8/
  $7E/$09/
  $AD/
  $3B/$C3/
  $7E/$02/
  $8B/$D8/
  $E2/$F7/
  $8B/$C3
  );

function Max2(var Arr; N: Integer): Integer;
  begin
    asm
    LDS SI,Arr                адрес  Arr  в  SI
    MOV CX,N                  значение  N  в  CX
    LODSW                     слово  в  AX , адрес  DS : SI ,  и  SI – 2
    MOV BX,AX                  AX  в  BX
    MOV AX,1                   1  в  AX
    CMP CX,AX                  сравнить,  AX – BX  ( флаг  SF )
    JLE @3                    перейти,  если  <= 0
    SUB CX,1                  CX – 1
@1: LODSW                      слово  в  AX ...
    CMP AX,BX                  сравнить ...
    JLE @2                     перейти ...
```

```
   MOV BX,AX                    AX  в  BX
@2: LOOP @1                     CX – 1 ; если  CX <> 0, то перейти
@3: MOV @Result,BX
   end
 end;
```

3. Обзор языков программирования

Сегодня, если учитывать *версии и диалекты*, насчитывается порядка двух тысяч языков программирования. Около ста названий можно получить из Интернета, если в поисковой системе набрать словосочетание «обзор языков программирования» и перейти по нескольким первым ссылкам. Перечислим наиболее широко известные ЯП.

FORTRAN (Formula Translator)
Первый язык программирования высокого уровня, разработан в 1954 г. фирмой IBM (США), первый транслятор появился в 1957 году. Язык предназначен для программирования научно-технических (вычислительных) задач. За многие годы его эксплуатации разработаны обширные библиотеки программ. Известно много версий и диалектов.
Фортран жив! Некоторые версии современного Фортрана используется при работе на параллельных суперкомпьютерах. Например, в РФЯЦ-ВНИИЭФ.
Пример. Программа на Фортране.

```
C   УПОРЯДОЧИВАНИЕ ЭЛЕМЕНТОВ МАССИВА
      DIMENSION K(20)
      READ (1,10) K
  10  FORMAT  (20I5)
      DO 20 I=1,19
      M=I+1
      DO 20 J=M,20
      IF (K(I)-K(J)) 20,20,30
  30  N=K(I)
      K(I)=K(J)
  20  K(J)=N
      WRITE (2,50) K
  50  FORMAT (‘МАССИВ K   ‘, 20I5/3X)
      STOP
      END
```

Достаточно знать хотя бы основы одного из современных универсальных языков программирования, чтобы понять, какой алгоритм здесь записан.
Вариант с подпрограммой может иметь такой вид:

```
      DIMENSION K(20)
      READ (1,10) K
  10  FORMAT  (20I5)
      CALL ORD(K,20)
      WRITE (2,50) K
  50  FORMAT (‘МАССИВ K   ‘, 20I5/3X)
      STOP
      END
```

```
SUBROUTINE ORD(L,N)
      DIMENSION L(N)
      DO 20 I=1,19
      M=I+1
      DO 20 J=M,20
      IF (L(I)-L(J)) 20,20,30
  30  N=L(I)
      L(I)=L(J)
  20  L(J)=N
      RETURN
      END
```

Здесь версия языка соответствует стандарту программного обеспечения компьютеров серии ЕС ЭВМ. Следует отметить, что эти машины работали в многозадачном режиме под управлением *операционной системы*. К тексту программы нужно было добавить несколько строк, написанных на *языке управления заданием*: где взять исходный текст, какой использовать компилятор, какой выделить объем памяти, нужны ли магнитные диски и магнитные ленты, куда отправить результат и т. д. Подробности можно найти в книге

[+] Фортран ЕС ЭВМ / Брич З.С., Капилевич Д.В., Котик С.Ю., Цагельский В.И. – М.: Статистика, 1978.

Кстати, до тех пор, пока не появились терминалы удаленного доступа, задания готовились на перфокартах. Текст программы писали на специальных бланках по 80 символов в строку. Первые позиции отводились под метки, операторы начинались с 6 позиции. Пробивать перфокарты разрешалось только специально обученным людям. Нетерпеливые рядовые программисты прорезали новые дырочки бритвенным лезвием и искусно затыкали лишние.

COBOL (Common Business Oriented Language = общепринятый деловой язык)
Появился в США в1958 г. Он ориентирован на решение задач обработки данных в учетно-экономической и управленческой области. Используется и сегодня.

ALGOL (Algorithmic Language)
Первое сообщение о языке, разработанном коллективом специалистов, появилось в 1960 г. Алгол широко использовался раньше, иногда применяется и сейчас. На основе Алгола-60 построено большое семейство языков программирования. Один из самых мощных его потомков – **Algol-68**. Полноценный компилятор создать не успели. Развитие этого языка было приостановлено в связи с появлением персональных компьютеров: для первых персоналок он был слишком сложен, а потом появились новые ЯП. Ближайший потомок Алгола – язык Паскаль.

PL/1 (Programming Language One)
Первая публикация относится к 1963 г. Фирма IBM планировала широко распространить этот универсальный язык программирования, но не получилось. Можно указать два недостатка языка: во-первых, он сильно привязан к архитектуре ЭВМ системы IBM 360/370 и, во-вторых, разрешает слишком многое по умолчанию (плохая семантика). Раньше считался языком для профессионалов, теперь фактически вышел из употребления.

Пример. Программа на языке PL/1.

```
BINOM: PROC OPTIONS (MAIN);
DO N=0 TO 10;
KB, KW=1; NW=N;
DO K=0 TO N/2;
PUT SKIP DATA (K);
IF K*N¬=0 THEN
DO; KB=KB*NW/KW;
KW=KW+1; NW=NW-1; END;
PUT DATA (KB);
END BINOM;
```

Этот пример заимствован из методической разработки
[+] Руководство по основам программирования на ПЛ/1 / сост. Фазылов В.Р., Беляева Е.Е., Соловьев В.Я. – Казань: Казан. гос. ун-т, 1984.

Хочется обратить внимание на то, что достаточно сложно понять без комментариев, что здесь записано. Назначение этой программы – вычисление биномиальных коэффициентов, то есть числа сочетаний из n по k. Но не по стандартной формуле

$$C_n^k = \frac{n!}{k!(n-k)!}$$

а с некоторыми хитростями, предпринятыми для экономии ресурсов и для надежности. Проще для программиста написать бы функцию для вычисления факториала и вызывать ее три раза :-)

Более полное руководство по языку PL/1
[+] Аугустон М.И., Балодис Р.П., Бардзинь Я.М. и др. Программирование на ПЛ/1 ОС ЕС. – М.: Статистика, 1979.

В 70-е годы прошлого века на мини-ЭВМ «Наири» использовался язык **АП** (*автоматическое программирование*), использующий русские слова в названиях команд.

Пример. Программа на языке АП.

```
1  допустим  a=1  b=-3  c=2
2  вычислим  d=√(b^2-4ac)
3  вычислим  x=(-b-d)/(2a)
4  вычислим  y=(-b+d)/(2a)
5  печатаем с 3 знаками  x  y
6  кончаем
исполним 1
```

Пример. Решение системы линейных алгебраических уравнений.

