5 Elementos básicos

Neste capítulo, discutem-se os elementos básicos que compõem a linguagem *Pascal*. Em primeiro lugar, são apresentados os símbolos e as palavras reservadas usados na construção de um código-fonte (seções 5.1 e 5.2, respectivamente). Na seção 5.3, são discutidas as alternativas para a inserção de comentários no código-fonte. A seção 5.4 trata da criação de identificadores válidos para a linguagem *Pascal*.

A partir da seção 5.5, são apresentados os tipos de dados utilizados em *Pascal*. São elencados desde os tipos básicos, como o inteiro e o caractere, até aqueles mais complexos, como os que são voltados à construção de estruturas de dados.

5.1 Símbolos

É possível inserir quaisquer símbolos num arquivo-fonte em *Pascal*. O compilador dará um significado adequado, a cada símbolo presente no código.

Os símbolos seguintes têm significados especiais, por se tratarem de operadores ou indicadores de que determinada parte do código não se trata de uma instrução:

e/ou símbolos que podem indicar ao compilador que aquela parte do código onde se encontram não é uma instrução:

```
+ * / = < > [ ] . , ( ) : ^ @ { } $ #
```

Os símbolos abaixo seguem a mesma regra, mas são sempre usados em par (note que '`<=`', sem espaço, é diferente de '`< =`');

```
<= >= := += -= *= /= (* *) (. .) //
```

Dentro de uma `string`,tais símbolos fazem parte da palavra e não serão interpretados pelo compilador de outra forma.

5.2 Palavras reservadas

Palavras reservadas são componentes da própria linguagem *Pascal* e não podem ser redefinidas, ou seja, denominar elementos criados pelo programador. Por exemplo, `begin`, palavra reservada que indica o início de um bloco de código, não pode ser nome rótulos, constantes, tipos ou quaisquer outros identificadores no programa.

O compilador *Pascal* não faz distinção entre maiúsculas e minúsculas. Assim, tanto identificadores quando palavras reservadas podem ser grafados com variações de maiúsculas e minúsculas sem alteração de significado: `begin`, `BEGIN`, `Begin` ou `BeGiN`.

Conforme o guia de referência da linguagem *Free Pascal*, versão de dezembro de 2008, as palavras reservadas são divididas em três grupos. O primeiro contém as palavras reservadas para a versão do *Turbo Pascal*[7]. O segundo compreende as palavras reservadas do *Delphi*, e o último encerra as do *Free Pascal*.

[7] `http://pt.wikipedia.org/wiki/Turbo_Pascal`

Esta diferença existe porque no *Turbo Pascal* as palavras do *Delphi* não são reservadas.

A tabela abaixo apresenta uma lista das palavras reservadas reconhecidas pelo compilador *Turbo Pascal*:

absolute	else	nil	set
and	end	not	shl
array	file	object	shr
asm	for	of	string
begin	function	on	then
break	goto	operator	to
case	if	or	type
const	implementation	packed	unit
constructor	in	procedure	until
continue	inherited	program	uses
destructor	inline	record	var
div	interface	reintroduce	while
do	label	repeat	with
downto	mod	self	xor

Tabela 1: Palavras reservadas do *Turbo Pascal*

São palavras reservadas do *Delphi*:

as	class	except
exports	finalization	finally
initialization	is	library
on	out	property
raise	threadvar	try

Tabela 2: Palavras reservadas do *Free Pascal*

O compilador *Free Pascal* suporta todas as palavras reservadas descritas acima e mais as seguintes:

dispose	false	true
exit	new	

Tabela 3: Palavras reservadas do *Delphi*

Atenção: as palavras reservadas variam conforme o compilador utilizado. Portanto, é indispensável consultar o guia de referência para a versão do compilador que tenha em mãos.

5.3 Comentários

Os comentários são colocados no arquivo-fonte para ajudar a entender o que determinado bloco de código faz. Comentários não são interpretados pelo compilador.

O *Free Pascal* suporta três tipos de comentários. Dois destes já eram utilizados por compiladores *Pascal* antigos; um terceiro vem da linguagem C).

