**UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC**

**CURSO DE CÎENCIA DA COMPUTAÇÃO**

**DAIARA PAES DA ROSA**

**TIAGO ALEFF DA SILVA**

**LINGUAGEM DE PROGRAMAÇÃO ZEUS**

**CRICIUMA**

**2016**

# SUMÁRIO

[SUMÁRIO](#h.gjdgxs)

[1. INTRODUÇÃO](#h.30j0zll)

[2. CRIANDO UM PROGRAMA](#h.3znysh7)

[3. COMENTÁRIOS](#h.2et92p0)

[4. TIPOS DE VARIÁVEIS](#h.tyjcwt)

[5. LABEL](#h.3dy6vkm)

[6. DECLARAÇÃO DE VARIÁVEIS](#h.1t3h5sf)

[6.1 VARIÁVEIS DO TIPO INTEIRO](#h.4d34og8)

[6.2 VARIÁVEIS DO TIPO ARRAY](#h.2s8eyo1)

[6.3 DECLARAÇÃO DE CONSTANTES](#h.17dp8vu)

[6.4 EXEMPLO DE PROGRAMA COM VARIÁVEIS](#h.3rdcrjn)

[6.5 O QUE NÃO É PERMITIDO PELA LINGUAGEM](#h.26in1rg)

[7. LER E IMPRIMIR DADOS DE VARIÁVEIS](#h.lnxbz9)

[8. PROCEDURES](#h.35nkun2)

[9. ESTRUTURAS DE CONTROLE](#h.1ksv4uv)

[9.1 IF E ELSE](#h.44sinio)

[9.2 IF ELSE ENCADEADOS](#h.2jxsxqh)

[9.3 CASE](#h.z337ya)

[10. ESTRUTURAS DE REPETIÇÃO](#h.3j2qqm3)

[10.1 ESTRUTURA FOR](#h.1y810tw)

[10.2 ESTRUTURA DE REPETIÇÃO WHILE](#h.4i7ojhp)

[10.3 ESTRUTURA DE REPETIÇÃO REPEAT UNTIL](#h.2xcytpi)

[11. OPERAÇÕES ARITMÉTICAS](#h.1ci93xb)

[12.OPERADORES LÓGICOS](#h.3whwml4)

# INTRODUÇÃO

Alguns conceitos são importantes antes de começarmos. Como a definição de parâmetros, que são variáveis declaradas com tipo definido na declaração de procedures, que veremos mais adiante.

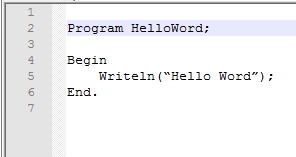
O escopo é o corpo de alguma estrutura indicando o início e o fim. O escopo de um programa é onde ele começa e onde termina, o escopo de uma estrutura de repetição também possui um início e um final como todas as outras estruturas. Entender o que é escopo é importante para saber quais instruções serão executadas ou não.

A linguagem Zeus é **case sensitive** isso quer dizer que as letras maiúsculas e letras minúsculas são diferentes umas das outras ( F é diferente de f ), é importante fazer esta observação por que existem linguagem de programação em que elas não são **case sensitive.**

# CRIANDO UM PROGRAMA

No nosso primeiro programa que vamos criar na linguagem Zeus vamos criar um programa que imprimirá na tela uma mensagem dizendo Hello Word.

Um programa desenvolvido na linguagem de programação de Zeus possui algumas funções como a definição de label’s, constantes, variáveis e procedures antes da manipulação delas dentro de um escopo chamado **begin end**. Ao criar um simples programa devemos definir primeiramente as label’s, constantes, variáveis e procedures respectivamente nessa ordem se elas existirem. Não podemos definir constantes antes de labels e nem procedures antes de variáveis por que a linguagem não suporta esse formato de declaração.Agora vamos ver o nosso primeiro programa:

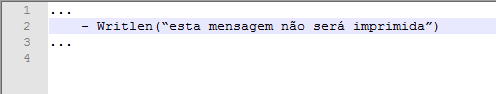


A palavra reservada **program** indica o início do programa chamado HelloWord, depois da palavra **program** é definido um nome para o programa seguido por ponto e vírgula. Depois do ponto e vírgula vem o escopo do programa que irá realizar as tarefas definidas pelo usuário da linguagem. As instruções **begin end** agrupam as instruções dentro do seu corpo, ao final de toda instrução dentro de **begin end** é colocado um ponto e vírgula. A instrução **writeln** imprime na tela a mensagem que está dentro das aspas duplas. No final do programa temos um ponto e esse ponto indica que o programa HelloWord chegou ao fim.

Bem todos os conceitos comentados até aqui serão abordados mais à frente. Bons estudos!

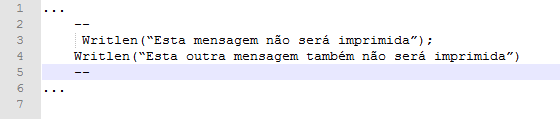
# COMENTÁRIOS

Na linguagem existem dois tipos de comentários. O comentário em linha e o comentário em bloco. O comentário em linha é utilizado para comentar apenas uma linha de códigos escritos pelo desenvolvedor definido pelo caractere traço “**-**“. Esse caractere ignora todas as instruções que estão na mesma linha e não apenas uma instrução. Veja o exemplo:



Daqui para frente os três pontinhos antes das instruções e no final dalas serão utilizanda para abstrair o resto do programa que não é importante para o exemplo.

O comentário em bloco é semelhante ao comentário simples, porém quando iniciado o caractere de comentário serão ignorados todas as instruções que estão entre os o caractere de início de comentário e o caractere de final de comentário. O comentário em bloco é definido pelos símbolos de início “**--**“ e o caractere de final de comentário “**--**“. A seguir o exemplo de comentário em bloco:



# TIPOS DE VARIÁVEIS

Na linguagem temos dois tipos de variáveis, as variáveis de tipo inteiro e os arrays de inteiros. A variáveis de tipo de inteiro podem receber valores negativos e positivos. O tamanho de casas decimais que um inteiro pode guardar é de – 2000000 e +2000000.

Nos arrays cada índice pode conter somente números positivos que vão até no máximo 2000000 e podem guardar variáveis inteiras com o valor máximo de casas decimais de +2000000. Um array é uma variável que guarda vários valores do mesmo tipo dentro da mesma variável. Para acessar um valor da variável devemos informar o índice que o valor que queremos está guardado.

Constantes são variáveis onde o valor dela não muda ao decorrer do programa. O valor dela é definido no momento da declaração e não pode ser alterado depois de definido, elas são do tipo inteiro.

# LABEL

As label’s são semelhantes a rótulosque são criadas através da palavra reservada **label** e um nome para a identificação. A função da **label** é poder criar um rótulo que seja possível chama-la e então o fluxo do programa será desviado para ela e o programa será executado para o começo da implementação da label.

...

Label recalcula;

...

Assim usamos a palavra reservada **label** para declarar um novo rótulo. Para declaramos várias **label**’s usamos a seguinte sintaxe:

...

Label recalcula, teste, erroCalculo;

...

Separando os identificadores das label’s com vírgulas é possível criar várias label’s e não podemos esquecer de usar o ponto e vírgula no final da instrução. Não é possível fazer declarações da seguinte forma:

...

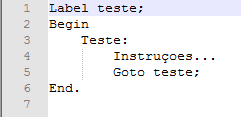
Label recalcula;

label teste;

label erroCalculo;

...

Declarar dessa forma é considerado um erro de sintaxe. Agora veremos um programa simples fazendo o uso de uma label somente para demostrar o que foi dito aqui até sobre ela.



Para chamar o rótulo **teste** basta usar a instrução **goto** que desviará o fluxo do programa para o início da implementação da **label** repetindo o corpo novamente. O **goto** também é chamado de desvio incondicional porque toda vez que ele é executado o fluxo do programa será desviado para a implementação da label.

