Resumo Técnicas de Desenvolvimento de Algoritmo by Victor Claudino

Prova 1: Estruturas de Dados para o Desenvolvimento de Algoritmos.

Hardware e Software:

Um computador é um dispositivo capaz de realizar cálculos e tomar decisões lógicas em velocidades bilhões de vezes mais rápidas que os seres humanos. Por exemplo, muitos dos computadores pessoais de hoje podem executar bilhões de adições por segundo. Uma pessoa que utilizasse uma calculadora de mesa portátil poderia gastar uma vida inteira realizando cálculos e, mesmo assim, não completaria o mesmo número de cálculos que um computador pessoal poderoso pode executar em um segundo! (Pontos a serem considerados, como você saberia se a pessoa somou esses números corretamente? Como você saberia se o computador somar os números corretamente!) Os supercomputadores mais rápidos de hoje podem executar milhares de trilhões (quatrilhões) de instruções por segundo! Para se ter uma ideia, um computador que executa um quatrilhão de instruções por segundo pode realizar mais de 100.000 cálculos por segundo para cada pessoa no planeta!

Computadores processam dados sob o controle de conjuntos de instruções chamados programas de computador. Esses pro gramas guiam o computador por meio de conjuntos ordenados de ações especificadas por pessoas a quem chamamos de programa dores de computador.

Um computador é composto de vários dispositivos (por exemplo, teclado, tela, mouse, disco rígido, memória, DVDs e unidades de processamento) que são denominados hardware. Os programas executados em um computador são chamados de software. Os custos de hardware bem diminuído drasticamente nos últimos anos, a ponto de os computadores pessoais serem se tornado um artigo comum. Você aprenderá métodos de desenvolvimento de software comprovados que podem reduzir os custos do desenvolvimento de software programação estruturada.

Organização de Computadores:

1-Unidade de Entrada: Essa é a seção receptora do computador. Ela obtém informações (dados e programas) de vários dispositivos de entrada e as coloca à disposição das outras unidades, de forma que possam ser processadas. A maioria das informações, hoje, é inserida nos computadores por meio de dispositivos como teclados e mouses. As informações também podem ser inseridas de muitas outras maneiras, incluindo comandos de voz, varredura de imagens e códigos de barras, leitura de dispositivos de armazenamento secundários (como discos rígidos, unidades de CD, unidades de DVD e unidades USB-também chamadas de pendrives) e recebimento de informações pela Internet (por exemplo, quando você baixa vídeos do YouTube", e-books da Amazon e outros meios similares);

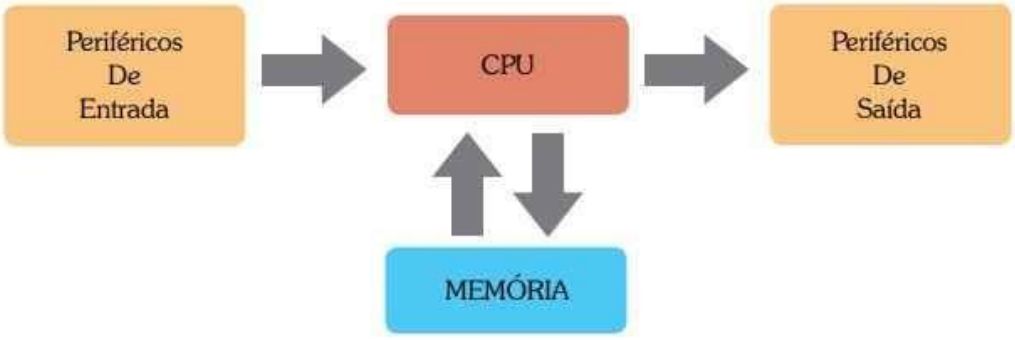
2-Unidade de Saída: Essa é a seção de expedição do computador. Ela pega as informações que foram processadas pelo computador e as coloca em vários dispositivos de saída para torná-las disponíveis ao uso fora dele. A maior parte das informações que saem de computadores hoje em dia é exibida em telas, impressa, tocada em players de áudio (como os populares iPods da Apple) ou usada para controlar outros dispositivos. Os computadores também podem enviar suas informações para redes como a Internet;

3-Unidade de Memória: Essa é a seção de armazenamento de acesso rápido do computador, que tem uma capacidade relativamente baixa. Retem informações obtidas por meio da unidade de entrada, de forma que elas possam estar imediatamente disponíveis para processamento quando forem necessárias. A unidade de memória retem também informações processadas ai que elas possam ser colocadas em dispositivos de saída pela unidade de saída. As informações na unidade de memória são voláteis: elas normalmente se perdem quando o computador é desligado. A unidade de memória é frequentemente chamada de memória ou memória primaria;

4-Unidade Lógica e Aritmética: Essa é a seção de processamento do computador. Ela é responsável por executar cálculos como adição, subtração, multiplicação e divisão. Também contém os mecanismos de decisão que permitem os computadores, por exemplo, comparar dois itens na unidade de memória e determinar se eles são ou não iguais. Nos sistemas atuais, a ULA está próxima a UC, dentro da CPU;

5-Unidade Central de Processamento: Essa é a seção administrativa de computador. Ela coordena e supervisiona o funcionamento das outras seções. A CPU diz à unidade de entrada quando as informações devem ser lidas para a unidade de memória, diz à ALU quando as informações da unidade de memória devem ser utilizadas em cálculos e diz à unidade de saída quando enviar as informações da unidade de memória para certos dispositivos de saída. Muitos dos computadores de hoje possuem várias CPUs e, portanto, podem realizar muitas operações simultaneamente esses computadores são chamados de multiprocessadores. Um processador multi-core executa e multiprocessamento em um único chip integrado, exemplo, um processador dual-core tem duas CPUs e um processador quad-core tem quatro CPU;

6-Unidade de Armazenamento Secundário: Categoria de dispositivos de armazenamento de dados de grande capacidade utilizada para armazenar programas ou dados que não estão sendo ativamente usados. Esses dispositivos, como discos rígidos, CDs, DVDs e unidades flash, servem como depósitos persistentes para informações, preservando-as mesmo quando o computador está desligado. Ao contrário da memória primária, as informações nos dispositivos de armazenamento secundário levam mais tempo para serem acessadas, mas oferecem uma capacidade substancialmente maior a um custo por unidade mais baixo. Exemplos incluem CDs, DVDs e unidades flash, que podem armazenar grandes quantidades de dados por longos períodos, variando de horas a anos.



Linguagem de Máquina, Simbólica e de Alto Nível:

Linguagem de Máquina:

A linguagem de máquina é a forma mais básica de linguagem de programação que um computador pode entender diretamente. Consiste em instruções binárias, representadas por sequências de 0s e 1s, correspondendo a operações específicas que o processador pode executar. Cada tipo de processador tem sua própria linguagem de máquina única. A programação em linguagem de máquina é complexa e de difícil compreensão para os humanos, exigindo conhecimento profundo da arquitetura do processador.

Linguagem Simbólica:

A linguagem simbólica é uma forma mais legível e mnemônica da linguagem de máquina. Usa símbolos e abreviações para representar as instruções em vez de códigos binários. Embora mais fácil de entender do que a linguagem de máquina, ainda está intimamente ligada à arquitetura do processador. Os mnemônicos são convertidos para linguagem de máquina durante um processo chamado montagem. Programar em linguagem simbólica é um passo adiante em termos de legibilidade, mas ainda requer um entendimento profundo do hardware subjacente.

Linguagem de Alto Nível:

Em contraste, as linguagens de alto nível são mais distantes da arquitetura do hardware. Elas usam instruções e palavras-chave que se aproximam da linguagem humana, facilitando a compreensão para os programadores. Exemplos incluem Python, Java e C++. As instruções em linguagens de alto nível não estão diretamente ligadas à arquitetura do processador e geralmente são independentes da plataforma, oferecendo portabilidade. Essas linguagens são mais eficientes e rápidas para os programadores, mas dependem de compiladores ou interpretadores para traduzir o código para linguagem de máquina compreensível pelo computador.

A Biblioteca-padrão de C:

A biblioteca-padrão de C é uma coleção de funções e macros pré-definidas que oferecem suporte a uma variedade de operações em linguagem C. Essas funções abrangem áreas como entrada/saída, manipulação de strings, alocação de memória, manipulação de arquivos, matemática, entre outras.

Principais características da biblioteca-padrão de C:

1-Entrada/Saída (stdio.h): Fornece funções como `printf` e `scanf` para entrada e saída de dados;

2- Manipulação de Strings (string.h): Oferece funções para operações em strings, como `strlen`, `strcpy` e `strcat`;

3-Alocação de Memória (stdlib.h): Contém funções como `malloc` e `free` para alocar e liberar memória dinamicamente;

4-Manipulação de Arquivos (stdio.h, stdlib.h): Inclui funções como `fopen`, `fclose`, `fread` e `fwrite` para trabalhar com arquivos;

5-Matemática (math.h): Fornece funções matemáticas, como `sqrt`, `sin`, `cos`, entre outras; 6-Caracteres e Tabela ASCII (ctype.h): Contém funções como `isalpha`, `isdigit`, para testar e manipular caracteres;

7-Tempo (time.h): Inclui funções relacionadas ao tempo, como `time` e `ctime`;

8-Ordenação e Pesquisa (stdlib.h): Oferece funções como `qsort` para ordenação e `bsearch` para pesquisa em arrays;

Essas são apenas algumas das muitas funções disponíveis na biblioteca-padrão de C. A inclusão dessas bibliotecas é realizada por meio de diretivas `#include` no início de um programa C para que suas funcionalidades possam ser utilizadas. O uso eficaz da biblioteca-padrão permite que os programadores escrevam código eficiente e portável em C.

C++:

C++ foi desenvolvido por Bjarne Stroustrup no Bell Laboratories. C++ tem suas raízes em C e apresenta várias características que melhoram a linguagem C, mas o mais importante é que fornece recursos para a programação orientada a objetos. C++ tornou-se dominante na indústria e no setor acadêmico.

Objetos são essencialmente componentes de software reutilizáveis que modelam itens do mundo real. O uso de uma abordagem de implementação e design modulares, orientada a objetos, pode tornar os grupos de desenvolvimento de software muito mais produtivos do que é possível, utilizando técnicas de programação que não suportam orientação a objetos.

Java:

Microprocessadores vêm provocando um impacto profundo nos aparelhos eletrodomésticos e eletrônicos de consumo. Reconhecendo, em 1991, a Sun Microsystems financiou um projeto corporativo anterior de desenvolvimento ao qual deu o codinome Green. O projeto resultou no desenvolvimento de uma linguagem baseada em C++, que seu criador, James Gosling, denominou Oak (carvalho, em inglês, em homenagem a uma árvore que ele via através da janela de seu escritório na Sun). Mais tarde, descobriram que já existia uma linguagem de programação chamada Oak. Quando um grupo de pessoas da Sun visitava uma cafeteria local, o nome Java foi sugerido e pegou.

No entanto, o projeto Green enfrentou algumas dificuldades. Na década de 1990, o mercado para aparelhos eletrônicos de consumo inteligentes não se desenvolvia tão rapidamente quanto a Sun havia previsto. O projeto corria o risco de ser cancelado. Por pura sorte, a popularidade da World Wide Web explodiu em 1993, e o pessoal da Sun viu o potencial imediato de usar o Java para criar páginas da Web com conteúdo dinâmico (por exemplo, interatividade, animações e coisas desse tipo). Isso deu nova vida ao projeto.

A Sun anunciou a Java publicamente em uma conferência em maio de 1995. Ela atraiu a atenção da comunidade comercial devido ao interesse fenomenal na World Wide Web. A Java agora é usada para desenvolver aplicativos empresariais de grande porte, para aumentar a funcionalidade de servidores web (os computadores que oferecem o conteúdo que os navegadores da web acessam), para oferecer aplicativos a aparelhos destinados ao consumo (tais como telefones celulares, pagers e assistentes pessoais digitais) e muitas outras finalidades.

