Algoritmos e Lógica de Programação

Professor Edson de Oliveira

2014 - 1

Sumário:

INTRODUÇÃO A LÓGICA DE PROGRAMAÇÃO

Noções de Lógica

Antes de aprendermos a desenvolver algoritmos, ou programas, precisamos entender um conceito muito utilizado em nosso cotidiano: Lógica. Lógica é algo que não se discute, por exemplo: “Pela manhã, o sol vai nascer e a tarde se por...”. Esta afirmação é óbvia, lógica e indiscutível. Não importa a crença, inteligência ou gosto, todos concordam com esta afirmação.

Aristóteles, filósofo grego, foi o criador da ciência da lógica, onde ele estabeleceu critérios para resolver os problemas com respostas binárias (opostas); a partir de premissas chegava-se as conclusões.

Lógica de Programação

A arquitetura magnética do computador, onde se processam as informações, é lógica; ou seja, uma combinação de passar ou não corrente elétrica caracteriza a informação. Teoricamente representado pelos algarismos 0 e 1.

Partindo deste princípio a concepção de um algoritmo deve ser tratada da mesma forma, logicamente.

A lógica de programação é extremamente necessária para quem quer, de fato, aprender a programar em qualquer linguagem de programação. Ela nos ajuda a pensar pela razão e não pela emoção, como em muitas situações fazemos.

Vamos dar um exemplo:

Um casal de namorados se ama, a garota pergunta ao garoto: “Você não me ama?”, ele, pela emoção, responderia que sim; mas pela razão, se responder sim, ele está confirmando a negação questionada por ela, de “não amar”.

Esta situação pode ser bem resolvida em nosso cotidiano, porém, para o computador quer dizer que “o garoto não ama a garota”, lembrem-se, o computador raciocina logicamente.

Devemos então entender e nos acostumarmos com a forma de “pensar” do computador, para não cometermos equívocos que possam ser fatais para o bom funcionamento do algoritmo.

Algoritmo

Algoritmo é um termo usado em diversas disciplinas exatas, como: Matemática, cálculo, física, química, entre outras. Nestas disciplinas Algoritmo é a forma como um problema é resolvido.

Na área de Lógica de programação o Algoritmo tem o mesmo significado. Muitos confundem e pensam que algoritmos é apenas a solução final do problema, mas não. Algoritmos é a especificação dos passos em ordem lógica que visam resolver o problema.

Geralmente, é criada uma linguagem hipotética – normalmente em português – para a padronização do algoritmo. Os critérios empregados são similares aos das linguagens de programação reais.

Didaticamente para o aluno, o fato da linguagem ser em português (pseudocódigo) facilita o aprendizado do algoritmo, até que ele tenha a disciplina necessária para encarar uma linguagem de programação real.

Programa

Todos os conceitos empregados ao algoritmo servem para o programa. O programa (código fonte) nada mais é do que um algoritmo desenvolvido em uma linguagem de programação real, ou seja, é o “objeto” que faz o Algoritmo realmente funcionar no computador.

TIPOS DE INFORMAÇÕES

Como vimos no capítulo anterior, para fazermos o programador interagir com o computador precisamos definir critérios. O computador é uma máquina de "processar informações (dados)", logo, estes dados deverão seguir algumas propriedades que caracterize e o diferencie dos demais dados.

Por estarmos tratando genericamente os padrões, os citados nesta literatura, não obedece totalmente o critério das linguagens de programação reais.

Temos que saber diferenciar o Nome da Informação do Conteúdo da informação.

Nome da Informação é o Substantivo que caracteriza a informação.

Conteúdo da informação é o que está contido neste substantivo. Esta informação pode ser alterada.

Exemplo:

No cabeçalho de uma prova há diversas informações. Uma delas é a Nota. Para um aluno esta nota pode ser 7, para outro 4.5 ou 10... Portanto, **Nome da Informação** é **NOTA** e **Conteúdo da Informação 7**, **4.5** ou **10**.

Nos próximos subtítulos falaremos de Conteúdo da Informação. Quando falarmos em **Variáveis de Memória** retomaremos o **Nome da Informação**.

Caracteres

São dados alfanuméricos[1] que servem para armazenar informações que contenham qualquer tipo de símbolo do teclado.

Exemplos: "Rua Ataliba Vieira, 1025", "Edson de Oliveira", "São Paulo".

Repare que no exemplo acima os bytes (caracteres) que compõem o endereço são alfanuméricos, algarismos e símbolos especiais. Todo Conteúdo da Informação caractere é delimitado por aspas ("").

Os dados do tipo caracteres ocupam na memória RAM o número de bytes nele contido, por exemplo: "computador" - 10 bytes, "casa" - 4 bytes e "Bom dia!" – 8 bytes.

Numéricos

São informações que só podem ser representadas por números, algarismos de 0 (zero) a 9 (nove).

Em alguns casos o armazenamento de conteúdos numéricos também serve ao tipo de dado Caractere.

Para melhorar a classificação dos dados sugerimos que definam como tipo Numérico somente as informações que possam gerar cálculos, como o Salário.

Inteiros

Informações numéricas que tem a característica de não serem fracionárias, por exemplo: Quantidade de filhos: 5, Número de alunos: 40 e Total de rodas: 4.

Os dados do tipo Inteiro ocupam na memória RAM 2 Bytes e pode armazenar números até o limite de 65535 positivo ou negativo.

As informações do parágrafo anterior podem variar dependendo da linguagem de programação utilizada.

Reais

Informações numéricas que tem a característica de ser fracionárias, por exemplo: Salário: 1254.25, Inflação: 0.75842 e Nota: 7.5.

[1] Algarismos, letras e caracteres especiais (vírgula, ponto, hífem, parênteses entre outros)

Os dados do tipo Real ocupam na memória RAM 4 Bytes e podem armazenar números com 9 dígitos significativos.

As informações do parágrafo anterior podem variar dependendo da linguagem de programação utilizada.

Vale lembrar que as linguagens de programação tratam números reais diferente do padrão brasileiro. Eles usam o ponto para separar a parte fracionária do número inteiro, enquanto usamos a vírgula.

Lógicos

Informações lógicas, como diz o próprio nome, tem a característica binária[2]. Armazenam valores booleanos (verdade/falso). Por exemplo: Maior de idade? Verdade; casado? falso e Tem dependentes? verdade.

Fica mais fácil entender estes tipos de dados em forma de perguntas lógicas, como nos exemplos acima.

Os dados do tipo Lógico ocupam na memória RAM 1 Bytes e só armazenam os valores .V. (verdade) e .F. (falso).

[2] Somente duas respostas uma oposta da outra. Na teoria .V. simboliza verdade e .F. falso.

REPRESENTAÇÕES DO ALGORITMO

Existem diversas formas para representarmos um algoritmo.

Apresentaremos neste capítulo algumas destas formas.

Vale lembrar, que no decorrer do curso nos apoiaremos teoricamente na Descrição Narrativa, Fluxograma e Pseudocódigo; e na prática em Linguagem de programação C.

Descrição Narrativa

Descrição narrativa é a forma de representação do algoritmo através de um texto que expresse os processos a serem executados para resolução do problema compassadamente.

Nesta forma de representação do algoritmo não é necessário técnicas, regras ou critérios; basta expressar os passos do algoritmo. Cada passo é uma instrução no Algoritmo.

Fluxograma (Diagrama de Blocos)

Fluxograma é a forma de representação dos passos do algoritmo através de figuras que representam as ações deste passo.

Nesta forma de representação do algoritmo usamos regras para cada uma das figuras/instruções. Começamos a despertar no aprendiz a necessidade e importância da disciplina na concepção de um algoritmo.

Pseudocódigo (Português Estruturado)

O pseudocódigo é uma linguagem hipotética que usamos para representar as instruções do algoritmo através de comandos pré-definidos.

Estes comandos e regras são padrões para todos os algoritmos.

A construção do algoritmo através do pseudocódigo é fundamental no aprendizado das regras de sintaxe, porque a obediência às regras será cobrada totalmente na utilização de uma linguagem de programação.

Linguagem de programação

Nos meios teóricos (Descrição Narrativa, Fluxograma e Pseudocódigo) a consequência de qualquer tipo de erro não é mostrado ao aprendiz na confecção do Algoritmo na prática.

A partir do momento que desenvolvemos o Algoritmo em uma linguagem de programação o compilador (software que permite o desenvolvimento de programas) despertará o erro por menor que ele seja.

Depois de concluído o Algoritmo em uma linguagem de programação o “programa” desenvolvido poderá ser executado pelo usuário.

ARMAZENAMENTO DE INFORMAÇÕES

Variáveis de memória

Variável é o meio magnético que o computador utiliza para armazenar – na memória RAM – informações dadas pelo programador ou operador do algoritmo. Uma variável, analogamente, é como um papel que armazena uma informação escrita e que pode ser recuperada.

Por ser um meio magnético, na falta de energia, por quaisquer motivos, acarretará na perda da informação nela armazenada.

Como diz o próprio nome, uma "variável" pode ter o seu Conteúdo da Informação mudado no decorrer do algoritmo.

Nomeação

Toda variável de memória devem ter um nome que a identifique (lembram-se do **Nome da Informação**?), diferenciando-a das demais contidas no mesmo algoritmo. Cada linguagem de programação temas suas regras para nomear variáveis. Algumas regras podem coincidir.

As regras mais comuns são:

- Começar com letra;
- Não conter caracteres especiais[3], exceto o sublinhado (_) e
- Não ser palavra reservada [4]da linguagem.

Atribuição

Atribuição nada mais é do que preencher o conteúdo de uma variável com valores definidos pelo usuário do algoritmo, pelo programador pelo programa.

Constantes de Memória

Tudo o que dissemos sobre Variáveis de Memória serve para constantes de memória. Exceto o último parágrafo, ou seja, o Conteúdo da Informação de uma Constante de Memória não pode ser mudado (Exceto pelo programador no desenvolvimento do algoritmo).

Uma vez que o Algoritmo esteja em modo de execução o valor atribuído à constante não se altera.

[3] Todos os caracteres que não seja numero ou letra
[4] Palavra que a linguagem de programação utilize para algum fim específico

PROCESSANDO OS DADOS

Operadores

Lógicos

Operadores lógicos, como diz o próprio nome, serve para avaliar expressões lógicas.

Foi tirada da Língua Portuguesa três palavras chaves – E, OU, e NÃO –. No Algoritmo estas palavras são chamadas de operadores Lógicos.

Estas palavras faz com que duas entidades se comuniquem. Assim, foram implementadas no computador, para uma comunicação entre o homem e a máquina.

Cada uma destas palavras tem o seu significado e regra.

Veja a tabela abaixo:

Operando 1	Operando 2	.E.	.OU.	.NÃO. operando 1	.NÃO. operando 2
.V.	.V.	.V.	.V.	.F.	.F.
.V.	.F.	.F.	.V.	.F.	.V.
.F.	.V.	.F.	.V.	.V.	.F.
.F.	.F.	.F.	.F.	.V.	.V.

OPERADOR LÓGICO .NÃO. ⇒ O operador lógico .NÃO. inverte o valor lógico do operando.

OPERADOR LÓGICO .E. ⇒ O operador lógico .E. só resulta verdade se todos os operandos forem verdadeiros, nos demais casos resulta falso.

OPERADOR LÓGICO .OU. ⇒ O operador lógico .OU. só resulta falso se todos os operandos forem falsos, nos demais casos resulta verdade.

Não importa o número de operandos envolvidos numa expressão, as regras são as mesmas.

A prioridade de execução dos operadores lógicos é .NÃO., .E. e .OU..

Em algumas linguagens de programação, existem outros operadores lógicos, como *ou exclusivo*; porém os estes foram derivados a partir dos descritos acima.

Em linguagem C:

Operando/Operador	Em C
.V.	1
.F.	0
.NÃO.	!
.E.	&&
.OU.	||

Aritméticos

Os operadores aritméticos servem para efetuar cálculos matemáticos em um algoritmo com dados Inteiros e Reais. Alguns símbolos são iguais aos da matemática, outros não. A ordem de prioridade de execução obedece a matemática.

Veja a tabela:

Operador	Em C	Descrição	Prioridade	Para dado
+	+	Adição	4	Inteiro e real

-	-	Subtração	4	Inteiro e real
*	*	Multiplicação	3	Inteiro e real
/	/	Divisão	3	Inteiro e real
DIV	/	Divisão inteira	3	Inteiro
MOD	%	Módulo	3	Inteiro
^	Não existe	Exponenciação	2	Inteiro e real
( )	( )	Parênteses	1	Inteiro e real

Caracteres

Teoricamente (fluxograma e Pseudocódigo), existe apenas um operador de caracteres: o sinal de adição (+).

Sua função é concatenar (juntar) as cadeias de caracteres (strings).

Em linguagem C este conceito não funciona; há funções apropriadas para este fim.

Relacionais

Este tipo de operador tem como objetivo comparar os operandos.

Veja a tabela:

Operador	Em C	Descrição
>	>	Maior
<	<	Menor
>=	>=	Maior ou igual
<=	<=	Menor ou igual
<>	!=	Diferente
=	==	Igual

Estes operadores podem ser utilizados com qualquer tipo de dados, não importando a classificação ou o tipo.

Expressões

A combinação dos operadores com os operandos[5] formam as expressões que o algoritmo irá calcular, retornando o resultado.

Esta informação geralmente é armazenada em uma variável de memória, podendo também ser exibidas diretamente nos dispositivos de saída do computador (monitor ou impressora).

Aritméticas

Os tipos de expressões matemáticas envolvem dados numéricos (inteiros e reais) e operadores aritméticos.

A precedência dos cálculos na expressão obedece a mesma ordem da matemática. Porém, uma expressão computacional só pode ser escrita em uma linha, diferentemente de uma expressão aritmética.

Exemplo:

Expressão Aritmética:

$$Y = \frac{4 \,.\, 6^3 + 9 - 4}{5}$$

Expressão Computacional Teórica (Fluxograma ou Pseudocódigo):

```
Y ← (4 * 6 ^3 + 9 - 4) / 5
```

[5] Valores e informações (dados) contidas no algoritmo

Expressão Computacional em C:

```
Y = (4 * (6 * 6 * 6) + 9 - 4) / 5;
```

Lógicas

As expressões lógicas processam operando lógicos.

As expressões as lógicas, também tem uma prioridade a ser considerada. Desconsiderando os parênteses (que quebram a prioridade) os operadores lógicos são executados na ordem: .NÃO., .E. e .OU..

Exemplo:

```
.NÃO.((.V..OU..F.).E.(.NÃO..F..E..V.))
.NÃO.( .V.         .E.(.V.      .E..V.))
.NÃO.( .V.         .E. .V.              )
.NÃO.  .V.
⇒ Resulta .F.
```

Caracteres

A única operação que podemos usar em uma expressão caractere, é a concatenação (+). “String” é o termo usado para identificar uma sequência de valores caracteres, logo, concatenação nada mais é do que juntar duas ou mais strings independentes.

Exemplo:

```
"Bom Dia"+"João Carlos"          Resulta ⇒ Bom DiaJoão Carlos
"Bom Dia "+"João Carlos"         Resulta ⇒ Bom Dia João Carlos
" Bom Dia "+" "+" João Carlos"   Resulta ⇒  Bom Dia   João Carlos
```

Mistas

Existem situações, em programação, onde precisamos montar expressões que envolvam dados de tipos diferentes.

A seguir, um exemplo de expressão mista:

Exemplo:

```
(4 + 7 <= 6).OU.(.NÃO..V..E..F.).E.("ABC" <> "BCA")
(11    <= 6).OU.(.F.      .E..F.).E.(.V.           )
      .F.    .OU. .F.             .E. .V.
      .F.    .OU. .F.
Resultado ⇒ .F.
```

COMANDOS SIMPLES

Comandos simples são as instruções que iniciam e terminam na mesma linha. Temos a instrução de **Entrada de Dados**, **Saída de dados** e **Processamento de dados**.

Antes de aprendermos os comandos precisamos saber a estrutura de um algoritmo e a de um programa em C.

Estrutura de um Pseudocódigo

```
programa <nome_programa>
var
    <lista_variáveis> : <tipos>
inicio
    <corpo_do_algoritmo>
fim
```

Estrutura de um Programa em C

```
#include <bibliotecas>
int main()
{
    <corpo do programa>;
}
```

Existem vários compiladores para compilar um programa em C.

Utilizaremos o Dev C++ por ser enxuto e gratuito.

Para que a aprendizagem seja melhor, recomendo que todos utilizem este compilador, todos os exemplos desta literatura serão expressos nele. Outros compiladores têm leves diferenças

Tipos de variáveis

Antes de entrarmos nos primeiros comandos é necessário sabermos como o pseudocódigo e linguagem C "chamam" os tipos de variáveis para podermos manipulá-la no programa.

Mostraremos apenas os tipos de variáveis mais importantes.

No Pseudocódigo	Em C
Inteiro	Int
Real	Float
Caractere	Char
Lógico	int[6]

[6] Em C não há um tipo lógico explicito. O tipo int o representa. Conteúdo 0 (zero) é falso e qualquer outro valor é verdade

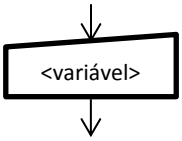
Entrada de dados – Leia / scanf()

Usamos este comando quando houver a necessidade da interação do usuário com o sistema em execução, ou seja, quando o programa precisar de alguma informação do usuário para o seu bom andamento.

Os periféricos usados para este fim são os de entrada de dados. O mais comum _e que usaremos no curso_ é o teclado.

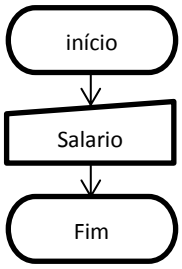
Fluxograma

Sintaxe:



No lugar de <variável> coloque somente o Nome da Variável que será lida. O mesmo vale para o pseudocódigo e C.

Exemplo:



Observação:

Um algoritmo representado no fluxograma simplesmente pede para o usuário digitar um valor na variável salário.

Pseudocódigo

Sintaxe:

```
Leia <variável>
```

Exemplo:

```
Leia Salario
```

Linguagem C

Sintaxe:

```
scanf("<formato da variável>",&<variável>);
```

O <formato da variável> deve ser trocado pela formatação corresponde à variável lida:

int	%d
float	%f
char	%c

Exemplo:

```
scanf("%f",&salario);
```

Observação:

Repare que o **<formato da variável>** deve ser colocado entre aspas e antes da <variável> deve conter o & (e comercial).

Saída de dados – Escreva /print()

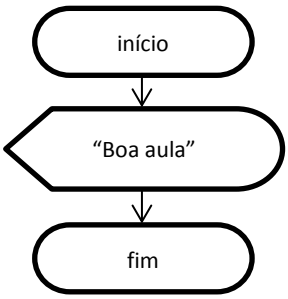
Usamos este comando quando houver a necessidade da interação do sistema com o usuário, ou seja, quando o programa precisa fornecer alguma informação ao usuário.

Os periféricos usados para este fim são os de saida de dados. O mais comum _e que usaremos no curso_ é o monitor.

Fluxograma

Sintaxe:

Exemplo:



Observação:

Neste exemplo aparecerá a mensagem "Boa aula" na tela.

Pseudocódigo

Sintaxe:

```
Escreva <expressão>
```

Exemplos:

```
Escreva salario                      // exibe o conteúdo da variável salário
Escreva "salário"                    // exibe a palavra salário
Escreva "Salário: R$ ", salario      // exibe o texto 'Salário: R$' e o conteúdo da
                                     // variável 'salario'
Escreva "Salário: R$ ", 3000 * 1.1   // Exibe o texto 'salário: R$' com o resultado
```

```
                                        // da expressão, 3300
```

Observação:

O termo <expressão> da sintaxe simboliza um texto, uma variável, uma constante, um calculo ou a combinação de qualquer um destes.

Linguagem C

Sintaxe:

```
printf("expressão"[,<lista de variáveis]);
```

Exemplos:

```
float salario = 3456;
printf("%f", salario);                  // saída: 3456.000000
printf("Salario", salario);             // saída: Salario
printf("Salario: R$ %.2f", salario);    // saída: Salário: R$ 3456.00
printf("Total: %.1f",salario + 100);    // saída: Total: 3556.0
int a = 5, b = 10;
printf("O dobro de %d é %d",a,b)        // saída: O dobro de 5 é 10
```

Observação:

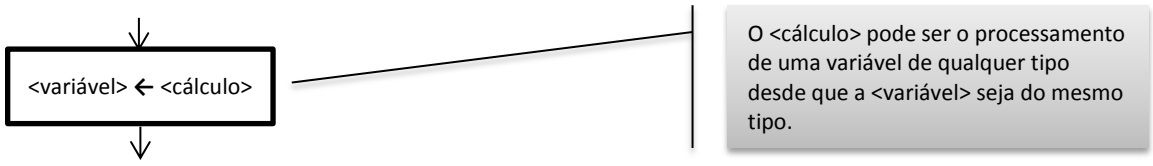
Para exibir uma variável ou um cálculo é necessário formatar a saída da variável. Com dados float '%.2f' formata a saída com duas casas decimais. Se o ".x" for inibido será exibida a informação com seis casas decimais.

Comando Processamento – Atribuição

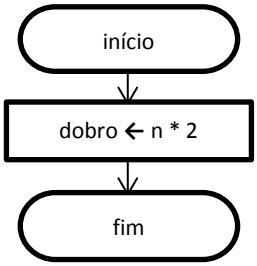
Usamos este comando quando← o computador precisar efetuar algum tipo de como cálculo, atribuição ou manipulação de dados. Neste comando não há a necessidade da interação do usuário.

Fluxograma

Sintaxe:



Exemplo:



Observação:

O cálculo pode envolver dados variáveis ou dados constantes. Por exemplo, no calculo acima (n * 2) envolve a variável 'n' e a constante '2'.
<variável> ⇒ é o local onde será armazenado o valor resultante do <cálculo>.
<cálculo> ⇒ o seu resultado será atribuído na <variável>.

Lembrete – Não existe um nome próprio para o comando processamento no pseudocódigo, como nos outros comandos, bastando assim colocar a expressão e atribuí-la, obrigatoriamente e exclusivamente, em uma variável.

Pseudocódigo

Sintaxe:

```
<variável> ← <cálculo>
```

Exemplo:

```
reajuste ← salario * 1.25
```

Linguagem C

Sintaxe:

```
<variável> = <cálculo>;
```

Exemplo:

```
reajuste = salario * 1.25;
```

Exemplo

A seguir, daremos o exemplo de um algoritmo construído em suas três representações e na linguagem C.

Problema: Dado o salário pelo usuário, fazer um algoritmo que calcule as férias (salário + 1/3), subtraindo a contribuição de INSS de 11%. Exibir as férias brutas, contribuição de INSS e as férias líquidas.

Entrada:[7] *2100.00* ***Saída:***[8] *2799.30 307.92 2491.38*

Descrição Narrativa:

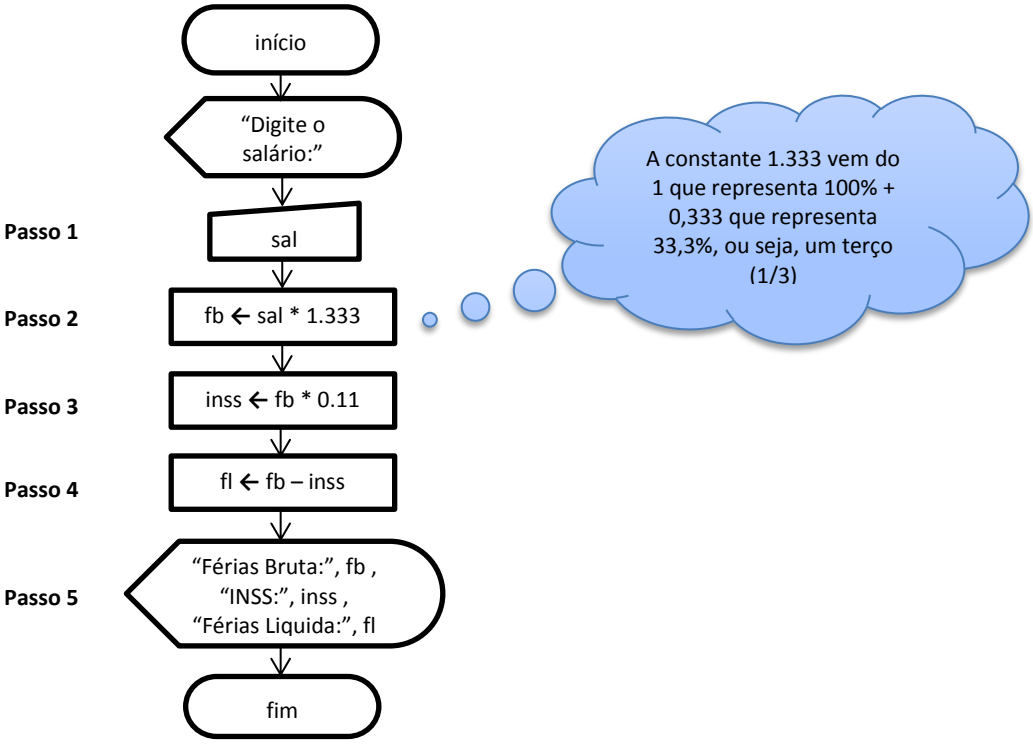
- **Passo 1:** Obter o salário do funcionário (SAL)
- **Passo 2:** Calcular as Férias Bruta (FB)
- **Passo 3:** Calcular o INSS (INSS)

[7] "Entrada" é o termo que usamos no enunciado dos Algoritmos para representar um exemplo de dado(s) inserido pelo usuário.

[8] "Saída" é o termo que usamos no enunciado dos Algoritmos para representar os valor(es) que serão apresentados depois do algoritmo ter processado as informações e finalizado.

- **Passo 4:** Calcular as Férias Líquida. (FL)
- **Passo 5:** Exibir um relatório detalhado das férias.

Fluxograma:



Pseudocódigo:

```
programa ferias_1
var
    sal , fb   : real
    fl , inss : real
inicio
    escreva "Digite o salário:"
    leia sal     // Passo 1
    fb←sal*1.333                    // Passo
    inss←fb*0.11                    // Passo
    fl←fb-inss // Passo 4
    escreva"Férias Bruta:",fb,
           "INSS:",inss,
           "Férias líquida:",fl     // Passo
fim
```

Parte 1: Nome do Algoritmo

Parte 2: Declaração das variáveis

Parte 3: Corpo do Algoritmo

Linguagem C:

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    float sal, fb, inss, fl;
    printf("Digite o salario:");
    scanf("%f",&sal);               // Passo 1
    fb = sal * 1.333;               // Passo 2
    inss = fb*0.11;                 // Passo 3
```

```
    fl = fb - inss;                  // Passo 4
    printf("Ferias Bruta: %.2f\nINSS: %.2f\nFerias liquida: %.2f\n",fb,inss,fl);
                                     // Passo 5
    system("pause");
}
```

SOBRE A LINGUAGEM C

Reservaremos este espaço pra falar da linguagem C.

Nesta material didático mostraremos na Linguagem C o suficiente para executar os algoritmos teóricos desenvolvidos em Fluxograma e Pseudocódigo, ou seja, o foco é desenvolver algoritmos: Lógica de Programação.

História

Em 1947, três cientistas do Laboratório Telefonia Bell, William Shockley, Walter Brattain, e John Bardeen criaram o transistor. A computação moderna teve início. Em 1956 no MIT o primeiro computador completamente baseado em transistores foi concluído, the TX-0. Em 1958 na Texas Instruments, Jack Kilby construiu o primeiro circuito integrado. Mas mesmo antes do primeiro circuito integrado existir, a primeira linguagem de alto nível já tinha sido escrita.

Em 1954 Fortran, a Formula Translator, foi escrito. Começou como Fortran I em 1956. Fortran veio a ser Algol 58, o Algorithmic Language, em 1958. Algol 58 veio a ser Algol 60 em 1960. Algol 60 veio a ser CPL, o Combined Programming Language, em 1963. CPL veio a ser BCPL, Basic CPL, em 1967. BCPL veio a ser B em 1969. B veio a ser C em 1971.

B foi a primeira língua da linhagem C diretamente, tendo sido criado no Bell Labs por Ken Thompson. B era uma linguagem interpretada, utilizada no início, em versões internas do sistema operacional UNIX. Thompson e Dennis Ritchie, também da Bell Labs, melhorou B, chamando-NB; novas prorrogações para NB criaram C, uma linguagem compilada. A maioria dos UNIX foi reescrito em NB e C, o que levou a um sistema operacional mais portátil.

B foi, naturalmente, o nome de BCPL e C foi o seu sucessor lógico.

A portabilidade do UNIX foi a razão principal para a popularidade inicial de ambos, UNIX e C; rather than creating a new operating system for each new machine, system programmers could simply write the few system dependent parts required for the machine, and write a C compiler for the new system; and since most of the system utilities were written in C, it simply made sense to also write new utilities in the language.

Fonte e outras informações: http://pt.wikibooks.org/wiki/Programar_em_C/Hist%C3%B3ria_da_linguagem_C

Versões

Uma Linguagem de Programação (compilador) é um software, sendo assim o software deve ser instalado no computador.

Existem versões em português e inglês. O problema é a versão do sistema operacional que comporta cada versão.

- Até o Windows 7: http://dev-c.softonic.com.br/
- Windows 8: http://dev-c.updatestar.com/pt/technical

Como sou avesso ao Windows 8 e não o uso, não tive como testar o ultimo link.

Peculiaridades da linguagem C

Para que todos consigam rodar um programa em C satisfatoriamente, falaremos sobre algumas peculiaridades da Linguagem C. Para tal utilizaremos o programa que fizemos no exemplo anterior com a exibição melhor ajustada.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      float sal, fb, inss, fl;
7      printf("Digite o salario:");
8      scanf("%f",&sal);    // Passo 1
9      fb = sal * 1.333;    // Passo 2
10     inss = fb*0.11;      // Passo 3
11     fl = fb - inss;      // Passo 4
12     printf("Ferias Bruta: %.2f\n",fb);
13     printf("INSS: %.2f\n",inss);
14     printf("Ferias liquida: %.2f\n",fl);
15                          // Passo 5
16     system("pause");
17 }
```

- Todos os comandos e palavras reservadas escrever em letras minúsculas;
- Coloca-se ";" (ponto e vírgula) em todo final de instrução;
- O compilador ignora espaços, tabulações e quebras de linha no meio do código;
- O padrão da linguagem C é criar variáveis com letras minúsculas. Caso uma ou mais letras seja maiúscula o compilador pode interpretar como outra variável;
- O programa só executa ("roda") se estiver isento de erros de compilação;
- Linhas 1 e 2: Inclusão das bibliotecas[9] utilizadas no programa. Para nós, estas bibliotecas serão suficientes até o final do curso;
- Linhas 4, 5 e 17: É o corpo da linguagem C, todas as instruções devem ser escritas entre as linhas 6 e 16;
- Linha 6: Nesta linha estão declaradas as variáveis. Em C as variáveis podem ser declaradas em qualquer linha do programa;
- Linha 8: leitura da variável 'sal'. No scanf() deve-se colocar o % e o formato da variável a ser lida (sem formatações ou \n) e antes da variável o &;
- Linha 9: Repare que quando vamos utilizar um dado constante em um programa _neste caso o 1.333_ usamos o PONTO para separar as casas decimais e não a vírgula;
- Os comentários em Linguagem C são // para comentar linha e /* */ para comentar múltiplas linhas. No exemplo acima os comentários estão na cor azul e
- Linhas 12, 13 e 14: nestas linhas estamos exibindo os valores com formatação de 2 casas decimais "%.2f". Este tipo de formatação funciona somente com dados do tipo float. "\n" quebra a linha na execução do algoritmo.

[9] Cada comando está contido em uma biblioteca

ESTRUTURAS DE DECISÃO

Estrutura Condicional: Se/If()

O comando "Se" é o primeiro dos comandos estruturados[10].

Ele analisa uma condição e toma a decisão de qual "bloco" será executado.

Temos variações desta estrutura:

Se... Então – If()...

Este comando é conhecido como "Se...Então...".

Ele executa o <bloco1> se a <Condição> resultar .V. (verdade); caso a condição resulte .F. (falso) o <Bloco1> é ignorado e o fluxo do programa segue a sequencia. Este comando pode executar somente um bloco.

Fluxograma

Sintaxe:

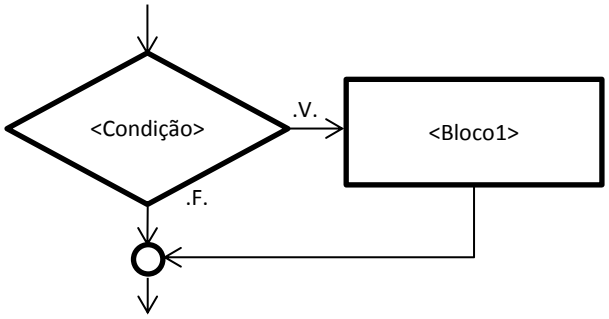


Figura 1 - Estrutura Se... Então...

Pseudocódigo

Sintaxe:

```
Se <condição> então
    <bloco1>
Fim_se
```

Linguagem C

Sintaxe:

```
if (<condição>)
{
    <bloco1>;
}
```

Observação:

Um programa em C é basicamente estruturado em **blocos** de código. Blocos nada mais são que conjuntos de instruções que atingem o mesmo objetivo, e devem ser delimitados com chaves (`{ ... }`). Um bloco também pode conter outros blocos. O Bloco é obrigatório quando houver mais do que uma instrução.

[10] São comandos que tem um início e fim próprio.

Exemplo

A seguir, daremos o exemplo de um algoritmo com a estrutura de decisão "Se... Então...".

Problema: Dado um número pelo usuário, exibir o seu número positivo.

Entrada: *45* ***Saída:*** *45*
Entrada: *-73* ***Saída:*** *73*

Comentários:

Neste problema devemos fazer algo com o número somente se ele for negativo, ou seja, se for negativo o transformamos em positivo; por outro lado se o número já for positivo mantemos o número. Independentemente do caso, devemos exibir o número no final do Algoritmo.

Descrição Narrativa:

- Ler o número (N)
- Se for negativo
  - Transformá-lo em positivo
- Exibir o número

Fluxograma:

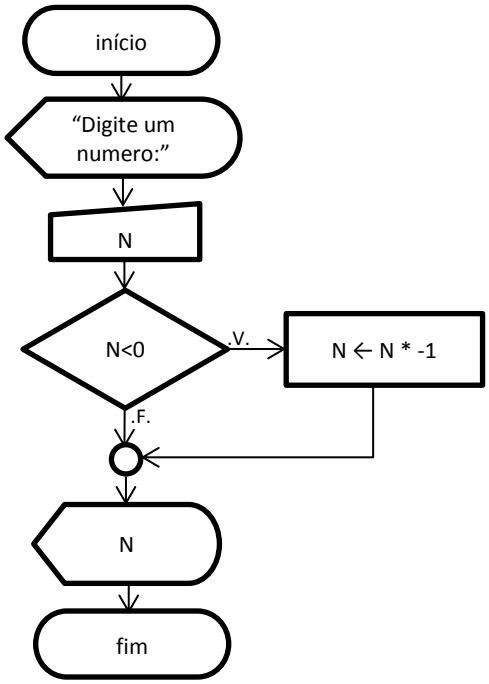


Figura 2 - Exemplo com a estrutura Se... Então...

Pseudocódigo

```
Programa ex_se_entao
Var
    N : inteiro
Inicio
    Escreva "Digite um Numero"
    Leia N
    Se N < 0 então
        N ← N * -1
    Fim_se
```

```
    Escreva N
Fim
```

Linguagem C

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int n;
    printf("Digite um numero:");
    scanf("%d",&n);
    if (n < 0)
    {
        n = n * -1;
    }
    printf("N = %d\n", n);
    system("pause");
}
```

Se... Então... Senão... / if() ... else

Este comando é conhecido como “Se...Então... Senão”.

Ele executa o <Bloco1> se a <Condição> resultar .V. (verdade) ou executa o <Bloco2> se a condição resultar .F. (falso). Dentre os comandos Se´s ele é o mais comum.

Independentemente do bloco executado, o algoritmo segue o fluxo na sequência.

Este comando executa um entre dois blocos possíveis, dependendo da condição resultante.

Fluxograma

Sintaxe:

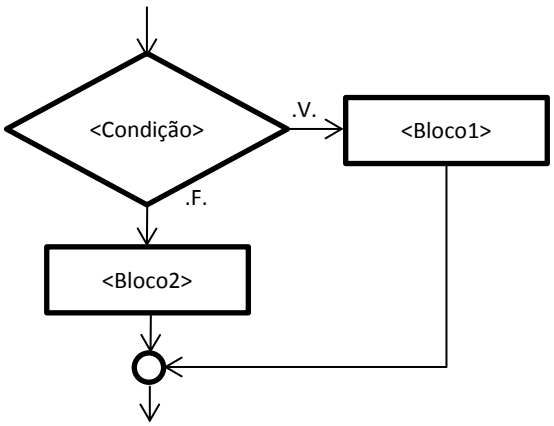


Figura 3 - Estrutura Se... Então... Senão...

Pseudocódigo

Sintaxe:

```
Se <condição> então
    <bloco1>
Senão
    <bloco2>
```

```
Fim_se
```

Linguagem C

Sintaxe:

```
if (<condição>)
{
    <bloco1>;
}
else
{
    <bloco2>;
}
```

Comentário:

Lembrando que as chaves só são obrigatórias se no <bloco> houver mais do que um comando.

J A M A I S coloque ";" (ponto e vírgula) no final da linha do if() ou na linha do else.

Exemplo

A seguir, daremos o exemplo de um algoritmo com a estrutura de decisão "Se... Então...Senão".

Problema: Dado um salário e o tempo de casa (em anos), calcular o aumento salarial de um funcionário de acordo com as seguintes regras: Se o tempo de casa for maior do que 5 anos atribuir 7,5% de aumento, senão atribuir 5%. Exibir o salário reajustado.

Entrada: *3423.50* *4* ***Saída:*** *3594.68*
Entrada: *1998.00* *9* ***Saída:*** *2147,85*

Comentários:

Este algoritmo necessita da entrada de duas informações pelo usuário: Salário (SAL) e Tempo de casa (TC). O valor de TC vai condicionar o percentual de aumento para o funcionário enquanto que o SAL servirá como base para o cálculo do aumento.

Descrição Narrativa:

- Ler o salário (SAL)
- Lero Tempo de Casa (TC)
- Se o tempo de casa for acima de 5 anos
    - Calcular 7,5% de aumento
- Se o Tempo de casa for menor ou igual a 5 anos
    - Calcular 5% de aumento
- Exibir o novo salário

Fluxograma:

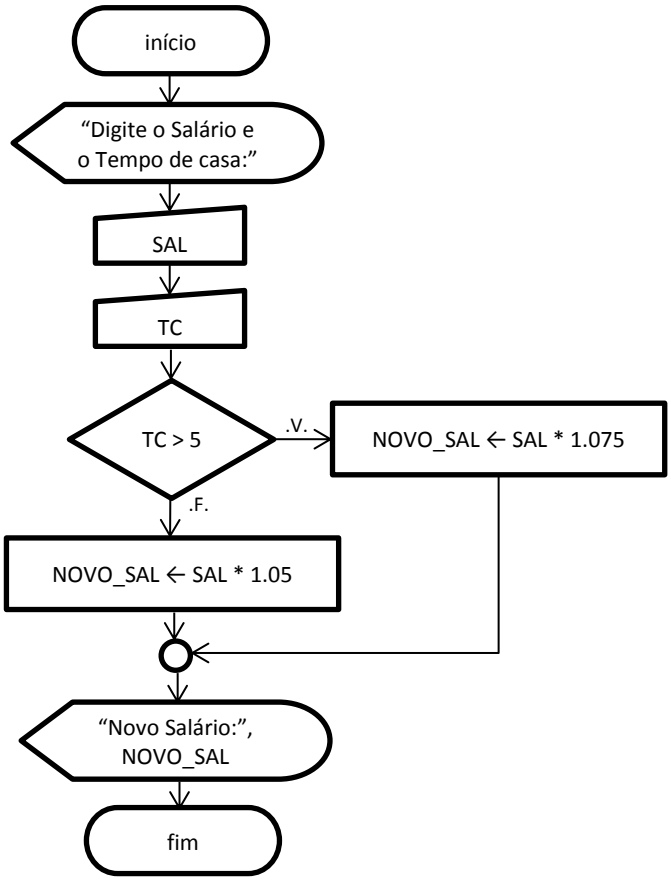


Figura 4 - Exemplo da estrutura Se... Então... Senão...

Pseudocódigo

```
Programa ex_se_entao_senao
Var
    SAL, TC, NOVOSAL : real
Inicio
    Escreva "Digite o Salário e o Tempo de Casa:"
    Leia SAL
    Leia TC
    Se TC > 5 então
        NOVO_SAL ← SAL * 1.075
    Senão
        NOVO_SAL ← SAL * 1.05
    Fim_se
    Escreva "Novo Salário: ", NOVO_SAL
Fim
```

Linguagem C

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    float sal, tc, novo_sal;
    printf("Digite o Salário e o Tempo de casa:");
    scanf("%f",&sal);
    scanf("%f",&tc);
```

```
        if (tc > 5)
        {
            novo_sal = sal * 1.075;
        }
        else
        {
            novo_sal = sal * 1.05;
        }
        printf("Novo Salário:", novo_sal);
        system("pause");
}
```

Se´s Encadeados

No “Se...Então...”, havia um possível bloco a ser executado; no “Se...Então...Senão...” havia um entre dois possíveis blocos a ser executado. O problema ocorre quando o algoritmo (ou parte dele) necessita de fazer a escolha de um entre mais de dois blocos diferentes. Para tal podemos usar o “Se Encadeado” que nada mais é do que um comando “Se” dentro do outro N vezes.

Olhe o andamento do fluxo do algoritmo através da figura abaixo:

Se a <Condição1> for verdadeira, ela executa o <Bloco1>, se falsa ela analisa a <condição2> repetindo o processo anterior até o <BlocoFinal>.

Existem outras variações no diagrama desta representação, mas esta é a mais comum.

Fluxograma

Sintaxe:

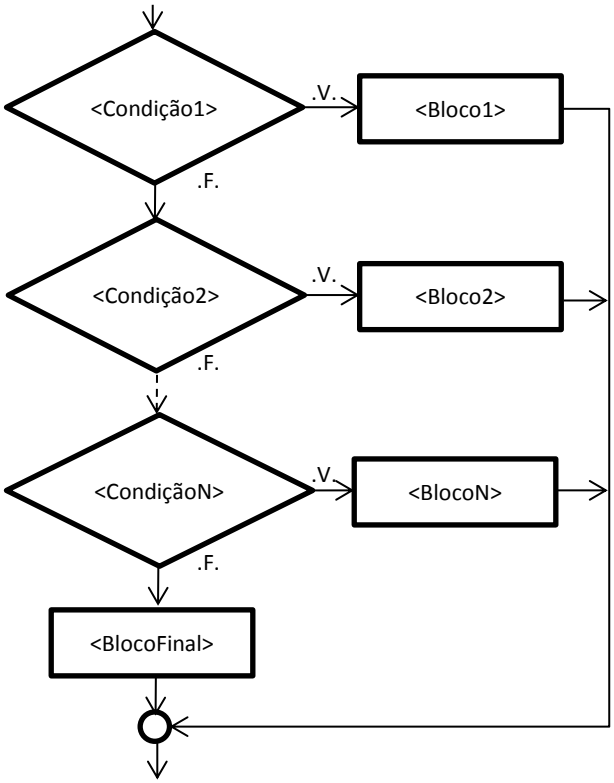


Figura 5 - Estrutura Se´s encadeados

Pseudocódigo

Sintaxe 1:

```
Se <condição1> então
    <bloco1>
Senão
    Se <condição2> então
        <bloco2>
    Senão
    ...
        Se <condiçãoN> então
            <blocoN>
        Senão
            <blocoFinal>
        Fim_se
    Fim_se
Fim_se
```

Sintaxe 2:

```
Se <condição1> então
    <bloco1>
Senão Se <condição2> então
    <bloco2>
Senão Se <condiçãoN> então
    <blocoN>
Senão
    <blocoFinal>
Fim_se
Fim_se
Fim_se
```

Comentários:

Na Sintaxe1 conseguimos ver as estruturas “Se’s” independentes; fica visível onde ela começa e termina.

Podemos usar a Sintaxe 2 também. A escrita é mais simples, mas a visualização de cada “Se” independente fica mais confusa.

Linguagem C

Sintaxe 1:

```
if (<condição1>)
    <bloco1>;
else
    if (<condição2>)
        <bloco2>;
    else
    ...
        if (<condiçãoN>)
            <blocoN>;
        else
            <blocoFinal>;
```

Comentário:

Em C não há “Fim_se” ou algo similar. Então, o final da estrutura passa a ser o ultimo “;” de uma instrução singular de um bloco ou o “}” do bloco com múltiplas instruções.

Sintaxe 1 - Plus:

Veja a mesma sintaxe anterior utilizando blocos.

```
if (<condição1>)
{
    <bloco1>;
}
else
{
    if (<condição2>)
    {
        <bloco2>;
    }
    else
    {
    ...
        if (<condiçãoN>)
        {
            <blocoN>;
        }
        else
        {
            <blocoFinal>;
        }
    }
}
```

Sintaxe 2:

```
if (<condição1>)
    <bloco1>;
else if (<condição2>)
    <bloco2>;
else if (<condiçãoN>)
    <blocoN>;
else
    <blocoFinal>;
```

Exemplo

A seguir, daremos o exemplo de um algoritmo com a estrutura de decisão “Se´s encadeados”.

Problema: Dado um número pelo usuário, informar se ele é “positivo”, “negativo” ou “nulo”, exibindo a mensagem correspondente.

Entrada: *87*	***Saída:*** *Positivo*
Entrada: *0*	***Saída:*** *Nulo*
Entrada: *-43*	***Saída:*** *Negativo*

Comentários:

Como visto no enunciado este algoritmo deve executar um de três caminhos possíveis a partir de UM dado fornecido pelo usuário. Para o desenvolvedor do algoritmo não importa qual condição ele deve analisar primeiro ou a ordem, o importante é analisar os três, afinal nós não conseguimos adivinhar o numero que o usuário digitará.

Descrição Narrativa:

- Ler o numero (N)
- Se o numero for igual a zero, exibir “nulo”
- Se o numero for maior do que zero, exibir “positivo”
- Caso contrário, exibir “negativo”.

Fluxograma

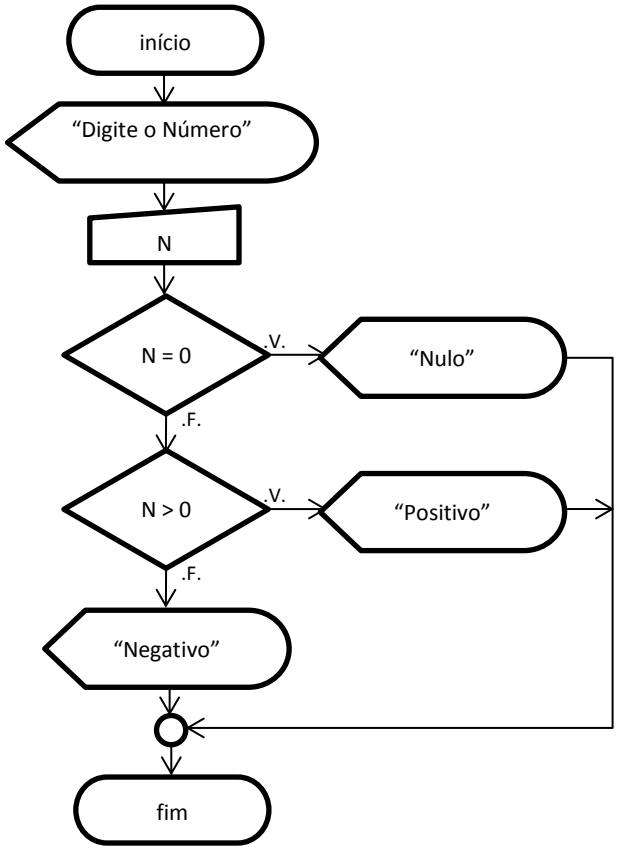


Figura 6 - Exemplo usando Se´s encadeados

Pseudocódigo

```
Programa ex_se_encadeado
Var
    N : inteiro
Inicio
    Escreva "Digite um numero: "
    Leia N
    Se N = 0 então
        Escreva "Nulo"
    Senão
        Se N > 0 então
            Escreva "Positivo"
        Senão
            Escreva "Negativo"
        Fim_se
    Fim_se
Fim
```

Linguagem C

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int n;
```

```
        printf("Digite um numero: ");
        scanf("%d",&n);
        if (n == 0)
            print("Nulo\n");
        else
            if (n > 0)
                printf("Positivo\n");
            else
                printf("Negativo\n");
        system("pause");
    }
```

ESTRUTURA DE SELEÇÃO

Escolha - switch()

Este comando é parecido com o comando “Se encadeado”. Este comando é mais prático em situações como menus, ou seja, diversas opções de escolha com condições singulares. Na verdade, poderíamos usar o Se para analisarmos diversas condições, mas o inconveniente é ter que fechar vários “fim_se”, por exemplo, se for montada uma estrutura com quatro caminhos diferentes com o comando Se, obrigatoriamente fecharíamos três “fim_se”..

A desvantagem do comando escolha ao se, é o fato dele não analisar estados com valores não equivalentes, ou seja, usar outro operador que não seja o “=”.

Fluxograma

Sintaxe:

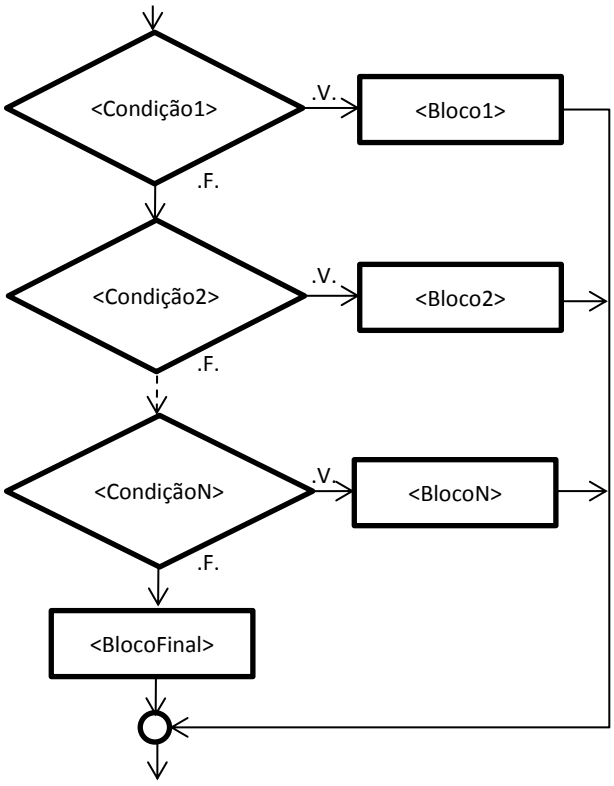


Figura 7 - Estrutura de Seleção: Escolha

Pseudocódigo

Sintaxe:

```
Escolha em <Identificador>
    Caso <Valor1> faça
        <Bloco1>
    Caso <Valor2> faça
        <Bloco2>
    ...
    Caso <ValorN> faça
        <BlocoN>
```

```
    [Caso Contrário
        <BlocoFinal>]
Fim_Escolha
```

Observações:

O <Identificador> é a variável, constante ou expressão que será analisada.

Caso os Valores não sejam identificados o fluxo do algoritmo vai para “Caso contrário” e executa o <BlocoFinal>. O trecho “Caso Contrário” não é obrigatório na construção desta estrutura.

Linguagem C

Sintaxe:

```
switch(<identificador>)
{
    case <Valor1> : <Bloco1>; [break;]
    case <Valor2> : <Bloco2>; [break;]
    ...
    case <ValorN> : <BlocoN>; [break;]
    [default      : <BlocoFinal>]
}
```

Observações:

Um bloco de chaves envolve todo conteúdo dos valores analisados.

O comando “Break” força o fluxo a sair da estrutura.

O default equivale ao “Caso Contrário” do Pseudocódigo.

A Linguagem C identifica somente valores char ou int.

Exemplo 1

A seguir, daremos o exemplo de um algoritmo com a estrutura “Escolha” analisando dados inteiros.

Problema: Dado um número pelo usuário, informar o dia da semana correspondente por estêncil.

Entrada: *4*	***Saída:*** *Quarta-Feira*
Entrada: *1*	***Saída:*** *Domingo*
Entrada: *9*	***Saída:*** *Erro – Dia Inválido*

Comentários:

Os valores válidos para o numero da semana são de 1 a 7. Qualquer numero fora deste intervalo é inválido.

Fluxograma

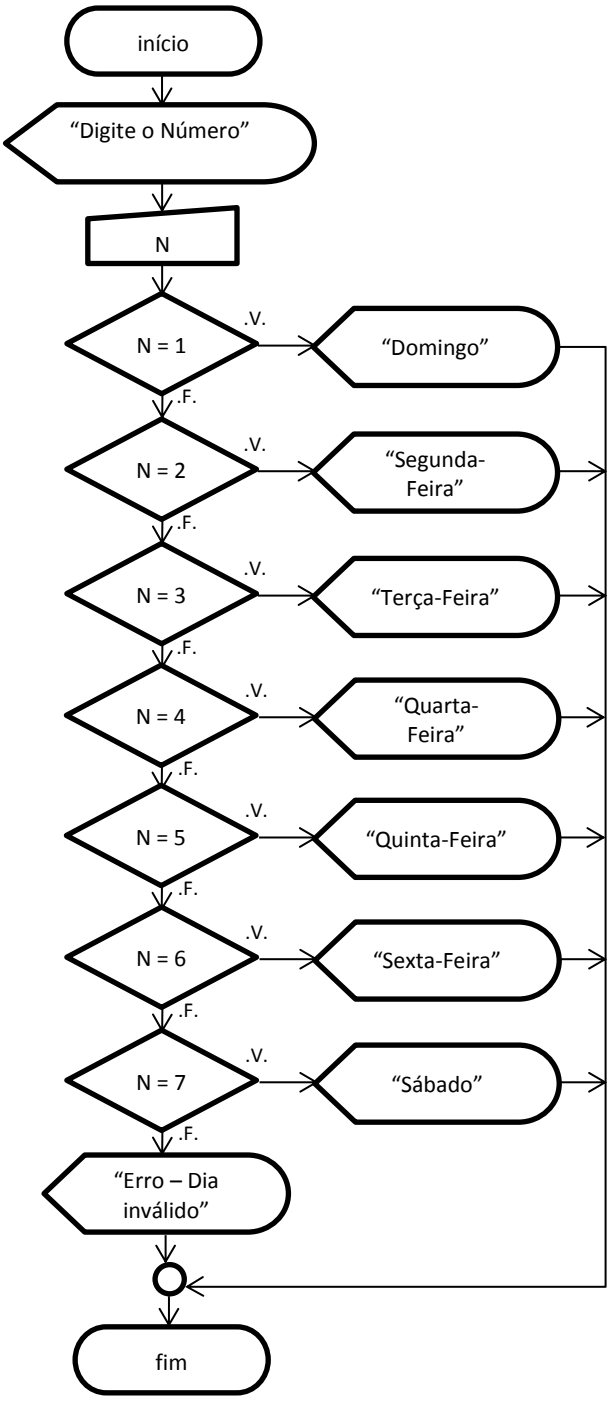


Figura 8 - Exemplo 1 da estrutura Escolha

Pseudocódigo:

```
Programa Exemplo_escolha
Var
    N : Inteiro
Inicio
    Escreva "Digite um numero:"
    Leia N
    Escolha em N
        Caso 1 faça
```

```
            Escreva "Domingo"
        Caso 2 faça
            Escreva "Segunda-feira"
        Caso 3 faça
            Escreva "Terça-feira"
        Caso 4 faça
            Escreva "Quarta-Feira"
        Caso 5 faça
            Escreva "Quinta-Feira"
        Caso 6 faça
            Escreva "Sexta-feira"
        Caso 7 faça
            Escreva "Sábado"
        Caso Contrário
            Escreva "Erro – Dia inválido"
    Fim_escolha
Fim
```

Linguagem C

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int n;
    printf("Digite um numero: ");
    scanf("%d",&n);
    switch(n)
    {
        case 1 : printf("Domingo\n"); break;
        case 2 : printf("Segunda-feira\n"); break;
        case 3 : printf("Terça-feira\n"); break;
        case 4 : printf("Quarta-feira\n"); break;
        case 5 : printf("Quinta-feira\n"); break;
        case 6 : printf("Sexta-feira\n"); break;
        case 7 : printf("Sabado\n"); break;
        default : printf("Erro – Dia invalido\n");
    }
    system("pause");
}
```

Exemplo 2

A seguir, daremos o exemplo de um algoritmo com a estrutura "Escolha" analisando dados char.

Problema: Dada uma letra minúscula, informar se ela é vogal ou consoante.

Entrada: *e* ***Saída:*** *Vogal*
Entrada: *z* ***Saída:*** *Consoante*

Comentários:

Neste algoritmo estamos considerando que foi digitada uma letra e minúscula. Reparem como o valor comparado será tratado em linguagem C.

Fluxograma

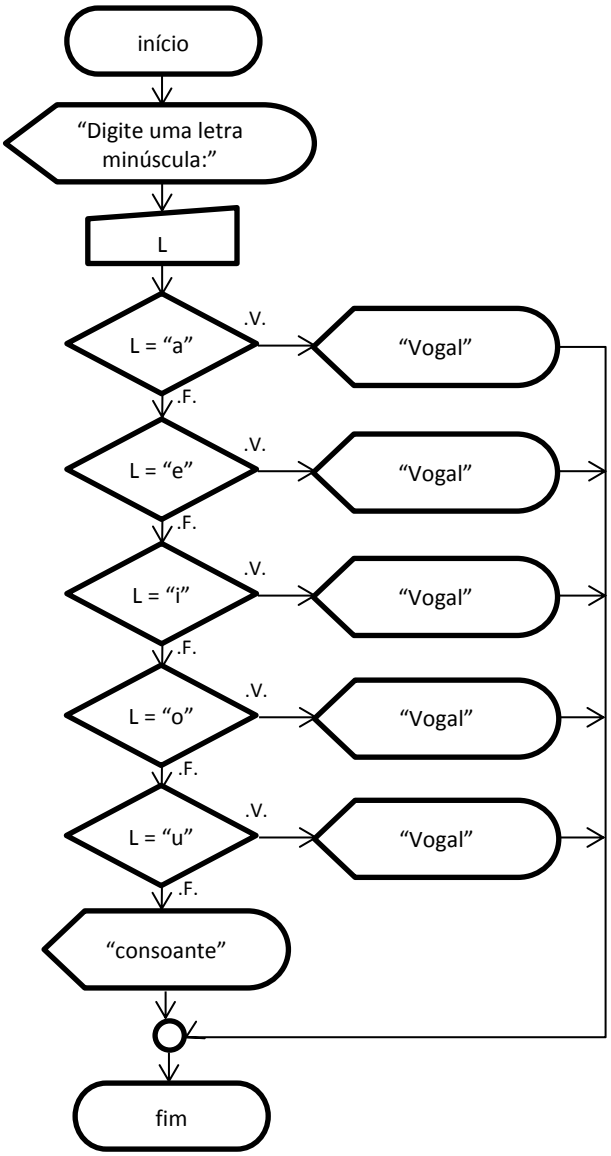


Figura 9 - Exemplo 2 estrutura Escolha

Pseudocódigo:

```
Programa Exemplo2_escolha
Var
    L : Caractere
Inicio
    Escreva “Digite uma letra minúscula:”
    Leia L
    Escolha em L
        Caso "a" faça
            Escreva “Vogal”
        Caso "e" faça
            Escreva “Vogal”
        Caso "i" faça
            Escreva “Vogal”
        Caso "o" faça
            Escreva “Vogal”
        Caso "u" faça
            Escreva “Vogal”
```

```
        Caso Contrário
            Escreva "Consoante"
    Fim_escolha
Fim
```

Linguagem C

```
#include <stdio.h>
#include <stdlib.h>

// FORMA 1
int main()
{
    char l;
    printf("Digite uma letra minuscula: ");
    scanf("%c",&l);
    switch(l)
    {
        case 'a' : printf("Vogal\n"); break;
        case 'e' : printf("Vogal\n"); break;
        case 'i' : printf("Vogal\n"); break;
        case 'o' : printf("Vogal\n"); break;
        case 'u' : printf("Vogal\n"); break;
        default : printf("Consoante\n");
    }
    system("pause");
}
```

Ou

```
#include <stdio.h>
#include <stdlib.h>

// FORMA2
int main()
{
    char l;
    printf("Digite uma letra minuscula: ");
    scanf("%c",&l);
    switch(l)
    {
        case 'a' : ;
        case 'e' : ;
        case 'i' : ;
        case 'o' : ;
        case 'u' : printf("Vogal\n"); break;
        default : printf("Consoante\n");
    }
    system("pause");
}
```

A execução das duas formas C é exatamente a mesma. Lembram que o "break" forçava a saída da estrutura? Então, na segunda forma se o fluxo entrar em 'a', 'e', 'i' ou 'o', o fluxo "escorrega" até o 'u', exibe "Vogal" e o "break" força a saída da estrutura. Qualquer outra letra que não seja uma das anteriores o fluxo é levado para o "default" e é exibido "Consoante".

ESTRUTURAS DE REPETIÇÃO

As estruturas de repetição (laços) são eficazes na montagem do algoritmo sempre que parte do Algoritmo necessite ser repetida, ou seja, utilizamos rotinas de repetição toda vez que um trecho do programa for redundante.

Existem três tipos de laços. Os laços condicionais: “Enquanto” e “Repita” e o laço Contador “Para”.

A seguir detalharemos cada um deles.

Laço Condicional: Enquanto – while()

O laço condicional Enquanto, como sugere o próprio nome, é um laço condicional e funciona enquanto a <condição> resultar Verdade. Assim, ela repete o <bloco de repetição> enquanto a condição for satisfeita; quando a condição resultar Falso o fluxo do programa segue a sua sequencia.

Neste laço a execução do bloco vem depois da análise da condição e o <bloco de repetição> pode ser executado nenhuma vez.

Fluxograma

Sintaxe:

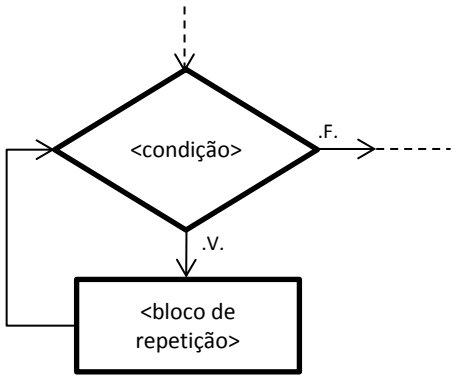


Figura 10 - Laço Enquanto

Pseudocódigo

Sintaxe:

```
Enquanto <condição> faça
    <bloco de repetição>
Fim_enquanto
```

Linguagem C

Sintaxe:

```
while (<condição>)
{
    <bloco de repetição>;
}
```

Exemplo

A seguir, daremos o exemplo de um algoritmo com o Laço condicional “Enquanto”.

Problema: Fazer um Algoritmo que some números até que o usuário digite Zero (0), exibindo o resultado.

Entrada: *5 9 3 -3 10 0* ***Saída:*** *24*

Entrada: *15 20 0* ***Saída:*** *35*

Comentários:

Neste algoritmo a quantidade de voltas executadas é indeterminada, somente quando usuário digitar zero terminará o laço e, na sequência, exibe o resultado da somatória.

Fluxograma

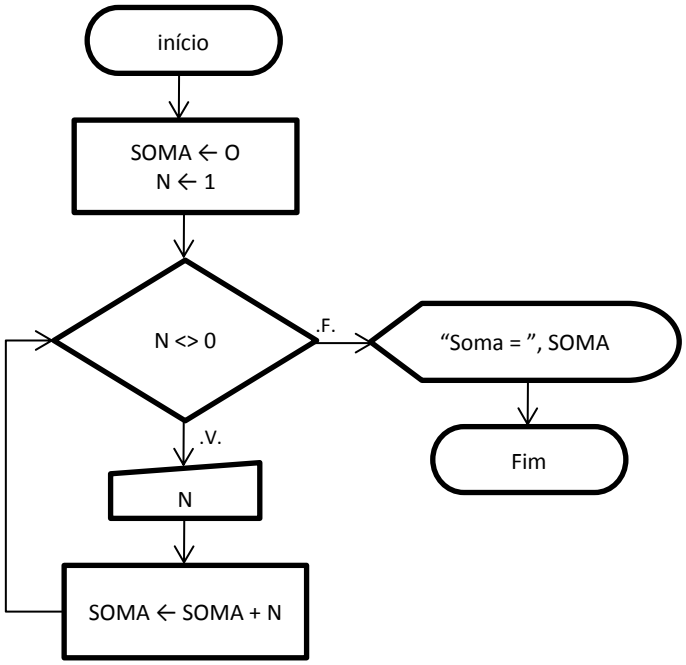


Figura 11 - Exemplo usando o Laço Enquanto

Pseudocódigo

```
Programa Ex_Enquanto
Var
    SOMA, N: Real
Inicio
    SOMA ← 0
    N ← 1
    Enquanto N <> 0 faça
        Leia N
        SOMA ← SOMA + N
    Fim_Enquanto
    Escreva "Soma =", SOMA
Fim
```

Linguagem C

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
```

```
    float soma, n;
    soma = 0;
    n = 1
    printf("Digite números ou zero para finalizar.\n");
    while(n != 0)
    {
        scanf("%f",&n);
        soma = soma + n;
    }
    printf("Soma = %.1f\n",soma);
    system("pause");
}
```

Laço Condicional: Repita– do ... while()

O laço condicional Repita, é um laço condicional e funciona até que a condição seja verdadeira. Assim, ela repete o <bloco de repetição> enquanto a condição não for satisfeita; quando a condição resultar Verdade o fluxo do programa segue a sua sequencia.

Neste laço a execução do bloco vem antes da análise da condição e o <bloco de repetição> é executado ao menos uma vez.

Fluxograma

Sintaxe:

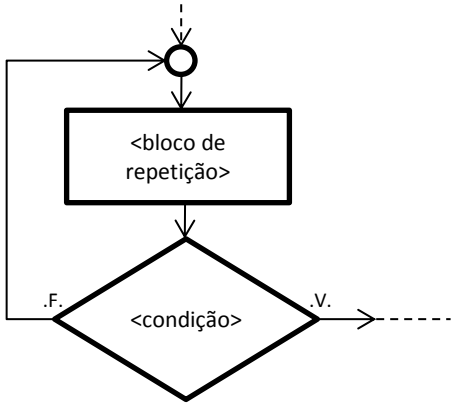


Figura 12 - Laço Repita

Pseudocódigo

Sintaxe:

```
Repita
    <bloco de repetição>
Até que <condição>
```

Linguagem C

Sintaxe:

```
do
{
    <bloco de repetição>;
}
while (<condição>)
```

Observação:

Na linguagem C o funcionamento do comando Repita é diferente do que diz a teoria, ou seja, o laço continua dando voltas enquanto a condição for verdadeira, não se for falso, como diz a teoria.

Exemplo

Para um melhor entendimento e comparação entre os laços Enquanto e Repita, utilizaremos o mesmo exemplo.

Problema: Fazer um Algoritmo que some números até que o usuário digite Zero (0), exibindo o resultado.

Entrada: *5 9 3 -3 10 0* ***Saída:*** *24*
Entrada: *15 20 0* ***Saída:*** *35*

Comentários:

Neste algoritmo a quantidade de voltas executadas é indeterminada, somente quando usuário digitar zero terminará o laço e, na sequência, exibe o resultado da somatória.

Fluxograma

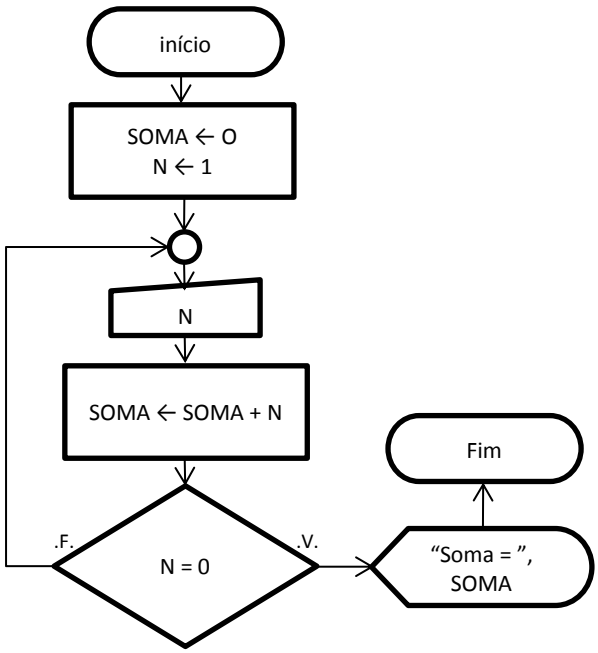


Figura 1213 - Exemplo usando o Laço Repita

Pseudocódigo

```
Programa Ex_Repita
Var
    SOMA, N: Real
Inicio
    SOMA ← 0
    N ← 1
    Repita
        Leia N
        SOMA ← SOMA + N
    Até que n = 0
    Escreva "Soma =", SOMA
```

```
Fim
```

Linguagem C

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    float soma, n;
    soma = 0;
    n = 1
    printf("Digite números ou zero para finalizar.\n");
    do
    {
        scanf("%f",&n);
        soma = soma + n;
    }
    while(n != 0)
    printf("Soma = %.1f\n",soma);
    system("pause");

}
```

Observação:

Repare que a condição do "while" mudou pela especificidade da linguagem C em executar o bloco de repetição enquanto for verdade a condição no laço Repita ao invés de sair do laço como diz a teoria.

Laço Contador: Para – for()

O laço Para é o conhecido laço contador. Ele é mais fácil de utilizar em situações onde o numero de voltas sejam previsíveis no algoritmo.

Todos os aspectos do laço são configurados dentro do próprio símbolo representativo do laço. Os termos existentes no laço são:

<cont> → Contador. Variável inteira que tem a responsabilidade de dizer qual é a volta atual que está sendoexecutada.

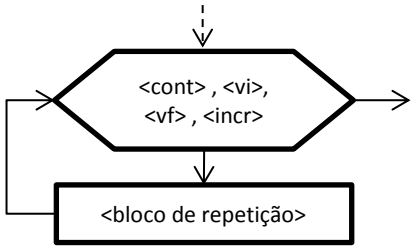
<v.i.> → Valor Inicial. Variável, constante ou expressão inteira que representa o valor dado ao contador na primeira volta do laço.

<v.f.> → Valor Final. Variável, constante ou expressão inteira que representa o valor final na qual o valor do contador finalizará.

<incr> →Incremento. Variável, constante ou expressão inteira que diz quanto o contador será incrementado a cada volta dada.

Fluxograma

Sintaxe:



Pseudocódigo

Sintaxe:

```
Para <cont> de <vi> até <vf> inc <incr> faça
    <bloco de repetição>
Fim_para
```

Linguagem C

Sintaxe:

```
for (<inicialização> ; <condição> ; <incremento>)
{
    <bloco de repetição>;
}
```

Exemplo

Problema: Dado um número, apresentar os seus dez múltiplos.

Entrada: *5* ***Saída:*** *5 10 15 20 25 30 35 40 45 50*

Entrada: *2* ***Saída:*** *2 4 6 8 10 12 14 16 18 20*

Comentários:

Diferentemente dos exemplos dos dois últimos laços, neste exemplo, a leitura do dado deve ser efetuada antes do laço, não dentro.

Fluxograma

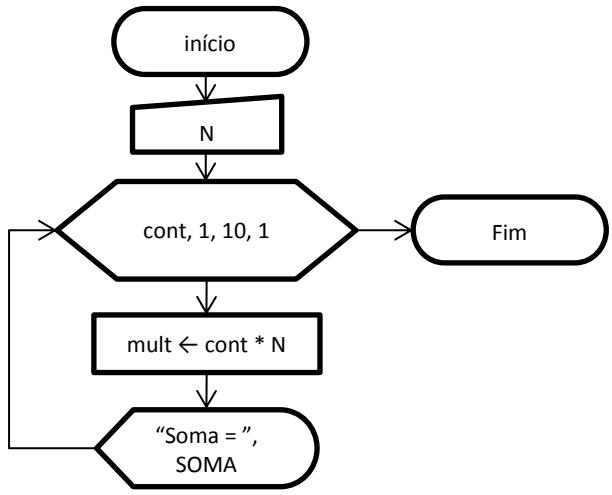


Figura 1314 - Exemplo usando o Laço Repita

Pseudocódigo

```
Programa Ex_Para
Var
    Cont, mult, N: Real
Inicio
    Leia N
    Para cont de 1 até 10 inc 1 faça
        Mult ← cont + N
        Escreva Mult
```

```
    Fim_para
Fim
```

Linguagem C

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int cont, n, mult;
    printf("Digite números ou zero para finalizar.\n");
    scanf("%d",&n);
    for (cont=1; cont<=10; cont++)
    {
        mult = cont * n;
        printf("%d\n",mult);
    }
    system("pause");
}
```