```
i=3  j=4  a
i=2  x
1  допустим  i=0
2  допустим  j=0
3  введем  a_ij
4  вставим  j=j+1
5  если  j-3≤0  идти к  3
```

6 вставим i=i+1
7 если i-2≤0 идти к 2
8 программа су (а 3 х)
9 печатаем с 5 знаками x_0 x_1 x_2
10 кончаем

Этот пример взят из книги
[+] Светозарова Г.И., Сигитов Е.В., Козловский А.В. Практикум по программированию на алгоритмических языках. – М.: Наука, 1980.

BASIC (Beginners All-purpose Symbolic Instruction Code = универсальный символический код для начинающих)
Разработали в 1964 г. Томас Курц и Джон Кемени, Дартмутский колледж. Язык был предназначен для обучения начинающих программистов, но неожиданно получил широкое распространение. У первых персональных компьютеров фирмы IBM был встроенный Бейсик-интерпретатор.

Историческая справка. Служебные и пользовательские программы у первых персональных компьютеров записывались на дискеты объемом 360 K (или даже меньше). Минимальный объем ОЗУ (RAM) 64 K, можно было наращивать эту память до 640 K. В старших адресах единого адресного пространства 1 Мб (доступного операционной системе **MS DOS**) в зоне ROM после BIOS размещался интерпретатор языка Бейсик.
На факультете ВМК КГУ две первых персоналки появились в 1986 г., их предоставило для обучения студентов ПО «Камаз».

Существует множество версий языка Бейсик.
Visual Basic (Microsoft) – современный объектно-ориентированный язык.

Pascal (Блез Паскаль – французский математик и философ)
Никлаус Вирт (Высшая техническая школа, Цюрих, Швейцария) предложил этот язык для обучения основам программирования. При его разработке было задано три условия: минимум базовых понятий, простой синтаксис и компактный компилятор.
Получился простой и удобный язык строгой дисциплины программирования!
Имеется много современных модификаций, с существенными расширениями и дополнениями.

Turbo Pascal – система программирования фирмы Borland International, первая версия появилась в 1983 г. Вариант языка Паскаль получил такое же название. Первый компилятор был объемом порядка 30 K, что привлекло внимание многих программистов. Ежегодно выходили новые версии системы, последняя версия 7.0 выпущена в 1992 г. В 1989 г. в версии 5.5 появились средства для объектного программирования.

При программировании на Турбо Паскале можно обходиться минимальными средствами. Достаточно иметь строчный компилятор *tpc.exe* и библиотеку стандартных модулей *turbo.tpl* (в сумме объем примерно 120 K). Скромные сегодня графические возможности поддерживают драйвер *egavga.bgi* и библиотека *graph.tpu*. Все происходит в режиме эмуляции MS DOS.
Программист на Турбо Паскале имеет доступ ко всем ресурсам машины.

Пример. Прямой доступ к видеопамяти.

```
uses DOS,CRT;
var  r: Registers;
BEGIN
Randomize;
r.ax:=$13;
Intr($10,r);
while not KeyPressed do
  begin
  Delay(1);
  mem[$A000:320*Random(199)+Random(319)]:=Random(255)
  end;
TextMode(CO80)
END.
```

Программист в системе Turbo Pascal мог писать фрагменты исходного кода непосредственно в машинных командах или на языке встроенного Ассемблера. Пример такой программы мы уже рассматривали.

В системе разработки приложений **Borland Delphi,** первая версия появилась в 1995 г., стало возможным программировать под **Windows**. Язык программирования Turbo Pascal был расширен до **Object Pascal**, сейчас этот язык стали называть **Delphi**.

Пример. Программа на Delphi: консольное приложение.

```
program Consol;
{$apptype console}
begin
Writeln('FFF');
Readln;
end.
```

Пример. Программа на Delphi: пустое окно.

```
program Window;
uses Forms;
var F: TForm;
begin
Application.Initialize;
Application.CreateForm(TForm,F);
   F.Caption:='PNB';
Application.Run;
end.
```

Последние версии среды разработки приложений Borland Delphi разрешают одновременно пользоваться языками: Delphi для Win32, Delphi для .NET, C++ и C# !

ADA (в честь Августы Ады Лавлейс, сотрудницы Чарльза Бэббиджа)
В 1975 г. Министерство обороны США организовало специальный комитет для разработки универсального языка программирования. В 1979 г. победителем конкурса был признан коллектив сотрудников фирмы Ханивелл Бюлль и Исследовательского

Центра Ханивелл (Жан Ичбиа и другие). Язык Ада создавался с целью обеспечить разработку программ с высокой степенью надежности и с многолетним сроком службы. В нем предусмотрена *обработка исключительных ситуаций*, а также возможно *параллельное программирование* для многопроцессорных компьютеров.

C

Язык Си появился в начале 70-х, автор – Деннис Ричи, Bell Laboratories. Это язык системного программирования, он разрабатывался для реализации ОС Unix (альтернатива языку ассемблера), но быстро превратился в язык общего пользования. Компактные программы, эффективный объектный код, но сложный синтаксис. Программирование на С требует хорошей подготовки («много темных закоулков»), довольно трудно читать чужие тексты программ.

C++

В начале 80-х Бьярн Страуструп дополнил язык С объектно-ориентированными средствами по образцу языка Simula 67. При этом была существенно улучшена семантика. Получилось современно и модно!

[+] Страуструп Б. Язык программирования С++. – Киев: «ДиаСофт», 1993.
[+] Шилд Г. Самоучитель С++. – СПб.: БХВ-СанктПетербург, 1998.

Пример. Программа на С++.

```
#include <iostream.h>
class comp
  {
  float re, im;
 public:
  void assign(float re_, float im_);
  void show();
  comp operator+(comp c);
  };
void comp::assign(float re_, float im_)
  {
  re = re_; im = im_;
  }
void comp::show()
  {
  cout << re << " i " << im << "\n";
  }
comp comp::operator+(comp c)
  {
  comp res;
  res.re = re+c.re; res.im = im+c.im;
  return res;
  }

int main()
  {
  comp a, b, c;
  a.assign(2,3);
  a.show();
  b.assign(4,8);
```

```
 c=a+b;
 c.show();
 return (0);
 }
```

C# (читается: «Си шарп»)
Разработчики – Андерс Хейлсберг, Скотт Уилтамут, Питер Гоулд (Microsoft). Язык для разработки платформы .NET Framework и для программирования под эту платформу.

Платформа .NET – среда управляемого выполнения программ фирмы Microsoft. Основные функции: обеспечение безопасности кода и совместимость различных языков программирования. Главная идея: используется *промежуточный машинно-независимый язык* CIL – Common Intermediate Language (IL).
Компилятор с языка программирования высокого уровня, адаптированного под .NET, переводит исходный код не в команды процессора, а на язык CIL. Среда выполнения приложений .NET компилирует CIL-код в машинные команды в режиме "по мере надобности" JIT – Just-In-Time.

Пример. Программа на C#.

```
using System;
public struct Comp
{
 public double Re;
 public double Im;
 public Comp(double re, double im)
  {
  Re = re;
  Im = im;
  }
 public static Comp operator +(Comp c1, Comp c2)
  {
  Comp Res = new Comp();
  Res.Re = c1.Re + c2.Re;
  Res.Im = c1.Im + c2.Im;
  return Res;
  }
}