# DECLARAÇÃO DE VARIÁVEIS

## VARIÁVEIS DO TIPO INTEIRO

Podemos declarar variáveis logo após o início do programa. Declaramos uma variável do tipo inteiro utilizando a palavra reservado **var** seguida pela identificação da variável. No seguinte exemplo, vemos uma declaração simples de variáveis do tipo inteiro.

...

Var totalNoEstoque : int ;

...

A palavra reservada utilizada após os dois pontos, int é usada para informar que aquela variável pertence ao tipo inteiro, então ela irá poder guardar somente valores permitidos pelo tipo dela. A seguir, mais exemplos de declarações simples:

...

Var minimoEstoque : int;

quantidadeDeFuncionarios : int;

valorMaximoVendido : int;

...

Essa é uma forma simples de se declarar variáveis. No final de cada declaração de variável deve seguir o ponto virgula para finalizar uma declaração. Também podemos declarar múltiplas variáveis na mesma declaração. Veja a seguir como fica o exemplo acima:

Var minimoEstoque, quantidadeDeFuncionarios, valorMaximoVendido : int;

Dessa forma abstraímos a palavra reservada var e o tipo e tipamos as variáveis apenas no final da declaração e o ponto e vírgula vai no final da declaração.

## VARIÁVEIS DO TIPO ARRAY

A declaração do array acontece de uma forma muito simples. Veja a seguir como quando declaramos variáveis do tipo array:

...

Var totalNoEstoque : array[ 0 .. 2 ] of int;

…

Nessa declaração o comando **var** permanece, junto com o nome da variável e o ponto e virgulo no final da instrução. Depois dos dois pontos usamos a palavra reservada **array** para informar que vamos criar um array e **of int** para informar que essa variável é do tipo inteiro. Entre os colchetes possuem dois números, o primeiro mais à esquerda representa em que valor numérico começa os índices e o segundo mais à direita separado pelos caracteres **“..”** informam até que valor numérico vai os índices. Ou seja, nesse exemplo existem três índices contando com o índice zero.

A seguir, um exemplo de várias declarações de **arrays**, mas note que todos eles possuirão a mesma quantidade de índices informadas.

Var minimoEstoque, quantidadeDeFuncionarios , valorMaximoVendido : array[ 0 .. 2 ] of int;

Entre os colchetes é informado quais são os índices disponíveis para uso, nesse caso são eles: o índice zero, um e dois. Não é possível acessar um índice do array que não foi declarado, sendo que ocorrerá um erro se tentar acessar um índice que não foi declarado.

## DECLARAÇÃO DE CONSTANTES

As constantes são declaradas usando a palavra reservada **const** seguida pela identificação da constante. Veja um exemplo de uma declaração simples de constantes:

...

Const temperaturaMaxima = 70;

...

Toda declaração de constantes deve usar a palavra reservada **const**, se ela for omitida na declaração da constante, um erro irá ocorrer parando o programa. Um valor para a variável deve ser atribuído com o sinal de igual, caso não seja informado um valor para a mesmo, um erro também irá ocorrer. No final de declarações de constantes deve-se usar o ponto e vírgula para informar que a instrução acabou.

Também é possível fazer declarações de várias constantes uma seguida da outra. Veja o exemplo a seguir:

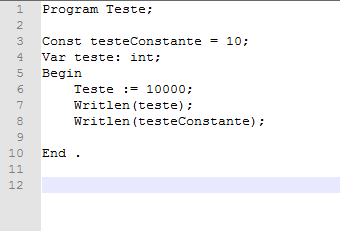
Const temperaturaMaxima = 70;

temperaturaMinima = 10;

Veja que para declarar mais de uma constante não deve se usar a palavra reservada **const**. No final de cada instrução de declaração de constantes deve haver o sinal de ponto e vírgula para indicar que é o final da declaração.

