Os comandos são inseridos na seção de comandos e podem ser, basicamente, classificados em sete categorias:

1. Comandos de atribuição
2. Comandos compostos
3. Comandos de entrada e saída
4. Comandos condicionais
5. Comandos de repetição
6. Comandos para tratamento de arquivos
7. Comandos auxiliares

O ponto e vírgula é usado na linguagem *Pascal* como separador de comandos, servindo para separar um comando dos comandos sub-seqüentes.

Um comando de atribuição é definido através da seguinte sintaxe:

```
variável := expressão ;
```

O tipo da expressão deve ser igual ao tipo da variável, com exceção de dois casos especiais onde:

- A variável é do tipo **real** e a expressão é do tipo **integer**
- A variável é do tipo **string** e a expressão é do tipo **char**

Exemplo. Sendo dados :

```
Var
 item: integer;
 saída: boolean;
 soma, Valor: real;
 caractere: char;
 cadeia: string
```

São válidos os seguintes comandos de atribuição:

```
item:= 0 ;
saida:= FALSE ;
soma:= valor1 + valor2 ;
caracter:= 'a' ;
cadeia:= 'Isso é uma cadeia de caracteres' ;
soma:= 9 ;
cadeia:= 'a' ;
```

Além de marcar o início e o fim da seção de comandos, o par **begin** e **end** define um par de instruções usado para combinar um conjunto de comandos em um *comando composto,* também chamado de *bloco de comandos.*.

Exemplo

```
if first < last then
  begin
   Temp := First;
   First := Last;
   Last := Temp;
  end;
```

Os comandos usados para entrada e saída de dados são definidos, pelo compilador, por meio de quatro comandos:

1. Read
2. Readln
3. Write
4. Writeln

Os comandos **read** e **readln** são usados para ler o valor de uma variável de um dispositivo de entrada de dados. A diferença entre os dois comandos é que o comando **readln** processa uma quebra de linha após a leitura do valor de uma variável, enquanto o **read** não o faz.

.

A leitura de dados pode ser direcionada para um arquivo, identificado por uma variável do tipo TEXT.

Sintaxe:

```
read( listaVariáveis );
```

Onde listaVariáveis é uma sequëncia de uma ou mais variáveis separadas por vírgula.

A sintaxe de um comando **read** para leitura a partir de um arquivo é:

```
read( variavelArquivo, listaVariáveis );
```

Onde variavelArquivo é uma variável do tipo TEXT.

Exemplo

```
Program PascalZIM ;
 var
  arq: Text ;
   caractere: char ;
 begin
   assign( arq, 'Teste.Pas' );
   reset( arq );
   while not eof( arq ) do
    begin
     read( arq, caractere );
     write( caractere );
   end;
 end.
```

Os comandos **write** e **writeln** são usados para imprimir o valor de uma sequência de expressões em um dispositivo de saída de dados. A diferença entre os dois comandos é que o comando **writeln** processa uma quebra de linha após imprimir o valor de uma sequência de expressões.

A escrita de dados pode ser direcionada para um arquivo, identificado através de uma variável do tipo TEXT.

A sintaxe de um comando **write** / **writeln** para impressão na tela de uma sequência de expressões é:

```
  write( expressão1 , expressão2 , .... , expressãon ) ;
```

A sintaxe de um comando **write** / **writeln** para impressão em arquivo de uma sequência de expressões é:

```
  write( variavelArquivo, expressão1 , expressão2 , .... , expressãon
) ;
```

Onde variavelArquivo é uma variável do tipo TEXT.

Exemplo

```
   Program PascalZIM ;
    var
      c: char ;
    begin
      writeln( 'Please press a key' );
      c := Readkey;
      writeln( ' Você pressionou ', c, ', cujo valor ASCII é ',
ord(c), '.' ) ;
    end.
```

Os parâmetros do comando **write** podem conter a especificação do seu comprimento. Tal especificação é definida através da seguinte regra sintática:

```
   expressão : tamanho
```

Onde expressão define um parâmetro *e tamanho* é uma expressão do tipo inteiro.

A impressão de constantes em ponto flutuante pode conter, além da especificação de comprimento, a especificação do número de casas decimais a serem impressas. Essa especificação é dada através da seguinte regra sintática:

```
   expressão : tamanho : casas decimais
```

Onde expressão é um parâmetro do tipo real, tamanho e casas decimais são expressões do tipo inteiro.