Fortran, Cobol, Pascal e Ada:

Centenas de linguagens de alto nível foram desenvolvidas, mas apenas algumas obtiveram ampla aceitação. A FORTRAN (FORmula TRANslator) foi desenvolvida pela IBM Corporation em meados da década de 1950 para ser usada no desenvolvimento de aplicativos científicos e de engenharia que exigem cálculos matemáticos complexos. Ainda é amplamente utilizada, especialmente em aplicativos de engenharia.

A COBOL (Common Business Oriented Language) foi desenvolvida no final da década de 1950 por fabricantes de computadores, pelo governo dos EUA e por usuários de computadores industriais. A COBOL é utilizada principalmente em aplicativos comerciais que exigem manipulação precisa e eficiente de grandes quantidades de dados. Muitos softwares para aplicações comerciais ainda são programados em COBOL.

Durante a década de 1960, o esforço para o desenvolvimento de software de grande porte encontrou sérias dificuldades. As entregas de software sofriam atrasos frequentes, os custos ultrapassavam em muito os orçamentos e os produtos finais não eram confiáveis. As pessoas perceberam que o desenvolvimento de software era uma atividade mais complexa do que haviam imaginado. Durante essa década, a pesquisa resultou na evolução da programação estruturada - um enfoque disciplinado para a escrita de programas mais claros e mais fáceis de testar, depurar e modificar do que os grandes programas produzidos com técnicas anteriores.

Um dos resultados mais tangíveis dessa pesquisa foi o desenvolvimento da linguagem de programação Pascal, pelo professor Niklaus Wirth, em 1971. Em homenagem ao matemático e filósofo do século XVII, Blaise Pascal, ela foi projetada para o ensino de programação estruturada e logo se tornou a linguagem de programação preferida na maioria das faculdades. Pascal não tinha muitos dos recursos necessários nas aplicações comerciais, industriais e do governo, de modo que não foi muito bem aceita fora dos círculos acadêmicos.

A linguagem Ada foi desenvolvida com o patrocínio do Departamento de Defesa (DoD) dos Estados Unidos durante a década de 1970 e início dos anos 1980. Centenas de linguagens separadas eram utilizadas para produzir sistemas de software de grande porte do DoD. O departamento queria uma linguagem que se ajustasse à maior parte de suas necessidades. A linguagem Ada recebeu esse nome em homenagem à Lady Ada Lovelace, filha do poeta Lord Byron. O primeiro programa de computador do mundo, datando do início do século XIX (para a Analytical Engine Mechanical Computing Device, projetada por Charles Babbage), foi escrito por ela. Uma característica importante da Ada é chamada de multitarefa; ela permite que os programadores especifiquem que muitas atividades devem acontecer em paralelo. Embora a multitarefa não faça parte do padrão C ou CVSL, ela está disponível em diversas bibliotecas completas.

Lógica:

Para os profissionais da área de tecnologia, o uso da lógica é um fator importante a ser considerado, porque a todo tempo os programadores e analistas e precisam resolver problemas do seu cotidiano, saber lidar com problemas de ordem administrativa de controle, de planejamento e estratégia requer atenção e boa técnica para representar esses problemas.

Lógica Aplicada no Desenvolvimento de Softwares ou Programas:

Muitos desenvolvedores “desenham” o processo do programa antes do desenvolvimento do código, esse desenhar seriam as etapas que o programa deveria realizar e uma forma testar os pontos chaves do programa verificando se não existe nenhum erro de lógica

de programação. Esses desenhos são chamados de diagramas de blocos que após a conclusão poderá ser interpretado por qualquer desenvolvedor e ser desenvolvido em qualquer linguagem disponível no mercado.

A técnica mais importante no projeto da lógica de programas denomina-se programação estruturada, a qual consiste em uma metodologia de projeto objetivando:

*Agilizar a codificação da escrita da programação;

*Facilitar a depuração da sua leitura;

*Permitir a verificação de possíveis falhas apresentadas pelos programas;

E deve ser composta por quatro passos fundamentais:

*Facilitar as alterações e atualizações dos programas;

*Escrever as instruções em sequência ligadas entre si apenas por estruturas sequencias, repetitivas ou de seleção;

*Escrever instruções em grupos pequenos e combiná-las;

*Distribuir módulos do programa entre os diferentes programadores que trabalharão sobre a supervisão de um programador sênior ou chefe.

Nomenclaturas:

Muitos profissionais utilizam a linguagem para denominação de símbolos que representam a linha de raciocínio lógico de fluxogramas, diagramas de blocos e algoritmos, vejam a definição de cada um:

Fluxograma: ferramenta utilizada pelos profissionais de análise de sistemas para descrever o fluxo de ação de uma atividade automatizada ou manual. Usa símbolos denominados pela norma ISO 5807:1985, é representado por alguns desenhos geométricos.

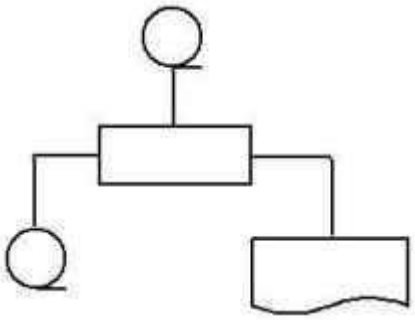
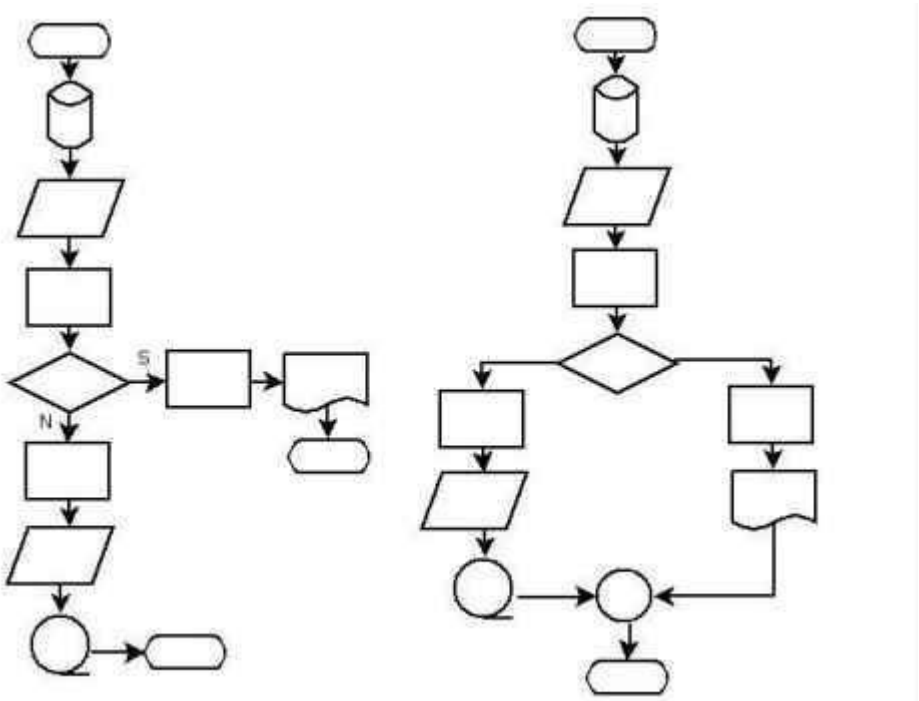
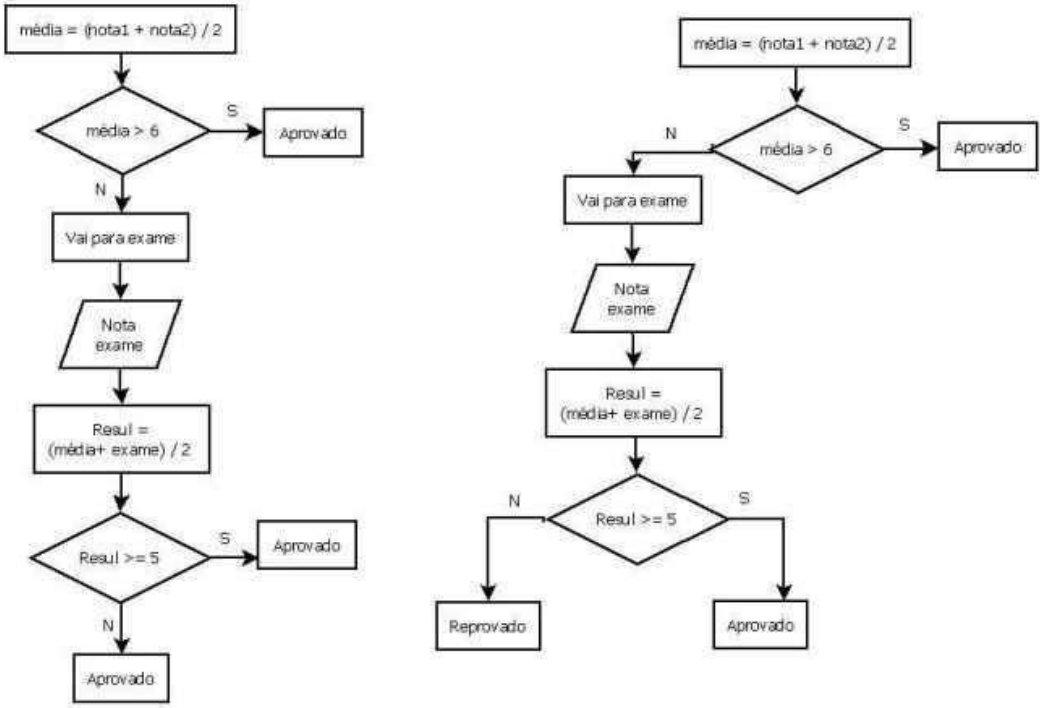


Diagrama de blocos: também conhecido como diagrama de fluxo (diferente de fluxograma), uma ferramenta utilizada pelo programador, onde o objetivo do uso dessa ferramenta é descrever o método e a sequência de ações ou eventos a serem executadas pelo computador. Também é utilizada diversas formas geométricas para representar as atividades, esses símbolos são conhecidos mundialmente e definidos pela norma ISO 5807:1985(E), após a criação do diagrama de blocos a próxima etapa seria a codificação do programa na linguagem escolhida pelo programador.



Algoritmo: conjunto de regras formais que serão utilizados para a resolução do problema, nessa solução pode-se dizer que estão embutidas as fórmulas de expressões aritméticas. Os algoritmos podem ser definidos por meio da escrita ou representado por meio de formas geométricas, resumindo o algoritmo pode ser definido como a resolução do problema e transformando esse problema em um programa que seja possível de ser executado por um programador.



A forma de representação gráfica, segundo ISO 5807:1985 é uma forma de representar os dados e os procedimentos a serem executados por um programa a partir da linha de raciocínio lógico de um programador para demonstrar o resultado final de um programa.

Resolução de Problemas:

Para que um diagrama de blocos seja desenvolvido de forma correta, deve-se levar em consideração como procedimentos prioritários as seguintes regras:

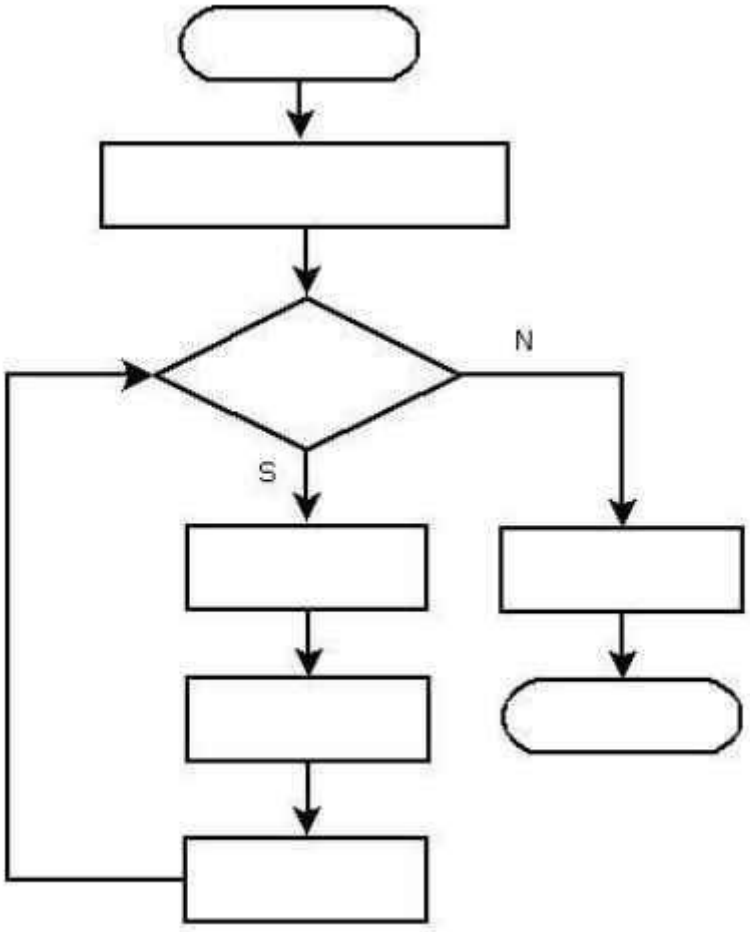
*Diagrama de blocos devem ser feitos e quebrados em vários níveis;

*Para o desenvolvimento correto de um diagrama de bloco, ele deve ser iniciado de cima para baixo;

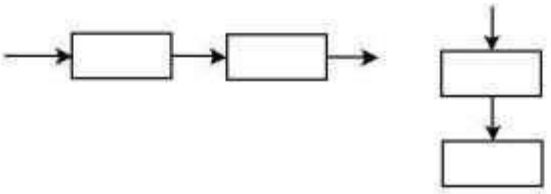
*Não se deve em hipótese alguma ocorrer cruzamentos de linhas de fluxo de dados entre os símbolos;

Os diagramas de blocos podem ser representados de várias formas e isso não impede que a solução seja entregue e sem erros, vejamos alguns exemplos.

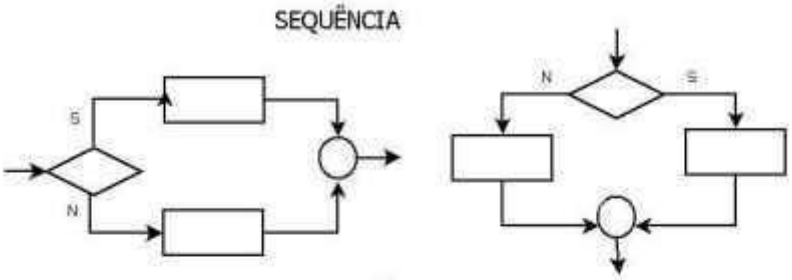
1-Linear: A técnica linear é conhecida como um modelo tradicional de desenvolvimento e resolução de problemas.



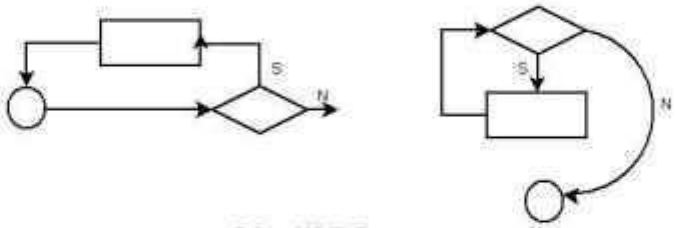
2-Estruturada: Essa técnica é mais utilizada pelos profissionais de processamento de dados. A sequência, a seleção e a iteração são as três estruturas básicas para a construção do diagrama de bloco.



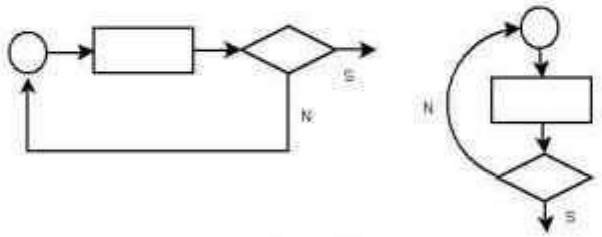
SEQUËNCIA



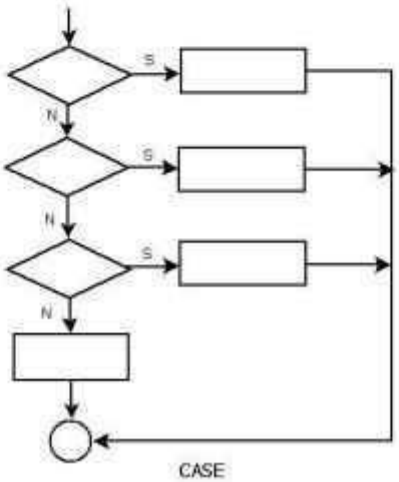
SE...ENTÃO... FIM_SE



DO... WHILE

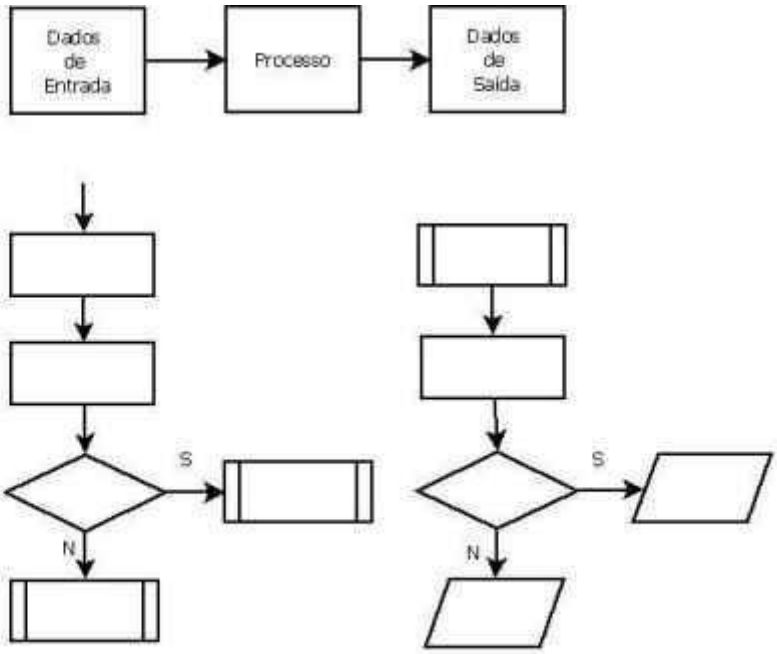


DO... UNTIL



CASE

3-Modular: A técnica da lógica modular deve ser elaborada como uma estrutura de partes independentes, denominadas de módulos, cujo procedimento é controlado por um conjunto de regras.



4-Português Estruturado: Essa técnica é baseada no Program Design Language – PDL, onde a liguagem é apresentada e codificada na língua portuguesa, ela foi desenvolvida com o propósito de ser um ferramenta comercial que poderia se utilizada com qualquer linguagem, por ser um escrita didática começou a ser utilizada para o ensino de programação, exemplo:

```
Programa  MÉDIA
Var
        Resultado: caractere
        N1, N2, N3, N4: real
        SOMA, MÉDIA: real
Inicio
        Leia N1, N2, N3, N4
        SOMA = N1 + N2 + N3 + N4
        MÉDIA =  SOMA / 4
        SE (MÉDIA>=6) ENTÃO
                RESULTADO = "APROVADO"
        SENÃO
                RESULTADO = "REPROVADO"
        FIM_SE
        Escreva "Nota 1: " , N1
        Escreva "Nota 2: " , N2
        Escreva "Nota 3: " , N3
        Escreva "Nota 4: " , N4
        Escreva "Soma: " , SOMA
        Escreva "Média: " , MÉDIA
        Escreva "Resultado: " , RESULTADO
fim
```

Códigos:

Para que haja a possibilidade de comunicação entre homem e máquina, é necessário que as palavras da linguagem sejam traduzidas para a linguagem de máquina e vice-versa. Para que isso seja possível, é necessário que se estabeleça qual sequência de bit´s corresponde a cada caractere usado na linguagem escrita. Um dos padrões de codificação bastante utilizado é o ASC II (American Standard Code for Information Interchange ou Código padrão Americano para Intercâmbio de Informação), estabelecido pelo ANSI (American National Standards Institute). Nessa codificação, cada caractere é representado por uma sequência de 8 bits (denominado byte). Cada caractere possui um código ASC diferente, incluindo a, A, á, ã, 1, #, &, -, }, etc. etc. Com 8 bits, temos 256 combinações diferentes de 0 a 255.

Podemos verificar esse fato acessando o código das teclas com o ALT esquerdo e uma sequência no teclado numérico. Como exemplo:

ALT + 65 = “A”

ALT + 66 = “B”

ALT + 33 = “!”

ALT + 166 = “ª”

ALT + 98 = “b”

E, assim, para cada caractere, temos um código, incluindo espaço, enter, ESC e etc.

Algoritmos:

Consideraremos um algoritmo uma sequência de instruções cuja execução resulta na realização de uma determinada tarefa. De uma maneira natural, alguns tipos de algoritmos estão presentes no nosso dia a dia, não necessariamente envolvendo aspectos computacionais. Uma receita de bolo, uma partitura musical, um manual de utilização do Playstation são algoritmos que descrevem, passo a passo, o que fazer para chegar a um resultado esperado.

Um algoritmo deve contemplar 3 exigências:

1-As instruções devem ser claras e não devem conter ambiguidades, nem qualquer coisa que impeça sua execução pelo processador;

2-Não pode haver dubiedade em relação à próxima ação a ser realizada após a execução de uma determinada ação;

3-Todas as instruções devem ser executadas num tempo finito;

Existem diversas formas de representação de algoritmos, mas não há um consenso com relação à melhor delas. O critério usado para classificar hierarquicamente essas formas está diretamente ligado ao nível de detalhe ou, inversamente, ao grau de abstração oferecido

Algumas formas de representação de algoritmos tratam os problemas apenas em nível lógico, abstraindo-se de detalhes de implementação, muitas vezes, relacionados com alguma linguagem de programação específica. Por outro lado, existem formas de representação de algoritmos que possuem uma maior riqueza de detalhes e, muitas vezes, acabam por obscurecer as ideias principais do algoritmo, dificultando seu entendimento.

