

# Natural

## Sumário

- [1 - Introdução e Hello World](#)
- [2 - Tipos de dados](#)
- [3 - Operadores](#)
- [4 - Entradas e Saídas](#)
- [5 - Processamento Condicional](#)
- [6 - Processamento de Strings](#)
- [7 - Processamento de Loop](#)
- [8 - Modularização e Subprogramas](#)
- [9 - Características do Natural](#)
- [10 - Comandos Gerais](#)
- [11 - Comandos de Entrada/Saída](#)
- [12 - Operações Aritméticas](#)
- [13 - Manipulação de Campos](#)
- [14 – Instruções Condicionais](#)
- [15 – Controle de Processamento \(loops\)](#)
- [16 - Processamento de Quebra Automática](#)
- [17 - Modularização do Sistema](#)
- [18 - Mapas](#)
- [19 – Window\(janelas\)](#)
- [20 – Subprogramas](#)
- [21 – Sub-rotinas Externas](#)
- [22 – Copycode](#)
- [23 – Help routines](#)
- [24 – Manipulação de Dados](#)
- [25 – Manipulação de Strings/Arrays](#)

# Natural

## 1 - Introdução e Hello World

Natural é uma [linguagem de programação 4GL](#), concebida para o desenvolvimento de [aplicações de negócios em multicamadas](#). Muitas empresas e organizações ao redor do mundo usam o Natural para desenvolver aplicações de [missão crítica](#) para os seus negócios. Essas aplicações são executadas em vários setores, por exemplo, bancário, seguros, logística, varejo, manufatura e organizações governamentais. Aplicações em Natural são compatíveis em várias plataformas, isto é, uma vez desenvolvidas, o código Natural pode ser executado em todas elas, como Mainframes, Linux, Unix, Windows e containers na nuvem. O Natural é uma linguagem procedural embutida com comandos poderosos para processamento de dados e gerenciamento de transações que acessam Adabas e SQL e muitos outros sistemas de bancos de dados. A sintaxe deve vir naturalmente, como disseram os criadores do Natural.

Vejamos um rápido exemplo para um programa Hello World. O Natural One é um ambiente de desenvolvimento moderno baseado em Eclipse, que suporta os processos e ferramentas DevOps.



```
*HELLOW.NSP
* >Natural Source Header 000000
/** New Program HELLOW.
 /**
 ** :author sagdfh
 write 'Hello World'
END
```

Observe os comentários lá em cima. Cada linha comentada tem de começar com barra ou apenas com asterisco. A única coisa que precisamos para um programa Hello World em Natural é duas linhas de código. Cada programa em Natural deve terminar com um comando "END" ou um ponto (.). Esse comando "WRITE" é usado para apresentar na saída o texto que está dentro das aspas simples.

Para executar o programa, clica-se com o botão direito do mouse no arquivo à esquerda selecionando "Run as natural application".



## Glossário

**Linguagem de programação 4GL:** as linguagens de programação de quarta geração, ou 4GL em sua abreviatura de origem inglesa, são linguagens de programação de alto-nível com objetivos específicos, como o desenvolvimento de softwares comerciais de negócios. Elas permitem ao programador especificar o que deve ser feito visando um resultado imediato.

**Sistemas distribuídos:** um sistema distribuído é uma coleção de programas de computador que utilizam recursos computacionais em vários pontos centrais de computação diferentes para atingir um objetivo comum e compartilhado. Os sistemas distribuídos visam remover gargalos ou pontos centrais de falha de um sistema.

**Aplicação de negócios em multicamadas:** a arquitetura de aplicações multicamadas é baseada no modelo de sistemas distribuídos, mas sob a perspectiva da Engenharia de Software é uma arquitetura que deve seguir o padrão da divisão em camadas, em pelo menos três níveis de preocupações: apresentação, lógica de domínio e dados. Sem dúvida, desenvolver software para atender necessidades empresariais é uma tarefa árdua. Envolve escolhas e decisões de projeto que podem tornar fácil ou difícil a evolução da solução construída. Quando se fala de software empresarial (aplicação corporativa), é um consenso que o melhor padrão arquitetural para diminuir a complexidade é dividir a aplicação em “camadas”, o que chamamos de uma arquitetura multicamadas.

**Missão crítica:** já imaginou o que aconteceria se o sistema do seu banco ficasse horas sem funcionar? Quais seriam as consequências se seu provedor de acesso à internet perdesse dados de seus clientes? Já pensou na situação caótica que a cidade de São Paulo viveria se os computadores do sistema metroviário simplesmente parassem? Para muitas empresas e setores de atividade, o uso de sistemas computacionais é imprescindível para a manutenção do negócio. Se tal sistema é vítima de uma falha que interrompa seu funcionamento ou que cause a perda de dados importantes, a empresa pode simplesmente falir. Para evitar esse tipo de transtornos, tais empresas "montam" seus sistemas como sendo de missão crítica. Em poucas palavras, missão crítica é um ambiente tecnológico construído para evitar a paralisação de serviços computacionais e a perda de dados

importantes a um negócio. Para isso, uma série de equipamentos e tecnologias é aplicada ao ambiente.

## 2 - Tipos de dados

Agora vamos ver alguns dos tipos de dados mais básicos disponíveis em Natural e como atribuir valores a eles. Para usar variáveis, primeiro temos que defini-las. Faremos a definição no início do programa em um bloco de "define data". As variáveis que iremos usar são apenas para uso local nesse programa, portanto iremos definir "local".

Agora iremos adicionar algumas variáveis para um empregado. A linguagem Natural me permite agrupar dados indicados por níveis, assim iremos atribuir o nível 1 para a variável "empregado". Em seguida a variável "nome" no nível 2, que será um dado Alfanumérico (String) com tamanho fixo ou dinâmico. Nesse caso a variável "nome" terá um tamanho máximo de 30 caracteres (A30).

A variável "age" será um número inteiro de 2 bytes (I2). Por fim, teremos a variável "salary" com um número decimal, nesse caso um valor numérico com 7 dígitos antes do ponto decimal e 2 dígitos depois do ponto decimal (N7.2).



```
*HELOW2.NSP
+ * >Natural Source Header 000000
-/** New Program HELLOW2.
 /**
 /** :author sagdfh
 -define data
 - local
 -1 employee
 2 name (A30)
 2 age (I2)
 2 salary (N7.2)
 end-define

 write 'Hello World'
 END
```

Agora iremos atribuir valores as variáveis. Assim moveremos 'John Doe' para o campo "name". Outra sintaxe comum para atribuições é através do dois pontos e igual (:=).

```
move 'John Doe' to name
age := 42
salary := 100000
```

Estenderemos o uso do comando "write" para exibir os dados de modo a emitir uma string de tamanho fixo e uma de tamanho variável.

```
*HELLOW2.NSP
+* >Natural Source Header 000000
-/** New Program HELLOW2.
 /**
 /** :author sagdfh
-define data
- local
-1 employee
 2 name (A30)
 2 age (I2)
 2 salary (N7.2)
end-define
move 'John Doe' to name
age := 42
salary := 100000

write 'Hello' name
write 'Age' age 'Salary' salary
END
```

Ao executar o programa teremos:

```

HELOW2.NSP Natural I/O
Natural Web I/O Output

Page 1 18-05-02 14:21:50
Hello John Doe
Age 42 Salary 100000.00

MORE

```

Data Type	Description	Definable Length	Examples
A	Alphanumeric	1 - 1073741824 (1GB) or dynamic	(A30) or (A) dynamic
B	Binary	1 - 1073741824 (1GB) or dynamic	(B30) or (B) dynamic
D	Date	-	(D)
F	Floating Point	4 or 8	(F4) or (F8)
I	Integer	1, 2 or 4	(I1), (I2) or (I4)
L	Boolean	True or False	(B)
N	Numeric (unpacked)	1-29 digits	(N7.2) or (N29)
P	Packed Numeric	1-29 digits	(P7.2) or (P29)
T	Time	-	(T)
U	Unicode (UTF-16)	1 - 536870912 (0.5 GB) or dynamic	(U30) or (U) dynamic

OBS: Os nomes de variáveis não diferenciam maiúsculas de minúsculas.

### 3 - Operadores

Agora veremos alguns dos diferentes operadores do Natural e ainda como inicializar variáveis durante sua definição. No exemplo seguinte iremos aumentar o salário de nossos empregados a partir da semana que vem.

Definiremos uma variável chamada "newsalary" do tipo N7.2.

2 newsalary (N7.2)

Também definiremos uma variável "salarydate" do tipo D para data. Para calcularmos a data para uma semana a partir de hoje, inicializamos a variável com a data atual através do comando "init <>". Dentro do sinal de maior e menor <>, usaremos uma variável interna do sistema <\*datx> para obter a data atual.

```
2 salarydate (D) init <*datx>
```

Criaremos uma representação alphanumérica (string) da data.

```
2 salarydateA (A10)
```

Para computar o salário novo, multiplicaremos por uma constante com muitas casas decimais, ou seja, arredondaremos o resultado de forma correta. Para tal, usaremos o comando "compute rounded" e uma atribuição padrão.

```
compute rounded newsalary := salary * 1.15000086
```

Para adicionarmos uma semana a data usaremos o "add".

```
add 7 to salarydate
```

Em muitos casos é pedido que se converta a variável para outros tipos. O comando mais comum é o "move edited". Ele permite usar uma máscara de edição para o formato requerido.

```
move edited salarydate (em=YYYY-MM-DD) to salarydateA
```

O "em" significa edited mask.

```
*HELLOW3.NSP
+ * >Natural Source Header 000000
-/** New Program HELLOW2.
 /**
 ** :author sagdfh
-define data
- local
-1 employee
 2 name (A30)
 2 age (I2)
 2 salary (N7.2)
 2 newsalary (N7.2)
 2 salarydate (D) init <*datx>
 2 salarydateA (A10)
end-define
move 'John Doe' to name
age := 42
salary := 100000

compute rounded newsalary := salary * 1.15000086
add 7 to salarydate

move edited salarydate (em=YYYY-MM-DD) to salarydateA

write 'Hello' name
write 'Age' age 'Salary' salary
write 'New salary' newsalary 'as of' salarydateA
END
```

Ao executarmos o programa:

```
HELLOW3.NSP Natural I/O
Natural Web I/O Output

Page      1                               18-06-13 17:03:11

Hello John Doe
Age      42 Salary   100000.00
New salary  115000.00 as of 2018-06-20
```

## Comandos

**move edited:** converter tipos de dados

**compute rounded:** para cálculos que precisem ser arredondados

**add:** incrementar

**subtract:** decrementar

**reset:** restaurar o valor inicial da variável. 0 ou em branco conforme o tipo de dado

Statement	Description	Example
<b>:=</b>	Assignment or calculation.	Salary := Salary * 1.15
<b>MOVE</b>	Moves the value of an operand to one or more fields.	<b>MOVE</b> 'John Doe' <b>TO</b> name
<b>MOVE EDITED</b>	Converts between datatypes	<b>MOVE EDITED</b> mydate ( <b>EM=YYMMDD</b> ) <b>TO</b> mydate-alfa
<b>COMPUTE</b>	Assignment or calculation – needed for "rounding" of results	<b>COMPUTE ROUNDED</b> Salary = Salary * 1.15000086
<b>ADD</b>	Adds two or more operands.	<b>ADD 1 TO IX</b>
<b>SUBTRACT</b>	Subtracts one or more operands.	<b>SUBTRACT 1 FROM IX</b>
<b>RESET</b>	Sets the value of a field to zero (if numeric) or blank (if alphanumeric), or to its initial value.	<b>RESET</b> name <b>RESET</b> employee

## 4 - Entradas e Saídas

Agora veremos as regras de entrada e saída mais comuns no Natural. Ao invés de atribuirmos valores a variáveis, podemos obtê-los usando o comando "input". Assim excluo as atribuições e as substituo por "input name age salary".

```
*HELLOW4.NSP
+ * >Natural Source Header 000000
+/** New Program HELLOW4.
+**
+** :author sagdfh
+define data local
+1 employee
+ 2 name (A30)
+ 2 age (I2)
+ 2 salary (N7.2)
+ 2 newsalary (N7.2)
+ 2 salarydate (D) init <*datx>
+ 2 salarydateA (A10)
+end-define

input name age salary

compute rounded newsalary := salary * 1.15000086
add 7 to salarydate

move edited salarydate (em=YYYY-MM-DD) to salarydateA

write 'Hello' name
write 'Age' age 'Salary' salary
write 'New salary' newsalary 'as of' salarydateA I
END
```

Ao executar o programa:



Ao inserir os dados e pressionar Enter teremos:

The screenshot shows a software interface titled "Natural Web I/O Output". At the top, there are tabs for "HELLOW4.NSP" and "Natural I/O". The main area displays the following text:

```
Page      1                               18-06-14  13:58:31
Hello Jane Doe
Age      65 Salary      6555.00
New salary    7538.26 as of 2018-06-21
```

Mesmo sem implementar alterações, esse programa também pode ser usado em um [ambiente batch](#), no qual a entrada e a saída são baseadas em arquivos.

Para demonstrar quão fácil é criar uma saída formatada para telas ou relatórios, veremos um pequeno loop baseado em dados, que pode nos trazer alguns dados do empregado para a exibição. A única coisa que precisamos é inserir "display" e as variáveis de nível de grupo ou individual. No caso, selecionaremos o nível de grupo "employee".

The screenshot shows a software interface titled "Natural Source Header 000000". The code editor contains the following Natural program:

```
* * >Natural Source Header 000000
* ** New Program HELLOW4B.
**
/* :author sagdfh
define data local
/*( imported data fields from Data Definition Module EMPLOYEES
1 EMPLOYEE VIEW OF EMPLOYEES
2 PERSONNEL-ID (A8) /* CNNNNNNN
2 FIRST-NAME (A20) /* FIRST/CHRISTIAN NAME
2 NAME (A20) /* SURNAME/FAMILY NAME
end-define

read EMPLOYEE
display EMPLOYEE
end-read
END
```

Ao executar o programa:

EMPLOYEE		
PERSONNEL ID	FIRST-NAME	NAME
50005800	SIMONE	ADAM
50005600	HUMBERTO	MORENO
50005500	ALEXANDRE	BLOND
50005300	ELISABETH	MAIZIERE
50004900	ALBERT	CAUDAL
50004600	BERNARD	VERDIE
50004300	MICHELE	GUERIN
50004200	BERNARD	VAUZELLE
50004100	ROBERT	CHAPUIS
50004000	JEAN	MONTASSIER
50003800	DANIEL	JOUSSELIN
50006900	PATRICK	BAILLET
50007600	JEAN-MARIE	MARX
50003700	LOUIS	D'AGOSTINO
50003500	MARC	LEROUGE
<b>MORE</b>		

Como você pode ver o comando "display" criou um relatório com número de página, data, hora e cabeçalho para o nível de grupo e para os campos. Ao pressionar enter, veremos os números de página mudando, enquanto o cabeçalho fica no topo de cada página nova.

O comando "input" é uma forma simples de obter entradas de um usuário online ou execução batch. O comando "display" possibilita uma formatação de campos muito básica mas que pode ser melhorada com posicionamento específico de campos e muito mais. Isso pode ser feito usando o editor de mapas que permite desenhar graficamente a tela ou o mapa.

Com o comando "input using map" você pode obter a entrada de dados a partir desse mapa. O "reinput" é normalmente utilizado quando há falha na validação de um campo e você deseja que o usuário corrija o erro. O comando "display" é para relatórios e oferece várias funcionalidades de formatação automática e o comando "write" se destina a saídas básicas. O comando "print" é especialmente útil para exibir strings dinâmicas longas, pois irá fazer automaticamente as quebras de linha, caso a saída não caiba numa mesma linha.

## Glossário

**Ambiente batch:** um ambiente batch é um sistema de processamento de dados no qual os trabalhos ou tarefas são executados em lote, ou seja, em lotes ou grupos de dados. Em outras palavras, é um ambiente em que os programas são executados em segundo plano,

sem interação do usuário, seguindo um fluxo pré-definido de processamento de dados. Nesse ambiente, os trabalhos são agrupados em lotes ou em grandes quantidades, que são processados sequencialmente por um sistema de processamento em lote. Cada trabalho dentro de um lote é executado de forma independente, sem interação do usuário. O sistema executa cada trabalho até o final, sem interrupções, e, em seguida, passa para o próximo trabalho do lote. Os ambientes batch são comuns em sistemas de processamento de transações financeiras, processamento de folha de pagamento, processamento de faturas e em outras áreas que exigem processamento de grande volume de dados em tempo diferido. Isso ocorre porque, em ambientes de processamento em lote, os trabalhos podem ser programados para serem executados fora do horário de pico, quando a carga no sistema é menor. Dessa forma, o processamento é realizado com mais eficiência e com menor impacto no desempenho do sistema em tempo real. Os sistemas de processamento em lote também são usados para processar grandes quantidades de dados em um único trabalho, como, por exemplo, quando se deseja gerar relatórios ou realizar cálculos complexos em grandes bases de dados. Em resumo, um ambiente batch é um sistema que processa trabalhos em lotes, sem interação do usuário, seguindo um fluxo pré-definido de processamento de dados.

## 5 - Processamento Condisional

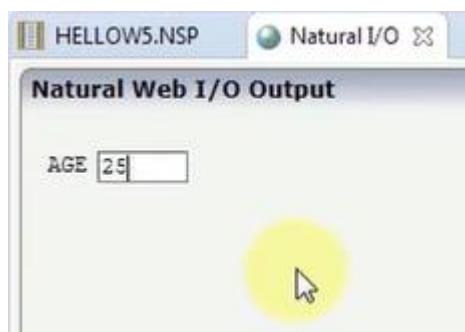
A forma mais simples de processamento condicional na maioria das linguagens é o comando "if". Iremos calcular um aumento de salário baseado na idade. Se a idade do empregado for inferior a 40, daremos a ele um aumento de 15%. Caso contrário, terá um aumento de 10%. Os blocos de "if" devem terminar com o comando "end-if".

```
HELLOW5.NSP ✘
+ * >Natural Source Header 000000
-/** New Program HELLOW5.
 /**
-** :author sagdfh
-define data local
-1 employee
- 2 name (A30)
- 2 age (I2)
- 2 salary (N7.2)
- 2 newsalary (N7.2)
end-define
move 100000 to salary

input age
if age < 40
compute rounded newsalary := salary * 1.15
else
compute rounded newsalary := salary * 1.10
end-if

write 'Age' age 'Salary' salary
write 'New salary' newsalary
END
```

Ao executar o programa, informamos a idade e pressionamos enter. Veremos que foi adicionado 15% ao salário.



```

HELOW5.NSP
Natural I/O

Natural Web I/O Output

Page 1
18-06-14 16:31:10

Age 25 Salary 100000.00
New salary 115000.00

```

Mas e se quisermos fazer a distinção entre vários valores ou intervalos? Poderemos fazê-lo com vários comandos "if" alinhados, mas o natural disponibiliza o comando "decide" que é mais conveniente, semelhante ao comando "switch" em outras linguagens.

É obrigatório definir o que deve acontecer se nenhum desses valores bater. No "none" a variável "newsalary" recebe "salary", ou seja, não ocorre nenhum cálculo.

```

* * >Natural Source Header 000000
/* New Program HELLOW5.
*/
/** :author sagdfh
define data local
 1 employee
    2 name (A30)
    2 age (I2)
    2 salary (N7.2)
    2 newsalary (N7.2)
end-define
move 100000 to salary

input age
decide on first value age
  value 25, 30
    compute rounded newsalary := salary * 1.15
  value 31:40
    compute rounded newsalary := salary * 1.10
  none
    newsalary := salary
end-decide

write 'Age' age 'Salary' salary
write 'New salary' newsalary
END

```

Se valores específicos ou intervalos não forem suficientes, o natural também tem uma variante da regra "decide" para expressões booleanas (true e false).

Veremos um exemplo com esse comando "decide for first condition" em que o valor é substituído pelo comando "when" e a condição que precisa ser aplicada.

```

input age
decide for first condition
when age > 30 and salary > 100000 I
    compute rounded newsalary := salary * 1.15
when age < 30 and salary > 110000
    compute rounded newsalary := salary * 1.10
when none
    newsalary := salary
end-decide

```

Assim, no primeiro "when", as condições são que a idade seja superior a 30 e o salário maior que 100000.

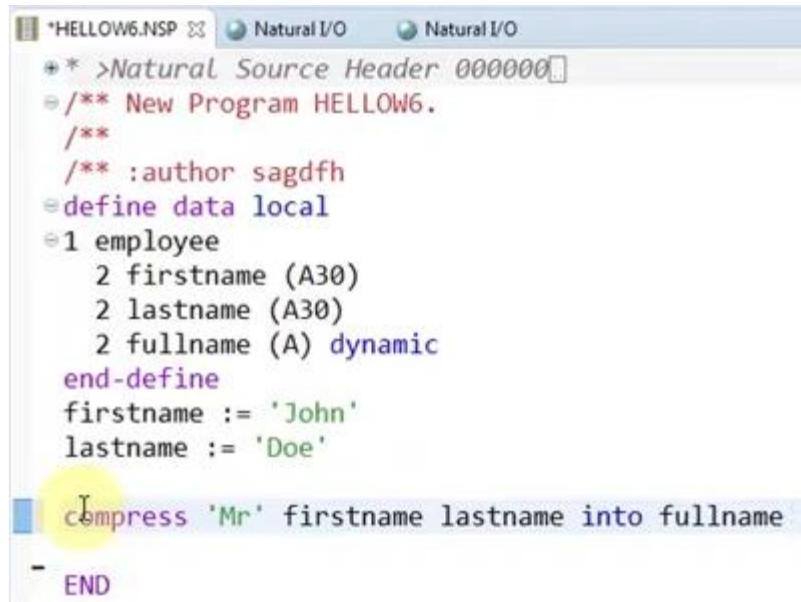
Statement	Description	Examples
IF	Performs statements depending on a logical condition. "ELSE" part is optional	<b>IF</b> age > 30 NewSalary := 100000 <b>ELSE</b> NewSalary := 110000 <b>END-IF</b>
DECIDE ON	Performs statements depending on the contents of a variable.	<b>DECIDE ON FIRST VALUE OF</b> age <b>VALUE</b> NewSalary := 110000 <b>NONE</b> NewSalary := Salary <b>END-DECIDE</b>
DECIDE FOR	Performs statements depending on logical conditions.	<b>DECIDE FOR FIRST CONDITION</b> <b>WHEN</b> Age > 30 and Salary > 100000 NewSalary := 110000 <b>WHEN NONE</b> NewSalary := Salary <b>END-DECIDE</b>

## 6 - Processamento de Strings

Agora veremos como processar Strings de texto no Natural. Vamos ver como analisar e concatenar Strings e como alterar o seu conteúdo para, por exemplo, maiúsculas. Para lhe mostrar como as cadeias são concatenadas ou compactadas, alterei minhas definições de variáveis para conterem um nome e um sobrenome do tipo (A30). Também há um campo "fullname" de tipo dinâmico para o resultado. Também atribui valores ao "nome" e ao "sobrenome".

2 firstname (A30)  
 2 lastname (A30)  
 2 fullname (A) dynamic

Posso agora compactar várias constantes ou variáveis numa String. Vou solicitar a compressão de “Mr”, “firstname” e “lastname” para dentro de “fullname”.



```
* * >Natural Source Header 000000
/* New Program HELLOW6.
**
/** :author sagdfh
define data local
@1 employee
    2 firstname (A30)
    2 lastname (A30)
    2 fullname (A) dynamic
end-define
firstname := 'John'
lastname := 'Doe'

compress 'Mr' firstname lastname into fullname
-
END
```

Em seguida, vou usar um “print fullname” para enviar para a tela. Vamos salvar e executar. Aqui temos o resultado de compactar as variáveis com um espaço adicionado entre cada elemento.

