PCS 3216 – Sistemas de Programação

Prof. João José Neto

16 – Linguagens de alto nível

Parte 1 – Conceitos e Generalidades

Introdução

- O foco principal da disciplina “Sistemas de Programação”, é nos programas de **software básico**, que compõem o conjunto de utilitários do sistema computacional, destinados a facilitar ao usuário a utilização da máquina e a construção de seus próprios aplicativos.
- Parte considerável da utilização dos sistemas computacionais envolve o manejo das notações empregadas na codificação dos programas a serem nele executados.
- Essas notações, ou **linguagens**, bem como os programas de sistema que as manipulam, tornam-se muito importantes nesse contexto, razão principal do estudo, ainda que superficial, nesta disciplina, das linguagens aí empregadas, bem como da sua análise e do seu processamento.

Conceitos

- Os principais conceitos e técnicas relacionados com o estudo das linguagens utilizadas nos sistemas de programação podem ser resumidos nos seguintes:
  - Linguagens de programação e seus paradigmas
  - Processadores de linguagens de programação
  - Ambientes e ferramentas de desenvolvimento

Linguagens

- **Linguagens** são notações com as quais o usuário pode dirigir ordens diretas ao computador, estabelecer tarefas a serem executadas, especificar condições a serem verificadas e ações correspondentes a serem efetuadas.
- Há vários **tipos** de linguagens, conforme sua utilização:
  - de **programação** (para codificar programas)
  - de **controle** (para o acionamento direto de funcionalidades)
  - de **script** (para ser interpretada diretamente pelo sistema)
  - de **interação** (para a condução de diálogos com o sistema)
  - de **uso específico** (para guiar programas aplicativos particulares)
- Na elaboração de sistemas de programação, destaca-se ainda a utilização de diversas **metalinguagens**, notações com as quais são feitas especificações formais das linguagens utilizadas.

Nível da linguagem

- Quanto ao grau de abstração que as linguagens assumem, é possível agrupá-las em diversas classes, que representem os elementos de uma hierarquia baseada no **nível de abstração** adotado:
  - de **máquina** (binária, numérica, simbólica)
  - **simbólica** de baixo nível (de montagem – assembly)
  - de **nível intermediário** (macros – macroassembly)
  - de **alto nível** (científicas, comerciais, visuais, etc.)
  - de **consulta** (uso específico, consulta a bancos de dados, etc.)
  - **natural** (uso de línguas naturais na comunicação com a máquina)

Plataforma

- De acordo com a **plataforma** (máquina hospedeira e seu sistema de programação) sobre a qual o programa que está sendo construído deve ser executado, podemos identificar dois grandes grupos:
  - **Plataforma real**: Neste caso, o sistema de programação é implantado diretamente sobre um computador físico (hardware)
  - **Plataforma virtual**: Neste caso, o sistema de programação é executado em uma máquina virtual, ou seja, em um simulador (software) que implementa a arquitetura desejada, sendo esse simulador, por sua vez, e executado em algum hardware físico disponível.

Ambientes e ferramentas

- Os sistemas de programação costumam proporcionar a seus usuários **ambientes e ferramentas** de auxílio à **elaboração e manutenção** dos seus programas.
- Alguns recursos dessa natureza são frequentes nos sistemas de programação modernos, e são voltados ao apoio de atividades referentes a:
  - Codificação, interpretação, compilação, programação, codificação, depuração de programas (suporte ao **desenvolvedor**)
  - Modularização, relocação, ligação, execução (**software básico**)
  - Comunicação, sincronização, organização, armazenamento, segurança, sinalização, virtualização (**sistema operacional**)
  - Programação multi-linguagem e multi-paradigma (**metodologia**)

Notações para a codificação de programas

- Para a programação dos computadores, vimos que a própria **máquina** disponibiliza a sua **linguagem binária**, que pode ser denotada de várias formas, **numéricas ou simbólicas**.
- Tal conjunto de notações são chamadas **linguagens de baixo nível** pois disponibilizam ao programador o uso direto dos componentes básicos do hardware.
- Contudo, tal nível de detalhe nem sempre é desejável na programação, sugerindo a busca de **outras notações** que tornem mais confortável a codificação dos programas.
- Isso motiva também que, além de disponibilizar linguagens de programação, o sistema de programação também forneça **ferramentas e ambientes** que facilitem a obtenção de programas executáveis, a partir de textos denotados nessas linguagens de programação.