Dentre as formas de representação de algoritmos mais conhecidas podemos citar:

Descrição Narrativa:

-Ler 2 notas;

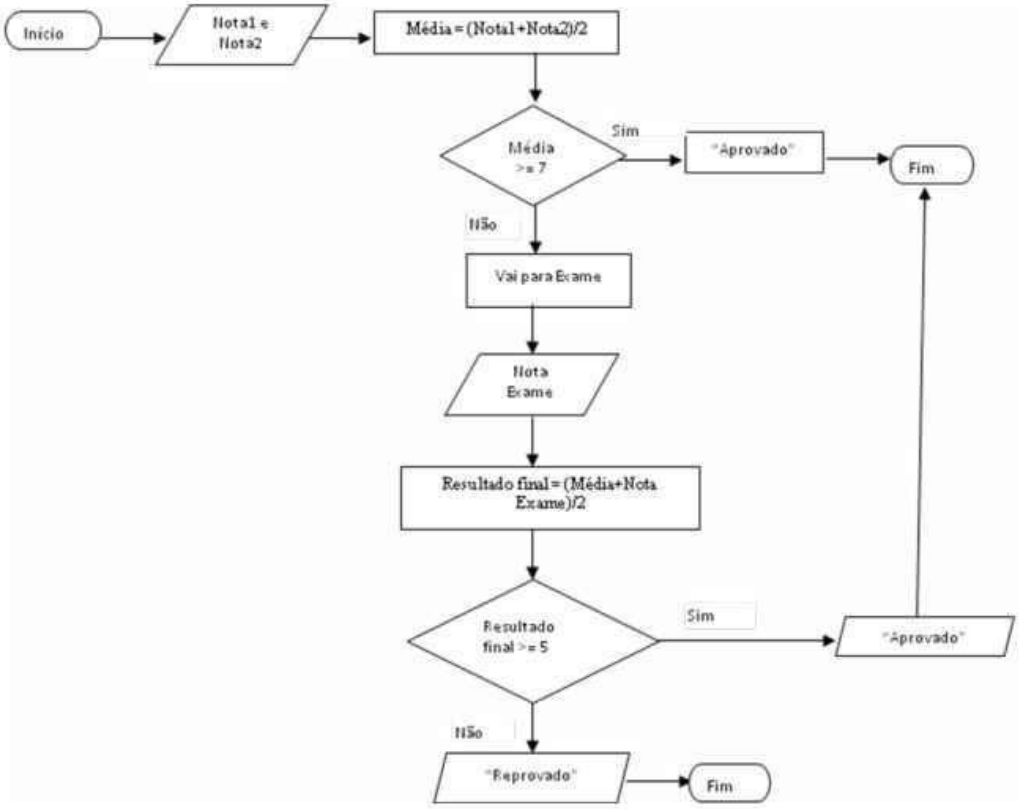
-Calcular a média entre elas;

-Imprimir aprovado se média for maior ou igual a sete;

-Se não resultar essa média, fazer exame;

-Se média entre Média final e exame for maior ou igual a 5, colocar aprovado; se não, colocar reprovado;

Fluxograma Convencional:



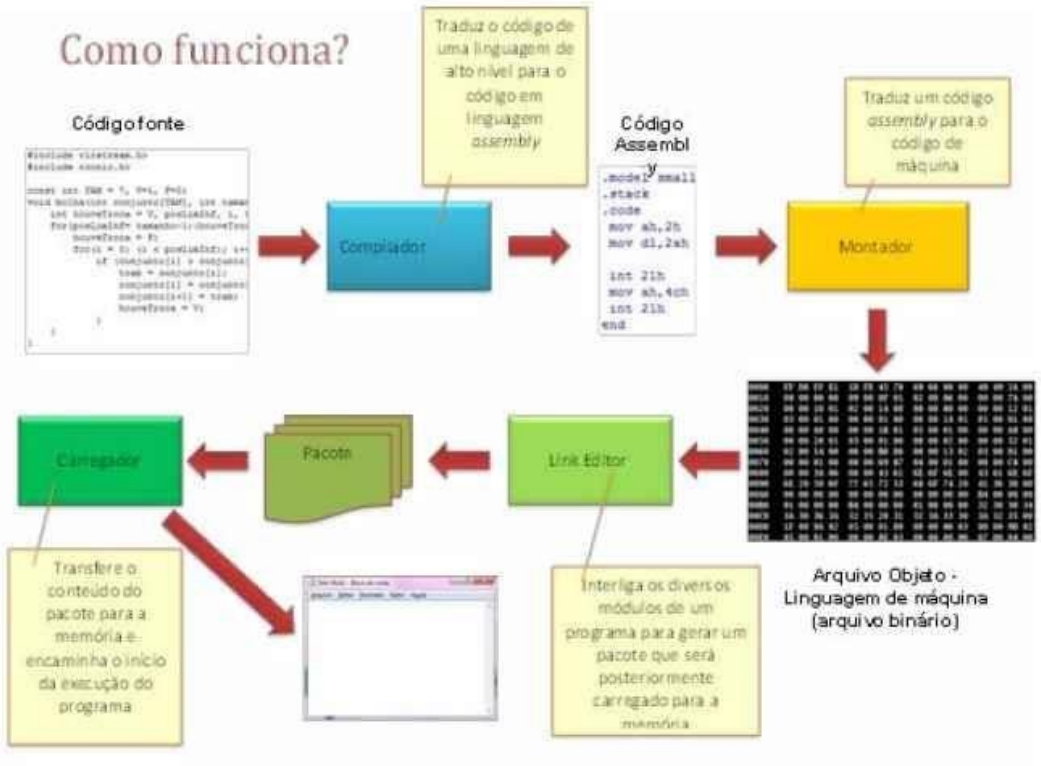
Pseudocódigo, também conhecido como Português Estruturado ou Portugol:

```
início
    real media, nota1, nota2, exame, final;
    escreva "Digite a 1ª nota: ";
leia nota1;
escreva "Digite a 2ª nota: ";
leia nota2;
media ¬ (nota1 + nota2)/2;
    se (media >= 7)
        escreva "Aprovado";
    senão
        escreva "Digite a nota de exame";
        leia exame;
        final ¬ (media + exame)/2;
        se (final >= 5)
            escreva "Aprovado";
        senão
            escreva "Reprovado";
        fim se
    fim se
fim
```

Usando uma Linguagem de Programação Java:

```
import javax.swing.*;
public class Algoritmo {
  public static void main(String args[]) {
    float media, nota1, nota2, exame, final;
    nota1 = Float.parseFloat(JOptionPane.showInputDialog("Digite a 1ª nota:"));
    nota2 = Float.parseFloat(JOptionPane.showInputDialog("Digite a 2ª nota:"));
    media = (nota1 + nota2)/2;
      if (media >= 7)
        JOptionPane.showMessageDialog(null, "Aprovado");
    else {
      exame = Float.parseFloat(JOptionPane.showInputDialog(""Digite a nota de exame"));
      final = (media + exame)/2;
        if (final >= 5)
        JOptionPane.showMessageDialog(null, "Aprovado");
      else
        JOptionPane.showMessageDialog(null, "Reprovado");
    }
  }
}
```

Compilação:

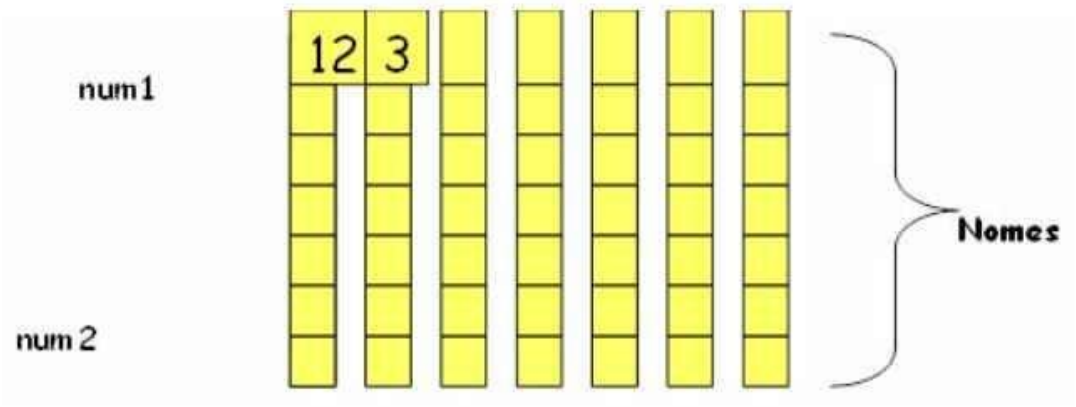


Variáveis:

Anteriormente admitimos que o processador era capaz de associar cadeias de caracteres a valores numéricos (Código ASC II). Foi dito, também, que as informações devem ser armazenadas em memória para que o processador seja capaz de capturar as informações e fazer operações sobre eles.

Para que a memória possa armazenar dados, ela é dividida em partes, chamadas posições de memória. O sistema operacional que gerencia o sistema de computação pode acessar cada umas dessas posições de memória para armazenar os dados. Para que isso seja possível, cada uma delas está associada a um endereço em bytes.

Em programação, uma variável é uma posição de memória reservada pelo seu programa cujo conteúdo pode ser modificado durante a execução de um programa (em tempo de execução), devendo ser associado sempre a um identificador e um tipo de dado.



Identificadores:

O identificador é uma sequência de letras, dígitos e caractere underline (_) escolhidos pelo programador e será utilizado no programa para se fazer referência àquela variável. Como um programa deve ser legível por outros programadores, o nome do identificador deve ter relação com o conteúdo que será armazenado na variável. Por exemplo, para uma variável que armazenará a idade de uma pessoa, o identificador poderá se chamar “idade” e não “x”.

Existem algumas regras para a criação de identificadores, ou seja, o identificador:

1-Não poderá começar com números, que são permitidos apenas após letras ou o underline (_);

2-Não poderá conter espaços em branco;

3-Não poderá conter caracteres especiais como ç, á, õ, #, etc.;

4-Não poderá ser uma palavra reservada da linguagem.

Exemplos de identificadores válidos:

-minha_variavel;

-sexo;

-aluno22;

-r2d2c3po;

-jymm1_neutr0n_afr1can0;

Exemplo de identificares inválidos:

-meu nome; (espaço)

-2r3poc; (começa com número)

-opção;(caracteres especiais – acentos)

-void; (palavra reservada do Java) **Tipos**

de Dados:

Um tipo de dado associado a uma variável é um conjunto de elementos que podem ser armazenados nela. Ele diz qual o tipo de informação que será armazenada naquela área de memória. Utilizaremos 4 tipos de dados distintos:

1-Inteiro: para armazenar valores inteiros, ou seja, que não possuem casas decimais. Nesse caso, não importa se o número é positivo ou negativo e nem há limites. O limite é apenas imposto quando escolhemos uma linguagem programação;

2-Real: para armazenar valores reais, ou seja, que podem possuir ponto flutuante (casas decimais);

3-String: para armazenar cadeias de caracteres como palavras e/ou frases;

4-Lógico: só consegue armazenar dois estados, sendo verdadeiro ou falso;

Exemplo de declaração de variável:

-inteiro idade;

-inteiro numero_filhos, quantidade_patas;

-real salario;

-real altura, peso;

-string nome;

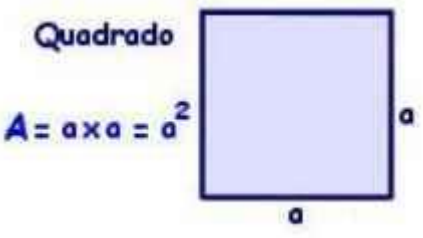
-logico pagou;

Prova 2: Estruturas Condicionais para o Desenvolvimento de Algoritmos.

Expressões:

O conceito de expressões está associado a fórmulas matemáticas, onde um conjunto de variáveis e constantes numéricas se relacionam através de operadores matemáticos, formando uma expressão e resultando em um valor.

Por exemplo, a fórmula da área do quadrado é A = lado x lado. Essa fórmula utiliza duas variáveis: A e lado, onde lado terá a medida do quadrado e A que resultará na operação de multiplicação dos lados. Em informática, uma expressão é a combinação de variáveis, constantes e operadores e que quando verificadas resultam um valor.



Operadores:

Os operadores são utilizados nas expressões, por exemplo, 3 + 2, os números 3 e 2 são relacionados por um operador representado pelo sinal + que significa adição. Os operadores se classificam em aritméticos, lógicos e literais, essa divisão depende do tipo de expressão que os mesmo

serão inseridos, existe ainda um tipo de operador que é o relacional, onde se compara informações e o resultado é um valor lógico.

