Capítulo II :
A Linguagem Pascal – Conceitos Fundamentais

(Niklaus Wirth, 1970)

1. Introdução

Um exemplo:

```
(* Programa para somar dois números reais *)
program somar (input, output);
var x, y, soma : real;
begin   read(x, y);
        soma:= x + y;
        write(soma)
end.
```

Observações:

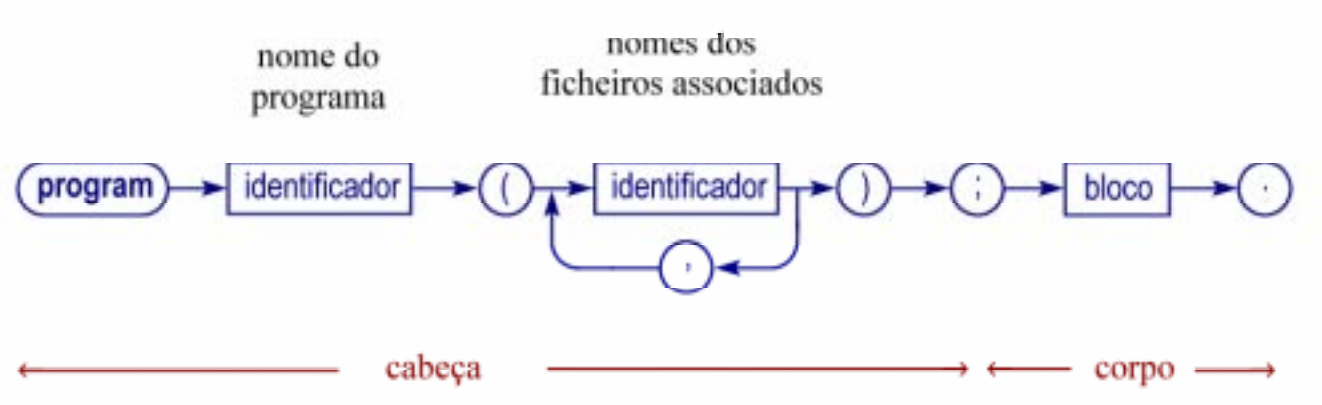
- Há Palavras-chave (p.ex. **begin** ou begin) que não podem ser usadas como identificadores
- As instruções são separadas por **;**
- O programa termina com **.**
- Há Instruções Compostas, delimitadas por **begin** … **end**
- Todas as variáveis são declaradas, de acordo com o seu tipo
- Os Comentários são assinalados por (* … *) ou por { …}
- As instruções são escritas em letras minúsculas
- A "indentação" torna o programa mais legível
- A Cabeça de um programa define o seu nome e o nome dos Ficheiros de Entrada e de Saída
- A Instrução de Atribuição é indicada por **:=**
- …

2. Os Diagramas de Sintaxe da Linguagem Pascal:

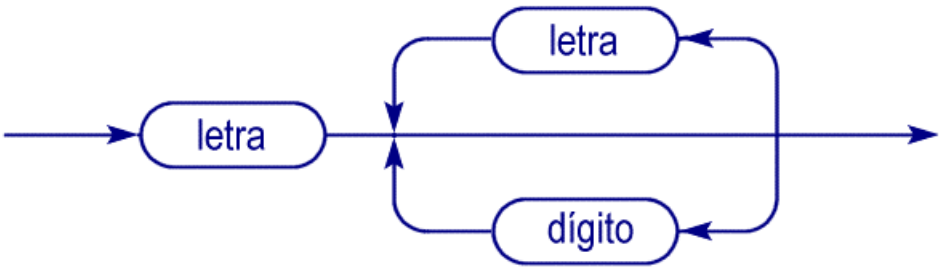
A sintaxe da linguagem Pascal está completamente definida num conjunto de diagramas.

> Um programa Pascal está sintaticamente correcto se e só se corresponder a um caminho ao longo dos diagramas.