A seguir, uma lista de comentários válidos:

```
(* Eis um comentario.
   Ele pode ter mais de uma linha *)

{ Um outro comentario.
  Tambem pode ter mais de uma linha}

// Um comentario que compreende apenas ate o final da linha
```

Você pode colocar comentários dentro de comentários, com a condição de que os tipos de comentários sejam diferentes. Exemplos:

```
(* Isto eh um comentario.
   Ele pode ter mais de uma linha
   {
      Aqui esta outro comentario dentro do primeiro
      Isso eh totalmente valido
      // Este eh um terceiro comentario. Tambem valido.
   }
)*

// Isto eh um comentario. (* Este eh outro comentario
   Porem eh um comentario invalido, pelo excesso de linhas
*)

{ Primeiro comentario
  {
     Segundo comentario
     O primeiro comentario sera fechado a seguir
  }
  E a chave a seguir esta sobrando
}
```

5.4 Identificadores

Identificadores são utilizados para denominar programas, rótulos, constantes, variáveis, tipos, procedimentos, funções e `units` criados pelo programador.

São regras, em *Pascal* para se constituir corretamente um identificador:

1. Um identificador não pode ser igual a uma palavra reservada;
2. Um identificador deve ter no máximo 255 caracteres;
3. Os símbolos que constituem um identificador podem ser:

- letras: ‘A’ até ‘Z’ ou ‘a’ até ‘z’;
- dígitos: 0 até 9;
- um sublinhado: _.

4. O primeiro símbolo deve ser uma letra ou o sublinhado.

Assim, não são aceitos espaços ou caracteres especiais em qualquer parte de um identificador. Até mesmo o cedilha (‘ç’) é considerado um caractere especial.

São exemplos de identificadores válidos:

- `a`, `a1`, `b568`, `codigo`, `raizquadrada`, `preco_especial`;
- `_veiculo`, `elemento_do_conjunto`, `raiz_quadrada`, `b54_`;
- `CorVeiculo`, `Ano`, `MES`, `SaLdO`, `Parcela_Quitada`;
- `Este_nome_eh_muito_longo_e_raramente_deve_ser_usado_mas_eh_valido`.

Exemplos de identificadores **inválidos**:

- `12porcento`, `4por4`: iniciam por número;
- `cor veiculo`: contém espaço em branco;
- `preço`: possui um caracter inválido (cedilha);
- `pássaro`: apresenta acentuação;
- `%b54_`: não inicia por letra ou por sublinhado.

Observações:

- Não se pode declarar uma variável com o mesmo nome que foi atribuído ao programa ou a qualquer outro identificador previamente declarado no mesmo escopo[8];
- Como o compilador *Pascal* ignora o estado de maiúsculas e minúsculas, os identificadores podem ser grafados de maneiras diferentes e ter o mesmo significado, como por exemplo: `saldo`, `SALDO`, `Saldo` ou `SaLdO`.

[8]Para informações sobre escopo, veja a seção 4.1.5)

5.5 Tipos de dados em *Pascal*

A linguagem *Pascal* é, por definição, uma linguagem fortemente tipada. Ser tipada significa que as variáveis e constantes possuem um tipo precisamente definido, isto é, o compilador deve conhecer detalhes de quantidade de bytes para seu armazenamento e o formato de conversão decimal/binário.

Já, uma linguagem dita fortemente tipada, com raras exceções, não permite que tipos sejam misturados. Tal ocorrência provoca erro de compilação ou até mesmo durante a execução do programa.

Na seção 4.1.3 mostrou-se que é possível declarar novos tipos sob demanda do programador. Na presente seção, serão enumerados todos os tipos considerados *básicos* para o compilador *Free Pascal*. Convém lembrar que isto pode mudar de um compilador para outro. Logo, procure sempre informações atualizadas para a versão do compilador que tem em uso.

Os tipos que interessam para a disciplina introdutória de programação são os seguintes:

- a família de tipos ordinal;
- a família de tipos real;
- `boolean` (booleano);
- `char` (caractere);
- `string` (cadeia de caracteres);
- `array` (matriz);
- `record` (registro);
- `file` (arquivo).

Contudo, é importante observar que para níveis avançados de programação existem outros tipos pré-definidos na linguagem. Consultar o guia de referência para maiores informações.

5.5.1 A família de tipos ordinal

Os tipos ordinais representam números conhecidos em linguagem informal como sendo os *tipos inteiros*.

Segundo o guia de referência da linguagem, em sua versão de dezembro de 2008, tem-se que, com exceção do `int64`, `qword` e dos tipos reais, todos os tipos básicos são do tipo ordinal.

Conservam as seguintes características:

1. São enumeráveis e ordenados, ou seja, é possível iniciar uma contagem de um por um destes números em uma ordem específica. Esta propriedade implica que as operações de incremento, decremento e ordem funcionam;

2. Existem menores e maiores valores possíveis.

A seguir é relacionada uma lista com os tipos inteiros e respectivas faixas de valores. O programador deve escolher o tipo mais conveniente à representação da variável, pois a diferença básica entre eles é a quantidade de bytes usada em memória e o fato de necessitar que números negativos sejam representados. Para um estudante de Ciência da Computação, a diferença está na interpretação do número em binário, se está em complemento de 2 ou em "binário puro".