Linguagens de programação

- A programação em **linguagens de baixo nível** (linguagem de máquina binária ou de outras bases numéricas, e linguagens simbólicas) carecem de recursos linguísticos para exprimir muitos fatos da programação, forçando o programador a trabalhar em um nível extremo de **proximidade com a máquina**.
- Isso é **desconfortável**, contraproducente, difícil e muito sujeito a falhas humanas, mesmo com a ajuda e muitas ferramentas úteis que foram criadas e extensivamente utilizadas para amenizar as dificuldades encontradas no uso das linguagens de baixo nível.

Elevação do nível de abstração

- Superou-se esse problema com a introdução de linguagens voltadas à manipulação de conceitos mais abstratos, que **elevaram o nível de abstração** exigido na programação, distanciando o programador do hardware e permitindo que se aproximasse um pouco mais das suas aplicações.
- Nos dias de hoje os programadores têm à disposição uma profusão de **linguagens de alto nível** dos mais variados tipos, estilos e características.
- Na maior parte das situações normais, as linguagens de alto nível **eliminam a necessidade de recursos** de programação característicos dos programas desenvolvidos em linguagens **de baixo nível**, como é o caso dos registradores do hardware, instruções de máquina, detalhes do sistema de interrupção, etc.

As linguagens de alto nível

- Permitem exprimir os programas de maneira muito **menos dependente de máquina** pela eliminação da referência explícita aos elementos do hardware.
- Na fase da elaboração da lógica dos programas, o emprego desse tipo de linguagens de programação sugere o uso de **diagramas** e de **pseudo-códigos.**
- Possibilitam e favorecem o uso de **notações** mais próximas da linguagem **humana**, suscitando a criação e a adoção de **linguagens artificiais** próprias para a programação, ditas **linguagens de alto nível**.

Linguagens pioneiras de alto nível

- Quatro linguagens, nascidas nos primórdios da história da computação, merecem destaque especial pelo **papel histórico e técnico** que desempenharam e desempenham até hoje na computação: **FORTRAN, BASIC, COBOL e LISP**.
  - **FORTRAN**, como líder da computação científica
  - **BASIC**, como precursora das linguagens de script
  - **COBOL**, como líder da computação comercial
  - **LISP**, pioneira das linguagens funcionais para aplicações em Inteligência Artificial
- A seguir, revemos amostras do aspecto físico de programas escritos nessas quatro linguagens (já comentadas em uma aula anterior)

Aspecto de um antigo programa FORTRAN

```
C       CALCULATE STATISTICS ON DATA FROM LOW SPEED READER
        SUM=0
        SUMSQ=0
        TYPE 100
100     FORMAT("ENTER THE NUMBER OF VALUES TO CACULATE STATISTICS ON",/)
        ACCEPT 10,N
10      FORMAT(I)
        DO 200 I=1,N
        READ 1,110,V
110     FORMAT(E)
        SUM=SUM + V
        SUMSQ=SUMSQ + V*V
        TYPE 120,I,V
120     FORMAT("VALUE",I,"IS",E,/)
200     CONTINUE
        SAMP=N
        AVRG=SUM/SAMP
        STD=SQTF(SUMSQ/SAMP - AVRG**2)
        TYPE 300,N,AVRG,STD
300     FORMAT("NUMBER OF VALUES",I,"MEAN",E,"STANDARD DEVIATION",E,/)
        END
```

Aspecto de um antigo programa BASIC

```
 10 INPUT "What is your name: "; U$
 20 PRINT "Hello "; U$
 25 REM
 30 INPUT "How many stars do you want: "; N
 35 S$ = ""
 40 FOR I = 1 TO N
 50 S$ = S$ + "*"
 55 NEXT I
 60 PRINT S$
 65 REM
 70 INPUT "Do you want more stars? "; A$
 80 IF LEN(A$) = 0 THEN GOTO 70
 90 A$ = LEFT$(A$, 1)
100 IF (A$ = "Y") OR (A$ = "y") THEN GOTO 30
110 PRINT "Goodbye ";
120 FOR I = 1 TO 200
130 PRINT U$; " ";
140 NEXT I
150 PRINT
```