Programa:



Identificador:



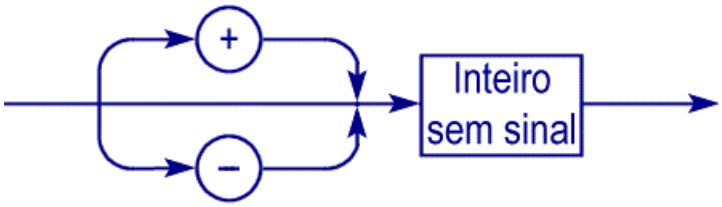
i.e. qualquer **sequência de letras e dígitos, começando por uma letra**.

O comprimento do identificador (número de letras/dígitos reconhecidos) varia consoante o computador utilizado.

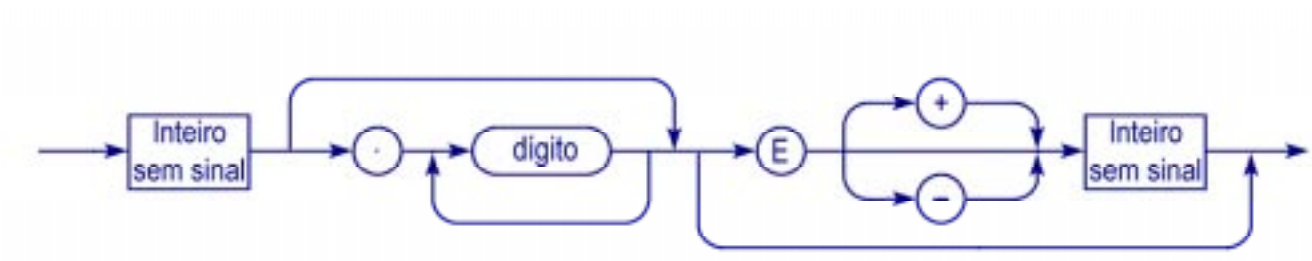
Inteiro sem sinal:

O número máximo de dígitos depende da representação de inteiros utilizada. Num computador de 16 bits, o intervalo permitido é [-32768 , 32767].

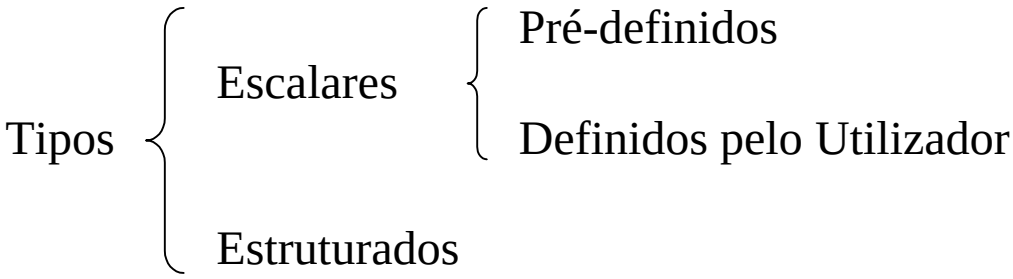
Inteiro:



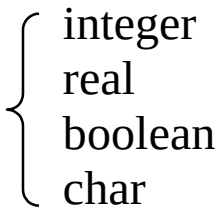
Número sem sinal:



3. Os Tipos de Informação:



3.1. Os Tipos Escalares Pré-definidos (standard):



3.1.1. Os Números Inteiros (integer):

A grandeza máxima depende da representação de inteiros utilizada.
Num computador de 16 bits, o intervalo permitido é [-32768 , 32767].

maxint = 32767 (constante pré-definida)

Operadores Aritméticos com Operando(s) e Resultado Inteiro:

+	adição
–	subtracção (operador binário) ou simétrico (operador unário)
*****	multiplicação
div	quociente da divisão inteira
mod	resto da divisão inteira: a **mod** b = a – ((a **div** b) * b)

Operadores Relacionais:

= <> < <= >= >

Funções com Argumento e Resultado Inteiro:

abs( )	valor absoluto	abs(n) = |n|
sqr( )	quadrado	sqr(n) = n^2
succ( )	sucessor	succ(n) = n+1
pred( )	predecessor	pred(n) = n−1

3.1.2. Os Números Reais (real):

A **grandeza** máxima (do expoente) e a **precisão** (número de algarismos significativos da mantissa) dependem da representação de reais no computador utilizado. A representação interna não é exacta e os resultados das operações podem introduzir imprecisões.