A impressão de uma linha em branco é dada através de um comando **writeln** como abaixo:

```
   writeln ;
```

Exemplo

```
    Program PascalZIM ;
     var
        arq: text;
      begin
        assign( arq,'teste.txt' ) ;
        rewrite( arq ) ;
        writeln;                                { Impressão na tela }
        writeln( 1:10, 2:20, 3:30 ) ;
        writeln( 'a':10, 'b':20, 'c':30 ) ;
        writeln( 'asd':10, 'bnm':20, 'cvb':30 ) ;
        writeln( 2.1:10, 3.2:20, 4.3:30 ) ;
        writeln( 2.1:10:2, 3.2:20:3, 4.3:30:4 ) ;
        writeln;                                 { Impressão em arquivo
}
        writeln( arq, 1:10, 2:20, 3:30 ) ;
        writeln( arq, 'a':10, 'b':20, 'c':30 ) ;
        writeln( arq, 'asd':10, 'bnm':20, 'cvb':30 ) ;
        writeln( arq, 2.1:10, 3.2:20, 4.3:30 ) ;
        writeln( arq, 2.1:10:2, 3.2:20:3, 4.3:30:4 ) ;
        close( arq ) ;
      End.
```

Permitem restringir a execução de comandos.

O compilador reconhece os seguintes comandos condicionais:

- if..then
- if...then...else
- case
- case...else

if

Possibilita restringir a execução de um conjunto de comandos.

A sintaxe de um comando **if...then** é:

```
if expressão then comando
```

Onde expressão é uma expressão condicional e comando é um comando simples ou um bloco de comandos.

O comando funciona da seguinte forma: se expressão for TRUE, então comando é executado; caso contrário *comando* não é executado.

Exemplo

```
if j <> 0 then result := I/J;
```

A sintaxe de um comando **if...then...else** é:

```
if expressão then comando1 else comando2
```

Onde expressão é uma expressão condicional, comando1 *e* comando2 um comando simples ou um bloco de comandos.

O comando funciona da seguinte forma: se expressão for TRUE, então comando1 é executado; caso contrário, comando2 é executado.

Exemplo

```
if j = 0 then
 write( j )
else
   write( m );
```

Em uma série de comandos **if** aninhados a cláusula **else** está ligada ao **if** mais próximo no aninhamento.
Uma seqüência de comandos como:

```
if expressão1 then if expressão2 then comando1 else comando2;
```

É reconhecido pelo compilador da seguinte forma:

```
if expressão1 then [ if expressão2 then comando1 else comando2 ];
```

case

Possibilita a escolha de um conjunto de comandos que serão executados, dentre várias alternativas de escolha.

Sintaxe

```
case selector of
  lista de constantes : comandos ;
  lista de constantes : comandos ;
  ...
  lista de constantes : comandos ;
  else comandos ;
end ;
```

Onde:

- seletor é uma expressão do tipo integer ou char ;

- lista de constantes é uma sequência de constantes do tipo integer ou char, separadas por vírgula (ao invés de uma constante é possível usar um intervalo de constantes, que consiste em duas constantes separadas por um par de pontos)

A cláusula else não é obrigatória, e os comandos associados a essa cláusula serão executados somente se nenhuma outra opção do case foi selecionada ;

Exemplo

```
  Program PascalZIM ;
    Var
      opcao : integer ;

    Begin
      write ( 'Entre com uma opcao: ' );
      readln ( opcao );

      // escolha da opcao
     case opcao of
      1 : writeln( 'Você escolheu a opção 1...' );
      2 : writeln( 'Você escolheu a opção 2...' );
      3 : writeln( 'Você escolheu a opção 3...' );
      else writeln( 'Você escolheu uma opção diferente de 1, 2, 3...'
);
     end ;
   End.
```

Exemplo

```
  Program PascalZIM ;
    const
       opSoma = '+' ;
       opSubtracao = '-' ;
       opProduto = '*' ;
       opDivisao = '/' ;

    Var
      opcao : char ;

    Begin
      write ( 'Entre com um operador: ' );
      readln ( opcao );

      // escolha da opcao
     case opcao of
      opSoma : writeln( 'Você escolheu soma... ' );
      opSubtracao : writeln( 'Você escolheu subtracao...' );
      opProduto : writeln( 'Você escolheu produto...' );
      opDivisao: writeln( 'Você escolheu divisao...' );
     end ;
   End.
```

Exemplo

```
  Program PascalZIM ;
    Var
      opcao : integer ;

    Begin
      write ( 'Entre com uma opcao: ' );
      readln ( opcao );
```

```
      // escolha da opcao
     case opcao of
      1, 2 : writeln( 'Você escolheu a opção 1 ou 2...' );
      3 : writeln( 'Você escolheu a opção 3...' );
      else writeln( 'Você escolheu uma opção diferente de 1, 2, 3...'
);
     end ;
   End.
```

Exemplo

```
  Program PascalZIM ;
    Var
      c: char;

    Begin
      write( 'Digite um caracte: ' );
      readln( c );

      case c of
        'A'..'Z', 'a'..'z': writeln( '=> Você digitou uma letra!' );
        '0'..'9':           writeln( '=> Você digitou um digito!' );
        '+', '-', '*', '/': writeln( '=> Você digitou um operador!' );
      else
        writeln( '=> Você digitou um caractere!' );
      end;
   End.
```

Comandos de Repeticao

Os comandos de repetição permitem que seja repetida a execução de um conjunto de comandos.