## EXEMPLO DE PROGRAMA COM VARIÁVEIS

Este exemplo a seguir é um programa que usa variáveis:

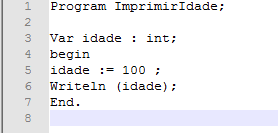


## O QUE NÃO É PERMITIDO PELA LINGUAGEM

Em relação a declarações não é permitido a linguagem declarar as propriedades fora de ordem. No início de cada programa há uma ordem para declarar propriedades da linguagem, e essa ordem é essa: primeiro, devemos declarar label, se houver; segundo, declaramos as variáveis locais, se houver; terceiro, declaramos as procedures, se houver.

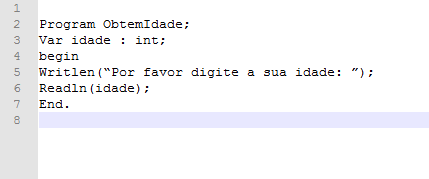
# LER E IMPRIMIR DADOS DE VARIÁVEIS

Para mostrar ao usuário uma mensagem de texto na tela podemos usar a instrução **writeln**. Essa instrução recebe como parâmetro um conjunto de caracteres ou variáveis e então exibira na tela o conjunto de caracteres ou o valor da variável.

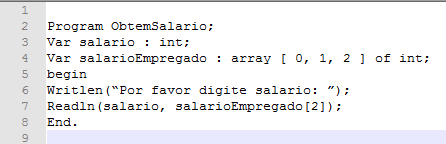


A variável idade é declarada antes do **begin end**, dentro do escopo é atribuído um valor para a variável idade e logo em seguida a instrução **writeln** imprimir o resultado da variável. Para exibir um texto no monitor devemos escrever este texto dentro da instrução **writlen** entre aspas, assim esse texto será exibido para o usuário.

Também é possível obter os caracteres digitados pelo usuário através do comando **readln** que recebe uma variável como parâmetro e atribui o valor digitado pelo usuário a ela. Por exemplo:



Por meio da instrução **readln** é possível obter a entrada digitada pelo usuário e atribuir esse valor a uma variável. Veja o seguinte exemplo:



Aqui temos duas variáveis dentro de **readln** separadas por vírgulas, esse método é usado para atribuir a mesma entrada digitada pelo usuário a mais de uma variável. A variável **salário** é do tipo inteiro enquanto que **salarioEmpregado** é um array e a entrada do usuário está sendo atribuída no índice dois.

# PROCEDURES

Cada procedure é como um método semelhantes a linguagens como Java, C++ e PHP. Cada procedure é declarada com a palavra reservada chamada **procedure** e com um nome de identificação para a mesma. No programa pode haver várias procedures cada uma separada pelos caracteres de ponto e vírgula.

...

Program ExemploProcedure;

Procedure nomeDaProcedure :

Writlen(“Ola mundo!”);

;

...

Os dois pontos depois do nome da procedure marcam o início do corpo e o ponto e vírgula indica o fim da procedure. Várias procedures podem ser declaradas uma depois a outra, sempre usando o ponto e vírgula para marcar o final de cada procedure. Exemplo de várias declarações de procedures:

...

Procedure nomeDaProcedure :

[código]

;

Procedure nomeDaProcedure2 :

[código]

;

...

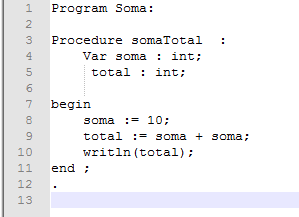
Essas estruturas também podem receber uma lista de argumentos entre parênteses depois do nome de identificação da procedure. Os argumentos são todos do tipo inteiro. Exemplo de uma lista de argumentos:

Procedure nomeDaProcedure ( variável1, variavel2, variavel3 : int ):

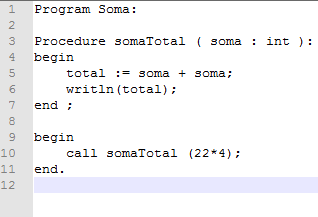
[código]

;

