- **Propósito**

Hall é um interpretador de algoritmos. Ele tem por objetivo tornar o aprendizado de lógica de programação uma disciplina mais atraente. Hall possui um conjunto de instruções pré programadas que permite aos seus usuários executarem seus programas, os quais foram escritos em uma pseudo linguagem de programação. Para isso, ele possui um conjunto de palavras e instruções similares às encontradas nas linguagens de programação tradicionais.

Particularmente, sua sintaxe se assemelha à sintaxe da linguagem C, ou seja, o desenvolvimento dos algoritmos é baseado na divisão do programa em módulos, denominados funções, as quais são identificadas por parênteses **()** colocados após o seu nome, e, suas instruções são agrupadas em blocos, os quais são delimitados por chaves **{}** e, cada comando do bloco é finalizado com um ponto e vírgula **;**.

As linguagens mais modernas e poderosas tais como C++, Java, Delphi, Visual Basic, C++ Builder, etc, utilizam essa sintaxe ou variantes dela. Bom Estudo !

- **Nome**

A denominação **HALL** é uma homenagem ao computador da nave Discovery que cruza o espaço para descobrir a origem de um instrumento que seres superiores, oriundos de outro planeta, colocaram na Terra para policiar os homens. O Discovery, um prodígio da técnica é super comandada por um computador, na verdade HAL (Heuristically-programmed ALgorithmic computer) obra prima da geração de computadores.

Discovery e **HAL** na verdade, são personagens da Obra de Arthur Clark, 2001/Odisséia Espacial, onde o autor apresenta um quadro do futuro da humanidade, dominada pelos instrumentos da técnica que ela própria criou mas, cujo controle lhe escapa por entre os dedos, numa angustiante confusão entre realidade e futuro, tempo e espaço, inteligência e transcendência.

Hall também é uma homenagem a Herman Hollerith (1860-1929), funcionário do departamento de recenseamento dos EUA, criador do cartão perfurado e mais tarde, fundador da IBM. O censo americano de 1880 demorou 7 anos para ser concluído e todo processamento foi realizado manualmente.

Hall é o primeiro andar no edifício da programação de computadores. 2001 é o ano em que a nave Discovery parte para a sua jornada. Boa Viagem !

- ## Teoria

Pequena explanação sobre a arquitetura de Hall.

Um problema é resolvido através de um algoritmo, isto é, o espaço do problema é mapeado para o espaço do algoritmo fazendo-se abstrações em estruturas de dados e processos que manipulam essas estruturas de dados. As estruturas de dados são mapeadas em **variáveis** e os processos em **funções**.

Hall é **estruturado**, ou seja, ele permite a criação de **rotinas isoladas** contendo **variáveis locais**. Desse modo, os nossos algoritmos consistem em uma coleção de uma ou mais rotinas (funções) com variáveis locais. Uma função é composta de um nome, uma lista de parâmetros entre parênteses **()** e o bloco de código associado. O bloco começa com um abre chaves **{** e, é seguido de uma ou mais instruções finalizadas com ponto e vírgula, e termina em um fecha chaves **}**. Uma instrução começa com uma palavra reservada, tais como: **se**, **caso**, **repita**, **enquanto**, **para** ou então é uma expressão, tais como: **x := 10**, **z := z + 2**, **leia(k)**, **escreva (w)**.

Todos os nossos programas começam com uma chamada à função principal denominada **algoritmo()** e terminam com o último fecha chaves **}** dessa função. Quaisquer outras funções contidas no programa precisam ser chamadas direta ou indiretamente a partir da função principal. O fato de existir uma função principal é para informar ao interpretador **Hall** onde ele deve iniciar a execução do algoritmo.

- **Chamada**

Há duas formas de se chamar o interpretador Hall, as quais são exibidas quando se digita ***HALL*** na linha de comando do sistema operacional. As duas formas são as seguintes:

1. Hall ”arquivo.hal” e
2. Hall ?

A 1ª forma ***Hall ”arquivo.hal”*** é utilizada quando desejamos que o interpretador execute algum programa e a 2ªforma, ***hall ?,*** apenas exibirá a tela de direitos autorais do interpretador. Os programas fontes deverão possuir a extensão ***hal***, mas na chamada a execução a especificação da mesma é opcional, isto é, pode-se executar o programa simplesmente chamando-se Hall “arquivo”.

As chamadas poderão ser realizadas com letras maiúsculas, minúsculas ou ainda combinação de maiúsculas e minúsculas indiferentemente. A única restrição , é a de que não deve-se utilizar acentos, cedilha ou outros marcadores.

- **Palavras reservadas**

As palavras e operadores abaixo são reservados pelo interpretador Hall, e, portanto, não podem ser utilizadas como identificadores

1. Dados
   - Tipos ascii, cadeia, inteiro, real
   - Classes basico, vetor, matriz, registro, conjunto, arquivo

2. Operadores
   - Aritméticos + - * / % ^
   - Atribuição := = <-
   - Relacionais == # > >= < <=
   - Lógicos & | !
   - Membridade " ' . () {} []
   - Sequencia ,
   - Finalizador ;

3. Fluxo
   - Decisão se, senao, selecao, caso
   - Laço repita, ateque, enquanto, para, ate, passo
   - Salto retorne, interrompa, continue

- **Funções reservadas**

As palavras abaixo são reservados pelo interpretador Hall, e, portanto, não podem ser utilizadas como identificadores.

Funções de Interface

- Escreva()
- Leia()
- Posicao()
- Moldura()
- Pausa()
- Tecla()
- Direitos()
- CopyRight()

Funções Matemáticas

- Sinal()
- ValorAbs()
- PartInt()
- PartDec()
- Quociente()
- Resto()
- Raiz()
- Potencia()
- Seno()
- Coseno()
- Tangente()
- ArcoSeno ()
- ArcoCoseno ()
- ArcoTangente ()
- Exponencial()
- LogDec()
- LogNeper()
- Logaritmo ()

Funções de Cadeia

- Comprimento()
- Concatena()
- SubCadeia()
- Primeiros()
- Ultimos()
- Compara()
- Maiusculas()
- Minusculas()
- Insere()
- SobrePoe()
- CadeiaParaInteiro()
- CadeiaParaReal()
- InteiroParaCadeia()
- RealParaCadeia()

Fernando Paim
22/junho/98

Funções

- LimpaTela ( );
- Escreva ( );
- Leia ( );
- Imprima ( );
- Posicao ( );
- Linha ( );
- Coluna ( );
- Moldura ( );
- Pausa ( );
- Direitos ( );
- Copyright ( );
- Tecla ( );
- CorDoTexto ( );
- CorDoFundo ( );

Função LimpaTela()

```
algoritmo ()

{
	limpatela();
}
```

Comentário:

Função Escreva()

```
algoritmo ()

{
        // escreve "Hello, World" na tela.

        escreva ( "Hello, World" );
}
```

Comentário:

Função Posicao (<linha>, <coluna>)

```
algoritmo ()

{
	// escreve "Hello, World" na tela na posicao de linha 10 e coluna 15.

	posicao ( 10, 15 );
	escreva ( "Hello, World");
}
```

Comentário:

Função Moldura (<l1>, <c1>, <l2>, <c2>)

```
algoritmo ()

{
	// Faz uma moldura na tela nas coordenadas seguintes
	// canto superior esquerdo (5,3) e canto inferior direito (21,70)

	limpatela ();
	moldura ( 2, 5,20,50);
	moldura (10,10,12,40);
	moldura (14,20,18,30);
}
```

Comentário:

Função Direitos()

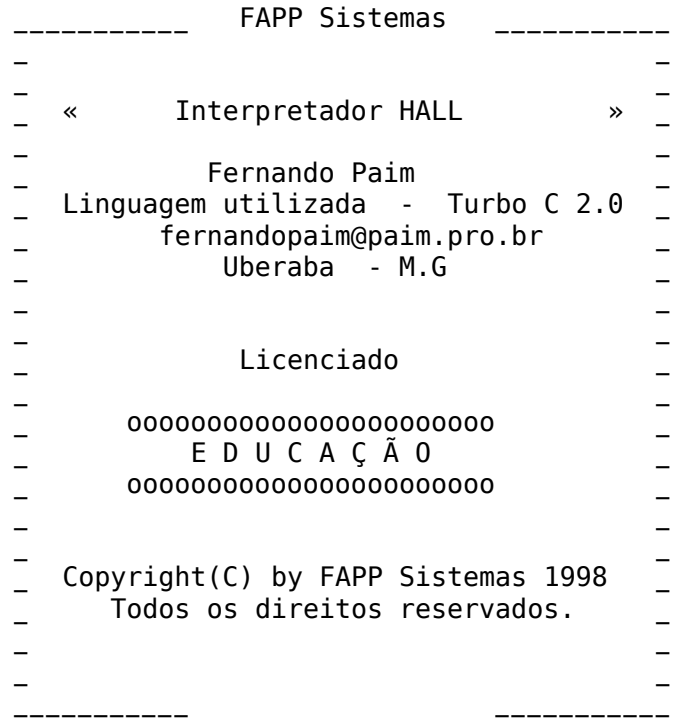
```
algoritmo ()

{
	// exibe a mensagem de direitos autorais do interpretador Hall.exe

	direitos ();
}
```

Comentário:

Exibe a tela de direitos autorais do interpretador HALL.EXE




```
C:\WINDOWS>
```

Função CopyRight()

```
algoritmo ()

{
        copyright ();
}
```

Comentário:

Exibe a mensagem de copyright do interpretador

Função Pausa()

```
algoritmo ()

{
        // exemplo da funcao interna pausa()

        escreva ( "pressione qualquer tecla para continuar" );

        pausa();

        escreva ("fim de algorimo");
}
```

Comentário:

Tipos de Dados

- Ascii
- Inteiro
- Real
- Cadeia

Tipos de dados aceitos interpretador

Fazer um algoritmo que declare variáveis, uma de cada tipo aceito pelo interpretador Hall e, ao final escreva "fim de declaração" na tela. Na frente de cada declaração coloque um comentário explicando-a.

```
algoritmo ()

{
        // declaracao dos tipos de variaveis aceitos pelo Hall

        ascii a;// declara uma variavel do tipo ascii
        inteiro i;          // declara uma variavel do tipo inteiro
        real r;  // declara uma variavel do tipo real
        cadeia c;         // declara uma variavel do tipo cadeia

        escreva ( "fim de declaracao" );

}
```

Comentário:

Operadores

- aritméticos + - * / % ^
- atribuição := = <-
- relacionais == # > >= < <=
- lógicos & | !
- membridade " ' . () {} []
- sequencia ,
- finalizador ;

- Operadores de atribuição aceitos pelo interpretador

Fazer um algoritmo que declare 4 variáveis do tipo inteiro e atribua valores a elas utilizando os 4 operadores de atribuição. Mostre o valor de cada uma na tela.

```
algoritmo ()

{
	// teste dos operadores de atribuicao

	inteiro x;	// declara uma variavel do tipo inteiro
	inteiro y;
	inteiro z;
	inteiro w;

	escreva ("testa os operadores de atribuicao");

	x = 10;
	y := 20;
	z <- 30;
	w ← 50;

	escreva ("o valor de x e: ", x);
	escreva ("o valor de y e: ", y);
	escreva ("o valor de z e: ", z);
	escreva ("o valor de w e: ", w);
}
```

Comentário:

- Chamada da função Moldura() com variáveis

```
algoritmo ()

{
	// teste da funcao moldura com variaveis

	inteiro a;
	inteiro b;
	inteiro c;
	inteiro d;

	a := 2;
	b := 5;
	c := 20;
	d := 50;

	moldura (a,b,c,d);

}
```

Comentário:

- Declaração de variáveis de tipos aceitos pelo interpretador

Fazer um algoritmo que atribua valores a cada um dos tipos aceitos pelo interpretador Hall e mostre os valores atribuídos na tela. Utilize as seguintes informações: nome da pessoa, sexo, idade em anos e salário.

```
algoritmo ()

{
	// teste de atribuicao de valores

	cadeia nome; // nome da pessoa ( tipo cadeia )
	ascii sexo;		// sexo da pessoa M/F ( tipo ascii )
	inteiro idade; // tipo inteiro (idade em anos da pessoa)
	real salario;	// valor do salario ( tipo real )

	nome := "Fernando Antonio";
	sexo := 'M';
	idade := 30;
	salario := 100;

	escreva ( "tipo cadeia - nome: ", nome );
	escreva ( "tipo ascii - sexo: ", sexo );
	escreva ( "tipo inteiro - idade: ", idade );
	escreva ( "tipo real - salario: ", salario );

}
```

Comentário:

- Funções Leia() e Escreva()

```
algoritmo ()

{
        // teste das funcoes leia() e escreva()

        inteiro i;                  // declara variavel do tipo inteiro
        inteiro j;
        inteiro k;
        inteiro p;

        escreva ( "testa entrada e saida de valores" );

        escreva ("digite o numero i: ");
        leia (i);
        escreva ("digite o numero j: ");
        leia (j);

        leia ("digite o numero k: ", k);
        leia ("digite o numero p: ", p);

        escreva ( "valor de i: ");
        escreva (i);
        escreva ( "valor de j: ");
        escreva (j);

        escreva ( "valor de k: ",k);
        escreva ( "valor de p: ",p);
}
```

Comentário:

Fazer um algoritmo que receba as seguintes informações de um usuário: o nome, o sexo, a idade em anos e o salário. Mostre os valores recebidos na tela.

```
algoritmo ()

{
	// teste de leitura de valores

	cadeia nome; // nome da pessoa ( tipo cadeia )
	ascii sexo;		// sexo da pessoa M/F ( tipo ascii )
	inteiro idade; // tipo inteiro (idade em anos da pessoa)
	real salario;	// valor do salario ( tipo real )

	leia ( "informe o nome: ", nome );
	leia ( "informe o sexo: ", sexo );
	leia ( "informe a idade: ", idade );
	leia ( "informe o salario: ", salario );

	escreva ( "nome informado: ", nome );
	escreva ( "sexo informado: ", sexo );
	escreva ( "idade informada: ", idade );
	escreva ( "salario informado: ", salario );

}
```

Comentário:

Fazer um algoritmo que receba as seguintes informações: nome da pessoa, sexo, idade em anos e valor do salário. Após receber essas informações exibi-las na tela e em seguida atribuir novos valores as mesmas, em seguida exibi-las novamente.

```
algoritmo ()

{
        // teste de leitura de valores

        cadeia nome; // nome da pessoa ( tipo cadeia )
        ascii sexo;              // sexo da pessoa M/F ( tipo ascii )
        inteiro idade; // tipo inteiro (idade em anos da pessoa)
        real salario;    // valor do salario ( tipo real )

        leia ( "informe o nome: ", nome );
        leia ( "informe o sexo: ", sexo );
        leia ( "informe a idade: ", idade );
        leia ( "informe o salario: ", salario );

        escreva ( "nome informado: ", nome );
        escreva ( "sexo informado: ", sexo );
        escreva ( "idade informada: ", idade );
        escreva ( "salario informado: ", salario );

        nome := "Fernando Antonio";
        sexo := 'M';
        idade := 30;
        salario := 100;

        escreva ( "nome atribuido: ", nome );
        escreva ( "sexo atribuido: ", sexo );
        escreva ( "idade atribuida: ", idade );
        escreva ( "salario atribuido: ", salario );

}
```

Comentário:

Fazer um algoritmo que receba dois números inteiros, calcule e mostre o resultado das operações aritméticas com tipos inteiros

```
algoritmo ()

{
        // teste dos operadores aritmeticos

        inteiro a;              // declara uma variavel do tipo inteiro
        inteiro b;              // declara uma variavel do tipo inteiro
        inteiro s;              // soma
        inteiro d;              // diferenca
        inteiro p;              // produto
        inteiro q;              // quociente
        inteiro t;              // potencia
        inteiro r;              // modulo (resto de divisao inteira)

        escreva ("testa os operadores aritmeticos");

        leia ("digite um numero a: ", a );

        leia ("digite um numero b: ", b );

        s := a + b;
        d := a - b;
        p := a * b;
        q := a / b;
        t := a ^ b;
        r := a % b;

        escreva ( "a soma e: ", s );
        escreva ( "a diferenca e: ", d );
        escreva ( "o produto e: ", p );
        escreva ( "o quociente e: " q );
        escreva ( "a potencia e: ", t );
        escreva ( "o resto da divisao inteira e: ", r );
}
```

Comentário:

Fazer um algoritmo que receba dois números inteiros, calcule e mostre o resultado das operações aritméticas com tipos reais.

```
algoritmo ()

{
	// teste dos operadores aritmeticos

	real a;		// declara uma variavel do real inteiro
	real b;		// declara uma variavel do real inteiro
	real s;		// soma
	real d;		// diferenca
	real p;		// produto
	real q;		// quociente
	real t;		// potencia
	real r;		// modulo (resto de divisao inteira)

	escreva ("testa os operadores aritmeticos com tipos reais");

	leia ("digite um numero a: ", a );

	leia ("digite um numero b: ", b );

	s := a + b;
	d := a - b;
	p := a * b;
	q := a / b;
	t := a ^ b;
	r := a % b;			// operacao não disponivel para tipos reais

	escreva ( "a soma e: ", s );
	escreva ( "a diferenca e: ", d );
	escreva ( "o produto e: ", p );
	escreva ( "o quociente e: " q );
	escreva ( "a potencia e: ", t );
	escreva ( "o resto da divisao inteira e: ", r );
}
```

Comentário:

Estruturas de Decisão

1. Se / Senao
2. Selecao / Caso

Instrução **se**

A instrução ***se*** é uma instrução de decisão. Ela faz com que um bloco de comandos seja executado dependendo se uma determinada condição de controle é verificada. Essa condição de controle é a instrução que permite desviar o fluxo de execução do programa.

A instrução <se> tem a seguinte sintaxe:

```
se (condicao)           ----> início da instrução
{       ----> abre chaves (início do bloco de instruções)
        instrução 1;
        instrução 2;        ----> instruções com ponto e vírgula
        instrução 3;
}       ----> fecha chaves (fim do bloco de instruções)
```

observação: a instrução ***se*** tem a instrução ***senao*** fazendo parte de sua estrutura. O bloco de comandos ***senao*** será executado se o teste da condição da instrução ***se*** não for verificado (resultar falso).

Instrucao **senao**

A instrução ***senao*** é uma instrução de decisão. Ela faz com que um bloco de comandos seja executado caso o teste da condição de controle da instrução ***se*** correspondente resulte falso.