Os comandos de repetição definidos pelo compilador são os seguintes:

- Repeat
- While
- For

Os comandos de desvio que podem ser utilizados nestes comandos são:

- Break
- Continue

for

O comando **for**, diferente dos comandos **repeat** e **while**, permite que uma sequência de comandos seja executada um número definido de vezes.

Sintaxe

```
   for contador := valorInicial to valorFinal do
```

```
        comando

for contador := valorInicial downto valorFinal do
    comando
```

Onde:

- contador é uma variável do tipo **integer** (ou **char**)
- valor Inicial e valor Final são expressões do tipo **integer** (ou do tipo **char**)
- comando pode ser um comando simples ou um comando composto

Funcionamento

1. O comando **for** armazena na variável contador o valor da expressão correspondente à valorInicial.

2. Se contador é maior (**for**...**to**) ou menor (**for**...**downto**) que valorFinal o comando **for** pára de executar. Caso contrário, comando é executado**.**

3. Após executar comando o valor armazenado em *contador* é incrementando ou decrementando (o **for**...**to** incrementa, e **for** ... **downto** decrementa).

4. Volta para o passo 2.

Exemplo

```
For i:= 2 to 63 do
  if data[ i ] > max then
    max := data[ i ] ;
```

Exemplo

```
For c:= 'a' to 'z' do
   write( c );
```

repeat

O comando **repeat** executa repetidamente uma sequência de comandos até que uma dada condição, resultantes da avaliação de uma expressão booleana, seja *verdadeira*.

Sintaxe

```
repeat
  comando1 ;
  ...
  comandon ;
until expressão ;
```

Onde expressão é uma expressão condicional.

Os comandos internos ao **repeat** são executados no mínimo uma vez, pois a condição de parada da repetição é avaliada somente após a primeira repetição.

Exemplo

```
repeat
  k := i mod j ;
  i := j ;
  j := k ;
until j = 0 ;
```

while

O comando **while** é semelhante ao comando **repeat**, com a diferença de que a condição para a execução repetida de comandos é avaliada antes da execução de qualquer comando interno da repetição.

Dessa forma, se a condição inicial para o **while** for *falsa*, a sequência de comandos definidos para o **while** não será executada nenhuma vez.

Sintaxe

```
while expressão do
 comando
```

Onde expressão é uma expressão lógica e comando pode ser um comando composto.

Exemplo.

```
while dado[ i ] <> x do i := i + 1;
```

break

Usado para forçar a saída de uma estrutura de repetição (while, for, repeat).

Sintaxe

```
break ;
```

Onde:

- O comando deve estar dentro do corpo de uma estrutura de repetição.
- O próximo comando a ser executado após o break é o comando que segue a estrutura de repetição.

Exemplo

```
Program PascalZIM;
  var
    contador: integer;
  begin
    contador := 1;

    { Repetição que é executada 5 vezes }

    while ( true ) do
      begin
        writeln( 'Contador vale:' , contador );
        contador := contador + 1;
        if( contador > 5 ) then
           break
        else
           continue;
      end ;

    { Impressão de uma mensagem após sair da repetição }

    writeln( 'Agora estou fora do while!' );
  end.
```

continue

Usado para desviar a execução dos comandos de uma estrutura de repetição(while, for, repeat) para a avalição da condição de loop.

Sintaxe

```
break ;
```

Onde:

- Esse comando deve estar dentro do corpo de uma estrutura de repetição.
- Após a execução do comando a repetição pode parar (se a condição de loop assim indicar) ou prosseguir com a execução do primeiro comando da repetição.

Exemplo

```
Program PascalZIM;
  var
    contador: integer;
  begin
    contador := 1;

    { Repetição que é executada 5 vezes }

    while ( true ) do
      begin
        writeln( 'Contador vale:' , contador );
        contador := contador + 1;
        if( contador > 5 ) then
           break
        else
           continue;
      end ;

    { Impressão de uma mensagem após sair da repetição }

    writeln( 'Agora estou fora do while!' );
  end.
```