Dentro de cada procedure é possível criar bloco com declarações de label, variáveis, constantes todas as declarações seguem a mesma regra já abordadas aqui. Veja um exemplo a seguir de uma procedure com uma estrutura semelhante:



Para chamar uma procedure é necessário estar dentro do corpo **begin end** utilizando o comando **call** seguindo pelo nome da procedure e com os parâmetros se houver, ela também aceita receber expressões aritméticas que retornam resultados como no exemplo a seguir:



# ESTRUTURAS DE CONTROLE

## IF E ELSE

As estruturas de condicionais são utilizadas para tomar decisões sobre um determinado bloco de código, decidindo se esse código será executado ou não. A estrutura de controle que será apresentada é mais conhecida como **if else** ou apenas **if**, essa estrutura inicia-se com a palavra reservada **if** seguida por uma condição, se esta condição for verdadeira então o código dentro do corpo do **if** será executada, caso a condição for falsa o código dentro do corpo será ignorado e não será executado. Veja um exemplo de uma estrutura **if**:

...

If temperaturaMaxima > 100

Then

Begin

Instrução 1;

Instrução 2;

End

...

Vamos estudar o código acima, depois do início do **if** há uma condição onde testa o valor de **temperaturaMaxima** se é maior que o número 100, se isso for verdade tudo que estiver dentro do **if** será executado. O final da estrutura **if** é marcada pela palavra reservada **end**, essa palavra marca o fim da estrutura e a partir desse ponto o que tiver em seguida não faz mais parte do corpo. Agora vamos ver o seguinte código:

...

If temperaturaMaxima > 100

Then

Instrução 1;

...

Nesse código não está faltando nenhuma instrução, ele está correto. Há o início da instrução usando o **if**, seguido por uma condição e a instrução **then**, logo em seguida há uma única instrução. Então, podemos concluir que sempre que há apenas uma única instrução dentro do **if** não precisamos abrir o **begin end**, assim o fim do corpo termina com a própria instrução.

Veja agora esse outro exemplo:

...

If temperaturaMaxima > 100

Then

Instrução 1;

Else

Instrução 1;

...

Nesse exemplo há um novo elemento chamado **else**, ele é uma palavra reservada que é utilizado para tomar decisões. Quando o **if** retornar falsa a condição, o que estiver dentro do **else** será executado. Dentro do **else** também segue as mesmas regras, quando há mais de uma instrução para ser executada então, deve se usar os comandos **begin end** para agrupar estas instruções e para que o programa saiba onde é o início e fim das instruções.

## IF ELSE ENCADEADOS

É possível usar **if else** encadeados para tomar decisões sobre o que fazer.

...

If temperaturaMaxima > 100 then

Instrução 1

Else If temperaturaMaxima > 1000 than

Instrução 1

Else If temperaturaMaxima > 10000 Than

Begin

If temperaturaMaxima > 100000 Than

Instrucao;

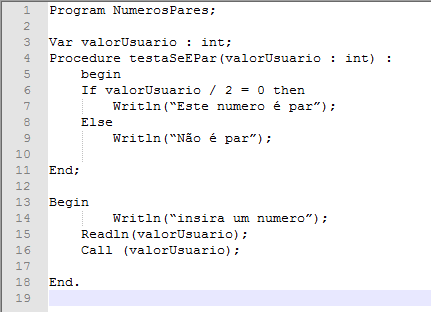
Else

Instrução;

End;

...

A seguir é um exemplo de um programa usando o **if else**:



## CASE

A estrutura de condição **case** é muito semelhante ao o **if else**, porque ela também compara valor de variáveis com outros valores pré-estabelecidos. Veja o exemplo a seguir:

...

Case total of

1000: begin

Instruçoes...

End;

End

...

Essa estrutura começa com a palavra reservada **case,** ela indica o início da estrutura enquanto o **end** indica o fim. A variável **total** deve possuir um valor inteiro que será comparado com o valor pré-estabelecido, se o valor de total corresponder à o valor (1000) então as instruções que estão depois dos dois pontos serão executadas, caso não seja igual a nenhum desses valores as instruções não serão executadas.