A instrução ***se/senao*** tem a seguinte sintaxe:

```
se (condição)      ----> início da instrução se
{               ----> abre chaves (início do bloco de instruções)
        instruções; ----> instruções com ponto e vírgula
}               ----> fecha chaves (fim do bloco de instruções)
senao                       ----> início da instrução senao
{               ----> abre chaves (início do bloco de instruções)
        instruções; ----> instruções com ponto e vírgula
}               ----> fecha chaves (fim do bloco de instruções)
```

- Estrutura SE simples

```
algoritmo ()
{
	// teste da instrucao <se>

	inteiro x;			// declara uma variavel do tipo inteiro

	escreva ( "testa se um numero e maior que 10" );

	leia ( "digite um numero x: ", x );

	se ( x > 10 )
	{
		escreva ( "o numero informado e maior que 10 -> ", x );
	}
}
```

Comentário:

- Estrutura Se/Senao simples

```
algoritmo ()

{
	inteiro i;			/* declara uma variavel do tipo inteiro */

	escreva ("testa se um numero e maior ou, menor ou igual a 10");

	leia ("digite um numero: ", i );

	se ( i > 10 )
	{
		escreva ( "o numero informado e maior que 10" );
	}
	senao
	{
		escreva ( "o numero informado nao e maior que 10" );
	}
}
```

Comentário:

Fazer um algoritmo que receba a resposta do usuário a seguinte pergunta: "deseja continuar (s/n): " e se a resposta for "sim", escreva a seguinte mensagem "usuário confirmou" senão, escreva "confirmação negada".

```
algoritmo ()

{
	ascii resp;

	leia ("confirme (s/n) ", resp);

	escreva("a resposta foi ", resp);

	se ( resp == 's' )
	{
		escreva ("usuario confirmou");
	}
	senao
	{
		escreva ("confirmacao negada");
	}
}
```

Comentário:

- Função Tecla()

```
algoritmo ()

{
	// exemplo da funcao interna tecla()

	ascii x;

	escreva ( "pressione qualquer tecla para continuar" );

	x := tecla();

	se ( x == 'a' )
	{
		escreva ("a tecla digitada foi ", x);
		escreva ("voce digitou a primeira letra do alfabeto");
	}
	senao
	{
		escreva ("a tecla digitada foi ", x);
		escreva ("outra letra ...");
	}
}
```

Comentário:

- Estrutura Se/Senao/Se

```
algoritmo ()

{
	inteiro i;		/* declara uma variavel do tipo inteiro */

	escreva ("testa se um numero e maior ou, menor ou igual a 10");

	leia ("digite um numero: ", i );

	se ( i > 10 )
	{
		escreva ( "o numero informado e maior que 10" );
	}
	senao se ( i == 10 )
	{
		escreva ( "o numero informado e igual a 10" );
	}
	senao se ( i > 0 )
	{
		escreva ( "o numero informado e menor que 10" );
	}
	senao
	{
		escreva ( "o numero informado e negativo" );
	}

}
```

Comentário:

- Operadores relacionais com estrutura de decisão Se simples

```
algoritmo ()
{
	// teste dos operadores relacionais

	inteiro a;		// declara uma variavel do tipo inteiro
	inteiro b;		// declara uma variavel do tipo inteiro

	escreva ("testa os operadores relacionais");

	leia ("digite um numero a: ", a );

	leia ("digite um numero b: ", b );

	se ( a == b )
	{
		escreva ( "a e b sao iguais" );
	}
	se ( a != b )
	{
		escreva ( "a e b sao diferentes, por !=" );
	}
	se ( a <> b )
	{
		escreva ( "a e b sao diferentes, por <>" );
	}
	se ( a > b )
	{
		escreva ( "a e maior que b" );
	}
	se ( a < b )
	{
		escreva ( "a e menor que b" );
	}
}
```

Comentário:

- Operadores relacionais com estrutura de decisão Se/Senao/Se

```
algoritmo ()

{
	// teste dos operadores relacionais

	inteiro a;		// declara uma variavel do tipo inteiro
	inteiro b;		// declara uma variavel do tipo inteiro

	escreva ("testa os operadores relacionais");

	leia ("digite um numero a: ", a );

	leia ("digite um numero b: ", b );

	se ( a == b )
	{
		escreva ( "a e b sao iguais" );
	}
	senao se ( a != b )
	{
		escreva ( "a e b sao diferentes, por !=" );
	}
	se ( a <> b )
	{
		escreva ( "a e b sao diferentes, por <>" );
	}
	se ( a > b )
	{
		escreva ( "a e maior que b" );
	}
	senao se ( a < b )
	{
		escreva ( "a e menor que b" );
	}
	senao se ( a == b )
	{
		escreva ( "a e b sao iguais" );
	}

}
```

Comentário:

- Operadores relacionais com estrutura de decisão Se/Senao

```
algoritmo ()

{
	// teste dos operadores relacionais

	inteiro a;		// declara uma variavel do tipo inteiro
	inteiro b;		// declara uma variavel do tipo inteiro

	escreva ("testa os operadores relacionais");

	leia ("digite um numero a: ", a );

	leia ("digite um numero b: ", b );

	se ( a == b )
	{
		escreva ( "a e b sao iguais" );
	}
	senao
	{
		se ( a != b )
		{
			escreva ( "a e b sao diferentes, por !=" );
		}
		se ( a <> b )
		{
			escreva ( "a e b sao diferentes, por <>" );
		}
	}
	se ( a > b )
	{
		escreva ( "a e maior que b" );
	}
	senao
		se ( a < b )
		{
			escreva ( "a e menor que b" );
		}
		senao
		{
			se ( a == b )
			{
				escreva ( "a e b sao iguais" );
			}
		}
	}
}
```

Comentário:

Fazer um algoritmo para exibir o menor de três números informados pelo usuário:

```
algoritmo()
{
	// exibe o menor de tres numeros

	inteiro a, b, c, m;

	leia ( "informe o numero a: ", a );
	leia ( "informe o numero b: ", b );
	leia ( "informe o numero c: ", c );

	se ( a > b )
	{
		m := b;
	}
	senao
	{
		m := a;
	}
	se ( m > c )
	{
		m := c;
	}
	escreva ( "o menor e: ", m );
}
```

Comentário:

Fazer um algoritmo que receba três valores inteiros e verifica se eles formam um triângulo. (utilize a estrutura se/senao se)

```
algoritmo ()
{
	// verifica se tres valores formam um triangulo

	inteiro x;
	inteiro y;
	inteiro z;

	escreva ( "verifica se tres valores formam um triangulo" );
	leia ( "informe o numero x: ", x );
	leia ( "informe o numero y: ", y );
	leia ( "informe o numero z: ", z );

	// verifica se podem formar um triangulo

	se ( x > y + z )
	{
		escreva ( "x > y + z: ");
		escreva ( "nao formam um triangulo");
	}
	senao se ( y > x + z )
	{
		escreva ( "y > x + z: ");
		escreva ( "nao formam um triangulo");
	}
	senao se ( z > x + y )
	{
		escreva ( "z > x + y: ");
		escreva ( "nao formam um triangulo");
	}
}
```

Comentário:

Fazer um algoritmo que receba três valores inteiros e verifica se eles formam um triângulo. (utilize a estrutura se/senao)

```
algoritmo ()
{
	// verifica se tres valores formam um triangulo

	inteiro x;
	inteiro y;
	inteiro z;

	escreva ( "verifica se tres valores formam um triangulo" );
	leia ( "informe o numero x: ", x );
	leia ( "informe o numero y: ", y );
	leia ( "informe o numero z: ", z );

	// verifica se podem formar um triangulo

	se ( x > y + z )
	{
		escreva ( "x > y + z: ");
		escreva ( "nao formam um triangulo");
	}
	senao
	{
		se ( y > x + z )
		{
			escreva ( "y > x + z: ");
			escreva ( "nao formam um triangulo");
		}
		senao se ( z > x + y )
		{
			escreva ( "z > x + y: ");
			escreva ( "nao formam um triangulo");
		}
	}
}
```

Comentário:

Fazer um algoritmo para verificar qual é o tipo de triângulo formado com três valores recebidos pelo teclado

```
algoritmo ()
{
	// verifica qual é o tipo de triângulo formado

	inteiro x;
	inteiro y;
	inteiro z;

	escreva ( "verifica o tipo de triângulo formado" );
	leia ( "informe o numero x: ", x );
	leia ( "informe o numero y: ", y );
	leia ( "informe o numero z: ", z );

	// verifica o tipo de triangulo formado

	se ( x == y )
	{
		se ( x == z )
		{
			escreva ( "triangulo equilatero" );
		}
	}
	se ( x == y )
	{
		escreva ( "triangulo isoceles" );
	}
	se ( x == z )
	{
		escreva ( "triangulo isoceles" );
	}
	se ( y == z )
	{
		escreva ( "triangulo isoceles" );
	}
	se ( x != y )
	{
		se ( y != z )
		{
			escreva ( "triangulo escaleno" );
		}
	}
}
```

Comentário:

- Operador lógico ***e***

```
algoritmo ()

{
	// teste do operador e-logico <&>

	inteiro i;		// declara uma variavel do tipo inteiro
	inteiro j;		// declara uma variavel do tipo inteiro

	escreva ("testa o operador e-l¢gico <&>");

	leia ("digite um numero i: ", i);

	leia ("digite um numero j: ", j);

	se ( i > 10 & j > 10 )
	{
		escreva ( "os numeros informados sao maiores que 10" );
	}
	senao
	{
		escreva ( "um ou ambos sao menores que 10" );
	}
}
```

Comentário:

- Operador lógico ***ou***

```
algoritmo ()

{
	// teste do operador ou-l¢gico <|>

	inteiro i;		// declara uma variavel do tipo inteiro
	inteiro j;		// declara uma variavel do tipo inteiro

	escreva ("testa o operador ou-l¢gico <|>");

	leia ("digite um numero i: ", i);

	leia ("digite um numero j: ", j);

	se ( i > 10 | j > 10 )
	{
		escreva ( "um ou ambos os numeros informados sao maiores que 10" );
	}
	senao
	{
		escreva ( "ambos os numeros informados sao menores que 10" );
	}
}
```

Comentário:

- Função Tecla() com operador lógico ***ou***

```
algoritmo ()

{
	ascii c;		// a variavel c podera assumir apenas 0 a 255

	escreva ( "tecle uma letra: " );

	c := tecla();

	se ( c == 'A' | c == 'a' )
	{
		escreva ( "voce digitou a primeira letra do alfabeto");
		escreva ( c );
	}
}
```

Comentário:

- Operador lógico ***e*** com teste de três condições

```
algoritmo ()

{
	// teste do operador <e> lógico <&>

	inteiro i;
	inteiro j;
	inteiro k;

	escreva ("testa o operador <e> lógico <&>");

	leia ("digite um numero i: ", i);
	leia ("digite um numero j: ", j);
	leia ("digite um numero k: ", k);

	se ( ( i > 10 & j == 10 ) & k < 5 )
	{
		escreva ( "os números informados estão ok" );
	}
	senao
	{
		escreva ( "os números informados estão fora da faixa" );
	}
}
```

Comentário:

- Operador lógico **ou** com teste de três condições

```
algoritmo ()

{
	// teste do operador <ou> lógico <|>

	inteiro i;
	inteiro j;
	inteiro k;

	escreva ("testa o operador <ou> lógico <|>");

	leia ("digite um numero i: ", i);
	leia ("digite um numero j: ", j);
	leia ("digite um numero k: ", k);

	se ( ( i > 10 | j == 10 ) | k < 5 )
	{
		escreva ( "os numeros informados estao ok" );
	}
	senao
	{
		escreva ( "os numeros informados estao fora da faixa" );
	}
}
```

Comentário:

- Operadores lógicos **e/ou** em testes de condições

```
algoritmo ()

{
	// teste do operador ou lógico <|>

	inteiro i;
	inteiro j;
	inteiro k;

	escreva ("testa o operador ou lógico <|>");

	leia ("digite um numero i: ", i);
	leia ("digite um numero j: ", j);
	leia ("digite um numero k: ", k);

	se ( ( i > 10 | j == 10 ) & k < 5 )
	{
		escreva ( "os números informados estão ok" );
	}
	senao
	{
		escreva ( "os números informados estão fora da faixa" );
	}
}
```

Comentário:

- Operadores lógicos **e/ou** em teste de condições com a estrutura de decisão **Se/Senao**

```
algoritmo ()

{
	// teste dos operadores e lógico (&) e ou lógico (|)

	inteiro i;		// declara uma variável do tipo inteiro
	inteiro j;		// declara uma variável do tipo inteiro

	leia ("digite um numero i: ", i);

	leia ("digite um numero j: ", j);

	se ( i > 10 & j > 10 )
	{
		escreva ( "os números informados são maiores que 10" );
	}
	senao
	{
		se ( i > 10 | j > 10 )
		{
			escreva ( "i ou j , maior que 10" );
		}
		senao
		{
			se ( i < 10 & j < 10 )
			{
				escreva ( "i e j são menores que 10");
			}
			senao
			{
				se ( i == 10 & j == 10 )
				{
					escreva ( "i e j são iguais a 10");
				}
				senao
				{
					se ( i == 10 | j == 10 );
					{
						escreva ( "i ou j , igual a 10");
					}
				}
			}
		}
	}
}
```

Comentário:

Fazer um algoritmo para verificar qual é o tipo de triângulo formado com três valores recebidos pelo teclado

```
algoritmo ()
{
	// verifica qual é o tipo de triângulo formado

	inteiro x;
	inteiro y;
	inteiro z;

	escreva ( "verifica o tipo de triângulo formado" );
	leia ( "informe o numero x: ", x );
	leia ( "informe o numero y: ", y );
	leia ( "informe o numero z: ", z );

	// verifica o tipo de triangulo formado

	se ( x == y & x == z )
	{
		escreva ( "triangulo equilatero" );
	}
	senao
	{
		se ( ( x == y | x == z ) | y == z )
		{
			escreva ( "triangulo isoceles" );
		}
		senao
		{
			escreva ( "triangulo escaleno" );
		}
	}
}
```

Comentário:

Fazer um algoritmo para verificar qual é o tipo de triângulo formado com três valores recebidos pelo teclado. (2ª versão)

```
algoritmo ()
{
	// verifica se três valores formam um triângulo

	inteiro x;
	inteiro y;
	inteiro z;

	escreva ( "verifica se três valores formam um triângulo" );
	leia ( "informe o numero x: ", x );
	leia ( "informe o numero y: ", y );
	leia ( "informe o numero z: ", z );

	// verifica se podem formar um triângulo

	se ( ( x < y + z & y < x + z ) & z < x + y )
	{
		// verifica o tipo de triângulo formado

		se ( x == y & x == z )
		{
			escreva ( "triângulo equilátero" );
		}
		senao
		{
			se ( ( x == y | x == z ) | y == z )
			{
				escreva ( "triângulo isóceles" );
			}
			senao
			{
				escreva ( "triângulo escaleno" );
			}
		}
	}
	senao
	{
		escreva ( "não formam triângulo");
	}
}
```

Comentário:

- Simulação de tipo de dado lógico. (1º modo)

```
algoritmo ()

{
	// tipo de dado lógico

	inteiro i;			// declara uma variável do tipo inteiro
	inteiro j;

	escreva ("simula o tipo de dado lógico");

	leia ("digite um numero i: ",i);

	se (i)
	{
		escreva ( "você informou um numero diferente de zero" );
	}
	senao
	{
		escreva ( "você informou o valor zero" );
	}
}
```

Comentário:

- Simulação de tipo de dado lógico. (2º modo)

```
algoritmo ()

{
	// tipo de dado lógico

	inteiro i;		// declara uma variável do tipo inteiro
	inteiro j;

	escreva ("tipo de dado lógico");

	leia ("digite um numero i: ",i);

	j := ( i > 10 );

	se (j)
	{
		escreva ( "o valor informado e maior que 10" );
	}
	senao
	{
		escreva ( "o valor informado não e maior que 10" );
	}
}
```

Comentário:

```
algoritmo()
{
	// teste do operador não lógico

	inteiro i;			// declara uma variável do tipo inteiro

	escreva ("testa o operador <!>");

	leia ("digite um numero i: ", i);

	se ( ! i )
	{
		escreva ( "o numero informado foi zero" );
	}
	senao
	{
		escreva ( "o numero informado e diferente de zero" );
	}
}
```

Comentário:

Instrução selecao

A instrução ***selecao*** é uma instrução de decisão. Ela é similar a instrução ***se***, permitindo desvios múltiplos de uma forma mais elegante. A instrução testa vários ***casos*** e executa apenas aquele que for verdadeiro.

A instrução ***selecao*** tem a seguinte sintaxe:

```
Selecao     -----> início da instrução selecao
{
        caso (condição 1)
        {                       -----> início do bloco
                instruções; -----> instruções do bloco do <caso1>
        {                       -----> fim do bloco
        caso (condição 2)
        {                       -----> início do bloco
                instruções; -----> instruções do bloco do <caso2>
        {                       -----> fim do bloco
        ...
}                               -----> fim da instrução selecao
```

observação: a instrução ***selecao*** tem a instrução ***senao*** fazendo parte de sua estrutura. O bloco de comandos ***senao*** será executado se nenhum dos ***casos*** for verificado, ou seja, o teste deles resultar falso.

Instrução caso

A instrução ***caso*** é uma instrução de decisão. Ela é parte integrante da instrução ***selecao*** e não pode aparecer em um programa como uma instrução isolada. A instrução ***caso*** permite realizar um teste de condição similar a instrução ***se***.
A instrução ***caso*** aparece na estrutura ***selecao*** com a seguinte sintaxe:

```
Selecao                 ----> início da instrução <selecao>
{
        caso (condição 1)       ----> teste da condição 1
        {
                instruções;     ----> instruções do bloco
        {
        caso (condição 2)       ----> teste da condição 2
        {
                instruções;     ----> instruções do bloco
        {
        ...
}                               ----> fim da instrução <selecao>
```

observação: a instrução ***caso*** realiza o teste da ***condição*** e, aquele que resultar verdadeiro, causará a execução do bloco de comandos associado. O bloco de comandos associados está colocado entre chaves {}.