Tipo	Faixa dos limites	Tamanho em bytes
`byte`	0 ..255	1
`shortint`	-128 .. 127	1
`smallint`	-32768 .. 32767	2
`word`	0 .. 65535	2
`integer`	`smallint` ou `longint`	2 ou 4
`cardinal`	`longword`	4
`longint`	-2147483648 .. 2147483647	4
`longword`	0 .. 2147483647	4
`int64`	-9223372036854775808 .. 9223372036854775807	8
`qword`	0 .. 18446744073709551615	8

Tabela 4: Tipos ordinais em *Pascal*

Funções predefinidas relacionadas: `abs`, `chr`, `dec`, `inc`, `int`, `odd`, `ord`, `power`, `pred`, `random`, `randomize`, `round`, `succ`, `trunc` e `val`. Vide anexo A.

5.5.2 Tipo enumerável

Os tipos enumeráveis são definidos pelo programador. Possuem uma certa seqüência de possíveis valores ordinais que as variáveis daquele tipo poderão assumir.

Exemplo:

```
program tipo_enum;
 type
    TipoEsportes = (Futebol, Basquete, Volei);
 var
    Esporte1, Esporte2: TipoEsportes;
 begin
    Esporte1 := Futebol;
    Esporte2 := Basquete;
    { Perceba que Esporte1 tem o valor Futebol, e nao 'Futebol', pois
        nao se trata de um tipo string. }
    if Esporte1 = Futebol then
       writeln('O primeiro esporte eh Futebol');
 end.
```

5.5.3 Tipo sub-faixa

Os tipos sub-faixas possuem valores numa determinada escala (que pode ser constituída de números ou letras.

Exemplo:

```
program tipo_sub_faixa;
 type
    Tipo_um_a_cinco = 1 .. 5; // Vai de 1 ate 5
    Tipo_a_a_f = 'a' .. 'f';  // Vai de a ate f
 var
    Numero: Tipo_um_a_cinco;
    Letra: Tipo_a_a_f;
 begin
    Numero := 3; // OK. Esta na faixa
    writeln(Numero);
    Letra := 'p'; // OK. Esta na faixa
    writeln(Letra);
    Letra := 'z'; // Esta fora da faixa estabelecida. O compilador
       mostrara um warning
    writeln(Letra);
 end.
```

5.5.4 A família de tipos real

Os tipos reais compreendem aqueles representados internamente pelo computador e interpretados como sendo de ponto flutuante[9].

Assim como no caso dos ordinais, variam basicamente na quantidade de bytes usada para cada endereço de memória e na representação de sinais, seja no expoente ou na mantissa.

Abaixo apresenta-se uma lista com os tipos reais e suas faixas de valores, conforme o guia de referência em sua versão de dezembro de 2008.

Tipo	Faixa dos limites	Tamanho em bytes
`real`	depende da plataforma	4 ou 8
`single`	1.5E-45 .. 3.4E38	4
`double`	5.0E-324 .. 1.7E308	8
`extended`	1.9E-4932 .. 1.1E4932	10
`comp`	-2E64+1 .. 2E63-1	8
`currency`	-922337203685477.5808 .. 922337203685477.5807	19-20

Tabela 5: Tipos reais em *Pascal*

Aqui reside uma das raras exceções da checagem de tipo da linguagem, pois uma variável do tipo real pode receber a soma de um real com um inteiro. Neste caso,

[9]Apesar de estarmos no Brasil, em *Pascal* os números em ponto flutuante são escritos com um ponto, não com vírgula.

antes da operação, o computador faz a conversão do inteiro para um formato em ponto flutuante.

Exemplo:

```
program type_real;
 var
    a, b, c: real;
    d: integer;
 begin
    a := 14.5;
    b := sqrt(10); // b eh igual a raiz de 10
    d := 10; / d eh inteiro
    c := a + d; // c eh a soma de um real com inteiro
    writeln('a=', a, ' b=', b, ' c=', c, ' d=', d);
    writeln;
    writeln('a=', a:0:2); // ajusta para duas casas decimais apos o
       ponto
  end.
```

Funções predefinidas relacionadas: `abs`, `arctan`, `cos`, `exp`, `frac`, `int`, `log`, `pi`, `power`, `random`, `randomize`, `round`, `sin`, `sqr`, `sqrt`, `trunc` e `val`. Vide anexo A.

