



Romenol

Documentação completa para a linguagem de programação **Romenol**.

1. Tipos de Dados Básicos

1.1. Tipos Primitivos

- **intreg**: Representa valores inteiros.

```
intreg a, b, c
```

- **sir**: Utilizado para declarar arrays (vetores) de dados do tipo cadeia de caracteres, nomes, ou outros usos (a definição exata de "sir" pode variar conforme as convenções do programador).

```
sir nomes[10], sobrenomes[5]
```

- **real**: Representa números com parte decimal.

```
real pi
```

- **caracter**: Representa um único caractere.

```
caracter letra
```

- **sir**: Representa sequências de caracteres (texto).

```
sir saudacao
```

- **bool**: Representa valores lógicos (verdadeiro ou falso).

```
bool ativo
```

- **void**: Utilizado em funções para indicar que não há valor de retorno.

```
funcie void exemplo() {  
    // Sem valor de retorno.  
}
```

2. Declaração de Variáveis e Arrays

As variáveis são declaradas utilizando seus respectivos tipos seguidos dos identificadores. No caso dos arrays, a declaração inclui o nome da variável seguido do tamanho entre colchetes.

Exemplo de declaração:

```
intreg a, b, c    // Declara três variáveis inteiros  
sir nomes[10]     // Declara um array de 10 elementos do tipo sir
```

A indexação dos arrays é feita através dos colchetes, onde a contagem inicia em 0. Assim, `nomes[0]` seria o primeiro elemento e `nomes[9]` o décimo.

3. Estruturas de Controle

3.1. Instruções Condicionais

A linguagem usa `daca` para construir estruturas condicionais do tipo "if". O bloco opcional `altfel` substitui o "else" para condições não satisfeitas.

Exemplo:

```
daca (x == 10) {  
    scrie("Número é 10")  
}  
  
daca (x > 10) {  
    scrie("Maior que 10")  
} altfel {  
    scrie("Menor ou igual a 10")  
}
```

3.2. Estruturas de Loop

A Romenol possui várias formas de laços de repetição:

3.2.1. Loop "cattimp" (Enquanto)

O laço `cattimp` atua como um *while loop*. A condição é verificada antes de executar o bloco.

Exemplo:

```
cattimp (x < 10) {  
    scrie(x)  
    x = x + 1  
}
```

3.2.2. Loop "fa" ... "cattimp" (Faça..Enquanto)

Esta variação funciona como um *do-while loop*, onde o bloco é executado inicialmente e a condição é verificada ao final.

Exemplo:

```
fa {  
    scrie(x)
```

```
x = x + 1  
} cattimp (x < 10)
```

3.2.3. Loop "pentru" (Para)

O laço `pentru` é utilizado para repetições com um contador, muito similar ao típico *for loop*.

Exemplo:

```
intreg i  
pentru (i = 0; i < 5; i = i + 1) {  
    scrie(i)  
}
```

4. Funções

Na **Romenol**, funções são definidas utilizando o comando `functie`, seguido do tipo de retorno, nome da função e uma lista de parâmetros entre parênteses. O corpo da função é delimitado por chaves `{ } .`

Dentro da função, variáveis locais podem ser declaradas e manipuladas. Para retornar um valor, utiliza-se o comando `intoarce` .

Exemplo de função para somar dois números:

```
functie intreg soma(intreg a, intreg b) {  
    intreg resultado  
    resultado = a + b  
    intoarce resultado  
}
```

5. Entrada e Saída

A linguagem dispõe de comandos simples para interação com o usuário:

- **Entrada:**

`citest(variavel)` lê um valor e o armazena na variável especificada. Também é possível ler diretamente de arrays, como `citest(vetor[3])` .

- **Saída:**

`scrive(valor)` escreve o valor na tela. Pode ser utilizado tanto para variáveis quanto para literais (como strings).

Exemplo:

```
citest(a)  
scrive("Mensagem")
```

6. Operadores

A **Romenol** suporta os operadores aritméticos, relacionais e lógicos comuns:

- **Aritméticos:** `+`, `,`, `,`, `/`

Exemplo:

```
resultado = (x + y) * z / 2
```

- **Relacionais:** `==`, `>`, `<`, `!=`

Exemplo:

```
daca (x > 5) { ... }
```

- **Lógicos:** `&&` (E lógico), `||` (OU lógico)

Exemplo:

```
comparacao = (a > b) && (b < c || a != c)
```

7. Estrutura de um Programa

Um programa em **Romenol** possui duas partes principais:

1. **Declarações e Funções Globais:**

Aqui são definidas variáveis globais e funções para uso em todo o programa.

2. **Bloco Principal:**

Delimitado pelas palavras-chave `inceput` e `sfarsit`, este bloco contém a lógica principal da aplicação.

Exemplo de estrutura:

```
intreg a, b, c
sir nomes[10], sobrenomes[5]

// Função de soma
functie intreg soma(intreg a, intreg b) {
    intreg resultado
    resultado = a + b
    intoarce resultado
}

inceput
    intreg x
    // Outras instruções...
sfarsit
```

8. Exemplo Completo

Abaixo, um exemplo que compila diversos conceitos abordados:

```
// Declaração de variáveis e arrays globais
intreg a, b, c
sir nomes[10], sobrenomes[5]

// Declaração de função para somar dois números
functie intreg soma(intreg a, intreg b) {
    intreg resultado
    resultado = a + b
    intoarce resultado
}

inceput
    intreg x
    // Estrutura condicional: se x for maior que 5, escreve x
```

```

daca (x > 5) {
    scrie(x)
}

// Loop while pedindo para enquanto x for menor que 10
cattimp (x < 10) {
    scrie(x)
    x = x + 1
}

// Loop for que itera de 0 a 4
intreg i
pentru (i = 0; i < 5; i = i + 1) {
    scrie(i)
}

// Loop do-while: executa o bloco e depois verifica a condição
fa {
    scrie(x)
    x = x + 1
} cattimp (x < 10)

// Estrutura condicional com alternativa (else)
daca (x == 10) {
    scrie("Número é 10")
}
daca (x > 10) {
    scrie("Maior que 10")
} altfel {
    scrie("Menor ou igual a 10")
}

// Operações de entrada
citeste(a)
citeste(vetor[3])

// Operações de saída
scrie(a)

```

```
scrie(vetor[2])
scrie("Mensagem")

// Atribuições e chamadas de função
x = 10
vetor[1] = x + 5
resultado = funcao(4) // Supondo uma função predefinida ou outra função declarada
x = soma(5, 7) // Chamada da função soma

// Expressões com operadores
resultado = (x + y) * z / 2
comparacao = (a > b) && (b < c || a != c)
sfarsit
```

Este exemplo reúne conceitos de declaração de variáveis, funções, estruturas de controle, operações aritméticas e fluxo de entrada/saída, demonstrando de forma integrada a sintaxe da linguagem.