- Estrutura de decisão Selecao/Caso simples

```
algoritmo ()
{
	inteiro i;			/* declara uma variável do tipo inteiro */

	escreva ("testa se um numero e igual, maior, ou menor que 10");

	escreva ("digite um numero: ");

	leia (i);		/* le o valor informado pelo teclado */

	selecao
	{
		caso ( i == 10 )
		{
			escreva ( "o numero informado e igual a 10" );
		}
		caso ( i > 10 )
		{
			escreva ( "o numero informado e maior que 10" );
		}
		caso ( i < 10 )
		{
			escreva ( "o numero informado e menor que 10" );
		}
	}
}
```

Comentário:

- Estrutura Selecao/Caso executa apenas um único caso verdadeiro

```
algoritmo ()
{
	inteiro i;		/* declara uma variavel do tipo inteiro */

	escreva ("testa se um numero e igual, maior, ou menor que 10");

	leia ("digite um numero: ", i );

	selecao
	{
		caso ( i == 10 )
		{
			escreva ( "o numero informado e igual a 10" );
		}
		caso ( i > 10 )
		{
			escreva ( "o numero informado e maior que 10" );
		}
		caso ( i > 20 )
		{
			escreva ( "o numero informado e maior que 20" );
		}
		caso ( i < 10 )
		{
			escreva ( "o numero informado e menor que 10" );
		}
	}
}
```

Comentário:

- Estrutura de decisão Selecao/Caso sequenciais.

```
algoritmo ()

{
	inteiro i;	/* declara uma variavel do tipo inteiro */
	inteiro j;	/* declara uma variavel do tipo inteiro */

	escreva ("testa se um numero e igual, maior, ou menor que 10");

	leia ( "digite um numero i: ", i );
	leia ( "digite um numero j: ", j );

	selecao
	{
		caso ( i == 10 )
		{
			escreva ( "i = 10" );
		}
		caso ( i > 10 )
		{
			escreva ( "i > 10" );
		}
		caso ( i < 10 )
		{
			escreva ( "i < 10" );
		}
	}
	selecao
	{
		caso ( j == 10 )
		{
			escreva ( "j = 10" );
		}
		caso ( j > 10 )
		{
			escreva ( "j > 10" );
		}
		caso ( j < 10 )
		{
			escreva ( "j < 10" );
		}
	}
}
```

Comentário:

- Operadores lógicos com a estrutura Selecao/Caso

```
algoritmo ()

{
	// teste dos operadores e lógico (&) e ou lógico (|)

	inteiro i;		// declara uma variável do tipo inteiro
	inteiro j;		// declara uma variável do tipo inteiro

	leia ("digite um numero i: ", i);

	leia ("digite um numero j: ", j);

	selecao
	{
		caso ( i > 10 & j > 10 )
		{
			escreva ( "os números informados são maiores que 10" );
		}
		caso ( i > 10 & j == 10 )
		{
			escreva ( "i e maior que 10 e j e igual a 10" );
		}
		caso ( i > 10 & j < 10 )
		{
			escreva ( "i e maior que 10 e j e menor que 10");
		}
		caso ( i == 10 & j == 10 )
		{
			escreva ( "i e j são iguais a 10");
		}
		caso ( i == 10 & j > 10 )
		{
			escreva ( "i e igual a 10 e j e maior que 10");
		}
		caso ( i == 10 & j < 10 )
		{
			escreva ( "i e igual a 10 e j e menor que 10");
		}
	}
}
```

Comentário:

- Estrutura Selecao/Caso aninhadas

```
algoritmo ()

{
	inteiro i;		/* declara uma variável do tipo inteiro */
	inteiro j;		/* declara uma variável do tipo inteiro */

	escreva ("testa se um numero e igual, maior, ou menor que 10");

	escreva ("digite um numero i: ");

	leia (i);	/* le o valor informado pelo teclado */

	escreva ("digite um numero j: ");

	leia (j);	/* le o valor informado pelo teclado */

	selecao
	{
		caso ( i == 10 )
		{
			selecao
			{
				caso ( j == 10 )
				{
					escreva ( "i = 10, j = 10" );
				}
				caso ( j > 10 )
				{
					escreva ( "i = 10, j > 10" );
				}
				caso ( j < 10 )
				{
					escreva ( "i = 10, j < 10" );
				}
			}
		}
		caso ( i > 10 )
		{
			selecao
			{
				caso ( j == 10 )
				{
					escreva ( "i > 10, j = 10" );
				}
				caso ( j > 10 )
				{
					escreva ( "i > 10, j > 10" );
				}
				caso ( j < 10 )
				{
					escreva ( "i > 10, j < 10" );
				}
			}
		}
		caso ( i < 10 )
		{
```

```
                        selecao
                        {
                                caso ( j == 10 )
                                {
                                        escreva ( "i < 10, j = 10" );
                                }
                                caso ( j > 10 )
                                {
                                        escreva ( "i < 10, j > 10" );
                                }
                                caso ( j < 10 )
                                {
                                        escreva ( "i < 10, j < 10" );
                                }
                        }
                }
        }
}
```

Comentário:

Estruturas de repetição

- Repita AteQue
- Repita Enquanto
- Repita
- Enquanto
- Faca
- Para
- For

Instrução repita ateque

A instrução ***repita ateque*** é uma instrução de laço. Ela faz com que um bloco de comandos seja executado repetidamente **até que** uma determinada condição de controle seja verificada. Essa condição de controle é a instrução que controla até quando o bloco de comandos delimitados pelas chaves {} será executado.

O laço ***repita ateque*** tem a seguinte sintaxe:

```
repita ateque (condicao)          ------> início do laço
{                                 ------> início do bloco de instruções
      instrução 1;
      instrução 2;                ------> instruções com ponto e vírgula
      instrução 3;
...
}                                 ------> fim do bloco de instruções
```

- Laço Repita ateque

```
algoritmo ()

{
	inteiro j;
	j := 0;

	escreva ("exibe os numeros de 0 a 10");
	escreva ("utiliza o laco repita");

	repita ateque ( j > 10 )
	{
		escreva (j);
		j := j + 1;
	}
}
```

Comentário:

- Laço Repita ateque aninhado

```
algoritmo ()

{
	inteiro i;
	inteiro j;

	escreva ("utiliza o laco repita / do");

	i := 1;
	j := 1;

	repita ateque ( i > 3 )
	{
		escreva (i);
		j := 1;
		repita ateque ( j > 3 )
		{
			escreva(j);
			j := j + 1;
		}
		i := i + 1;
	}
}
```

Comentário:

Exibir os números e seus quadrados

```
algoritmo ()

{
	// exibe os números e seus quadrados

	inteiro n;		// o numero
	inteiro q;		// quadrado do numero

	escreva ( "exibe os números de 0 a 10 e seus quadrados" );

	n := 0;

	repita ateque ( n > 10 )
	{
		q := n * n;
		escreva ( "número: ", n, "quadrado: ", q );
		n := n + 1;
	}
}
```

Comentário:

Cálculo da soma acumulada

```
algoritmo ()

{
	inteiro primeiro;
	inteiro ultimo;
	inteiro ValorAtual;
	inteiro ValorAnterior;
	inteiro SomaAcumulada;
	inteiro SomaAtual;

	primeiro := 1;
	ultimo := 10;
	ValorAnterior := primeiro;
	ValorAtual := primeiro;
	SomaAcumulada := 0;
	SomaAtual := 0;

	repita ateque ( ValorAtual > ultimo )
	{
		SomaAtual := SomaAcumulada + ValorAtual;
		SomaAcumulada := SomaAtual;
		ValorAnterior := ValorAtual;
		ValorAtual := ValorAnterior + 1;
	}
	escreva ( "o valor da soma e ", SomaAtual );
}
```

Comentário:

Calcular a média de n números lidos

```
algoritmo ()

{
	// calcula a media de n números

	real s; // soma dos números
	real n; // quantidade de números lidos
	real i;  // numero lido
	real m;		// media calculada

	s := 0;
	n := 0;
	i := 1;

	repita ateque ( i == 0 )
	{
		leia ("digite um numero ou 0 p/ sair: ", i);
		s := s + i;
		n := n + 1;
	}
	n := n - 1;
	m := s / n;
	escreva ( "a soma dos números foi ..: ", s );
	escreva ( "o numero de termos foi ..: ", n );
	escreva ( "a media e .....: ", m );
}
```

Comentário:

Totaliza somatório

```
algoritmo ()

{
	// totaliza uma serie da forma s = 1 + 1/2 + 1/3 + 1/4 + ... + 1/n

	real s;  // soma dos termos
	real t;  // um termo da serie
	inteiro n;		// limite dos numeros
	inteiro i;		// contador para n

	s := 0;
	i := 1;

	leia ("digite o limite da serie: ", n);

	repita ateque ( i > n )
	{
		t := 1 / i;
		s := s + t;
		i := i + 1;
	}
	escreva ( "a soma e: ", s );
}
```

Comentário:

Somatório de séries

```
algoritmo ()

{
	// totaliza uma serie da forma s = 1/n + 2/n-1 + 3/n-2 + ... + n/1

	real s; // soma da serie
	inteiro num;   // numerador do termo
	inteiro den;    // denominador do termo
	inteiro n;        // numero lido

	s := 0;

	leia ("digite o limite da serie: ", n);

	repita ateque ( num > n )
	{
		den := n - num + 1;
		s := s + num/den;
		num := num + 1;
	}

	escreva ( "a soma e: ", s );
}
```

Comentário:

Cálculo da função seno através de somatório

```
algoritmo ()

{
	// calcula a função seno através de somatórios

	real A; // angulo que se deseja calcular o seno
	real S; // valor do seno do arco A
	real T1;
	real T2;
	real T3;

	escreva ("calcula a função seno através da serie de MAC-LAURIM");

	A := 0;

	repita ateque ( A > 45 )
	{
		T1 := A^3 / 6;
		T2 := A^5 / 120;
		T3 := A^7 / 5040;
		S := S - T1 + T2 - T3;
		S := S * 3.141592 / 180.0;
		escreva ( "valor de A: ", A );
		escreva ( "valor do seno: ", S );
		A := A + 1;
	}
}
```

Comentário:

Gera os números de Fibonacci

```
algoritmo()
{
	// exibe os números de fibonacci ate 20º termo

	inteiro a;
	inteiro b;
	inteiro c;
	inteiro n;

	a := 0;
	b := 1;

	repita ateque ( n > 20 )
	{
		c := a + b;
		escreva (c);
		a := b;
		b := c;
		n := n + 1;
	}
}
```

Comentário:

Números de Fibonacci

```
algoritmo()
{
	// exibe os números de fibonacci menores que 1000

	inteiro a;
	inteiro b;
	inteiro c;

	a := 1;
	b := 1;

	repita ateque ( c > 1000 )
	{
		c := a + b;
		escreva (c);
		a := b;
		b := c;
	}
}
```

Comentário:

Fatorial de um número

```
algoritmo ()
{
	// calcula o fatorial de um numero

	inteiro n;		// o numero
	inteiro f;		// fatorial do numero

	escreva ( "calcula o fatorial de um numero" );
	escreva ( "informe o numero : " );
	leia (n);

	f := 1; // elemento neutro da multiplicação

	repita ateque ( n == 0 )
	{
		f := f *n;			// fat(n) = n(n-1)(n-2)(n-3)...
		n := n - 1;
	}
	escreva ( "o valor do fatorial e: " );
	escreva (f);
}
```

Comentário:

Capicuas são números que tem o mesmo valor se lidos da esquerda para direita ou da direita para esquerda. Exemplo: 44, 232, etc. Fazer um algoritmo que determine e escreva todos os números inteiros menores que 10000 que são capicuas.

```
algoritmo()
{
	inteiro N;			// numero gerado
	inteiro Unidade;
	inteiro Dezena;
	inteiro Centena;
	inteiro Milhar;

	N := 1000;
	repita ateque ( N > 3000 )
	{
		Milhar := N / 1000;
		Centena := ( N - Milhar*1000 ) / 100;
		Dezena := ( N - Milhar*1000 - Centena*100 ) / 10;
		Unidade := N - Milhar*1000 - Centena*100 - Dezena*10;
		se ( Milhar == Unidade & Centena == Dezena )
		{
			escreva ( "capicua: ", N );
		}
		N := N + 1;
	}
}
```

Comentário:

Números capícuas e quadrados perfeitos

```
algoritmo()

{
	// 3025, 30+25 = 55 e 55ý = 3025

	inteiro N, A, B, C;

	N := 1000;
	repita ateque ( N > 9999 )
	{
		A := N / 100;
		A := pint (A);
		B := N - ( A * 100 );
		C := A + B;
		C := C ^ 2;
		se ( C == N )
		{
			escreva(N);
		}
		N := N + 1;
	}
}
```

Comentário:

Num frigorífico existem 90 bois. Cada boi traz preso em seu pescoço um cartão contendo seu número de identificação e seu peso. Fazer um algoritmo que escreva o número e peso do boi mais gordo e do boi mais magro.

```
algoritmo()
{
	inteiro NumeroBois;  // numero de bois
	real PesoMaisGordo; // peso do boi mais gordo
	real PesoMaisMagro; // peso do boi mais magro
	inteiro NumeroBoi;    // numero de identificacao do boi
	real PesoBoi;           // peso do boi
	inteiro NumeroMaisGordo;   // identificacao do boi mais gordo
	inteiro NumeroMaisMagro;   // identificacao do boi mais magro

	NumeroBois := 0;
	PesoMaisGordo := 0;
	PesoMaisMagro := 2000;

	repita ateque ( NumeroBois > 10 )
	{
		leia ( "informe o numero de identificacao: ", NumeroBoi );
		leia ( "informe o peso do boi: ", PesoBoi );
		se ( PesoBoi > PesoMaisGordo )
		{
			PesoMaisGordo := PesoBoi;
			NumeroMaisGordo := NumeroBoi;
		}
		se ( PesoBoi < PesoMaisMagro )
		{
			PesoMaisMagro := PesoBoi;
			NumeroMaisMagro := PesoBoi;
		}
		NumeroBois := NumeroBois + 1;
	}
	escreva ( "Numero do boi mais gordo: ", NumeroMaisGordo );
	escreva ( "Peso do boi mais gordo: ", PesoMaisGordo );
	escreva ( "Numero do boi mais magro: ", NumeroMaisMagro );
	escreva ( "Peso do boi mais magro: ", PesoMaisMagro );
}
```

Comentário:

Supondo que a população de um país A seja da ordem de 90000000 habitantes com uma taxa anual de crescimento de 3% e que a população de uma país B seja aproximadamente de 200000000 de habitantes com uma taxa de crescimento anual de 1.5%, fazer um algoritmo que calcule e escreva o numero de anos necessários para que a população do país A ultrapasse ou iguale a população do pais B, mantidas as taxas atuais de crescimento.

```
algoritmo()
{
	inteiro PopA;  // populacao do pais A
	inteiro PopB;  // populacao do pais B
	real TaxaCrescA;		// taxa de crescimento do pais A
	real TaxaCrescB;		// taxa de crescimento do pais B
	inteiro NumAnos;		// numero de anos para A -> B

	PopA := 90000000;
	PopB := 200000000;
	TaxaCrescA := 0.03;
	TaxaCrescB := 0.015;
	NumAnos := 0;

	repita ateque ( PopA >= PopB )
	{
		PopA := PopA + PopA * TaxaCrescA;
		PopB := PopB + PopB * TaxaCrescB;
		NumAnos := NumAnos + 1;
	}
	escreva ( "populacao do pais A: ", PopA );
	escreva ( "populacao do pais B: ", PopB );
	escreva ( "tempo decorrido (anos): ", NumAnos );
}
```

Comentário:

Laço repita com teste no início aninhados

```
algoritmo ()

{
	inteiro i;
	inteiro j;
	i := 0;
	j := 0;

	escreva ("utiliza o laço repita aninhado");

	repita ateque ( i > 5 )
	{
		j := 0;
		repita ateque ( j > 5 )
		{
			escreva (i);
			escreva (j);
			j := j + 1;
		}
		escreva ( "proximo =>" );
		i := i + 1;
	}
}
```

Comentário:

Instrução Repita Enquanto

A instrução ***enquanto*** é uma instrução de laço. Ela faz com que um bloco de comandos (instruções entre pares de chaves) seja executado repetidamente enquanto uma determinada condição de controle for verificada. Essa condição de controle é a instrução que controla até quando o bloco de comandos entre as chaves, o corpo do laço, será executado.

O laço ***repita enquanto*** tem a seguinte sintaxe:

```
Repita enquanto (condição)          ----> início do laço
{                                   ----> (início do bloco de instruções)
      instrução 1;
      instrução 2;                  ----> instruções com ponto e vírgula
      instrução 3;

}                                   ----> (fim do bloco de instruções)
```

observação: a instrução ***repita enquanto*** testa a condição no início do laço, ou seja, antes de qualquer instrução do bloco ser executada.

- Laço Repita enquanto (<condição>)

```
algoritmo ()

{
	inteiro i;
	i := 0;

	escreva ("exibe os números pares de 0 a 100");
	escreva ("utiliza o laço enquanto");

	repita enquanto ( i <= 100 )
	{
		escreva (i);
		i := i + 2;
	}
}
```

Comentário:

Uma pesquisa sobre características físicas da população de uma determinada região coletou os seguintes dados, referentes a cada habitante, para serem analisados: sexo (masculino, feminino), cor dos olhos (azuis, verdes, castanhos), cor dos cabelos (loiros, castanhos, pretos) e a idade em anos. Para cada habitante, foi digitada uma linha com esses dados e a última linha que não corresponde a ninguém tem o valor de idade igual a -1. Fazer um algoritmo que determine e escreva:
a) a maior idade dos habitantes
b) a porcentagem de indivíduos do sexo feminino cuja idade está entre 18 e 35 anos inclusive e que tenham olhos verdes e cabelos loiros

```
algoritmo()
{
	inteiro idade;		// idade do entrevistado
	inteiro MaiorIdade;	// maior idade encontrada
	inteiro Entrevistados;			// numero de pessoas entrevistadas
	inteiro Percentual;	// percentual do item b
	inteiro Mulheres;			// numero de individuos que atendem ao item b
	ascii TipoSexo;		// sexo (M-m ou F-f)
	ascii CorCabelo;			// cabelo (Loiro,Castanho,Preto)
	ascii CorOlhos;				// olhos (azuis,verdes,castanhos)

	MaiorIdade := 0;
	Entrevistados := 0;
	Mulheres := 0;

	Repita enquanto ( idade != -1 )
	{
		escreva
( "=================================================");
		escreva ( "Pesquisa de preferencias" );
		escreva ( "sexo (masculino,feminino)" );
		escreva ( "cabelos (loiros, castanhos, pretos)" );
		escreva ( "olhos (azuis, verdes, castanhos)" );
		escreva ( "caracteristica procurada: " );
		escreva ( "> mulher, entre 18 e 35 anos, loira, olhos verdes" );
		escreva
( "=================================================");

		leia ( "informe a idade: ", idade );
		se ( idade != -1 )
		{
			leia ( "informe o sexo: ", TipoSexo );
			leia ( "informe a cor dos cabelos: ", CorCabelo );
			leia ( "informe a cor dos olhos: ", CorOlhos );
			se ( idade > MaiorIdade )
			{
				MaiorIdade := idade;
			}
			se ( idade >= 18 & idade <= 35 )
			{
				se ( TipoSexo == 'f' & CorOlhos == 'v' )
				{
					se ( CorCabelo == 'l' )
					{
						Mulheres := Mulheres + 1;
					}
				}
			}
```

```
                        Entrevistados := Entrevistados + 1;
                }
        }
        Percentual := 100 * Mulheres / Entrevistados;
        escreva ( "A maior idade e: ", MaiorIdade );
        escreva ( "O percentual de mulheres ...: ", Percentual );
}
```

Comentário:

```
algoritmo()
{
	inteiro idade;		// idade do entrevistado
	inteiro MaiorIdade;	// maior idade encontrada
	inteiro Entrevistados;		// numero de pessoas entrevistadas
	inteiro Percentual;	// percentual do item b
	inteiro Mulheres;		// numero de individuos que atendem ao item b
	ascii TipoSexo;		// sexo (M-m ou F-f)
	ascii CorCabelo;		// cabelo (Loiro,Castanho,Preto)
	ascii CorOlhos;		// olhos (azuis,verdes,castanhos)

	MaiorIdade := 0;
	Entrevistados := 0;
	Mulheres := 0;
	idade := 0;

	Repita enquanto ( idade != -1 )
	{
		leia ( "informe o sexo: ", TipoSexo );
		leia ( "informe a cor dos cabelos: ", CorCabelo );
		leia ( "informe a cor dos olhos: ", CorOlhos );
		leia ( "informe a idade: ", idade );
		se ( idade # -1 )
		{
			se ( idade > MaiorIdade )
			{
				MaiorIdade := idade;
			}
			selecao
			{
				caso ( idade >= 18 )
				caso ( idade <= 35 )
				caso ( TipoSexo == 'f' )
				caso ( CorOlhos == 'v' )
				caso ( CorCabelo == 'l' )
				{
					Mulheres := Mulheres + 1;
				}
			}
			Entrevistados := Entrevistados + 1;
		}
	}
	Percentual := 100 * Mulheres / Entrevistados
	escreva ( "A maior idade e: ", MaiorIdade );
	escreva ( "O percentual de mulheres ...: ", Percentual );
}
```