5.5.5 Tipo `boolean` (booleano)

O `boolean` (booleano) é um tipo lógico que assume apenas um entre dois possíveis valores: `false` ou `true`, que equivalem respectivamente a falso ou verdadeiro (0 ou 1).

Sempre que um valor diferente de 0 for associado a uma variável booleana, esta será verdadeira.

Exemplo:

```
program type_bool;
var
 a, b, c, d: boolean;
begin
 a := false;
 b := true;
 c := boolean(0);     // valor igual a zero: false
 d := boolean(-10);   // valor  diferente de zero: true
 writeln('Variaveis: a=', a, ' b=', b, ' c=', c, ' d=', d);
end.
```

As variáveis ou constantes do tipo `boolean` são elementos fundamentais na definição e entendimento de *expressões booleanas*, que serão apresentadas na seção 6.2.

5.5.6 Tipo `char` (caractere)

O tipo `char` (ou caractere) armazena apenas um caracter da tabela ASCII[10] e ocupa exatamente um byte de memória. Na realidade, não é o caractere que é armazenado na memória, mas o código ASCII correspondente.

O código ASCII do caractere 'A', por exemplo, é 65. Então, se for atribuído 'A' a uma variável do tipo caractere, na verdade será atribuído o valor 65 para esta variável. Pode-se atribuir diretamente o valor ASCII a uma variável caractere inserindo-se o sustenido na frente do número correspondente (por exemplo, `a := #65`).

Exemplo:

```
program type_char;
 var
    a, c, d: char;
  b: integer;
 begin
  a := 'A';           // armazena 65 na variavel a, que equivale a 'A'
  b := ord('A') + 2; // obtem o codigo ASCII de 'A' (65) e acrescenta 2
                      // observe que b, que recebe isto, eh um inteiro
  c := chr(b);        // armazena em c o caractere correspondente ao
                      // valor 67 em ASCII
  d :=  #68;
  writeln('Variaveis: a=', a, ' b=', b, ' c=', c, ' d=', d);
  // retorna a=A b=67 c=C
end.
```

Por se tratar de um tipo ordinal, o tipo char pode ser utilizado em comandos `case` (ver seção 7.5.3).

Funções predefinidas relacionadas: `chr`, `lowercase`, `ord` e `upcase`. Vide anexo A.

5.5.7 Tipo `string` (cadeia de caracteres)

O tipo `string` é utilizado para armazenar palavras, isto é, seqüências de símbolos ASCII.

Em *Pascal*, as palavras, assim como o tipo `char`, devem estar entre apóstrofos (aspas simples). Pode-se concatenar *strings* e caracteres usando o operador de adição ('+').

Exemplo:

```
program type_str;
var
  a, b, d: string;
  c: char;
begin
  a := 'hello ';
  b := 'world';
  c := '!';
  d := a + b + c;
```

[10] `http://pt.wikipedia.org/wiki/ASCII`

```
  writeln(d);      // hello world!
end.
```

Funções predefinidas relacionadas: `concat`, `copy`, `delete`, `insert`, `length`, `lowercase`, `pos`, `upcase` e `val`. Vide anexo A.

5.6 Tipo `array` (matriz)

Em sua forma elementar, o tipo `array` é utilizado para armazenar, sob um mesmo nome de variável, uma quantidade fixa de elementos do mesmo tipo. Por isto, o tipo `array` também é conhecido como um *arranjo homogêneo*. Para formas de uso mais complexas recomendamos a leitura do guia de referência da linguagem *Pascal*.

A declaração de uma variável do tipo `array` é feita assim:

```
array [<lista de tipos enumeraveis>] of <tipo>
```

onde `<tipo>` é qualquer tipo previamente definido pelo programador ou pré-definido pela linguagem, e `<lista de tipos enumeraveis>` é uma lista de faixas de números separadas por vírgula.

Por restrição dos tipos enumeráveis, os valores devem ser do tipo ordinal e conhecidos em tempo de compilação.

Os casos mais comuns de matrizes são de uma e de duas dimensões, embora possam ser declaradas com dimensões maiores. As de uma única dimensão são costumeiramente denominadas *vetores*, as de duas dimensões são chamadas de *matrizes* propriamente ditas.

Exemplos:

```
program tipo_array_1;
 const
    MAX = 10; MIN=-5;
 var
    // exemplos de vetores
    alunos: array[MIN..MAX] of string;
    notas: array[1..3] of integer;
    dados: array[0..MAX-1] of real;

    // exemplos de matrizes
    tabela: array[MIN..MAX, 1..10] of real;
    tabuleiro: array[1..3,1..MAX+1] of integer;
    dados: array[0..MAX-1,MIN..MAX] of real;

    // exemplos de array de dimensao 3
    cubo: array[MIN..MAX, 1..10, 0..50] of real;
 end.
```

Para se acessar o conteúdo de um vetor deve-se usar o nome da variável e um

indexador do tipo ordinal entre colchetes.

