ALGORITMOS

PROF. LOURIVAL
lourival@dimap.ufrn.br

PRIMEIRA PARTE

CAPÍTULO 1 : INTRODUÇÃO

1.1 – Lógica e Algoritmo

A lógica é a “arte de bem pensar”. É a “ciência das formas do pensamento”. O raciocínio é a forma mais complexa do pensamento e a lógica estuda a “correção do raciocínio”. Podemos ainda dizer que a lógica tem em vista a “ordem da razão”, por isso, ela estuda e ensina a colocar “ordem no pensamento”.

O objetivo principal do estudo da lógica de programação é a construção de algoritmos coerentes e válidos.

Um algoritmo pode ser definido como uma sequência de passos que visam atingir um objetivo bem definido.

Na medida em que precisamos especificar uma sequência de passos, precisamos utilizar ordem, ou seja, “pensar com ordem”, portanto precisamos utilizar lógica.

Algoritmos são comuns no nosso cotidiano, como, por exemplo, uma receita de bolo. Nela está descrita uma série de ingredientes necessários e uma sequência de diversos passos que devem ser fielmente cumpridos para que se consiga fazer o alimento desejado, conforme se esperava antes do início das atividades (objetivo bem definido).

Quando elaboramos um algoritmo devemos especificar ações claras e precisas, que a partir de um estado inicial, após um período de tempo finito, produzem um estado final previsível e bem definido. Isto significa que o algoritmo fixa um padrão de comportamento a ser seguido , uma norma de execução a ser trilhada, com vistas a alcançar, como resultado final, a solução de um problema, garantindo que sempre que executado, sob as mesmas condições, produza o mesmo resultado.

Programar é basicamente construir algoritmos.

Um programa computacional consiste na tradução de um algoritmo em uma forma “inteligível” para o computador.

Algoritmo é uma sequência ordenada e finita de operações bem definidas e eficazes que, quando executadas sobre dados convenientes, produz uma solução ou a indicação de que a solução não pôde ser obtida.

1.2 – Problema de lógica

Leia cuidadosamente este problema e preencha a tabela abaixo para mostrar a sua solução.

Para um lanche após o cinema, cinco casais: os Castro, os Lima, os Morais, os Nunes e os Guerra, decidiram comer uma pizza na Cantina Italiana. Cada um dos casais pediu um tipo diferente de pizza (sem ordem específica): de enchovas, de milho verde, de cogumelo, de cebola e de calabresa, e um tamanho diferente de pizza, que variou de 40, 35, 30, 25 e 20 centímetros de diâmetro. Quando vieram as contas, cada casal devia uma quantia diferente de dinheiro por sua pizza, variando em 7,50; 7,00; 6,75; 6,25 e 6,00 reais. Pelas dicas abaixo, você poderá deduzir o tipo e o tamanho de pizza que cada um ordenou, e o preço pago.

1. A pizza que os Morais pediram custou mais de R$6,00.
2. A pizza de 30 cm de diâmetro custou R$0,50 a menos do que a pizza de cogumelo, e a pizza de cogumelo custou menos do que a pizza de calabresa.
3. A pizza dos Castro custou mais do que a pizza de 35 cm de diâmetro.
4. A dos Lima custou R$0,75 a mais do que a pizza de 25 cm de diâmetro.
5. A pizza de enchovas custou R$0.75 a mais do que a pizza de 20 cm de diâmetro.
6. A dos Guerra custou R$0,75 a mais do que a pizza de cebola, que não era a mais barata.

CASAL	TIPO	TAMANHO	PREÇO

1.3 – Exemplo de um Algoritmo

Vamos utilizar a nossa linguagem corrente para descrever o comportamento na resolução de uma determinada atividade, como, por exemplo, a troca de uma lâmpada queimada.

Algoritmo

_ acionar o interruptor;
_ se a lâmpada não acender, então
    _ pegar uma escada;
    _ posicionar a escada embaixo da lâmpada;
    _ buscar uma lâmpada nova;
    _ subir na escada;
    _ retirar a lâmpada queimada;
    _ colocar a lâmpada nova;
    _ acionar o interruptor;
    _ enquanto a lâmpada não acender, faça
        _ retirar a lâmpada queimada;
        _ colocar uma lâmpada nova;
        _ acionar o interruptor;

1.4 – Exercícios Propostos

1. Um homem precisa atravessar um rio com um barco que possui capacidade para carregar, apenas, ele mesmo, e mais uma de suas três cargas, que são: um lobo, um bode e um maço de alfafa. Escreva um algoritmo, ou seja, indique todas as ações necessárias para que o homem consiga atravessar o rio sem perder suas cargas.

2. Três jesuítas e três canibais precisam atravessar um rio; para tal dispõem de um barco com capacidade para duas pessoas. Por medidas de segurança, não se deve permitir que em alguma margem a quantidade de jesuítas seja inferior à de canibais. Elabore um algoritmo indicando as ações que concretizam a travessia com segurança.