Aspecto de um antigo programa COBOL

```
IDENTIFICATION DIVISION.

PROGRAM-ID. COBOL-DEMO.
AUTHOR.     M. A. COVINGTON.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-PC.
OBJECT-COMPUTER. IBM-PC.

DATA DIVISION.

WORKING-STORAGE SECTION.
77  SUM    PICTURE IS S999999, USAGE IS COMPUTATIONAL.
77  X      PICTURE IS S999999, USAGE IS COMPUTATIONAL.

PROCEDURE DIVISION.

START-UP.
    MOVE 0 TO SUM.
GET-A-NUMBER.
    DISPLAY "TYPE A NUMBER:" UPON CONSOLE.
    ACCEPT X FROM CONSOLE.
    IF X IS EQUAL TO 0 GO TO FINISH.
    ADD X TO SUM.
    GO TO GET-A-NUMBER.
FINISH.
    DISPLAY SUM UPON CONSOLE.
    STOP RUN.
```

Aspecto de um antigo programa Lisp

```
(DEFINE (DRIVER)
        (DRIVER-LOOP <THE-PRIMITIVE-PROCEDURES> (PRINT '|LITHP ITH LITHTENING|)))

(DEFINE (DRIVER-LOOP PROCEDURES HUNOZ)
        (DRIVER-LOOP-1 PROCEDURES (READ)))

(DEFINE (DRIVER-LOOP-1 PROCEDURES FORM)
        (COND ((ATOM FORM)
               (DRIVER-LOOP PROCEDURES (PRINT (EVAL FORM '() PROCEDURES))))
              ((EQ (CAR FORM) 'DEFINE)
               (DRIVER-LOOP (BIND (LIST (CAADR FORM))
                                  (LIST (LIST (CDADR FORM) (CADDR FORM)))
                                  PROCEDURES)
                            (PRINT (CAADR FORM))))
              (T (DRIVER-LOOP PROCEDURES (PRINT (EVAL FORM '() PROCEDURES))))))
```

Figure 1
Top Level Driver Loop for a Recursion Equations Interpreter

Processamento de linguagens de alto nível

- Apesar da estrutura relativamente simples das linguagens de alto nível, a obtenção manual de **códigos executáveis** a partir das **linguagens de alto nível** não é nada trivial.
- Isso torna essencial a criação de processos **automáticos** para torná-las executáveis, tais como a **tradução** e a **interpretação**
- **Compiladores** são os programas de sistema que executam a tarefa da **tradução** e têm, para as linguagens de alto nível, um papel similar ao que têm os montadores, para as linguagens de baixo nível.
- **Interpretadores** são programas de sistema que analisam o programa, executando, de acordo com os resultados da análise, as correspondentes tarefas de **interpretação**. Desempenham papel similar ao dos simuladores de script, ou, para linguagens de baixo nível, dos simuladores de máquinas virtuais.

Compiladores

- Uma das formas de implementar uma linguagem de alto nível é através do uso de **compiladores**.
- Compiladores convertem a linguagem de **entrada** (de **alto nível**) para alguma forma que seja **executável**, direta ou indiretamente, no **computador** ou na **máquina virtual** que se deseja utilizar.
- Por exemplo, a sua linguagem de **saída** pode ser a **linguagem de máquina** binária ou simbólica, ou alguma linguagem de **alto nível** para a qual um compilador esteja **disponível** na plataforma computacional a ser utilizada.
- Embora mais complexo, o processo da **compilação** de programas em linguagem de alto nível é conceitualmente muito **similar ao da montagem** de programas escritos em linguagem simbólica.
- Os compiladores recebem como **entrada** um programa-fonte, escrito em **linguagem de alto nível.**
- Geram como **saída** um programa equivalente em alguma outra linguagem, **tipicamente uma linguagem de máquina** (em geral, relocável)

Interpretadores

- Uma **alternativa** para processar programas denotados em linguagem de alto nível é a utilização de **interpretadores**
- Esses programas de sistema **percorrem o programa** a executar, analisando-o instrução por instrução e aplicando, a cada passo, decisões resultantes dessa análise.
- Não geram código de máquina, mas **simulam a execução** direta **dos comandos** do programa.
- **Interpretadores** não efetuam conversões no programa a executar, mas em lugar disso, para cada um dos diversos passos presentes no script de entrada, promovem a execução dos procedimentos que produzam os resultados adequados.
- A implementação de interpretadores e de ambientes de execução para linguagens de script é mais uma aplicação oportuna dos **motores de eventos** estudados na simulação de sistemas reativos guiados por eventos.