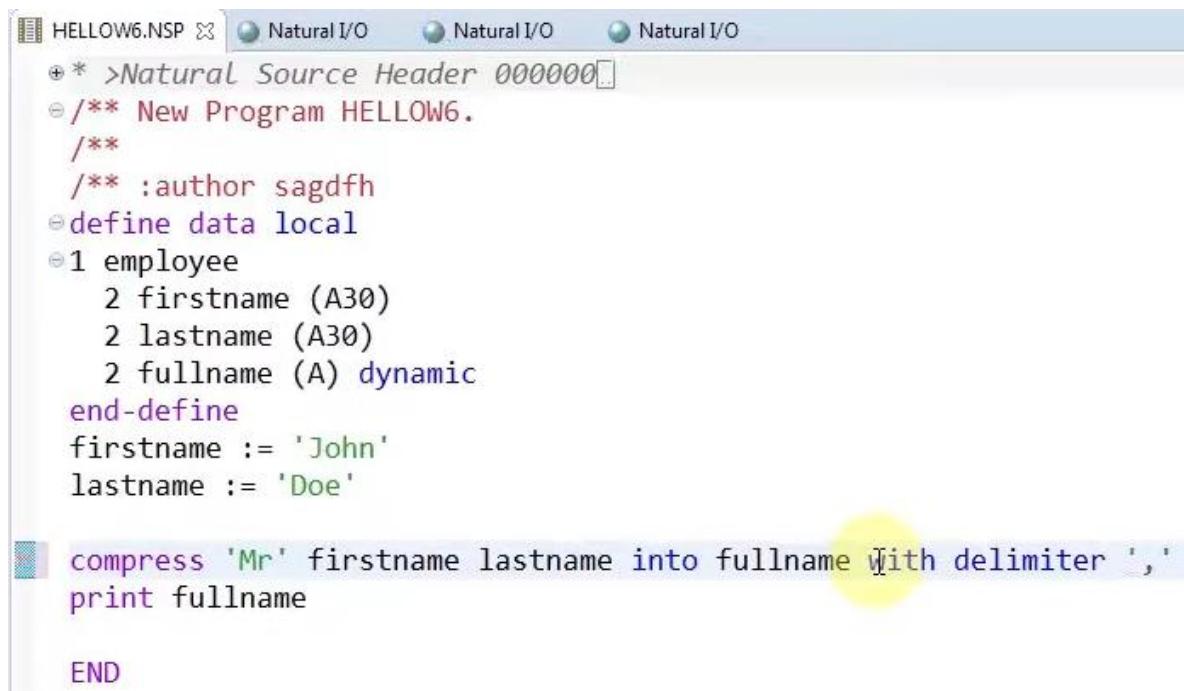
#### Natural Web I/O Output

Page 1

18-06-21 10:42:23

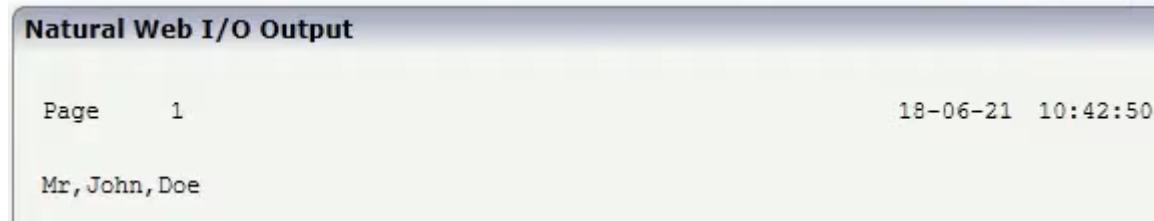
Mr John Doe

Há várias opções adicionais para o comando de compactação. Por exemplo, posso definir um delimitador ao invés do espaço, como uma vírgula, caso precise criar [um arquivo CSV separado por vírgula](#).



```
+ * >Natural Source Header 000000..  
+/** New Program HELLOW6.  
+**  
+** :author sagdfh  
+define data local  
+1 employee  
+ 2 firstname (A30)  
+ 2 lastname (A30)  
+ 2 fullname (A) dynamic  
+end-define  
+firstname := 'John'  
+lastname := 'Doe'  
  
+compress 'Mr' firstname lastname into fullname with delimiter ','  
+print fullname  
  
END
```

Ao executar o programa:



Natural Web I/O Output

Page	1	18-06-21 10:42:50
Mr, John, Doe		

Também podemos pegar os valores separados por vírgulas e coloca-los em uma matriz. Para fazer isso, defino uma matriz simples com três ocorrências do tipo (A30).

1 elements (A30/1:3)

Assim vou separar “fullname” nos elementos da matriz.

separate fullname into elements(\*)

O asterisco entre parênteses indica que não estamos lidando com um índice específico da matriz, mas com a própria matriz. Para exibir isso na tela, dessa vez vou usar o comando display que vai listar os valores de toda a matriz linha a linha.

```
HELLOW6B.NSP
+* >Natural Source Header 000000
@/** New Program HELLOW6B.
 /**
 ** :author sagdfh
 @define data local
 @1 employee
   2 firstname (A30)
   2 lastname (A30)
   2 fullname (A) dynamic
 *
 1 elements (A30/1:3)
 end-define
 firstname := 'John'
 lastname := 'Doe'

 compress 'Mr' firstname lastname into fullname with delimiter ','

 separate fullname into elements(*)
 display elements(*)

END
```

Ao executar o programa:

```
Natural Web I/O Output

Page      1                               18-06-21 12:41:43

ELEMENTS
-----
Mr
John
Doe
```

No exemplo seguinte vou mostrar como examinar uma string para determinar o número de ocorrências de um caractere ou string específica, como uma vírgula por exemplo. Também retornaremos o índice da primeira ocorrência da vírgula. Criei duas variáveis do tipo (I2) para guardar esses valores.

1 n (I2) → número de ocorrências  
1 ix (I2) → posição da primeira ocorrência

Assim vou inserir “examine fullname for ‘,’ giving number n position ix”.

```
HELLOW6C.NSP
+* >Natural Source Header 000000
/** New Program HELLOW6B.
**
/** :author sagdfh
#define data local
1 employee
2 firstname (A30)
2 lastname (A30)
2 fullname (A) dynamic
*
1 n (i2) /* Number of occurrences
1 ix (i2) /* position of first occurrence
end-define
firstname := 'John'
lastname := 'Doe'

compress 'Mr' firstname lastname into fullname with delimiter ','

examine fullname for ',' giving number n position ix

print fullname
write 'number' n 'index' ix|
```

END

Ao executar o programa:

```
Natural Web I/O Output

Page      1                               18-06-21 13:33:32
Mr, John, Doe
number    2 index      3
```

E aqui temos nosso resultado do número de vírgulas e a posição da primeira no índice.

O comando “examine” tem várias outras funcionalidades, como substituir caracteres específicos que podem ser usados para transformações em maiúsculas e minúsculas. A sintaxe para transformar em maiúsculas é “examine fullname translate into up”.

```
HELL0W6D.NSP
/* >Natural Source Header 000000
/** New Program HELLOW6C.
/*
/** :author sagdfh
#define data local
#1 employee
 2 firstname (A30)
 2 lastname (A30)
 2 fullname (A) dynamic
end-define
firstname := 'John'
lastname := 'Doe'

compress 'Mr' firstname lastname into fullname with delimiter ','

print 'Before:' fullname

examine fullname translate into upper

print 'After :' fullname

END
```

Ao executar o programa:

Natural Web I/O Output	
Page	1
	18-06-21 15:22:26
Before: Mr,John,Doe	
After : MR,JOHN,DOE	

## Comandos

- Compress:** compactar constantes e variáveis numa String de destino.
- Separate:** faz o contrário do “compress”, separando os elementos individuais de uma String em campos, frequentemente com uma matriz.
- Examine:** nos permite pesquisar e substituir caracteres e determinar a quantidade de um determinado caractere e seu índice.
- Examine Translate:** é uma forma simples de transformar uma String em maiúsculas ou minúsculas.
- Substring:** pode ser usada em conjunto com as outras regras para definir que apenas uma parte da cadeia deva ser processada. Em alguns casos a função também pode ser usada para definir um local na String de destino.

Statement	Description	Example
<b>COMPRESS</b>	Concatenates the value of two or more fields into a single field.	<b>COMPRESS</b> 'Mr' firstname lastname <b>INTO</b> fullname <b>COMPRESS</b> elements(*) <b>INTO</b> fullname
<b>SEPARATE</b>	Separates the content of a field into two or more fields.	<b>SEPARATE</b> fullname <b>INTO</b> elements(*)
<b>EXAMINE</b>	Scans a field for a specific value and replaces it, and/or counts how often it occurs.	<b>EXAMINE</b> fullname <b>GIVING</b> NUMBER n <b>POSITION</b> ix
<b>EXAMINE TRANSLATE</b>	Translates the characters contained in a field into upper-case or lower-case, or into other characters.	<b>EXAMINE</b> fullname <b>TRANSLATE</b> <b>INTO</b> UPPER
<b>SUBSTRING</b>	Function to specify only a part of the string based on offset and length – for both source or target.	NameWithoutTitle := <b>SUBSTRING</b> (fullname, 4, 30) <b>MOVE</b> 'xx' <b>TO</b> <b>SUBSTRING</b> (fullname,4,2)

## Glossário

**Arquivo CSV:** CSV é um formato de arquivo que significa “comma-separated-values” (valores separados por vírgulas). Isso significa que os campos de dados indicados neste formato normalmente são separados ou delimitados por uma vírgula. Para entender de uma forma mais prática, vamos supor que você tenha uma planilha que contenha os dados a seguir:

João	2018	Belo Horizonte
Maria	2019	Rio de Janeiro

Esses dados poderiam ser lidos em um arquivo CSV separados por vírgulas e por um espaçamento de linha, como no exemplo a seguir:

João,2018,Belo Horizonte  
Maria,2019,Rio de Janeiro

## 7 - Processamento de Loop

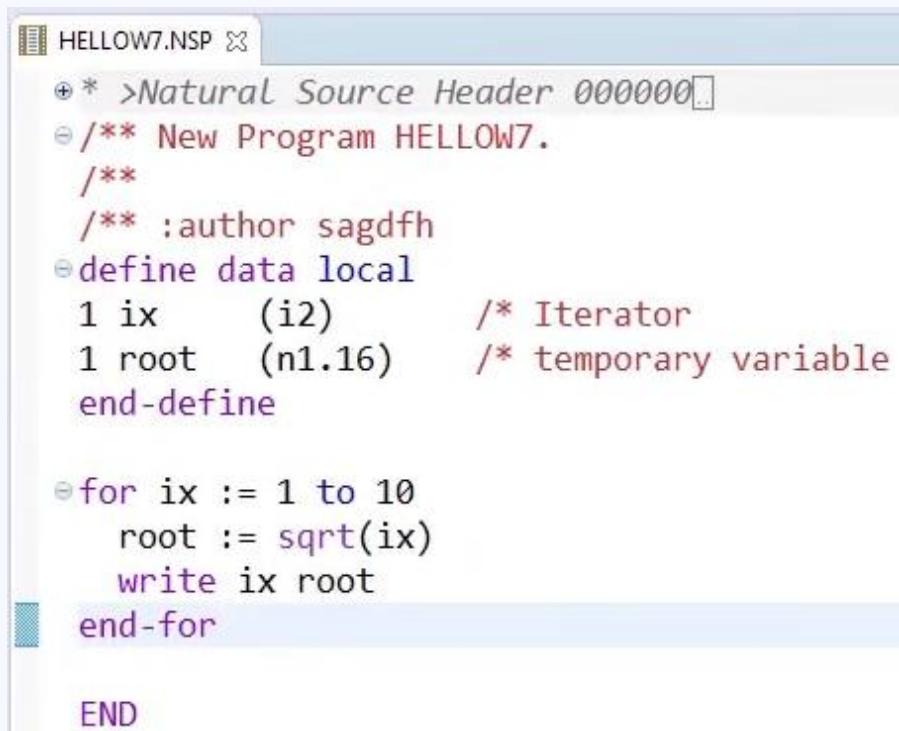
Neste capítulo mostraremos como usar os dois tipos de laço no natural. O laço controlado com o “for” e o laço flexível com o “repeat”.

O laço “for” é usado em situações em que se sabe antecipadamente quantas iterações se deseja processar. A sintaxe requer uma variável de controle do laço (ix) que definimos aqui como um inteiro de dois bytes (i2). Também definimos uma variável chamada “root” para guardar o valor quadrado, com 1 dígito antes do ponto decimal e 16 dígitos depois do ponto decimal.

1 ix (i2)  
1 root (n1.16)

Então vamos escrever o “for” mas a variável “ix” é definida com valores inicial e final, que nesse caso são de 1 até 10. Esses valores podem ser constantes ou variáveis.

Vamos calcular a raiz quadrada usando a função integrada de raiz quadrada do Natural e deixar que ele imprima a variável de controle e a raiz. Veremos como a raiz é calculada para cada um dos dez valores.



```
HELLOW7.NSP ✘
+ * >Natural Source Header 000000..
⊖ /** New Program HELLOW7.
    /**
    /** :author sagdfh
⊖define data local
    1 ix      (i2)      /* Iterator
    1 root    (n1.16)    /* temporary variable
end-define

⊖for ix := 1 to 10
    root := sqrt(ix)
    write ix root
end-for

END
```

Ao executar o programa:

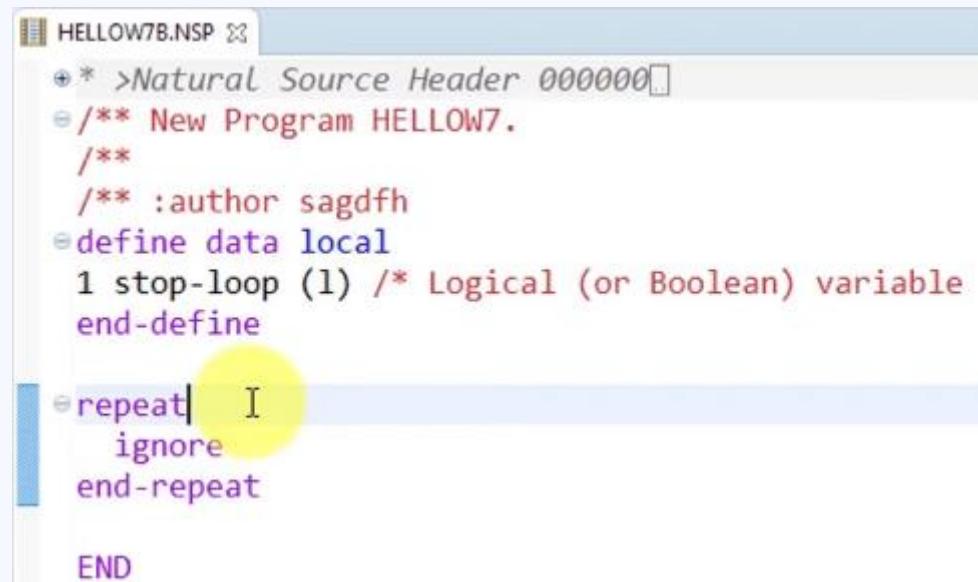


Natural Web I/O Output

Page	1	18-11-06 12:18:46
1	1.00000000000000000000	
2	1.4142135623730951	
3	1.7320508075688772	
4	2.00000000000000000000	
5	2.2360679774997898	
6	2.4494897427831779	
7	2.6457513110645907	
8	2.8284271247461903	
9	3.00000000000000000000	
10	3.1622776601683795	

Ao contrário do laço “for”, o laço com “repeat” tem mais opções relacionadas as condições que fazem com que ele pare de executar. Vamos criar um laço de repetição básico com o “repeat” para iniciar o laço, em seguida de uma instrução fictícia qualquer e o “end-repeat” para o fim do laço. Este programa seria sintaticamente válido, mas também

muito tolo, porque literalmente faria o natural entrar em um loop eterno. Em outras palavras, não tente isso em casa.

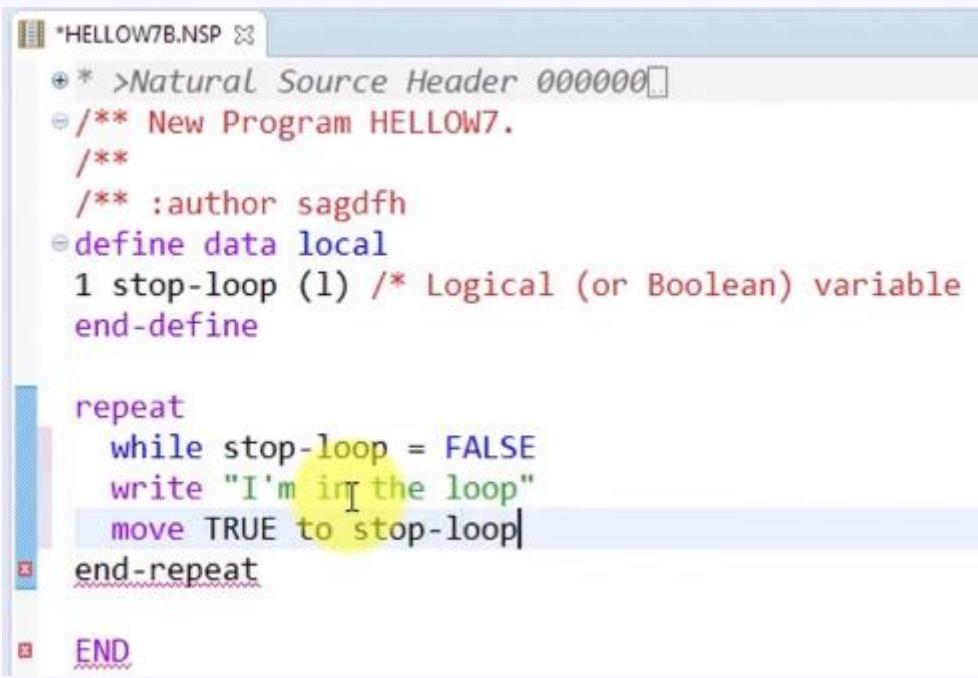


```
* >Natural Source Header 000000
/** New Program HELLOW7.
**
/** :author sagdfh
define data local
1 stop-loop (1) /* Logical (or Boolean) variable
end-define

repeat I
ignore
end-repeat

END
```

Lembre-se de especificar uma condição que faça com que o laço termine. A instrução “repeat” nos dá duas maneiras de fazer isso. A primeira é a sintaxe “while” na qual você especifica uma condição lógica. Normalmente no início do laço. Indicando que se essa condição não for verdadeira o restante do laço não deverá ser executado. Criamos um boolean. Há uma instrução “while” no começo para testar essa condição, uma instrução “write” para mostrar que estamos no laço e finalmente uma instrução “move” para alterar a variável.



```
* >Natural Source Header 000000
/** New Program HELLOW7.
**
/** :author sagdfh
define data local
1 stop-loop (1) /* Logical (or Boolean) variable
end-define

repeat
while stop-loop = FALSE
write "I'm in the loop"
move TRUE to stop-loop
end-repeat

END
```

Ao salvar e executar isso você verá que só entraremos no laço uma vez.

**Natural Web I/O Output**

Page 1	18-11-06 15:43:52
I'm in the loop	

A segunda sintaxe para encerrar um laço de repetição é o “repeat until” em que a condição normalmente é posicionada na parte inferior do laço para determinar se o laço deve terminar agora ou se deve ser repetido. Então se eu deslocar o teste para baixo veremos que a sintaxe “until” é formulada de forma lógica para ser exatamente o oposto da condição “while”. Então repetiremos o laço até que a variável “stop loop” seja verdadeira.

HELLOW7C.NSP

```
④ * >Natural Source Header 000000
④ /** New Program HELLOW7C.
 /**
 /**
 :author sagdfh
 define data local
 1 stop-loop (1) /* Logical (or Boolean) variable
 end-define

 repeat
   write "I'm in the loop"
   move TRUE to stop-loop
   until stop-loop = TRUE
 end-repeat

END
```

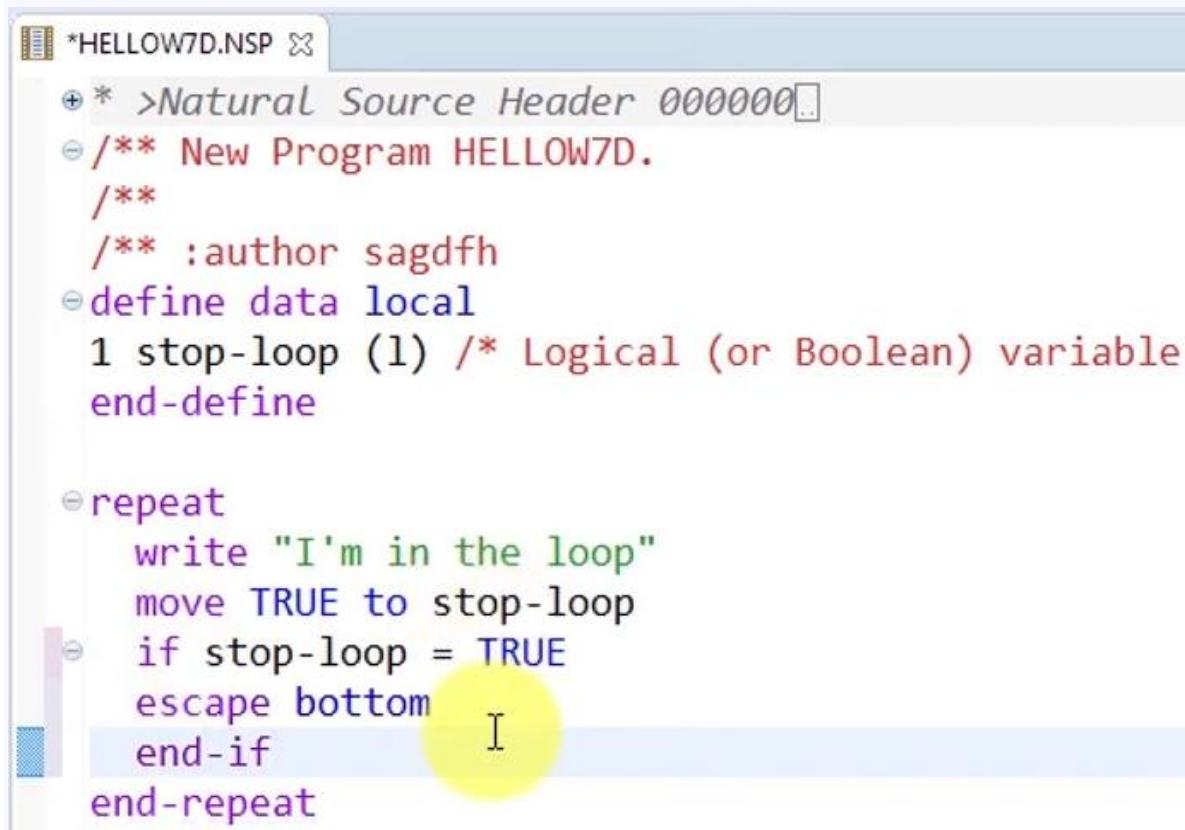
Ao executarmos o programa entraremos no laço somente uma vez.

HELLOW7C.NSP Natural I/O

**Natural Web I/O Output**

Page 1	18-11-06 16:34:46
I'm in the loop	

Há também uma terceira opção de saída de emergência em que uma instrução “if” é usada em combinação com a instrução “escape bottom” para finalizar o laço atual. Então vamos substituir o “until” por uma estrutura “if” e adicionar “escape bottom” que pode ficar em qualquer parte do laço.

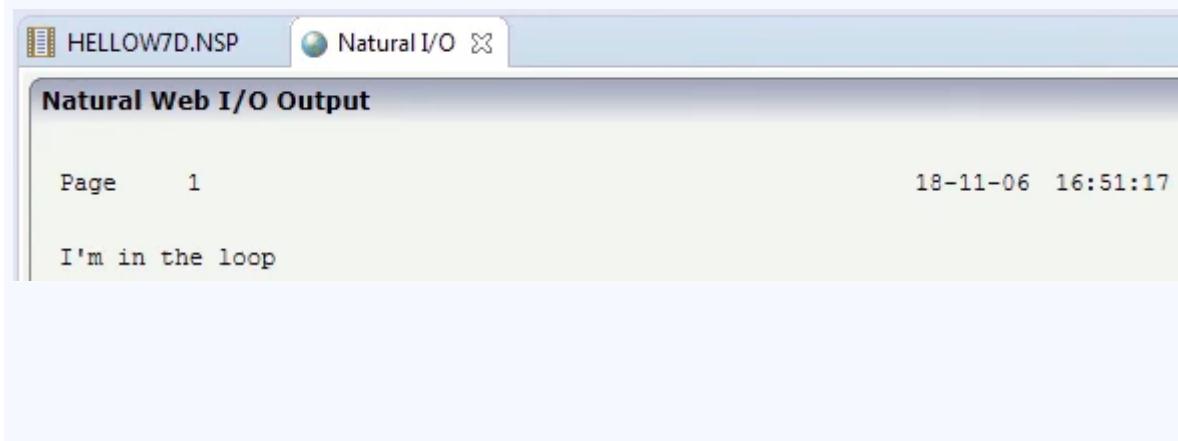


```
*HELLOW7D.NSP
+ * >Natural Source Header 000000
-/** New Program HELLOW7D.
 /**
 /** :author sagdfh
 -define data local
 1 stop-loop (1) /* Logical (or Boolean) variable
 end-define

 -repeat
   write "I'm in the loop"
   move TRUE to stop-loop
 -  if stop-loop = TRUE
     escape bottom
   end-if
 end-repeat
```

END

Ao executar o programa temos:



```
HELLOW7D.NSP Natural I/O
Natural Web I/O Output

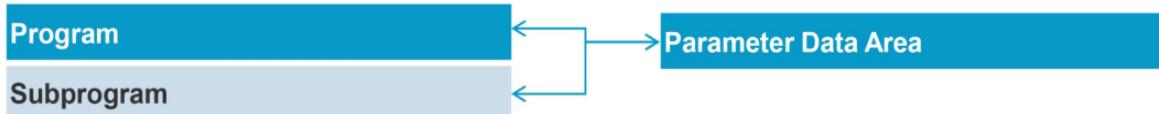
Page      1          18-11-06  16:51:17
I'm in the loop
```

Resumindo, o laço “for” é usado para executar o laço um número específico de vezes. A sintaxe “step” permite diminuir o valor ou ajustar o intervalo. O “repeat while” testa a condição no início do laço e o “repeat until” no final. O “escape bottom” em uma instrução “if” pode ser usado em qualquer lugar, mas é menos estruturado.