Tipos de Expressões:

Expressões aritméticas resultam um tipo de dado numérico inteiro ou real, somente são permitidos operadores aritméticos e variáveis numéricas nessas expressões. Vejamos, na tabela a seguir, as operações aritméticas:

Operadores aritméticos	
Operador	**Operação**
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
**	Exponenciação
%	Resto

Como na matemática, em computação existe uma ordem de execução dos operadores, por exemplo, primeiro são resolvidas as expressões com os símbolos * e /, em seguida + e -.

Expressões Lógicas:

Expressões lógicas são aquelas que retornam ou resulta um valor lógico e só existem dois tipos de dados, verdadeiros ou falsos. Na expressão lógica, utilizamos um operador lógico. A tabela a seguir ilustra a utilização e o resultado da expressão:

TABELA VERDADE					
A	**B**	**.NÃO. A**	**.NÃO. B**	**A .OU. B**	**A .E. B**
F	F	V	V	F	F
F	V	V	F	V	F
V	F	F	V	V	F
V	V	F	F	V	V

O operador lógico NÃO sempre inverterá o valor do seu operando. O operador OU resultará verdadeiro, quando um dos seus operando for Verdadeiro, já o operando E só resultará verdade quando todos os seus operando forem Verdadeiros.

Operadores Relacionais:

Ainda existe outro tipo de operadores: o relacional, que pode ser utilizado entre as operações lógicas.

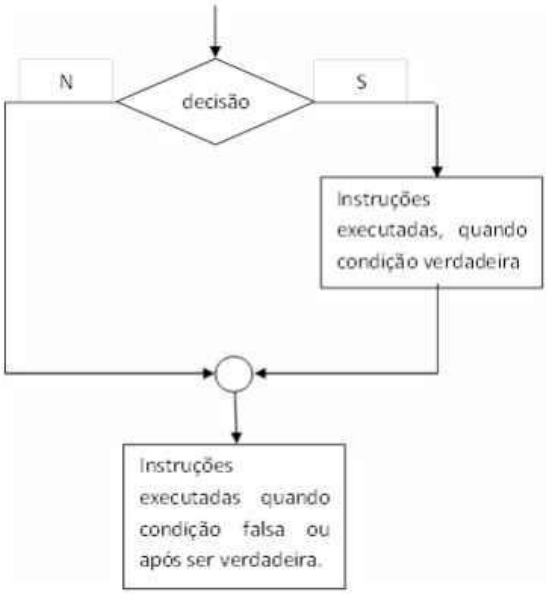
Operador Relaciona	
Operador	**Operação**
=	Igual
<>	Diferente
<	Menor
<=	Menor ou Igual
>	Maior
>=	Maior ou Igual

Desvio Condicional Simples:

Uma decisão simples é composta pela instrução se... então...fim, se a condição estabelecida fora verdadeira, serão executadas todas as instruções definidas entre o se... então e, depois, serão executadas todas as instruções que existirem após o fim.

Diagrama de Blocos:

Se você observar as letras S e N, além das linhas com seta indicando a direção do processamento, a letra S representa o sim, isto significa que ela será executada quando a operação lógica verdadeira for atribuía a ela. O N de não será executada quando a operação lógica for falsa. O símbolo do losângulo é representado pela decisão e deve ser utilizado quando há a necessidade de se executar uma decisão dentro do programa. Nós obteremos uma resposta da decisão sempre com base em uma pergunta, com essa resposta, atribuiremos o resultado verdadeiro ou falso como exibido no diagrama a seguir.



Em alguns casos, programadores utilizam as letras V para representar o Sim e F para representar o Não, a ordem também vai depender da lógica do programador, alguns utilizam o lado do V (Sim) a esquerda e F (Não) a direita.

Traduzindo o fluxograma para o português estruturado, ficaria da seguinte forma:

se (<condição) então
    <instruções para condição verdadeira>
Fim

<instruções para condição falsa ou após ser verdadeira>

Como um exemplo, considere o seguinte problema:

Ler dois valores numéricos, efetuar a adição e apresentar o seu resultado caso o valor somado seja maior que 10. Veja como ficaria o diagrama de blocos e, também, o português estruturado.

Algoritmo:

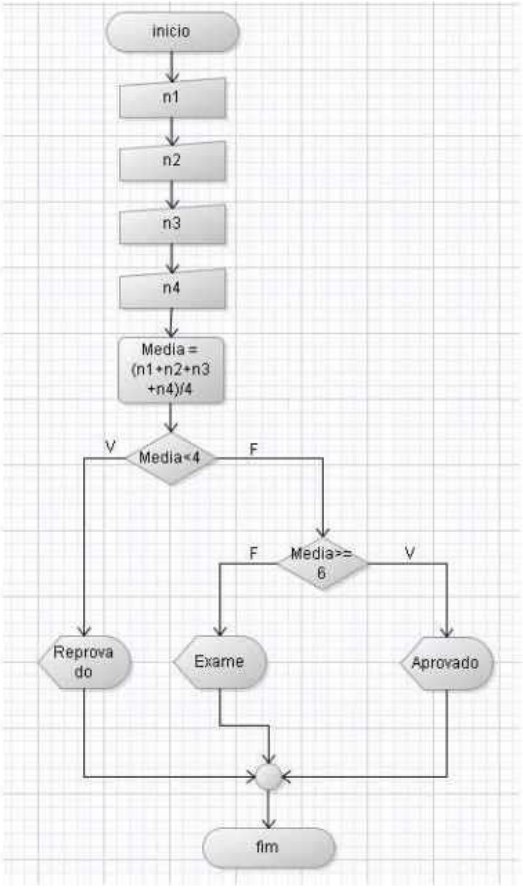
-Criar duas variáveis, chamaremos de A e B;
-Efetuar a soma dos valores e atribuir a variável X;
-Apresentar o valor da soma na variável X, caso o valor atribuído seja maior que 10.

```
Programa SOMA_NUMEROS
início
Var
    X: inteiro
    A: inteiro
    B: inteiro
Leia A
Leia B
X = A + B
Se ( X > 10 ) então
    Escreva X
Fim_Se
Fim
```

Primeiro, criamos as variáveis que serão utilizadas no programa, depois definimos o tipo das variáveis, após a definição é solicitada a entrada de dados para atribuir valores as variáveis. Depois dos valores atribuídos, realizamos a operação de soma atribuindo o resultado a variável X. Nesse momento, é questionado no programa se o valor na variável X é maior do que 10, se a resposta for verdadeira é exibido o valor da variável X.

Desvio Condicional Composta:

A estrutura condicional composta é similar com a estrutura condicional simples. Sempre um comando será executado independente da condição, ou seja, caso a condição seja verdadeira, os comandos da condição serão executados ou os comandos da condição falsa serão executados.



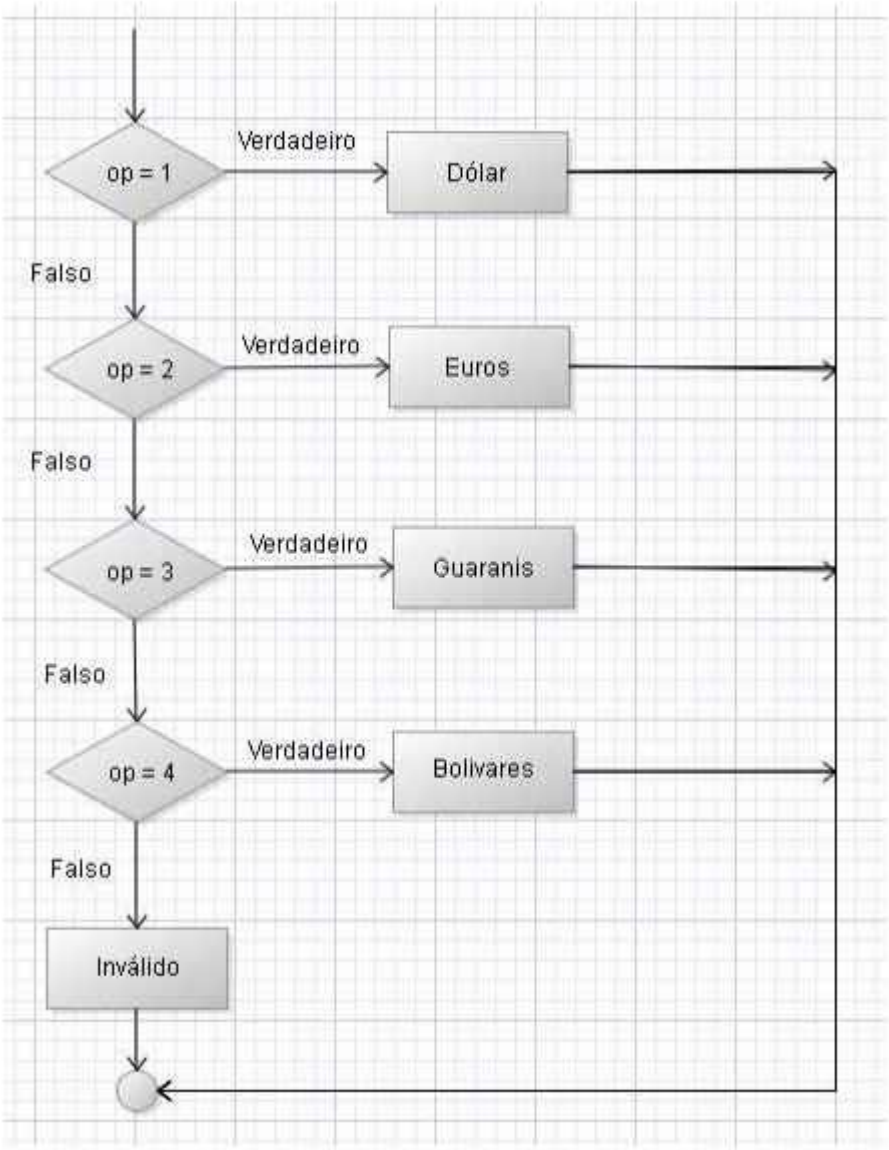

```
Programa MEDIA
início
Var
    N1: real
    N2: real
    N3: real
    N4: real
    MEDIA: real
Leia N1
Leia N2
Leia N3
Leia N4

MEDIA = (N1+N2+N3+N4)/4
Se ( MEDIA < 4 ) então
    Escreva "REPROVADO"
Else
    Se (MEDIA>=6) então
        Escreva "APROVADO"
    Else
        Escreva "EXAME"
    Fim_Se
Fim_Se
Fim
```

Estrutura de Decisão Switch Case:

O comando switch case pode ser útil quando precisamos fazer diversas verificações e executar diferentes comandos de acordo com o valor de alguma variável. Essa funcionalidade é muito similar à estrutura condicional composta, a maior diferença está no tipo da variável que está sendo avaliada, o tipo da variável em uma estrutura switch só pode ser char ou inteiro.

Por exemplo, no fluxograma abaixo, se o usuário selecionar a opção 1 via teclado, o sistema é direcionado para calcular o dólar, caso contrário, se a opção for igual a 2, o sistema é direcionado para calcular o euro e assim por diante.



