



# ЯЗЫКИ ПРОГРАММИРОВАНИЯ

*Язык формирует особый способ нашего мышления  
и предопределяет, о чем мы можем думать.*

*Б.Л. Ворф*

# Содержание

2

- Понятие языка программирования (ЯП)
- Классификация ЯП по уровню абстракции
- Элементы языка программирования высокого уровня

# Язык программирования

3

- *Язык программирования* – система обозначений для *абстрактного* описания вычислений.
- Абстрактное описание вычислений
  - не зависит от архитектуры конкретного компьютера
  - может быть переведено в детализированную форму для последующего выполнения на компьютере.
  - осуществляется специализированной программой (*ассемблер, компилятор, интерпретатор*).

# Классификация языков

4

- По отношению к архитектуре компьютера
  - ▣ Языки высокого уровня абстракции
  - ▣ Языки низкого уровня абстракции
- По парадигме (принципу постановки и решения задач)
  - ▣ Императивные языки
  - ▣ Декларативные языки
  - ▣ Параллельные языки
  - ▣ ...
- По назначению
  - ▣ Языки научных вычислений
  - ▣ Языки прикладного программирования
  - ▣ Языки баз данных
  - ▣ ...

# Языки высокого и низкого уровня

5



**Языки  
высокого уровня**  
Pascal/Delphi, Ada,  
C++, LISP, occam,  
PROLOG, Perl,  
Python, BASIC,  
FORTRAN, C, ..



**Языки  
низкого уровня**

Языки ассемблеров,  
языки машинных  
команд

- Языки высокого уровня абстракции близки к естественному языку.
- Языки низкого уровня абстракции близки к машинным командам.

# Языки машинных команд

6

- *Язык машинных команд* представляет собой набор инструкций, исполняемых процессором конкретного компьютера.
- Составные части машинной команды:
  - *код операции*, определяющий действие, которое должен выполнить процессор (сложение, сравнение значений, пересылка значения и др.);
  - *адресная часть*, содержащая адреса (номера ячеек памяти) операндов (величин, над которыми производится операция) и результата.

КОП	АдрОперанд1	АдрОперанд2	АдрРезультат
10111000	00000001	00000000	00000000

Загрузить 1  
в ячейку  
памяти с  
адресом 0

# Язык машинных команд

7

- Достоинства:
  - ▣ Программа не нуждается в трансляции.
  - ▣ Программа имеет небольшой размер и выполняется быстрее, чем программа, транслированная с другого языка.
- Недостатки:
  - ▣ Большая сложность разработки программ.
  - ▣ Узкая область применения программы (один конкретный компьютер).

## Машинный код программы вычисления факториала

```
10111010 00001100
00000001 00101110
10001001 00010110
10010001 00000010
10001011 00011110
00101100 00000000
10001110 11011010
10100011 ...
```