class Class
{
 static void Main()
  {
  Comp X = new Comp(1,-2);
  Comp Y = new Comp(2,5);
  Comp Z = X + Y;
  Console.WriteLine("{0} i {1}",Z.Re, Z.Im);
  Console.ReadLine();
  }
}
```

Классификация языков программирования

Идеальной системы классификации нет (и, вероятно, быть не может). Вопросы классификации объектов в программировании обсуждаются, например, в книге
[+] Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. – СПб.: Издательство Бином, 2000.

Классификация языков программирования проводится или на основе реализованных в них принципов программирования (или *парадигм*) или в соответствии с классом задач, для решения которых используется язык.

Императивные ЯП : программа состоит из последовательности инструкций (команд, операторов). В соответствии с архитектурой фон Неймана выполнение каждой инструкции изменяет содержимое памяти компьютера.
Типичные представители: Fortran, Algol, Pascal, Basic, C, C++, C#.

Функциональные (или аппликативные) ЯП : программа представляет собой последовательность функций и выражения, которые нужно вычислить.
Структура программы на функциональном языке может быть такой:

$$function_n (... function_2 (function_1 (data)) ...)$$

Считается, что реализация функциональных языков на компьютерах с архитектурой фон Неймана не эффективна.
Типичные представители: Lisp и Scheme, ML (MetaLanguage), Miranda, Haskel.
(Язык Lisp предложил Джон Мак-Карти в конце 1950-х годов, он ориентирован на обработку списков. Язык ML иногда преподают как первый язык программирования.)

Процедурные ЯП : программа представляет алгоритм решения задачи.
Все примеры, приведенные выше, относились именно к процедурным языкам.

Декларативные (логические) ЯП : программа состоит из определений отношений между объектами и поставленной цели. Алгоритм не записывается.
Типичный представитель: Prolog (Programming in Logic, 1973, Алан Кольмероэ).

Объектно-ориентированные (объектные) ЯП : данные представляют собой объекты стандартных или пользовательских классов; алгоритмы реализованы как последовательности вызовов методов этих объектов, включая конструкторы и деструкторы.
Типичные представители: Smalltalk, C++, Delphi, C#, Ada, Java.

Кроме того, обычно выделяют в особые группы

Языки программирования баз данных
Языки программирования компьютерных сетей
Языки моделирования (CASE-системы)

и так далее ...

Критерии оценки ЯП

Программы пишут, отлаживают и сопровождают! Каждый язык имеет и преимущества, и недостатки. Критерии оценки могут быть как объективные, так и субъективные. Иногда имеются внешние причины, влияющие на оценку. Например, если особые требования

предъявляет заказчик программного продукта. Иногда программистская фирма или иная организация, преследуя корыстные цели, искусственно поддерживает тот или иной язык программирования.

Вот несколько критериев, конфликтующих друг с другом [1] :

- *Простота* – как в использовании, так и в изучении.
- *Гибкость* – возможность реализации алгоритмов простыми средствами.
- *Надежность* – строгая типизация; обнаружения ошибок во время трансляции.
- *Естественность* – близость к предметной области.
- *Мобильность* – свободный перенос программ с машины на машину.
- *Стоимость…*

На выбор языка программирования часто влияют субъективные причины (личные вкусы программиста или, что еще хуже, заказчика).

4. Данные и операции

Данные и *операции* – фундаментальные понятия в программировании.
Как правило, любую конструкцию программы можно отнести к одной из двух этих категорий. Но в некоторых случаях один и тот же двоичный код может рассматриваться и как операции, и как данные (например, если программа модифицирует текст другой программы или даже свой собственный фрагмент).
При программировании на низком уровне нужно знать, как данные размещаются в памяти и какие команды можно использовать при выполнении операций. Языки программирования высокого уровня используют *абстрактные модели* данных и операций. Не все данные и операции нужно описывать …

Данные имеют *атрибуты*:
имя; адрес; значение; тип; область видимости и время жизни.

Имя (или *идентификатор*) – конечная последовательность символов.
В большинстве языков при записи имен используются буквы и цифры, иногда некоторые другие символы – например, знак подчеркивания. В составных именах чаще всего в качестве разделителя используется точка. В одних языках различаются строчные и прописные буквы, а в других – нет.
Ключевые слова – имена, имеющие особый смысл в языке программирования.
Например, begin, end, if. Ключевые слова обычно *зарезервированы*, но иногда они могут *переопределяться*. Обычно имена встроенных типов и функций являются *предопределенными*, при этом их можно переопределять. Это может привести к серьезным ошибкам.

Пример.

```
const
        pi = 1.1111111;
begin
Writeln(pi);
end.
```

Вопрос: что изменится, если из этой программы на Паскале убрать две первые строки?

Адрес – номер ячейки памяти компьютера, начиная с которой размещается значение. Адрес, как правило, назначается служебными программами (менеджером памяти) автоматически. Если имени ставится в соответствие адрес, то происходит *связывание*. Различают *раннее* связывание (во время компиляции) и *позднее* связывание (во время выполнения). Связывание может быть разрушено. Две операции используются при связывании и при его отмене: выделение памяти и освобождение памяти.
Иногда программист имеет возможность указать, в каких ячейках памяти разместить данные. Можно указать адрес явно или неявно.

Пример.

```
var
    r:  real;
    s:  string[6] absolute r;
    t:  string[4000] absolute $B800:$0000;
```

В примере на языке Турбо Паскаль явный адрес указан в виде пары *сегмент : смещение*. (со знака $ начинаются шестнадцатеричные числа).
Так было принято в операционной системе MS DOS (это поддерживается в Windows и сейчас в режиме эмуляции MS DOS). Причина проста: для адресации ячеек памяти в 1 Мб используются пятизначные шестнадцатеричные числа, а шина данных у первых персоналок была 16-битная – на нее можно было выставить только четыре цифры. Поэтому адрес передавался по частям: например, $1A20F = $1A20 : $000F или, по-другому, $1A00 : $020F.
Так что в MS DOS используется *сегментная модель памяти*.

Карта распределения памяти (MS DOS)

00000 – 9FFFF (минимум 64 К)	Служебная зона
	Память для прикладных программ
	Стек для прикладных программ
	Служебная зона (стирается и восстанавливается)
A0000 – BFFFF	Видеопамять 128 К (у первых машин было меньше)
F0000 – FFFFF	BIOS (здесь же когда-то был встроенный Бейсик)

Первые 640 К + 128 К – память RAM, последние 256 К – память ROM.

Таким образом, в последнем примере переменная t размещена в видеопамяти – скорее всего там, где находится первая страница текста (если видеокарта работает в стандартном текстовом режиме). При попытке присвоить этой переменной какое-либо значение должен меняться видимый на экране текст.

Уточним, где и как размещаются в памяти данные программы.
В системе программирования Turbo Pascal распределением памяти занимаются служебные программы библиотечного модуля System (он автоматически подключается ко всем пользовательским программам, программист об этом не должен беспокоиться).
Та часть памяти RAM, которая отводится прикладным программам, распределяется следующим образом (после загрузки исполняемого файла):

Префикс программного сегмента
Сегмент кода основной программы
Сегмент кода последнего модуля
.....
Сегмент кода первого модуля
Сегмент кода модуля System
Сегмент данных (глобальные переменные)
Сегмент стека
Оверлеи
Динамически распределяемая память (куча) : занятая часть свободная часть список свободных блоков

Префикс программного сегмента – небольшая (всего 256 байтов) служебная запись. *Сегмент кода* – машинные команды, которые сгенерировал транслятор, обрабатывая исходный текст на алгоритмическом языке.

Библиотечные модули, если они используются, размещаются в памяти следом за машинным кодом, причем последний в списке uses будет загружен первым (на младшие адреса). Если в разных модулях имеются описания тех или иных конструкций (типов данных, констант или переменных) с одинаковыми именами, то доступными будут ближайшие к коду – то есть из модуля, последнего в списке. Этот порядок иногда приводит программиста в недоумение, когда после добавления к программе нового библиотечного модуля вдруг перестают работать как надо фрагменты старых модулей, которые были давно отлажены.

Как уже было сказано, модуль System всегда в памяти.

Сегмент данных – именно та область памяти, где хранятся значения глобальных переменных (и констант). Область эта не слишком велика, все сегменты не могут быть более чем 64 К. У такой модели памяти один плюс: для нумерации ячеек памяти в сегменте достаточно использовать только смещение (короткий адрес).

Стек в системе Turbo Pascal по умолчанию всего 16К. Это значение можно увеличить – или директивами компилятора в тексте, или изменением параметров интегрированной среды разработки, – но не более чем до 64 К.

Оверлеи сегодня используются крайне редко. Они были придуманы на тот случай, когда машинный код программы не помещался в один сегмент. Особыми средствами можно было разбить программу на фрагменты (оверлеи), которые размещались по одному в специальную область сегмента кода и заменялись при необходимости на другие во время выполнения программы.

Самая интересная часть памяти на последней карте – *куча* (Heap, или *динамически распределяемая память*). Заметим, что ее объем (по сравнению с сегментами) достаточно велик: из 640 К памяти RAM под Heap может быть отведено более 400 К (есть простые средства для определения размера кучи).

Итак, как же пользоваться динамически распределяемой памятью?

Для этого в большинстве языков программирования предусмотрены специальные типы данных – *указатели, стандартные* (без типа) и *типизированные*. В языке Паскаль стандартный указатель – тип pointer, данные такого типа занимают в памяти 4 байта и их значения интерпретируются как адреса (сегмент и смещение).

Типизированные указатели обычно определяет программист.

Пример.