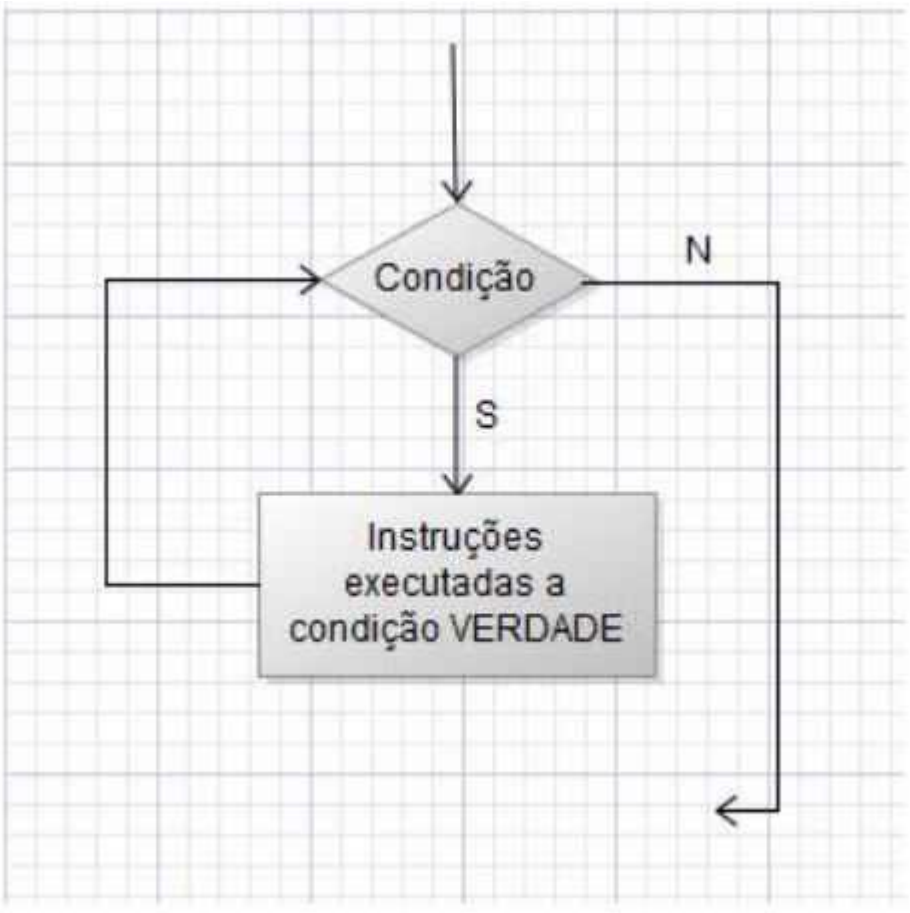
Prova 3: Estruturas de Repetição para o Desenvolvimento de Algoritmos.

Estruturas de Repetição:

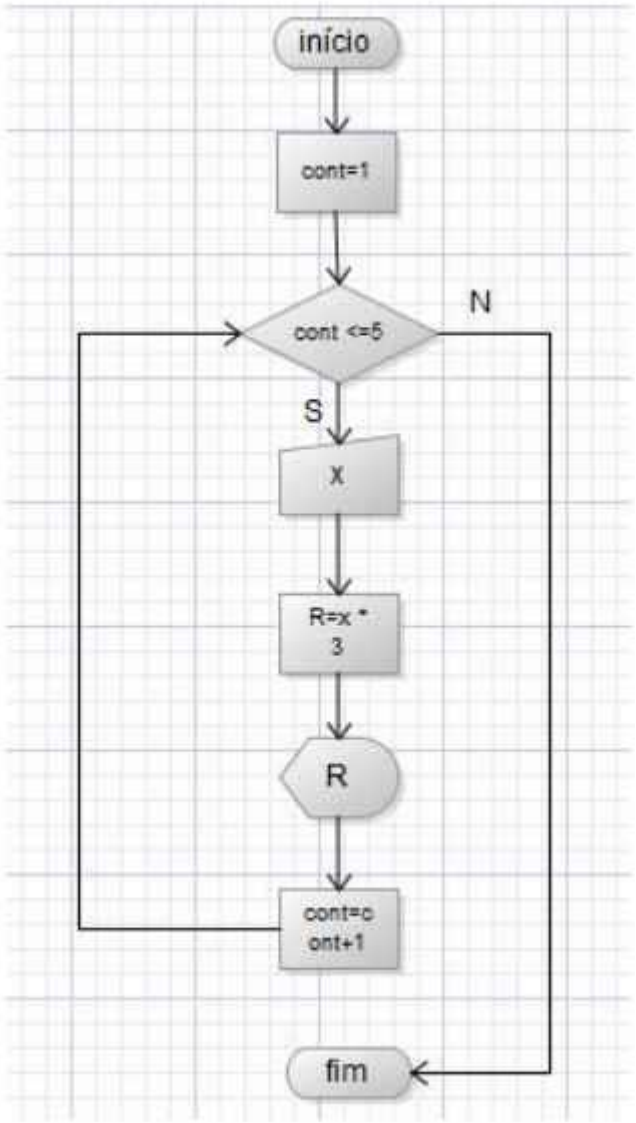
Em algumas circunstâncias, é necessária a repetição de um trecho do programa algumas quantidades de vezes. Sendo assim, deverá ser criado um laço que repita o processamento n vezes quantas forem necessárias.

Repetição com teste no início do laço:

Esta estrutura executa um teste no início da repetição. Caso o retorno do teste seja VERDADEIRO, o trecho de instruções subordinado a ele é executado. A estrutura enquanto... faça... fim_enquanto tem o seu funcionamento controlado por decisão. É muito parecido com a estrutura de decisão vista na unidade anterior.



Para representarmos o laço de repetição acima, iremos resolver o problema que segue. Veja o digrama de bloco abaixo e o seu respectivo português estruturado.




```
Português estruturado:
Programa ENQUANTO
Var
      cont,x,r: inteiro
inicio
      cont = 1
      enquanto (cont<=5) faça
            leia x
            r = x * 3
            escreva r
      fim_enquanto
fim
```

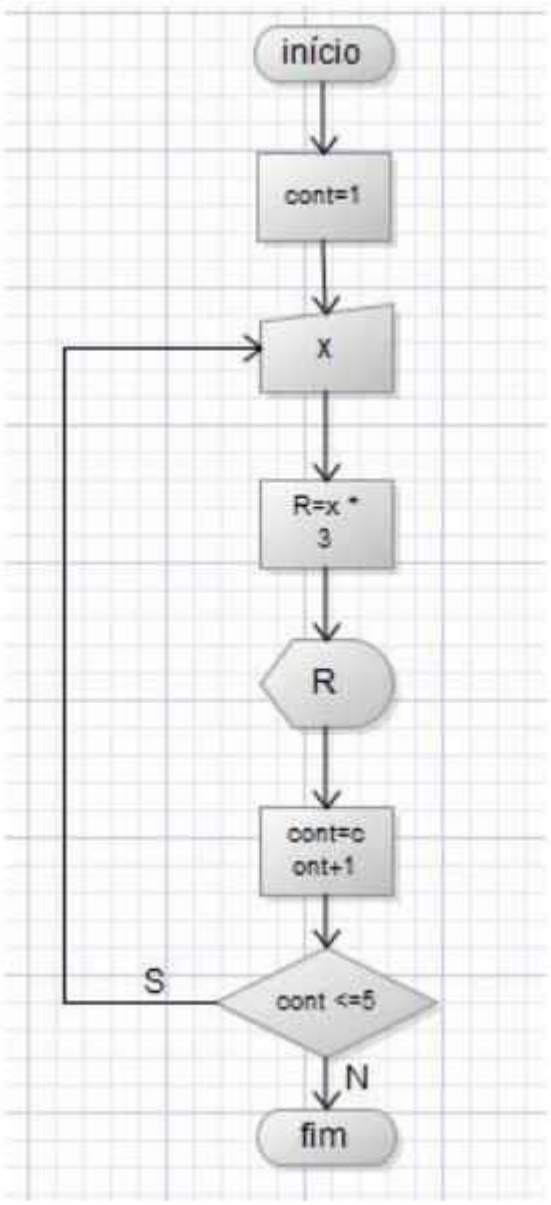
Algoritmo:

1- Variável cont = 1;
2- Enquanto cont for menor ou igual a 5;
3- Ler a variável X;
4- Efetuar a multiplicação de X por 3;
5- Exibir o novo valor da variável R;
6- Quanto o valor de cont for maior do que 5, encerrar o programa.

Repetição com teste no fim do laço:

Esta estrutura executa um teste no fim da repetição e é chamada de repita. A estrutura repita ... até_que também é controlada por uma decisão, mas, neste caso, o conjunto de instrução é executado, no mínimo, uma vez; diferente da repetição enquanto, que somente executa o conjunto de instruções caso a sentença seja verdadeira. Portanto a estrutura de repetição chamada repita tem seu funcionamento invertido, pois sempre executa uma vez as instruções até que a condição seja verdadeira.

Para representarmos o laço de repetição acima, iremos resolver o seguinte problema. Veja o digrama de bloco abaixo e o seu respectivo português estruturado.



Português estruturado:

```
Programa REPITA
Var
cont,x,r: inteiro
inicio
    cont = 1
    repita
            leia x
            r = x * 3
            escreva r
            cont = cont + 1
    até_que (cont >5 )
fim
```

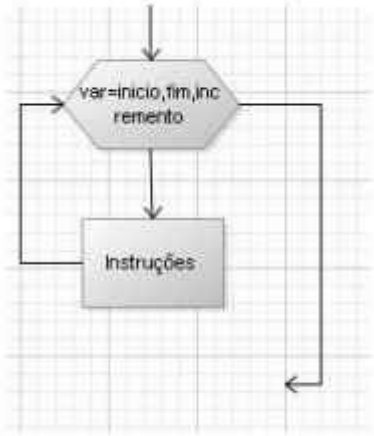
Algoritmo:

1- Variável cont = 1;
2- Ler a variável X;
3- Efetuar a multiplicação de X por 3;
4- Exibir o novo valor da variável R;
5- Atribuir o valor atual de cont + 1;
6- Quanto o valor de cont for maior do que 5, encerrar o programa.

Repetição com variável de controle:

Nos exemplos anteriores vimos duas formas de desenvolver laços de repetição. Uma delas é utilizar o enquanto e a outra, o repita. As repetições que possuem um quantidade de vezes determinada poderão ser elaboradas por meio da estrutura de laços para ... de...até...passo... faça...fim_para.

No diagrama de blocos, será utilizada uma variável a ser controlada com a atribuição de valores de início, fim e incremento, separados por vírgula.



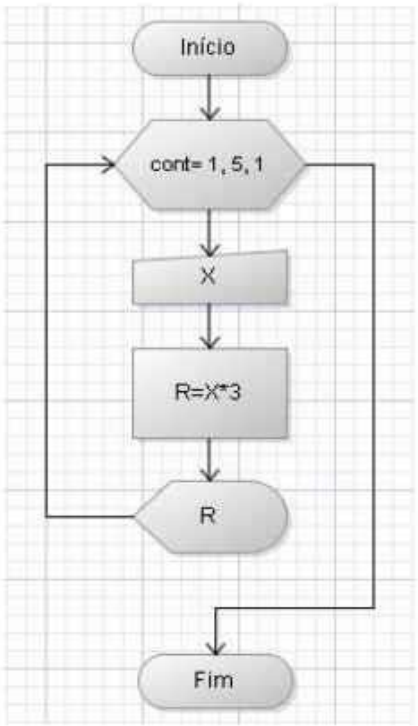
Português estruturado

para variável **de** início **até** fim **passo** incremento **faça**

instruções

fim_para

Para fixarmos a ideia, vamos nos manter nos exemplos vistos anteriormente. Solicitar a leitura de um valor, multiplicar esse valor por 3, exibir o resultado da multiplicação. Esse processo deve ser executado 5 vezes.



Português estruturado:

Programa REPITA

Var

cont,x,r: inteiro

inicio

para cont **de** 1 **até** 5 **passo** 1 **faça**

leia x

r = x * 3

escreva r

fim_para

fim

Algoritmo:

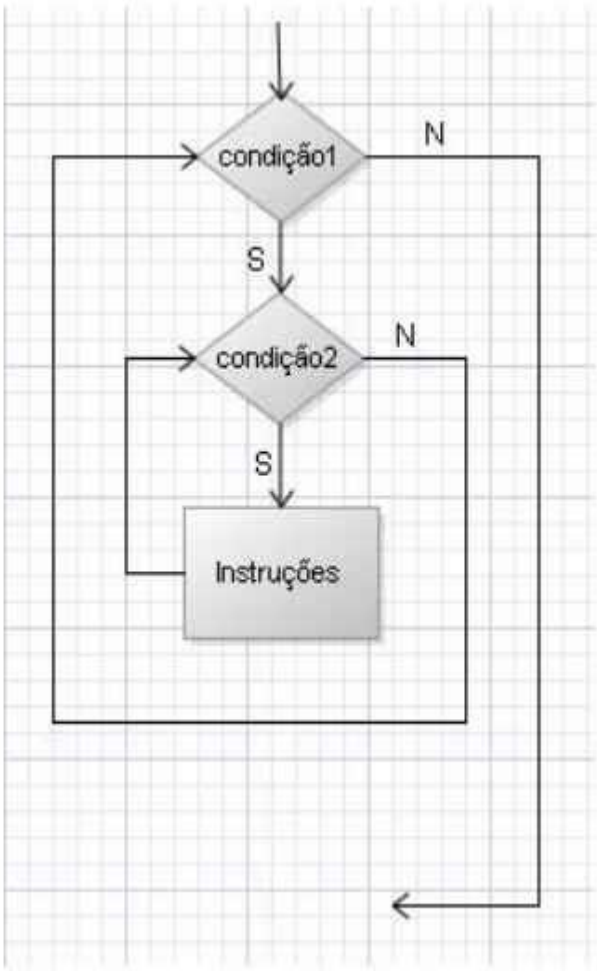
1- Definir um contador, iniciando em 1 e terminando em 5;
2- Ler a variável X;
3- Efetuar a multiplicação de X por 3;
4- Exibir o novo valor da variável R;
5- Quando o valor de cont for maior do que 5, encerrar o programa.

Será executada a repetição entre as instruções para e fim_para. A variável CONT é a variável de controle, que será inicializada com 1, incrementada de 1, por meio da instrução passo até que o valor da variável cont seja igual a 5.

Estrutura de Controles Encadeados:

O usuário precisa conhecer os comandos de entrada, de processamento e de saída; deve conhecer a estrutura de decisão e de repetições. Conhecendo essas técnicas, saberá utilizar em conjunto. Veja os exemplos:

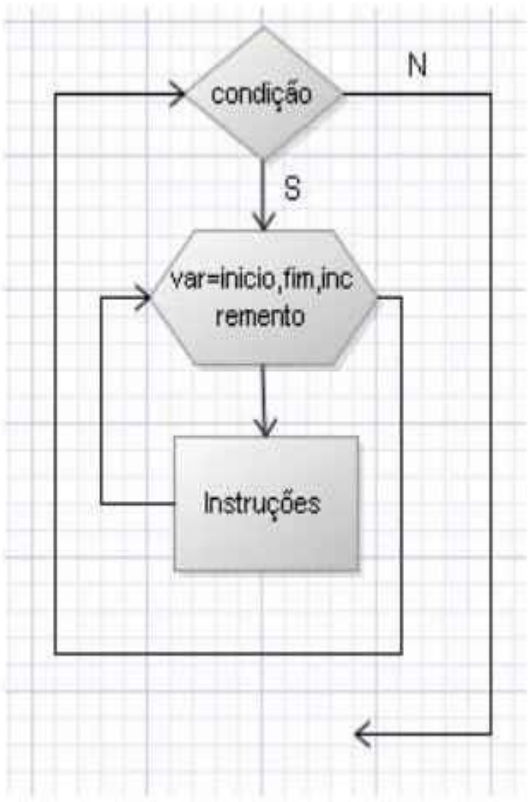
Encadeamento de Estrutura Enquanto com Enquanto:



Português estruturado

```
Enquanto (condição1) faça
        Enquanto (codição2) faça
                Instruções
        Fim_enquanto
Fim_enquanto
```

Encadeamento de Estrutura Enquanto com Para:



Português estruturado

Enquanto (condição) faça

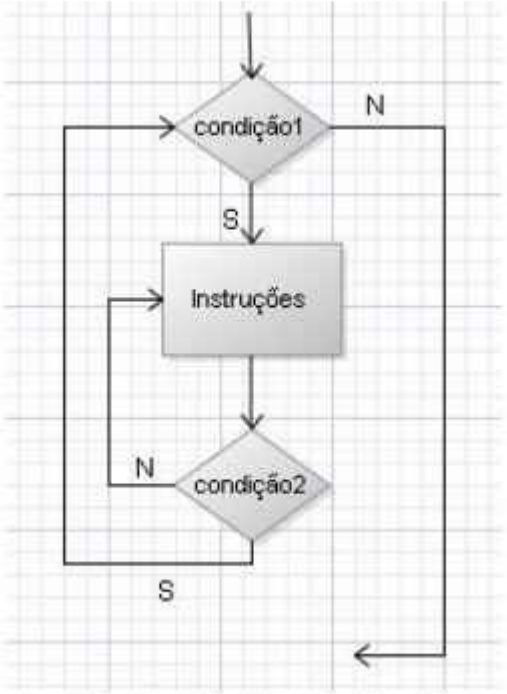
para variável de início até fim passo incremento faça

instruções

fim_para

Fim_enquanto

Encadeamento de Estrutura Enquanto com Repita:



Português estruturado

Enquanto (condição1) faça

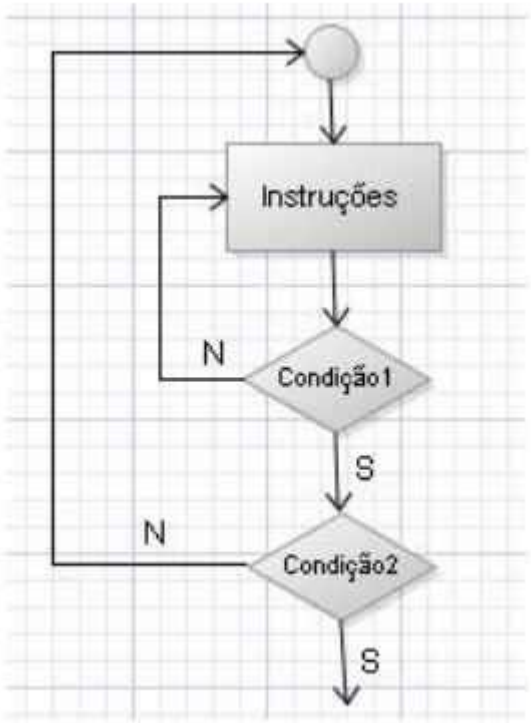
Repita

Instruções

Até_que(condição2)

Fim_enquanto

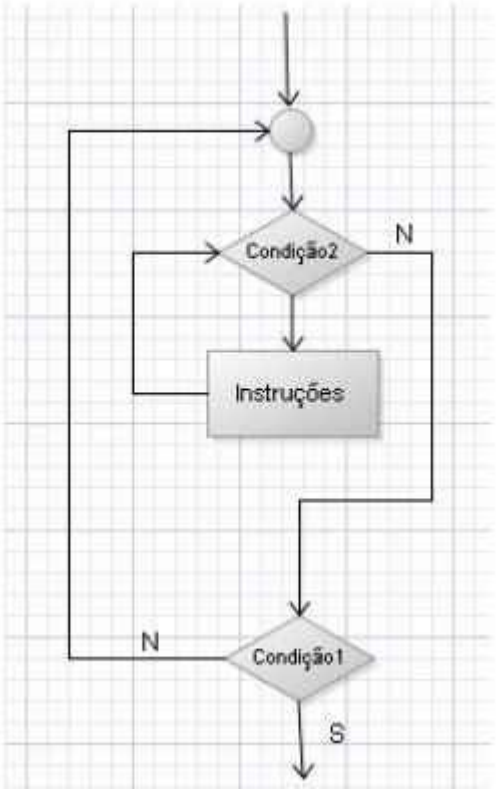
Encadeamento de Estrutura Repita com Repita:



Português estruturado

```
Repita
  Repita
        Instruções
  Até_que(condição1)
Até_que(condição2)
```

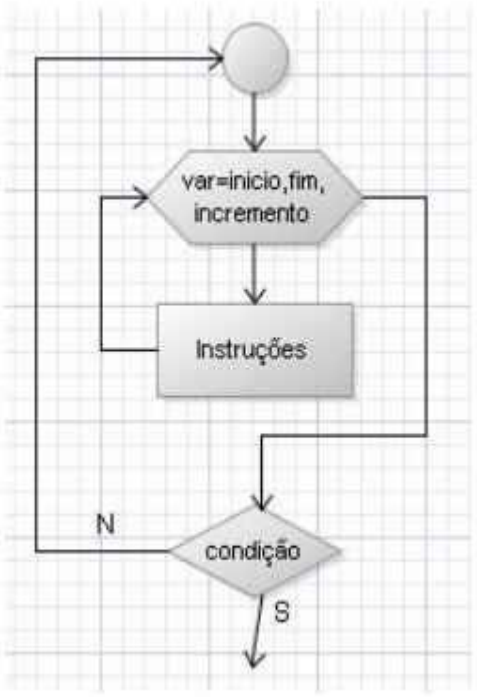
Encadeamento de Estrutura Repita com Enquanto:



Português estruturado

```
Repita
    Enquanto (condição2) faça
          Instruções
    Fim_enquanto
    Até_que(condição1)
Até_que(condição1)
```

Encadeamento de Estrutura Repita com Para:



Português estruturado

Repita

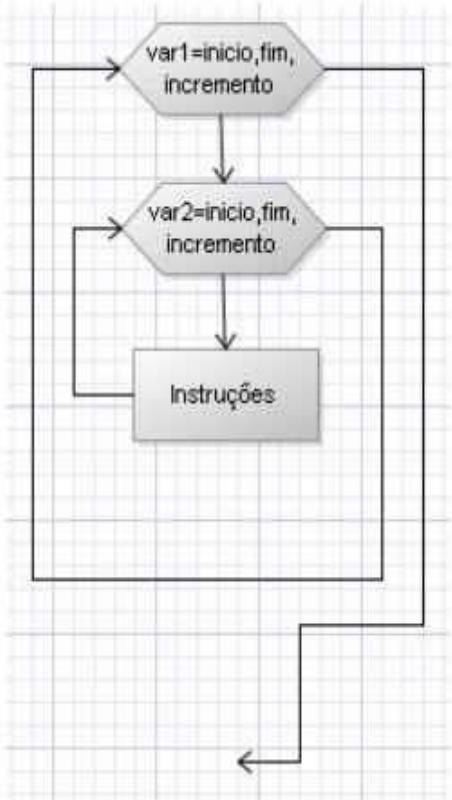
para **var** de **início** até **fim** passo **incremento** faça

instruções

fim_para

Até_que(condição)

Encadeamento de Estrutura Para com Para:



Português estruturado

para **var1** de início até **fim** passo **incremento** faça

para **var2** de **início** até **fim** passo **incremento** faça

instruções

fim_para

fim_para

Prova 4: Funções e Recursividade no Desenvolvimento de Algoritmos.

Introdução:

A complexidade dos algoritmos está intimamente ligada à da aplicação a que se destinam. Em geral, problemas complicados exigem algoritmos extensos para sua solução. Sempre é possível dividir problemas grandes e complicados em problemas menores e de solução mais simples. Assim, pode-se solucionar cada um destes pequenos problemas separadamente, criando algoritmos para tal (subalgoritmos). Um subalgoritmo é um nome dado a um trecho de um algoritmo mais complexo e que, em geral, encerra em si próprio um pedaço da solução de um problema maior a qual o algoritmo que a ele está subordinado.

Em resumo, os subalgoritmos são importantes na:

- Subdivisão de algoritmos complexos, facilitando o seu entendimento;
- Estruturação de algoritmos, facilitando principalmente a detecção de erros e a documentação de sistemas;
- Modularização de sistemas, que facilita a manutenção de softwares e a reutilização de subalgoritmos já implementados.

A ideia da reutilização de software tem sido adotada por muitos grupos de desenvolvimento de sistemas de computador, devido à economia de tempo e trabalho que proporcionam. Seu principal objetivo é solucionar uma série de tarefas e possibilitar o acréscimo de novos algoritmos. A este conjunto dá-se o nome de biblioteca. No desenvolvimento de novos sistemas, procura-se ao máximo basear sua concepção em subalgoritmos já existentes na biblioteca, de modo que a quantidade de software realmente novo que deve ser desenvolvido é minimizada. Muitas vezes os subalgoritmos podem ser úteis para encerrar em si certa sequência de comandos que é repetida várias vezes num algoritmo. Nestes casos, os subalgoritmos proporcionam uma diminuição do tamanho de algoritmos maiores.

Mecanismo de Funcionamento:

Um algoritmo completo é dividido num algoritmo principal e diverso subalgoritmos (tantos quantos forem necessários). O algoritmo principal é aquele por onde a execução do programa sempre se inicia. Este pode eventualmente invocar os demais subalgoritmos ou subprogramas.

As definições dos subalgoritmos estão sempre colocadas no trecho após a definição das variáveis globais e antes do corpo do algoritmo principal, veja o modelo abaixo:

```
Algoritmo  <nome do algoritmo>
    Var  <definição das variáveis globais>
    <definições dos subalgoritmos>
Inicio
    <corpo do algoritmo principal>
Fim.
```

Durante a execução do algoritmo principal, quando se encontra um comando de invocação de um subalgoritmo, a execução do mesmo é interrompida. A seguir, passa-se à execução dos comandos do corpo do subalgoritmo. Ao seu término, retoma-se a execução do algoritmo que o chamou (no caso, o algoritmo principal) no ponto onde foi interrompida (comando de chamada do subalgoritmo) e prossegue-se pela instrução imediatamente seguinte. Note, também, que é possível que um subalgoritmo chame outro através do mesmo mecanismo.

Definição de Subalgoritmos:

A definição de um subalgoritmo consta de:

- Um cabeçalho, onde estão definidos o nome e o tipo do subalgoritmo, bem como os seus parâmetros e variáveis locais;
- Um corpo, onde se encontram as instruções (comandos) do subalgoritmo;
- O nome de um subalgoritmo é o nome simbólico pelo qual ele é chamado por outro algoritmo;
- O corpo do subalgoritmo contém as instruções que são executadas cada vez que ele é invocado;
- Variáveis locais são aquelas definidas dentro do próprio subalgoritmo e só podem ser utilizadas pelo mesmo.

Parâmetros são canais por onde os dados são transferidos pelo algoritmo chamador a um subalgoritmo, e vice-versa. Para que possa iniciar a execução das instruções em seu corpo, um subalgoritmo às vezes precisa receber dados do algoritmo que o chamou e, ao terminar sua tarefa, o subalgoritmo deve fornecer ao algoritmo chamador os resultados da mesma. Esta comunicação bidirecional pode ser feita de dois modos que serão estudados mais à frente: por meio de variáveis globais ou por meio da passagem de parâmetros.

O tipo de um subalgoritmo é definido em função do número de valores que o subalgoritmo retorna ao algoritmo que o chamou. Segundo esta classificação, os algoritmos podem ser de dois tipos:

- funções, que retornam um, e somente um, valor ao algoritmo chamador;
- procedimentos, que retornam zero (nenhum) ou mais valores ao algoritmo chamador.

Na realidade, a tarefa desempenhada por um subalgoritmo do tipo função pode perfeitamente ser feita por outro do tipo procedimento (o primeiro é um caso particular deste). Esta diferenciação é feita por razões históricas, ou, então, pelo grande número de subalgoritmos que se encaixam na categoria de funções.

Funções:

O conceito de Função é originário da idéia de função matemática (por exemplo, raiz quadrada, seno, cosseno, tangente, logaritmo, entre outras), onde um valor é calculado a partir de outro(s) fornecido(s) à função.

A sintaxe da definição de uma função é dada a seguir:

```
Função <nome> ( <parâmetros> ) <tipo_de_dado>
    Var <variáveis locais>
Início
    <comando composto>
Fim
```

Temos que:

- <nome> é o nome simbólico pelo qual a função é invocada por outros algoritmos;
- <parâmetros>são os parâmetros da função;
- <tipo de dado> é o tipo de dado da informação retornado pela função ao algoritmo chamador;
-<variáveis locais> consiste na definição das variáveis locais à função. Sua forma é análoga à da definição de variáveis num algoritmo;
-<comando composto> é o conjunto de instruções do corpo da função.

Dentro de um algoritmo, o comando de invocação de um subalgoritmo do tipo função sempre aparece dentro de uma expressão do mesmo tipo que o do valor retornado pela função. A invocação de uma função é feita pelo simples aparecimento do nome da mesma, seguido pelos respectivos parâmetros entre parênteses, dentro de uma expressão. A função é executada e, ao seu término, o trecho do comando que a invocou é substituído pelo valor retornado pela mesma dentro da expressão em que se encontra, e a avaliação desta prossegue normalmente.

Dentro de uma função, e somente neste caso, o comando Retorne é usado para retornar o valor calculado pela mesma. Ao encontrar este comando, a expressão entre parênteses é avaliada, a execução da função é terminada neste ponto e o valor da expressão é retornado ao algoritmo chamador. Vale lembrar que uma expressão pode ser uma simples constante, uma variável ou uma combinação das duas por meio de operadores. Esta expressão deve ser do mesmo tipo que o valor retornado pela função.

O algoritmo a seguir é um exemplo do emprego de função para calcular o valor de um número elevado ao quadrado.

```
Algoritmo Exemplo_de_função
Var X, Y : real

Função Quad(real w) real
Var
    real Z
Inicio
    Z = w * w
    Retorna Z
Fim

Inicio
    Escreva "Digite um número"
    Leia X
    Y = Quad(X)
    Escrever X, " elevado ao quadrado é = ", Y
Fim.
```

Do exemplo anterior é importante notar que:

- A função Quad toma W como parâmetro do tipo real, retorna um valor do tipo real e possui Z como uma variável local real;
- O comando de invocação da função Quad aparece no meio de uma expressão, no comando de atribuição dentro do algoritmo principal.

Procedimentos:

Um procedimento é um subalgoritmo que retorna zero (nenhum) ou mais valores ao (sub) algoritmo chamador. Estes valores são sempre retornados por meio dos parâmetros ou de variáveis globais, mas nunca explicitamente, como no caso de funções. Portanto, a chamada de um procedimento nunca surge no meio de expressões, como no caso de funções. Pelo contrário, a chamada de procedimentos só é feita em comandos isolados dentro de um algoritmo, como as instruções de entrada (Leia) e saída (Escreva) de dados.

A sintaxe da definição de um procedimento é:

```
Procedimento  <nome>  ( <parâmetros> )
    Var  <variáveis locais>
Inicio
    <comando composto>
Fim.
```

Temos que:

-<nome> é o nome simbólico pelo qual o procedimento é invocado por outros algoritmos;
-<parâmetros> são os parâmetros do procedimento;
-<variáveis locais> são as definições das variáveis locais ao procedimento. Sua forma é análoga à da definição de variáveis num algoritmo;
-<comando composto> é o conjunto de instruções do corpo do procedimento, que é executado toda vez que o mesmo é invocado.

O exemplo a seguir é um exemplo simples, onde um procedimento é usado para escrever o valor das componentes de um vetor.

```
Algoritmo Exemplo_procedimento

Var
     matriz[1..10] de real

Procedimento ESC_VETOR()
Var
     Inteiro i
Inicio
     para i de 1 até 10 faça
          Escreva vet[i]
     fim_para
Fim

Procedimento LER_VETOR()
Var
     inteiro i
Inicio
     para i de 1 até 10 faça
          ler vet[i]
     fim_para
Inicio
     LER_VETOR()
     ESC_VETOR()
Fim
```

No exemplo é conveniente observar:

- A forma de definição dos procedimentos LER_VETOR( ) e ESC_VETOR( ), que não possuem nenhum parâmetro, e usam a variável local i para “varrer” os componentes do vetor, lendo e escrevendo um valor por vez;
- A forma de invocação dos procedimentos, por meio do seu nome, seguido de seus eventuais parâmetros (no caso, nenhum), num comando isolado dentro do algoritmo principal.

Variáveis Globais e Locais:

Variáveis globais são aquelas declaradas no início de um algoritmo. Estas variáveis são visíveis (isto é, podem ser usadas) no algoritmo principal e por todos os demais subalgoritmos. Variáveis locais são aquelas definidas dentro de um subalgoritmo e, portanto, somente visíveis (utilizáveis) dentro do mesmo. Outros subalgoritmos, ou mesmo o algoritmo principal, não podem utilizá-las.

No exemplo a seguir são aplicados estes conceitos.

```
Algoritmo Exemplo_variáveis_locais_e_globais
Var real X
Inteiro I

Função FUNC()real
Var
    X : matriz[1..5] de inteiro
    Y caracter[1...10] de inteiro
Inicio
...
Fim

Procedimento PROC
Var  lógico Y
Inicio
...
X = 4 * X
I = I + 1
...
Fim

Início
...
X = 3.5
...
Fim.
```

É importante notar no exemplo anterior que:

- As variáveis X e I são globais e visíveis a todos os subalgoritmos, à exceção da função FUNC, que redefine a variável X localmente;
- As variáveis X e Y locais ao procedimento FUNC não são visíveis ao algoritmo principal ou ao procedimento PROC. A redefinição local do nome simbólico X como uma matriz[5] de inteiro sobrepõe (somente dentro da função FUNC) a definição global de X como uma variável do tipo real;

- A variável Y dentro do procedimento PROC, que é diferente daquela definida dentro da função FUNC, é invisível fora deste procedimento;
- A instrução X = 8.5 no algoritmo principal, bem como as instruções X = 4 * X e I = I + 1 dentro do procedimento PROC, atuam sobre as variáveis globais X e I.

Parâmetros:

Parâmetros são canais pelos quais se estabelece uma comunicação bidirecional entre um subalgoritmo e o algoritmo chamador (o algoritmo principal ou outro subalgoritmo). Dados são passados pelo algoritmo chamador ao subalgoritmo, ou retornados por este ao primeiro por meio de parâmetros.

Parâmetros formais são os nomes simbólicos introduzidos no cabeçalho de subalgoritmos, usados na definição dos parâmetros do mesmo. Dentro de um subalgoritmo trabalha-se com estes nomes da mesma forma como se trabalha com variáveis locais ou globais.

```
Função Média(real X, real Y) : real
Inicio
    Retorne (X + Y) / 2
Fim
```

No exemplo anterior, X e Y são parâmetros formais da função Média.

Parâmetros reais são aqueles que substituem os parâmetros formais quando da chamada de um subalgoritmo. Por exemplo, o trecho seguinte de um algoritmo invoca a função Média com os parâmetros reais 8 e 7 substituindo os parâmetros formais X e Y

Z := Média(8, 7)

Assim, os parâmetros formais são úteis somente na definição (formalização) do subalgoritmo, ao passo que os parâmetros reais substituem-nos a cada invocação do subalgoritmo. Note que os parâmetros reais podem ser diferentes a cada invocação de um subalgoritmo. Por fim, o uso do método é vantajoso devido ao seu estímulo à modularização de sistemas, que proporciona a criação de programas claros, fáceis de entender e, portanto, de manutenção mais barata.