Operadores Aritméticos com Resultado Real se pelo menos um dos Operandos fôr Real:

+	adição
−	subtracção (operador binário) ou simétrico (operador unário)
*	multiplicação
/	quociente real, mesmo que os dois operandos sejam inteiros

Operadores Relacionais:

= <> < <= >= >

(Mas não é “seguro” testar a igualdade/desigualdade com reais!)

Funções com Argumento Real e Resultado Inteiro:

trunc( ) truncatura

round( ) arredondamento: $round(x) = \begin{cases} trunc(x+0.5) \text{ se } x \geq 0 \\ trunc(x-0.5) \text{ se } x < 0 \end{cases}$

Funções com Resultado Real se o Argumento fôr Real:

abs( ) valor absoluto
sqr( ) quadrado

Funções com Resultado Real (Argumento Inteiro ou Real):

sin( ) seno (argumento em radianos)
cos( ) cosseno (argumento em radianos)
arctan( ) arco tangente (resultado em radianos)
exp( ) exponencial (base **e**)
ln( ) logaritmo natural (base **e**)
sqrt( ) raíz quadrada

3.1.3. Os Valores Lógicos (boolean):

{ false , true }

Operadores Lógicos:

and	conjunção	∧
or	disjunção	∨
not	negação	¬

Tabela de Verdade:

a	b	a and b	a or b	not a
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

Função com Argumento Inteiro e Resultado Lógico:

odd( ) ímpar: odd(n) = { true se n é um número ímpar ; false se par

3.1.3. Os Caracteres (char):

O Conjunto Total dos Caracteres varia com o Computador e o Compilador, mas incui sempre:

{A, B, C, …, Z}	subconjunto das letras maiúsculas
{a, b, c, …, z}	subconjunto das letras minúsculas
{0, 1, 2, …, 9}	subconjunto dos dígitos
{+, –, ∗, /, , (,), …}	símbolos usuais

- Cada Caracter é representado entre plicas (′ ′), p.ex.:
  - ′a′ representa uma letra e não uma variável
  - ′4′ não representa um número
  - ′+′ não representa um operador aritmético.

- Cada Caracter tem uma Representação Interna (ver tabela do Código ASCII) que é um Número Natural.

- Os elementos de cada Subconjunto estão ordenados e são válidos os operadores relacionais:

  ′A′ < ′B′ ′A′ < ′Z′ ′A′ = ′A′ ′4′ < ′5′

Funções de Transferência (entre Caracteres e Inteiros):

ord( )	ordem (Representação Interna) de um Caracter
chr( )	Caracter, se existir, representado por um Número Natural

- ord( ) e chr( ) são funções inversas, i.e.:

  chr(ord(c)) = c ord(chr(i)) = i

- Os Operadores Relacionais entre Caracteres correspondem aos Operadores Relacionais entre os Inteiros da sua Representação, p.ex.:

  c1 < c2 <=> ord(c1) < ord(c2)

Funções de Argumento e Resultado Caracter:

succ( )	sucessor:	chr(ord(c) + 1)
pred( )	predecessor:	chr(ord(c) – 1)

Um Exemplo: Programa para calcular o valor de uma Expressão Aritmética com dois operandos.

Especificação:

Entrada: Pretendemos escrever uma expressão do tipo:

2.5+3.0

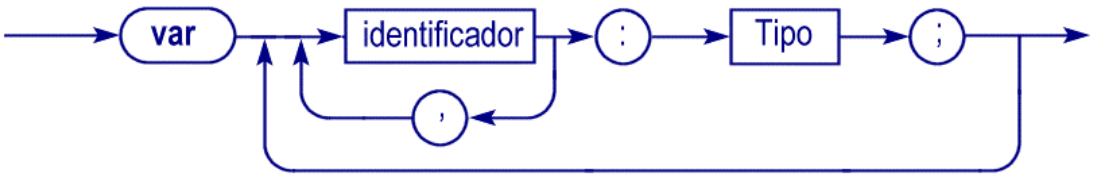
Saída: e obter o resultado na forma:

2.5+3.0 = 5.5

Programa em Pascal:

```
program calculadora(input, output);
var   operador: char;
      x, y, resultado: real;

begin   read(x, operador, y);
        case  operador of
              '+': resultado := x+y;
              '–': resultado := x–y;
              '*': resultado := x*y;
              '/':  resultado := x/y
        end;
        write(x, operador, y, '=', resultado);
end.
```

4. Declaração de Variáveis:



Exemplo:

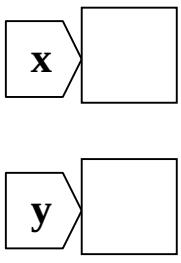
```
var  operador, letra: char;
     x, y, resultado: real;
     i, j, k, soma: integer;
     bissexto: boolean;
```

A Declaração de uma Variável:

- **Cria o Identificador;**
- **Cria uma Célula de Memória com o Formado pretendido;**
- **Associa o Identificador ao Endereço de Memória dessa Célula.**

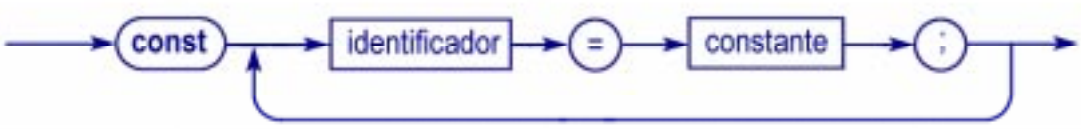
Exemplo:

```
var  x, y: real;
```



Foram definidas duas células em formato de tipo real, acessíveis através dos identificadores x e y.
Estes espaços permanecem "vazios" (variável indefinida), até que uma instrução (atribuição, leitura, …) os venha preencher.

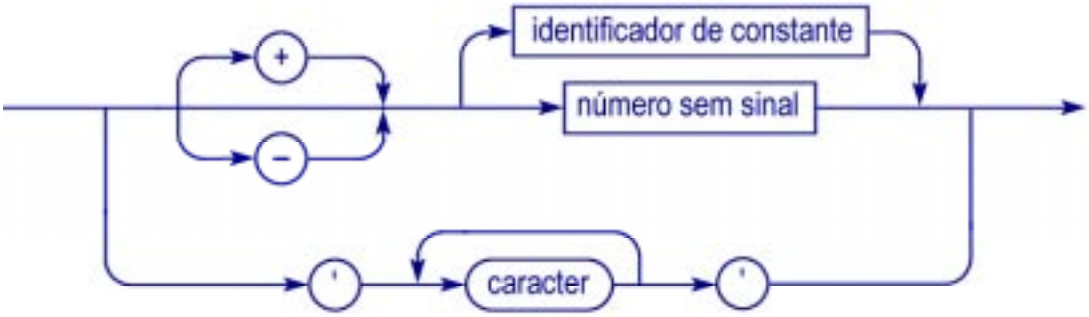
5. Declaração de Constantes:



Exemplo:

```
const pi=3.14159265;
      e=2.7182;
      limite=100;
      asteriscos='*****';
```

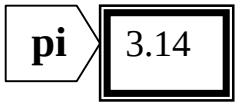
Constante:



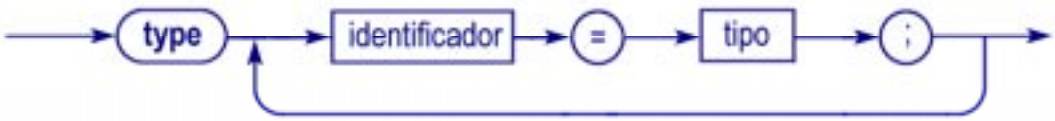
O efeito da Declaração de uma Constante é semelhante ao da Declaração de uma Variável, com a diferença de que o Valor da Constante não pode ser alterado no decorrer do programa.
O Tipo da Constante é definido implicitamente.