Comentário:

Um comerciante deseja fazer o levantamento do lucro das mercadorias que ele comercializa. Para isto, mandou digitar uma linha para cada mercadoria serem com o nome, preço de compra e preço de venda das mesmas. Fazer um algoritmo que determine e escreva quantas mercadorias proporcionam lucro < 10%, 10% <= lucro <= 20% e lucro > 20%, determine e escreva o valor total de compra e de venda de todas as mercadorias assim como o lucro total.

```
algoritmo()
{
	cadeia Mercadoria;	// nome da mercadoria
	real PrecoCompra;		// preco de compra do produto
	real PrecoVenda;	// preco de venda do produto
	real Lucro;		// lucro da venda
	real LucroTotal;		// lucro total da venda
	real ValorTotalCompra;	// valor total da compra
	real ValorTotalVenda;	// valor total da venda
	inteiro i;	// contador de mercadorias com lucro abaixo de 10%
	inteiro j;	// contador de mercadorias com lucro entre 10% e 20%
	inteiro k;	// contador de mercadorias com lucro acima de 20%
	ascii resp;

	resp := 's';
	repita enquanto ( resp == 's' | resp == 'S' )
	{
		leia ( "informe o nome da mercadoria: ", Mercadoria );
		leia ( "informe o preco de compra: ", PrecoCompra );
		leia ( "informe o preco de venda: ", PrecoVenda );
		Lucro := ( PrecoVenda - PrecoCompra ) / PrecoCompra;
		se ( Lucro < 0.1 )
		{
			i := i + 1;
		}
		senao
		{
			se ( Lucro > 0.1 & Lucro <= 0.2 )
			{
				j := j + 1;
			}
			senao
			{
				k := k + 1;
			}
		}
		LucroTotal := LucroTotal + Lucro;
		ValorTotalCompra := ValorTotalCompra + PrecoCompra;
		ValorTotalVenda := ValorTotalVenda + PrecoVenda;
		leia ( "deseja continuar (s/n):  ", resp );
		escreva
( "==========================================" );
	}
	escreva ( "mercadorias com lucro < 10%: ", i );
	escreva ( "mercadorias com lucro entre 10% e 20%: ", j );
	escreva ( "mercadorias com lucro acima de 20%: ", k );
}
```

Comentário:

Fazer um algoritmo para calcular os rendimentos e o saldo acumulado trimestralmente

```
algoritmo ()

{
	// calcula os rendimentos e o saldo acumulado trimestralmente

	real c;		// capital inicial
	real i;		// taxa de juros trimestral
	real m;			// montante acumulado apos o tempo t
	real r;		// rendimentos anuais
	inteiro t;		// tempo em anos da aplicacao
	inteiro n;		// contador para o numero de anos t
	inteiro k;		// numero de trimestres em t anos

	mensagem ( "calcula os rendimentos e o saldo acumulado" );

	leia ( "informe o capital inicial R$: ", c);
	leia ( "informe o valor da taxa trimestral: ",i);
	leia ( "informe o tempo de aplicacao [anos]: ",t);

	n := 1;		// numero de periodos de tres meses
	k := 4 * t;		// ha 4 trimestres em um ano

	i := i / 100;

	escreva ( "trimestre, capital, rendimento, montante" );

	repita enquanto ( n <= k )
	{
		m := c * ( ( 1 + i ) ^ n );
		r := m - c;
		escreva ( n );
		escreva ( c );
		escreva ( r );
		escreva ( m );
		n := n + 1;
	}
}
```

Comentário:

- Laço Repita enquanto aninhado

```
algoritmo ()

{
	inteiro i;
	inteiro j;
	i := 0;
	j := 0;

	escreva ("utiliza o laco enquanto aninhado");

	repita enquanto ( i <= 5 )
	{
		j := 0;
		repita enquanto ( j <= 5 )
		{
			escreva (i);
			escreva (j);
			j := j + 1;
		}
		escreva ( "proximo =>" );
		i := i + 1;
	}
}
```

Comentário:

Estrutura de Repetição – Laço Repita

- Laço repita

```
algoritmo ()
{
	inteiro j;
	j := 0;

	escreva ("exibe os números de 0 a 100");
	escreva ("utiliza o laço repita");

	repita ( j > 100 )
	{
		escreva (j);
		j := j + 1;
	}
}
```

Comentário:

- Laço enquanto

```
algoritmo ()
{
	inteiro j;
	j := 0;

	escreva ("exibe os números de 0 a 100");
	escreva ("utiliza o laço repita");

	enquanto ( j <= 100 )
	{
		escreva (j);
		j := j + 1;
	}
}
```

Comentário:

- Laço Faca

```
algoritmo ()
{
	inteiro j;
	j := 0;

	escreva ("exibe os números de 0 a 100");
	escreva ("utiliza o laço repita");

	faca
	{
		escreva (j);
		j := j + 1;
	}
	ateque ( j > 100 )
}
```

Comentário:

Estrutura de Repetição – Laço Para

- Laço Para

```
algoritmo ()

{
        inteiro i;                  // declara uma variavel do tipo inteiro

        escreva ("exibe os numeros impares de 0 a 100");
        escreva ("exemplo do laco para");

        para ( i := 1 ate 100 passo 2 )
        {
                escreva (i);
        }
}
```

Comentário:

- Laço Para com variáveis

```
algoritmo ()

{
	inteiro CONT;
	inteiro LIMITE;
	inteiro INC;

	escreva ("exibe os numeros ímpares de 0 a 100");
	escreva ("exemplo do laco para");

	LIMITE := 100;
	INC := 2;

	para ( CONT := 1 ate LIMITE passo INC )
	{
		escreva (CONT);
	}
}
```

Comentário:

Exibir os números ímpares

```
algoritmo ()

{
	inteiro C;
	inteiro L;
	inteiro I;

	escreva ("exibe os numeros ímpares de 0 a 100");
	escreva ("exemplo do laço para");

	L := 100;
	I := 2;

	para ( C := 1 ate L passo I )
	{
		escreva (C);
	}
}
```

Comentário:

- Laço Para aninhado

```
algoritmo ()

{
	inteiro i;			// declara uma variavel do tipo inteiro
	inteiro j;			// declara uma variavel do tipo inteiro
	i := 0;		// inicializa a variavel com o valor zero
	j := 0;		// inicializa a variavel com o valor zero

	escreva ("exemplo do laco para aninhado");

	para ( i := 0 ate 10 passo 2 )
	{
		escreva (i);

		para ( j := 1 ate 10 passo 2 )
		{
			escreva (j);
		}
		escreva ( " " );
	}
}
```

Comentário:

Cronômetro digital

```
algoritmo ()
{
	inteiro h;		// horas
	inteiro m;		// minutos
	inteiro s;		// segundos

	escreva ( "cronometro digital" );

	para ( h := 1 ate 23 passo 1 )
	{
		para ( m := 1 ate 59 passo 1 )
		{
			para ( s := 1 ate 59 passo 1 )
			{
				escreva (h);
				escreva (m);
				escreva (s);
				escreva (":");
			}
		}
	}
}
```

Comentário:

Estrutura de Repetição – Laço For

- Laço For

```
algoritmo ()

{
        inteiro i;                  // declara uma variavel do tipo inteiro

        escreva ("exibe os numeros impares de 0 a 100");
        escreva ("exemplo do laco para");

        for ( i := 1; i <= 100; i := i + 2 )
        {
                escreva (i);
        }
}
```

Comentário:

Instruções de Salto – Interrompa

- Instrução ***interrompa*** com o laço repita

```
algoritmo ()

{
	inteiro j;
	j := 0;

	escreva ("exibe os numeros de 0 a 50");
	escreva ("utiliza o laco repita com a instrucao interrompa");

	repita ( j > 100 )
	{
		escreva (j);
		j := j + 1;
		se ( j > 50 )
		{
			interrompa;
		}
	}
}
```

Comentário:

- Instrução ***interrompa*** com o laço enquanto

```
algoritmo ()

{
	inteiro j;
	j := 0;

	escreva ("exibe os numeros de 0 a 50");
	escreva ("utiliza o laco enquanto com a instrucao interrompa");

	enquanto ( j <= 100 )
	{
		escreva (j);
		j := j + 1;
		se ( j > 50 )
		{
			interrompa;
		}
	}
}
```

Comentário:

- Instrução ***interrompa*** com o laço para

```
algoritmo ()

{
	inteiro j;
	j := 0;

	escreva ("exibe os numeros de 0 a 50");
	escreva ("utiliza o laco para com a instrucao interrompa");

	para ( j := 0 ate 100 passo 1 )
	{
		escreva (j);
		j := j + 1;
		se ( j > 50 )
		{
			interrompa;
		}
	}
}
```

Comentário:

- Instrução ***interrompa*** com laço repita aninhado

```
algoritmo ()

{
	inteiro i;
	inteiro j;

	escreva ("instrucao interrompa aninhadas");
	escreva ("utiliza o laco repita com a instrucao interrompa");

	repita ( i > 5 )
	{
		repita ( j > 100 )
		{
			escreva (j);
			j := j + 1;
			se ( j > 20 )
			{
				j := 0;
				escreva ("proxima");
				interrompa;
			}
		}
		i := i + 1;
		se ( i > 2 )
		{
			interrompa;
		}
	}
}
```

Comentário:

- Instrução ***interrompa*** com o laço enquanto aninhado

```
algoritmo ()

{
	inteiro i;
	inteiro j;

	escreva ("instrucao interrompa aninhadas");
	escreva ("utiliza o laco enquanto com a instrucao interrompa");

	enquanto ( i <= 5 )
	{
		enquanto ( j <= 100 )
		{
			escreva (j);
			j := j + 1;
			se ( j > 20 )
			{
				j := 0;
				escreva ("proxima");
				interrompa;
			}
		}
		i := i + 1;
		se ( i > 2 )
		{
			interrompa;
		}
	}
}
```

Comentário:

Instruções de Salto – Continue

- Instrução ***continue*** com o laço repita

```
algoritmo ()

{
	inteiro i;
	inteiro j;

	escreva ("exibe os numeros de 0 a 50");
	escreva ("utiliza o laco repita com a instrucao continue");

	j := 1;
	escreva ( "numero da vez:", j );

	repita ( i > 100 )
	{
		escreva (i);
		i := i + 1;
		se ( i > 50 & j <= 3 )
		{
			j := j + 1;
			escreva ( "" );
			escreva ( "numero da vez:", j );
			i := 1;
			continue;
		}
	}
}
```

Comentário:

Funções Matemáticas

- Função Quociente()

```
algoritmo ()

{
	// teste da funcao quociente

	real a;
	real b;
	real r1;
	real r2;

	escreva ("teste do operador de divisao / e da funcao quoc()");

	leia ("digite um numero a: ", a);
	leia ("digite um numero b: ", b);

	r1 := a / b;

	escreva ( "o quociente da divisao de a por b e: ", r1 );

	r2 := quociente ( a, b );

	escreva ( "o quociente da divisao de a por b e: ", r2 );
}
```

Comentário:

- Função Resto()

```
algoritmo ()

{
	// teste de operador aritmetico e funcao

	inteiro a;		// declara uma variavel do tipo inteiro
	inteiro b;		// declara uma variavel do tipo inteiro
	inteiro r1;		// resultado
	inteiro r2;		// resultado

	escreva ("teste do operador resto <%> e da funcao resto()");

	leia ("digite um numero a: ", a);
	leia ("digite um numero b: ", b);

	r1 := a % b;

	escreva ( "o resto da divisao inteira de a por b e: ", r1 );

	r2 := resto( a, b );

	escreva ( "o resto da divisao inteira de a por b e: ", r2 );
}
```

Comentário:

- Função Sinal()

```
algoritmo ()

{
	// teste da funcao interna sinal()

	inteiro x;

	escreva ("testa a funcao interna sinal()");

	leia ("digite um numero real x: ", x);

	se ( sinal(x) == -1 )
	{
		escreva ( "o numero informado e negativo");
	}
	senao
	{
		escreva ( "o numero informado e positivo");
	}
}
```

Comentário:

- Função PartInt()

```
algoritmo ()

{
        // teste da função interna partint()

        real x;
        real y;

        escreva ("testa a funcao interna partint()");

        leia ("digite um numero real x: ", x);

        y := PartInt (x);

        escreva ( "a parte inteira e ", y );
}
```

Comentário:

- Função Exponencial()

```
algoritmo ()

{
        // teste da funcao interna exponencial()

        real a; // valor do expoente
        real s; // resultado da chamada a exponencial()

        escreva ("testa a funcao interna exponencial()");

        leia ("informe o expoente : ", a);

        s := exponencial (a);

        escreva ( "valor do expoente informado ..... ", a );
        escreva ( "valor de e elevado ao expoente .... ", s );
}
```

Comentário:

- Função Seno() (argumento em graus)

```
algoritmo ()

{
        // teste da funcao interna seno()

        real a; // arco lido da entrada
        real s; // valor do seno

        escreva ("testa a funcao interna seno()");

        leia ("informe o arco : ", a);

        s := seno (a,G);        // chama a funcao seno com argumento em graus

        escreva ( "valor do arco informado ..... ", a );
        escreva ( "valor do seno ..... ", s );
}
```

Comentário:

- Função Seno() (argumento em radianos)

```
algoritmo ()

{
        // teste da funcao interna seno()

        real a; // arco lido da entrada
        real s; // valor do seno

        escreva ("testa a funcao interna seno()");

        leia ("informe o arco : ", a);

        s := seno (a,R);        // chama a funcao seno com argumento em radianos

        escreva ( "valor do arco informado ..... ", a );
        escreva ( "valor do seno ..... ", s );
}
```

Comentário:

- Função Coseno() (argumento em graus)

```
algoritmo ()

{
        // teste da funcao interna coseno()

        real a; // arco lido da entrada
        real s; // valor do co-seno

        escreva ("testa a funcao interna coseno()");

        leia ("informe o arco : ", a);

        s := coseno (a,G);     // chama a funcao cosen com argumento em graus

        escreva ( "valor do arco informado ..... ", a );
        escreva ( "valor do co-seno ..... ", s );
}
```

Comentário:

- Função Coseno() (argumento em radianos)

```
algoritmo ()

{
        // teste da funcao interna coseno()

        real a; // arco lido da entrada
        real s; // valor do co-seno

        escreva ("testa a funcao interna coseno()");

        leia ("informe o arco : ", a);

        s := coseno (a,R);      // chama a funcao cosen com argumento em radianos

        escreva ( "valor do arco informado ..... ", a );
        escreva ( "valor do co-seno ..... ", s );
}
```

Comentário:

- Função Potencia()

```
algoritmo ()

{
	// teste da funcao potencia()

	real a; // base, tipo real
	real b; // expoente, tipo real
	real r;  // potencia, resultado

	escreva ("teste do operador de potenciacao");

	leia ("digite um numero a: ", a );

	leia ("digite um numero b: ", b );

	r := potencia(a,b);

	escreva ( "a elevado a b e: ", r );
}
```

Comentário:

- Função LogNeper()

```
algoritmo ()

{
        // teste da funcao interna logneper()

        real a; // valor
        real s; // resultado da chamada a logneper()

        escreva ("testa a funcao interna logneper()");

        leia ("informe o numero : ", a);

        s := logneper (a);

        escreva ( "valor informado ..... ", a );
        escreva ( "valor do logaritmo na base e ... ", s );
}
```

Comentário:

- Função LogDec()

```
algoritmo ()

{
        // teste da funcao interna logdec()

        real a; // valor
        real s; // resultado da chamada a logdec()

        escreva ("testa a funcao interna logdec()");

        leia ("informe o numero : ", a);

        s := logdec (a);

        escreva ( "valor informado ..... ", a );
        escreva ( "valor do logaritmo na base 10 ... ", s );
}
```

Comentário:

- Função Raiz()

```
algoritmo ()
{
	// teste da funcao interna raiz()

	real x;
	real r;

	escreva ("testa a funcao interna raiz()");

	leia ( "digite um numero: ", x );

	r := raiz (x);

	escreva ( "o numero informado foi ....: ", x );
	escreva ( "a raiz do numero e .......: ", r );
}
```

Comentário:

Para se determinar o número de lâmpadas necessárias para cada cômodo de uma residência, existem normas que dão o mínimo de potência de iluminação exigida por metro quadrado conforme a utilização desse cômodo. Supondo que só serão usadas lâmpadas de 60 watts, fazer um algoritmo que:
a)Leia um número indeterminado de linhas contendo cada uma: o nome do cômodo de uma residência, a classe de iluminação deste cômodo e a largura e comprimento desse cômodo
b)calcule e escreva para cada cômodo: o nome do cômodo, a área do cômodo, a potência de iluminação e o número de lâmpadas necessárias.
c)para toda a residência, o total de lâmpadas e a potência total

Observação:
Se o numero de lâmpadas calculado for fracionário, considerar o menor inteiro que contenha esse número. A última linha que não entrara nos cálculos conterá no lugar do nome do cômodo a palavra vazio.