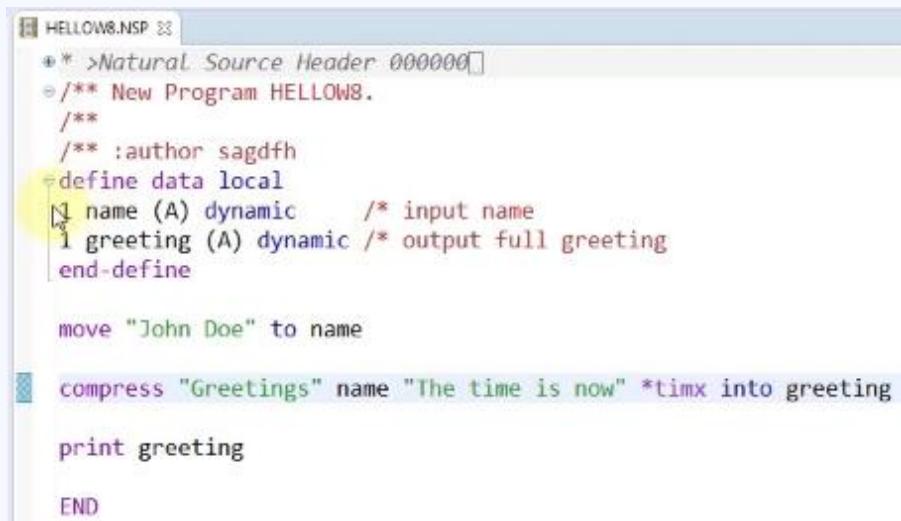
Statement	Description	Example
FOR	Initiates a processing loop and controls the number of times the loop is to be processed.	<b>FOR</b> ix := 1 <b>TO</b> 10 <b>WRITE</b> ix <b>END-FOR</b>
	Use <b>STEP</b> for descending or intervals For loop statements must always be ended with <b>END-FOR</b> .	<b>FOR</b> ix := 10 <b>TO</b> 1 <b>STEP</b> -1 <b>WRITE</b> ix <b>END-FOR</b>
REPEAT	Initiates a processing loop (and can terminate it based on a specified condition).	<b>REPEAT WHILE</b> stop-loop = FALSE .. <b>END-REPEAT</b>
	The <b>WHILE</b> or <b>UNTIL</b> conditions can be placed either at the start or end of the loop.  The loop statements must always be ended with <b>END-REPEAT</b>	<b>REPEAT</b> .. <b>UNTIL</b> stop-loop = TRUE <b>END-REPEAT</b>

## 8 - Modularização e Subprogramas

Neste capítulo vamos ver como o código fonte do Natural pode ser separado em módulos ou objetos facilmente reutilizáveis. Na nossa demonstração você verá como a lógica de um programa principal pode ser refatorada em um módulo reutilizável, neste caso um “subprogram” e como usamos o “parameter data area” para fazer a interface.



Neste exemplo, criei um programa principal “Hello World”, que compacta a saudação e a variável “name” em uma String, que inclui ainda a data e a hora.



```
* >Natural Source Header 000000
/** New Program HELLOW8.
**
/** :author sagdfh
define data local
  name (A) dynamic /* input name
  greeting (A) dynamic /* output full greeting
end-define

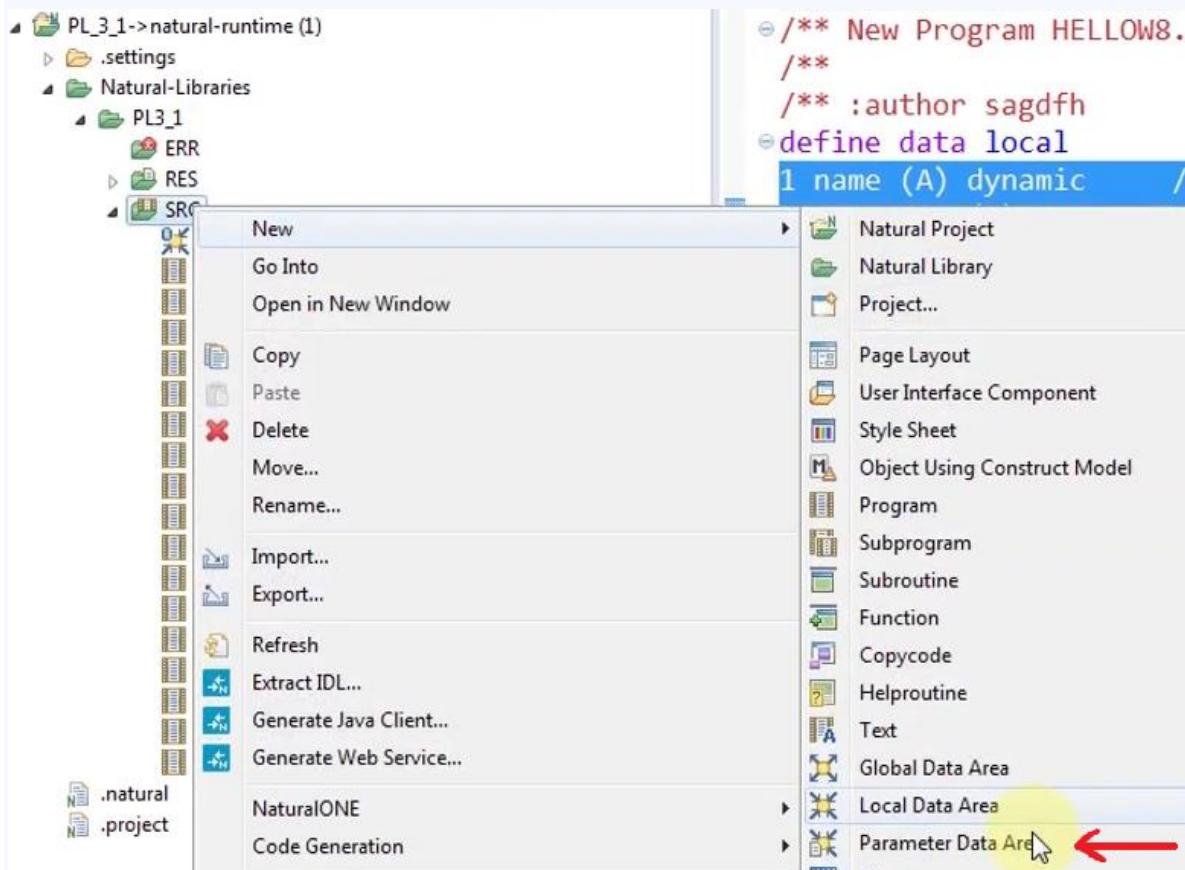
move "John Doe" to name

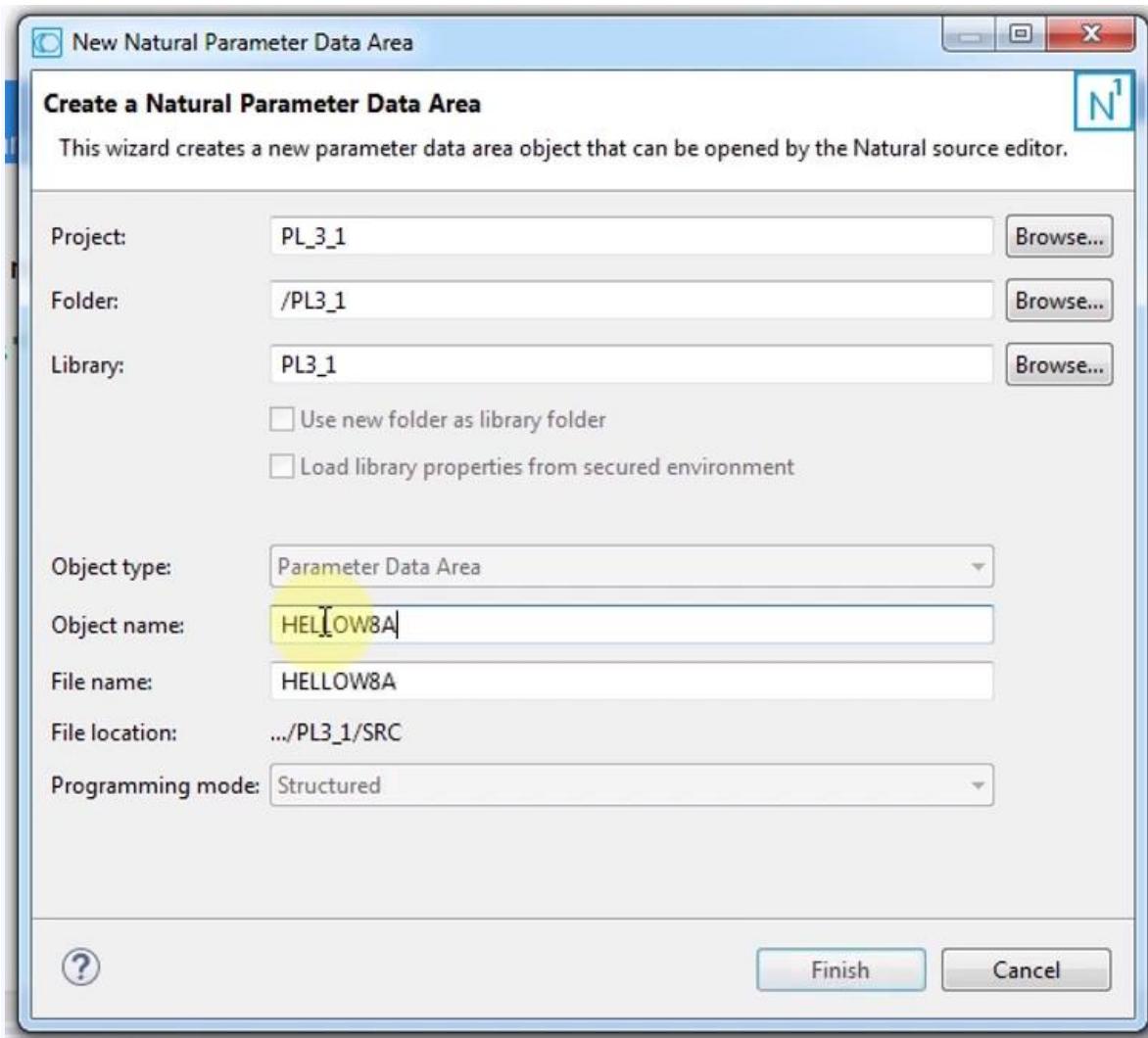
compress "Greetings" name "The time is now" *timx into greeting

print greeting

END
```

Como as saudações são frequentemente usadas em demonstrações como essa, gostaria de disponibilizar essa parte do código para outros programas. Minha abordagem seria primeiro criar um módulo que contém as variáveis necessárias. Clicamos com o botão direito em “src + new + parameter data area” e lidamos um nome.

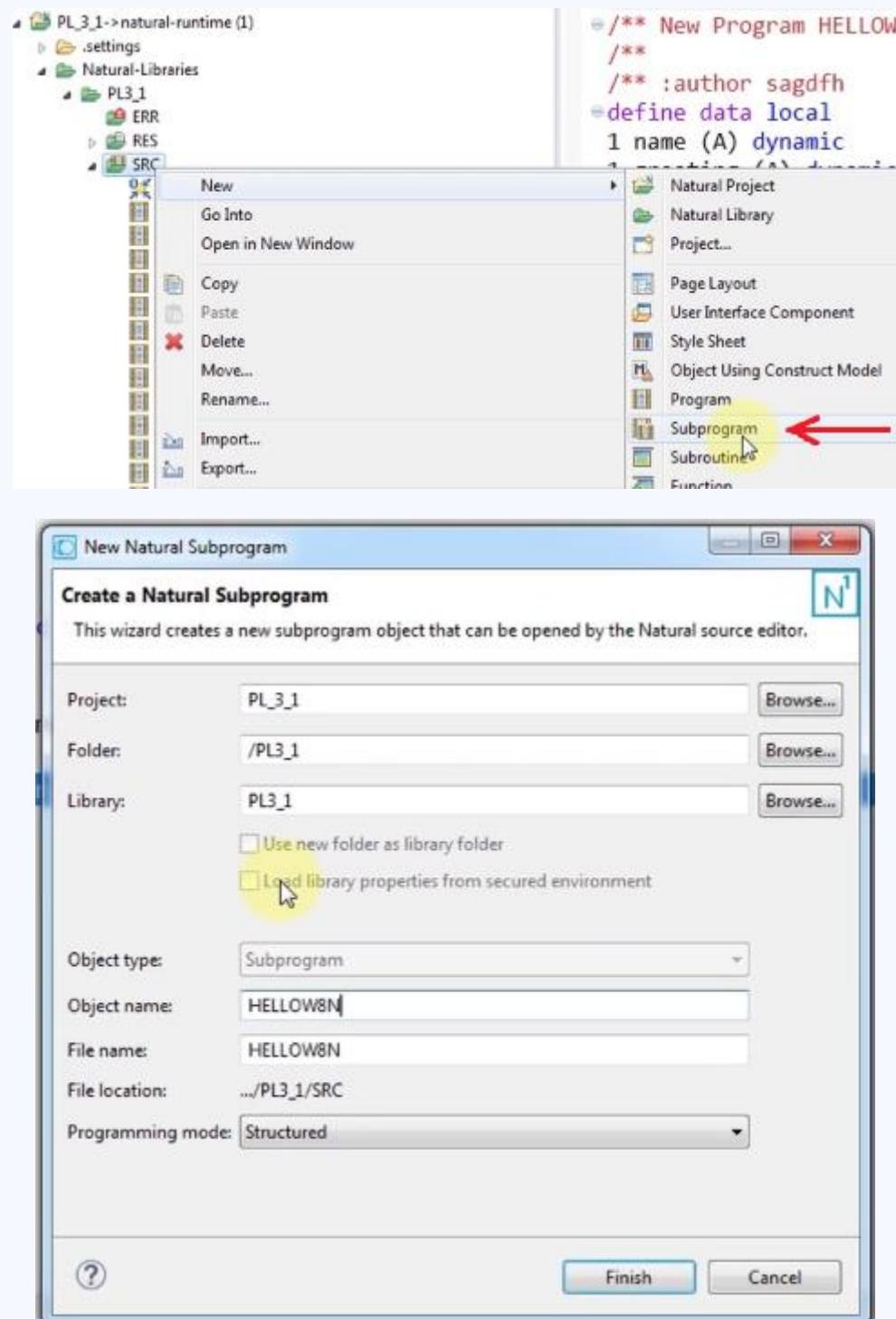




Depois colocamos as variáveis no novo módulo e salvamos.

```
HELLOW8.NSP HELLOW8A.NSA
@DEFINe DATA PARAMETER
+/* >Natural Source Header 000000
+/** New Parameter Data Area HELLOW8A.
 /**
 /**
 :author sagdfh
 1 name (A) dynamic /* input name
 1 greeting (A) dynamic /* output full greeting
END-DEFINe
```

Agora vamos voltar ao programa e copiar o código fonte que nos interessa. Clicamos com o botão direito em “src + new + subprogram” para criar um subprograma que controla essa lógica de negócios avançada e lhe damos um nome.



Colocamos a lógica:

```
+ * >Natural Source Header 000000
-/** New Subprogram HELLOW8N.
/**
/** :author sagdfh
compress "Greetings" name "The time is now" *timx into greeting
END
```

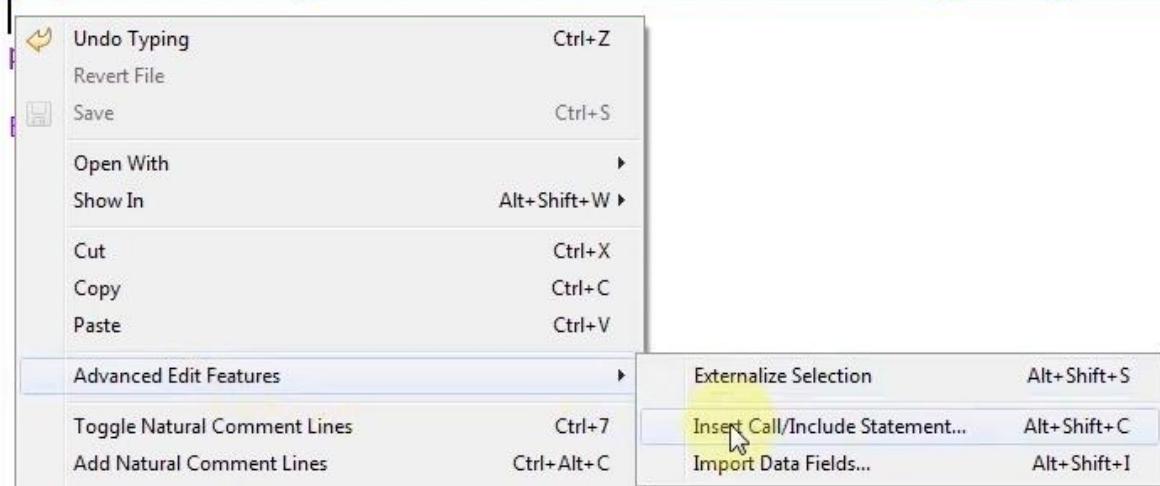
Como criamos o módulo de variáveis, podemos reutilizá-lo para definir os parâmetros. Fazemos isso adicionando um bloco “define data parameter”. Adicionamos a palavra-chave “using” afim de indicar que estamos usando o módulo e em seguida, o nome do módulo.

```
+ * >Natural Source Header 000000
-/** New Subprogram HELLOW8N.
/**
/** :author sagdfh
define data parameter
using hellow8a
end-define
compress "Greetings" name "The time is now" *timx into greeting
END
```

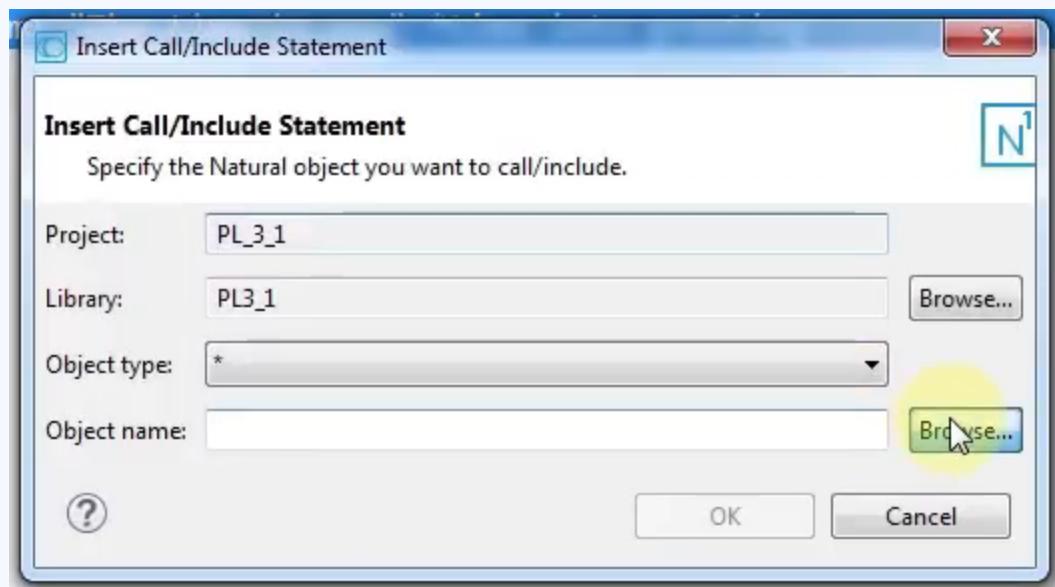
Se agora voltarmos ao programa, podemos substituir a lógica de negócios por uma chamada para o subprograma. Fazemos isso ao clicar com o botão direito, escolher "advanced edit features" e a opção de inserir a instrução "insert call include statement".

```
move "John Doe" to name
```

```
compress "Greetings" name "The time is now" *timx into greeting
```



Vamos procurar o subprograma que acabamos de criar. Selecione-o e clique em OK.



Em seguida a instrução “callnat” carrega o subprograma e todos os parâmetros são criados automaticamente.

```
* >Natural Source Header 000000
/* New Program HELLOW8.
*/
/** :author sagdfh
define data local
1 name (A) dynamic      /* input name
1 greeting (A) dynamic /* output full greeting
end-define

move "John Doe" to name

CALLNAT "HELLOW8A" name greeting

print greeting

END
```

A última coisa que precisamos fazer é reutilizar as definições de variáveis que criamos no início. Caso o parâmetro mude em algum momento geralmente precisamos apenas recompilar os módulos.

```
* >Natural Source Header 000000
/* New Program HELLOW8.
*/
/** :author sagdfh
define data local
- I using hellow8a
end-define

move "John Doe" to name

CALLNAT "HELLOW8A" name greeting

print greeting

END
```

Ao executar o programa temos:

HELLOW8.NSP	HELLOW8A.NSA	HELLOW8N.NSN	Natural I/O
<b>Natural Web I/O Output</b>			
Page	1	18-11-07 15:12:21	
Greetings John Doe The time is now 15:12:21			

Esta é uma lista completa de todos os tipos de objetos no Natural:

Programs and Subordinate Routines	Data Areas
Program	Parameter Data Area
Subprogram	Local Data Area
Subroutine	Global Data Area
Copycode	
Function	
Database Connection	User interfaces
DDM (Data Definition Module)	Map
	Helproutine
	Dialog (only windows)
	Ajax GUI (browser based)

Os subprogramas são um tipo de objeto único mais importante em muitas instalações do Natural atualmente, porque podem ser disponibilizados a plataformas de clientes externos como Java, .Net e a popular interface API Rest com apenas alguns cliques.

## 9 - Características do Natural

O Natural é uma ferramenta de desenvolvimento de 4ª geração desenvolvida pela Software AG e distribuída no Brasil pela Consist. Além de possuir a versão para MVS, possui versões para UNIX, OS/2 e Windows. O MVS foi o sistema operacional mais utilizado nos mainframes IBM System/370 e System/390.

Algumas características do Natural são:

- Linguagem de programação com acesso a diversos bancos de dados: Adabas (hierárquico e relacional), Oracle, Db2, entre outros.
- Utiliza diversos editores para criar programas, funções, telas pré-formatadas e áreas de dados.
- Permite uma programação modularizada.
- Utiliza funções e variáveis do sistema.
- Permite execução on-line e batch.
- Possui utilitário de testes de programação.
- Possui utilitário de criação/manutenção de mensagens de erros de aplicação.
- Qualquer aplicação pode ser facilmente portada para várias outras plataformas.

Os objetos Natural (programas, mapas, áreas de dados) são armazenados em bibliotecas (Libraries), com estrutura parecida com o diretório DOS e podem ter no máximo 8 caracteres como nome. Mesmo sendo objetos de diferentes tipos, não podem possuir o mesmo nome.

As principais vantagens do Natural em relação às linguagens tradicionais são:

- Maior facilidade de acesso a Banco de Dados (Adabas, Db2, Oracle).
- Maior velocidade de programação.
- Maior dinamismo em testes e manutenção de programas.
- Menor preocupação com os demais recursos de instalação.

## Componentes

O Natural tem os seguintes componentes principais:

- Núcleo: executa os comandos e instruções e gerencia os demais componentes.
- Compilador: de execução interativa;
- Módulos de interface:
  - Com o Sistema Operacional: executa todos os comandos de entrada/saída e demais comunicações.
  - Com o monitor de TP: gerencia todos os recursos necessários à aplicação.
  - Com o Adabas: executa todos os comandos Adabas e controla vários processos de acessos/atualizações.
- System File: arquivo Adabas de suporte ao Natural.

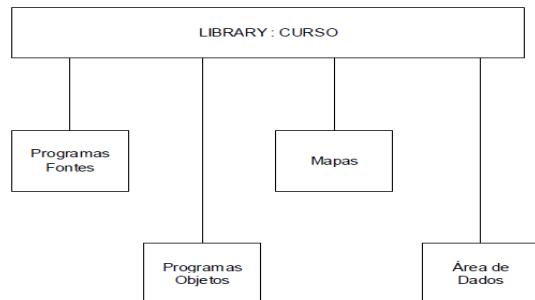
## System File

O System File é um arquivo Adabas reservado para uso do Natural, que contém:

- Todos os programas Natural, tanto em formato fonte (programas) como em formato objeto (compilado), agrupados em bibliotecas.
- Os módulos de definição de arquivos, ou DDM's (Data Definition Modules), com a definição para o Natural dos arquivos Adabas e suas userviews.
- As mensagens de erro do Natural.
- Os textos da função "Help".

## Bibliotecas

São formadas por um conjunto de programas fontes e/ou carga, geralmente pertencentes ao mesmo sistema. Em uma sessão Natural, as "Libs" são estabelecidas pelo comando "LOGON". As chamadas de programas a programas são feitas dentro da mesma biblioteca.



## **Programas**

Os programas Natural, fontes e objetos são identificados por nomes com formação igual aos nomes das bibliotecas, ou seja, até 8 caracteres, sendo o primeiro alfabético. Fontes e objetos podem (mas não devem) ter nomes diferentes. Os programas fontes são formados por linhas de até 72 posições, numeradas por 4 dígitos. Esta numeração é gerada pelo Natural durante a criação do programa, e é usada tanto pelo compilador e editores do Natural como também tem importantes funções lógicas dos programas. Dentro das linhas, a disposição das instruções (statements ou comandos de programas) é livre, não havendo instruções nem parâmetros posicionais.

Comentários podem ser incluídos de duas formas:

- Toda linha será considerada um comentário se contiver \* ou \*\* nas duas primeiras posições.
- A parte de uma linha à direita de /\* será considerada um comentário.