```
var
   x: real; yP: ^real;
begin
x:=1.23;
New(yP);
yP^:=x;
Writeln(x,’  ‘,yP^);
Dispose(yP);
end.
```

Здесь переменная x размещается в сегменте данных, 6 байтов отводится под тип real. Переменная yP (суффикс P поставлен для напоминания, что имеем дело с указателем) также размещается в сегменте данных, ей отводится 4 байта. Процедура New выделяет в куче необходимый кусок (в примере 6 байтов) и сохраняет в переменной-указателе адрес первого из них. Таким образом, yP указывает на место в куче, где хранится значение типа real, а доступ к нему осуществляется по имени yP^. Место в куче можно освободить с помощью процедуры Dispose.
Есть также процедуры GetMem и FreeMem …

Интересный вопрос: какое значение по умолчанию получает переменная сразу после того, как она объявлена (что хранится в отведенных для нее байтах памяти, пока туда ничего не было специально записано)?

Разумеется, в куче выгоднее размещать данные большой длины. Следующий пример показывает, как разместить там массив.

Пример.

```
type
   Mas = array [1..1000] of real;
   MasP = ^Mas;
var
   a: Mas;  bP: MasP;
begin
New(bP);
……..
Dispose(bP);
end.
```

«По техническим причинам» место в куче выделяется не побайтно, а блоками по 16 байтов (это называется *нормализация*). При вызове процедуры New номера освободившихся блоков вносятся в *список свободных блоков*, хранящийся в старших адресах кучи. Может оказаться так, что куча пустая, а разместить в ней ничего даже среднего размера нельзя, поскольку освобождалась она раньше маленькими кусками.
Есть простое правило: из кучи извлекают первым то, что положили туда последним.

Специальные переменные модуля System хранят значения параметров кучи:
   HeapOrg – начало кучи
   HeapPtr – начало свободной зоны
   FreeList – начало списка свободных блоков
   HeapEnd – конец кучи
Для проверки, сколько имеется свободного места, используют функции:

MemAvail – возвращает объем свободной зоны
MaxAvail – максимально возможный объем для размещения данных

Как вычислить адрес переменной или какого-нибудь иного объекта?
Пример.

p:=@x; p:=Addr(x);

Функции Ofs и Seg определяют сегмент и смещение по указателю, а функция Ptr изготавливает указатель из сегмента и смещения.

В Windows 3.1 также использовалась сегментная модель памяти, а в Windows 95 модель памяти *линейная* (плоская).

Константы размещаются в памяти точно так же, как и переменные, но их значения запрещено изменять. В Турбо Паскале поддерживаются константы различных конструкций. Типизированные константы – переменные с начальным значением.

Значение – это содержимое ячеек памяти, выделенных в процессе связывания. Записанный в память двоичный код интерпретируется в соответствии с назначенным типом.

Тип данных – система соглашений о том, как интерпретировать значения (как они размещаются в памяти, какие допустимые диапазоны и т. д.).

Например, в языке Паскаль следующие типы данных имеют такие диапазоны значений:

byte	1 байт	0 .. 255
integer	2 байта	-32768 .. 32767 (2^{15}=32768)
real	6 байтов	2.9×10^{-39} .. 1.7×10^{38}
pointer	4 байта	0000:0000 .. FFFF:FFFF

Как интерпретировать записанные в отведенный кусок памяти биты – дело не всегда тривиальное.
Для типа integer система такая: один бит отводится на знак, но отрицательные значения записываются в дополнительной арифметике – код инвертируется и к результату добавляется единица.

Что получится, если запустить программку

```
var k: integer;
begin
k:=32767;
Writeln(k,'  ',k+1);
Readln;
end.
```

Значения данных типа real хранятся так: один бит под знак (S), 39 битов под мантиссу (F) и оставшиеся 8 битов – под порядок (E). Используется формула: $(-1)^S \times (1.F) \times 2^{E-129}$
Поэтому минимальное и максимальное значения для этого типа
$1 \times 2^{-128} \approx 2.939 \times 10^{-39}$ и $2 \times 2^{126} \approx 1.701 \times 10^{38}$

Тип char требует 1 байт памяти, а значения типа string занимают 256 байтов (в нулевом байте хранится количество символов).

Функция SizeOf имеется во многих языках программирования, она возвращает длину в байтах для указанного типа или переменной.

Уже было сказано, как разместить данные в памяти по конкретному адресу (там, где уже что-либо размещено, или непосредственно начиная с указанной ячейки памяти). Еще один

Пример.

```
var
   r: real;
   s: string[6] absolute r;
begin
Writeln(r,'   ',s);
s:='abcpnb';
Writeln(r,'   ',s);
 r:=3.1415926536;
Writeln(r,'   ',s);
end.
```

Различают языки программирования со *слабой типизацией* и с *сильной типизацией*. Это зависит от того, разрешено или не разрешено автоматически преобразовывать значения одного типа в значения другого типа при вычислениях (эта операция называется *приведение*). Вот пример для экспериментов:

```
var  k, j: integer;  r: real;    .....   r:=k-j;  k:=r-j;  k:=j-r;
```

Еще пример:

```
var  a: char;   .....   a:=2;   Writeln(a);
```

Область видимости данных определяет структура программы.

Паскаль

```
program P;
var x,y,z: word;
procedure P1(a: word);
   var z,u,v: word;
   procedure P2(a: word);
      var v,w: word;
      begin
      .....
      end;
   begin
   .....
   end;
procedure P3(a: word);
   var z,u,v: word;
   begin
   .....
   end;
```

```
begin
.....
end.
```

C

```
int x,y,z;
void main(void);
{
int x;
.....
}
void f1(int a);
{
int y;
.....
}
void f2(int a);
{
int z;
.....
}
```

Время жизни данных зависит от места и способа их размещения в памяти: глобальные данные живут столько же, сколько сама программа. Локальные данные и параметры подпрограмм размещаются в стеке и живут, разумеется, не слишком долго. Время жизни данных в куче может устанавливать программист.

В соответствии с общей концепцией архитектуры фон Неймана *операции* изменяют данные, в процессе этого *исходные данные* преобразуются в *результат*.

Операции выполняются на *низком* и на *высоком* уровне. Это, разумеется, условность: выполняются они всегда на самом низком уровне, а вот планировать их можно на уровне армий и фронтов :-) .
Процедуры и *функции* (подпрограммы) – операции самого высокого уровня. Вообще говоря, можно весь алгоритм вычислений свести к одному шагу (к вызову одной единственной процедуры).

При использовании подпрограмм обмен данными проводится с помощью *формальных и фактических параметров*. Грамотный программист знает, что данные могут передаваться в процедуры и функции или по значению или по ссылке.

В языке Turbo Pascal это делается так. Пусть, например, заголовок функции