Tabela de potência

Utilização	classe	potência /m2
Quarto	1	15
sala de tv	1	15
salas	2	18
cozinha	2	18
varandas	2	18
escritório	3	20
banheiro	3	20

```
algoritmo()
{
	cadeia NomeComodo;		// denominacao do comodo
	inteiro LampadasComodo;		// numero de lampadas por comodo
	real PotenciaExigida;// potencia exigida por comodo
	real Larg;		// largura do comodo
	real Comp;		// comprimento do comodo
	real Area;		// area do comodo
	inteiro TotalLampadas;		// total de lampadas para a residencia
	real PotenciaTotal;		// potencia total para a residencia
	inteiro classe;		// classe do comodo
	real r;  // variavel auxiliar
	ascii resp;

	repita ( resp == 'n' | resp == 'N' )
	{
		leia ( "informe o nome do comodo: " , NomeComodo );
		leia ( "informe a classe do comodo: ", classe );
		leia ( "informe a largura do comodo: ", Larg );
		leia ( "informe o comprimento do comodo: ", Comp );

		Area := Larg * Comp;
		se ( classe == 1 )
		{
			PotenciaExigida := Area * 15;
		}
		senao
		{
			se ( classe == 2 )
			{
				PotenciaExigida := Area * 18;
			}
			senao
```

```
                {
                        se ( classe == 3 )
                        {
                                PotenciaExigida := Area * 20;
                        }
                }
        }
        LampadasComodo := PotenciaExigida / 60;
        r := Resto ( PotenciaExigida, 60 );
        se ( r <> 0 )
        {
                LampadasComodo := LampadasComodo + 1;
        }
        escreva ( "comodo: ", NomeComodo );
        escreva ( "area: ", Area );
        escreva ( "potencia exigida: ", PotenciaExigida );
        escreva ( "numero de lampadas: ", LampadasComodo );
        escreva
( "=======================================" );
        PotenciaTotal := PotenciaTotal + PotenciaExigida;
        TotalLampadas := TotalLampadas + LampadasComodo;
        leia ( "deseja continuar (s/n): ", resp );
    }
    escreva ( "potencia total: ", PotenciaTotal );
    escreva ( "total de lampadas: ", TotalLampadas );
}
```

Comentário:

Resolução da equação do 2º grau

```
algoritmo ()

{
	// equacao do 2 grau

	real a;
	real b;
	real c;
	real d;
	real x1;
	real x2;

	escreva ("equacao do 2 grau ax2 + bx + c = 0");
	leia ("coeficiente a: ", a);
	leia ("coeficiente b: ", b);
	leia ("coeficiente c: ", c);
	d := b^2 - (4*a*c);
	se ( d > 0 )
	{
		escreva ("raizes distintas: delta = ", d);
		x1 := (-b + raiz(d)) / (2*a);
		x2 := (-b - raiz(d)) / (2*a);
		escreva ("x1 = ", x1);
		escreva ("x2 = ", x2);
	}
	senao
	{
		se ( d == 0 )
		{
			escreva ("raizes iguais: delta = ", d);
			x1 := (-b + raiz(d)) / (2*a);
			escreva ("x1 = ", x1);
			escreva ("x2 = ", x1);
		}
		senao
		{
			escreva ("raizes imaginárias: delta = ", d);
			escreva ("as raizes não são reais");
		}
	}
}
```

Comentário:

Cálculo da hipotenusa e dos senos e co-senos de um ângulo de um triângulo retângulo

```
algoritmo ()

{
	// teste das funcoes internas sen() e cosen()

	real a; // hipotenusa do triangulo retangulo
	real b; // cateto
	real c; // cateto
	real d; // valor do seno
	real e; // valor do co-seno

	escreva ("seno e cosseno dos angulos de um triangulo retangulo");

	leia ("informe o cateto b : ", b);
	leia ("informe o cateto c : ", c);

	a := ( b^2 + c^2 );
	a := raiz (a);
	d := b / a;		// seno do angulo B = oposto / hipotenusa
	e := c / a;		// co-seno do angulo B = adjacente / hipotenusa

	escreva ( "hipotenusa a ..... ", a );
	escreva ( "cateto b ..... ", b );
	escreva ( "cateto c ..... ", c );
	escreva ( "seno do angulo B ..... ", d );
	escreva ( "co-seno do angulo B ..... ", e );
}
```

Comentário:

Funções de Cadeia

- Função Comprimento()

```
algoritmo ()
{
	// teste da funcao interna comprimento()

	inteiro i, j;
	cadeia nome;

	nome := "Amanda B. Paim & Marina B. Paim";
	i := comprimento (nome);
	j := comprimento ("Amanda B. Paim & Marina B. Paim");
	escreva (i);
	escreva (j);
}
```

Comentário:

- Função Concatena()

```
algoritmo ()
{
	cadeia x1, x2, x3;

	x4 := concatena ( "fernando", "paim" );

	x1 := "janeiro";
	x2 := "maio";
	x3 := concatena(x1,x2);
	escreva(x3);

	x1 := concatena(x1,"paim");
	escreva ( x1 );
	x2 := concatena("fernando",x1);
	escreva ( x2 );
}
```

Comentário:

- Função Primeiros()

```
algoritmo ()
{
	cadeia nome, subx1;

	nome := "fernando antonio de padua paim";

	subx1 := primeiros ( nome, 12 );

	escreva (nome);

	escreva (subx1);

}
```

Comentário:

- Função SubCadeia()

```
algoritmo ()
{
	cadeia nome;
	cadeia subx1;
	cadeia subx2;
	cadeia subx3;
	cadeia subx4;
	cadeia subx5;

	nome := "fernando antonio de padua paim";

	subx1 := subcadeia ( nome, 1, 8 );
	subx2 := subcadeia ( nome, 10, 7 );
	subx3 := subcadeia ( nome, 18, 2 );
	subx4 := subcadeia ( nome, 21, 5 );
	subx5 := subcadeia ( nome, 27, 4 );

	escreva (nome);
	escreva (subx1);
	escreva (subx2);
	escreva (subx3);
	escreva (subx4);
	escreva (subx5);
}
```

Comentário:

- Função SubCadeia

```
algoritmo ()
{
	cadeia nome;
	cadeia subx1;
	inteiro i;
	inteiro j;

	nome := "fernando antonio de padua paim";

	i := 8;
	j := 8;
	subx1 := subcadeia ( nome, i, j );
	escreva (nome);
	escreva (subx1);
}
```

Comentário:

- Função Ultimos()

```
algoritmo ()
{
	cadeia nome;
	cadeia subx1;

	nome := "fernando antonio de padua paim";

	subx1 := ultimos ( nome, 12 );

	escreva (nome);
	subx1 := ultimos ( nome, 70 );
	escreva (subx1);
}
```

Comentário:

- Função Compara()

```
algoritmo ()
{
	// teste do tipo de dado cadeia

	cadeia nome1;		// declara uma variavel do tipo cadeia
	cadeia nome2;		// declara uma variavel do tipo cadeia
	inteiro i;

	leia ( "informe um nome ", nome1 );
	leia ( "informe um nome ", nome2 );
	i := ordem ( nome1, nome2 );
	se ( i == -1 )
	{
		escreva ("ordem trocada");
	}
	senao
	{
		escreva ("ordem alfabetica");
	}
}
```

Comentário:

Capicuas são números que tem o mesmo valor se lidos da esquerda para direita ou da direita para esquerda. Exemplo: 44, 232, etc. Fazer um algoritmo que determine e escreva todos os números inteiros menores que 10000 que são capicuas.

```
algoritmo()
{
	cadeia N;		// numero lido
	ascii Unidade;
	ascii Dezena;
	ascii Centena;
	ascii Milhar;

	leia ( "informe um numero: ", N );

	Milhar := Primeiros ( N, 1 );
	Centena := SubCadeia ( N, 2, 1 );
	Dezena := SubCadeia ( N, 3, 1 );
	Unidade := Ultimos ( N, 1 );
	se ( Milhar == Unidade & Centena == Dezena )
	{
		escreva ( "o numero informado e capicua" );
	}
	senao
	{
		escreva ( "o numero informado nao e capicua" );
	}
}
```

Comentário:

Vetores

- Atribuição de valores para vetor do tipo inteiro.

```
algoritmo ()
{
	vetor inteiro f[2];

	f[1] := 10;
	f[2] := 15;
	escreva ( f[1] );
	escreva ( f[2] );
}
```

Comentário:

- Atribuição de valores para vetor do tipo cadeia

```
algoritmo ()
{
	inteiro i;
	vetor cadeia mes[12];

	mes[1]  := "janeiro";
	mes[2]  := "fevereiro";
	mes[3]  := "marco";
	mes[4]  := "abril";
	mes[5]  := "maio";
	mes[6]  := "junho";
	mes[7]  := "julho";
	mes[8]  := "agosto";
	mes[9]  := "setembro";
	mes[10] := "outubro";
	mes[11] := "novembro";
	mes[12] := "dezembro";

	i := 1;

	repita ( i > 12 )
	{
		escreva ( mes[i] );
		i := i + 1;
	}
}
```

Comentário:

- Entrada e saída de valores para vetor. (tipo inteiro)

```
algoritmo ()
{
	inteiro i;
	vetor real j [ 5 ];

	i := 1;

	repita ( i > 5 )
	{
		leia ( "informe valor: ", j[i] );
		i := i + 1;
	}

	i := 1;

	repita ( i > 5 )
	{
		escreva ( "valor informado: ", j[i] );
		i := i + 1;
	}
}
```

Comentário:

- Entrada e saída de valores para vetor. (tipo ascii)

```
algoritmo ()
{
	inteiro i;
	vetor ascii j[5];

	i := 1;

	repita ( i > 5 )
	{
		leia ( "informe valor: ", j[i] );
		i := i + 1;
	}

	i := 1;

	repita ( i > 5 )
	{
		escreva ( "valor informado: ", j[i] );
		i := i + 1;
	}
}
```

Comentário:

- Entrada e saída de valores para vetor. (tipo cadeia)

```
algoritmo ()
{
	inteiro i;
	vetor cadeia j[5];

	i := 1;

	repita ( i > 5 )
	{
		leia ( "informe valor: ", j[i] );
		i := i + 1;
	}

	i := 1;

	repita ( i > 5 )
	{
		escreva ( "valor informado: ", j[i] );
		i := i + 1;
	}
}
```

Comentário:

Quadrado dos valores

```
algoritmo ()
{
	inteiro i;
	vetor inteiro j[5];

	i := 1;

	repita ( i > 5 )
	{
		leia ( "informe valor: ", j[i] );
		i := i + 1;
	}

	i := 1;

	repita ( i > 5 )
	{
		j[i] := j[i] ^ 2;
		escreva ( "novo valor: ", j[i] );
		i := i + 1;
	}
}
```

Comentário:

Elementos pares

```
algoritmo ()
{
	inteiro i;
	vetor inteiro j[20];

	para ( i := 1 ate 20 passo 1)
	{
	j[i] := i;
	}
	para ( i := 1 ate 20 passo 1)
	{
		se ( j[i]%2 == 0 )
		{
			escreva ("par: ", j[i] );
		}
		senao
		{
			escreva ("impar: ", j[i] );
		}
	}
}
```

Comentário:

Soma de vetores

```
algoritmo ()
{
	inteiro i;
	vetor inteiro A[5];
	vetor inteiro B[5];
	vetor inteiro C[5];

	para ( i := 1 ate 5 passo 1 )
	{
		A[i] := i;
		B[i] := 2*i;
	}
	para ( i := 1 ate 5 passo 1 )
	{
		C[i] := A[i] + B[i];
	}
	para ( i := 1 ate 5 passo 1 )
	{
		escreva ( C[i] );
	}
}
```

Comentário:

- Atribuição entre elementos de vetores e variáveis simples

```
algoritmo ()
{
	inteiro i;
	inteiro k;
	vetor inteiro j [ 5 ];

	i := 1;
	repita ( i > 5 )
	{
		leia ( "informe valor: ", j[i] );
		i := i + 1;
	}
	para ( i := 1 ate 5 passo 1 )
	{
		k := j[i];
		escreva (k);
	}
}
```

Comentário:

- Inicialização de vetores do tipo ascii

```
algoritmo ()
{
	inteiro i;
	vetor ascii j[5];

	i := 1;

	repita ( i > 5 )
	{
		j[i] := 'A';
		i := i + 1;
	}

	i := 1;

	repita ( i > 5 )
	{
		escreva ( j[i] );
		i := i + 1;
	}
}
```

Comentário:

Conta o número de vogais. (1ª versão)

```
algoritmo ()
{
	// conta o numero de vogais

	inteiro i;
	inteiro j;
	vetor ascii letra[5];

	para ( i := 1 ate 5 passo 1 )
	{
		leia ("informe letra: ", letra[i] );
		selecao
		{
			caso ( letra[i] == 'a' )
			{
				j := j + 1;
			}
			caso ( letra[i] == 'e' )
			{
				j := j + 1;
			}
			caso ( letra[i] == 'i' )
			{
				j := j + 1;
			}
			caso ( letra[i] == 'o' )
			{
				j := j + 1;
			}
			caso ( letra[i] == 'u' )
			{
				j := j + 1;
			}
		}
	}
	escreva ( "numero de vogais : ", j );
}
```

Comentário:

Conta o número de vogais. (2ª versão)

```
algoritmo ()
{
	// conta o numero de vogais

	inteiro i;			// indexador para o laço de leitura das 10 letras
	inteiro j;			// indexador para o vetor de vogais
	inteiro nv;			// numero de vogais
	ascii letra;		// letra a ser lida do teclado
	vetor ascii vogal[5];	// relacao das 5 vogais

	vogal[1] := `a`;
	vogal[2] := `e`;
	vogal[3] := `i`;
	vogal[4] := `o`;
	vogal[5] := `u`;

	para ( i := 1 ate 10 passo 1 )
	{
		leia ("informe letra: ", letra );
		para ( j := 1 ate 5 passo 1 )
		{
			se ( letra == vogal[j] )
			{
				nv := nv + 1;
			}
		}
	}
	escreva ( "numero de vogais : ", nv );
}
```

Comentário:

- Pesquisa sequencial I

```
algoritmo()
{
	vetor inteiro Valor[10];
	inteiro i;
	inteiro ValorPesquisado;
	inteiro N;
	inteiro c;
	inteiro a;

	para ( i := 1 ate 10 passo 1 )
	{
		leia ( "informe valor: ", Valor[i] );
	}
	leia ( "informe o valor a ser pesquisado: ", ValorPesquisado );
	N := 10;
	a := 1;
	c := 2;
	i := 1;
	enquanto ( i <= N & Valor[i] <> ValorPesquisado )
	{
		i := i + 1;
		a := a + 1;
		c := c + 2;
	}
	se ( i > N )
	{
		escreva ( "valor nao encontrado" );
	}
	senao
	{
		escreva ( "valor encontrado na posicao ", i );
	}
	escreva ( "numero de comparacoes: ", c );
	escreva ( "numero de atribuicoes: ", a );
}
```

Comentário

O algoritmo acima exemplifica a técnica de pesquisa sequencial para a busca de valores em uma determinada estrutura de dados. A pesquisa sequencial é a técnica mais simples utilizada para se efetuar a busca de valores em qualquer universo de valores dados. A pesquisa sequencial consiste em percorrer a estrutura a partir do seu início até o seu final ou até a posição onde se encontra a chave procurada. No exemplo, utilizou-se um vetor de dimensão 10 de tipos numéricos inteiros o qual irá representar a estrutura de dados na qual se deseja efetuar a pesquisa de um valor recebido pelo teclado. Um vetor é uma variável composta unidimensional, logo necessitamos de um índice, e apenas um, para referenciar os seus elementos, tal objetivo é cumprido pela variável ***i***, que tem a função aqui de indexar ou subscritar o vetor ***Valor***.
Um vetor é também uma variável homogênea, isto é, agrupa valores todos do mesmo tipo sendo que, foi escolhido o tipo inteiro por questões de

simplicidade. Na verdade a questão da simplicidade refere-se a entrada dos dados, uma vez que, os mesmos serão recebidos via teclado e, desse modo sua digitação é mais rápida. Foi utilizado a variável ***N***, dispensável no exemplo, apenas para tornar o algoritmo mais genérico, por exemplo, se a dimensão do vetor tivesse que ser outra. O valor de ***N*** deve ser o mesmo valor da dimensão do vetor em que se deseja efetuar a pesquisa. Então, uma vez recebidos os valores para o vetor ***Valor*** e inicializadas de forma adequada as variáveis ***i*** e ***N*** tem início o processo da pesquisa sequencial. Conforme explicitado acima, a pesquisa sequencial varre a estrutura de dados desde o seu início ***(i = 1)***, até o seu final ***(i <= N)*** ou até que o valor pesquisado seja encontrado ***(Valor[i] <> ValorPesquisado)***, quando então também o processo deverá ser encerrado. O cumprimento dessas duas condições está garantido pelo laço enquanto na expressão ***enquanto (i<=N & Valor[i]<>ValorPesquisado)***. Uma vez finalizado o laço, pelo estouro de uma das condições deve-se testar para verificar qual foi a condição que causou o escape do laço. Isto é feito comparando-se o valor da variável ***i*** com o valor limite, no caso a dimensão do vetor ***N***. Se ***(i > N)*** então o valor pesquisado não foi encontrado no vetor ***Valor***, pois ***Valor[N]*** é o último elemento do vetor e quem causou a saída do laço foi o fato de ***i*** tornar-se maior que ***N*** rompendo o laço ***enquanto (i <= N)***. Por outro lado, se ***i <= N*** então quem causou o escape do laço foi o fato de a condição ***Valor[i] <> ValorPesquisado*** revelar-se falsa, ou seja o vetor ***Valor*** contem pelo menos um elemento igual ao valor que esta sendo pesquisado, e o valor de ***i*** neste momento indica a posição desse elemento no vetor ***Valor***. No processo acima, podemos notar que para cada iteração do laço enquanto são realizadas duas comparações, ***i < N*** e ***Valor[i] <> ValorPesquisado***. Supondo então que o valor pesquisado fosse o último elemento do vetor de dimensão ***N*** (pior caso) teríamos um total de 2N comparações. Esse valor pode ser diminuído.

- Pesquisa sequencial II

```
algoritmo()
{
	vetor inteiro Valor[11];
	inteiro i;
	inteiro ValorPesquisado;
	inteiro N;
	inteiro c;
	inteiro a;

	para ( i := 1 ate 10 passo 1 )
	{
		leia ( "informe valor: ", Valor[i] );
	}
	leia ( "informe o valor a ser pesquisado: ", ValorPesquisado );
	N := 10;
	Valor[N+1] := ValorPesquisado;
	c := 1;
	a := 1;
	i := 1;
	enquanto ( Valor[i] <> ValorPesquisado )
	{
		i := i + 1;
		a := a + 1;
		c := c + 1;
	}
	se ( i == N + 1 )
	{
		escreva ( "valor nao encontrado" );
	}
	senao
	{
		escreva ( "valor encontrado na posicao ", i );
	}
	escreva ( "numero de comparacoes: ", c );
	escreva ( "numero de atribuicoes: ", a );
}
```

Comentário:

O algoritmo acima otimiza o número de comparações realizadas no laço ***enquanto*** durante o processo de pesquisa. Isto é conseguido eliminando-se o teste da condição ***(i <= N)*** do algoritmo anterior e permanecendo apenas o teste ***enquanto (Valor[i] <> ValorPesquisado)***. Para garantir que o indexador ***i*** não ultrapasse o limite ***N*** do vetor ***Valor*** é criado um nó, isto é, declara-se um vetor de dimensão ***N+1*** e utiliza-se a última posição para guardar o valor que se pretende pesquisar. Dessa forma, assegura-se que o valor a ser pesquisado sempre será encontrado, mas somente será válido se for de posição diferente de ***N+1***, pois esse elemento não pertence ao universo dos valores dados. Assim, para o pior caso, aquele em o valor pesquisado se encontra na última posição do vetor, o número de comparações realizado e ***N+1***. Isto da uma melhoria de desempenho de 50%

- Pesquisa sequencial III

```
algoritmo()
{
	vetor inteiro Valor[11];
	inteiro i;
	inteiro ValorPesquisado;
	inteiro N;
	inteiro c;
	inteiro a;

	para ( i := 1 ate 10 passo 1 )
	{
		leia ( "informe valor: ", Valor[i] );
	}
	leia ( "informe o valor a ser pesquisado: ", ValorPesquisado );
	N := 10;
	Valor[N+1] := ValorPesquisado;
	c := 1;
	a := 1;
	i := 1;
	enquanto ( Valor[i] <> ValorPesquisado )
	{
		se ( Valor[i+1] <> ValorPesquisado )
		{
			i := i + 2;
		}
		senao
		{
			i := i + 1;
		}
		c := c + 2;
		a := a + 1;
	}
	se ( i == N + 1 )
	{
		escreva ( "valor nao encontrado" );
	}
	senao
	{
		escreva ( "valor encontrado na posicao ", i );
	}
	escreva ( "numero de comparacoes: ", c );
	escreva ( "numero de atribuicoes: ", a );
}
```

Comentário

O algoritmo acima otimiza o numero de comparações realizadas no laço ***enquanto*** durante o processo de pesquisa. Isto é conseguido eliminando-se o teste da condição ***(i <= N)*** do algoritmo anterior e permanecendo apenas o teste ***enquanto (Valor[i] <> ValorPesquisado)***. Para garantir que o indexador ***i*** não ultrapasse o limite ***N*** do vetor ***Valor*** é criado um nó, isto é, declara-se um vetor de dimensão ***N+1*** e utiliza-se a última posição para guardar o valor que se pretende pesquisar. Dessa forma, assegura-se que o valor a ser pesquisado sempre será encontrado, mas somente será válido se

for de posição diferente de ***N+1***, pois esse elemento não pertence ao universo dos valores dados. Assim, para o pior caso, aquele em o valor pesquisado se encontra na última posição do vetor, o número de comparações realizado é ***N+1***. Isto da uma melhoria de desempenho de 50%. Esse valor ainda pode ser melhorado fazendo-se o indexador ***i*** saltar de 2 em 2 ao invés de 1 em 1. Como podemos ver no algoritmo acima, o número de comparações e equivalente aos casos anteriores mas o número de atribuições do indexador é reduzido à metade o que causa, segundo estudos, um aumento de 30% no desempenho da pesquisa.