Apoio a Metodologias de Programação

- O advento da Engenharia de Software, na década de 1970, motivou que as linguagens de programação incluíssem cada vez mais **recursos para disciplinar e tornar eficiente** o desenvolvimento de programas, através da adoção de **linguagens aderentes a metodologias** de programação.
- Esta tendência emergiu com o advento da **Programação Estruturada**, que suscitou a inclusão, nas linguagens da época, de características tais como a **estruturação dos programas em blocos**, o uso do conceito de **variáveis locais e globais**, e diversos outros recursos importantes, hoje incorporados em praticamente todas as linguagens de programação modernas.
- A seguir, para ilustrar, mostra-se esquematicamente a influência da metodologia de programação estruturada sobre a concepção de muitas linguagens da época.

Programação Estruturada

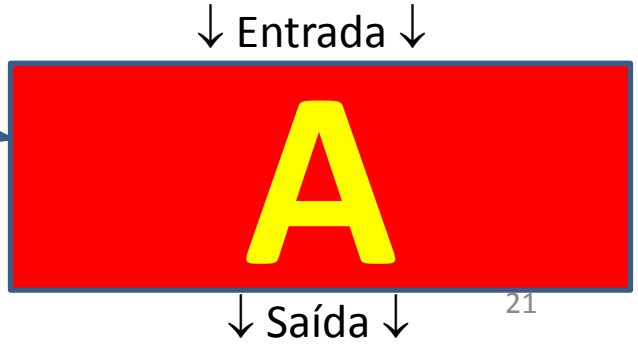
Acompanhou a programação estruturada a introdução do uso dos **diagramas de Nassi-Schneidermann**, como representação gráfica dos grafos planares garantidos pelo teorema de Boehm-Jacopini, diagramas esses que foram adotados como apoio à metodologia de programação estruturada, das primeiras criadas pela Engenharia de Software, então emergente.

Um programa, nesta notação, é denotado como um **bloco** retangular, com uma só entrada (lado superior) e uma só saída (lado inferior). Nesta notação, qualquer **retângulo** pode ser **substituído** por outro, desde que este tenha uma das **três formas básicas** apresentadas a seguir (**sequência, decisão, repetição**). Isso deu a ela a possibilidade de servir como substrato para a técnica de **refinamentos sucessivos**.

Representação de um Bloco:

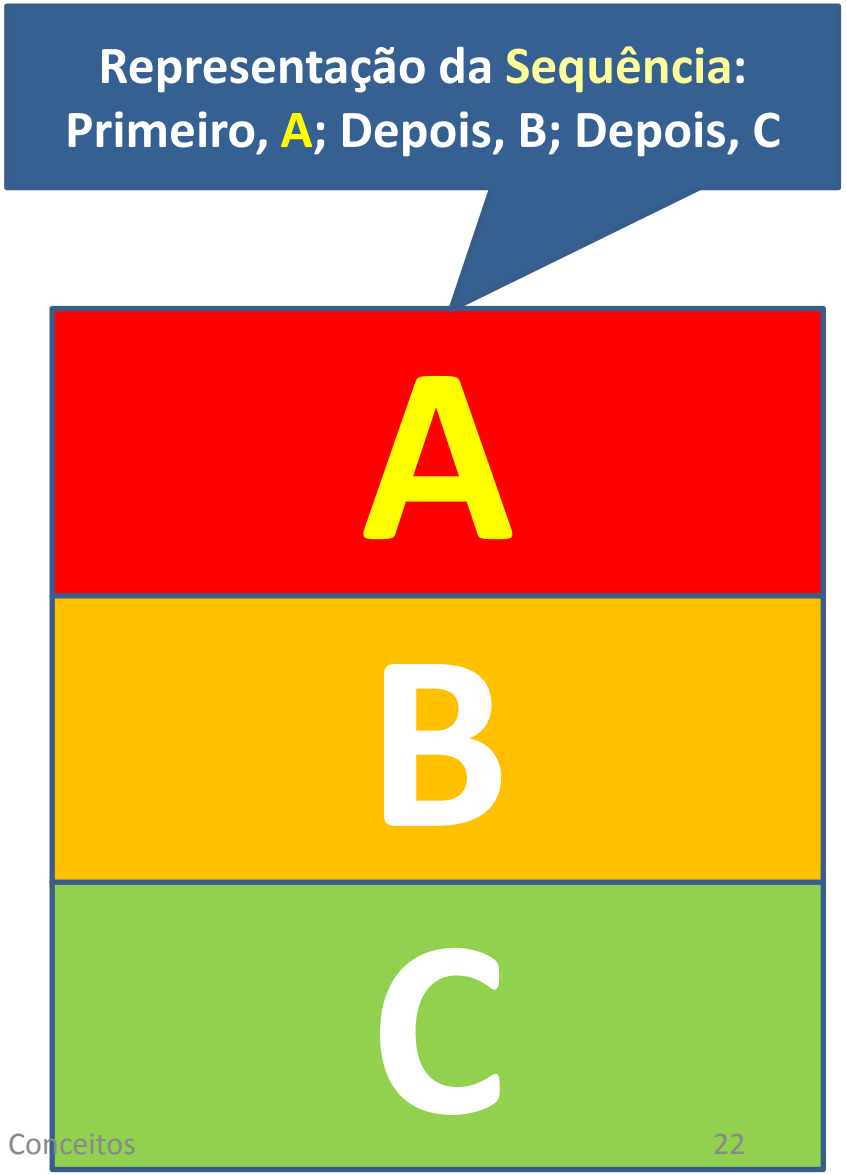
A representa qualquer lógica.

É única a entrada do bloco, pela parte superior, e sua saída, também é única, pela parte inferior.



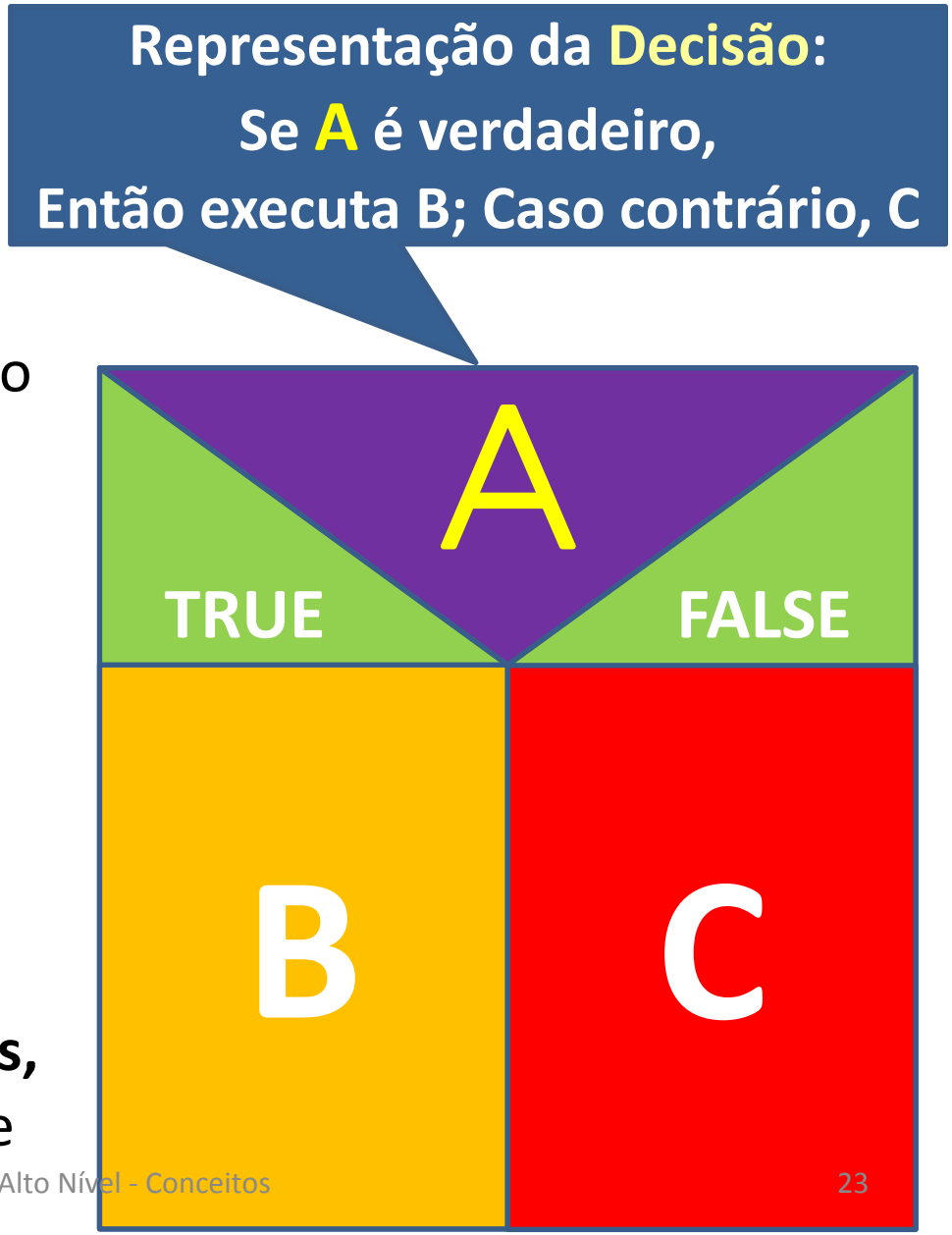
Bloco Básico e Sequência

Qualquer **Bloco Básico** de um programa (um bloco que represente uma lógica com uma só entrada e uma saída apenas) é também representado por um retângulo.
Logo, um **programa** completo também se representa por um retângulo.
Blocos básicos podem ser justapostos na vertical, formando a representação da **Sequência** (ao lado).
As sequências, pela forma como foi convencionada a representação dos blocos básicos, são executadas iniciando-se pelo bloco representado pelo retângulo superior, passam pelos intermediários, e terminam no inferior.



Decisão

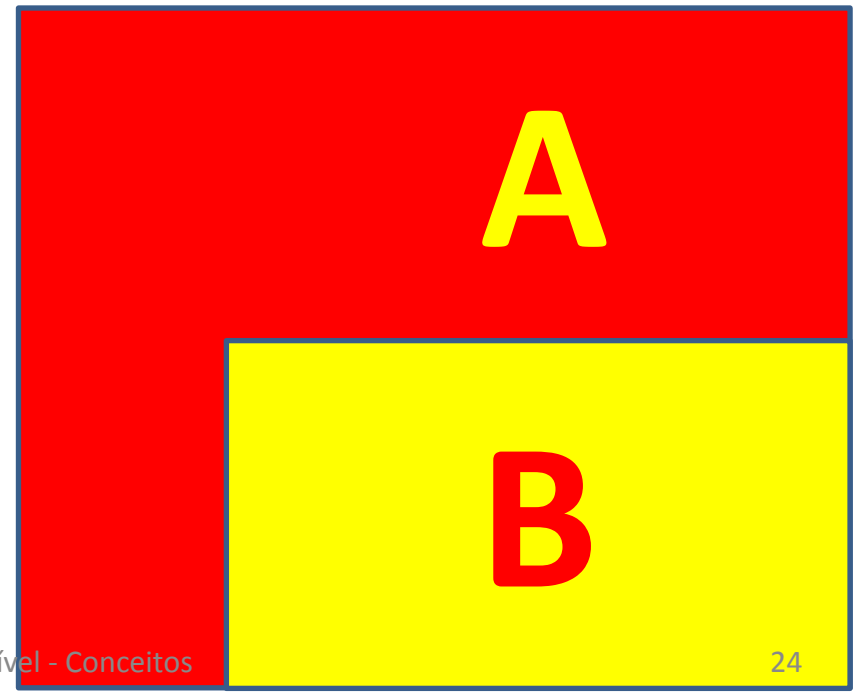
Decisões são blocos básicos com lógica condicional.
São representadas por um retângulo subdividido em três outros: uma **condição** (no retângulo superior) e dois retângulos justapostos horizontalmente, correspondendo a **duas ações** (mutuamente exclusivas) a serem executadas caso a condição seja respectivamente **verdadeira** (retângulo esquerdo) ou **falsa** (retângulo direito).
Sendo eles **mutuamente exclusivos,** não pode haver comunicação entre os blocos básicos B e C.



Repetição

Repetições são representadas por um retângulo subdividido em duas partes na vertical: a parte superior, como no caso da **condição**, representa a condição de parada da repetição (**A**, na figura ao lado), e o retângulo inferior representa a **ação B** a ser repetitivamente tomada enquanto a condição **A** permanecer **verdadeira**.

Representação da Repetição:
Enquanto A for verdadeiro,
executa repetidamente B
Caso contrário, termina



Paradigmas

- **De programação** – Caracterizam padrões de como um programador constrói os seus programas.
- **De linguagens de programação** – Caracterizam padrões de codificação, em geral aderentes às necessidades de algum paradigma de programação adotado.
- Por diversas razões na prática o uso de um paradigma pode ser combinado com o de outros, criando sistemas **híbridos**, que mesclam conceitos e práticas de diferentes paradigmas.
- A variedade de paradigmas propicia o aparecimento de **linguagens de programação aderentes,** que privilegiam a adoção de técnicas e métodos próprios de cada paradigma.
- Na sequência, comentam-se mais um pouco os detalhes dos paradigmas mais adotados pelas linguagens de programação.

Conforme seus fundamentos de apoio

- Conforme os fundamentos teóricos em que se apoiam os seus métodos de programação:
  - Paradigma **Procedimental** – a programação procedimental (ou imperativa) se baseia na utilização de procedimentos iterativos, apoiada nas teorias da Semântica Operacional e das Máquinas de Turing.
  - Paradigma **Funcional** – a programação neste paradigma faz uso extensivo de formulações envolvendo funções recursivas, fortemente apoiadas na teoria do Cálculo Lambda e da Semântica Denotacional.
  - Paradigma **Lógico** – os programas desenvolvidos neste paradigma são construídos com base nos fundamentos da Lógica de Primeira Ordem e da Semântica Axiomática.

Conforme a metodologia adotada

- Conforme as características nos métodos e técnicas de programação adotados para a programação, tem-se:
  - Paradigma **Imperativo** – o programa indica à máquina as sucessivas modificações a fazer sobre seus dados e variáveis de estado para o cálculo da saída desejada.
  - Paradigma **Funcional** – sem empregar variáveis ou estados, o programa constitui uma formulação recursiva do modelo matemático de uma função que, aplicada aos dados de entrada, produz a saída esperada.
  - Paradigma **Declarativo** – programas declarativos fazem uso de uma especificação das propriedades matemáticas da solução desejada do problema, e seus mecanismos automáticos de inferência deduzem tal solução buscando a forma mais adequada de combinar essas propriedades.

Paradigma Imperativo

- Define a programação na forma de uma sequência de instruções ou comandos que sucessivamente **modificam** os dados de **entrada** e o **estado** do programa.
- Também conhecido como **procedimental**, este paradigma se fundamenta nas teorias da **Máquina de Turing e da Semântica Operacional.**
- Os programas desta categoria assumem a forma de textos contendo **comandos**, cada qual ordena ao computador a **execução** de tarefas específicas a eles associadas, na **sequência** determinada pelo programador. Ex.: **Linguagem C**.

Desdobramentos

- Desdobramentos do paradigma imperativo incluem:
  - **Programação Estruturada** – usa só **sequência, decisão e repetição** como elementos construtivos dos programas. Precursora da orientação a objetos, a Programação Estruturada representa programas como **composições de abstrações** mais elevadas: **subrotinas e funções**. Fundamentado no **Teorema de Boehm-Jacopini**. Ex.: Linguagens **Algol, Pascal.**
  - **Programação Visual** – este paradigma se apoia fortemente no conceito de **sistemas reativos**, ou da simulação orientada a eventos: um ambiente de programação permite ao programador dispor ícones sobre uma tela e associá-los a procedimentos de tratamento, a serem executados sempre que o ícone receber um estímulo, do usuário ou do próprio programa em execução. Ex.: **Linguagem Visual Basic**.
  - **Programação Orientada a Objetos** – com raras implementações puras, a orientação a objetos costuma ser geralmente combinada com o paradigma imperativo**. Ex.: Linguagem Smalltalk.**

Paradigma Funcional

- Neste paradigma, a computação é descrita através da avaliação de **funções matemáticas**, dispensando a declaração e a modificação de dados.
- O paradigma funcional tem como fundamento as teorias matemáticas das **funções recursivas**, do **Cálculo Lambda** e da **Semântica Denotacional**.
- Programas funcionais são constituídos por declarações de **funções mutuamente dependentes, paramétricas e recursivas**, e sua execução se resume à avaliação da função que representa o programa como um todo.
- Uma linguagem aderente ao paradigma funcional é a **Linguagem LISP**.