Exemplo:

```
0010 * Estas duas linhas (0010 e 0020)
0020 ** são comentários.
0030 FORMAT LS=80 /* Bem como esta parte da linha (0030)
.
.
0200 END
```

## **DDM**

DDM's são módulos que descrevem um arquivo ADABAS e seus campos, ou uma userview de um arquivo ADABAS. As DDM's são identificadas por um nome de até 32 caracteres, nome este também usado para identificar os arquivos nos programas Natural.

Em um programa Natural todas as características dos campos de banco de dados são retiradas das DDM's não havendo a necessidade de se definir esses campos nos programas.

O processo de obtenção das características dos campos é feito em tempo de compilação, e elas são agregadas aos programas quando estes são catalogados. Portanto, os programas em formato objeto não utilizam as DDM's.

A estrutura de DDM's é mostrada a seguir:

***** NATURAL LIST COMMAND *****						03/01/2003				
User X121212 - List DDM VEHICLES -						Library CURSO				
DDM	DBID	0	DDM	FNR	119	VSAM Name	Default Sequence	Page	1	
T	L	DB	Name			F	Leng	S	D	Remarks
-	-	-	-	-	-	-	-	-	-	-
1	AA	REG-NUM				A	15	N	D	CAR'S REGISTRATION N
1	AB	CHASSIS-NUM				I	4		F	
1	AC	PERSONNEL-ID				A	8	D		CNNNNNNNN
G	1	CD	CAR-DETAILS							DESCRIPTION OF THE C
2	AD	MAKE				A	20	N	D	
2	AE	MODEL				A	20	N		
2	AF	COLOR				A	10	N	D	
1	AG	YEAR				N	4,0	N		YEAR OF THE CAR'S MA
1	AH	CLASS				A	1	F	D	P=PRIVAT
1	AI	LEASE-PUR				A	1	F		L=LEASED
1	AJ	DATE-ACQ				N	8,0	N		DATE WHEN CAR WAS AC
1	AL	CURR-CODE				A	3	N		CURRENCY OF MAINTENA
M	1	AM	MAINT-COST			P	7,0	N		MAINTENANCE COST IN
1	AN	DAT-ACQ-DESC				B	4	N	S	CAR AGE BASED ON DAT
*			----- SOURCE FIELD(S) -----							
*			DATE-ACQ(5-6)							
*			DATE-ACQ(3-4)							
1	AO	MODEL-YEAR-MAKE				A	22	N	S	MODEL YEAR + CAR MAK
*			----- SOURCE FIELD(S) -----							
*			YEAR(1-2)							
*			MAKE(1-20)							

***** NATURAL LIST COMMAND *****						03/01/2003				
User X121212 - List DDM DB2TST-CADASTRO_CURSO -						Library CURSO				
DDM	DBID	250	DDM	FNR	1	VSAM Name	Default Sequence ?	Page	1	
T	L	DB	Name			F	Leng	S	D	Remarks
-	-	-	-	-	-	-	-	-	-	-
1	OA	CD_USU				A	11		D	
1	AA	COD_CPF_CGC				P	14,0		D	
1	AB	NOM				A	60		D	
1	AC	NR_SEQL_END				I	2		D	
1	AD	SEXO				A	1		D	
1	AE	NVL_ECLD				I	2		D	
1	AF	DT_ATL				A	10		D	

## Objetos Natural

Um objeto Natural nada mais é do que um componente de um sistema. Os diversos tipos de objetos Natural são:

- COPYCODE: Pedaço de código fonte, que pode ser aproveitado em vários programas. Único objeto Natural que não é compilado. Nos programas onde o COPYCODE vai ser aproveitado, através do comando INCLUDE, ele será compilado, juntamente com o programa.

- GLOBAL: Objeto utilizado para definição de variáveis. Todos os objetos que se referenciarem a uma GLOBAL, terão as mesmas variáveis, e estas manterão os valores após o término de um objeto, permitindo que um objeto altere valores e outro reconheça as alterações.

- LOCAL: Objeto utilizado para definição de variáveis. A diferença entre uma LOCAL e uma GLOBAL é que valores das variáveis locais somente serão reconhecidos pelo objeto onde foram declaradas, deixando de existir quando do término do mesmo.

- PARAMETER: Objeto utilizado para definição de variáveis que serão utilizadas entre programas e subprogramas.

- MAP: Utilizado para entrada e exibição de dados. Permite a definição de Regras de Validação para os campos, inclusive com acesso à Base de Dados.

- Helproutine: Tipo de objeto que permite ser definido como ajuda a um campo de entrada de dados. Quando o cursor estiver posicionado num campo e for teclado "?", será chamada a HELPROUTINE que estiver associada ao campo.

- PROGRAM: Principal objeto Natural. É a partir dele que são chamados os outros componentes.

- SUBPROGRAM (N): Objeto chamado por outro, com passagem de parâmetros, que permite o retorno de dados ao objeto chamador.

- SUBROUTINE: Uma sub-rotina é um pedaço de código geralmente executado mais de uma vez ou de diversas partes de um programa. Pode ser interno, definido dentro de um programa ou fora dele (externo). Quando se escolhe a segunda opção, o tipo de objeto que conterá as sub-rotinas é o SUBROUTINE.

OBS: As letras sublinhadas nos nomes dos objetos podem ser utilizadas logo após o comando EDIT, evitando a digitação completa do tipo de objeto.

## 10 – Comandos Gerais

### Set Control

Utilizado para executar comandos de controle de terminal de dentro de um programa.  
Exemplo:

SET CONTROL 'M-2' → Exibe mensagens de erro no rodapé da tela  
SET CONTROL 'N' → Simula um ENTER

### Set Key

Comando utilizado para ativar/desativar as teclas de função.

```
[ON ]
SET KEY [OFF]
[ALL]

[programa]
SET KEY PFn = [ON      ]  [NAMED [OFF      ]
                   [OFF      ]           [operando]
                   [HELP     ]]
```

SET KEY ALL → ativa todas as PF's para uso.

SET KEY PF2=PGM → ativa PF2 para executar o programa especificado.

SET KEY OFF → desativa todas as PF's.

SET KEY ON → reativa todas as PF's.

SET KEY PF4=HELP → ativa a tecla F4 para servir de HELP de campo.

SET KEY PF7='PRGP0010' NAMED 'MENU' → ativa a tecla PF7 para executar o programa PRGP0010 e coloca a string MENU na régua de PF's.

```
DEFINE DATA LOCAL
1 #PF4 (A56)
1 #FCT (A8)
END-DEFINE
...
SET KEY PF1
PF2
PF3 = `MENU`
PF4 = #PF4
PF5 = `LISTA EMPREGADO`
PF6 = 'XXXP0070' NAMED 'IMPRIME'
...
...
```

## Mask

Permite que o conteúdo das posições de um campo seja checado. MASK é justificado à esquerda. Os sinais de \* ou %, inseridos na definição do MASK indicam que qualquer número de posições podem ser ignoradas na verificação. O . (ponto), , (vírgula) ou \_ (underline), significam que esta posição não será verificada. A checagem pode ser feita por constantes, variáveis, tipos, datas e intervalos.

Exemplo de constante:

IF #X = MASK(..'VA') → as posições 3 e 4 do campo #X serão verificadas para o valor 'VA'.
---

TIPO	TESTE DE VERIFICAÇÃO
A	Alfabético minúsculo ou maiúsculo
N	Numérico
C	Alfanumérico
H	Hexadecimal
L	Alfabético minúsculo
U	Alfabético maiúsculo

Exemplo:

IF #X = MASK(AA..NN) → as posições 1 e 2 serão checadas para valores alfabéticos, e as posições 5 e 6 serão checados para valores numéricos.
--

DATA	TESTE DE VERIFICAÇÃO
YYYY	Validação de ano com 4 posições
YY	Validação de ano com 2 posições
MM	Valida mês
DD	Valida dia

Exemplo:

IF #X = MASK(MM' / DD) → as posições 1 e 2 serão checadas para Mês, a posição 3 para '/' e as posições 4 e 5 para validar a data no mês.
--

Exemplo de checagem de variáveis e constantes:

Permite a validação de dados constantes ou variáveis. "X" representa a posição dentro do campo. Exemplos:

#X (A5) init<'ABC E'> #Y (A4) init<'23C1'> #Z (A4) INIT<'0230'>
#X ← MASK(..'E') #Y ← MASK(NN.N) #Z ← MASK(..DD) #Z ← MASK(MMDD) #Z ← MASK(AA.N)

O MASK pode também ser definido numa variável auxiliar, tornando o processamento mais flexível. Exemplo:

MOVE '*'R**' TO #M IF #CAMPO NE MASK #M
--

A Verificação feita na variável #CAMPO será verdadeira se for encontrada a letra “R” em qualquer ponto desta, uma vez que o sinal “\*” ignora qualquer número de caracteres.

## Format

A instrução ‘FORMAT’ especifica valores para os parâmetros globais que se aplicam à formatação de relatórios e mapas.

```
FORMAT {parâmetro}
```

```
FORMAT AL=7 /* Exibição de 7 bytes do campo
               LS=15 /* Torna um página com 15 linhas
               PS=50 /* Permite exibição em 50 colunas
```

## 11 – Comandos de Entrada/Saída

### Input

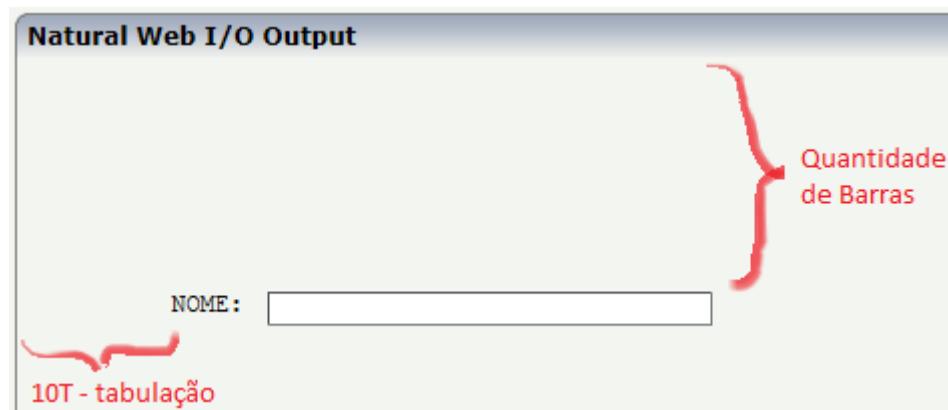
Instrução para obter dados para o programa ou para mostrar os dados de programa.

```
+ * >Natural Source Header 000000..
@ define data local
 1 nome (a30)
 1 idade (i2)
end-define

@ /* as barras apos o input significam a
   /* quantidade de linhas a serem puladas
   /* 10T => espaço da margem a esquerda
      input ///// 10T 'NOME: ' nome

      write 'Nome: ' nome

END
```



Parâmetros:

AD=[r][a][i/o][c][f]

r → apresentação: Blink, Default, Intensified, Non-display

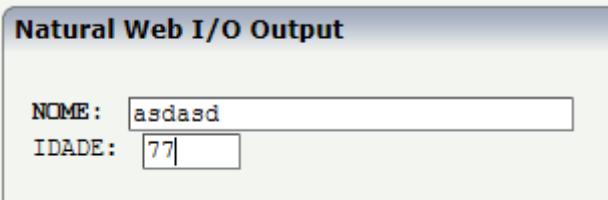
a → alinhamento: Left, Right

i/o → modo: Input (A), Modifiable, Output

c → tratamento de caracteres: I minúsculas para maiúsculas, W aceita minúsculas

f → caracter de preenchimento (filler): definido pelo usuário.

```
+ * >Natural Source Header 000000..  
-define data local  
1 nome (a30)  
1 idade (i2)  
end-define  
  
/* O i apos a string o deixa em negrito  
/* A barra pula uma linha  
input 'NOME: ' (i) nome / 'IDADE: ' idade  
  
write 'NOME: ' nome 'IDADE: ' idade  
  
END
```



```
+ * >Natural Source Header 000000..  
define data local  
1 nome (a30)  
1 idade (i2)  
end-define  
  
/* n => non-display. Indicado p/ senhas  
/* t => transforma para maiusculas  
input (ad=nt) 'NOME: ' nome  
  
write 'NOME: ' nome  
  
END
```

#### Natural Web I/O Output

NOME:

#### Natural Web I/O Output

Page 1

23-05-15 10:56:11

NOME: ASDASDAS

```

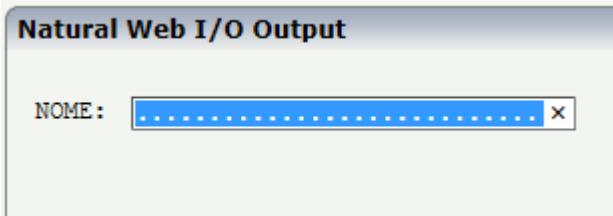
+ * >Natural Source Header 000000[]
define data local
1 nome (a30)
1 idade (i2)
end-define

/* '.' é um caracter de preenchimento
/* definido pelo usuario
input (ad='.') 'NOME: ' nome

write 'NOME: ' nome

END

```



```

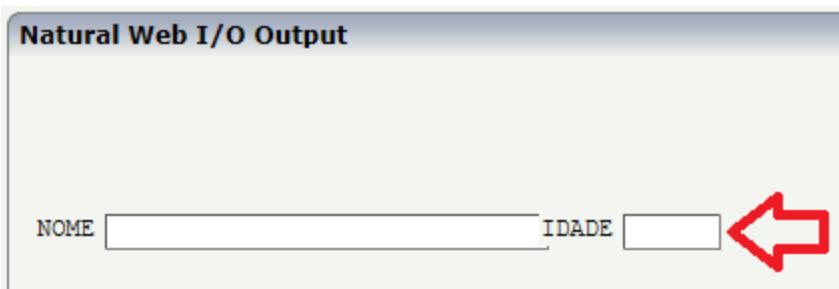
+ * >Natural Source Header 000000[]
define data local
1 nome (a30)
1 idade (i2)
end-define

/* mark 2 => inicia o programa com o
/* cursor no segundo input (idade)
input mark 2 /// nome idade

write 'NOME: ' nome 'IDADE: ' idade

END

```



INPUT com mensagem na linha de mensagem:

INPUT WITH TEXT 'Digite a agência. ' 'AGENCIA:' #AGENCIA (AD=MI')\_

INPUT com utilização de mapa externo:

INPUT USING MAP 'XXXM0001'

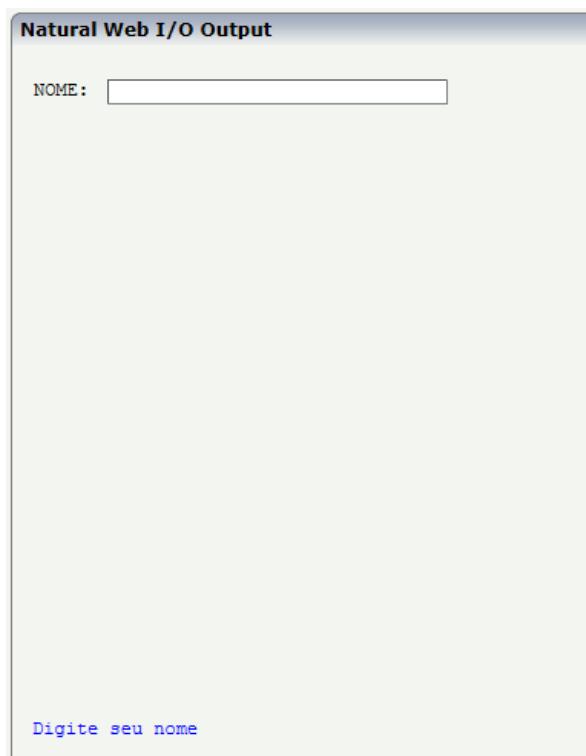
Apresentando mensagem na linha de mensagem:

INPUT WITH TEXT #MSG USING MAP 'XXXM0014'

Marcando um determinado campo:

INPUT WITH TEXT #MSG MARK #CODIGO USING MAP 'XXXM0002'

```
+ * >Natural Source Header 000000..  
define data local  
1 nome (a30)  
1 idade (i2)  
end-define  
  
input with text 'Digite seu nome' 'NOME: ' nome  
  
write 'NOME: ' nome  
  
END
```



## Reinput

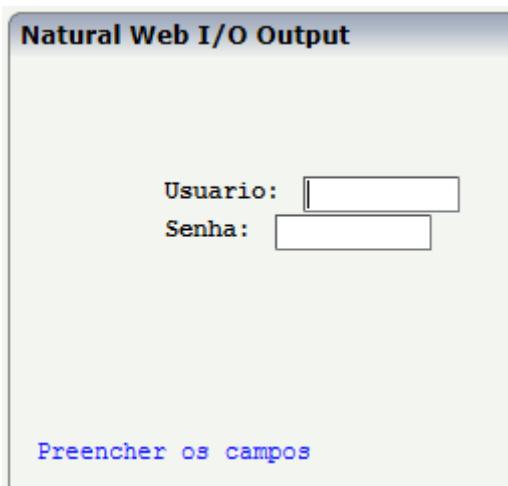
Instrução para voltar a executar a última instrução INPUT.

REINPUT [FULL] (mensagem) MARK \*(campo cursor) [alarm]

Entre uma instrução INPUT e o REINPUT correspondente, não pode haver nenhuma instrução WRITE ou DISPLAY, pois para a execução do REINPUT, a última comunicação com a tela tem obrigatoriamente que ter sido com um INPUT.

A cláusula FULL atualiza todos os campos da tela que foram alterados entre o INPUT e o REINPUT. Se não for especificada, não atualiza a tela com o novo conteúdo, mesmo que haja alteração dos mesmos.

```
+ * >Natural Source Header 000000..  
define data local  
1 usuario (a10)  
1 senha (a10)  
end-define  
  
input (ad=m) with text 'Preencher os campos' //  
    10T 'Usuario: ' (i) usuario /  
    10T 'Senha: ' (i) senha (ad=n) /  
  
if usuario = ''  
    reinput full 'Informe o usuario.' mark *usuario alarm  
endif  
  
if senha = ''  
    reinput full 'Informe a senha.' mark *senha alarm  
endif  
  
END
```



### Natural Web I/O Output

Usuario:   
Senha:

[Informe a senha.](#)

Utilizando um mapa para a entrada de dados, o programa anterior ficaria da seguinte maneira:

```
DEFINE DATA LOCAL
1 #USUARIO      (A8)
1 #SENHA        (A8)
END-DEFINE

INPUT WITH TEXT 'Preencher os campos.' USING MAP 'XXXM0020'
IF #USUARIO = ''
  REINPUT FULL 'Informe código de usuário.' MARK *#USUARIO ALARM
END-IF
IF #SENHA = ''
  REINPUT FULL 'Informe a senha.' MARK *#SENHA ALARM
END-IF
END
```

## Display

Instrução para imprimir dados do programa, em formato de colunas. A instrução DISPLAY automaticamente inclui cabeçalhos, com o mesmo nome da variável, caso não seja informado um cabeçalho diferente.

NOTITLE → suprime a linha de cabeçalho da página.

Rep → número do relatório. O Natural pode controlar até 32 relatórios simultâneos, numerados de 0 a 31.

```
* >Natural Source Header 000000
define data local
 1 vw VIEW OF VEICULOS
   2 PLACA (A7)
   2 NOME (A30)
end-define

read vw
  display notitle vw
end-read

END
```

### Natural Web I/O Output

PLACA	NOME
AA 0001	CATIA SARAIVA
AA 0009	
AM 9594	JOSE LUDGERO SOUZA F2T215440
BC 1032	JOSE LUDGERO SOUZA
AW 0732	JOSE LUDGERO SOUZA
AW 0772	JOSE LUDGERO SOUZA
AX 0957	JOSE LUDGERO SOUZA
AM 0425	JANIE G DOMINGUES F2T212511
AM 4028	JANIE G DOMINGUES F2T213044
AA 9664	JANIE G DOMINGUES F2T281935

## Natural Web I/O Output

Page 1 23-05-15 13:24:34

VW

PLACA	NOME
AA 0001	CATIA SARAIVA
AA 0009	
AM 9594	JOSE LUDGERO SOUZA F2T215440
BC 1032	JOSE LUDGERO SOUZA

Sem o "notitle" a pagina, a data e a hora sao visiveis.

```
* * >Natural Source Header 000000
define data local
1 vw VIEW OF VEICULOS
 2 PLACA (A7)
 2 NOME (A30)
end-define

/* sf => espacamento entre as colunas (de 1 a 30)
/* hc => posicao cabecalho das colunas (center, left, right)
/* uc => caracter p/ sublinhar cabecalho (default: "-")
/* fc => caracter preenchimento cabecalho (default: branco)
read vw
  display (sf=10 hc=c uc=_ fc=+) vw
end-read

END
```

## Natural Web I/O Output

Page 1 23-05-15 13:38:51

VW

+PLACA+	+++++NOME+++++
AA 0001	CATIA SARAIVA
AA 0009	
AM 9594	JOSE LUDGERO SOUZA F2T215440
BC 1032	JOSE LUDGERO SOUZA
AW 0732	JOSE LUDGERO SOUZA
AW 0772	JOSE LUDGERO SOUZA
AX 0957	JOSE LUDGERO SOUZA
AM 0425	JANIE G DOMINGUES F2T212511
AM 4028	JANIE G DOMINGUES F2T213044
AA 9664	JANIE G DOMINGUES F2T281935

```
+ * >Natural Source Header 000000..  
define data local  
1 vw VIEW OF VEICULOS  
2 PLACA (A7)  
2 NOME (A30)  
end-define  
  
/* podemos renomear as colunas  
read vw  
    display 'PLACA' placa 'NOME' nome  
end-read  
  
END
```

#### Natural Web I/O Output

Page 1

23-05-15 13:46:52

PLACA	NOME
AA 0001	CATIA SARAIVA
AA 0009	
AM 9594	JOSE LUDGERO SOUZA F2T215440
BC 1032	JOSE LUDGERO SOUZA
AW 0732	JOSE LUDGERO SOUZA
AW 0772	JOSE LUDGERO SOUZA
AX 0957	JOSE LUDGERO SOUZA

```

+ * >Natural Source Header 000000..
@define data local
@ 1 vw VIEW OF VEICULOS
  2 BAIRRO (A10)
  2 PLACA (A7)
  2 MARCA (N6.0)
  2 NOME (A30)
end-define

@ /* AL => tamanho maximo p/ exibicao alfanumerico
@ /* NL => tamanho maximo p/ exibicao numerico
@ /* SF => espacamento entre colunas
@ read vw
  display (sf=5) placa nome(al=10) marca(nl=3)
end-read

END

```

Natural Web I/O Output		
Page	1	23-05-15 14:01:02
PLACA	NOME	MARCA
-----	-----	-----
AA 0001	CATIA SARA	502
AA 0009		901
AM 9594	JOSE LUDGE	099
BC 1032	JOSE LUDGE	699
AW 0732	JOSE LUDGE	699
AW 0772	JOSE LUDGE	699
AX 0957	JOSE LUDGE	699

## Write

A instrução 'WRITE' causa a impressão de uma ou mais linhas com formatação livre, e com as seguintes características:

- Se uma linha lógica excede a linha física, ela será dividida em duas, mas nenhum campo será truncado.
- Não há geração de cabeçalhos, e o tamanho de cada elemento determina as posições de impressão.
- Faixas de valores de campos múltiplos e grupos periódicos são dispostos horizontalmente na mesma linha.

```

1.      WRITE NOTITLE `=``#PRIMEIRO-NOME`=`#SOBRENOME`=`#ULTIMO-NOME //  

        `L O C A L I Z A Ç Ã O` /  

        `CIDADE:` #CIDADE /  

        `PAÍS:` #PAIS //  

        ...  

2.      WRITE NOTITLE 5X #NOME 50T #PROFISSAO  

        ...  

3.      WRITE / `MÉDIA CIDADE:` C*SALARIO(1) AVER(SALARIO(1)) //  

        ...  

4.      WRITE / `MÉDIA SALÁRIO` C*SALARIO(1) AVER(SALARIO(1)) //  

        ...  

5.      WRITE NOTITLE (AL=16 NL=8)  

        `=``#CODIGO`=`#NOME`=`#TELEFONE (AL=10 EM=XXX-XXXXXX)  

+ * >Natural Source Header 000000..  

@define data local  

@ 1 vw VIEW OF VEICULOS  

  2 BAIRRO (A10)  

  2 PLACA (A7)  

  2 MARCA (N6.0)  

  2 NOME (A30)  

end-define  

/* 10x => espacamento a esquerda  

@read vw  

  write 10x 'PLACA: ' placa 'NOME: ' nome 'BAIRRO: ' bairro  

end-read  

END

```

### Natural Web I/O Output

Page	1	23-05-15 14:24:49
PLACA: AA 0001 NOME: CATIA SARAIVA		BAIRRO: FLORES
PLACA: AA 0009 NOME:		BAIRRO: CACHOEIRIN
PLACA: AM 9594 NOME: JOSE LUDGERO SOUZA F2T215440		BAIRRO: CENTRO
PLACA: BC 1032 NOME: JOSE LUDGERO SOUZA		BAIRRO: ALVORADA 2
PLACA: AW 0732 NOME: JOSE LUDGERO SOUZA		BAIRRO: HILEIA 1
PLACA: AW 0772 NOME: JOSE LUDGERO SOUZA		BAIRRO: ALVORADA 2

## Write Title/Trailer

'WRITE TITLE' define um título para um relatório, em substituição ao título padrão do Natural.