3. Elabore um algoritmo que mova três discos de uma haste para outra, utilizando uma terceira como auxiliar. Os discos são de tamanhos diferentes e os menores são dispostos sobre os maiores (Torre de Hanói). Pode-se mover um disco de cada vez para qualquer haste, contanto que nunca seja colocado um disco maior sobre um menor.

4. Elabore um algoritmo para calcular a média parcial de um aluno de Algoritmo após as três primeiras avaliações e dizer a sua situação: aprovado, reprovado ou prova final, e nesse último caso quanto precisa para ser aprovado.

1.5 – Programação Estruturada

Programação estruturada é a técnica de construir e formular algoritmos de uma forma sistemática. Consiste numa metodologia de projeto de programas visando facilitar a escrita, a leitura, a verificação apriori, a manutenção e a modificação de programas. Os algoritmos estruturados utilizam em sua sintaxe um número muito limitado de instruções e estruturas básicas (sequência, seleção e repetição), que correspondem a formas de raciocínio intuitivamente óbvias.

1.6 – Formas de representação de algoritmos

1.6.1 – Descrição Narrativa

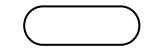
Uso da linguagem natural, como no nosso primeiro exemplo.
Temos a inconveniência da má interpretação, originando ambigüidades e imprecisões.
Vejamos mais um exemplo: a troca de um pneu furado.
Analisar as ambigüidades e imprecisões.

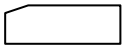
Algoritmo

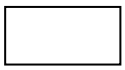
- afrouxar ligeiramente as porcas;
- suspender o carro;
- retirar as porcas e o pneu;
- colocar o pneu reserva e as porcas;
- abaixar o carro;
- dar o aperto final nas porcas.

1.6.2 – Fluxograma

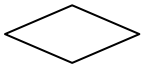
Uso de formas geométricas distintas produzindo ações distintas.
Principais figuras:

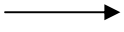

Início ou fim do fluxograma.


Entrada de dados.

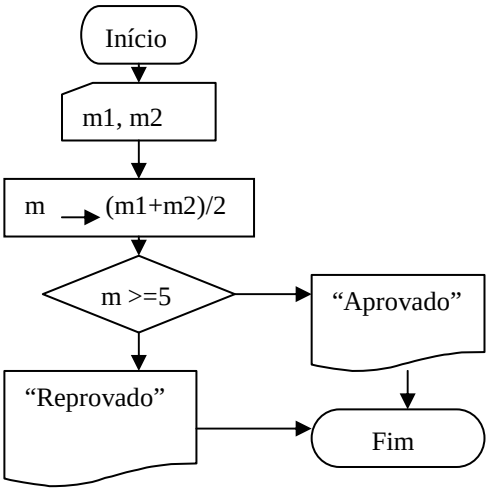

Cálculo de expressões.


Saída de resultados.


Tomada de decisão


Fluxo.

Exemplo: Cálculo de uma média aritmética com um teste.



1.6.3 – Pseudocódigo

Uso de linguagem própria, aproximando-se mais das linguagens de alto nível, chamado, também de pseudolinguagem ou ainda portugol.

Uma linguagem natural (português, inglês, espanhol, etc.) apresenta o inconveniente da ambigüidade de alguns termos, apesar de muitos recursos de comunicação.

Uma linguagem de programação apresenta os inconvenientes das poucas instruções existentes (poucos recursos de comunicação) apesar da grande precisão nos comandos.

Forma geral:

```
Algoritmo       < nome_do_algoritmo >
                < declaração_de_variáveis >
Início
                < Instruções >
Fim
```

Exemplo: Cálculo da média do exemplo anterior.

```
Algoritmo Média_do_aluno
        Real: m1,m2,media
Início
        Escreva("Digite as duas notas:")
        Leia(m1,m2)
        media ← (m1+m2)/2
        Se (média >= 5) então
                Escreva ("APROVADO")
        Senão
                Escreva ("REPROVADO")
        Fim_se
Fim
```

Posteriormente, veremos detalhadamente cada item desse algoritmo.

A partir desse momento daremos toda atenção, somente, a este tipo de representação de algoritmo.

O pseudocódigo não tem os inconvenientes da ambigüidade de uma linguagem natural, nem os rigores de uma linguagem de programação de alto nível. É um português estruturado em “frases” (comandos) correspondentes as estruturas básicas de programação.

CAPÍTULO 2 : TIPOS DE DADOS

Qualquer trabalho realizado por um computador é baseado na manipulação das informações contidas em sua memória. A grosso modo, estas informações podem ser classificadas em dois tipos:
- as instruções: leituras, atribuições, operações, etc.;
- os dados propriamente dito: valores a serem processados.

A memória é um conjunto de células identificadas univocamente por um número chamado endereço.

1 célula = 1 byte = 8 bits
1 bit possui 2 estados: 0 e 1 (dígitos binários).
1 byte possui 2^8 = 256 estados possíveis.

Os dados podem ser numéricos, literais e lógicos, chamados tipos básicos. Os dados numéricos dividem-se em números inteiros e números reais.