Agora veja este outro exemplo:

...

Case total of

1000, 100, 10: begin

Intruções

End;

End

…

Note que agora temos outros três valores que podemos testar para executar as nossas instruções. Se o valor de **total** for igual a 1000, ou 100, ou 10 então serão executadas as instruções depois dos dois pontos.

Veja agora outro exemplo:

Case total of

1000, 100, 10: begin

Intruções

End;

5000, 500, 50: begin

Intruções

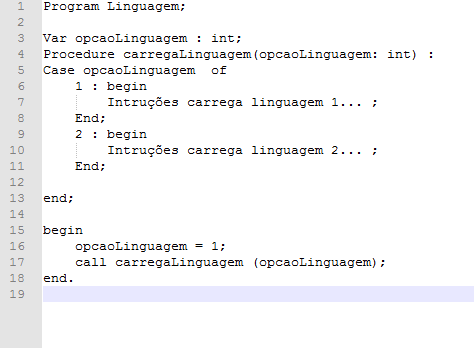
End;

2000:

Uma intrução

End;

Nesse exemplo, possuímos mais casos do que os outros exemplos. É possível fazer estas comparações determinando instruções diferentes para cada caso. As instruções **begin end** apenas são necessários se haver mais de uma instrução, se haver apenas uma instrução não será necessário utiliza-las e nem usar o ponto e vírgula.



# ESTRUTURAS DE REPETIÇÃO

As estruturas de repetição são utilizadas para executar as mesmas tarefas várias vezes sem que seja necessária escrever o mesmo código diversas vezes. Um exemplo seria imprimir todos os números pares de zero a cem isso daria um trabalho muito grande.

## ESTRUTURA FOR

Também conhecida como laço de repetição, a estrutura **for** irá repetir uma série de instruções dentro do seu escopo, até que a sua condição informada não seja mais verdadeira e quando a instrução for falsa o laço não será repetido.

Veja o seguinte exemplo:

...

for pares := 1 to pares <= 200 do

instrução;

...

O início laço começa com o uso da palavra reservada **for** e finaliza com o final da instrução que será executada se haver apenas uma instrução. Logo depois da palavra **for** devemos informar uma variável e atribuir um valor a ela. Depois de definida o valor da variável é utilizado a instrução **to** para informal qual será a condição de parada do loop. Deve-se tomar cuidado para que a condição de parada em algum momento seja falsa, caso contrário, o loop seria executado infinitas vezes, chamado assim de loop infinito.

Nesse exemplo não foi usado **begin end** por que é utilizado apenas uma instrução, caso avesse mais de uma instrução é obrigado a utilizar as instruções **bengin e end** para indicar o final do loop.

Veja outro exemplo a seguir:

...

for pares := 1 to pares <= 200 do

begin

pares := pares + 1;

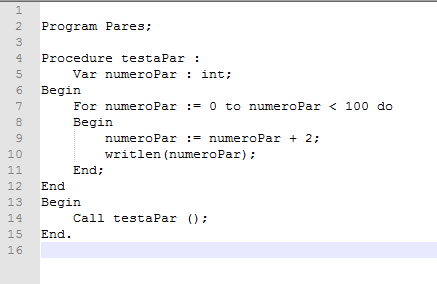
instruções;

end

...

Nesse **for** as instruções que estão dentro do seu escopo serão repetidas até que a variável pares possua um valor menor e igual a 200, dentro do escopo temos uma operação de soma que somará a variável **pares** mais um toda vez que o laço passar por essa instrução. No corpo **begin end** utilizamos o uso do ponto e vírgula cada vez que o iniciarmos ou finalizarmos uma nova instrução.

O programa a seguir foi criado utilizando o **for**:



Esse programa foi criado para imprimir todos os números pares de zero até cem e para isso é utilizado o **for** para poder fazer um incremento somando mais dois na variável chamada **numeroPar** que possui um valor definido como zero. Este é um programa simples criado para demostrar o uso dessa estrutura.

## ESTRUTURA DE REPETIÇÃO WHILE

O laço de repetição while é outra estrutura de repetição semelhante ao **for** repetindo as instruções dentro do seu escopo enquanto a condição definida for verdadeira, caso a condição seja falsa as instruções dentro do seu escopo não serão mais executadas.

Primeiro exemplo:

...

While tempo < 60 do

Begin

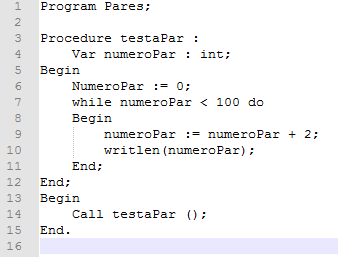
Instrução..

End

...

O laço começa com o uso da instrução **while** seguido por uma condição de parada. A palavra **do** indicará o inicio do corpo da estrutura. O uso de **begin e end** será obrigatório se haver mais de uma instrução a ser executada.

Agora veremos um programa simples criado para demonstrar o uso do while:



Este é o mesmo programa dado na estrutura de repetição **for**, porém em vez de utilizarmos o **for** estamos utilizando o **while**. Podemos notar que a variável **numeroPar** não recebe o valor zero dentro do while, mas sim antes da instrução.

## ESTRUTURA DE REPETIÇÃO REPEAT UNTIL

Essa estrutura é muito semelhante ao **while** e também muito simples. A diferença é a estrutura começa executando as instruções dentro do escopo e só no final é que a condição é testada, portanto ela executa o escopo e então testa a condição. Tanto o while quanto o for testam a condição estabelecida antes executar a condição e se ela for falsa as instruções dentro do corpo não serão executadas. Veja a seguir um exemplo:

Repeat

Begin

intruções

end until condição

Nesse exemplo a estrutura iniciou com a palavra reservada **repeat**, logo em seguida o corpo é executado e é usado a palavra **until** para indicar qual será a condição de parada do loop. O repeat é menos utilizado do que os outros laços apresentado até o momento.

# OPERAÇÕES ARITMÉTICAS

Agora vamos entender um pouco como funciona as operações aritméticas. Em Zeus podemos fazer as quatro tipos de operações matemáticas e semelhante a matemática também é possível definir a prioridade dessas operações como veremos mais adiante.

Agora vamos ver algumas operações básicas que podemos fazer:

...

Salario := 1000;

Writlen (salario);

...

Saída

1000

Já vimos está instrução anteriormente, mas ainda cabe explicação para ela, nesse caso possuímos apenas uma atribuição de valor da variável **salario** e depois esse valor é imprimido para o usuário do programa, resultando uma saída com o valor 1000.

Agora veremos um exemplo de operações matemáticas de adição que é representada pelo caractere **+**:

...

Salario := 1000 + 50;

Total := salario + 1000;

Writlen(total)

...

Saída

2050

Nesse exemplo vemos duas operações aritméticas, a primeira operação é uma soma de 1000 com 50 e depois é atribuído esse valor a variável **salario**. A variável **salario** tem o valor agora de 1050, na próxima linha é somado o valor que salario possui com mais 1000 e por último é atribuído a variável total que é somado na próxima linha. Também é possível fazer mais de uma adição por instrução, veja o exemplo:

...

Salario := 1000 + 50;

Total := salario + 1000 + 1000 + 1000;

Writlen(total)

...

Saída

2050

Este exemplo serve apenas para mostrar como é possível fazer várias operações de adição em uma instrução, as operações matemáticas são sempre realizadas da esquerda para a direita, somando os valores e pegando o resultado e somando com o valor mais à direita. Não é necessário mudar a prioridade de uma expressão onde há somente adição, porque o valor final são será alterado pela ordem das operações.

As subtrações são realizadas da mesma forma que as adições elas possuem a mesma prioridade para serem executadas em uma operação, as subtrações são representadas pelo caractere **–** (menos). Veja o seguinte exemplo:

...

Salario := 1000 – 50;

Total := salario – 50 – 50;

...

Para mudar a prioridade de uma operação devemos utilizar os sinais de parênteses. Por exemplo:

...

Total := 30 + (2 \* 15) + 5 ;

...

Nesse caso, como está sendo usado os parênteses para indicar quem deverá ser calculado primeiro, será calculado primeiro o que está dentro dos parênteses e o resultado seria somado com o 30 e então com o 5. As operações de multiplicação e divisão possuem uma prioridade maior para serem calculadas. As operações de adição são representadas pelo caractere \* (asterisco), enquanto que as operações de divisão são representadas pelo caractere de / (barra).

Outro exemplo de operações matemáticas é este:

...

Total := 30 + 2 \* 15 / 5 ;

...

O calculo da multiplicação e da divisão possuem a mesma prioridade, então quem estiver mais à esquerda será calculada primeiro e depois as outras expressões que estão mais à direita. Os cálculos de adição e subtração possuem a mesma prioridade, porém menor do que a multiplicação e divisão. O exemplo a seguir usa os parênteses para mudar a ordem de prioridade:

...

Total := (30 + 2) / 15 );

...

O primeiro calculo a ser feito será a soma, seguido pela divisão. A divisão será calculada primeiro por causa do uso dos parênteses alteram a prioridade do cálculo.

A tabela seguinte mostra os caracteres que são usados para as operações aritméticas.

|  |  |
| --- | --- |
| Adição | + |
| Subtração | **-** |
| Multiplicação | **\*** |
| Divisão | **/** |

# 12.OPERADORES LÓGICOS

Os operadores lógicos são muito utilizados em estruturas condicionais com e com estruturas de repetição. Esses operadores nos permitem definir uma condição de parada para um loop ou se em uma estrutura de condição para o bloco ser executado. Os operadores lógicos testam se a condição é verdadeira ou falsa, por esse motivo são chamados de operadores lógicos.

As expressões com o uso desses operadores devem são lidos da esquerda para a direita, somente o uso de parênteses pode alterar a ordem de leitura, semelhante a operações aritméticas como vimos anteriormente.

Os operadores lógicos são definidos pelos caracteres presentes na tabela abaixo:

|  |  |
| --- | --- |
| = | Igual |
| < | Menor que |
| > | Maior que |
| >= | Maior ou igual |
| <= | Menor ou igual |
| <> | Diferente |
| Or | Ou |
| And | E |
| Not | Não |

O operador de igualdade testa se o valor de duas variáveis que estão sendo comparadas é igual uma a outra:

...

If salario = pagamento then

...

O operador menor que compara o valor da variável da esquerda com o valro da variável da direita, caso seja menor o valor retornado será verdadeiro:

...

If salario < pagamento then

...

O operador maior ou igual compara o valor da variável da esquerda com a variável da direita. Enquanto o valor for maior e ou igual ela retornará verdadeira.

...

If salario >= pagamento then

...

O operador menor ou igual compara o valor da varivel da esquerda retornando verdadeira até que o valor da mesma seja menor.

...

If salario <= pagamento then

...

O operador de diferença é o contrario do operador de igualdade retornando verdadeiro se as duas variáveis possuem valores difentes

...

If salario <> pagamento then

...

O operador **or** é utilizado para ligar expressões diferentes. Por exemplo, queremos comparar se o salário é menor que 2000 ou maior que 1000, então utilizando o operador **or** para ligar estas expressões:

...

If (salario < 2000) or (salario > 1000) then

...

Os parênteses são utilizados para separar as expressões. Porém se desejamos comparar se **salario** seja menor que 2000 e maior que 1000 utilizamos o operador **and**:

...

If (salario < 2000) and (salario > 1000) then

...

O operador de negação **not** é utilizado para negar uma condição. Se a expressão retornar verdadeira o operador **not** irá inverter esse valor para falso, mas se a expressão retornar falsa o operador negação irá retornar verdadeira, exemplo:

If not (salario < 2000) then

...