'WRITE TRAILER' define uma ou mais linhas a serem impressas no rodapé de um relatório.

```
+ * >Natural Source Header 000000
define data local
 1 vw VIEW OF VEICULOS
    2 BAIRRO (A10)
    2 PLACA (A7)
    2 MARCA (N6.0)
    2 NOME (A30)
end-define

/* write title left => define um titulo
/* no canto esquerdo
write title left 'NOME E PLACA'

skip 1 /* gera linha em branco

/* write trailer left => linhas impressas
/* no rodapé de um relatorio
write trailer left 'Página' *page-number

skip 1 /* gera linha em branco

read vw
  display placanome
end-read

END
```

## Natural Web I/O Output

NOME E PLACA 

PLACA NOME

---

AA 0001 CATIA SARAIVA  
AA 0009  
AM 9594 JOSE LUDGERO SOUZA F2T215440  
BC 1032 JOSE LUDGERO SOUZA  
AW 0732 JOSE LUDGERO SOUZA  
AW 0772 JOSE LUDGERO SOUZA

AA 0044

AA 0046

AA 0050

AX 0859 JANIE G DOMINGUES F2T217342

AM 0074 JANIE G DOMINGUES F2T217320

AM 8890 JOSE LUDGERO SOUZA F2T216030

AA 0056

AA 0057

Página 1 

## Newpage

Este comando provoca um avanço de página do relatório.

```
NEWPAGE (rep) IF LESS THAN operand1 LINES LEFT  
WHEN
```

O 'NEWPAGE' avança a página e provoca a impressão do cabeçalho na página seguinte.

IF LESS THAN: se especificada a cláusula, ocorrerá quebra de página somente quando houver menos linhas do que o especificado no parâmetro.

---

...

```
NEWPAGE (1) IF LESS THAN 5 LEFT
```

...

---

## Eject

Este comando provoca um avanço de página do relatório, sem gerar cabeçalho na próxima página.

```
EJECT (rep) IF LESS THAN operand1 LINES LEFT  
WHEN
```

IF LESS THAN: se especificada a cláusula, ocorrerá quebra de página somente quando houver menos linhas do que o especificado no parâmetro.

---

...

```
EJECT (1) IF LESS THAN 5 LEFT
```

...

---

## Skip

Gera linhas em branco em um relatório.

```
* >Natural Source Header 000000..  
define data local  
1 valor1 (n3) init <3>  
1 valor2 (n3) init <5>  
end-define  
  
write 'Valor:' valor1  
skip 1 /* gera linha em branco  
write 'Valor:' valor2  
  
END
```

### Natural Web I/O Output

Page 1

Valor: 3

Valor: 5

## 12 – Operações Aritméticas

### Add

Comando utilizado para efetuar a operação de soma.

```
+ * >Natural Source Header 000000..  
define data local  
1 a (n2)  
1 b (n3.2) /* 3 antes da virgula e 2 depois  
1 c (n2)  
1 data (d)  
end-define  
  
/* T0 => os valores sao adicionados a  
/* variavel A  
add +5 -2 -1 to a  
write a  
  
/* GIVING => o resultado da soma dos valores  
/* é armazenado na variavel B  
add 0.231 3.6 giving b  
write b  
  
/* ROUNDED => o resultado é arredondado  
/* ajustando-se ao tamanho de C  
add rounded 2.9 3.8 giving c  
write c  
  
/* Atribui a data atual para a variavel data  
/* em seguida adiciona 7  
move *datx to data  
add 7 to data  
write data  
  
END
```

#### Natural Web I/O Output

Page	1	23-05-15 15:48:36
	2	
	3.83	
	7	
	23-05-22	

## Subtract

Comando utilizado para efetuar a operação de subtração.

```
+ * >Natural Source Header 000000..  
define data local  
1 a (n2.2) init <50>  
1 b (p2.2)  
1 c (p2.1) init <2.4>  
end-define  
  
/* A - 6 = 44  
subtract 6 from a  
write 'A: ' a  
  
/* 11 - 6 = 5  
subtract 6 from 11 giving a  
write 'A: ' a  
  
/* a(5) - (3 + 4 = 7) => -2  
/* -2 é colocado dentro de b  
subtract 3 4 from a giving b  
write 'A:' a 'B: ' b  
  
/* a(5) - (-7) => 12  
/* 12 é colocado dentro do b  
subtract -3 -4 from a giving b  
write 'A:' a 'B: ' b  
  
/* 2.4 - 2.06 = 0.34 => 0.3  
subtract rounded 2.06 from c  
write 'C: ' c  
  
END
```

### Natural Web I/O Output

Page	1	23-05-15 16:25:04
A:	44.00	
A:	5.00	
A:	5.00	B: -2.00
A:	5.00	B: 12.00
C:	0.3	

## Multiply

Comando utilizado para efetuar a operação de multiplicação.

```
* >Natural Source Header 000000
define data local
 1 a (p2.2) init <2>
 1 b (p2.2)
 1 c (i2)
end-define

/* a(2) * 3 = 6
/* 6 é armazenado em a
multiply a by 3
write 'A: ' a

/* 3 * a(6) = 18
/* 18 é armazenado em b
multiply 3 by a giving b
write 'B: ' b

/* 3 * 3.6 = 10.8
/* Arredonda de acordo com o campo receptor
/* como "c" aceita apenas inteiro
/* o valor fica em c fica 11
multiply rounded 3 by 3.6 giving c
write 'C: ' c

END
```

### Natural Web I/O Output

Page 1

23-05-16 10:32:53

A: 6.00  
B: 18.00  
C: 11

## Divide

Comando utilizado para efetuar a operação de divisão.

```
* >Natural Source Header 000000..  
/* 0 # é somente para diferenciar a  
/* variavel dos outros comandos  
define data local  
1 #A (n5) init <20>  
1 #B (n5.2)  
1 #C (n3.2)  
1 #D (n1)  
1 #E (n1) init <3>  
1 #F (n1)  
end-define  
  
/* A(20) / 5 = 4 | 4 é armazenado em A  
divide 5 into #A  
write 'A: ' #A  
  
/* A(4) / 5 = 0.80 | 0.80 é armazenado em B  
divide 5 into #A giving #B  
write 'B: ' #B  
  
/* 3.1 / 3 = 1.03 | 1.03 é armazenado em C  
divide 3 into 3.1 giving #C  
write 'C: ' #C  
  
/* 3.1 / 3 = 1.03 | 1.03 é armazenado em D  
/* porem, como D tem o tipo n1  
/* ele aceita apenas 1 algarismo  
/* entao pega apenas o 1  
divide 3 into 3.1 giving #D  
write 'D: ' #D  
  
/* E(3) / 2 = 1.5 | 1 é armazenado em E  
/* pois E tem o tipo n1 que aceita apenas  
/* 1 algarismo  
/* remainder => resto da divisao é armazenado  
/* em F  
divide 2 into #E remainder #F  
write 'E: ' #E 'F: ' #F  
END
```

## Natural Web I/O Output

Page	1	23-05-16 11:00:26
A:	4	
B:	0.80	
C:	1.03	
D:	1	
E:	1 F:	1

## Compute

A instrução ‘COMPUTE’ executa operações aritméticas e permite a utilização de funções do Natural.

```
* >Natural Source Header 000000..  
/* 0 # é somente para diferenciar a  
/* variavel dos outros comandos  
define data local  
1 #A (n3)  
1 #B (n3)  
1 #C (n3.4)  
1 #D (n3.4) init <31.3567>  
end-define  
  
/* compute => executa operacoes aritmeticas  
/* e permite funcoes do Natural  
  
/* o resultado da expressao é 7  
/* 7 é armazenado em A  
compute #A = 3 * 2 + 4 / 2 - 1  
write 'A: ' #A  
  
/* 3.4 * 2.7 = 9.18  
/* 9 é armazenado em B  
compute rounded #B = 3.4 * 2.7  
write 'B: ' #B  
  
/* o comando compute é opcional desde  
/* que tenha o := (ponto igual)  
/* sqrt é a funcao de raiz quadrada  
/* raiz quadrada de 31.3567 é 5.5997  
#C := sqrt(#D)  
write 'C: ' #C  
  
END
```

## Natural Web I/O Output

Page 1

23-05-16 11:19:23

A: 7

B: 9

C: 5.5997

## 13 - Manipulação de Campos

### Compress

A instrução 'COMPRESS' é utilizada para combinar o conteúdo de dois ou mais operandos para um campo.

```
* >Natural Source Header 000000..  
define data local  
1 #NOME (a20)  
1 #SOBRENOME (a20)  
1 #NOMECOMPLETO (a20)  
1 #DIA (i2)  
1 #MES (i2)  
1 #ANO (i4)  
1 #DATA (a20)  
end-define  
  
#NOME := 'Joca'  
#SOBRENOME := 'Tigre'  
#DIA := 11  
#MES := 05  
#ANO := 2023  
  
/* compress => combinar conteúdo p/ um campo  
  
/* combina nome e sobrenome p/ nomecompleto  
compress #NOME #SOBRENOME into #NOMECOMPLETO  
write 'Nome Completo: ' #NOMECOMPLETO  
  
/* leaving no => não deixa espaço em branco  
/* entre os dados  
compress #DIA #MES #ANO into #DATA leaving no  
write 'Data: ' #DATA  
  
/* with delimiter => especifica um caractere p/  
/* servir de separador entre os campos  
compress #DIA #MES #ANO into #DATA with delimiter '/'  
write 'Data: ' #DATA  
  
END
```

## Natural Web I/O Output

Page 1

23-05-16 11:46:59

Nome Completo: Joca Tigre  
Data: 1152023  
Data: 11/5/2023

## Move

A instrução ‘MOVE’ copia o conteúdo de um campo para outro ou atribui um valor a uma variável.

```
* >Natural Source Header 000000..  
define data local  
1 nome1 (a20) init <'Joca'>  
1 nome2 (a20)  
1 num1 (n2)  
1 pessoas  
    2 primeironome (a20) init <'Jose'>  
    2 idade (n2) init <33>  
1 pessoas2  
    2 primeironome (a20)  
    2 idade (n2)  
end-define  
  
/* move => copia o conteúdo p/ outro ou  
/* atribui um valor a uma variável  
move nome1 to nome2  
write 'Nome2:' nome2  
  
/* rounded arredonda de acordo com o  
/* tamanho do receptor  
move rounded 1.97 to num1  
write 'Numero1:' num1  
  
write '-----'  
  
/* movimenta todas as variáveis de uma estrutura  
/* p/ outra desde que tenham o mesmo nome  
move by name pessoas to pessoas2  
write pessoas2  
  
END
```

### Natural Web I/O Output

```
Page      1                               23-05-16 13:33:06
Nome2: Joca
Numero1:  2
-----
Jose          33
```

```
* >Natural Source Header 000000
define data local
1 pessoas1
  2 primeironome1 (a20) init <'Jose'>
  2 idade1 (n2) init <33>
1 pessoas2
  2 primeironome2 (a20)
  2 idade2 (n2)
end-define

/* move by position => movimenta todos os campos
/* de um estrutura p/ outra, podendo ter nomes
/* diferentes
/* Regras:
/* - nº de campos nos 2 grupos precisa ser igual
/* - niveis dos campos precisam ser iguais
/* - se houver array, precisam ter as mesmas dimensoes
move by position pessoas1 to pessoas2

display pessoas2

END
```

### Natural Web I/O Output

```
Page      1                               23-05-16 13:41:24
PESSOAS2
PRIMEIRONOME2      IDADE2
-----
Jose          33
```

```
* >Natural Source Header 000000..  
define data local  
1 nome (a10)  
1 data (a10)  
end-define  
  
/* move all => preenche uma variavel string  
/* com o parametro especificado  
move all '*' to nome  
write 'Nome:' nome  
  
skip 1 /* pula 1 linha  
  
/* edited => movimenta um campo com uma mascara  
/* para outro campo  
move edited *datx(em=dd/mm/yyyy) to data  
write 'Data:' data  
  
END
```

#### Natural Web I/O Output

Page	1	23-05-16 13:53:48
Nome: *****		
Data: 16/05/2023		

## Substring

```
* >Natural Source Header 000000..  
define data local  
1 nome1 (a10) init <'Joca Tigre'>  
1 nome2 (a10)  
end-define  
  
/* substring => move a porcao de um campo  
/* p/ outro campo  
/* move substring(varivael1, i, q) to variavel2  
/* i => posicao inicial | q => quantidade bytes  
move substring(nome1,6,5) to nome2  
write nome2  
  
END
```

### Natural Web I/O Output

Page 1

23-05-16 14:10:20

Tigre

## Reset

A instrução ‘RESET’ estabelece valores nulos, de acordo com o tipo de variável, ou os valores especificados inicialmente.

A cláusula INITIAL restabelece o valor original definido no DEFINE DATA pela cláusula INIT.

DEFAULT DE INICIALIZAÇÃO		
A	Alfanumérico	Branco
B	Binário	0 (zero)
N	Numérico	0 (zero)
P	Numérico Compactado	0 (zero)
I	Inteiro	0 (zero)
L	Lógico	False
F	Ponto Flutuante	0 (zero)
D	Data	0 (zero)
T	Hora	00:00:00

```
* >Natural Source Header 000000..  
define data local  
1 nome (a10)  
1 idade (n3) init <231>  
1 numero (n3)  
1 logico (l)  
1 data (a10)  
end-define  
  
nome := 'Joca'  
numero := 34  
logico := true  
move edited *datx(em=dd/mm/yyyy) to data  
  
write 'Nome:' nome / 'Idade:' idade / 'Numero:' numero /  
      'Logico:' logico / 'Data:' data  
  
skip 1  
  
/* reset estabelece valores nulos de acordo com  
/* o tipo da variavel ou dos valores especificados  
/* inicialmente  
reset nome idade numero logico data  
write 'Nome:' nome / 'Idade:' idade / 'Numero:' numero /  
      'Logico:' logico / 'Data:' data  
  
skip 1  
  
/* reset initial restabelece o valor original  
/* definido no define data pelo init  
reset initial idade  
write 'Idade inicial:' idade  
  
END
```

## Natural Web I/O Output

Page 1 23-05-16 14:54:51

Nome: Joca  
Idade: 231  
Numero: 34  
Logico: X  
Data: 16/05/2023

Nome:  
Idade: 0  
Numero: 0  
Logico:  
Data:

Idade inicial: 231

## Separate

A instrução ‘SEPARATE’ é usada para separar o conteúdo de um campo para dois ou mais campos (alfanuméricos, array ou grupos).

```
* >Natural Source Header 000000
define data local
1 a (a10)
1 b (a10)
1 c (a10)
1 n1 (a10)
1 n2 (a10)
1 array (a3/3)
end-define

/* separate é usado p/ separar o conteúdo de um campo
/* p/ 2 ou mais campos (alfanumericos, array ou grupos)
separate 'a,b,c' into a b c
write 'A:' a 'B:' b 'C:' c

skip 1

/* delimiter faz a separação a partir do caractere
/* indicado
/* se omitido, o delimitador é qualquer caractere
/* menor que a letra "a" minuscula
separate '123.4,234.1' into n1 n2 with delimiter ','
write 'n1:' n1 'n2:' n2

skip 1

separate '1 2 3' into array(*)
write 'Array com 3 posicoes:' array(*)

END
```

```
Page      1

A: a          B: b          C: c

n1: 123.4      n2: 234.1

Array com 3 posicoes: 1   2   3
```

## Examine

A instrução 'EXAMINE' pesquisa um campo (alfanumérico ou array) por uma opção e altera, deleta ou conta o número de ocorrências de um campo.

```
* >Natural Source Header 000000..  
define data local  
1 letras1 (a10) init <'aaabbcccd'>  
1 letras2 (a10) init <'aabbccdde'>  
1 letras3 (a10) init <'aaabbccc'>  
1 letras4 (a10) init <'aaabbccc'>  
1 array1 (a2/4) init (1) <'aa'> (2) <'bb'> (3) <'cc'> (4) <'dd'>  
1 i (i1)  
1 j (i1)  
1 z (i1)  
end-define  
  
/* examine pesquisa um campo (alfa ou array) por uma opcao  
/* e altera, deleta ou conta  
  
/* giving position obtem a partir de qual byte o 'bbb' inicia  
/* se o 'bbb' nao for encontrado, retorna 0  
examine letras1 for 'bbb' giving position i  
write i  
  
/* giving index obtem o indice em que o 'dd' ocorre  
/* caso nao seja encontrado, retorna 0  
examine array1(*) for 'dd' giving index j  
write j  
  
/* giving number obtem a quantidade de vezes que  
/* ocorreu o 'bb', se n for encontrado retorna 0  
examine array1(*) for 'bb' giving number z  
write z  
  
/* replace troca o 'cc' pelo 'xy'  
examine letras2 for 'cc' replace 'XY'  
write letras2  
  
/* replace first troca apenas a primeira  
/* ocorrencia  
examine letras3 for 'b' replace first 'X'  
write letras3  
  
/* delete apaga a ocorrencia especificada  
/* se first for especificado, apaga apenas  
/* a primeira ocorrencia  
examine letras4 for 'bbb' delete  
write letras4  
  
END
```

```
4  
4  
1  
aabXYddee  
aaaXbbccc  
aaaccc
```

## 14 - Instruções Condicionais

### If

A instrução 'IF' é usada para controlar a execução de grupos de instruções com base em uma condição lógica.

```
+ * >Natural Source Header 000000..  
define data local  
1 pessoa (a20)  
1 salario (n5)  
end-define  
  
input pessoa salario  
  
if salario >= 4000  
    write notitle 'Salario do 'pessoa' é maior/igual que 4000.'  
else  
    write notitle 'Salario do 'pessoa' é menor que 4000.'  
endif  
  
END
```

### Natural Web I/O Output

Salario do Joca	é menor que 4000.
-----------------	-------------------

## Decide On / Decide For

A instrução ‘DECIDE’ é uma estrutura de múltipla escolha (switch).

```
+ * >Natural Source Header 000000..  
define data local  
1 valor (n3)  
end-define  
  
input valor  
  
/* decide é semelhante ao switch  
/* decide on => usado p/ executar uma ou mais  
/* ações, dependendo de um valor  
/* os valores podem ser específicos ou intervalos  
/* none é executado se nenhum valor for encontrado  
/* o none é obrigatório  
/* ignore é usado quando n se deseja executar nada  
decide on first value of valor  
    value 1, 2, 3, 4  
        write 'Numeros de 1 a 4'  
    value 5:10  
        write 'Numeros de 5 a 10'  
    none  
        ignore  
end-decide  
  
END
```

### Natural Web I/O Output

Page 1

23-05-17 10:33:14

Numeros de 5 a 10

```
+ * >Natural Source Header 000000..  
define data local  
1 valor (n3)  
end-define  
  
input valor  
  
/* o decide for first condition executa uma ou mais  
/* ações, dependendo de multiplas condicoes  
decide for first condition  
when valor >= 1 and valor <= 10  
    write 'Valor de 1 a 10'  
when valor > 10 and valor < 20  
    write 'Valor maior que 10 e menor que 20'  
when none  
    ignore  
end-decide  
  
END
```

#### Natural Web I/O Output

Page 1

23-05-17 11:06:51

Valor maior que 10 e menor que 20

FIRST → para após a primeira condição verdadeira.

EVERY → executa todas as condições descritas.

```
+ * >Natural Source Header 000000..  
⊖ define data local  
  1 valor (n3)  
end-define  
  
input valor  
  
⊖ /* every => executa todas as instrucoes  
   /* que satisfaçam as condicoes  
   /* se o valor for 5, a primeira e a  
   /* segunda condição serao executadas  
⊖ decide on every value of valor  
⊖   value 1:6  
     write 'Valor de 1 a 6'  
⊖   value 4:10  
     write 'Valor de 7 a 10'  
⊖   none  
     write 'Valores acima de 10'  
end-decide  
  
END
```

#### Natural Web I/O Output

Page	1	23-05-17 11:38:21
Valor de 1 a 6		
Valor de 7 a 10		

ANY → é executado se alguma das condições forem verdadeiras.

```
+ * >Natural Source Header 000000..  
define data local  
1 valor (n3)  
end-define  
  
input valor  
  
/* as instrucoes de dentro do any sao executadas  
/* se alguma das condicoes forem verdadeiras  
/* nesse exemplo a condicao 1:6 é executada  
/* juntamente com o any  
decide on first value of valor  
  value 1:6  
    write 'Valor de 1 a 6'  
  value 4:10  
    write 'Valor de 7 a 10'  
  any value  
    write 'any'  
  none  
    write 'Valores acima de 10'  
end-decide  
  
END
```

#### Natural Web I/O Output

Page	1	23-05-17 13:06:52
Valor de 1 a 6		
any		

```

+ * >Natural Source Header 000000..
define data local
1 valor (n3)
end-define

input valor

/* as instrucoes de dentro do any sao executadas
/* se alguma das condicoes forem verdadeiras
/* nesse exemplo a condicao 1:6 e 4:10 sao executadas
/* juntamente com o any, devido ao comando every
decide on every value of valor
value 1:6
    write 'Valor de 1 a 6'
value 4:10
    write 'Valor de 7 a 10'
any value
    write 'any'
none
    write 'Valores acima de 10'
end-decide

END

```

#### Natural Web I/O Output

Page	1	23-05-17 13:12:17
Valor de 1 a 6		
Valor de 7 a 10		
any		

ALL → é executado se todas as condições forem verdadeiras. (A opção EVERY tem de ser especificada).

```
+ * >Natural Source Header 000000..  
define data local  
1 valor (n3)  
end-define  
  
input valor  
  
write 'Valor digitado:' valor  
  
skip 1  
  
/* all => as instrucoes de dentro do all  
/* sao executadas somente se todas as  
/* condicoes forem verdadeiras  
/* é obrigado o uso do every  
decide on every value of valor  
    value 1:6  
        write 'Valor de 1 a 6'  
    value 4:10  
        write 'Valor de 7 a 10'  
    value 6:8  
        write 'Valor de 5 a 8'  
    all value  
        write 'all'  
    none  
        write 'Valores acima de 10'  
end-decide  
  
END
```

#### Natural Web I/O Output

Page	1	23-05-17 13:22:35
Valor digitado: 6		
Valor de 1 a 6		
Valor de 7 a 10		
Valor de 5 a 8		
all		

## 15 – Controle de processamento (loops)

### For

Instrução utilizada para iniciar um processamento de loop e controlar o número de vezes que o loop será executado.

```
* >Natural Source Header 000000.
define data local
 1 i (i2)
 1 numpar (n2)
end-define

/* i => iterador, valor alterado a cada loop
/* 2 => valor inicial
/* 10 => valor final
/* 2 => incremento ou decremento
for i 2 10 2
  write 'Número par:' i
end-for

END
```

#### Natural Web I/O Output

Page	1	23-05-17 13:41:51
Número par:	2	
Número par:	4	
Número par:	6	
Número par:	8	
Número par:	10	

## Repeat

A instrução ‘REPEAT’ inicia um loop de processamento cujo final é controlado por uma condição lógica. Se assemelha ao while e ao do while.

```
+ * >Natural Source Header 000000..  
define data local  
1 condicao1 (1)  
1 condicao2 (1) init <false>  
end-define  
  
/* repeat until se assemelha ao do while  
/* a condicao é posicionada no inferior do laço  
/* o write é executado até que condicao1 seja true  
/* aqui o write é executado apenas 1 vez  
repeat  
    write 'Estou no loop do repeat until.'  
    condicao1 := true  
    until condicao1 = true  
end-repeat  
  
skip 1  
  
/* a condicao é posicionada no inicio do laço  
/* como condicao2 inicia como false, o write é  
/* executado. Em seguida o condicao2 recebe true  
/* o write é executado apenas uma vez  
repeat  
    while condicao2 = false  
    write 'Estou no loop do repeat while.'  
    condicao2 := true  
end-repeat  
  
END
```

### Natural Web I/O Output

Page 1

23-05-17 14:10:29

Estou no loop do repeat until.

Estou no loop do repeat while.

Se a condição lógica for omitida, o loop deve ser encerrado com instruções do tipo 'ESCAPE BOTTOM' ou 'STOP'.

```
+ * >Natural Source Header 000000..  
define data local  
1 condicao1 (1)  
1 condicao2 (1) init <false>  
end-define  
  
/* se while ou until forem omitidos podemos encerrar  
/* o loop com escape bottom ou stop  
repeat  
    write 'Estou no loop do repeat usando escape bottom e o stop.'  
    condicao1 := true  
    if condicao1 = true  
        /* escape bottom  
        stop  
    end-if  
end-repeat  
  
END
```

#### Natural Web I/O Output

Page 1

23-05-17 14:20:13

Estou no loop do repeat usando escape bottom e o stop.

## Accept / Reject

Estes comandos são usados para aceitar (ACCEPT) ou rejeitar (REJECT) registros, de acordo com condições lógicas em processamento de loop. O Natural é quem faz o teste e não o banco de dados. Por essa razão, o registro é recuperado (lido), para que o teste seja efetuado. Os campos utilizados para teste podem ser campos da view ou variáveis de usuário.