2.1 – Dados Numéricos

Os números inteiros e os números reais são armazenados de formas diferentes nos computadores.

Números inteiros: representados sem parte fracionária e sem ponto (chamado também de número de ponto fixo).

Exemplo: 86 0 -19 23456

Números reais: representados com parte fracionária e com ponto (chamado também de número de ponto flutuante).

Exemplo: 85.3 -9.453 10.0 6. 0.00

2.2 – Dados Literais

São sequências de caracteres contendo letras, dígitos e/ou símbolos especiais. São chamados, também, “alfanuméricos”, “cadeia de caracteres” ou “strings”.

Serão representados nos algoritmos entre aspas.

O comprimento de um dado literal é a quantidade de caracteres que ele tem.

Exemplo:

“UFRN” comprimento 4

“ ”	comprimento 1
“”	comprimento 0
“18/04/99”	comprimento 8

2.3 – Dados Lógicos

Usados para representar dois únicos valores lógicos possíveis: verdadeiro e falso.

Representados nos algoritmos como: .V. (verdadeiro) e .F. (falso).

2.4 – Armazenamento de Dados na Memória

2.4.1 – Dados do Tipo inteiro

Ocupa a memória com um número de bytes de acordo com a chamada “palavra do computador”. Num computador de 16 bits ele ocupa 2 bytes, e o inteiro longo ocupa 4 bytes.
Número inteiro: 2 bytes (- 32768 , 32767)
Inteiro longo: 4 bytes (- 2147438648 , 2147438647)

2.4.2 – Dados do Tipo real

Ocupa a memória com o dobro de bytes ocupado pelo tipo inteiro, chamado precisão simples. A precisão dupla ocupa o dobro da precisão simples.
Número real: 4 bytes (1.5E-45 , 3.4E+38)
Precisão dupla: 8 bytes (5.0E-324 , 1.7E+308)

2.4.3 – Dados do Tipo Lógico

Ocupa 1 byte na memória (.V. ou .F.)

2.4.4 – Dados do tipo Literal

Associa a cada caractere um código numérico variando de 0 a 255.
Aloca espaço contíguo de memória igual ao comprimento do literal, destinando um byte para cada caractere da informação.
Dentre as diversas tabelas propostas para a representação de caracteres, a de maior aceitação foi a Tabela ASCII (American Standard Code for Information Interchange).

Exemplo: “casa”

Endereço	Informação
1000	c (99)
1001	a (97)
1002	s (115)
1003	a (97)

CAPÍTULO 3 : VARIÁVEIS E EXPRESSÕES

3.1 – Variáveis

Variável é uma entidade destinada a guardar dados. Ela possui três atributos: nome, tipo e informação.

O nome de uma variável tem a função de diferenciá-la das demais. Adotaremos as seguintes regras para o nome:

- deve necessariamente começar com uma letra;
- não deve conter nenhum símbolo especial, exceto o caractere sublinha.

Exemplos: A2 , max , hora_aula , LADO1 , nome_do_aluno

O tipo de uma variável é o tipo de dado que ela pode armazenar, e a informação é o valor que ela armazena naquele momento.

Todas as variáveis utilizadas em um algoritmo devem ser definidas (declaradas) antes de serem utilizadas.

Uma declaração de variáveis é uma instrução para reservar uma quantidade de memória apropriada para armazenar o tipo especificado e indicar que o seu conteúdo será refenciado pelo nome dado.

Utilizaremos a seguinte sintaxe para declaração das variáveis nos nossos algoritmos:

<tipo> : <lista_de_variáveis>

Numa mesma linha poderão ser definidas uma ou mais variáveis do mesmo tipo. Deve-se separá-las por vírgulas. Variáveis de tipos diferentes em linhas diferentes.

Exemplo:
Inteiro : ano , mes , idade
Real : salario , troco
Lógico : opção , flag
Literal[30] : nome , profissão

Nesta última declaração, o valor 30, entre colchetes, indica o número máximo de caracteres que cada variável (nome ou profissão) pode armazenar.

3.2 – Expressões

Expressão é uma combinação de variáveis, constantes e operadores que, uma vez avaliada, resulta num valor.

Operadores são elementos funcionais que atuam sobre operandos e produzem um determinado resultado.

Os operadores se classificam quanto ao número de operandos em:

- binários (dois operandos);
- unários (um operando).

E quanto ao tipo de dados dos operandos em:

- **aritméticos;**
- **lógicos;**
- **literais.**

Temos ainda os operadores **relacionais**, que é um caso especial, pois permitem comparar pares de operandos de tipos de dados iguais (apenas numéricos ou literais), resultando sempre um valor do tipo lógico.

3.2.1 – Expressões Aritméticas

O resultado da avaliação é do tipo numérico (inteiro ou real).
Operadores aritméticos:

Operador	Tipo	Operação	Prioridade
+	binário	adição	4
-	“	subtração	4
*	“	multiplicação	3
/	“	divisão	3
**	“	exponenciação	2
+	unário	manutenção de sinal	1
-	“	inversão de sinal	1

Se todos os operandos da expressão são inteiros, o resultado é inteiro.
Se todos os operandos da expressão são reais, o resultado é real.
Se os operandos são mistos (inteiros e reais), o resultado da expressão é real.

3.2.2 – Expressões Lógicas

O resultado da avaliação é do tipo lógico (.V. ou .F.).

Operadores lógicos:

Operador	Tipo	Operação	Prioridade
.OU.	binário	disjunção	3
.E.	binário	conjunção	2
.NÃO.	unário	negação	1

Opradores relacionais:

Operador	Comparação
=	igual
<>	diferente
<	menor
<=	menor ou igual

>	maior
>=	maior ou igual

Os operadores relacionais são usados quando se deseja efetuar comparações entre expressões de mesmo tipo (apenas expressões numéricas ou literais, não expresões lógicas), e o resultado é sempre um valor lógico.

3.2.3 – Expressões Literais

Não há padronização para seus operadores. Vamos considerar apenas o operador de concatenação (+) .

Exemplo: "sonha" + "dor" resulta "sonhador"

3.3 – Avaliação de Expressões

Regras:

a) Observar a prioridade dos operadores (os de maior prioridade devem ser avaliados primeiros) . Se houver empate, considera-se a expressão da esquerda para a direita.
b) Os parênteses alteram a prioridade, forçando a avaliação da subexpressão em seu interior.
c) Entre os quatro grupos de operadores existentes a ordem de avaliação é:
   1) aritméticos
   2) literais
   3) relacionais
   4) lógicos.

3.4 – Exemplos

Escrever as expressões matemáticas, utilizando os operadores aritméticos dados (como devem ser escritos nos algoritmos).

1) $5x^3 + 7x^2 - 3x - 1$

   Resp.: 5.0*x**3 + 7.0*x**2 – 3.0*x –1.0

2) $x_0 + v_0t - \frac{1}{2} gt^2$

   Resp.: xo + vo*t – 0.5*g*t**2

3) $\sqrt{p(p-a)(p-b)(p-c)}$

Resp: (p*(p-a)*(p-b)*(p-c))**(1.0/2.0)

4) $\sqrt[3]{(5x^2 + 4x^3)^2}$

Resp.: (5.0*x**2 + 4.0*x**3)**(2.0/3.0)

5) $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Resp.: ((x2-x1)**2 + (y2-y1)**2)**0.5

6) $\dfrac{-b + \sqrt{b^2 - 4ac}}{2a}$

Resp: (- b + (b * b – 4.* a * c)**0.5)/(2 * a)

7) $\dfrac{4}{3}\pi r^3$

Resp: 4.*pi*r**3/3.

8) $$x - \cfrac{3a + 2b}{\cfrac{a - 1}{1 + \cfrac{a + b}{2x}}}$$

Resp: (3.*a + 2.*b)/(x – (a –1.)/(1. + (a + b)/(2.*x)))

9) $(x^{x+2y} + y^{y+2x})^{x+y}$

Resp: (x**(x+2.*y)+y**(y+2.*x))**(x+y)

CAPÍTULO 4 : INSTRUÇÕES PRIMITIVAS

São os componentes básicos que efetuam tarefas essenciais para a operação dos computadores, como entrada e saída de dados e a movimentação dos mesmos na memória.

4.1 – Dispositivo de Entrada

É o meio pelo qual as informações são transferidas pelo usuário ou pelos níveis secundários de memória ao computador. Exemplos: teclado, fitas, discos magnéticos, mouse, scanner.

4.2 – Dispositivo de Saída

É o meio pelo qual as informações (geralmente os resultados da execução de um programa) são transferidas pelo computador ao usuário ou aos níveis secundários de memória. Exemplos: vídeo, impressora, fitas, discos magnéticos.

4.3 – Sintaxe

É a forma como os comandos devem ser escritos, a fim de que possam ser entendidos pelo tradutor de programas.

4.4 – Semântica

É o significado, ou seja, o conjunto de ações que serão exercidas pelo computador durante a execução do referido comando.

4.5 – Instrução de Atribuição

É a principal maneira de se armazenar uma informação numa variável.

Sintaxe : < nome_da_variável > ← < expressão >

Semântica :

1) avaliação da expressão

2) armazenamento do valor resultante na posição de memória correspondente à variável que aparece à esquerda do comando.

Importante :

Deve haver compatibilidade entre o tipo de dado resultante da avaliação da expressão e o tipo de dado da variável (a não ser, propositadamente, com tipos numéricos).

Exemplos:

```
area ← base * altura
delta ← b ** 2 - 4.0 * a * c
contador ← contador + 1
nome ← “Caetano Veloso”
music ← “Dias de Luta” + “ e Flores em Você”
w ← .V.
p ← x > y .E. y > z
q ← (a > = 0 .E. a <= 10) .OU. (a >= 100 .E. a <= 1000)
```

4.6 – Instrução de Entrada de Dados

Sintaxe : Leia (< lista_de_variáveis >)

Semântica : Os dados são fornecidos ao computador por meio de um dispositivo de entrada e armazenados nas posições de memória das variáveis cujos nomes aparecem na lista.

Exemplo:

```
Leia ( x )
Leia ( a , b , c )
```

4.7 – Instrução de Saída de Dados

Sintaxe : Escreva (< lista_de_expressões >)

Semântica : Os argumentos são enviados para o dispositivo de saída. No caso de uma lista de variáveis, o conjunto de cada uma delas é pesquisado na posição de memória correspondente a variável. No caso de argumento constante(número, literal ou lógico) este é enviado diretamente ao referido dispositivo. E no caso de expressões, após sua avaliação, segue como uma constante.

Exemplos:

```
Escreva ("Programa elaborado pelo aluno Thiago.")
Escreva ("Digite um número inteiro positivo:")
Escreva ("Lados do triângulo: ", L1 , L2 , L3 )
Escreva ("Area do circulo = ", pi*r**2)
Escreva ("Area = ", x*y, "Perimetro = ", 2*(x + y))
```

4.8 – Regras Básicas (Interface com o Usuário: fase de execução)

1) Toda vez que um programa estiver esperando que o usuário forneça a ele um determinado dado (operação de leitura), ele deve antes enviar uma mensagem dizendo o que o usuário deve digitar, por meio de um instrução de saída.

2) Antes de enviar qualquer resultado ao usuário, um programa deve escrever uma mensagem explicando o significado do mesmo.

4.9 – Exemplos de algoritmos

1) Dado o preço unitário e a quantidade de um produto, imprimir o valor total da compra.

```
Algoritmo Total
        Real: preco_unit , preco_total
        Inteiro: quantid
Início
        Escreva( "Programa que calcula o preco de uma certa quantidade de um produto. ")
```

```
        Esvreva( “Digite o preco unitário e a quantidade: “)
        Leia( preco_unit , quantid )
        preco_total ← preco_unit * quantid
        Escreva( “Preco total =  “ , preco_total )
Fim
```

2) Calcular a área e o perímetro de um retângulo, sendo dados as medidas dos lados.

```
Algoritmo Retângulo
        Real: L1 , L2 , area , perimetro
Início
        Escreva( “Digite as medidas dos lados do retangulo: “)
        Leia( L1 , L2 )
        area ← L1 * L2
        perimetro ← 2 * (L1 + L2)
        Escreva( “O valor da area é : “ , area )
        Escreva( “O valor do perimetro é : “ , perimetro )
Fim
```

3) Calcular o valor da função f(x) = (3x - 1) / 5 nos extremos do intervalo [a , b] (dados os valores de a e b), e em mais dois valores do seu interior, de modo que os quatros valores do intervalo estejam igualmente espaçados.

```
Algoritmo Valor_de_uma_função
        Real: a , b, h , y1 , y2 , y3 , y4
Início
        Escreva( “Entre com dois números distintos na ordem crescente: “)
        Leia( a , b )
        h ← ( b - a ) / 3.
        y1 ← ( 3. * a - 1. ) / 5.
        y2 ← ( 3. * ( a + h ) - 1. ) / 5.
        y3 ← ( 3. * ( b - h ) - 1. ) / 5.
        y4 ← ( 3. * b - 1. ) / 5.
        Escreva( “x = “ , a, “          y = “ , y1)
        Escreva(“ x = “ , a + h, “      y = “ , y2)
        Escreva(“ x = “ , b - h, “      y = “ , y3)
        Escreva(“ x = “ , b, “          y = “ , y4)
Fim
```

4) Algoritmo para dizer a soma de cinco números com , no máximo, 4 dígitos, antes mesmo das cinco parcelas serem digitadas.

```
Algoritmo Advinha
```

```
        Inteiro:  x
Início
        Escreva ( “Digite um  número com  4 algarismos: “)
        Leia ( x )
        Escreva ( “ O resultado da nossa conta será: “, 19998+x )
        Escreva ( “ Digite o segundo número  com  4 dígitos: “)
        Leia ( x )
        Escreva ( “ O meu  número <terceiro>é : “,  9999 - x )
        Escreva ( “ Digite o quarto número com 4 dígitos: “ )
        Leia ( x )
        Escreva ( “ Finalmente o quinto número é : “,  9999 - x )
Fim
```

4.10 – Rastreamento de um algoritmo

O rastreamento de um algoritmo consiste na execução manual, com dados representativos para registrar os valores tomados pelas variáveis em cada passo do algoritmo.

Para facilitar o acompanhamento, colocamos todos os dados numa tabela de variáveis. Devemos fazer tantos testes quantos forem necessários para nos convencermos de que o algoritmo está perfeito.

Exemplo de um rastreamento do algoritmo do exemplo 4) acima:

x	Saída
3452	Digite um número com 4 algarismos: 3452
1538	O resultado da nossa conta será: 23450
5172	Digite o segundo número com 4 dígitos: 1538
	O meu número <terceiro> é: 8461
	Digite o quarto número com 4 dígitos: 5172
	Finalmente o quinto número é: 4827

Exemplo de um rastreamento do algoritmo do exemplo 3) acima:

a	b	h	y1	y2	y3 y4	
4.0	8.5	1.5	2.2	3.1	4.0	4.9

Saída:

```
Entre com dois números distintos na ordem crescente: 4.0   8.5
x = 4.0        y = 2.2
x = 5.5        y = 3.1
x = 7.0        y = 4.0
x = 8.5        y = 4.9
```

4.11 – Exercícios propostos

1. Encontrar o consumo médio de um veículo, conhecidos a distância total e o volume de combustível consumido para percorrer tal distância.

2. Calcular a média parcial de um aluno da UFRN, dadas as suas três primeiras notas.

3. Calcular o valor da função $f(x,y) = 3x^2 + 2y^2 - xy$ em um ponto qualquer do plano cartesiano.

4. Leia uma temperatura em graus centígrados e imprima a equivalente em graus farheneit (F = 9C/5 + 32).

5. Leia uma quantidade de chuva dada em polegadas e imprima a equivalente em milímetros (1 polegada = 25,4 milímetros).

CAPÍTULO 5 : CONTROLE DE FLUXO DE EXECUÇÃO

Controle de fluxo de execução é a sequência em que as instruções são executadas num algoritmo.

5.1 – Comando Composto

É um conjunto de comandos simples como atribuição, entrada, saída ou algumas construções (estruturas) apresentadas a seguir.

5.2 – Estrutura Sequencial

Cada comando é executado somente após o término do comando anterior.

5.3 – Estrutura de Decisão

O fluxo de instrução a ser seguido é escolhido em função do resultado da avaliação de uma ou mais condições.

Classificação quanto ao número de condições:

- uma condição (decisão simples) : estrutura do SE
- várias condições (decisão múltipla) : estrutura do ESCOLHA

5.3.1 – Estrutura de Dcisão do Tipo SE.

Sintaxe:

Se (< condição >) então	ou	Se (< condição >) então
< comando1 >		< comando1 >
senão		Fim_se
< comando2 >		
Fim_se		

Semântica:

A condição é avaliada. Se o resultado for verdadeiro então comando1 é executado e o fluxo do algoritmo prossegue com o primeiro comando após o Fim_se. Se o resultado for falso, então comando2 é executado e, ao término do mesmo, o fluxo de execução prossegue com o primeiro comando após Fim_se. Há casos em que senão comando2 é omitido. Dessa forma, quando a condição é falsa, o fluxo de execução prossegue normalmente para a primeira instrução após o Fim_se, como se o comando Se não existisse.

5.3.1.1 – Exemplos

1) Determinar se uma pessoa é maior ou menor de idade.

```
Algoritmo Maioridade
        Inteiro: idade
Início
        Escreva( “Digite a idade (maior do que zero): “)
        Leia( idade )
        Se (idade > 0) então
                Se ( idade >= 18 ) então
                        Escreva ( “Maior de idade. “)
                senão
                        Escreva( “Menor de idade. “)
                Fim_se
        Senão
                Escreva (“ Idade incorreta. “)
        Fim_se
Fim
```

Rastreamento: Vamos testar o algoritmo para idade igual a 35.

idade	saída:
35	Digite a idade: 35
	Maior de idade.

2) Calcular o quociente e o resto de uma divisão inteira.

```
Algoritmo Divisão
    Inteiro: x,y,quo,res
Início
    Escreva ( “Digite dois números inteiros:”)
    Leia (x,y)
    Se (y <> 0) então
        quo ←   x / y
        res ←   x – quo * y
        Escreva (“Quociente = “, quo, “Resto = “, res )
    senão
        Escreva (“Não existe divisão por zero.”)
    Fim_se
Fim
```

5.3.1.2 – SE’s Aninhados ou Encaixados

Uma alternativa pode envolver outras decisões.

Exemplo: Dados três números, determinar o maior e o menor entre eles.

```
Algoritmo Max_min
    Real: a, b, c, max, min
Início
    Escreva (“Digite tres numeros: “)
    Leia (a, b, c)
    Se ( a < b ) então
        Se ( b < c ) então
            min ← a
            max ← c
        senão
            max ← b
            Se ( a < c ) então
                min ← a
            senão
                min ← c
            Fim_se
        Fim_se
    senão
        Se ( b > c ) então
            min ← c
            max ← a
        senão
            min ← b
            Se ( a > c ) então
```

```
                        max ← a
                    senão
                        max ← c
                    Fim_se
                Fim_se
            Fim_se
            Escreva (“Maior numero = “, max)
            Escreva (“Menor numero = “, min)
Fim
```

5.3.2 – Estrutura de Decisão do Tipo ESCOLHA

Sintaxe:

```
Escolha(<expressão>)
    Caso(<condição1>)faça
        <comando1>
    Caso(<condição2>)faça
        <comando2>
    .
    .
    .
    Caso(<condiçãot>)faça
        <comandot>
    senão
        <comandok>
Fim_escolha
```

A expressão deve assumir um valor e cada uma das condições deve ser com respeito a expressão. Após a execução de qualquer dos comandos, o fluxo de execução passa para o primeiro comando após Fim_escolha.

5.3.2.1 – Exemplo

1) Programa que simula uma calculadora com as quatro operações aritméticas.

```
Algoritmo Calculadora
    Real: num1,num2
    Literal[2]: op
Inicio
    Escreva(“Digite um numero, o operador e outro numero: “)
    Leia(num1,op,num2)
    Escolha(op)
```

```
                Caso(op=”+”)faça
                        Escreva(num1,op,num2,” =”, num1+num2)
                Caso(op=”– “ )faça
                        Escreva(num1,op,num2,” =”, num1 – num2)
                Caso(op=”* “)faça
                        Escreva(num1,op,num2,” =”,num1* num2)
                Caso(op=” / ”)faça
                        Se(num2<>0)então
                            Escreva(num1,op,num2,” =”,num1/ num2)
                        Senão
                            Escreva(“Não existe divisão por zero.”)
                        Fim_se
                Senão
                        Escreva(“Operador desconhecido.”)
        Fim_escolha
Fim
```

5.4 – Estrutura de Repetição

Uma estrutura de repetição tem por objetivo repetir um trecho de programa um certo número de vezes. É também chamada de laço. Os laços podem ser contados ou condicionais. Os laços contados possuem um número de repetições conhecido. É a estrutura de repetição do tipo PARA – FAÇA. Os laços condicionais possuem um número indeterminado de repetições, dependendo de uma condição. Com a condição no início do trecho, temos a estrutura de repetição do tipo ENQUANTO – FAÇA. Com a condição no final do trecho, temos a estrutura de repetição do tipo REPITA – ATÉ.

5.4.1 – Estrutura de Repetição do Tipo PARA – FAÇA

Sintaxe:

```
Para <var> de <ini> até <fim> passo <inc> faça
        <comando>
Fim_para
```

var é necessariamente uma variável inteira (variável de controle do laço).
ini, **fim** e **inc**, são expressões inteiras (constantes, variáveis ou expressões).
Costumamos omitir **passo <inc>** quando **inc** é 1.

var assume inicialmente o valor de **ini** e testa se não ultrapassou o valor de **fim**. Caso afirmativo o trecho a ser repetido (**comando**) é executado e **var** é incrementado do valor **inc**, e novamente é feito o teste. Caso o teste seja negativo (em algum momento será) a repetição chega ao seu final e a sequência de execução prossegue.

5.4.1.1 – Exemplo

1) Cálculo do fatorial de um número inteiro não-negativo.

```
Algoritmo Fatorial
        Inteiro: num,k,fat
Início
        Escreva("Digite um número:")
        Leia(num)
        Se(num >= 0)então
                fat ← 1
                Para k de 2 até num faça
                        fat ← fat*k
                Fim_para
                Escreva("Fatorial de", num, " igual a " ,fat)
        Senão
                Escreva("Não existe fatorial de número negativo.")
        Fim_se
Fim
```

5.4.2 – Estrutura de Repetição do Tipo ENQUANTO – FAÇA

Sintaxe

```
Enquanto (<expressão lógica>)faça
        <comando>
Fim_enquanto
```

Se o valor lógico de **expressão lógica** é verdadeiro, então o trecho (**comando**) será executado, e novamente **expressão lógica** é avaliada, e assim por diante. Quando o valor de **expressão lógica** é falso, a execução segue para a instrução após Fim_enquanto. No trecho a ser repetido, a **expressão lógica** deve ser alterada para que não tenhamos um laço infinito.

5.4.2.1 – Exemplo

1) Cálculo do mdc entre dois números inteiros positivos.

```
Algoritmo MDC
        Inteiro: a, b
Início
        Escreva ("Digite dois numeros inteiro positivos: ")
        Leia (a,b)
        Se ( a>0  .e.  b>0 ) então
                Enquanto ( a< > b ) faça
                        Se ( a > b ) então
                                a ← a – b
                        senão
                                b ← b – a
                        Fim_se
                Fim_enquanto
                Escreva (" mdc = ", a)
        Senão
```

```
            Escreva (“Dados incorretos.”)
        Fim_se
Fim
```

5.4.3 – Estrutura de Repetição do Tipo REPITA – ATÉ

Sintaxe:

```
Repita

    <comando>

Até (<expressão lógica>)
```

Como o critério de parada fica no final da estrutura, o trecho a ser repetido é executado, sempre, pelo menos, uma vez. O laço chega ao seu final, quando o valor de **expressão lógica** é verdadeiro, o contrário do comando **Enquanto-faça**.

5.4.3.1 – Exemplo

1) Imprimir os divisores de um número inteiro positivo dado.

```
Algoritmo Divisores
    Inteiro: num, div
Início
    Repita
        Escreva (“Digite um numero inteiro positivo: “)
        Leia (num)
    Até (num > 0)
    Escreva (“Divisores do número “, num)
    div ← 1
    Repita
        Se ( num / div * div = num) então
            Escreva (div)
        Fim_se
        div ← div + 1
    Até (div > num)
```

Fim

5.5 – Exercícios Propostos

Faça algoritmos para resolver os problemas abaixo.