Paradigma Declarativo

- No paradigma declarativo (ou lógico), o programador define seus programas na forma de uma **descrição formal** das **propriedades matemáticas da solução** procurada.
- A programação declarativa se fundamenta nas teorias da **Lógica de Primeira Ordem** e da **Semântica Axiomática**.
- Neste paradigma, o programador define **regras lógicas** destinadas a embasar a busca automática de respostas a perguntas feitas ao sistema, para assim resolver os problemas.
- O programador fornece a um ambiente de execução um conjunto de **regras** e uma série de **fatos**, cabendo ao computador, dada uma **questão formulada pelo usuário** através de uma linguagem adequada, determinar a combinação adequada das regras e fatos iniciais que validem as asserções formuladas. Ex.: **Linguagem PROLOG**.

Componentes das Linguagens Imperativas

- As linguagens usuais de programação se apresentam em geral na forma de textos nos quais se podem identificar estruturas com diversas granularidades, entre as quais:
  - **Elementos básicos de representação** – tipicamente os programas se codificam usando caracteres ASCII, agrupados em arquivos de texto.
  - **Elementos léxicos** – formas elementares compostas de caracteres ASCII, tipicamente: identificadores, numerais inteiros em diversas bases, numerais de ponto flutuante, palavras reservadas, sinais de pontuação, sinais de agrupamento e separação, etc.
  - **Comentários** – sem valor para o computador, são textos explicativos dos programas, muito úteis para o programador.
  - **Abreviaturas** – são tipos de abstração, utilizadas nos programas, assumindo a forma de funções, classes, subrotinas, macros, etc.
  - **Construções sintáticas gerais** – listas de: identificadores, constantes, parâmetros e argumentos; sequências; expressões; seletores, etc.
  - **Construções sintáticas específicas** – declarações, comandos, blocos, estruturas de dados, estruturas de controle, etc.

Expressões aritméticas

- As **linguagens** de alto nível mais antigas tinham o seu foco na **codificação de fórmulas** matemáticas.
- A uma das primeiras linguagens de alto nível que surgiram foi, por essa razão, dado o nome de

  FORTRAN = **FOR**mula **TRAN**slation
- Entre outras das mais importantes **linguagens pioneiras,** podem ser destacadas:

  o **Basic**, o **Cobol** e o **Lisp**
- Incrivelmente, linguagens tão antigas continuam a ser **utilizadas intensivamente até os dias de hoje** em todo o mundo, para finalidades práticas**.**

Comandos Imperativos

- Diversos tipos usuais de comandos representam, nas linguagens de programação, **simplificações** de algumas construções **sintáticas** que são frequentes em línguas naturais:
  - **atribuição** – let, :=, =, move
  - **condicionais** – if..goto, if..then, if..then..else
  - de **desvio** – go to, branch
  - **iterativos** (*loops*) – while, for, do..until, repeat
  - de **múltipla escolha** – case, switch case
  - de **entrada e saída** – read, print, get, put
  - de **chamada de subrotina** – call, perform

Estruturas de Controle

- Na década de 1970, a então emergente Engenharia de Software iniciava sua atuação com a criação de formas de programação voltadas à **otimização da produtividade** do programador, e à **simplificação** da tarefa de **desenvolvimento e depuração** de programas.
- Nessa linha, na ocasião a Programação Estruturada foi introduzida como nova maneira de programar, com apoio teórico do importante **teorema de Boehm-Jacopini**, segundo o qual são suficientes apenas três estruturas de controle de fluxo: a **sequência**, a **decisão** e a **repetição,** para a representação planar da lógica de qualquer programa, **sem o auxílio de desvios**.

Conclusão

- Após esse pequeno estudo panorâmico sobre programação e linguagens de programação de alto nível, temos elementos objetivos para iniciar uma análise estrutural mais detalhada dessa categoria de linguagens de programação.
- A finalidade desse estudo é a de colher informações adicionais, que sejam necessárias para o desenvolvimento de processadores de linguagens de alto nível (compiladores e interpretadores, temas das próximas aulas).

FIM