Pode haver combinação de vários ACCEPT / REJECT numa leitura, mas o comando REJECT deve ser codificado antes do comando ACCEPT.

```
+ * >Natural Source Header 000000..  
define data local  
 1 vw VIEW OF VEICULOS  
    2 PROPRIETARIO (A60)  
    2 BAIRRO (A10)  
    2 PLACA (A7)  
end-define  
  
/* Accept => usado p/ aceitar registros de acordo  
/* com condicoes logicas em loops  
/* sf => espacamento entre colunas  
/* al => tamanho maximo p/ exibicao alfanumerico  
read vw by placa = 'AA 0001' ending at 'AA 0100'  
  accept if bairro = 'CENTRO'  
  display proprietario(al=20) bairro placa  
end-read  
  
END
```

### Natural Web I/O Output

Page	1	23-05-17 15:27:06
<hr/>		
PROPRIETARIO	BAIRRO	PLACA
KWAN TANIGUCHI	CENTRO	AA 0025
JONAN FERREIRA DA SI	CENTRO	AA 0057
EMILIO TAIROVICHE	CENTRO	AA 0058
RAFAEL BARCELAR DE S	CENTRO	AA 0090

## Escape Top / Bottom / Routine

ESCAPE TOP → causa o retorno para o início do processamento de loop. Dar-se-á nova repetição do processo de loop.

ESCAPE BOTTOM → encerra o processamento do loop, e continuará com o primeiro comando após o loop. Se a cláusula 'LABEL' for especificada, o processamento seguirá com o comando identificado pelo 'LABEL' ou pelo número de referência. Se a cláusula 'IMMEDIATE' for especificada, não se dará a evolução das funções 'STANDARD' (Aver, count, max) e as funções de quebras por fim de processamento.

ESCAPE ROUTINE → a opção indica que a rotina (que pode ter sido invocada por um PERFORM, CALLNAT, FETCH RETURN ou pelo programa principal) abandonará o controle retornando ao nível superior. No caso do programa principal, o Natural receberá o controle. Pode-se usar também a cláusula 'IMMEDIATE'.

```
+ * >Natural Source Header 000000..  
define data local  
1 i (i2) init <1>  
end-define  
  
/* escape top => retorna p/ o inicio do loop  
/* escape bottom => encerra o loop. Continuará  
/* com o primeiro comando apos o loop  
repeat  
    if i = 5  
        escape bottom  
    end-if  
    i := i + 1  
    write i  
    escape top  
end-repeat  
  
write 'Fim do loop.'  
  
END
```

### Natural Web I/O Output

Page	1	23-05-17 15:51:01
	2	
	3	
	4	
	5	
	Fim do loop.	

---

```
0010 REPEAT
0020 .
0030 .
0040 .
0050 REPEAT
0060 .
0070 .
0080
    ➤ 0090 REPEAT
    0100 .
    0110 .
    0120 .
    0130 ESCAPE TOP
    0140 ESCAPE BOTTOM
    0150 ESCAPE BOTTOM (0010) IMMEDIATE
    0160 .
    0170 .
    0180 .
    ➤ 0190 END-REPEAT */ (0090)
    0200 .
    0210 .
    0220 .
    0230 END-REPEAT */ (0050)
    0240 .
    0250 .
    0260 .
    ➤ 0270 END-REPEAT */ (0010)
    0280 .
    0290 .
    0300 .
    0310 END
```

---

## End

A instrução ‘END’ indica fim de programa. Todos os objetos têm de ter o END como último statement, com exceção do COPYCODE e do GET.

---

```
FIND ...
READ ...
END-READ
END-FIND
END
```

---

## **Terminate**

O comando termina a execução do programa e do Natural, retornando o controle ao monitor de teleprocessamento (CICS, TSO, etc.).

---

```
...
IF *PF-KEY = 'PF5' OR = 'PF17'
  TERMINATE
END-IF
```

---

## **Stop**

Termina a execução de um programa e retorna o controle para o Natural.

---

```
...
IF #OPÇÃO = 9
  STOP
END-IF
```

---

## **Limit**

A instrução 'LIMIT', limita o número de loop's para comandos de leitura (READ, FIND, HISTOGRAM) de acordo com o valor de 'n', retornando uma mensagem.

---

LIMIT *n*

---

```
...
LIMIT 4
*
READ EMPLOY-VIEW BY NAME STARTING FROM `BAKER`
...
END-READ
```

---

## 16 - Processamento de Quebra Automática

### At Start Of Data

Comando que executa uma série de comandos no início da leitura de um file.

---

```
[AT] START [OF] DATA  
...  
END-START
```

---

Deve ser codificado dentro do laço de leitura.

---

```
FIND EMPRESA WITH SIGLA-UF = 'MG'  
    AT START OF DATA  
        WRITE 'Início do Relatório.'  
    END-START  
END-FIND
```

---

### At End Of Data

Comando que executa uma série de comandos ao término da leitura de um file.

---

```
[AT] END [OF] DATA  
...  
END-ENDDATA
```

---

Deve ser codificado dentro do laço de leitura.

---

```
FIND EMPRESA WITH SIGLA-UF = 'MG'  
    AT START OF DATA  
        WRITE 'Término do Relatório.'  
    END-START  
END-FIND
```

---

## **At Top Of Page**

Comando que executa uma série de comandos toda vez que uma página for inicializada para impressão de um relatório.

---

```
[AT] TOP [OF] PAGE  
...  
END-TOPPAGE
```

---

Pode ser codificado em qualquer lugar do programa.

---

```
...  
AT TOP OF PAGE  
    WRITE 'Relatório de Funcionários em Férias.'  
END-TOPPAGE  
...
```

---

## **At End Of Page**

Comando que executa uma série de comandos quando uma condição de fim de página for detectada.

---

```
[AT] END [OF] PAGE  
...  
END-ENDPAGE
```

---

Pode ser codificada em qualquer parte do programa.

---

```
...  
AT END OF PAGE  
    WRITE 'Página:' #PAG(EM=ZZ9)  
END-ENDPAGE  
...
```

---

## At Break

A instrução 'AT BREAK' especifica os campo para cujos valores serão feitos controles de quebra automáticos ou manuais, e também os procedimentos a serem executados nestas quebras.

---

```
[AT] BREAK [OF] campo [/n/]
```

...

```
END-BREAK
```

---

campo: atributo do arquivo, cujo conteúdo será utilizado para teste de quebra, ou seja, o AT BREAK terá os comandos internos executados toda vez que ocorrer uma alteração do valor do campo em relação ao conteúdo anterior.

/n/: pode-se especificar a quebra de um certo número de caracteres iniciais do campo, através da cláusula /n/. Por exemplo, se quisermos uma quebra quando houver alteração da primeira letra de um nome, basta codificar AT BREAK OF NAME /1/.

Mais de um AT BREAK pode ser codificado numa operação de leitura, e quando acontece um quebra, as quebras de níveis anteriores também acontecem.

Funções disponíveis na quebra:

- COUNT → quantidade de registros lidos
- SUM → soma de valores
- OLD → conteúdo do campo de quebra antes de ocorrer a quebra
- AVER → média de valores
- MAX → maior valor lido
- MIN → menor valor lido

Exemplo de quebra simples:

---

```
READ EMPREGADO BY CARGO
  WHERE DEPTO = 'VENDAS'
  AT BREAK OF CARGO
    WRITE '-' (70) /
      10T 'CARGO:' OLD(CARGO) (AD=I) /
      40T 'SALARIO MEDIO:' AVER(SALARIO) /
      40T 'QTDE. FUNCIONARIOS:' COUNT(SALARIO) /
      40T 'TOTAL SALARIO:' SUM(SALARIO)
  END-BREAK
  DISPLAY MATRICULA NOME CARGO DEPTO
END-READ
```

---

Exemplo de quebra composta:

---

```
FIND EMPREGADO WITH DEPTO = 'COMPRAS'
  SORTED BY SECAO SETOR SERVICO
  DISPLAY MATRICULA NOME CARGO SALARIO
  AT BREAK OF SERVICO
    WRITE 10T 'CARGO:' OLD(CARGO) /
    40T 'MENOR SALARIO:' MIN(SALARIO) /
    40T 'MAIOR SALARIO:' MAX(SALARIO) /
    40T 'SALARIO MEDIO:' AVER(SALARIO) /
    40T 'TOTAL SALARIO:' SUM(SALARIO) /
    40T 'QTDE. FUNCIONARIOS:' COUNT(SALARIO)
  END-BREAK
  AT BREAK OF SETOR
    WRITE 10T 'CARGO:' OLD(CARGO) /
    40T 'MENOR SALARIO:' MIN(SALARIO) /
    40T 'MAIOR SALARIO:' MAX(SALARIO) /
    40T 'SALARIO MEDIO:' AVER(SALARIO) /
    40T 'TOTAL SALARIO:' SUM(SALARIO) /
    40T 'QTDE. FUNCIONARIOS:' COUNT(SALARIO)
  END-BREAK
  AT BREAK OF SECAO
    WRITE 10T 'CARGO:' OLD(CARGO) /
    40T 'MENOR SALARIO:' MIN(SALARIO) /
    40T 'MAIOR SALARIO:' MAX(SALARIO) /
    40T 'SALARIO MEDIO:' AVER(SALARIO) /
    40T 'TOTAL SALARIO:' SUM(SALARIO) /
    40T 'QTDE. FUNCIONARIOS:' COUNT(SALARIO)
  END-BREAK
END-FIND
```

---

## 17 - Modularização do Sistema

Adasdasd  
Asdasdas  
Asdasdasd  
asdasdasd

## 18 - Mapas

**Natural Web I/O Output**

CADASTRO DE CLIENTE

Nome  Idade  Sexo

**Natural Web I/O Output**

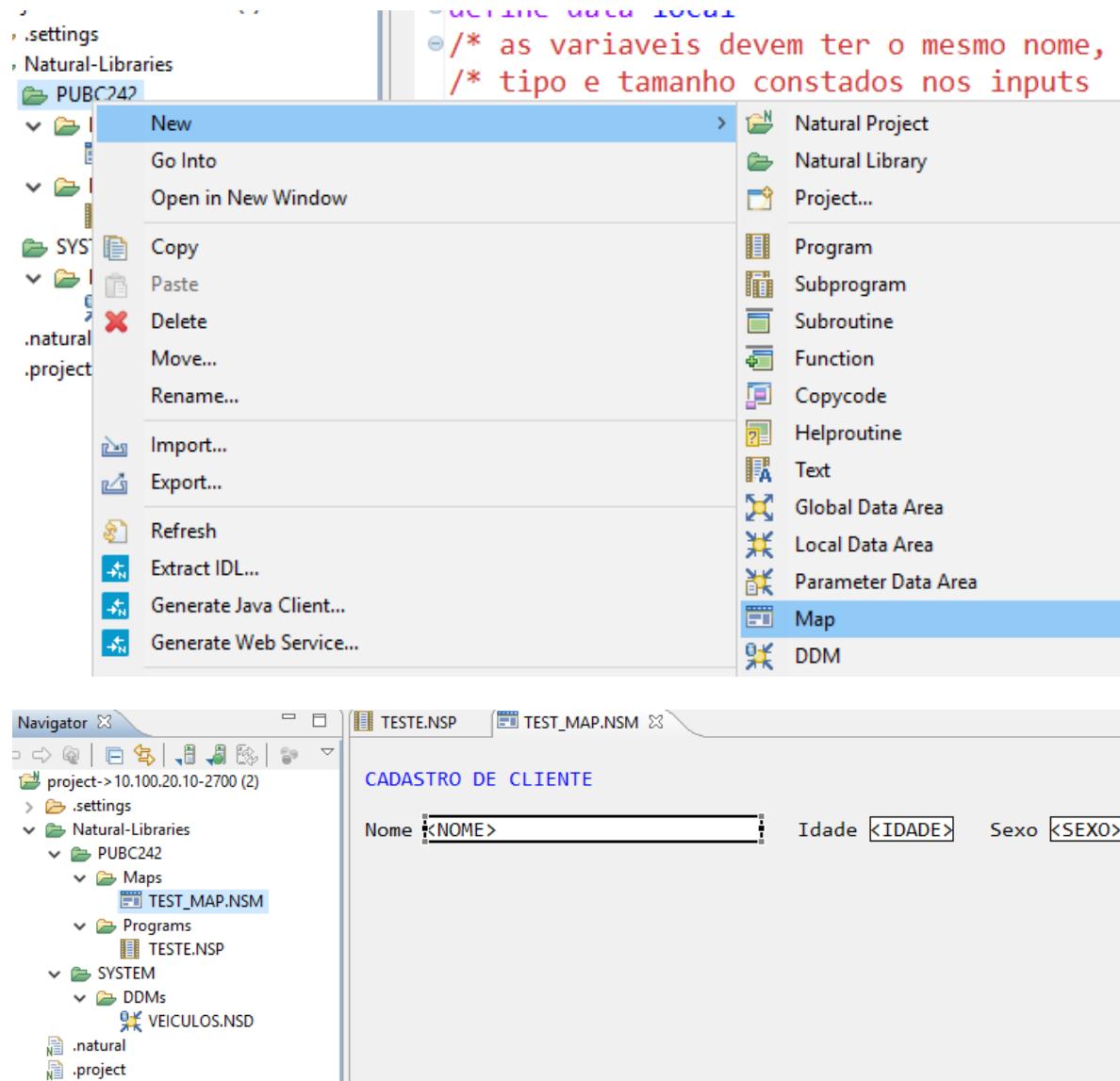
CADASTRO DE CLIENTE

Nome  Idade  Sexo

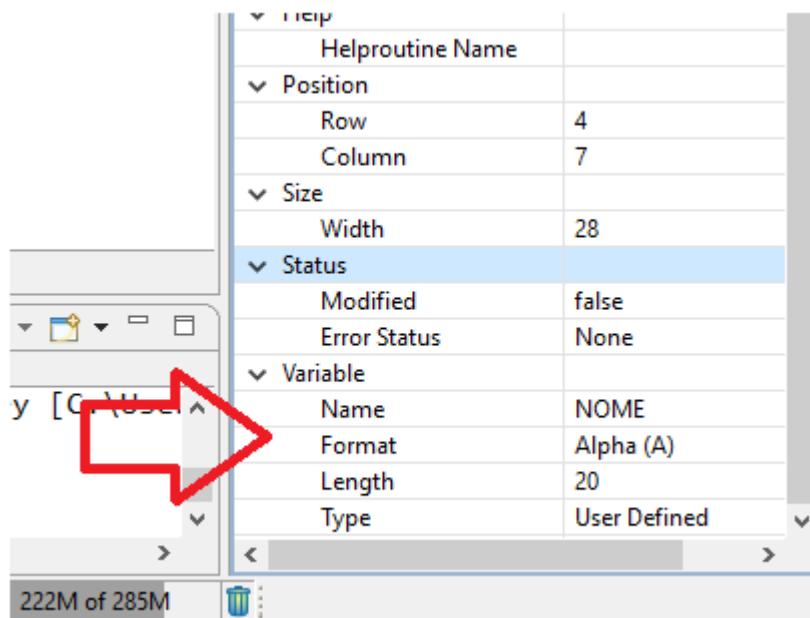
**Natural Web I/O Output**

Page	1	23-05-18 14:49:26
CLIENTE		
IDADE	NOME	SEXO
-----	-----	-----
33	ASDASD	M

## Criando o mapa



Ao selecionar o input, no canto inferior direito podemos mudar as propriedades do input selecionado. O nome, tipo e tamanho devem ser o mesmo no programa principal dentro de define data local.



```
+ * >Natural Source Header 000000.
define data local
/* as variaveis devem ter o mesmo nome,
/* tipo e tamanho constados nos inputs
1 CLIENTE
  2 IDADE (I2)
  2 NOME (A020)
  2 SEXO (A001)
end-define

/* TEST_MAP => nome do mapa
INPUT USING MAP 'TEST_MAP'

/* sf => espacamento entre as colunas
/* hc => posicao do titulo da
/* colunas. Center, left ou right
display (SF=2 HC=L) CLIENTE

END
```

## 19 - Window

```
+ * >Natural Source Header 000000..  
 DEFINE DATA LOCAL  
 1 IDADE (I2)  
 1 NOME (A020)  
 1 SEXO (A001)  
 END-DEFINE  
  
 /* janela1 é o nome  
  
 /* size 10 * 100 => tamanho da janela linhas X colunas  
 /* size auto => tamanho da janela é determinado automaticamente  
  
 /* base => posicao da abertura da janela  
 /* base pode ser 5 / 12 ou TOP/BOTTOM LEFT/RIGHT  
  
 /* framed on => janela terá moldura  
  
 /* position on => se a janela a ser exibida for maior que a  
 /* tela fisica, será mostrado na ultima linha uma informacao  
 /* que existe mais janela a ser exibida  
  
 /* set window => torna a janela ativa  
 /* set window off => desabilita a janela  
  
 DEFINE WINDOW JANELA1  
 size auto  
 base top right  
 title 'Cadastro de Cliente'  
 framed on position off  
  
 SET WINDOW 'JANELA1'  
 INPUT USING MAP 'test_map'  
 SET WINDOW OFF  
  
 END
```

## Natural Web I/O Output

### Cadastro de Cliente

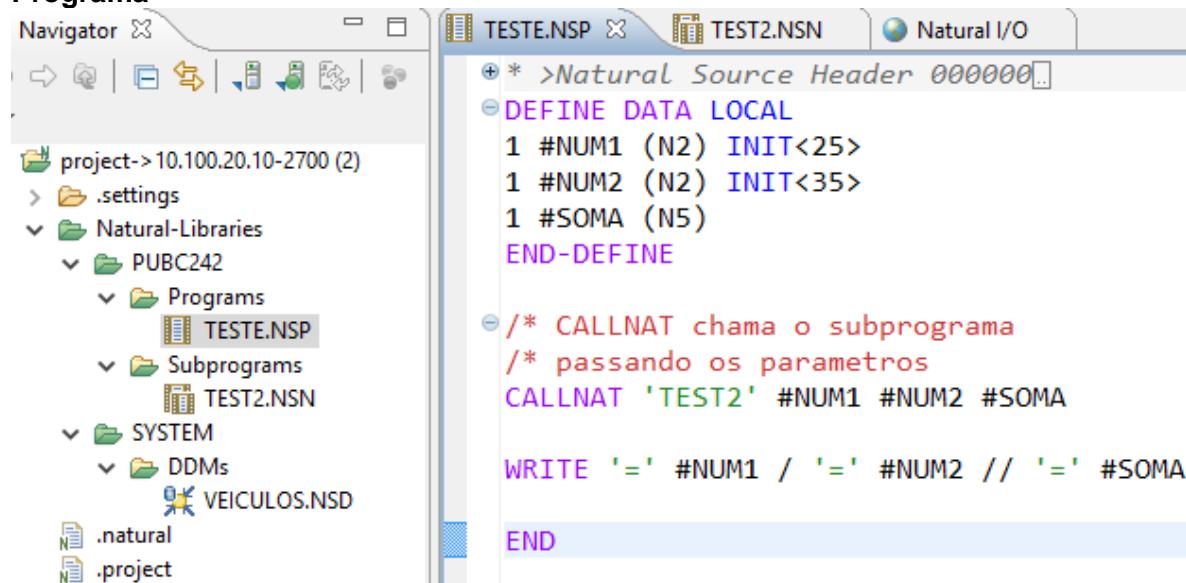
#### CADASTRO DE CLIENTE

Nome  Idade  Sexo

## 20 - Subprogramas

São módulos que não podem ser executados independentemente. Sua chamada é através do comando CALLNAT. A passagem de dados para os subprogramas é muito eficiente, pois não há movimentação de campos, apenas o endereço em memória é passado.

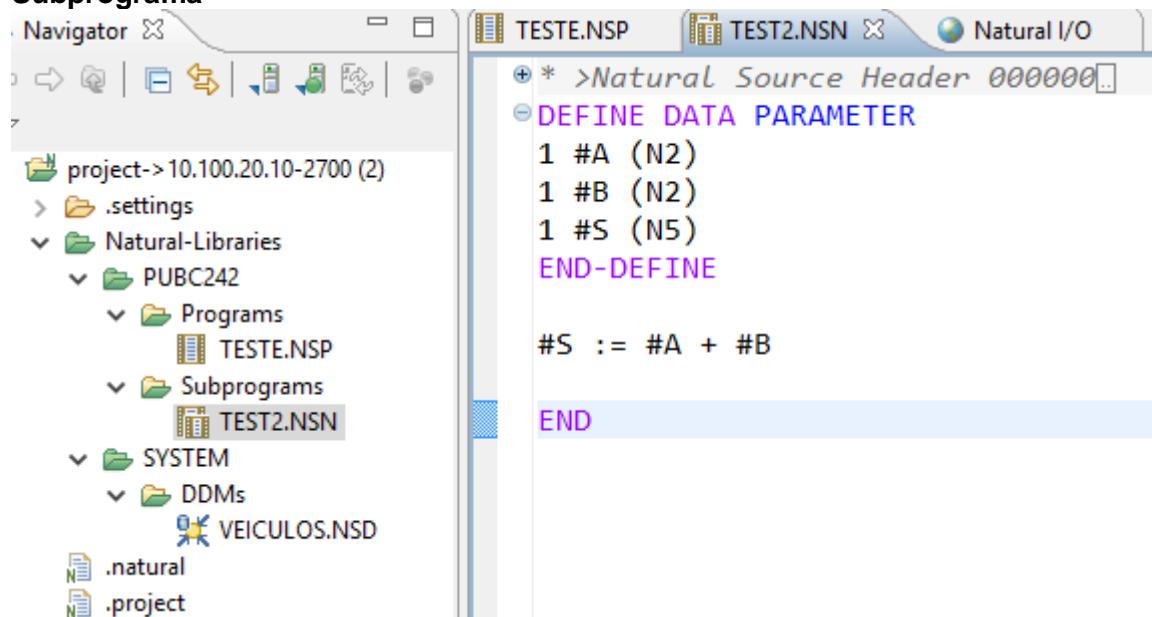
### Programa



The screenshot shows the SIMATIC Manager interface. On the left, the Navigator pane displays a project structure for '10.100.20.10-2700 (2)' containing '.settings', 'Natural-Libraries' (with 'PUBC242' selected), 'Programs' (with 'TESTE.NSP' selected), 'Subprograms' (with 'TEST2.NSN' selected), 'SYSTEM', 'DDMs', and files '.natural' and '.project'. On the right, the main area shows the code for 'TESTE.NSP':

```
* >Natural Source Header 000000..  
DEFINE DATA LOCAL  
1 #NUM1 (N2) INIT<25>  
1 #NUM2 (N2) INIT<35>  
1 #SOMA (N5)  
END-DEFINE  
  
/* CALLNAT chama o subprograma  
/* passando os parametros  
CALLNAT 'TEST2' #NUM1 #NUM2 #SOMA  
  
WRITE '=' #NUM1 / '=' #NUM2 // '=' #SOMA  
  
END
```

### Subprograma



The screenshot shows the SIMATIC Manager interface. The Navigator pane is identical to the previous screenshot. On the right, the main area shows the code for 'TEST2.NSN':

```
* >Natural Source Header 000000..  
DEFINE DATA PARAMETER  
1 #A (N2)  
1 #B (N2)  
1 #S (N5)  
END-DEFINE  
  
#S := #A + #B  
  
END
```

**Natural Web I/O Output**

Page	1	23-05-19 11:17:59
#NUM1:	25	
#NUM2:	35	
#SOMA:	60	

## 21 - Sub-rotina Externa

A sub-rotina externa não pode ser executada independentemente, pois deve ser acionada através do comando PERFORM. A grande novidade é a passagem de parâmetros no próprio comando PERFORM.

Os dados são recebidos na sub-rotina, através de uma PDA (Parameter Data Area) interna ou externa, onde os campos deverão estar na mesma ordem, formato e tamanho.

### Programa

```

* >Natural Source Header 000000..
DEFINE DATA LOCAL
1 #NUM1 (N2) INIT<25>
1 #NUM2 (N2) INIT<35>
1 #SOMA (N5)
END-DEFINE

/* PERFORM chama a sub-rotina
/* passando os parametros
PERFORM SOMA #NUM1 #NUM2 #SOMA

WRITE '=' #NUM1 / '=' #NUM2 // '=' #SOMA

END

```

**Sub-rotina**

```

TESTE.NSP SOMA.NSS Natural I/O
+ * >Natural Source Header 000000...
DEFINE DATA PARAMETER
1 #NUM1 (N2)
1 #NUM2 (N2)
1 #SOMA (N5)
END-DEFINE

DEFINE SUBROUTINE SOMA
#SOMA := #NUM1 + #NUM2
END-SUBROUTINE

END

```

**Natural Web I/O Output**

Page	1	23-05-19 11:36:22
#NUM1:	25	
#NUM2:	35	
#SOMA:	60	

## 22 - Copycode

Um objeto do tipo copycode contém uma parte do código-fonte que pode ser incluída em outro objeto por meio de uma instrução INCLUDE.

Portanto, se você tiver um bloco de instruções que deve aparecer de forma idêntica em vários objetos, poderá usar um objeto copycode em vez de codificar o bloco de instruções várias vezes. Isso reduz o esforço de codificação e também garante que os blocos sejam realmente idênticos.

O copycode é incluído na compilação, ou seja, as linhas do código-fonte do copycode não são inseridas fisicamente no objeto que contém a instrução INCLUDE, mas serão incluídas no processo de compilação e, portanto, fazem parte do objeto catalogado resultante.