1. Calcular a soma dos números pares entre 15 e 55.

2. Calcular a soma dos números ímpares compreendidos entre dois outros números inteiros dados (não incluí-los na soma).

3. Dado um número inteiro positivo maior que 1, dizer se ele é primo ou não.

4. Imprimir o maior, o menor e a média aritmética de n números quaisquer dados.

5. Imprimir os n primeiros números da Sequência de Fibonacci (1 1 2 3 5 8 13 21 …).

6. Calcular a soma de todos os múltiplos de um certo número inteiro dado compreendido entre dois outros números inteiros também dados (não incluí-los na soma).

CAPÍTULO 6 : EXEMPLOS DE ALGORITMOS

1. A multa por excesso de velocidade é baseada em quanto você se excedeu além do limite máximo permitido. Supõe-se que a multa seja computada da seguinte forma:

velocidade acima do limite(km/h) ..... multa

1	a	10	.....	R$ 10,00
11	a	20	.....	R$ 20,00
21	a	30	.....	R$ 30,00
31	a	40	.....	R$ 40,00
41	ou	mais	.....	R$ 50,00

Dados o limite de velocidade e a velocidade com que você vinha, qual o

valor de sua multa?

```
Algoritmo Velocidade
    Inteiro:  lim_vel, vel_mot,  dif_vel, aux
    Real:  multa
Inicio
```

```
    Escreva ( “Digite o limite máximo permitido: “)
    Leia ( lim_vel )
    Escreva ( “Digite a velocidade com que vinha o motorista: “)
    Leia ( vel_mot )
    Se ( vel_mot  > lim_vel ) então
        dif_vel ← vel_mot  -  lim_vel
        aux  ←  ( dif_vel  - 1 ) / 10
        Escolha ( aux )
            Caso ( aux = 0 ) faça
                multa  ←  10.00
            Caso ( aux = 1) faça
                multa  ←  20.00
                    Caso ( aux = 2 ) faça
                multa  ←  30.00
            Caso ( aux = 3 ) faça
                multa  ←  40.00
            Senão
                multa  ←  50.00
        Fim_escolha
        Escreva (“Valor da multa = “, multa )
    Senão
        Escreva (“Não existe multa.”)
    Fim_se
Fim
```

2. Cálculo do máximo divisor comum entre dois números dados.

```
Algoritmo MDC
        Inteiro: a,b,r
Início
        Escreva (“Digite dois números inteiros: “)
        Leia ( a, b )
        Se ( a < 0 ) então
                a ← - a
        Fim_se
        Se ( b < 0 ) então
                 b ← - b
        Fim_se
        Enquanto ( b <> 0 ) faça
                r ← a - a/b*b
                a ← b
                b ← r
        Fim_enquanto
        Escreva ( “MDC = “, a )
Fim
```

3. Imprime os divisores de um número inteiro dado.

```
Algoritmo Divisores
        Inteiro: num,div
Início
        Repita
                Escreva (“Digite um  número inteiro  positivo”)
                Leia ( num )
        Até ( num > 0 )
        Escreva ( “Divisores do número  “, num )
        Para div de 1 até num faça
                Se ( num/div*div = num ) então
                        Escreva ( div )
                Fim_se
        Fim_para
Fim
```

4. Imprime o maior, o menor e a média aritmética de n números dados.

```
Algoritmo Maior_menor
        Real: x,  maior, menor, média
        Inteiro : n , i
Início
        Escreva  (“Digite a quantidade de números “)
        Leia ( n )
        Escreva ( “Digite o primeiro número : “)
        Leia ( x )
        menor ← x
        maior  ← x
        media ← x
        Para i de 2 até n faça
                Escreva ( “Digite mais um número “)
                Leia ( x )
                Se ( x > maior ) então
                        maior  ← x
                Senão
                        Se ( x < menor ) então
                                menor ← x
                        Fim_se
                Fim_se
```

```
            media ← media + x
        Fim_para
        media ← media / n
        Escreva ( “Maior  número = “, maior )
        Escreva ( “Menor número = “, menor )
        Escreva ( “Media aritmetica = “, media )
Fim
```

5.Cálculo do fatorial de um número inteiro não-negativo.

```
Algoritmo Fatorial
        Inteiro: numero, fatorial, aux
Início
        Escreva ( “Digite um número inteiro não-negativo “)
        Leia ( numero )
        Se ( numero >= 0 ) então
                fatorial  ←  1
                aux   ← 2
                Enquanto ( aux <= numero) faça
                        fatorial  ←  fatorial * aux
                        aux  ←  aux + 1
                Fim_enquanto
                Escreva ( “Fatorial de “, numero, “ = “, fatorial  )
        Senão
                Escreva ( “Não existe fatorial de número negativo)
        Fim_se
Fim
```