```
function A(i: integer; var w: word; const r: real): boolean;
```

Здесь имеем три формальных параметра, относящихся к различным категориям.

Параметры-значения : копии их значений передаются в подпрограмму через стек. При этом фактическим параметром может быть любое выражение, совместимое по типу.

Параметры-переменные : только их адреса передаются через стек. Фактическим параметром должна быть переменная того же типа.

Параметры-константы : через стек передается адрес, но при этом запрещено изменять исходное значение. Фактическим же параметром может быть выражение, совместимое по типу (*версия TP 7.0*).

Можно использовать параметры-переменные и параметры-константы без типа (ответственность возлагается на программиста). В таком случае в подпрограмму передается адрес (значение указателя без типа), он может быть интерпретирован в дальнейшем как угодно – на усмотрение автора программы.

Обратим особое внимание на то, что в большинстве систем программирования имеется много полезных процедур и функций в стандартных библиотеках! Рекомендуется время от времени читать документацию.

Иногда можно вызывать функции как процедуры. При этом возвращаемое значение просто теряется (изымается из стека, но нигде не сохраняется).
Имеет смысл, например, оформлять в виде функций с возвращаемым булевским значением алгоритмы решения задач, когда нет гарантии, что решение всегда может быть получено. Типичный пример – решение системы линейных алгебраических уравнений методом Гаусса. Во время обратного хода возможно переполнение (или вообще деление на нуль). В сомнительных случаях пишем так:

```
if  Gauss(.....)
then  .....
else  ..... ;
```

а если уверены, что операция корректна, то просто

```
Gauss(.....);
```

Можно передавать в качестве параметров имена процедур и функций. Для этого в языке Turbo Pascal предусмотрены *процедурные типы* данных.

Пример.

```
type
   Func = function(x: real): real;

function Integral(F: Func; a,b: real): real;
   begin
   F:=.....;
   end;

{$F+}
function Dem(t: real): real;
  begin
  Dem:=sqrt((t+1)/(t-1));
  end;
{$F-}

begin
Writeln(Integral(F,0,1));
end.
```

Здесь используются директивы компилятора, заставляющие его создавать для функции Dem длинный адрес – сегмент и смещение. Помним: данные хранятся в сегменте данных, а машинные коды – в сегменте кода. Если указать для точки входа в функцию только смещение (короткий адрес), то оно отсчитывается от начала сегмента кода, а смещения адресов параметров берутся от начала сегмента данных.

Операции условно «среднего» уровня предусмотрены в синтаксисе языка программирования. Так можно называть операции (операторы), которые используются в выражениях. Они близки к машинным командам, но иногда бывают более сложными.

Turbo Pascal поддерживает:

арифметические операции
логические операции
операции с битами
операции со строками
операции отношения
адресная операция @

Арифметические операции (унарные и бинарные) предусмотрены для данных целых и вещественных типов: + - * / div mod
Арифметические выражения составляются из *операндов,* арифметических операций, круглых скобок и вызовов функций.
Важно помнить: операции имеют *приоритет*. Порядок вычисления выражений, хотя и оговаривается в стандарте языка, может зависеть от используемого транслятора.

Практически во всех языках программирования используется

оператор присваивания <имя переменной> := <выражение>;

Как уже было сказано раньше, различают языки программирования с сильной типизацией и со слабой типизацией. При слабой типизации (а также при неосторожных действиях программиста) возможны *побочные эффекты*!

Пример.

```
function f(var x: real): real;
begin
x:=x*x;
f:=x;
end;
var x: real;
begin
x:=1.2;
Writeln(x+f(x),'   ',f(x)+x);
end.
```

Здесь, конечно, все просто. Но если функция f () была описана в библиотечном модуле, исходник которого мы поленились посмотреть ...

Логических операций четыре: not and or xor

В TP реализовано полное и укороченное вычисление логических выражений. Рекомендуется изучить инструкцию, поскольку по умолчанию вычисление укороченное, а это может привести к побочным эффектам. Примеры – в книгах.

Предусмотрены *операции с битами* : not and or xor shl shr
Хотя процессор работает с кодами минимальной длины 8 битов, можно кое-что выполнить и на битовом уровне.

Пример.

```
const
   Mask=$FF;
var
   W: word;
begin
Readln(W);
Writeln('Младший байт',' ',(W and Mask));
Writeln('Старший байт',' ',((W shr 8) and Mask));
Readln;
end.
```

В этом примере заведена *маска*, с помощью которой из слова выделяются отдельные байты.
Вопрос: какая нужна маска, чтобы увидеть отдельные биты в байте?

Операции отношения: = <> < <= >= > используются в логических выражениях. Особая операция in используется при работе с множествами.

Операции самого низкого уровня представляют собой машинные команды или команды языка Ассемблера.

5. Типы данных

В соответствии с тем, что было сказано раньше, *тип данных* – система соглашений о том, как интерпретировать значения (как они размещаются в памяти, какие допустимые диапазоны и т. д.).

В программах используются как *стандартные* типы данных, так и типы, *определяемые программистом.*
Приведем краткий обзор стандартных типов данных в разных языках и покажем, как определяются пользовательские типы данных.

Turbo Pascal

Типы данных подразделяются на

простые
- целый Shortint, Integer, Longint, Byte, Word (знаковые и беззнаковые)
- логический Boolean, ByteBool, WordBool, LongBool *(версия 7.0)*
- символьный Char (полный набор ASCII-символов)
- перечисляемый
- тип-диапазон
- вещественный Single, Real, Double, Extended, Comp

структурированные
указатели
процедурные
объектные

Простые типы и использовать просто. Нужно только следить за границами изменения значений.

Простые перечисляемые типы приближают язык программирования к языку предметной области. Например,

```
type
  Color = (white,  red,   green,  blue,  black);
  Operation = (Add,  Sub,  Mul,  Dvd);
```

Каждое значение из списка отождествляется с целым числом: 0, 1, и так далее.

Типы-диапазоны (ограниченные типы) повышают надежность программ. Например, если значения переменной определяют позицию символа в строке экрана, то имеет смысл определить

```
type
   Position = 0..79;
```

Для всех порядковых типов определены *операции сравнения* и операции
Dec(), Inc(), Ord(), Pred(), Succ(), Odd()
Имеются функции преобразования типа:
Chr(), Low(), High(), Ord(), Round(), Trunc()

Пример.

```
type
  Color = (red, green, blue);
var
  x, y, z: Color;
begin
x:=red;
y:=Color(1);
z:=Succ(y);
Writeln(Ord(x),'  ',Ord(y),'  ',Ord(z));
if z > y then Writeln('z > y') else Writeln('z <= y');
Readln;
Dec(x);
Writeln(Ord(x));
Readln;
end.
```

Наблюдается ли побочный эффект?

Еще один эксперимент: поместим в хвост программы строки

```
Writeln(integer(z));
Readln;
```

Borland Delphi

Типы данных подразделяются на *фундаментальные* и *генерируемые* !
Для фундаментальных типов места в памяти отводится ровно столько, сколько предусмотрено в описании языка. Для генерируемых типов может быть и меньше, это зависит от имеющихся в наличии ресурсов памяти (если места мало, то длина может быть уменьшена).

Перечислим некоторые из простых типов языка Delphi (раньше его называли Object Pascal). По группам, в первой строке – фундаментальные типы, во второй – генерируемые.

Целые типы:
Shortint, Smallint, Longint, Int64; Byte, Word, Longword
Integer, Cardinal (32 бита или меньше)

Символьные типы:
Char, AnsiChar, WideChar
(Язык Delphi поддерживает две кодировки: ASCII и ANSII. Тип WideChar – для символов в кодировке UNICODE, размер памяти 2 байта.)

Вещественные типы:
Single, Real48, Double, Extended
Real

Таким образом, привычные для программиста на Паскале типы Integer и Real стали генерируемыми.

C++

В языках C и C++ к простым типам относятся: char, int, float, double .
(поддерживается разная длина в 16- и 32-битных моделях данных).
Широко используются *модификаторы длины* short, long, signed, unsigned .
Например,
unsigned long int 32 бита 0 . . 4294967295

Предусмотрены модификаторы доступа: const и volatile . Первый запрещает изменять значения переменных (эти значения назначаются при инициализации), а второй предупреждает транслятор, что значения данных могут изменяться, но не самой программой, а каким-либо другим образом.
Например,
```
const volatile unsigned char *port = 0x30;
```

У перечисляемых типов, например
```
enum suit (clubs, diamonds, hearts, spades);
enum answer (yes, no, maybe = -1);
```
значениям могут соответствовать любые целые константы, в том числе и отрицательные.

Разрешается автоматическое преобразование типов в смешанных выражениях. Есть два основных правила:

char, short, enum > int, **остальное** > unsigned
long double > double > float > unsigned long > long > unsigned > int
Приведение типа может быть назначено явно, например так:

```
x = (float)  ((int) y + 1;
```

Turbo Pascal

Структурированные типы подразделяются на:
- тип-массив
- тип-запись
- тип-множество
- тип-файл

Типы-массивы используются в программах очень часто. Например,

```
type
   Mas = array [1..1000]  of  Real;
   Tab = array [1..10, 1..15]  of  Integer;
```

Формальными параметрами подпрограмм могут быть *открытые массивы*.