Conseqüentemente, quando você modifica o código-fonte do copycode, também precisa catalogar todos os objetos que usam esse copycode.

Copycode não pode ser executado sozinho. Não pode ser guardado, mas apenas salvo.

Uma instrução END não deve ser colocada dentro de um copycode.

```
TESTE.NSP CP_CD1.NSC OI.NSP Natural I/O
+ * >Natural Source Header 000000
DEFINE DATA LOCAL
1 #NUM1 (N2)
1 #NUM2 (N2)
1 #SOMA (N5)
END-DEFINE

/* executa aqui a copycode CP_CD1
INCLUDE CP_CD1

/* M => mode: modifiable
/* D => apresentacao: default
/* L => alinhamento: left
/* T => tratamento caracter: transforma em maiuscula
/* ZP => determina se os valores de campo de 0
/* devem ser exibidos ou nao
INPUT (AD=MDLT ZP=OFF)
    'ENTRE COM UM NÚMERO...:' #NUM1 /
    'ENTRE COM OUTRO NÚMERO:' #NUM2 //
    '-' (27) /* Aqui o traço (-) é imprimido 27X
    'PF3 - SAI'

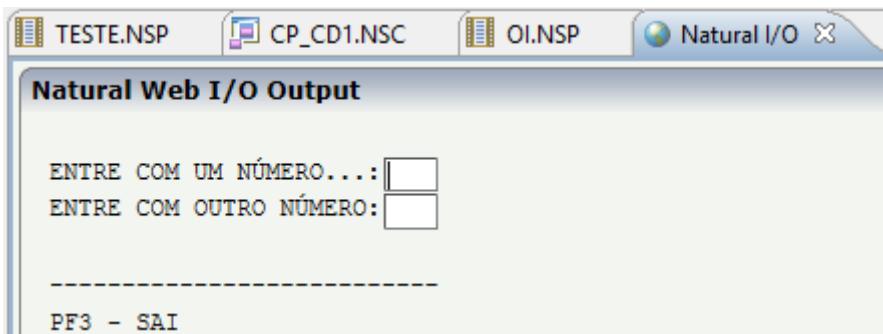
WRITE '=' #NUM1 / '=' #NUM2

END
```

```
TESTE.NSP CP_CD1.NSC OI.NSP Natural I/O
* >Natural Source Header 000000
* :Mode S
* :CP
* <Natural Source Header
/* SET KEY ALL => Faz com que todas as chaves
/* sejam sensíveis ao programa. Todas as atribuições
/* de função a qualquer tecla são substituídas
SET KEY ALL

/* Ao teclar f3 o programa OI é executado
SET KEY PF3 = 'OI'
```

```
* >Natural Source Header 000000..  
/* A instrução TERMINATE é usada para encerrar uma sessão  
/* Natural. Uma instrução TERMINATE pode ser colocada em  
/* qualquer lugar dentro de um programa Natural. Quando  
/* uma instrução TERMINATE é executada, nenhum processamento  
/* de fim de página ou loop final será executado.  
TERMINATE  
END
```



Natural Web I/O Output

Page 1 23-05-19 13:38:46

#NUM1: 22

#NUM2: 4

## 23 - Help routines

As rotinas de ajuda têm características específicas para facilitar o processamento de pedidos de ajuda. Eles podem ser usados para implementar sistemas de ajuda complexos e interativos. Eles são criados com o editor de código-fonte.

Um usuário Natural pode invocar uma rotina de ajuda Natural inserindo o caractere de ajuda em um campo ou pressionando a tecla de ajuda (geralmente PF1). O caractere de ajuda padrão é um ponto de interrogação (?).

O caractere de ajuda deve ser digitado apenas uma vez.

Se ainda não estiver especificada, a chave de ajuda pode ser especificada com a instrução SET KEY:

SET KEY PF1=HELP

Uma rotina de ajuda só pode ser invocada por um usuário se tiver sido especificada no programa ou mapa a partir do qual deve ser invocada.

Uma rotina de ajuda pode ser especificada:

- em um programa e em um mapa.

O nome de uma rotina de ajuda pode ser especificado com o parâmetro de sessão HE de uma instrução INPUT:

INPUT (HE='HELP2112')

ou usando o editor de mapas.

O nome de uma rotina de ajuda pode ser especificado como uma constante alfanumérica ou como uma variável alfanumérica contendo o nome. Se for uma constante, o nome da rotina de ajuda deve ser especificado entre apóstrofos.

O processamento de uma rotina de ajuda pode ser interrompido com uma instrução ESCAPE ROUTINE.

Tenha cuidado ao usar instruções END OF TRANSACTION ou BACKOUT TRANSACTION em uma rotina de ajuda, porque isso afetará a lógica de transação do programa principal.

Uma rotina de ajuda pode acessar a área de dados global atualmente ativa (mas não pode ter sua própria área de dados global). Além disso, pode ter sua própria área de dados local (LDA).

Os dados também podem ser passados de/para uma rotina de ajuda por meio de parâmetros. Uma rotina de ajuda pode ter até 20 parâmetros explícitos e um parâmetro implícito. Os parâmetros explícitos são especificados com o operando HE após o nome da rotina de ajuda:

HE='MYHELP','001'

O parâmetro implícito é o campo para o qual a rotina de ajuda foi invocada:

INPUT #A (A5) (HE='YOURHELP','001')

onde 001 é um parâmetro explícito e #A é o parâmetro implícito.

Isso é especificado na instrução DEFINE DATA PARAMETER da rotina de ajuda como:

```
DEFINE DATA PARAMETER
 1 #PARM1 (A3)          /* explicit parameter
 1 #PARM2 (A5)          /* implicit parameter
END-DEFINE
```

Observe que o parâmetro implícito (#PARM2) pode ser omitido. O parâmetro implícito é utilizado para acessar o campo para o qual a ajuda foi solicitada e para retornar dados da helproutine para o campo. Por exemplo, você pode implementar um programa de calculadora como uma helproutine e ter o resultado dos cálculos retornado ao campo.

Quando a ajuda é chamada, a helproutine é chamada antes que os dados sejam passados da tela para as áreas de dados do programa. Isso significa que as helproutines não podem acessar os dados inseridos na mesma transação de tela.

## 24 - Manipulação de Dados

```
+ * >Natural Source Header 000000..  
DEFINE DATA LOCAL  
/* nesse exemplo os arrays possuem o mesmo  
/* tamanho (3,2). 3 linhas e 2 colunas  
/* o valor fica dentro dos caracteres <>  
1 #A (N2/3,2) INIT  
    (1,1) <1> (1,2) <1>  
    (2,1) <2> (2,2) <0>  
    (3,1) <3> (3,2) <2>  
1 #B (N2/3,2) INIT  
    (1,1) <3> (1,2) <6>  
    (2,1) <4> (2,2) <8>  
    (3,1) <8> (3,2) <3>  
END-DEFINE  
  
/* A barra significa pular uma linha  
/* '-'(22) => o traço será repetido 22x  
WRITE 'VALORES INICIAIS DE #A' / '-'(22)  
WRITE '=' #A(1,1) '=' #A(1,2) /  
      '=' #A(2,1) '=' #A(2,2) /  
      '=' #A(3,1) '=' #A(3,2)  
  
WRITE / 'VALORES INICIAIS DE #B' / '-'(22)  
WRITE '=' #B(1,1) '=' #B(1,2) /  
      '=' #B(2,1) '=' #B(2,2) /  
      '=' #B(3,1) '=' #B(3,2)  
  
/* SOMA DO ARRAY "A" COM O "B"  
ADD #A(*,*) TO #B(*,*)  
  
WRITE / 'VALORES FINAIS DE #B' / '-'(20)  
WRITE '=' #B(1,1) '=' #B(1,2) /  
      '=' #B(2,1) '=' #B(2,2) /  
      '=' #B(3,1) '=' #B(3,2)  
END|
```

## Natural Web I/O Output

Page 1 23-05-22 10:20:35

VALORES INICIAIS DE #A

```
-----
#A: 1 #A: 1
#A: 2 #A: 0
#A: 3 #A: 2
```

VALORES INICIAIS DE #B

```
-----
#B: 3 #B: 6
#B: 4 #B: 8
#B: 8 #B: 3
```

VALORES FINAIS DE #B

```
-----
#B: 4 #B: 7
#B: 6 #B: 8
#B: 11 #B: 5
```

MORE [ ]

```

+ * >Natural Source Header 000000[]

DEFINE DATA LOCAL
 1 #A (N2/3,2) INIT
    (1,1) <1> (1,2) <1>
    (2,1) <2> (2,2) <0>
    (3,1) <3> (3,2) <2>
 1 #B (N2/3,2) INIT
    (1,1) <3> (1,2) <6>
    (2,1) <4> (2,2) <8>
    (3,1) <8> (3,2) <3>
 1 #C (N2) INIT <6>
END-DEFINE

WRITE 'VALORES INICIAL DE #C' / '-'(21)
WRITE '=' #C /

WRITE 'VALORES INICIAIS DE #A' / '-'(22)
WRITE '=' #A(1,1) '=' #A(1,2) /
      '=' #A(2,1) '=' #A(2,2) /
      '=' #A(3,1) '=' #A(3,2)

WRITE / 'VALORES INICIAIS DE #B' / '-'(22)
WRITE '=' #B(1,1) '=' #B(1,2) /
      '=' #B(2,1) '=' #B(2,2) /
      '=' #B(3,1) '=' #B(3,2)

/* SOMA 6 APENAS A PRIMEIRA LINHA DE B
ADD #C TO #B(1,*)

WRITE / 'VALORES FINAIS DE #B' / '-'(20)
WRITE '=' #B(1,1) '=' #B(1,2) /
      '=' #B(2,1) '=' #B(2,2) /
      '=' #B(3,1) '=' #B(3,2)

END

```

## Natural Web I/O Output

Page 1

23-05-22 10:35:12

VALORES INICIAL DE #C

-----  
#C: 6

VALORES INICIAIS DE #A

-----  
#A: 1 #A: 1  
#A: 2 #A: 0  
#A: 3 #A: 2

VALORES INICIAIS DE #B

-----  
#B: 3 #B: 6  
#B: 4 #B: 8  
#B: 8 #B: 3

VALORES FINAIS DE #B

-----  
#B: 9 #B: 12  
#B: 4 #B: 8  
#B: 8 #B: 3

MORE

```

+ * >Natural Source Header 000000..]
@DEFINE DATA LOCAL
 1 #A (N2/3,2) INIT
    (1,1) <1> (1,2) <1>
    (2,1) <2> (2,2) <0>
    (3,1) <3> (3,2) <2>
END-DEFINE

WRITE 'VALORES INICIAIS DE #A' / '-'(22)
WRITE '=' #A(1,1) '=' #A(1,2) /
      '=' #A(2,1) '=' #A(2,2) /
      '=' #A(3,1) '=' #A(3,2)

/* SOMA 2 A SEGUNDA COLUNA DE "A"
ADD 2 TO #A(*,2)

WRITE / 'VALORES FINAIS DE #A' / '-'(20)
WRITE '=' #A(1,1) '=' #A(1,2) /
      '=' #A(2,1) '=' #A(2,2) /
      '=' #A(3,1) '=' #A(3,2)
END

```

### Natural Web I/O Output

Page 1

23-05-22 11:23:07

VALORES INICIAIS DE #A

```

-----
#A: 1 #A: 1
#A: 2 #A: 0
#A: 3 #A: 2

```

VALORES FINAIS DE #A

```

-----
#A: 1 #A: 3
#A: 2 #A: 2
#A: 3 #A: 4

```

## 25 - Manipulação de Strings/Arrays

### Examine

A instrução examine pesquisa uma string em um campo (alfanumérico ou array), através de uma opção, e altera ou deleta ou conta a quantidade de ocorrência de um campo.

```
+ * >Natural Source Header 000000..  
 DEFINE DATA LOCAL  
 1 #A (A15) INIT <'TX YZU'>  
 1 #B (A10) INIT <'XP01'>  
 1 #C (A10) INIT <'XP01'>  
 1 #CONT1 (N2)  
 1 #CONT2 (N2)  
 1 #CONT3 (N2)  
 END-DEFINE  
  
/* EXAMINE => PESQUISA UMA STRING EM UM CAMPO (ALFA OU ARRAY)  
/* GIVING NUMBER => OBTEM A QUANTIDADE DE VEZES QUE OCORREU  
/* ESPAÇO EM BRANCO NA VARIÁVEL #A, SE NAO FOR ENCONTRADO  
/* RETORNA 0  
EXAMINE #A FOR ' ' GIVING NUMBER #CONT1  
WRITE 'QUANTIDADE DE ESPAÇO EM BRANCO SEM FULL:' #CONT1  
  
/* EXAMINE FULL => PROCURA NA VARIÁVEL TODA DE ACORDO  
/* COM SEU TAMAÑO O ESPAÇO EM BRANCO  
EXAMINE FULL #B FOR ' ' GIVING NUMBER #CONT2  
WRITE 'QUANTIDADE DE ESPAÇO EM BRANCO COM FULL:' #CONT2  
  
/* RETORNA 0 POIS NAO ENCONTROU ESPAÇO EM BRANCO  
EXAMINE #C FOR ' ' GIVING NUMBER #CONT3  
WRITE 'QUANTIDADE DE ESPAÇO EM BRANCO SEM FULL:' #CONT3  
  
END
```

#### Natural Web I/O Output

Page 1

23-05-23 10:13:05

```
QUANTIDADE DE ESPAÇO EM BRANCO SEM FULL: 1  
QUANTIDADE DE ESPAÇO EM BRANCO COM FULL: 6  
QUANTIDADE DE ESPAÇO EM BRANCO SEM FULL: 0
```

## Examine Delete / Replace

Com a opção DELETE/REPLACE é possível pesquisar o campo pelo valor especificado e excluir ou substituir as ocorrências encontradas pelo valor desejado. Se for especificado FIRST para qualquer das opções, apenas a primeira ocorrência encontrada será excluída ou substituída.

```
+ * >Natural Source Header 000000
 DEFINE DATA LOCAL
 1 #A (A10) INIT<'XYZ'>
 1 #B (A10) INIT<'XYZ'>
 1 #C (A10) INIT<'XXYYZ'>
 1 #D (A10) INIT<'XYZZK'>
 END-DEFINE

 /* EXAMINE DELETE => PESQUISA O VALOR NA VARIAVEL
 /* LOGO EM SEGUIDA O DELETA
 WRITE '=' #A
 EXAMINE #A FOR 'Y' DELETE
 WRITE '#A DEPOIS DO EXAMINE / DELETE: ' #A

 SKIP 1

 /* EXAMINE REPLACE => PESQUISA O VALOR NA VARIAVEL
 /* LOGO EM SEGUIDA O SUBSTITUI PELO VALOR ESPECIFICADO
 /* APOS O REPLACE
 WRITE '=' #B
 EXAMINE #B FOR 'Y' REPLACE '3'
 WRITE '#B DEPOIS DO EXAMINE / REPLACE: ' #B

 SKIP 1

 /* REPLACE FIRST => APENAS A PRIMEIRA OCORRENCIA
 /* ENCONTRADA SERÁ EXCLUIDA OU SUBSTITUIDA
 WRITE '=' #C
 EXAMINE #C FOR 'Y' REPLACE FIRST '*'
 WRITE '#C DEPOIS DO EXAMINE / REPLACE: ' #C

END
```

## Natural Web I/O Output

```
Page      1                               23-05-23 11:04:20

#A: XYZ
#A DEPOIS DO EXAMINE / DELETE: XZ

#B: XYZ
#B DEPOIS DO EXAMINE / REPLACE: X3Z

#C: XXYYZ
#C DEPOIS DO EXAMINE / REPLACE: XX*YZ
```

## Examine Giving Number

A opção GIVING NUMBER fornece o número de ocorrências do valor pesquisado no campo fonte. Esta informação refere-se ao campo antes das operações com REPLACE ou DELETE.

```
+* >Natural Source Header 000000..
```

```
 DEFINE DATA LOCAL
1 #A (A15) INIT<'XYZXYZXYZXYZXYZ'>
1 #CONT (N2)
END-DEFINE

WRITE '=' #A

SKIP 1

/* GIVING NUMBER => fornece o nº de ocorrências do valor
/* pesquisado
/* Aqui o Z será trocado por *
EXAMINE #A FOR 'Z' REPLACE WITH '*' GIVING NUMBER #CONT

WRITE '#A DEPOIS DO EXAMINE / REPLACE: ' #A //
      'NUMERO DE OCORRÊNCIA DE *: ' #CONT

END
```

## Natural Web I/O Output

Page 1

23-05-23 11:17:32

#A: XYZXYZXYZXYZ

#A DEPOIS DO EXAMINE / REPLACE: XY\*XY\*XY\*XY\*XY\*

NUMERO DE OCORRÊNCIA DE \*: 5

## **Examine Giving Position**

A opção GIVING POSITION fornece a posição da primeira ocorrência do valor pesquisado.

## Natural Web I/O Output

Page 1

23-05-23 11:28:58

#A: XYZXYZXYZXYZXYZXYZXYZ

#A DEPOIS DO EXAMINE / DELETE: XXXXXXXX

POSIÇÃO DO PRIMEIRO YZ ANTES DO DELETE: 2

## Examine Giving Length

A opção GIVING LENGTH fornece o tamanho do campo pesquisado depois que todas as operações de REPLACE / DELETE forem feitas.

```
④ DEFINE DATA LOCAL
 1 #A (A10) INIT<'XYZX'>
 1 #CONT (N2)
END-DEFINE

WRITE '=' #A

SKIP 1

④ /* GIVING LENGTH => fornece o tamanho do campo pesquisado
/* depois que todas as operacoes de REPLACE/DELETE forem
/* feitas
/* Aqui o "k*k" substituirá o "Z" e apos isso seu novo tamanho
/* será armazenado na variavel #CONT
EXAMINE #A FOR 'Z' REPLACE 'k*k' GIVING LENGTH #CONT

WRITE '#A DEPOIS DO EXAMINE / REPLACE: ' #A //
      'O NUMERO DE CARACTERES APOS O REPLACE: ' #CONT

END|
```

### Natural Web I/O Output

Page	1	23-05-23 11:52:41
#A: XYZX		
#A DEPOIS DO EXAMINE / REPLACE: XYk*kX		
O NUMERO DE CARACTERES APOS O REPLACE: 6		

## Examine Giving Index

Com a opção GIVING INDEX é possível localizar um determinado valor em um ARRAY e fornecer a primeira ocorrência do ARRAY em que o valor de pesquisa foi encontrado. Esta opção também é válida para arrays multidimensionais.

```
* >Natural Source Header 000000..  
DEFINE DATA LOCAL  
/* #TAB (A1/5,3,3) => matriz tridimensional com  
/* 5 linhas, 3 colunas e 3 tabelas  
1 #TAB (A1/5,3,3) INIT (1,2,3) <'A'> (5,1,2) <'B'>  
1 #LINHA (N2)  
1 #COLUNA (N2)  
1 #TABELA (N2)  
END-DEFINE  
  
/* GIVING INDEX => localiza um valor em um ARRAY e  
/* fornece o indice da primeira ocorrência  
/* EXAMINE #TAB(*,*,*) => busca em todas as linhas, colunas  
/* e tabelas  
EXAMINE #TAB(*,*,*) FOR 'A' GIVING INDEX #LINHA #COLUNA #TABELA  
  
WRITE 'A OCORRÊNCIA DO VALOR "A" é' //  
      'LINHA.:' #LINHA /  
      'COLUNA:' #COLUNA /  
      'PLANO.:' #TABELA /  
  
EXAMINE #TAB(*,*,*) FOR 'B' GIVING INDEX #LINHA #COLUNA #TABELA  
  
WRITE 'A OCORRÊNCIA DO VALOR "B" é' //  
      'LINHA.:' #LINHA /  
      'COLUNA:' #COLUNA /  
      'PLANO.:' #TABELA /  
  
END
```

## Natural Web I/O Output

Page 1

23-05-24 10:34:47

A OCORRÊNCIA DO VALOR 'A' é

LINHA.: 1  
COLUNA: 2  
PLANO.: 3

A OCORRÊNCIA DO VALOR 'B' é

LINHA.: 5  
COLUNA: 1  
PLANO.: 2

```
+ * >Natural Source Header 000000..  
DEFINE DATA LOCAL  
1 #MARCA (A10/5) INIT <'FIAT','CITROEN','VOLVO','AUDI','FORD'>  
1 #VEICULO (A15/5) INIT <'PALIO','PICASSO','REVOLUTION','A6','KA'>  
  
1 #MARCA_AUX (A10)  
1 #INDICE (N1)  
END-DEFINE  
  
#MARCA_AUX := 'AUDI'  
  
/* AQUI É PESQUISADO O VALOR "AUDI" QUE ESTA NA VARIÁVEL  
/* #MARCA_AUX  
/* A PESQUISA É FEITA DENTRO DO VETOR #MARCA  
/* O INDICE DO VALOR PESQUISADO É ARMAZENADO NA VARIÁVEL  
/* #INDICE  
EXAMINE #MARCA(*) #MARCA_AUX GIVING INDEX #INDICE  
  
DISPLAY #MARCA(*) #VEICULO(*) #MARCA_AUX #INDICE #MARCA(#INDICE)  
#VEICULO(#INDICE)  
  
END
```

Natural Web I/O Output					
Page	1				23-05-24 10:52:44
#MARCA	#VEICULO	#MARCA_AUX	#INDICE	#MARCA	#VEICULO
<hr/>					
FIAT	PALIO	AUDI	4	AUDI	A6
CITROEN	PICASSO				
VOLVO	REVOLUTION				
AUDI	A6				
FORD	KA				

## Examine Substring

A opção SUBSTRING permite que uma porção do campo seja examinada. Depois do nome (operando1), deve ser especificada a primeira posição a ser examinada (operando2) e então o tamanho (operando3) da parte do campo a ser pesquisada.

```
* >Natural Source Header 000000..
```

```
DEFINE DATA LOCAL
1 #D (A26) INIT <'TESTECOMANDOEXAMINENOCURSO'>
1 #NUMBER (N2)
1 #POSICAO (N2)
END-DEFINE

/* EXAMINE SUBSTRING => permite que uma porcao do campo
/* seja examinada
/* SUBSTRING(#D,13,7) => primeiro a variavel, seguido da
/* primeira posicao a ser examinada e depois o tamanho
/* do campo a ser pesquisado
/* Neste exemplo a pesquisa sera feita em EXAMINE
/* O Nº DE OCORRENCIAS E ARMAZENADO NA VARIABEL #NUMBER
/* A POSICAO DO VALOR A SER PESQUISADO E ARMAZENADO NA
/* VARIABEL #POSICAO
EXAMINE SUBSTRING(#D,13,7) 'MINE' GIVING #NUMBER POSITION #POSICAO

WRITE #D / '=' #NUMBER / '=' #POSICAO

END
```

### Natural Web I/O Output

Page 1 23-05-24 11:29:24

TESTECOMANDOEXAMINENOCURSO

#NUMBER: 1

#POSICAO: 4

```
+ * >Natural Source Header 000000
@DEFINE DATA LOCAL
 1 #NOME (A15/8) INIT <'LUCIANO','LUIZA','ANA','LUCIA',
      'PERDIGAO','ROSA','MONICA','LUCAS'>
 1 #NUMBER (N2)
END-DEFINE

/* A PESQUISA SERÁ FEITA NO VETOR #NOME
/* EM CADA VALOR A PESQUISA INICIARÁ NA POSICAO 1
/* O TAMANHO A SER PESQUISADO SERÁ DE 3 BYTES, OU
/* SEJA, APENAS NAS 3 PRIMEIRAS LETRAS DOS NOMES
/* O Nº DE OCORRENCIAS DO VALOR PESQUISADO SERÁ
/* ARMAZENADO NA VARIABEL #NUMBER
EXAMINE SUBSTRING(#NOME(*),1,3) 'LUC' GIVING #NUMBER

WRITE #NOME(*) // '=' #NUMBER|
```

END

### Natural Web I/O Output

Page 1 23-05-24 11:43:15

LUCIANO

LUIZA

ANA

LUCIA

PERDIGAO

ROSA

MONICA

LUCAS

#NUMBER: 3

## Examine Pattern

A opção PATTERN tem a capacidade de pesquisar um campo excluindo posições selecionadas.

“.” (ponto), “?” (interrogação) e “\_” (underline), representam uma posição a ser ignorada.