Exemplos:

```
alunos[2]:= 'Fulano de Tal';
notas[1]:= 95;
read(dados[9]);

tabela[1,5]:= 0.405;
write(tabuleiro[2,1]);
read(dados[0,0]);

cubo[1,2,3]:= PI-1;
```

Tentar acessar um índice fora da faixa definida no tipo enumerável gera erro de execução no programa.

O compilador não aceita que se passe o tipo `array` como parâmetro em funções e procedimentos, por isto normalmente o programador deve declarar um novo tipo para o `array`.

Exemplo:

```
type
   vetor_de_reais = array [1..10] of real;
var
   v: vetor_de_reais;
```

Vale ressaltar que o tipo `string` é uma forma especial de `array` de caracteres (`char`). Suponha uma palavra com 10 letras. Significa que esta `string` contém 10 caracteres. Para acessar cada caractere separadamente, utilize:
`nome_variavel[indice_do_caractere_desejado]`.

Exemplo:

```
var
   s: string;
begin
   s := 'hello world!';
   writeln (s[4]);      // resultado eh a segunda letra L de hello.
end.
```

5.7 Tipo `record` (registro)

O tipo `record` é usado para aglomerar sob um mesmo nome de variável uma coleção de outras variáveis de tipos potencialmente diferentes. Por isto é uma estrutura heterogênea, contrariamente ao tipo `array` que é homogêneo.

A sintaxe para se usar uma variável do tipo `record` é a seguinte:

```
record
   <Id_1> : <tipo_1>;
   <Id_2> : <tipo_2>;
    ..................
   <Id_n> : <tipo_n>;
end;
```

onde `<Id_i>` é um identificador e `<tipo_i>` é um tipo qualquer da linguagem ou previamente definido pelo programador.

Exemplo:

```
program tipo_record_1;
 var
    cliente = record
       nome: string;
       idade: integer;
       cpf: longint;
       sexo: char;
       endereco: string;
       salario: real;
    end;
```

Normalmente é boa política declarar um tipo para o `record`, como é mostrado a seguir:

Exemplo:

```
program tipo_record_1;
 type
    tipo_cliente = record
       nome: string;
       idade: integer;
       cpf: longint;
       sexo: char;
       endereco: string;
       salario: real;
    end;
 var
    cliente: tipo_cliente;
```

5.8 Tipo `file` (arquivo)

O tipo `file` é usado basicamente para armazenamento de dados em memória secundária. Contudo, é possível também se utilizar deste tipo para gravação na memória principal.

Desta forma, o tamanho que os arquivos podem ocupar dependem basicamente do espaço livre em disco ou em RAM.

Um arquivo é declarado da seguinte maneira:

```
var F: file of <tipo>;
```

observando-se que `<tipo>` pode ser uma estrutura de qualquer tipo, exceto o próprio tipo `file`.

É possível não se informar o tipo do arquivo, o que resulta em um arquivo sem tipo. Segundo o guia de referência, isto é equivalente a um arquivo de bytes.

Para se usar uma variável do tipo arquivo, alguns comandos são necessários:

assign
: associa o nome da variável (interno ao programa) a um nome de arquivo (em disco, por exemplo). O nome do arquivo é dependente do sistema operacional;

reset
: abre o arquivo para somente leitura;

rewrite
: abre o arquivo para leitura e escrita;

close
: fecha o arquivo;

Segue um exemplo em que um arquivo de registros (`record`) é usado para armazenar uma agenda em disco.

```
type
     agenda = record
          nome: string;
          rg, fone: longint;
     end;
var
     F: file of agenda;

begin
     assign (F,'~/dados/agenda.db');
     reset (F);
     // comandos que manipulam a agenda
     close (F);
end.
```

Um arquivo especial é o do tipo `text`, que armazena dados que podem ser manipulados pelas bibliotecas padrão de entrada e saída, inclusive as entradas default `input` (teclado) e `output` (tela).

O código a seguir mostra o uso de um arquivo do tipo `text`.

```
var
     F: text;
     i: integer;
begin
     assign (F,'~/dados/entrada.txt');
```

```
      reset (F);
      read (F,i);
      // comandos que manipulam a variavel i
      write (F,i); // imprime i no arquivo texto (ASCII);
      write (i);   // imprime i na tela;
      close (F);
end.
```