```
procedure  Zero (var  x:  array  of  Real);
   var  j :  Integer;
   begin
   for  j := 0  to  High(x)  do  x[j] := 0;
   end;
```

По умолчанию нижняя граница индекса – нуль, а верхняя определяется встроенной функцией High().
Символьные строки в Паскале представляют собой массивы:

```
   string [n] = array [0 . . n] of  Char;
```

Поэтому можно обращаться к отдельным символам строк. Если размер строки не указан, то память резервируется по максимуму: на 255 элементов.
Пример.

```
   var
      s: string[50];          t: string;
```

В Turbo Pascal имеются также ASCIIZ-строки – переменные типа PChar. Размер таких строк может доходить до 65535 байтов, признак конца строки – символ с ASCII-кодом 0. Если переустановить ключ компилятора $X+ (так по умолчанию) в $X-, то переменные типа PChar будут рассматриваться как указатели на символьный тип char.
Для объединения в единое целое данных различных типов предусмотрены типы-записи. Например,

```
 type
    Comp = record
       Re, Im:  Real;
       end;
    Data = record
       Year:  Integer;
       Month:  1 . . 12;
       Day:  1 . . 12;
```

```
        end;
```

При доступе к полям записей используются *составные имена*:

```
 var
    D: Data;
 begin
    D.Year := 2008;
```

Возможны различные варианты объявления записей, есть несколько способов обращения к полям. Во примеры:

```
   var
      A, B, C:  record   Re, Im  Real;  end;

  const
      OneR: Comp = (Re : 1;  Im : 0);

   with A do
      begin  Re:=1.2;  Im := -3.7;  end;
```

Вариантные записи позволяют интерпретировать одни и те же области памяти как аписи с разными полями. По аналогии с принятыми обозначениями в Delphi можно задавать размеры прямоугольника разными способами:

Пример.

```
type
  Point = record
    x, y : Integer;
    end;
  Rectangle = record
    case  Word  of
      0: (Left, Top, Right, Bottom: Integer);
      1: (TopLeft, BottomRight: Point);
      end;
var R: Rectangle;
begin
R.Left:=0; R.Top:=7; R.Right:=12; R.Bottom:=25;
Writeln(R.TopLeft.x,'  ',R.TopLeft.y);
Readln;
end.
```

C++

В языках C и C++ структурированные типы данных в целом такие же, как в Паскале, но имеются некоторые отличия.

Переопределяются типы данных с помощью конструкций вида

```
   typedef  double  real;
```

У массивов по умолчанию нижняя граница индекса равна нулю.

Пример из С :

```
#include  <stdio.h>
int  main(void)
{
   int  a[100];
   int  j;
   for (j=0;  j<100;  ++j)  a[j] = j;
   for (j=0;  j<200;  ++j)  printf("%d ", a[j]);
   return  0;
}
```

Здесь есть одно тонкое место (побочный эффект). На печать выводится символов больше, чем имеется в массиве. Язык С не отслеживает границы индексов у массивов (из экономии времени, пусть будет виноват программист), но в С++ такая проверка предусмотрена.

Имя массива рассматривается как указатель на первый элемент массива. Поэтому можно сделать так:

```
   int  a[100];
   int  *p;
   p = &a[0];
```

но можно последнюю строку написать проще:

```
   p = a;
```

Для размещения массивов в динамической памяти используют функцию malloc(), а для освобождения памяти – функцию free() .
Например, в С

```
   char *p;
   p = malloc(1000);
   p[77] = .....
   free(p);
```

Первые две строки этого кода не работают в С++ (типизированные указатели перестали быть совместимыми), поэтому надо написать

```
    p = (char *) malloc(1000);
```

Имеются также функции calloc() и realloc(), а в С++ new() и delete().

У многомерных массивов индексы стали более самостоятельными:

```
    double  matr  [10] [20] [30];
```

Объединения, как и вариантные записи Паскаля, позволяют интерпретировать одни и те же участки памяти как значения разных типов. Например,

```
union  intchar  {
   int i;
   char c;
};
```

Кстати, язык С допускает разные модели организации сегментной памяти:

- tiny – код, данные и стек в одном сегменте (максимальная скорость работы!);
- small – код 64К, данные 64 К;

- medium – код занимает несколько сегментов, данные – только один;
- compact – код 64К, а данные – несколько сегментов

Структуры в С++ используются по следующей схеме:

```
struct  addr  {                          объявление структуры,
   char  name[30];                       имя типа часто называют
   char  building[4];                    ярлык или шаблон,
   char  street[40];                     а поля – членами структуры
   char  city[20];
   unsigned  long  int  zip;
};