“\*” (asterisco) e “%” (percentual), representam uma posição ou mais a serem ignoradas.

```
+ * >Natural Source Header 000000..  
 DEFINE DATA LOCAL  
1 #TAB (A10/4) INIT<'CARRO','CARROÇA','GARRAFA','CAVALO'>  
1 #I (N3)  
1 #N (N3)  
END-DEFINE  
  
/* PATTERN => pesquisa um campo excluindo as posicoes  
/* selecionadas  
  
/* NOS EXEMPLOS ABAIXO O Nº DE OCORRENCIAS DO VALOR  
/* PESQUISADO SERÁ ARMAZENADO NA VARIABEL #N  
  
/* ponto(.) interrogação(?) e underline(_) representam  
/* uma posicao a ser ignorada  
WRITE 'PESQUISANDO COM APENAS UM CAMPO ENTRE O "A" E O "O"'  
FOR #I 1 4  
    EXAMINE #TAB(#I) PATTERN 'A.O' GIVING NUMBER #N  
    WRITE #TAB(#I) '-' #N  
END-FOR  
  
SKIP 1  
  
/* asterisco(*) e percentual(%) representam uma posicao  
/* ou mais a serem ignoradas  
WRITE 'PESQUISANDO COM UM OU MAIS CAMPOS ENTRE O "A" E O "O"'  
FOR #I 1 4  
    EXAMINE #TAB(#I) PATTERN 'A*O' GIVING NUMBER #N  
    WRITE #TAB(#I) '-' #N  
END-FOR  
  
END|
```

## Natural Web I/O Output

Page 1

23-05-24 15:24:02

PESQUISANDO COM APENAS UM CAMPO ENTRE O 'A' E O 'O'

CARRO	-	0
CARROÇA	-	0
GARRAFA	-	0
CAVALO	-	1

PESQUISANDO COM UM OU MAIS CAMPOS ENTRE O 'A' E O 'O'

CARRO	-	1
CARROÇA	-	1
GARRAFA	-	0
CAVALO	-	1

## Examine With Delimiters

A opção WITH DELIMITERS é utilizada para um valor que está entre espaço em branco ou qualquer outro que não seja letra ou número.

WITH DELIMITERS + operando => é usado p/ procurar um valor delimitado pelo caracter especificado.

```
* >Natural Source Header 000000..  
DEFINE DATA LOCAL  
1 #T (A40) INIT<'A-BC A B C .A. .B. .C. -A- -B-AB'>  
1 #P1 (N3)  
1 #P2 (N3)  
1 #P3 (N3)  
END-DEFINE  
  
/* AQUI O "A" É PESQUISADO NA VARIÁVEL #T E O  
/* N° DE OCORRENCIAS É ARMAZENADO NA VARIÁVEL  
/* #P1  
EXAMINE #T FOR 'A' GIVING NUMBER #P1  
  
/* WITH DELIMITERS => USADO P/ UM VALOR QUE ESTÁ  
/* ENTRE ESPAÇO EM BRANCO OU QUALQUER OUTRO QUE NAO  
/* SEJA LETRA OU NUMERO  
EXAMINE #T FOR 'A' WITH DELIMITERS GIVING NUMBER #P2  
  
/* AQUI SERÁ PESQUISADO O VALOR "A" QUE SE ENCONTRA  
/* ENTRE TRAÇOS(-)  
EXAMINE #T FOR 'A' WITH DELIMITERS '-' GIVING NUMBER #P3  
  
WRITE '=' #P1 / '=' #P2 / '=' #P3  
  
END
```

### Natural Web I/O Output

Page	1	23-05-24 16:00:03
#P1:	5	
#P2:	4	
#P3:	2	

## Examine Translate

TRANSLATE transforma caracteres de um campo p/ maiúsculo, p/ minúsculo ou p/ outro caracter especificado. INTO LOWER CASE, o conteúdo será transformado para minúsculo. INTO UPPER CASE, o conteúdo será transformado p/ maiúsculo.

A sintaxe EXAMINE ... TRANSLATE USING + operando, tem como valores de tradução os caracteres contidos no operando, que é uma tabela de tradução definida com formato (A2) ou (B2). Quando EXAMINE encontrar o primeiro caracter da tabela no campo, faz a troca do caracter do campo para o segundo caracter definido na tabela. A opção INVERTED altera a direção de troca no operando.

```
* >Natural Source Header 000000.
DEFINE DATA LOCAL
 1 #T (A40) INIT<'A-BC A B C .A. .B. .C. -A- -B-AB'>
 1 #Q (A40) INIT<'*-@# * @ # .* .@. .#. -* -@-*@'>
 1 #T1 (A2/3) INIT<'A*', 'B@', 'C#'>
 1 #NOME (A10) INIT<'JOCA TIGRE'>
END-DEFINE

/* TRANSLATE => TRANSFORMA CARACTERES DE UM CAMPO
/* P/ MAIUSCULO, P/ MINUSCULO OU P/ OUTRO CARACTER
/* ESPECIFICADO

/* AQUI A VARIABEL #T SERÁ EXAMINADA OS VALORES SERÃO
/* TRANSFORMADOS DE ACORDO COM O VETOR #T1
/* DE ACORDO COM O VETOR #T1, O "A" SERÁ TROCADO
/* POR ASTERISCO(*), O "B" POR ARROBA(@) E
/* ASSIM SUCESSIVAMENTE
WRITE '#T ANTES DO 1º EXAMINE: ' #T
EXAMINE #T TRANSLATE USING #T1(*)
WRITE '#T DEPOIS DO 1º EXAMINE: ' #T

SKIP 1

/* AQUI A VARIABEL #Q SOFRERÁ A TRANSFORMAÇÃO
/* INVERTENDO, ONDE TIVER (*) (@) (#) SERÁ A, B E C
WRITE '#Q ANTES DO 2º EXAMINE: ' #Q
EXAMINE #Q TRANSLATE USING INVERTED #T1(*)
WRITE '#Q DEPOIS DO 2º EXAMINE: ' #Q

SKIP 1

/* TRANSLATE INTO LOWER CASE TRANSFORMA A STRING
/* P/ MINUSCULO
WRITE '#NOME ANTES DO 3º EXAMINE: ' #NOME
EXAMINE #NOME TRANSLATE INTO LOWER CASE
WRITE '#NOME DEPOIS DO 3º EXAMINE: ' #NOME

END
```

## Natural Web I/O Output

```
Page      1                               23-05-25 10:44:20

#T ANTES DO 1º EXAMINE: -A-BC A B C .A. .B. .C. -A- -B-AB
#T DEPOIS DO 1º EXAMINE: -*-*@ # * @ # .*. .@. .#. -* -@-*@

#Q ANTES DO 2º EXAMINE: -*-*@ # * @ # .*. .@. .#. -* -@-*@
#Q DEPOIS DO 2º EXAMINE: -A-BC A B C .A. .B. .C. -A- -B-AB

#NOME ANTES DO 3º EXAMINE: JOCA TIGRE
#NOME DEPOIS DO 3º EXAMINE: joca tigre
```

## Separate

Separa o conteúdo de um campo alfanumérico em dois ou mais campos alfanuméricos ou ocorrências de um ARRAY alfanumérico.

```
+ * >Natural Source Header 000000..
```

DEFINE DATA LOCAL

```
1 #A (A30) INIT<'XXXXXXXXXXXXXXXXXX.YYYY.ZZZZZZZZ'>
1 #A1 (A15)
1 #A2 (A15)
1 #A3 (A15)
END-DEFINE
```

/\* SEPARTE => SEPARA O CONTEUDO DE UM CAMPO  
/\* ALFANUMERICO EM 2 OU MAIS CAMPOS ALFANUMERICOS  
/\* OU OCORRENCIAS DE UM ARRAY ALFANUMERICO

```
WRITE '=' #A
SKIP 1
```

/\* COLOCA TODOS OS "X" NA VARIABEL #A1  
/\* TODOS OS "Y" NA VARIABEL #A2  
/\* TODOS OS "Z" NA VARIABEL #A3

```
SEPARATE #A INTO #A1 #A2 #A3
```

```
WRITE '=' #A1 / '=' #A2 / '=' #A3
```

END|

## Natural Web I/O Output

Page 1

23-05-25 11:08:03

#A: XXXXXXXXXXXXXXXX.YYYY.ZZZZZZZZ

#A1: XXXXXXXXXXXXXXXX

#A2: YYYY

#A3: ZZZZZZZZ

```
* >Natural Source Header 000000..  
DEFINE DATA LOCAL  
1 #A (A32) INIT<'1111111111 222'>  
1 #A1 (A10) INIT<'JOCA'>  
1 #A2 (A10) INIT<'JOCA'>  
1 #A3 (A10) INIT<'JOCA'>  
END-DEFINE  
  
/* NESSE EXEMPLO TEMOS 3 VARIAVEIS QUE RECEBERÃO  
/* OS VALORES DA VARIAVEL #A, POREM NA VARIAVEL  
/* TEMOS APENAS 2 GRUPOS DE VALORES (1 e 2)  
/* O RESULTADO DISSO SERÁ QUE O VALOR DA 3ª VARIAVEL  
/* SERÁ ANULADA PELO NATURAL  
  
WRITE 'ANTES DO SEPARATE'  
WRITE '=' #A / '=' #A1 / '=' #A2 / '=' #A3  
  
SEPARATE #A INTO #A1 #A2 #A3  
  
SKIP 1  
  
WRITE 'APÓS O SEPARATE'  
WRITE '=' #A1 / '=' #A2 / '=' #A3  
END|
```

## Natural Web I/O Output

Page 1 23-05-25 11:20:41

ANTES DO SEPARATE

```
#A: 1111111111 222
#A1: JOCA
#A2: JOCA
#A3: JOCA
```

APÓS O SEPARATE

```
#A1: 1111111111
#A2: 222
#A3:
```

\* >Natural Source Header 000000..

DEFINE DATA LOCAL

```
1 #A (A20) INIT<'111..22222'>
1 #A1 (A20)
1 #A2 (A20)
1 #A3 (A20)
END-DEFINE
```

/\* AQUI TEMOS 2 DELIMITADORES CONSECUTIVOS
/\* OS DELIMITADORES SE TORNAM ESPAÇOS EM BRANCO

SEPARATE #A INTO #A1 #A2 #A3

WRITE '=' #A / '=' #A1 / '=' #A2 / '=' #A3

END|

## Natural Web I/O Output

Page 1 23-05-25 11:35:18

```
#A: 111..22222
#A1: 111
#A2:
#A3: 22222
```

```

+ * >Natural Source Header 000000..
 DEFINE DATA LOCAL
 1 #A (A20) INIT<'JOCA TIGRE'>
 1 #ARRAY (A10/2)
 END-DEFINE

 /* SEPARA O CONTEUDO DA VARIABEL #A PARA
 /* DENTRO DO ARRAY
 SEPARATE #A INTO #ARRAY(*)

 WRITE '=' #A / '=' #ARRAY(*)

END

```

### Natural Web I/O Output

Page	1	23-05-25 11:40:39
#A:	JOCA TIGRE	
#ARRAY:	JOCA	TIGRE

## Separate With Delimiters

Podemos separar o conteúdo a partir de um delimitador especificado.

```

+ * >Natural Source Header 000000..
 DEFINE DATA LOCAL
 1 #A (A32) INIT<'1111111111@2222222222@CCCCCCCCC'>
 1 #A1 (A10)
 1 #A2 (A10)
 1 #A3 (A10)
 END-DEFINE

 /* SEPARATE WITH DELIMITERS => separa o conteudo da
 /* variavel especificando um delimitador
 SEPARATE #A INTO #A1 #A2 #A3 WITH DELIMITERS '@'

 WRITE '=' #A // '=' #A1 / '=' #A2 / '=' #A3

END

```

## Natural Web I/O Output

Page 1

23-05-25 11:47:25

#A: 1111111111@2222222222@CCCCCCCCCC

#A1: 1111111111  
#A2: 2222222222  
#A3: CCCCCCCCCC

## Separate Giving Number

GIVING NUMBER fornece o número de campos resultantes diferentes de brancos.

```
+ * >Natural Source Header 000000..  
- DEFINE DATA LOCAL  
 1 #A (A32) INIT<'1111111111.22233.44555'>  
 1 #A1 (A10) INIT<'LUCIANO'>  
 1 #A2 (A10) INIT<'LUCIANO'>  
 1 #A3 (A10) INIT<'LUCIANO'>  
 1 #A4 (A10) INIT<'LUCIANO'>  
 1 #CONT (N3)  
END-DEFINE  
  
- /* GIVING NUMBER fornece o nº de ocorrências  
/* de campos diferentes de brancos  
WRITE 'ANTES DO SEPARATE'  
WRITE '=' #A / '=' #A1 / '=' #A2 / '=' #A3 / '=' #A4  
SEPARATE #A INTO #A1 #A2 #A3 #A4 GIVING NUMBER #CONT  
  
SKIP 1  
  
WRITE 'APOS O SEPARATE'  
WRITE '=' #A1 / '=' #A2 / '=' #A3 / '=' #A4 //  
  'NÚMEROS DE CAMPOS RESULTANTES:' #CONT  
  
END|
```

## Natural Web I/O Output

```
Page      1                               23-05-25 15:28:37

ANTES DO SEPARATE
#A: 1111111111.22233.44555
#A1: LUCIANO
#A2: LUCIANO
#A3: LUCIANO
#A4: LUCIANO

APOS O SEPARATE
#A1: 1111111111
#A2: 22233
#A3: 44555
#A4:

NÚMEROS DE CAMPOS RESULTANTES:   3
```

## Separate Left Justified

LEFT JUSTIFIED remove brancos à esquerda dos campos.

```
+ * >Natural Source Header 000000..
- DEFINE DATA LOCAL
 1 #A (A30) INIT<'111. 1111. 22233'>
 1 #A1 (A20)
 1 #A2 (A20)
 1 #A3 (A20)
END-DEFINE

- /* AQUI NO WRITE TEREMOS OS GRUPOS DE VALORES
/* MOSTRADOS ENTRE OS PONTOS(.)
/* NA VARIABEL #A1 TEREMOS '111'
/* NA VARIABEL #A2 TEREMOS ' 1111'
/* NA VARIABEL #A3 TEREMOS ' 22233'
/* NOTE QUE TEMOS ESPAÇOS EM BRANCO
SEPARATE #A INTO #A1 #A2 #A3 WITH DELIMITERS '.'
WRITE 'ALINHAMENTO SEM LEFT JUSTIFIED:' /
#A1 '#A1' / #A2 '#A2' / #A3 '#A3' //

/* LEFT JUSTIFIED => remove brancos a esquerda do campo
SEPARATE #A LEFT JUSTIFIED INTO #A1 #A2 #A3 WITH DELIMITERS '.'
WRITE 'ALINHAMENTO COM LEFT JUSTIFIED:' /
#A1 '#A1' / #A2 '#A2' / #A3 '#A3' //

END
```

## Natural Web I/O Output

Page 1

23-05-25 15:41:56

ALINHAMENTO SEM LEFT JUSTIFIED:

```
111          #A1
1111         #A2
22233        #A3
```

ALINHAMENTO COM LEFT JUSTIFIED:

```
111          #A1
1111         #A2
22233        #A3
```

## Separate Substring

A opção SUBSTRING permite que seja feito um SEPARATE em parte de um campo. Depois do nome do campo (operando1), especifica-se a posição inicial (operando2), e então o tamanho a pesquisar (operando3) da porção do campo.

```
+ * >Natural Source Header 000000..
④ DEFINE DATA LOCAL
 1 #A (A20) INIT<'11111.22222.33333'>
 1 #A1 (A5)
 1 #A2 (A5)
 1 #A3 (A5)
END-DEFINE

④ /* SEPARATE SUBSTRING => é realizado um separate em
/* parte de um campo
/* Depois da variavel, especifica-se a posicao inicial
/* e entao o tamanho da porcao a se pesquisar
/* Nesse exemplo sera pesquisado na porcao '11.222'
SEPARATE SUBSTRING(#A,4,6) INTO #A1 #A2 #A3

WRITE '=' #A // '=' #A1 / '=' #A2 / '=' #A3

END|
```

### Natural Web I/O Output

```
Page      1                               23-05-25  15:54:42
#A: 11111.22222.33333
#A1: 11
#A2: 222
#A3:
```

### Separate Ignore

Com a opção IGNORE o Natural ignorará se não houver campos suficientes para receber o resultado do SEPARATE.

```
* >Natural Source Header 000000..
DEFINE DATA LOCAL
1 #A (A20) INIT<'11111.22222.33333'>
1 #A1 (A5)
1 #A2 (A5)
END-DEFINE

/* SEPARATE IGNORE => se nao houver campos
/* suficientes, o Natural irá ignorar
/* O restante do valor da variavel #A('33333')
/* nao será armazenada em nenhuma variavel
SEPARATE #A INTO #A1 #A2 IGNORE

WRITE '=' #A // '=' #A1 / '=' #A2

END|
```

### Natural Web I/O Output

```
Page      1                               23-05-25  16:09:43
#A: 11111.22222.33333
#A1: 11111
#A2: 22222
```

## Separate Remainder

Com a opção REMAINDER operando, a seção de valor que não for carregada nos campos receptores, será carregada no operando. Este conteúdo do operando poderá ser utilizado, por exemplo, para um outro SEPARATE.

```
* >Natural Source Header 000000..  
DEFINE DATA LOCAL  
1 #A (A20) INIT<'RRRRR.00000.AAAAAA'>  
1 #A1 (A5)  
1 #A2 (A5)  
1 #R (A5)  
END-DEFINE  
  
/* SEPARATE REMAINDER => a porção que nao for  
/* armazenada nas variaveis será carregada na  
/* variavel #R apos o remainder  
SEPARATE #A INTO #A1 #A2 REMAINDER #R  
  
WRITE '=' #A // '=' #A1 / '=' #A2 / '=' #R  
  
END|
```

### Natural Web I/O Output

Page	1	23-05-25 16:20:30
 #A: RRRRR.00000.AAAAAA  #A1: RRRRR #A2: 00000 #R: AAAAAA		

## Separate Retained with Delimiters

Normalmente os caracteres delimitadores não são carregados nos campos receptores do SEPARATE. Com a opção WITH RETAINED DELIMITERS, o delimitador será também carregado no campo receptor.

```
* >Natural Source Header 000000
DEFINE DATA LOCAL
 1 #A (A20) INIT<'RRRRR.00000.AAAAAA'>
 1 #A1 (A5)
 1 #A2 (A5)
 1 #A3 (A5)
 1 #A4 (A5)
 1 #A5 (A5)
END-DEFINE

/* Normalmente os caracteres delimitadores não são
/* carregados nos campos receptores do SEPARATE
/* Com a opção WITH RETAINED DELIMITERS, o delimitador
/* será também carregado no campo receptor
SEPARATE #A INTO #A1 #A2 #A3 #A4 #A5 IGNORE WITH RETAINED DELIMITERS

WRITE '=' #A // '=' #A1 / '=' #A2 / '=' #A3 / '=' #A4 / '=' #A5

END
```

### Natural Web I/O Output

Page	1	23-06-06 14:14:05
 #A: RRRRR.00000.AAAAAA  #A1: RRRRR #A2: . #A3: 00000 #A4: . #A5: AAAAAA		

## Compress

A instrução COMPRESS serve para concatenar o conteúdo de dois ou mais campos em um outro campo. Os operandos a serem combinados podem ser de qualquer formato, porém o campo resultante deve ser alfanumérico. Pode-se utilizar as variáveis de sistema, como \*NUMBER ou \*COUNTER, diretamente no comando COMPRESS.

```
+* >Natural Source Header 000000.
DEFINE DATA LOCAL
 1 #A (A40) INIT<'Curso de Natural e Adabas - Dia'>
 1 #B (N2) INIT<10>
 1 #C (A50)
END-DEFINE

/* A instrução COMPRESS serve para concatenar o conteúdo
/* de dois ou mais campos em um outro campo. Os operandos
/* a serem combinados podem ser de qualquer formato, porém
/* o campo resultante deve ser alfanumérico. Pode-se
/* utilizar as variáveis de sistema, como *NUMBER ou *COUNTER,
/* diretamente no comando COMPRESS

COMPRESS #A #B TO #C LEAVING NO

WRITE '=' #A / '=' #B // '=' #C

END|
```

### Natural Web I/O Output

Page	1	23-06-06 14:30:29
#A: Curso de Natural e Adabas - Dia		
#B: 10		
#C: Curso de Natural e Adabas - Dia10		

```
* >Natural Source Header 000000..  
 DEFINE DATA LOCAL  
1 #NOME (A8) INIT<'Joca'>  
1 #SOBRENOME (A8) INIT<'Tigre'>  
1 #NOME_COMPLETO (A30)  
END-DEFINE  
  
/* PM=I => inverte os dados  
  
COMPRESS #NOME(PM=I) #SOBRENOME INTO #NOME_COMPLETO  
  
WRITE '=' #NOME / '=' #SOBRENOME // '=' #NOME_COMPLETO  
  
END
```

#### Natural Web I/O Output

Page	1	23-06-06 14:52:03
#NOME:	Joca	
#SOBRENOME:	Tigre	
#NOME_COMPLETO:	acoJ Tigre	

## Compress Delimiters

A opção DELIMITER(S) permite especificar um caractere delimitador entre cada valor no campo resultante. Se não for especificada a opção DELIMITERS(S) é colocado um espaço em branco entre os valores. Se for especificado DELIMITER(S) sem caractere algum, será assumido o caractere especificado no parâmetro ID='delimitador' do Natural, cujo valor default é "," (vírgula). Se um dos operandos do comando COMPRESS for numérico e tiver seu valor zerado, o zero aparecerá no campo resultante.

```
+ * >Natural Source Header 000000..  
 DEFINE DATA LOCAL  
 1 #DIA (N2) INIT<10>  
 1 #MES (N2) INIT<02>  
 1 #ANO (N4) INIT<2004>  
 1 #DATA (A10)  
 END-DEFINE  
  
/* DELIMITER(S) permite especificar um caractere delimitador  
/* entre cada valor no campo resultante  
COMPRESS #DIA #MES #ANO INTO #DATA WITH DELIMITERS '/'  
WRITE '=' #DIA / '=' #MES / '=' #ANO / '=' #DATA  
  
skip 1  
  
/* Se não for especificada a opção DELIMITERS, é colocado um  
/* espaço em branco entre os valores  
COMPRESS #DIA #MES #ANO INTO #DATA  
WRITE '=' #DIA / '=' #MES / '=' #ANO / '=' #DATA  
  
skip 1  
  
/* Se for especificado DELIMITERS sem caractere algum, será  
/* assumido o caractere do Natural, cujo valor default é ","  
COMPRESS #DIA #MES #ANO INTO #DATA WITH DELIMITERS  
WRITE '=' #DIA / '=' #MES / '=' #ANO / '=' #DATA  
  
skip 1  
  
/* Se um dos operandos do comando COMPRESS for numérico e  
/* tiver seu valor zerado, o zero aparecerá no campo resultante  
#DIA := 0  
COMPRESS #DIA #MES #ANO INTO #DATA WITH DELIMITERS  
WRITE '=' #DIA / '=' #MES / '=' #ANO / '=' #DATA  
  
END
```

## Natural Web I/O Output

Page	1	23-06-06 15:16:18
#DIA:	10	
#MES:	2	
#ANO:	2004	
#DATA:	10/2/2004	
#DIA:	10	
#MES:	2	
#ANO:	2004	
#DATA:	10 2 2004	
#DIA:	10	
#MES:	2	
#ANO:	2004	
#DATA:	10,2,2004	
#DIA:	0	
#MES:	2	
#ANO:	2004	
#DATA:	0,2,2004	

## Compress Leaving no Space

A opção LEAVING NO SPACE serve para suprimir o espaço em branco entre dois ou mais campos concatenados.

```
+ * >Natural Source Header 000000..  
④ DEFINE DATA LOCAL  
 1 #DIA (N2) INIT<10>  
 1 #MES (N2) INIT<02>  
 1 #ANO (N4) INIT<2004>  
 1 #DATA (A10)  
END-DEFINE  
  
④ /* LEAVING NO SPACE serve para suprimir o espaço  
/* em branco entre dois ou mais campos concatenados  
  
COMPRESS #DIA #MES #ANO INTO #DATA  
WRITE '=' #DIA / '=' #MES / '=' #ANO / '=' #DATA  
  
skip 1  
  
COMPRESS #DIA #MES #ANO INTO #DATA LEAVING NO  
WRITE '=' #DIA / '=' #MES / '=' #ANO / '=' #DATA  
  
END|
```

### Natural Web I/O Output

Page	1	23-06-06 15:26:26
#DIA:	10	
#MES:	2	
#ANO:	2004	
#DATA:	10 2 2004	
#DIA:	10	
#MES:	2	
#ANO:	2004	
#DATA:	1022004	

## Compress Numeric

NUMERIC inclui os caracteres de sinal e decimal na concatenação de valores.

```
+ * >Natural Source Header 000000..  
 DEFINE DATA LOCAL  
 1 #NUM1 (N3.2) INIT<-123.45>  
 1 #NUM2 (N2.2) INIT<-9.1>  
 1 #NUM3 (A12)  
 END-DEFINE  
  
/* NUMERIC inclui os caracteres de sinal e  
/* decimal na concatenação de valores  
  
COMPRESS #NUM1 #NUM2 INTO #NUM3 WITH DELIMITER ''  
WRITE 'COMPRESS SEM NUMERIC:' #NUM3  
  
SKIP 1  
  
COMPRESS NUMERIC #NUM1 #NUM2 INTO #NUM3 WITH DELIMITER ''  
WRITE 'COMPRESS COM NUMERIC:' #NUM3  
  
END
```

### Natural Web I/O Output

Page	1	23-06-07 10:19:14
COMPRESS SEM NUMERIC: 12345 910		
COMPRESS COM NUMERIC: -123.45 -9.1		

## Concatenacao com Hifen

```
+ * >Natural Source Header 000000
DEFINE DATA LOCAL
1 #S (A79)
END-DEFINE

MOVE 'Frase concatenada no' - ' programa Natural' TO #S

WRITE #S

END|
```

### Natural Web I/O Output

Page 1 23-06-07 10:23:51

Frase concatenada no programa Natural