- Pesquisa binária (dados ordenados)

```
algoritmo()
{
	vetor inteiro Valor[10];
	inteiro i;
	inteiro ValorPesquisado;
	inteiro inferior;
	inteiro superior;
	inteiro meio;
	inteiro N;

	para ( i := 1 ate 10 passo 1 )
	{
		leia ( "informe valor: ", Valor[i] );
	}
	leia ( "informe o valor a ser pesquisado: ", ValorPesquisado );
	N := 10;
	inferior := 1;
	superior := N;
	enquanto ( inferior <= superior )
	{
		meio := ( (inferior + superior) / 2 ) + 1;
		se ( Valor[meio] > ValorPesquisado )
		{
			superior := meio - 1;
		}
		senao
		{
			se ( Valor[meio] == ValorPesquisado )
			{
				inferior := superior + 1;
			}
			senao
			{
				inferior := meio + 1;
			}
		}
	}
	se ( Valor[meio] <> ValorPesquisado )
	{
		escreva ( "valor nao encontrado" );
	}
	senao
	{
		escreva ( "valor encontrado na posicao ", meio );
	}
}
```

Comentário:

- Ordenação numérica. (1ª versão)

```
algoritmo ()
{
	// ordena uma lista de 5 inteiros

	inteiro i;
	inteiro j;
	inteiro anterior;
	inteiro posterior;
	vetor inteiro A[5];

	// leitura dos valores

	para ( i := 1 ate 5 passo 1 )
	{
		leia ( "informe valor: ", A[i] );
	}

	// ordenacao dos valores lidos

	i := 1;

	enquanto ( i <= 5 )
	{
		anterior := A[i];
		j := i + 1;
		repita ( j > 5 )
		{
			posterior := A[j];
			se ( anterior > posterior )
			{
				A[i] := posterior;
				A[j] := anterior;
				anterior := posterior;
			}
			j := j + 1;
		}
		i := i + 1;
	}

	// exibicao do resultado

	para ( i := 1 ate 5 passo 1 )
	{
		escreva ( A[i] );
	}
}
```

Comentário:

- Ordenação numérica. (2ª versão)

```
algoritmo ()
{
	// ordena 10 valores reais

	inteiro atu;		// indexador do elemento de referencia
	inteiro pro;		// índexador dos elementos seguintes
	inteiro AUX;		// auxiliar na troca de valores entre elementos
	vetor inteiro N[10];

	para ( atu:=1 ate 10 passo 1)
	{
		leia ( "informe valor: ", N[atu] );
	}
	para ( atu:=1 ate 10 passo 1 )
	{
		pro := atu + 1;
		repita ( pro > 10 )
		{
			se ( N[pro] < N[atu] )
			{
				AUX := N[atu];
				N[atu] := N[pro];
				N[pro] := AUX;
			}
			pro := pro + 1;
		}
	}
	escreva ( "valores ordenados" );
	para ( atu:=1 ate 10 passo 1)
	{
		escreva ( N[atu] );
	}
}
```

Comentário:

- Ordenação alfabética

```
algoritmo ()
{
	inteiro i;
	inteiro j;
	inteiro n;
	vetor cadeia nome [5];
	cadeia aux;

	para ( i := 1 ate 5 passo 1 )
	{
		leia ( "informe nome: ", nome [i] );
	}
	para ( i := 1 ate 5 passo 1 )
	{
		j := i + 1;
		repita ( j > 5 )
		{
			n := compara ( nome[i], nome[j] );
			se ( n == -1 )
			{
				aux := nome [i];
				nome [i] := nome [j];
				nome [j] := aux;
			}
			j := j + 1;
		}
	}
	para ( i := 1 ate 5 passo 1 )
	{
		escreva ( nome [i] );
	}
}
```

Comentário:

Cálculo de diferença de dias entre duas datas

```
algoritmo()
{
	// calcula a diferenca de dias entre uma data qq e 21/06/1983

	inteiro dia;
	inteiro mes;
	inteiro ano;
	inteiro dif;
	inteiro ref;
	inteiro numdias;
	inteiro i;
	vetor inteiro diasdomes[12];

	numdias := 0;
	ref := 172;			// 21 de junho de 1983

	diasdomes [1]  := 31;
	diasdomes [2]  := 28;
	diasdomes [3]  := 31;
	diasdomes [4]  := 30;
	diasdomes [5]  := 31;
	diasdomes [6]  := 30;
	diasdomes [7]  := 31;
	diasdomes [8]  := 31;
	diasdomes [9]  := 30;
	diasdomes [10] := 31;
	diasdomes [11] := 30;
	diasdomes [12] := 31;

	leia ( "informe o dia: ", dia );
	leia ( "informe o mes: ", mes );
	leia ( "informe o ano: ", ano );

	se ( dia # 0 & dia <= diasdomes[mes] )
	{
		se ( mes >= 1 & mes <= 12 )
		{
			se ( ano == 1983 )
			{
				para ( i := 1 ate 12 passo 1 )
				{
					numdias := numdias + diasdomes[i];
				}
				numdias := numdias - diasdomes[mes];
				numdias := numdias + dia;
				dif := numdias - ref;
				escreva ( "diferenca igual a : ", dif );
			}
			senao
			{
				escreva ( "data invalida" );
			}
		}
	}
}
```

Comentário:

Quantidades de números primos entre até um valor dado

```
algoritmo ()
{
	inteiro ITER;
	inteiro SIZE;
	inteiro i;
	inteiro k;
	inteiro j;
	inteiro count;
	vetor inteiro flags[20];		// 256

	ITER := 20;
	SIZE := 20;

	escreva ( "processando ..." );

	para ( j := 1 ate ITER passo 1 )
	{
		count := 0;
		para ( i := 1 ate SIZE passo 1 )
		{
			flags[i] := 0;
		}
		para ( i := 2 ate SIZE passo 1 )
		{
			se ( flags[i]==0 )
			{
				// encontrou um primo
				count := count + 1;
				para ( k := i + i ate SIZE passo i )
				{
					flags[k] := 1;
					}
				}
			}
		}
	escreva ( "iteracoes", ITER );
	escreva ( "primos", count );
}
```

Comentário:

Matrizes

- Declaração, entrada e saída de valores para matrizes.

```
algoritmo ()
{
	inteiro i;
	inteiro j;
	matriz inteiro R [3:5];

	para ( i:=1 ate 3 passo 1 )
	{
		para ( j:=1 ate 5 passo 1 )
		{
			leia ( "informe valor: ", R[i][j] );
		}
	}
	para ( i:=1 ate 3 passo 1 )
	{
		para ( j:=1 ate 5 passo 1 )
		{
			escreva ( R[i][j] );
		}
	}
}
```

Comentário:

- Referência aos elementos de uma matriz

```
algoritmo ()
{
	inteiro i;
	inteiro j;
	matriz inteiro R [3:5];

	para ( i:=1 ate 3 passo 1 )
	{
		para ( j:=1 ate 5 passo 1 )
		{
			leia ( "informe valor: ", R [i,j] );
		}
	}

	para ( i:=1 ate 3 passo 1 )
	{
		para ( j:=1 ate 5 passo 1 )
		{
			R [i,j] := R [i,j] + 90;
		}
	}

	para ( i:=1 ate 3 passo 1 )
	{
		para ( j:=1 ate 5 passo 1 )
		{
			escreva ( R [i,j] );
		}
	}
}
```

Comentário:

- Matriz transposta.

```
algoritmo()
{
	matriz inteiro a[2:3];
	matriz inteiro b[3:2];
	inteiro i;
	inteiro j;

	para ( i := 1 ate 2 passo 1 )
	{
		escreva ( "linha ", i );
		para ( j := 1 ate 3 passo 1 )
		{
			leia ( "elemento: ", a[i,j] );
		}
	}
	para ( i := 1 ate 2 passo 1 )
	{
		para ( j := 1 ate 3 passo 1 )
		{
			b[j,i] := a[i,j];
		}
	}
	para ( i := 1 ate 3 passo 1 )
	{
		escreva ( "linha ", i );
		para ( j := 1 ate 2 passo 1 )
		{
			escreva ( "transposta: ", b[i,j] );
		}
	}
}
```

Comentário:

```
algoritmo()
{
	vetor cadeia mes[12];
	vetor inteiro QuantMotor1[12];
	vetor inteiro QuantMotor2[12];
	real LucroMensalMotor1;
	real LucroMensalMotor2;
	real CustoMensalMotor1;
	real CustoMensalMotor2;
	real LucroAnualMotor1;
	real LucroAnualMotor2;
	real CustoAnualMotor1;
	real CustoAnualMotor2;
	real LucroMotor1;
	real LucroMotor2;
	real CustoMotor1;
	real CustoMotor2;
	real LucroTotal;
	real CustoTotal;
	inteiro i;

	// inicializacao

	mes[1]  := "janeiro";
	mes[2]  := "fevereiro";
	mes[3]  := "marco";
	mes[4]  := "abril";
	mes[5]  := "maio";
	mes[6]  := "junho";
	mes[7]  := "julho";
	mes[8]  := "agosto";
	mes[9]  := "setembro";
	mes[10] := "outubro";
	mes[11] := "novembro";
	mes[12] := "dezembro";

	leia ( "informe o custo do motor 1: ", CustoMotor1 );
	leia ( "informe o lucro do motor 1:" , LucroMotor1 );
	i := 1;
	enquanto ( i <= 12 )
	{
		escreva ( mes[i] );
		leia ( ":quantidade produzida: ", QuantMotor1 [i] );
		i := i + 1;
	}
	leia ( "informe o custo do motor 2: ", CustoMotor2 );
	leia ( "informe o lucro do motor 2:" , LucroMotor2 );
	i := 1;
	enquanto ( i <= 12 )
	{
		escreva ( mes[i] );
		leia ( ":quantidade produzida: ", QuantMotor2 [i] );
		i := i + 1;
	}
	escreva ( "resultado mensal" );
	i := 1;
	enquanto ( i <= 12 )
	{
```

```
                LucroMensalMotor1 := QuantMotor1[i] * LucroMotor1;
                LucroMensalMotor2 := QuantMotor2[i] * LucroMotor2;
                CustoMensalMotor1 := QuantMotor1[i] * CustoMotor1;
                CustoMensalMotor2 := QuantMotor2[i] * CustoMotor2;
                escreva ( "Lucro mensal: ", LucroMensalMotor1, LucroMensalMotor2 );
                escreva ( "Custo mensal: ", CustoMensalMotor1, CustoMensalMotor2 );
                LucroAnualMotor1 := LucroAnualMotor1 + LucroMensalMotor1;
                LucroAnualMotor2 := LucroAnualMotor2 + LucroMensalMotor2;
                CustoAnualMotor1 := CustoAnualMotor1 + CustoMensalMotor1;
                CustoAnualMotor2 := CustoAnualMotor2 + CustoMensalMotor2;
                i := i + 1;
        }
        LucroTotal := LucroAnualMotor1 + LucroAnualMotor2;
        CustoTotal := CustoAnualMotor1 + CustoAnualMotor1;
        escreva ( "o lucro anual do motor 1 e: ", LucroAnualMotor1 );
        escreva ( "o custo anual do motor 1 e: ", CustoAnualMotor1 );
        escreva ( "o lucro anual do motor 2 e: ", LucroAnualMotor2 );
        escreva ( "o custo anual do motor 2 e: ", CustoAnualMotor2 );
        escreva ( "o lucro total: ", LucroTotal );
        escreva ( "o custo total: ", CustoTotal );
}
```

Comentário:

Registros

- Declaração de registros

```
algoritmo ()

{
	// registros

	registro R { inteiro i, real r, ascii a, cadeia c };

	leia ( "informe valor inteiro: ", R.i );
	leia ( "informe valor real: ", R.r );
	leia ( "informe valor ascii: ", R.a );
	leia ( "informe valor cadeia: ", R.c );

	escreva ( "valor informado inteiro: ", R.i );
	escreva ( "valor informado real: ", R.r );
	escreva ( "valor informado ascii: ", R.a );
	escreva ( "valor informado cadeia: ", R.c );

}
```

Comentário:

- Referência aos campos de um registro

```
algoritmo ()
{
	registro R
	{
		inteiro codigo,
		cadeia nome,
		ascii sexo,
		real salario
	};

	leia ( "informe o codigo...: ", R.codigo );
	leia ( "informe o nome.....: ", R.nome );
	leia ( "informe o sexo.....: ", R.sexo );
	leia ( "informe o salario..: ", R.salario );

	escreva ( "codigo...: ", R.codigo );
	escreva ( "nome.....: ", R.nome );
	escreva ( "sexo.....: ", R.sexo );
	escreva ( "salario..: ", R.salario );

}
```

Comentário:

Conjuntos

- Declaração de Conjuntos

```
algoritmo ()
{
	conjunto j [5]
	{
		inteiro num,
		real r,
		ascii a
	};

	escreva ( "fim de declaracao" );

}
```

Comentário:

- Referência aos elementos de um conjunto

```
algoritmo ()
{
	conjunto j[3]
	{
		inteiro num,
		real r,
		ascii a
	};

	leia ( "inteiro ", j[1].num );
	leia ( "real ", j[1].r );
	leia ( "ascii ", j[1].a );

	leia ( "inteiro ", j[2].num );
	leia ( "real ", j[2].r );
	leia ( "ascii ", j[2].a );

	leia ( "inteiro ", j[3].num );
	leia ( "real ", j[3].r );
	leia ( "ascii ", j[3].a );

	escreva ( "inteiro ", j[1].num );
	escreva ( "real ", j[1].r );
	escreva ( "ascii ", j[1].a );

	escreva ( "inteiro ", j[2].num );
	escreva ( "real ", j[2].r );
	escreva ( "ascii ", j[2].a );

	escreva ( "inteiro ", j[3].num );
	escreva ( "real ", j[3].r );
	escreva ( "ascii ", j[3].a );

}
```

Comentário:

- Atribuição de valores aos campos de um conjunto

```
algoritmo ()
{
	conjunto j [5]
	{
		inteiro num,
		real r,
		ascii a
	};
	inteiro i;

	para ( i := 1 ate 5 passo 1)
	{
		j[i].num := i;
		j[i].num := j[i].num + 10;
	}
	para ( i := 1 ate 5 passo 1)
	{
		escreva ( j[i].num );
	}
}
```

Comentário:

- Operação de busca em conjuntos

```
algoritmo ()
{
	conjunto Tabela[5] { inteiro codigo, cadeia profissao };
	inteiro i;		// subscritor para elementos do conjunto
	inteiro cod;	// codigo a ser pesquisado
	ascii resp;		// resposta sim/não

	resp := 's';
	enquanto ( resp == 's' )
	{
		para ( i := 1 ate 5 passo 1)
		{
			leia ( "codigo da profissao: ", Tabela[i].codigo );
			leia ( "nome da profissao: ", Tabela[i].profissao );
		}
		leia ( "informe o codigo procurado", cod );
		i := 1;
		repita ( i > 5 )
		{
			se ( Tabela[i].codigo == cod )
			{
				interrompa;
			}
			i := i + 1;
		}
		se ( i <= 5 )
		{
			escreva ( "profissao encontrada: ", Tabela[i].profissao );
		}
		senao
		{
			escreva ( "codigo nao consta na tabela" );
		}
		leia ( "deseja continuar (s/n): ", resp );
	}
}
```

Comentário:

```
Algoritmo ()
{
	registro Banco
	{
		real BalancoMinimoDiario,
		inteiro QuantidadeTransacoes,
		real TaxaServico
	};
	conjunto Conta[5]
	{
		inteiro Numero,
		real Saldo,
		real Diario
	};
	inteiro i;
	real valor;
	ascii operacao;

	// antes de abrir a agencia

	leia ( "balanco minimo diario..... ", Banco.BalancoMinimoDiario );
	leia ( "quantidade de transacoes.... ", Banco.QuantidadeTransacoes );
	leia ( "taxa de servico..... ", Banco.TaxaServico );

	repita ateque ( Banco.QuantidadeTransacoes == 0 )
	{
		leia ( "Numero da conta..... ", i );
		leia ( "Valor da transacao..... ", Valor );
		leia ( "Codigo da transacao..... ", Operacao );
		Conta[i].Numero := i;
		se ( Operacao == 'D' )
		{
			Conta[i].Diario := Conta[i].Diario + Valor;
			Conta[i].Saldo := Conta[i].Saldo + Valor;
		}
		senao
		{
			se ( Operacao == 'R' )
			{
				se ( Conta[i].Saldo > Valor )
				{
					Conta[i].Diario := Conta[i].Diario - Valor;
					Conta[i].Saldo := Conta[i].Saldo - Valor;
				}
				senao
				{
					escreva ( "saldo insuficiente para operacao" );
				}
			}
		}
		Banco.QuantidadeTransacoes := Banco.QuantidadeTransacoes - 1;
	}

	// verifica de diario foi menor que balanco minimo

	para ( i := 1 ate 5 passo 1 )
	{
		se ( Conta[i].Saldo < Banco.BalancoMinimoDiario )
```

```
                {
                        Conta[i].Saldo := Conta[i].Saldo –
                                                Conta[i].Saldo*Banco.TaxaServico;
                }
        }

        // exibindo os resultados

        para ( i := 1 ate 5 passo 1 )
        {
                escreva ( "conta nº ", i, " saldo R$ ", Conta[i].Saldo );
        }
}
```

Comentário:

Modularização – Funções de Usuário

- Declaração e definição de funções

```
algoritmo ()
{
	inteiro n1, n2, n3, n4;

	leia ("digite o valor de n1: ", n1 );

	leia ("digite o valor de n2: ", n2 );

	n3 := soma(n1,n2);
	n4 := diferenca(n1,n2);

	escreva ("a soma de n1 e n2 e:  ", n3 );
	escreva ("a diferenca de n1 e n2 e: ", n4 );
}

//--------------------------------------------------------------------------

soma ( inteiro a, inteiro b )
{
	inteiro x;

	x = a + b;
	retorne x;
}

//--------------------------------------------------------------------------

diferenca ( inteiro a, inteiro b )
{
	inteiro x;

	x = a - b;
	retorne x;
}
```