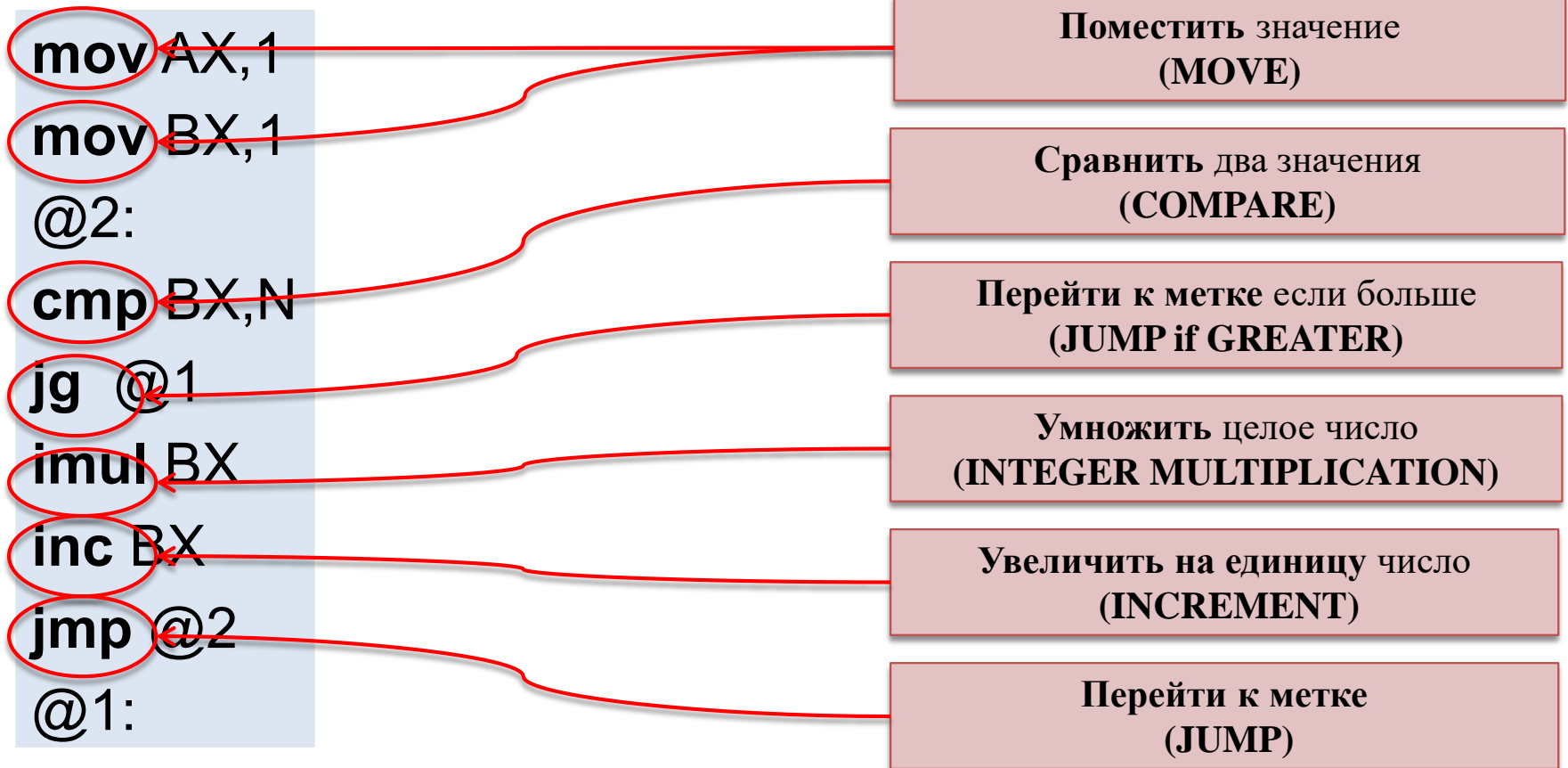
# Языки ассемблеров

- *Язык ассемблера* представляет собой язык машинных команд, в котором вместо численных кодов используются их символьные синонимы, более понятные программисту.
  - MOV – пересылка данных
  - ADD – сложение
  - SUB – вычитание
  - JMP – переход
  - ...
- *Ассемблер* – программа, транслирующая синонимы в машинные команды.



# Язык ассемблера (пример)

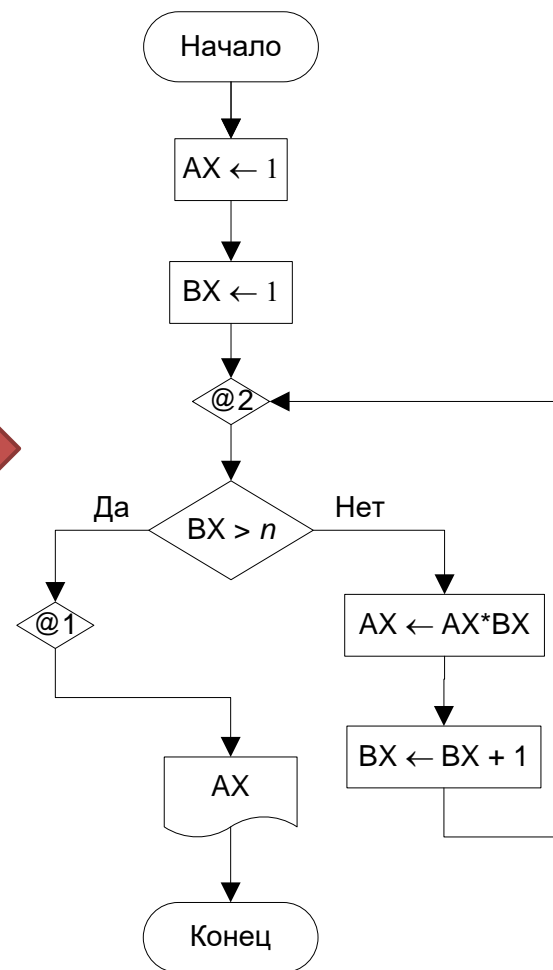
9



# Язык ассемблера (пример)

10

```
mov AX,1
mov BX,1
@2:
cmp BX,N
jg @1
imul BX
inc BX
jmp @2
@1:
```



# Язык ассемблера

11

- Достоинства:
  - ▣ Упрощение разработки.
  - ▣ Расширение области применения
    - класс компьютеров с данной архитектурой.
  - ▣ Возможность создания эффективных системных программ.
- Недостатки:
  - ▣ Сложность разработки прикладных программ.

Машинный код	Язык ассемблера
10111010	<b>mov</b> AX,1
00001100	<b>mov</b> BX,1
00000001	@2:
00101110	<b>cmp</b> BX,N
10001001	<b>jg</b> @1
00010110	<b>imul</b> BX
10010001	<b>inc</b> BX
00000010	<b>jmp</b> @2
10001011	@1:
00011110	
00101100	
...	

# Применение языков низкого уровня

12

- Низкоуровневое системное программирование
  - Разработка драйверов устройств
  - Разработка библиотек системных функций
  - Разработка модулей операционных систем
  - Разработка модулей систем управления базами данных
  - ...

# Языки программирования высокого уровня абстракции

13

## □ *Язык высокого уровня*

- близок к естественному языку;
- не зависит от архитектуры конкретного компьютера;
- позволяет программисту формулировать алгоритм в понятных ему терминах структур данных и операций над ними (вместо понятных компьютеру кодов операций, адресов ячеек памяти и др.).

# FORTRAN – первый язык высокого уровня

14

- FORTRAN (FORmula TRANslator) – первый язык высокого уровня, разработанный в 1954-57 гг. под руководством Дж. Бэкуса в компании IBM.
- Основное применение – научные и инженерные вычисления.
- Успешно пережил многие другие языки.
- В настоящее время имеется гигантское количество библиотек программ решения различных математических, физических, инженерных и др. задач на FORTRAN.



Джон Бэкус  
1924-2007

# ЯЗЫКИ ВЫСОКОГО УРОВНЯ (ПРИМЕР)

15

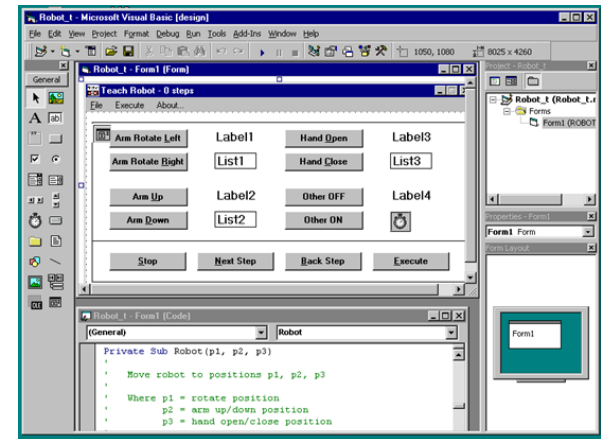
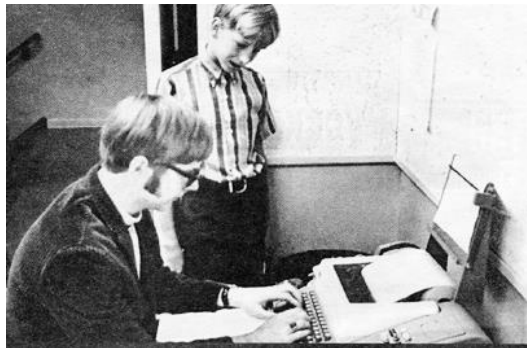
Машинный код	Язык ассемблера	FORTRAN
10111010	<b>mov</b> AX,1	<b>C</b> Вычисление факториала
00001100	<b>mov</b> BX,1	<b>INTEGER</b> N, F, i
00000001	@2:	<b>PRINT</b> *, 'Введите число'
00101110	<b>cmp</b> BX,N	<b>READ</b> *, N
10001001	<b>jb</b> @1	<b>DO</b> I = 1, N
00010110	<b>imul</b> BX	F = F * i
10010001	<b>inc</b> BX	<b>END DO</b>
00000010	<b>jmp</b> @2	<b>PRINT</b> * N,'! = ', F
10001011	@1:	<b>END</b>
00011110		
00101100 ...		