Exemplo:

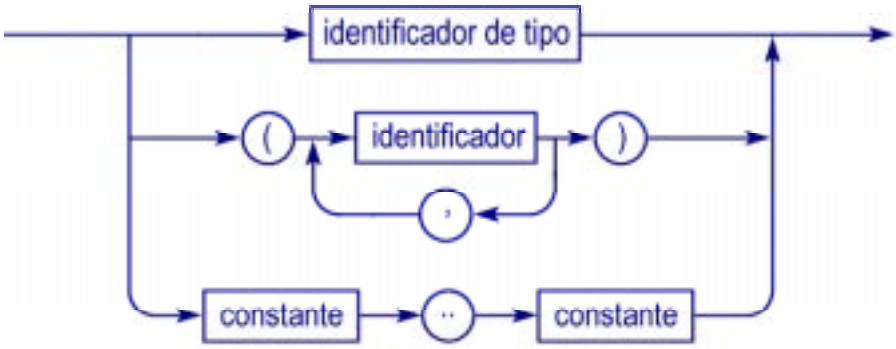
```
const pi=3.14159265;
```



6. Declaração de Tipos:



Tipo:



Os Tipos Definidos pelo Utilizador:

6.1. Definição por Enumeração:

Exemplo:

```
type  diadasemana=(segunda, terca, quarta, quinta,
      sexta, sabado, domingo);
      cor=(amarelo, verde, vermelho, azul);
      naipe=(ouros, copas, paus, espadas);

var   feriado: diadasemana;
      tinta: cor;
      trunfo: naipe;
```

O modo como os identificadores são listados define uma relação de ordem, por isso são válidos os Operadores Relacionais, bem como as funções pred( ), succ( ) e ord( ).

Exs.
```
segunda<terca
amarelo<=vermelho
succ(sabado)=domingo
ord(verde)=1                (de 0 a n–1)
```

6.2. Definição por Subdomínio:

(válido para todos os tipos Escalares, excepto os Reais))

Exemplo:

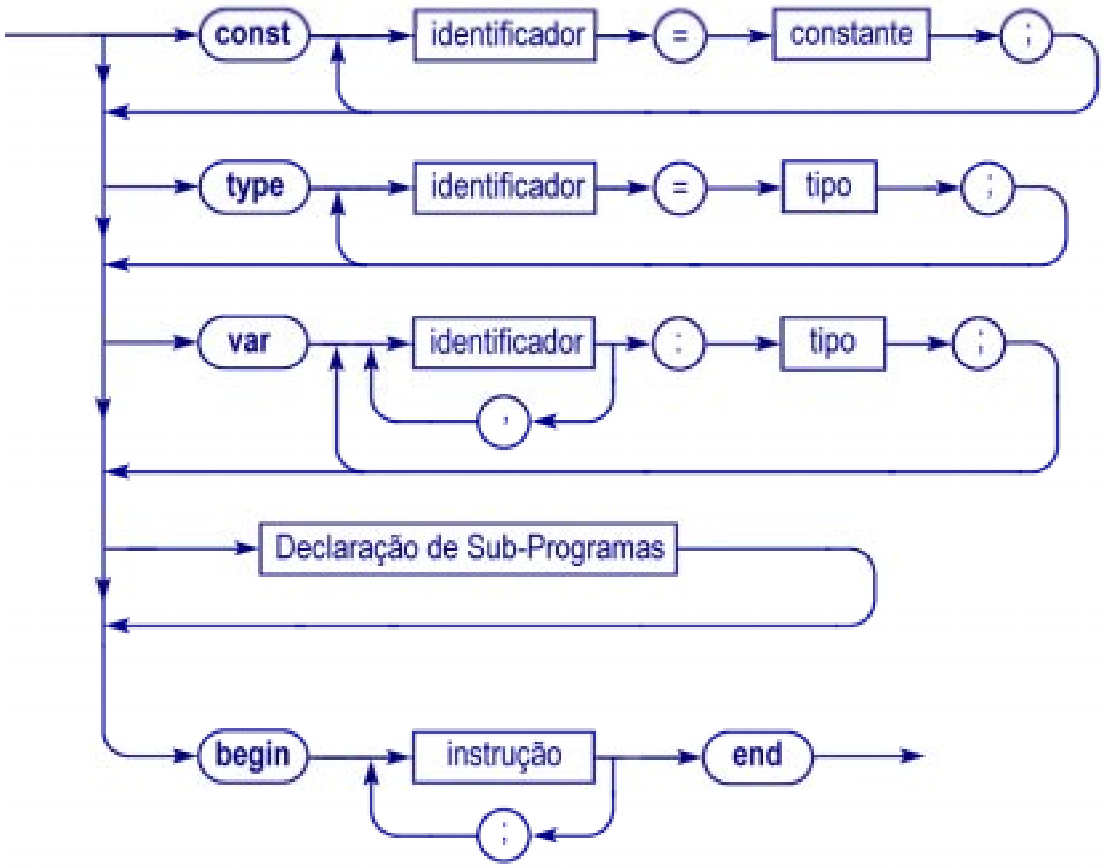
```
type  digito=0..9;
      letramaiuscula='A'.. 'Z';
      diautil=segunda..sexta;

var   i, j: digito;
      inicial: letramaiuscula;
```

O Tipo também pode ser definido implicitamente na própria Declaração da Variável.

Exemplo:

```
var   i, j: 0..9;
      inicial: 'A'.. 'Z';
```

7. O Bloco de um Programa:



Problema:

Calcular a soma de duas expressões horárias, dadas na forma **hh/mm/ss**.

Exemplo:

```
   4h    25m    32s
+ 12h    44m    50s
-------------------
  17h    10m    22s
```

Variáveis utilizadas:

```
    h1    m1    s1
+   h2    m2    s2
-----------------
  hres  mres  sres
```

Programa em Pascal:

```
(* Programa para Somar Horas, Minutos e Segundos *)
program somartempos(input, output);

type   natural = 0..maxint;
var    h1,h2,hres, aux : natural;
       m1, m2, mres, s1, s2, sres : 0..60;

begin  (* Leitura dos Dados*)
       write('Escreva o valor da primeira parcela,
              em horas, minutos e segundos');
       read(h1, m1, s1);
       writeln;
       write('Escreva o valor da segunda parcela,
              em horas, minutos e segundos');
       read(h2, m2, s2);
       writeln;
       (* Calculo da soma*)
       sres:=(s1+s2) mod 60;
       aux:=(s1+s2) div 60;
       mres:=(m1+m2+aux) mod 60;
       aux:=( m1+m2+aux) div 60;
       hres:=h1+h2+aux;
       (* Escrita do Resultado*)
       writeln('Soma =',
              hres, 'horas', mres, 'minutos', sres, 'segundos')
end.
```

Exercícios:

- O programa repete operações. Como evitá-lo?
- E se o resultado fôr superior a 24 horas?
- Como apresentar o resultado no formato da própria adição?

Programa em Pascal (2ª Versão):

```
…

(* Calculo da soma*)
aux  := s1 + s2;
sres := aux mod 60;
aux  := aux div 60;
aux  := m1 + m2 + aux;
mres:= aux mod 60;
aux  := aux div 60;
hres := h1+h2+aux;
…
```