Comentário:

- Cálculo da Combinação Simples

```
algoritmo ()
{
	inteiro n;
	inteiro p;
	inteiro c;
	inteiro num;
	inteiro den1;
	inteiro den2;

	leia ( "informe o valor de n: ", n );
	leia ( "informe o valor de p: ", p );
	num := fatorial(n);
	c := n - p;
	den1 := fatorial(c);
	den2 := fatorial(p);
	c := num / ( den1 * den2 );
	escreva ( "A combinacao de n tomados p a p e",c );

}

//---------------------------------------------------------------------------

fatorial (inteiro f)
{

	// retorna o fatorial do numero n

	inteiro fat;
	fat := 1;

	enquanto ( f > 1 )
	{
		fat := fat * f;
		f := f - 1;
	}
	retorne fat;
}
```

Comentário:

A combinação simples permite responder a perguntas do seguinte tipo:
1.Quantas comissões de 3 elementos podemos formar com um grupo de 4 pessoas ?
Resposta. É a combinação de 4 tomados 3 a 3, representado por C(4,3).
2.De quantas maneiras se pode escalar um time de basquete com uma equipe de 7 jogadores ?
Resposta. É a combinação de 7 tomados 5 a 5. C(7,5)

- Números Primos

```
algoritmo ()
{
	// exibe os numeros primos de 2 a 500

	inteiro i;		// os numeros de 2 a 500
	inteiro n;		// limite superior da tabela
	inteiro p;		// 1 se for primo, 0 se não

	n := 500;
	i := 2;

	enquanto ( i <= n )
	{
		p := primo (i);
		se ( p == 1 )
		{
			// entao i e primo
			escreva (i);
		}
		i := i + 1;
	}
}

//---------------------------------------------------------------------------

primo ( inteiro k )
{
	inteiro j;		// todos os anteriores a k
	inteiro h;		// resto da divisao dos j por k
	inteiro f;		// retorno da funcao 0-nao primo, 1-primo

	j := 2;
	h := 0;
	f := 1;

	enquanto ( j < k )
	{
		h := k % j;		// h e o resto da divisao de k por j
		se ( h == 0 )
		{
			// divisao exata, logo k nao e primo
			f := 0;		// identifica que nao e primo
			j := k;		// para forcar a saida do laco
		}
		j := j + 1;		// incremento da variavel de controle
	}

	retorne f;		// retorna da funcao ( 0 ou 1 )
}
```

Comentário:

- Fatorial

```
algoritmo ()
{
	// chamada de funcao

	inteiro L;		// limite informado
	inteiro F;		// fatorial de cada numero
	inteiro s;		// somatorio dos fatoriais

	inteiro i;

	escreva ( "efetua o somatorio dos fatoriais menores que L" );
	escreva ("digite o limite L: ");
	leia (L);

	repita ( i > L )
	{
		F := fatorial (i);
		s := s + F;
		i := i + 1;
	}
	escreva ( "a soma dos fatoriais inferiores e: " );
	escreva ( s );
}


fatorial ( inteiro n )
{
	// calcula o fatorial de um numero

	inteiro f;		// fatorial do numero n

	f := 1;		// elemento neutro da multiplicacao

	repita ( n == 1 )
	{
		f := f * n;
		n := n - 1;
	}
	retorne (f);
}
```

comentário:

- Ordenação alfabética

```
algoritmo ()
{
	inteiro i;
	inteiro j;
	inteiro n;
	vetor cadeia nome [5];

	para ( i := 1 ate 5 passo 1 )
	{
		leia ( "informe nome: ", nome [i] );
	}
	para ( i := 1 ate 5 passo 1 )
	{
		j := i + 1;
		repita ( j > 5 )
		{
			n := ordem ( nome[i], nome[j] );
			se ( n == -1 )
			{
				troca ( nome [i], nome [j] );
			}
			j := j + 1;
		}
	}
	para ( i := 1 ate 5 passo 1 )
	{
		escreva ( nome [i] );
	}
}

//--------------------------------------------------------------------------

troca ( cadeia nome[i], cadeia nome[j] )

{
	cadeia aux;

	aux := nome [i];
	nome [i] := nome [j];
	nome [j] := aux;

	retorne;

}
```

Comentário:

- Comprimento

```
algoritmo()
{
	// exemplo de funcao criada pelo usuario programador

	cadeia m;
	inteiro i;

	leia ( "informe uma cadeia: ", m );
	i := MinhaFuncaoComprimento ( m );
	escreva ( "o comprimento da cadeia e: ", i );
}


MinhaFuncaoComprimento ( cadeia x )
{
	inteiro i;
	ascii c;

	c := ' ';
	i := 1;
	enquanto ( i <= 128 & c <> 0 )
	{
		c := SubCadeia ( x, i, 1 );
		i := i + 1;
	}
	i := i - 2;		// uma iteração de i e um <ENTER>
	retorne i;
}
```

Comentário:

- Rei

```
algoritmo()
{
     inteiro col;
     inteiro sentido;

     col := 2;
     sentido := 1;

     limpatela();

     enquanto (1)
     {
           se ( col == 55 )
           {
                 apaga_rei ( col);
                 pisca_rei ( col);
                 sentido := 0;
           }
           exibe_rei ( col );
           apaga_rei ( col );
           se ( sentido == 1 )
           {
                 col := col + 1;
           }
           senao
           {
                 col := col - 1;
           }
     }
}

exibe_rei ( inteiro col )

{
     posicao (2, col);  escreva ( " ======================== " );
     posicao (3, col);  escreva ( "          0000            " );
     posicao (4, col);  escreva ( "        |||||||||         " );
     posicao (5, col);  escreva ( "       0  ^   ^  0        " );
     posicao (6, col);  escreva ( "     000  0   0  000      " );
     posicao (7, col);  escreva ( "    0000    |     0000    " );
     posicao (8, col);  escreva ( "   0000   (===)    0000   " );
     posicao (9, col);  escreva ( "      0     0     0       " );
     posicao (10,col);  escreva ( "       0 0 0 0 0 0        " );
     posicao (11,col);  escreva ( "         000000           " );
     posicao (12,col);  escreva ( "          0000            " );
     posicao (13,col);  escreva ( "           00             " );
     posicao (14,col);  escreva ( " ======================== " );
}

apaga_rei ( inteiro col )

{
     posicao (2, col);  escreva ( "                          " );
     posicao (3, col);  escreva ( "                          " );
     posicao (4, col);  escreva ( "                          " );
     posicao (5, col);  escreva ( "                          " );
```

```
     posicao (6, col);  escreva ( "                          " );
     posicao (7, col);  escreva ( "                          " );
     posicao (8, col);  escreva ( "                          " );
     posicao (9, col);  escreva ( "                          " );
     posicao (10,col);  escreva ( "                          " );
     posicao (11,col);  escreva ( "                          " );
     posicao (12,col);  escreva ( "                          " );
     posicao (13,col);  escreva ( "                          " );
     posicao (14,col);  escreva ( "                          " );
}


pisca_rei ( inteiro col )

{
     posicao (2, col);  escreva ( " ======================== " );
     posicao (3, col);  escreva ( "          0000            " );
     posicao (4, col);  escreva ( "        |||||||||         " );
     posicao (5, col);  escreva ( "       0  ^   ^  0        " );
     posicao (6, col);  escreva ( "     000  -   0  000      " );
     posicao (7, col);  escreva ( "    0000    |    0000     " );
     posicao (8, col);  escreva ( "   0000   (===)    0000   " );
     posicao (9, col);  escreva ( "      0     0     0       " );
     posicao (10,col);  escreva ( "       0 0 0 0 0 0        " );
     posicao (11,col);  escreva ( "         000000           " );
     posicao (12,col);  escreva ( "          0000            " );
     posicao (13,col);  escreva ( "           00             " );
     posicao (14,col);  escreva ( " ======================== " );
}
```

Comentário:

- Valores por extenso

```
algoritmo()
{
	inteiro N;
	cadeia Extenso;
	ascii resp;

	resp := 's';

	repita ateque ( resp == 'n' | resp == 'N' )
	{
		leia ( "informe um numero: ", N );
		Extenso := ExtensoAteTresDigitos ( N );
		escreva ( Extenso );
		escreva ( "deseja continuar (s/n)" );
		resp := Tecla();
	}
}


ExtensoAteTresDigitos ( inteiro Num )
{
	cadeia T;

	vetor cadeia unidade[9];
	vetor cadeia dezena[9];
	vetor cadeia centena[9];
	vetor cadeia unidez[9];
	inteiro i, j, AUX;
	vetor inteiro p[3];

	unidade[1] := "um";
	unidade[2] := "dois";
	unidade[3] := "tres";
	unidade[4] := "quatro";
	unidade[5] := "cinco";
	unidade[6] := "seis";
	unidade[7] := "sete";
	unidade[8] := "oito";
	unidade[9] := "nove";

	dezena[1] := "dez";
	dezena[2] := "vinte";
	dezena[3] := "trinta";
	dezena[4] := "quarenta";
	dezena[5] := "cinquenta";
	dezena[6] := "sessenta";
	dezena[7] := "setenta";
	dezena[8] := "oitenta";
	dezena[9] := "noventa";

	centena[1] := "cento";
	centena[2] := "duzentos";
	centena[3] := "trezentos";
	centena[4] := "quatrocentos";
	centena[5] := "quinhentos";
	centena[6] := "seiscentos";
```

```
centena[7] := "setecentos";
centena[8] := "oitocentos";
centena[9] := "novecentos";

unidez[1] := "onze";
unidez[2] := "doze";
unidez[3] := "treze";
unidez[4] := "quatorze";
unidez[5] := "quinze";
unidez[6] := "dezesseis";
unidez[7] := "dezessete";
unidez[8] := "dezoito";
unidez[9] := "dezenove";

j := 1;
i := 3;

T := "";

AUX := Num;
repita ateque ( i < 1 )
{
        p[j] := ParteInt ( AUX / ( 10 ^ ( i - 1 ) ) );
        AUX := AUX - ( p[j] * ( 10 ^ ( i - 1 ) ) );
        j := j + 1;
        i := i - 1;
}

se ( Num > 0 )
{
        se ( Num == 100 )
        {
                T := "Cem";
        }
        senao
        {
                se ( p[1] > 0 )
                {
                        T := Centena[p[1]];

                        se ( p[2] + p[3] > 0 )
                        {
                                T := Concatena ( T, " e" );
                        }
                }
                se ( p[2] == 1 & p[3] > 0 )
                {
                        T := Concatena ( T, " ", UniDez[p[3]], " " );
                }
                senao
                {
                        se ( p[2] > 0 )
                        {
                                T := Concatena ( T, " ",  Dezena[p[2]] );
                                se ( p[3] > 0 )
                                {
                                        T := Concatena ( T, " e" );
                                }
```

```
                                }
                                se ( p[3] > 0 )
                                {
                                        T := Concatena ( T, " ", Unidade[p[3]] );
                                }
                        }
                }
        }
        retorne T;
}
```

Comentário:

- Jogo da Velha

```
inteiro i;
matriz ascii M[3:3];
ascii ganhador;

algoritmo()
{
	// jogo da velha

	InicializaJogoVelha ();

	//ExibirQuadro ();

	enquanto ( ganhador == ' ' )
	{
		DesafianteJoga ();
		ganhador := Verifica ();
		se ( ganhador == ' ' )
		{
			ComputadorJoga ();
			ganhador = Verifica ();
		}
	}
	ExibeResultado ( );
}


InicializaJogoVelha ()
{
	inteiro i,j;

	para ( i := 1 ate 3 passo 1 )
	{
		para ( j := 1 ate 3 passo 1 )
		{
			M[i,j] := ' ';
		}
	}
	retorne;
}


ExibirQuadro ()
{
	inteiro i;

	para ( i := 1 ate 3 passo 1 )
	{
		escreva ( M[i][1], " | ", M[i][2], " | ", M[i][3] );
		se ( i <> 2 )
		{
			escreva ( "--- | --- | ---" );
		}
	}
}

DesafianteJoga ()
```

```
{
	// vez do desafiante

	inteiro x,y;

	leia ( "linha: ", x, "coluna: ", y );

	se ( M[x,y] <> ' ' )
	{
		escreva ( "posicao invalida, tente novamente" );
		DesafianteJoga ();
	}
	senao
	{
		M[x,y] := 'X';
	}
}


Verifica ( )
{
	// verifica se houve vencedor

	inteiro i,j;

	// verifica as linhas
	i := 1;
	enquanto ( i <= 3 )
	{
		se ( ( M[i,1] == M[i,2] ) & ( M[i,1] == M[i,3] ) )
		{
			retorne M[i,1];
		}
		i := i + 1;
	}
	// verifica as colunas
	i := 1;
	enquanto ( i <= 3 )
	{
		se ( M[1,i] == M[2,i] & M[1,i] == M[3,i] )
		{
			retorne M[1,i];
		}
		i := i + 1;
	}
	// verifica as diagonais
	se ( M[1,1] == M[2,2] & M[2,2] == M[3,3] )
	{
		retorne M[1,1];
	}
	senao
	{
		se ( M[1,3] == M[2,2] & M[2,2] == M[3,1] )
		{
			retorne M[1,3];
		}
	}
	// empate
```

```
        para ( i := 1 ate 3 passo 1 )
        {
                para ( j := 1 ate 3 passo 1 )
                {
                        se ( M[i,j] == ' ' )
                        {
                                interrompa;
                        }
                        j := j + 1;
                }
                se ( M[i,j] == ' ' )
                {
                        interrompa;
                }
                i := i + 1;
        }
        se ( i*j > 9 )
        {
                retorne 'E';
        }
}

ComputadorJoga ()
{
        // vez do computador jogar ( joga na primeira casa livre )

        inteiro i,j;

        para ( i := 1 ate 3 passo 1 )
        {
                para ( j := 1 ate 3 passo 1 )
                {
                        se ( M[i,j] == ' ' )
                        {
                                interrompa;
                        }
                        j := j + 1;
                }
                se ( M[i,j] == ' ' )
                {
                        interrompa;
                }
                i := i + 1;
        }
        se ( i*j <= 9 )
        {
                M[i,j] := 'O';
        }
}

ExibeResultado ( )
{
        selecao
        {
                caso ( ganhador == 'X' )
                {
                        escreva ( "voce ganhou !" );
                }
```

```
                caso ( ganhador == 'O' );
                {
                        escreva ( "eu ganhei !!!" );
                }
                senao
                {
                        escreva ( "empatamos !" );
                }
        }
}
```

Comentário:

Arquivos

Tipos de arquivos

- Sequencial
- Indexado
- Texto

Arquivo Sequencial

Funções para o tratamento de arquivos sequenciais

- Declaração
- Vincular()
- Associar()
- Criar()
- Abrir()
- Ler()
- Gravar()
- Posicionar()

Algoritmos para manutenção em arquivo sequencial

- Criação de arquivo sequencial

```
algoritmo()
{
	arquivo sequencial ceva;
	registro Cerveja
	{
		inteiro codigo,
		cadeia nome,
		inteiro estoque
	};

	vincular ( ceva, "cerveja.dat" );
	associar ( ceva, Cerveja );
	criar ( ceva );
}
```

Comentário:

O algoritmo acima ilustra o mínimo código necessário para se criar um arquivo sequencial. Um arquivo qualquer (sequencial, indexado ou texto) é criado através de uma chamada à função ***cria()*** passando como argumento para essa função o nome lógico do arquivo ou o seu nome físico. O nome lógico do arquivo é aquele especificado em sua declaração, ***ceva*** no exemplo acima e, o nome físico é aquele especificado na chamada da função ***vincular()***, no exemplo, ***cerveja.dat***. A chamada à função ***vincular()*** é necessária para a criação de qualquer arquivo, uma vez que, é através dela que será realizado o vínculo entre a variável de memória que irá manipular os dados para o arquivo e o arquivo físico propriamente dito. Um arquivo sequencial será sempre um arquivo associado a um determinado grupo de informações, ou seja, a um registro. Desse modo, torna-se necessário estabelecer uma associação entre o arquivo sequencial que será criado e o registro de informações que definirá o seu tipo (lay-out). Essa operação é realizada através de uma chamada a função ***associar()*** passando como argumentos a identificação do arquivo sequencial (seu nome lógico ou seu nome físico) e o nome do registro que definirá o seu tipo (a sua estrutura). Uma chamada a função ***cria()***, para arquivos sequenciais, irá gerar um erro de execução se o mesmo ainda não estiver sido vinculado e associado. Observe finalmente que, não é necessário ***fechar*** o arquivo uma vez que, na operação de criação o mesmo não é aberto.

- Inclusão de dados em arquivo sequencial

```
algoritmo()
{
```

```
	arquivo sequencial ceva;
	registro Cerveja
	{
		inteiro codigo,
		cadeia nome,
		inteiro estoque
	};
	inteiro CodigoCerveja, Quantidade, achou;
	cadeia NomeCerveja;
	ascii RESP;

	vincular ( ceva, "cerveja.dat" );
	associar ( ceva, Cerveja );
	abrir ( ceva );
	repita ateque ( RESP == 'N' | RESP == 'n' )
	{
		leia ( "Codigo da cerveja..... ", CodigoCerveja );
		posicionar ( ceva, INICIO );
		achou := 0;
		enquanto ( achou == 0 & fda ( ceva ) == 0 )
		{
			ler ( ceva, Cerveja );
			se ( Cerveja.codigo == CodigoCerveja )
			{
				achou := 1;
			}
		}
		se ( achou == 0 )
		{
			leia ( "Nome da cerveja..... ", NomeCerveja );
			leia ( "Quantidade..... ", Quantidade );
			leia ( "confirma inclusao (s/n): ", RESP );
			se ( RESP == 'S' | RESP == 's' )
			{
				Cerveja.Codigo := CodigoCerveja;
				Cerveja.Nome := NomeCerveja;
				Cerveja.Estoque := Quantidade;
				Gravar ( ceva, Cerveja );
			}
		}
		senao
		{
			escreva ( "codigo ja cadastrado !" );
		}
		leia ( "deseja continuar (s/n): ", RESP );
	}
	fechar ( ceva );
}
```

Comentário:

O algoritmo acima exemplifica o processo de inclusão de dados (registros) em um arquivo sequencial. Em um arquivo sequencial, o controle da duplicidade de informações deve ser realizado pelo usuário (programador). No exemplo, foi assumido que o campo codigo (Cerveja.Codigo) será a informação que permitirá realizar esse controle. O campo Cerveja.Codigo é considerado então a chave de acesso ao arquivo pois, é através dele que

pode-se recuperar os dados que já estão armazenados no arquivo e, é através dele também que pode-se verificar se um dado código já existe no arquivo ou não. Desse modo, toda vez que um novo registro estiver sendo incluído no arquivo, deve-se antes verificar a sua existência, pois de outro modo, uma duplicidade será gerada. Assim, deve-se percorrer o arquivo, do início ao final, sequencialmente (daí o seu nome) para verificar se o código que se deseja incluir já existe no mesmo. Como o algoritmo de inclusão está dentro de um laço de inclusões, é necessário reposicionar o indicador do arquivo em seu início, o que é conseguido com a chamada a função ***posicionar (ceva, INICIO)***. Como a leitura do arquivo é sequencial até o seu final, é necessário descobrir quando o fim do arquivo foi atingido, pois do contrário um erro seria gerado pela tentativa de se ler o arquivo após o seu final. A função ***fda()*** (fim de arquivo) permite identificar quando o fim de arquivo foi atingido. A função ***fda()*** retorna um valor diferente de 0 (zero) quando o fim de arquivo foi atingido. Então o nosso laço de pesquisa será executado enquanto o fim de arquivo não for atingido. Por outro lado, se um registro foi encontrado o laço de pesquisa também deverá ser encerrado. Essa condição é controlada pela variável ***achou***, que receberá o valor 1 se um registro com valor de Cerveja.Codigo for igual ao código pesquisado. Desse modo, a expressão ***enquanto (achou == 0 & fda (ceva) == 0)*** controla o laço de pesquisa. Se o código não existe, então as informações complementares são solicitadas e é realizada a gravação do registro. A gravação de registro em arquivo sequencial é realizada com a chamada a função gravar() passando como argumentos o identificador do arquivo e o identificador do registro associado. Finalmente, é boa prática de programação definir variáveis de leitura correspondentes aos campos de um registro do arquivo sequencial, posteriormente, antes da operação de gravação, esses dados são movidos para os campos do registro e então a operação de gravação tem início. (este procedimento foi realizado no algoritmo).