struct  addr   vin, pnb;                 объявление переменных

struct  addr   prepod[120];              массив структур

struct  addr  *p;                        указатель на структуру

p = &pnb;                                адрес помещается в указатель

pnb.name        p->city                  доступ к полям

vin = pnb;                               присваивание
```

Обратим внимание: доступ к полям структуры осуществляется или с помощью оператора «точка», или с помощью оператора «стрелка».

6. Способы описания языков программирования

Напомним несколько определений.
Язык программирования – это система обозначений и понятий для описания структур данных и алгоритмов.
Алфавит языка – набор символов, которые разрешено использовать.
Синтаксис – система правил, по которым записываются конструкции языка.
Семантика – набор правил, на основе которых следует истолковывать эти конструкции.
Имеется некоторое различие между понятиями «система правил» и «набор правил».

Идеального способа описания языка программирования пока не найдено. Одна из причин – неоднозначность конструкций естественных языков, которые используются при этом.

Кроме того, трудности описания языков программирования связаны с интересами субъектов: авторы языка стремятся изложить свои идеи строго, лаконично и красиво, разработчиков транслятора интересует совсем другое в описании языка – как быстрее и эффективнее анализировать исходный текст, а программистам хочется программировать, а не тратить время на изучение документации.

Различают неформальное и формальное описание языков программирования. В учебниках, как правило, изложение неформальное: можно писать так, а можно немного по-другому. Формальное описание языка требует больших усилий.
Вот примеры, mach mit, mach wie, mach besser ☺.

Обычно все начинается с описания синтаксиса. Для этого используются некоторый вспомогательный язык – ***метаязык***. Наиболее часто используются следующие способы описания языков программирования:

нотация Бэкуса-Наура
синтаксические диаграммы Вирта
формальные грамматики

6.1. Нотация Бэкуса-Наура

Этот способ описания языков программирования был предложен в начале 1960-х годов (Дж. Бэкус – главный разработчик компилятора для языка Фортран, П. Наур – один из авторов языка Алгол-60).

Рассмотрим фрагмент текста, написанный на исторически первом метаязыке:

<целое число> ::= <целое без знака> | + < целое без знака >
| – < целое без знака >
<целое без знака> ::= <цифра> | <целое без знака> <цифра>
<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<идентификатор> ::= <буква> | <идентификатор> <буква> | <идентификатор>
<цифра> | <идентификатор> _

Описание языка представляет собой конечную последовательность ***металингвистических форм*** (БНФ), состоящих из двух частей, разделенных ***метасимволом*** ::= (по определению).
Каждая БНФ определяет ***металингвистическую переменную*** (МЛП), имя которой записывается на метаязыке. При этом используются метасимволы < и > . Справа записываются допустимые варианты значений МЛП, которые составляются из символов алфавита языка, метасимволов и МЛП. Для краткости записи используется также металингвистическая связка | (или).
Обратим внимание: в металингвистических формах присутствует ***рекурсия*** (явная или неявная), что затрудняет чтение и понимание.

Последовательности БНФ могут быть нисходящими и восходящими. Например, сначала можно объяснить, что такое идентификатор, а потом – что такое буква. Но можно поступить и наоборот.

Предложенное несколько позже ***расширение БНФ*** (РБНФ) позволило сократить длину металингвистических форм. Добавлено две конструкции:

- необязательные элементы помещаются в квадратные скобки;
- повторяющиеся элементы (может быть, и ни разу) помещаются в фигурные скобки.

Например, можно написать так:

<целое число> ::= [+ | –] <целое без знака>
<целое без знака> ::= <цифра> { <цифра> }
<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Еще пример:

<программа на Паскале> ::= [<заголовок>] [<объявления >] **begin**
[<инструкции>] **end.**
<заголовок> ::= **program** <идентификатор> **;**

и так далее ...

Существуют ***неформализуемые правила синтаксиса***, которые не описываются БНФ (контекстные условия, правила статической семантики и т. д.)

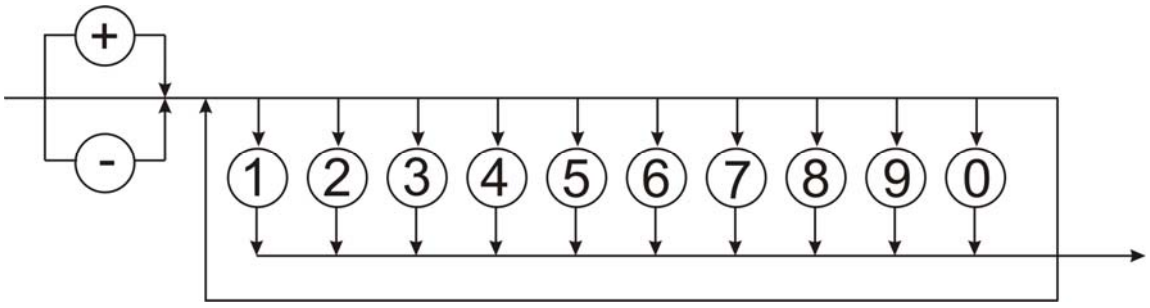
Например, в некоторых языках программирования:
– не различаются строчные и прописные буквы;
– нельзя использовать одинаковые идентификаторы для переменных;
– идентификатор должен быть описан перед использованием;
– имеются количественные ограничения на структуру МЛП;
– задаются условия совместимости операндов в выражениях;
и так далее ...

Сводка синтаксических правил языка Ада дана в приложении Е (с. 173 – 182) в книге [+] Язык программирования АДА (предварительное описание). – М.: Финансы и статистика, 1981 .

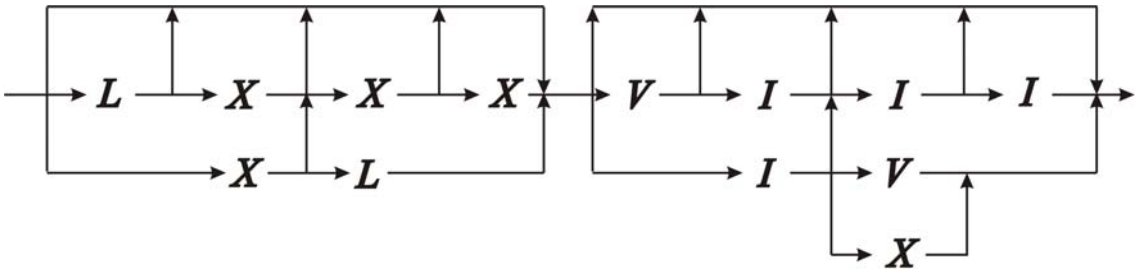
6.2. Синтаксические диаграммы Вирта

впервые использовались для описания синтаксиса языка Pascal, а позже – языков Модула-2 и Фортран-77. Они представляют собой графическую форму РБНФ, представленных в виде графов с одним входом и одним выходом.

Классический пример – определение множества целых чисел [2]:



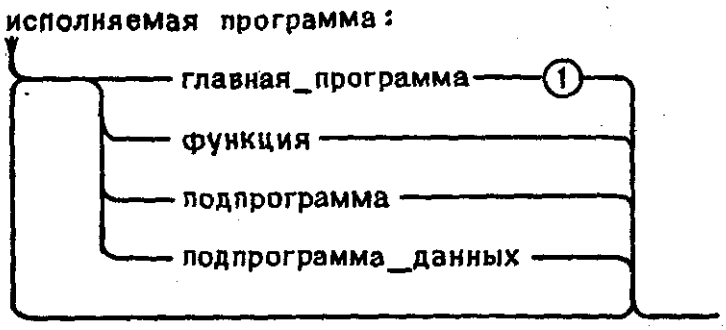
В книге [1] дана диаграмма, определяющая римские числа (от 1 до 89)



Описание синтаксиса языка Фортран 77 в форме синтаксических диаграмм дано в книге

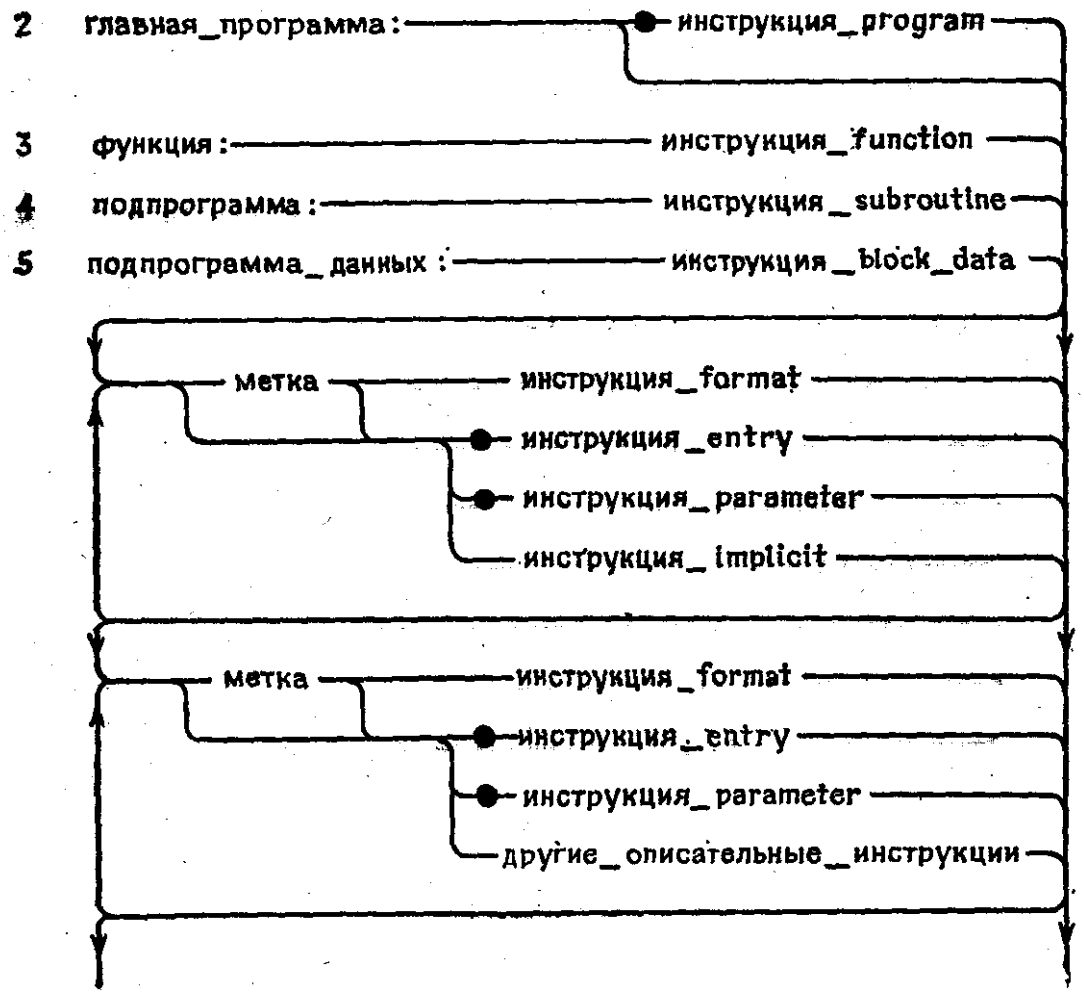
[+] Катцан Г. Язык Фортран 77. – М.: Мир, 1972
на с.181-202. Приведем в качестве примера первую и последнюю страницу этого описания, на рисунках легко найти неформализуемые правила синтаксиса.

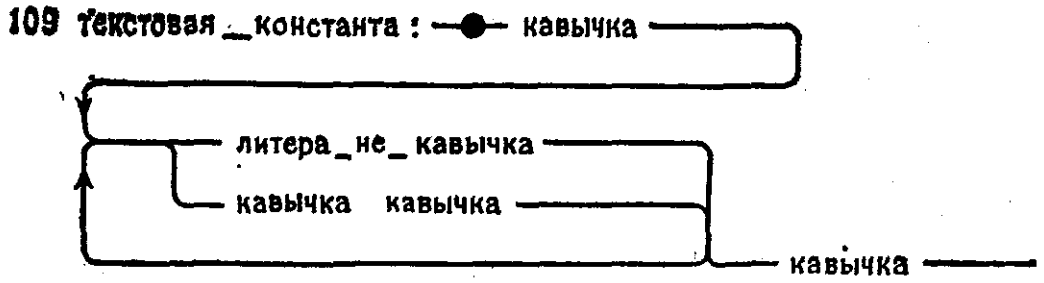
СИНТАКСИЧЕСКИЕ ДИАГРАММЫ ЯЗЫКА ФОРТРАН 77



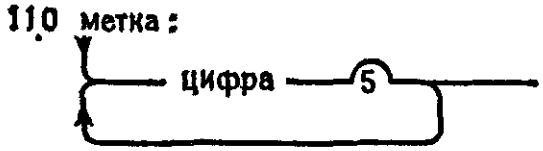
(1) *В исполняемой программе должна быть ровно одна главная программа.*

Исполняемая программа может содержать внешние процедуры, запрограммированные не на Фортране.

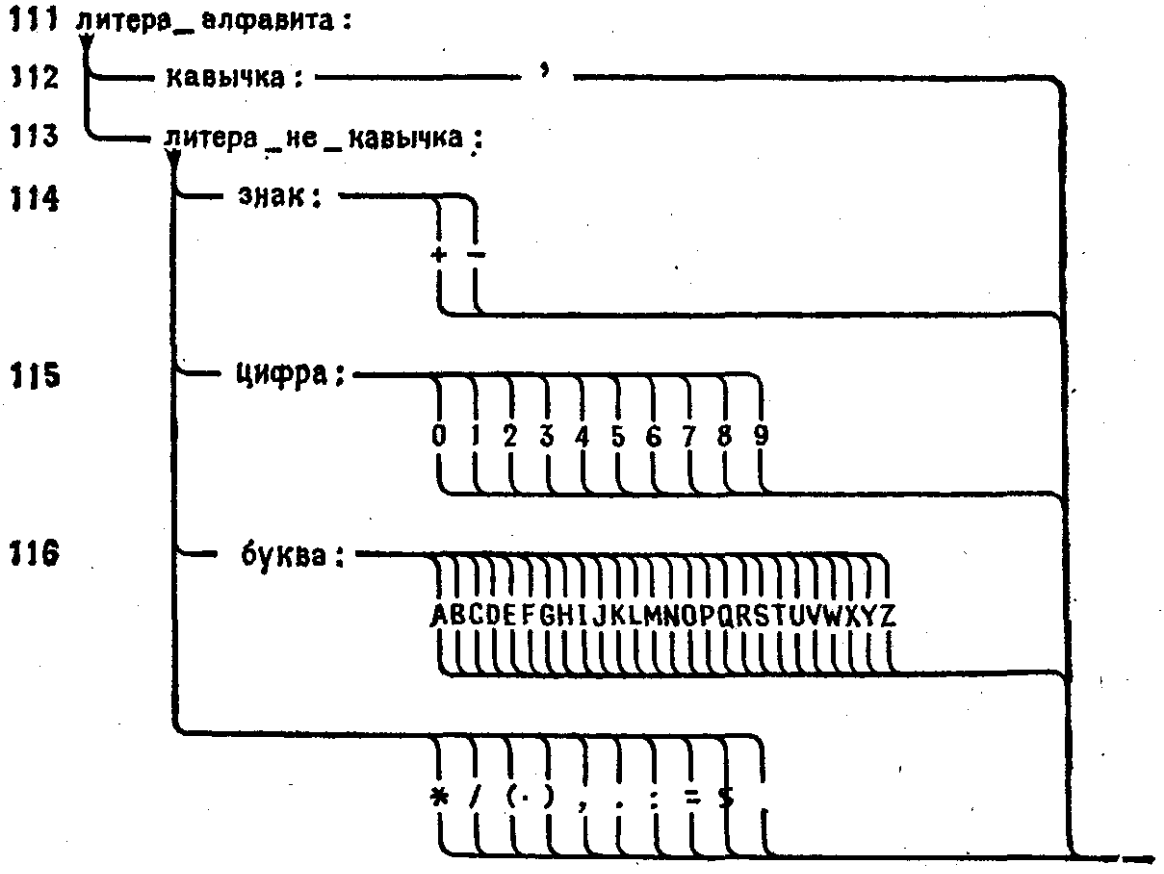




(109) *Кавычка в текстовой константе представляется в виде двух подряд стоящих кавычек, не разделенных пробелами.*



(110) *Метка должна содержать цифру, отличную от нуля.*



(111) *Пробел является литерой алфавита. В алфавит могут входить также другие литеры.*

В БНФ и синтаксических диаграммах содержатся также некоторые элементы семантики (например, названия металингвистических переменных имеют семантический смысл).

Но не всегда смысл синтаксически правильного предложения содержится в нем. Известный пример предложения русского языка (И.Щерба):

Глокая куздря штеко бодланула бокра и кудрячит бокренка.

Правила ***семантики*** обычно включаются в руководство по языку программирования в форме пояснительного текста с примерами.

Полное и строгое описание семантики – сложная проблема! Для формального описания семантики используются различные методы, некоторые из них будут рассмотрены позднее.

Операционная семантика объясняет, как программы на языке программирования выполняются на компьютере. При этом различаются высокий и низкий уровень выполнения (от принципа работы той или иной конструкции до преобразования двоичных кодов в памяти компьютера).

Пример. Оператор цикла с точки зрения синтаксиса записывается по следующему шаблону:

for <переменная> := <значение 1> to <значение 2> step <значение 3>
do <оператор> ;

Смысл этой записи состоит в том, что сначала переменная цикла получает значение 1, выполняется оператор, затем значение переменной цикла увеличивается … и так далее.