Введите число: 6  
6! = 720

# Язык BASIC

16

- BASIC (Beginner's All-Purpose Symbolic Instructions Code) создан в 1964 Томасом Куртцем и Джоном Кемени (США).
- Visual Basic является фирменным языком компании Microsoft для разработки приложений ОС Windows.





# Язык BASIC

17

FORTRAN	BASIC
<p><b>C</b> Вычисление факториала <b>INTEGER</b> N, F, i <b>PRINT</b> *, 'Введите число' <b>READ</b> *, N <b>DO</b> I = 1, N     F = F * i <b>END DO</b> <b>PRINT</b> * N, '! = ', F <b>END</b></p>	<p><b>REM</b> Вычисление факториала <b>DIM</b> N, F, i <b>AS INTEGER</b> <b>INPUT</b> "Введите число: ", N F = 1 <b>FOR</b> i = 1 <b>TO</b> N     F = F * i <b>NEXT</b> <b>PRINT</b> N, "! = ", F <b>END</b></p>

# Язык Pascal/Delphi

18

- Создан в 1971 профессором Никлаусом Виртом (Швейцария) для обучения студентов программированию.
- В 1984 Филипп Кан (США) разработал транслятор и среду программирования Turbo Pascal, основал компанию Borland.
- Позднее компания Borland выпустила среду программирования Delphi на базе обновленного языка Object Pascal.
- Pascal/Delphi – один из самых популярных языков разработки прикладных программ в различных областях.



# Язык Pascal/Delphi

19

BASIC	Pascal/Delphi
<pre>REM Вычисление факториала DIM N, F, i AS INTEGER INPUT "Введите число: ", N F = 1 FOR i = 1 TO N F = F * i NEXT PRINT N,"! = ", F END</pre>	<pre>{ Вычисление факториала } Program CalcFactorial; function Fact(N: Integer) : Integer; var i, F: Integer; begin     F := 1;     for i := 1 to N do         F := F * i;     Result := F; end; var N: Integer; begin     Write('Введите число: '); Read(N);     Write(N, '! = ', Fact(N)); end.</pre>

# Языки C и C++

20

- Язык C создан Деннисом Ритчи (США) в 1971 в ходе разработки ОС UNIX.
  - C – один из самых популярных языков разработки системных программ (разработка ОС UNIX/Linux и Windows, драйверы устройств и др.).
- Бьярн Страуструп (Дания) в 1983 дополнил C концепцией ООП и назвал язык C++.
  - C++ – один из самых популярных языков разработки прикладных программ в различных областях.



# Язык C

21

Pascal/Delphi	C
<pre>Program CalcFactorial; function Fact(N: Integer) : Integer; var i, F: Integer; begin     F := 1;     for i := 1 to N do         F := F * i;     Result := F; end; var N: Integer; begin     Write('Введите число: '); Read(N);     Write(N, '! = ', Fact(N)); end.</pre>	<pre>// Вычисление факториала int Fact(int N) {     int i, F;      F = 1;     for (i = 1; i&lt;=N; i++)         F *= i;     return F; } int N; void main (void) {     printf("Введите число: "); scanf(&amp;N);     printf("%d! = %d", N, Fact(N)); }</pre>

# ЯЗЫКИ ВЫСОКОГО УРОВНЯ

22

- Достоинства
  - Существенное упрощение разработки прикладной программы по сравнению с языками низкого уровня.
  - Возможность переноса разработанной программы на другие компьютеры.
- Относительные недостатки
  - Как правило, меньшая эффективность транслированных программ по сравнению с программами на языках низкого уровня.
- Языки *высокого* уровня необходимы для разработки *прикладных* программ.

# ЯЗЫКИ ВЫСОКОГО VS НИЗКОГО УРОВНЯ

23

- Языки высокого уровня
  - ▣ Наличие понятия *типа данных*.
  - ▣ Независимость от архитектуры конкретного компьютера (*мобильность программ*).
  - ▣ Развитые управляющие структуры и средства описания структур данных.
  - ▣ Близость к естественному языку.
- Языки низкого уровня
  - ▣ Отсутствие понятия *типа данных*.
  - ▣ Зависимость от архитектуры конкретного компьютера (*отсутствует мобильность программ*).
  - ▣ Примитивные управляющие структуры и средства описания структур данных.
  - ▣ Близость к машинному языку.

# Составные части языка

24

- *ЯП* = алфавит + синтаксис + семантика.
- *Алфавит* – символы для записи конструкций языка
  - ▣ из каких знаков составляются конструкции языка.
- *Синтаксис* – правила записи конструкций языка
  - ▣ какие конструкции принадлежат языку.
- *Семантика* – смысл конструкций языка
  - ▣ как конструкции должны обрабатываться компьютером.



# Алфавит языка программирования

25

- Алфавит языка состоит из букв, цифр и лексем.
- *Лексема* – наименьшая единица языка, имеющая самостоятельный смысл.
- Основные классы лексем:
  - ▣ *спецсимволы*
    - PAS: +, -, ^, >, <, <>, := и др.
    - C: +, -, \*, >, <, !=, = и др.
  - ▣ *ключевые слова языка*
    - PAS: begin, case, const, for, repeat, while, type и др.
    - C: main, switch, const, for, do, while, typedef и др.

# Способы задания синтаксиса

26

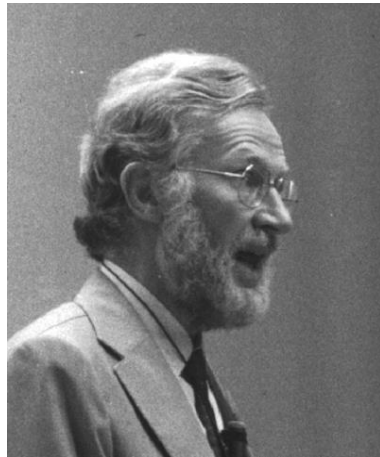
- Расширенные формулы Бэкуса-Наура (РБНФ)
- Синтаксические диаграммы

# Расширенные формулы Бэкуса-Наура

27



**Джон Бэкус**  
1924-2007



**Питер Наур**  
род. 1928

- Разработаны  
Дж. Бэкусом в 1960 г. и  
использованы П. Науром  
для описания синтаксиса  
языка программирования  
Алгол-60.


# Построение РБНФ

28

- *Нетерминальные символы* – заключаются в угловые скобки  $\langle \dots \rangle$  и используются для обозначения синтаксических конструкций языка.
- *Терминальные символы* образуют алфавит языка.
- *Метасимволы*
  - $::=$  есть по определению
  - | или
- *Дополнительные метасимволы (расширение БНФ)*
  - $[ \dots ]$  повторение символа 0 или 1 раз
  - $\{ \dots \}$  повторение символа произвольное число раз (в т.ч. нуль)
  - $( \dots )$  группировка символов

# РБНФ: примеры

29

- $\langle \text{пароль} \rangle ::= \langle \text{место} \rangle \langle \text{действие} \rangle \langle \text{предмет} \rangle ?$   
 $\langle \text{место} \rangle ::= \text{здесь} \mid \text{там}$   
 $\langle \text{действие} \rangle ::= \text{продается} \mid \text{ремонтируется}$   
 $\langle \text{предмет} \rangle ::= [ \text{славянский} ] \text{шкаф} \mid [ \text{никелированная} ] \text{тумбочка}$
- $\langle \text{ответ} \rangle ::= [ \langle \text{обычная фраза} \rangle ] ( \langle \text{провал} \rangle \mid \langle \text{норма} \rangle )$   
 $\langle \text{обычная фраза} \rangle ::= ( \text{Хорошая} \mid \text{Плохая} ) \text{погода, не правда ли?}$   
 $\langle \text{провал} \rangle ::= \{ \langle \text{предмет} \rangle ? \} \text{Да, он в отличном состоянии.}$   
 $\langle \text{норма} \rangle ::= \langle \text{предмет} \rangle \text{уже продан.}$
- $\langle \text{сказка} \rangle ::= \langle \text{белый бычок} \rangle \mid \langle \text{белый бычок} \rangle \langle \text{сказка} \rangle$   
 $\langle \text{белый бычок} \rangle ::=  \mid \langle \text{пусто} \rangle$   
 $\langle \text{пусто} \rangle ::=$
- $\langle S \rangle ::= a \langle A \rangle$   
 $\langle A \rangle ::= b \mid c \langle A \rangle$   
ac, abc, abbc, abbbc
- $\langle S \rangle ::= a \{ b \} c$   
ac, abc, abbc, abbbc

# Синтаксические диаграммы

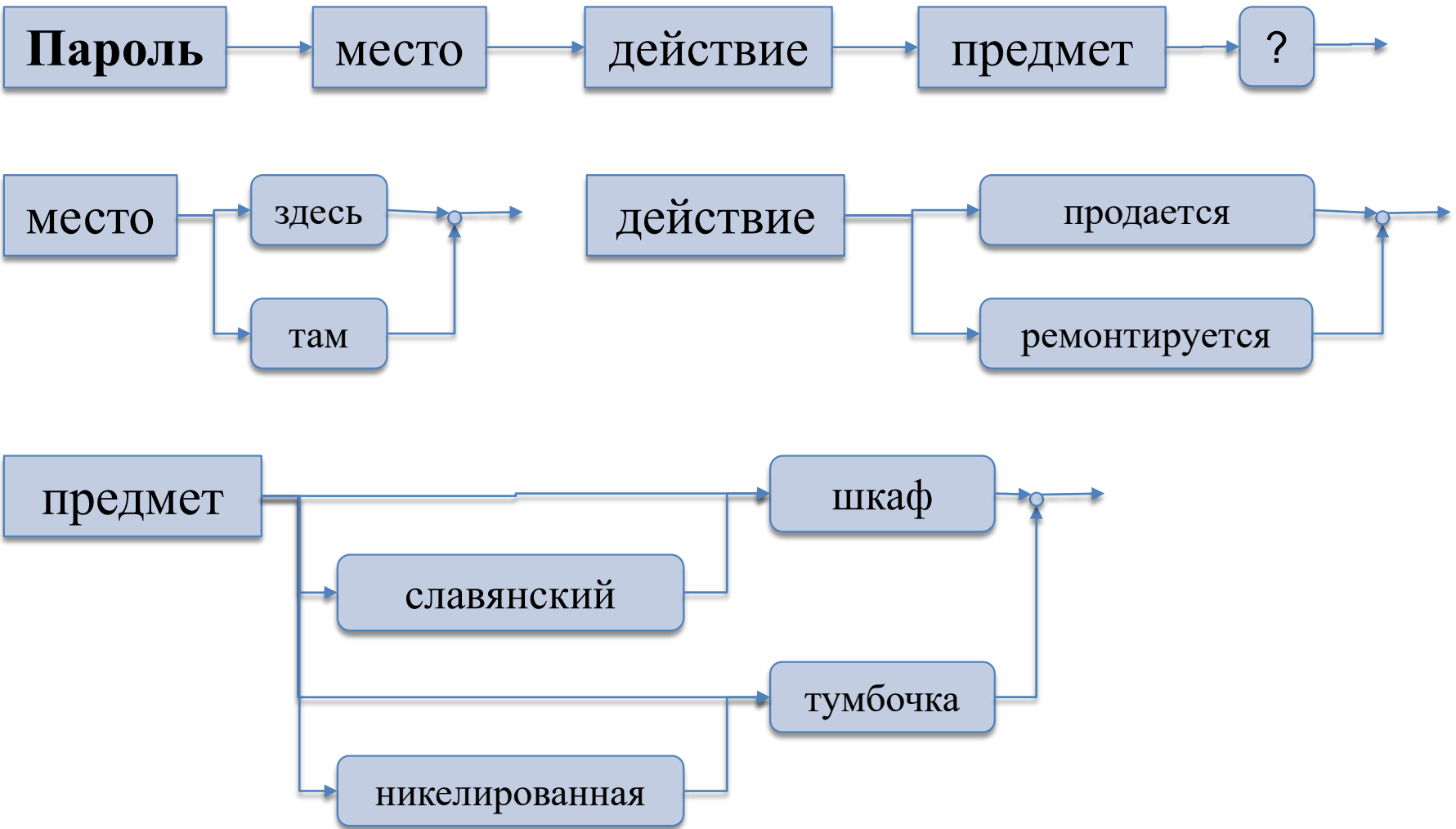
30

- *Синтаксические диаграммы* – графическое изображение РБНФ.
- Нетерминальные символы  
**Нетерминал**
- Терминальные символы  
**Терминал**
- Метасимволы



# Синтаксические диаграммы: пример

31



# Синтаксические диаграммы: пример

32

**Ответ**

обычная фраза

провал

норма

провал

Да, он в отличном состоянии.

предмет

?



# Синтаксис: основные ловушки

33

- Ограничения на длину идентификаторов  
`current_winner_number`  
`current_width`
- Регистр символов  
`Number`, `NUMBER`, `number`
- Комментарии
  - ▣ Однострочные  
`--` в Ada и PL/SQL, `//` в C++, `C` в Fortran
  - ▣ Многострочные  
`/* ... */` в C, `{ ... }` и `(* ... *)` в Pascal
  - ▣ Пропущенные операторы  
`{ a:=b+c;`  
`{ Комментарий }`

# Синтаксис: основные ловушки

34

## □ Похожие символы

Присвоить: **:=** в Pascal, **=** в C и Fortran

Равно: **=** в Pascal, **==** в C, **.eq.** в Fortran

## □ Значащие символы, или когда взрываются ракеты?

DO 10 I = 1,100 C

Это оператор цикла

DO 10 I = 1.100 C

Это оператор присваивания

C

**DO10I = 1.100**

C

т.к. пробел игнорируется!



# Семантика

35

- Пример: семантика оператора `if C then S1 else S2`
  - *Формальное описание*  
( $C \Rightarrow \text{Семантика } S1$ ) & ( $\neg C \Rightarrow \text{Семантика } S2$ )
  - *Неформальное описание*  
Проверяется условие после `if`; если результат Истина, то выполняется оператор, следующий после `then`, иначе выполняется оператор, следующий за `else`.
- Формальное описание семантики всех конструкций языка может быть очень сложным и, как правило, не применяется.

# Семантика языка и правильность программ

36

- Формальное описание семантики дает возможность доказать *правильность программы* в следующем смысле:
  - оператор – аксиома, описывающая преобразование состояние, для которого верно некоторое входное утверждение, в состояние, для которого верно некоторое выходное утверждение
  - семантика программы "вычисляется" путем построения входных и выходных утверждений на основе утверждений для отдельных операторов
  - результат вычислений – доказательство того, что если входные данные удовлетворяют утверждению на входе, то выходные данные удовлетворяют утверждению на выходе.

# Основные элементы языков программирования

37

- Типы данных, переменные, константы
- Операторы
- Выражения
- Подпрограммы
- Библиотеки

# Типы данных

38

- *Тип данных* – множество значений и множество операций над этими значениями.
- Определение множеств значений и операций зависит от языка программирования и его конкретной реализации.
- Примеры:
  - типы Integer в Pascal и int в C – это конечное множество целочисленных значений (в количестве  $\cong 65$  тыс. или 4 млрд. – в зависимости от компьютера) и конечного набора операций (+, -, \*, >, ...)
  - тип "массив" – индексированная совокупность элементов других (ранее определенных) типов с операцией индексации; в Ada над массивами определена также операция присваивания.

# Переменная, константа

39

- *Значение* – интуитивно понятный термин.
- *Литерал* – последовательность символов, которая в программе непосредственно задает значение (например: 201, 2.01, '0', "Строка", TRUE).
- *Представление* – строка битов для указанного значения (например: представление целого числа 201 есть 11001001).
- *Переменная* – имя ячеек, содержащих представление значения указанного типа, которое может изменяться во время выполнения программы.
- *Константа* – имя ячеек, содержащих представление значения указанного типа, которое *не* может изменяться во время выполнения программы.

# Операторы

40

- *Операторы* – конструкции языка для управления вычислениями.
  - *Оператор присваивания* изменяет значение указанной переменной на указанное новое значение.  
Pascal :  $A := A + 1;$   
C :  $m[a * i] = r[k + 2] * a;$
  - *Оператор вызова подпрограммы* активирует подпрограмму, передавая ей указанные параметры.
  - *Управляющие операторы* используются для изменения порядка выполнения вычислений.



# Управляющие операторы

41

- *Условные операторы* позволяют выбрать одну из нескольких альтернативных последовательностей управления.

```
if A>5 then
    Z:=1
else
    Z:=A+B;
case K of
    0..3 : P:=1;
    7, 9 : P:=233;
else
    P:=0;
end;
```

- *Операторы цикла* позволяют повторить выполнение последовательности операторов.

```
for (i=0; i<N; i++)
    A[i]=0;
while i<N do begin
    F := i*F;
    i:=i+1;
end;
```

# Выражения

42

- *Выражение* – правило для вычисления значения. Значение выражения имеет один и только один тип.
- Выражение состоит из *операндов* (переменные, константы и др.) и *операций* (отрицание, умножение и др.).
- Примеры выражений:
  - -X
  - A+B
  - (C mod 7 – D div 8) \* 5 + \$1A
  - Z/3.14 + 1E-3
  - Flag <> Error
  - 5<2
  - not (A or B and C)
  - 'TO BE ' + 'OR' + 'NOT' + 'TO BE'

# Подпрограммы

- *Подпрограмма* – сегмент программы, состоящий из объявлений данных и операторов, которые можно многократно *вызывать* (call) из различных частей программы.
- *Параметры подпрограммы* – последовательность значений, передаваемая в подпрограмму при ее вызове для задания различных вариантов выполнения, передачи исходных данных и получения результатов.

# Подпрограммы

44

- Подпрограмма – важный элемент программной структуры.
  - Каждая подзадача программы должна быть оформлена в виде подпрограммы.
  - Подпрограммы позволяют разделить программу на небольшие, хорошо понятные и читаемые части.
- Виды подпрограмм:
  - Pascal : процедуры (procedure), функции (function)
  - C : функции
  - FORTRAN: суб-рутины (SUBROUTINE)
  - OO-языки: методы

# Пример: программа с процедурами

45

```
{ myprog2.pas, 3-июля-24  
Иванов И.И.  
Пример программы с процедурами }
```

```
Program MyProg2;
```

```
procedure InpData (var A, B, C: Integer);  
{ Осуществляет ввод исходных данных с клавиатуры }  
begin  
...  
end;  
procedure Calculation (A, B, C: Integer; var Result: Real);  
{ Выполняет некоторые вычисления с исходными данными }  
begin  
...  
end;  
procedure OutData (Result: Real);  
{ Осуществляет вывод результатов вычислений на экран }  
begin  
...  
end;  
var  
A, B, C: Integer; R: Real;  
begin  
InpData (A,B,C);  
Calculation (A,B,C, R);  
OutData (R);  
end.
```

*определение*  
**Подпрограммы**  
*вызов*

# Библиотеки подпрограмм

46

- *Библиотека* – вспомогательная неисполняемая программная единица, содержащая определения подпрограмм и данных, которые могут быть вызваны из основной программы.
- Библиотеки необходимы для коллективной разработки больших программных систем (размером от 1 тыс. строк).
- Примеры библиотек:
  - Pascal : нет языковых средств
  - Object Pascal : unit
  - Modula : module
  - C : средства отдельной трансляции
  - Ada : package

# Пример: использование библиотеки

47

```
{ myprog2.pas, 3-июля-24
Иванов И.И.
Пример программы,
использующей библиотеку }
Program MyProg2;
uses UCalc;
procedure InpData (var A, B, C: Integer);
{ Осуществляет ввод исходных данных с клавиатуры }
begin
  ...
end;
procedure OutData (Result: Real);
{ Осуществляет вывод результатов вычислений на
экран }
begin
  ...
end;
var
  A, B, C: Integer; R: Real;
begin
  InpData (A, B, C);
  Calculation (A, B, C, R);
  OutData (Result);
end.
```

```
{ ucalc.pas, 3-июля-24
Петров П.П.
Пример библиотеки. }
unit Ucalc;
```

interface

```
procedure Calculation
(A, B, C: Integer; var Result: Real);
{ Выполняет некоторые вычисления с
исходными данными }
```

implementation

```
procedure Calculation
(A, B, C: Integer; var Result: Real);
{ Выполняет некоторые вычисления с
исходными данными }
begin
  ...
end;
```

```
...
end.
```

# Стандартизация языков

- *Стандарт языка* – официальный документ одной из международных организаций по стандартам (ISO – Int. Standards Org., ANSI – Amer. Nat. Standards Inst. и др.), в котором зафиксированы алфавит, синтаксис и семантика языка.
- *Мобильная программа* выполняется одинаково на различных компьютерах и/или операционных системах.
- Мобильность программ возможна, если для языка существует стандарт и различные трансляторы языка поддерживают данный стандарт.



# Заключение

- Язык программирования (ЯП) – система обозначений для абстрактного описания вычислений.
  - ▣ Составные части ЯП: алфавит, синтаксис, семантика.
    - Способы задания синтаксиса: РБНФ, синтаксические диаграммы
  - ▣ Основные элементы ЯП:
    - Типы данных, переменные, константы
    - Операторы
    - Выражения
    - Подпрограммы
    - Библиотеки
- Классификация ЯП
  - ▣ Языки низкого уровня абстракции
  - ▣ Языки высокого уровня абстракции