Observação: Em um arquivo sequencial, as informações (os registros) estão colocadas sequencialmente no arquivo, ou seja, na ordem em foram gravadas. Isto significa que, embora o nosso campo de pesquisa, Cerveja.Codigo seja o campo chave de acesso, o mesmo não estará ordenado dentro do arquivo. Aliás essa é uma das características básicas do arquivo sequencial, não há obrigatoriedade de organização (indexação) dos dados.

- Consulta de dados em arquivo sequencial

```
algoritmo()
{
	arquivo sequencial ceva;
	registro Cerveja
	{
		inteiro codigo,
		cadeia nome,
		inteiro estoque
	};
	inteiro CodigoCerveja;
	inteiro Quantidade;
	cadeia NomeCerveja;
	ascii RESP;
	inteiro achou;

	vincular ( ceva, "cerveja.dat" );
	associar ( ceva, Cerveja );
	abrir ( ceva );
	repita ateque ( RESP == 'N' | RESP == 'n' )
	{
		leia ( "Codigo da cerveja..... ", CodigoCerveja );
		posicionar ( ceva, INICIO );
		achou := 0;
		enquanto ( achou == 0 & fda ( ceva ) == 0 )
		{
			ler ( ceva, Cerveja );
			se ( Cerveja.codigo == CodigoCerveja )
			{
				achou := 1;
			}
		}
		se ( achou == 1 )
		{
			escreva ( "codigo da cerveja..... ", Cerveja.Codigo );
			escreva ( "nome da cerveja..... ", Cerveja.Nome );
			escreva ( "estoque atual..... ", Cerveja.Estoque );
		}
		senao
		{
			escreva ( "codigo nao cadastrado !" );
		}
		leia ( "deseja continuar (s/n): ", RESP );
	}
	fechar ( ceva );
}
```

Comentário:

- Alteração de dados em arquivo sequencial

```
algoritmo()
{
	arquivo sequencial ceva;
	registro Cerveja
	{
		inteiro codigo,
		cadeia nome,
		inteiro estoque
	};
	inteiro CodigoCerveja, Quantidade, achou;
	cadeia NomeCerveja;
	ascii RESP;

	vincular ( ceva, "cerveja.dat" );
	associar ( ceva, Cerveja );
	abrir ( ceva );
	repita ateque ( RESP == 'N' | RESP == 'n' )
	{
		leia ( "Codigo da cerveja..... ", CodigoCerveja );
		posicionar ( ceva, INICIO );
		achou := 0;
		enquanto ( achou == 0 & fda ( ceva ) == 0 )
		{
			ler ( ceva, Cerveja );
			se ( Cerveja.codigo == CodigoCerveja )
			{
				achou := 1;
			}
		}
		se ( achou == 1 )
		{
			escreva ( "Codigo da cerveja..... ", Cerveja.Codigo );
			escreva ( "Nome da cerveja..... ", Cerveja.Nome );
			escreva ( "Quantidade..... ", Cerveja.Estoque );
			leia ( "Nome da cerveja..... ", NomeCerveja );
			leia ( "Quantidade..... ", Quantidade );
			leia ( "confirma alteracao (s/n): ", RESP );
			se ( RESP == 'S' | RESP == 's' )
			{
				Cerveja.Codigo := CodigoCerveja;
				Cerveja.Nome := NomeCerveja;
				Cerveja.Estoque := Quantidade;
				ReGravar ( ceva, Cerveja );
			}
		}
		senao
		{
			escreva ( "codigo nao cadastrado !" );
		}
		leia ( "deseja continuar (s/n): ", RESP );
	}
	fechar ( ceva );
}
```

- Exclusão de dados em arquivo sequencial

```
algoritmo()
{
	arquivo sequencial ceva;
	registro Cerveja
	{
		inteiro codigo,
		cadeia nome,
		inteiro estoque
	};
	inteiro CodigoCerveja, achou;
	ascii RESP;

	vincular ( ceva, "cerveja.dat" );
	associar ( ceva, Cerveja );
	abrir ( ceva );
	repita ateque ( RESP == 'N' | RESP == 'n' )
	{
		leia ( "Codigo da cerveja..... ", CodigoCerveja );
		posicionar ( ceva, INICIO );
		achou := 0;
		enquanto ( achou == 0 & fda ( ceva ) == 0 )
		{
			ler ( ceva, Cerveja );
			se ( Cerveja.codigo == CodigoCerveja )
			{
				achou := 1;
			}
		}
		se ( achou == 1 )
		{
			escreva ( "Codigo da cerveja..... ", Cerveja.Codigo );
			escreva ( "Nome da cerveja..... ", Cerveja.Nome );
			escreva ( "Quantidade..... ", Cerveja.Estoque );
			leia ( "Nome da cerveja..... ", NomeCerveja );
			leia ( "Quantidade..... ", Quantidade );
			leia ( "confirma exclusao (s/n): ", RESP );
			se ( RESP == 'S' | RESP == 's' )
			{
				Cerveja.Codigo := CodigoCerveja;
				Cerveja.Nome := NomeCerveja;
				Cerveja.Estoque := Quantidade;
				Deletar ( ceva, Cerveja );
			}
		}
		senao
		{
			escreva ( "codigo nao cadastrado !" );
		}
		leia ( "deseja continuar (s/n): ", RESP );
	}
	fechar ( ceva );
}
```

- Entrada de dados com atualização em arquivo sequencial

```
algoritmo()
{
	arquivo sequencial ceva;
	registro Cerveja
	{
		inteiro codigo,
		cadeia nome,
		inteiro estoque
	};
	inteiro CodigoCerveja;
	inteiro Quantidade;
	ascii RESP;
	inteiro achou;

	vincular ( ceva, "cerveja.dat" );
	associar ( ceva, Cerveja );
	abrir ( ceva );
	repita ateque ( RESP == 'N' | RESP == 'n' )
	{
		leia ( "Codigo da cerveja..... ", CodigoCerveja );
		posicionar ( ceva, INICIO );
		achou := 0;
		enquanto ( achou == 0 & fda ( ceva ) == 0 )
		{
			ler ( ceva, Cerveja );
			se ( Cerveja.codigo == CodigoCerveja )
			{
				achou := 1;
			}
		}
		se ( achou == 1 )
		{
			escreva ( "Codigo da cerveja..... ", Cerveja.Codigo );
			escreva ( "Nome da cerveja..... ", Cerveja.Nome );
			escreva ( "Quantidade..... ", Cerveja.Estoque );
			leia ( "Quantidade..... ", Quantidade );
			leia ( "confirma entrada (s/n): ", RESP );
			se ( RESP == 'S' | RESP == 's' )
			{
				Cerveja.Estoque := Cerveja.Estoque + Quantidade;
				ReGravar ( ceva, Cerveja );
			}
		}
		senao
		{
			escreva ( "codigo nao cadastrado !" );
		}
		leia ( "deseja continuar (s/n): ", RESP );
	}
	fechar ( ceva );
}
```

- Saída de dados com atualização em arquivo sequencial

```
algoritmo()
```

```
{
	arquivo sequencial ceva;
	registro Cerveja
	{
		inteiro codigo,
		cadeia nome,
		inteiro estoque
	};
	inteiro CodigoCerveja;
	inteiro Quantidade;
	ascii RESP;
	inteiro achou;

	vincular ( ceva, "cerveja.dat" );
	associar ( ceva, Cerveja );
	abrir ( ceva );
	repita ateque ( RESP == 'N' | RESP == 'n' )
	{
		leia ( "Codigo da cerveja..... ", CodigoCerveja );
		posicionar ( ceva, INICIO );
		achou := 0;
		enquanto ( achou == 0 & fda ( ceva ) == 0 )
		{
			ler ( ceva, Cerveja );
			se ( Cerveja.codigo == CodigoCerveja )
			{
				achou := 1;
			}
		}
		se ( achou == 1 )
		{
			escreva ( "Codigo da cerveja..... ", Cerveja.Codigo );
			escreva ( "Nome da cerveja..... ", Cerveja.Nome );
			escreva ( "Quantidade..... ", Cerveja.Estoque );
			leia ( "Quantidade..... ", Quantidade );
			leia ( "confirma saida (s/n): ", RESP );
			se ( RESP == 'S' | RESP == 's' )
			{
				se ( Quantidade > Cerveja.Estoque )
				{
					escreva ( "estoque nao disponivel" );
				}
				senao
				{
					Cerveja.Estoque := Cerveja.Estoque - Quantidade;
					ReGravar ( ceva, Cerveja );
				}
			}
		}
		senao
		{
			escreva ( "codigo nao cadastrado !" );
		}
		leia ( "deseja continuar (s/n): ", RESP );
	}
	fechar ( ceva );
}
```

Comentário

- Manutenção em arquivo sequencial – todas as operações básicas em um único algoritmo

```
algoritmo()
{
	arquivo sequencial ceva;
	registro Cerveja
	{
		inteiro codigo,
		cadeia nome,
		inteiro estoque
	};
	inteiro CodigoCerveja;
	inteiro Quantidade;
	cadeia NomeCerveja;
	ascii RESP;
	inteiro achou;
	inteiro opcao;

	vincular ( ceva, "cerveja.dat" );
	associar ( ceva, Cerveja );
	abrir ( ceva );
	repita ateque ( RESP == 'N' | RESP == 'n' )
	{
		escreva ( "1-inclui, 2-altera, 3-consulta, 4-exclui 5-finaliza" );
		leia ( "informe opcao: ", opcao );
		se ( opcao <> 5 )
		{
			leia ( "Codigo da cerveja..... ", CodigoCerveja );
			posicionar ( ceva, INICIO );
			achou := 0;
			enquanto ( achou == 0 & fda ( ceva ) == 0 )
			{
				ler ( ceva, Cerveja );
				se ( Cerveja.codigo == CodigoCerveja )
				{
					achou := 1;
				}
			}
			se ( achou == 0 )
			{
				se ( opcao == 1 )
				{
					leia ( "Nome da cerveja..... ", NomeCerveja );
					leia ( "Quantidade..... ", Quantidade );
					leia ( "confirma inclusao (s/n): ", RESP );
					se ( RESP == 'S' | RESP == 's' )
					{
						Cerveja.Codigo := CodigoCerveja;
						Cerveja.Nome := NomeCerveja;
						Cerveja.Estoque := Quantidade;
						Gravar ( ceva, Cerveja );
					}
				}
				senao
				{
					escreva ( "codigo nao cadastrado !" );
```

```
                        }
                    }
                    senao
                    {
                        se ( opcao == 1 )
                        {
                            escreva ( "codigo ja cadastrado" );
                        }
                        senao se ( opcao == 2 )
                        {
                            escreva ( "Codigo da cerveja. ", Cerveja.Codigo );
                            escreva ( "Nome da cerveja.... ", Cerveja.Nome );
                            escreva ( "Quantidade..... ", Cerveja.Estoque );
                            leia ( "Nome da cerveja..... ", NomeCerveja );
                            leia ( "Quantidade..... ", Quantidade );
                            leia ( "confirma alteracao (s/n): ", RESP );
                            se ( RESP == 'S' | RESP == 's' )
                            {
                                Cerveja.Codigo := CodigoCerveja;
                                Cerveja.Nome := NomeCerveja;
                                Cerveja.Estoque := Quantidade;
                                ReGravar ( ceva, Cerveja );
                            }
                        }
                        senao se ( opcao == 3 )
                        {
                            escreva ( "codigo da cerveja. ", Cerveja.Codigo );
                            escreva ( "nome da cerveja... ", Cerveja.Nome );
                            escreva ( "estoque atual... ", Cerveja.Estoque );
                        }
                        senao se ( opcao == 4 )
                        {
                            escreva ( "Codigo da cerveja. ", Cerveja.Codigo );
                            escreva ( "Nome da cerveja.... ", Cerveja.Nome );
                            escreva ( "Quantidade..... ", Cerveja.Estoque );
                            leia ( "confirma exclusao (s/n): ", RESP );
                            se ( RESP == 'S' | RESP == 's' )
                            {
                                Deletar ( ceva, Cerveja );
                            }
                        }
                    }
                }
                leia ( "deseja continuar (s/n): ", RESP );
        }
        fechar ( ceva );
}
```

Comentário:

- Cópia de arquivo sequencial

```
algoritmo()
{
	arquivo sequencial ceva1;
	arquivo sequencial ceva2;
	registro Cerveja
	{
		inteiro codigo,
		cadeia nome,
		inteiro estoque
	};

	vincular ( ceva1, "cerveja.dat" );
	vincular ( ceva2, "cerveja2.dat" );
	associar ( ceva1, Cerveja );
	associar ( ceva2, Cerveja );
	criar ( ceva2 );
	abrir ( ceva1 );
	abrir ( ceva2 );

	posicionar ( ceva1, INICIO );
	enquanto ( fda ( ceva1 ) == 0 )
	{
		ler ( ceva1, Cerveja );
		se ( Cerveja.codigo <> 0 )
		{
			Gravar ( ceva2, Cerveja );
		}
		// posicionar ( ceva1, PROXIMO );
	}
	fechar ( ceva1 );
	fechar ( ceva2 );
}
```

Comentário:

- Filtro de dados para arquivos sequenciais

```
algoritmo()
{
	registro Reg
	{
		inteiro codigo,
		cadeia nome,
		ascii sexo
	};
	arquivo sequencial pessoas;
	arquivo sequencial homens;
	arquivo sequencial mulheres;
	ascii RESP;

	vincular ( pessoas, "pessoas.dat" );
	vincular ( homens, "homens.dat" );
	vincular ( mulheres, "mulheres.dat" );
	associar ( pessoas, Reg );
	associar ( homens, Reg );
	associar ( mulheres, Reg );
	criar ( homens );
	criar ( mulheres );
	abrir ( mulheres );
	abrir ( pessoas );
	abrir ( homens );

	posicionar ( pessoas, INICIO );

	enquanto ( fda ( pessoas ) == 0 )
	{
		ler ( pessoas, Reg );
		se ( Reg.codigo <> 0 )
		{
			se ( Reg.sexo == 'f' )
		{
			Gravar ( mulheres, Reg );
		}
		senao
		{
			Gravar ( homens, Reg );
		}
		//posicionar ( pessoas, PROXIMO );
	}
	fechar ( pessoas );
	fechar ( homens );
	fechar ( mulheres );
}
```

Comentário:

Arquivo Indexado

Funções para o tratamento de arquivos indexados

- Declaração
- Vincular()
- Associar()
- Criar()
- Abrir()
- Ler()
- Gravar()
- Posicionar()

Arquivo Texto

Funções para o tratamento de arquivos texto

- Declaração
- Vincular()
- Criar()
- Abrir()
- Ler()
- Gravar()
- Posicionar()

Algoritmos para manutenção em arquivo texto

- Criação de arquivo texto

```
algoritmo()
{
	arquivo texto ceva;

	vincular ( ceva, "cerveja.txt" );
	criar ( ceva );
}
```

Comentário:

- Inclusão de dados em arquivo texto (utilizando a referência ao identificador lógico do arquivo)

```
algoritmo()
{
	arquivo texto x;
	cadeia nome;
	inteiro n;
	real r;
	cadeia b;

	vincular ( x, "fapp.txt" );
	criar ( x );
	abrir ( x );
	nome := "Amanda e Marina Batista Paim";
	n := 10;
	r := 4.50;
	gravar( x, "fernando paim" );
	gravar( x, nome );
	gravar ( x, n );
	gravar ( x, r );
	fechar ( x );
}
```

- Inclusão de dados em arquivo texto (utilizando a referência ao identificador físico do arquivo)

```
algoritmo()
{
	arquivo texto x;
	cadeia nome;
	inteiro n;
	real r;
	cadeia b;

	vincular ( x, "fapp.txt" );
	criar ( "fapp.txt" );
	abrir ( "fapp.txt" );
	nome := "Amanda e Marina Batista Paim";
	n := 10;
	r := 4.50;
	gravar ( "fapp.txt", "fernando paim" );
	gravar ( "fapp.txt", nome );
	gravar ( x, "fapp.txt", n );
	gravar ( x, "fapp.txt", r );
	fechar ( x, "fapp.txt", x );
}
```

Comentário:

- Inclusão de dados em arquivo texto (todos os tipos de dados)

```
algoritmo()
{
	arquivo texto x;
	cadeia nome;
	inteiro n;
	real r;
	cadeia b;

	vincular ( x, "fapp.txt" );
	criar ( x );
	abrir ("fapp.txt");

	nome := "Amanda e Marina Batista Paim";
	n := 10;
	r := 4.50;
	gravar ( x, "fernando paim", "rosa monica", 10, 10.56 );
	gravar ( x, nome, n, r );
	gravar ( x, nome, "*", n, "*", r );
	gravar ( x, "fernando paim", ";", "rosa monica", ";" , 10, ";" , 10.56 );
	fechar("fapp.txt");
}
```

Comentário:

- Cópia de dados em arquivo texto

```
Algoritmo()
{
	arquivo texto x;
	arquivo texto y;
	cadeia f;
	inteiro i;

	vincular ( x, "fapp.txt" );
	vincular ( y, "fapp2.txt" );
	criar ( y );
	abrir ( x );
	abrir ( y );
	para ( i := 1 ate 5 passo 1 )
	{
		ler ( x, f );
		gravar ( y, f );
	}
	fechar ( x );
	fechar ( y );
}
```

Comentário:

- Posicionamento de dados em arquivo texto

```
algoritmo()
{
	arquivo sequencial x;
	cadeia f;

	vincular ( x,"fapp.txt" );
	abrir ( x );
	posicionar ( x, FINAL );
	ler ( x, f );
	escreva ( f );
	fechar ( x );
}
```

Comentário: