PCS 3216 – Sistemas de Programação

Aula 17

Linguagens de alto nível

CONCEITOS

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 uso de notações para a codificação dos programas a serem nele executados.
- Essas notações, ou **linguagens**, bem como os programas de sistema que as manipulam (**compiladores** e **interpretadores**), 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 (**compiladores** e **interpretadores**)
  - **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, 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 programas.
- Alguns recursos dessa natureza são costumeiros 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 os 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 das diversas ferramentas que foram criadas e extensivamente utilizadas para tornar o uso das linguagens de baixo nível o mais cômodo possível.

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 aproximando-o 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

- Exprimem os programas de maneira muito **menos dependente de máquina,** ou seja, eliminando referências explícitas a 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, vemos amostras do aspecto físico de programas escritos nessas quatro linguagens

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 dos montadores para as linguagens simbólicas de baixo nível.
- **Interpretadores** de linguagens de alto nível 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 a dos simuladores de máquinas virtuais para linguagens de baixo nível,.

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 a linguagem de máquina** (em geral, relocável)

Interpretadores

- Uma **alternativa** para o processamento de programas em linguagem de alto nível é a utilização de **interpretadores**
- Esses programas de sistema **percorrem o programa** a executar, analisando instrução a instrução o programa a ser executado, e aplicando as decisões resultantes da 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, executam procedimentos que gerem os correspondentes resultados.
- A implementação 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, nos anos 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 às metodologias** de programação.
- O início desta tendência aconteceu com a **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.
- A seguir, para ilustrar, mostra-se esquematicamente a influência da metodologia de programação estruturada sobre a concepção de muitas linguagens da época.

Programação Estruturada

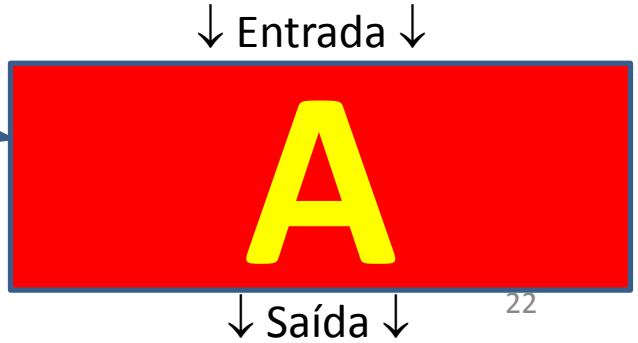
Acompanhou a programação estruturada a introdução do uso dos **diagramas de Nassi-Schneidermann**, como representação gráfica dos grafos planares garantidos pelo teorema de Boehm-Jacopini, diagramas esses que foram adotados como apoio à metodologia de programação estruturada, das primeiras criadas pela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

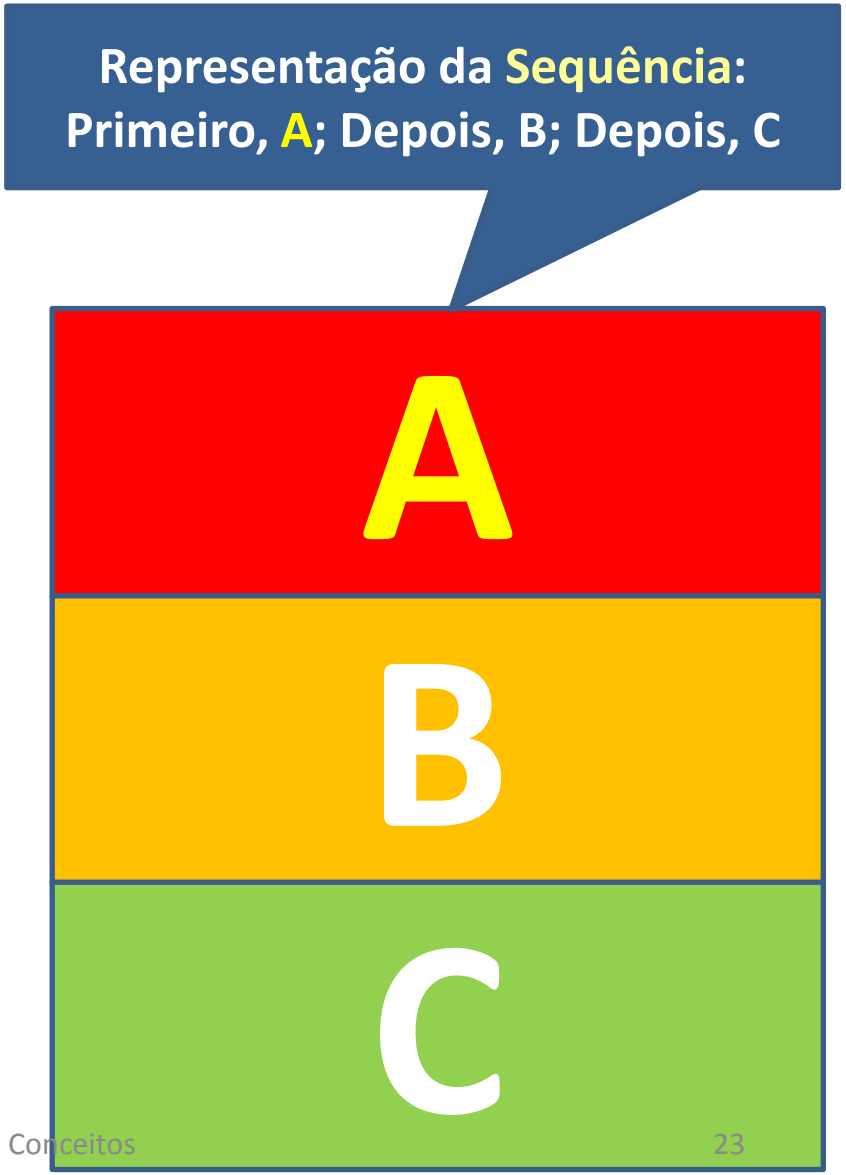
A representa qualquer lógica.

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



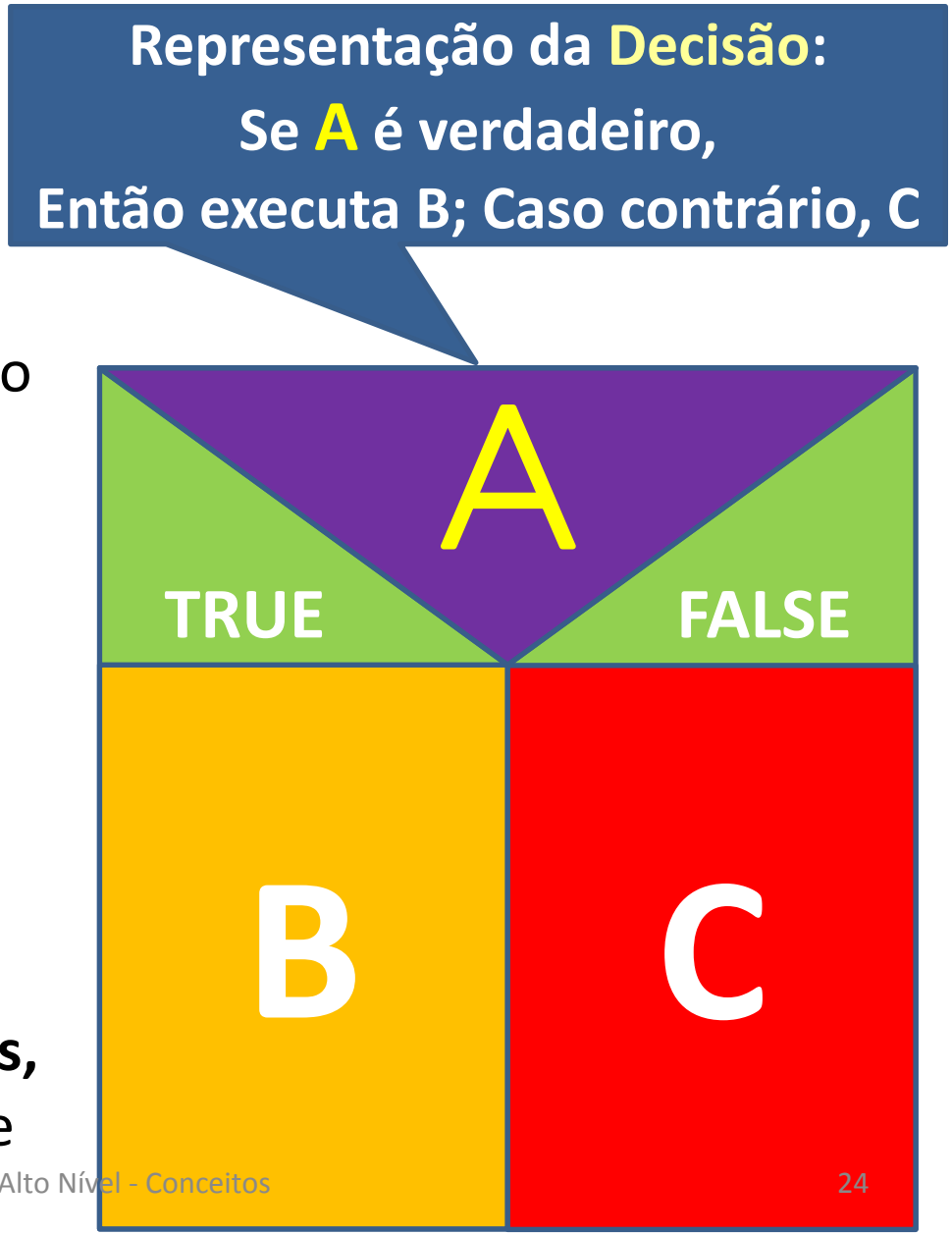
Bloco Básico e Sequência

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

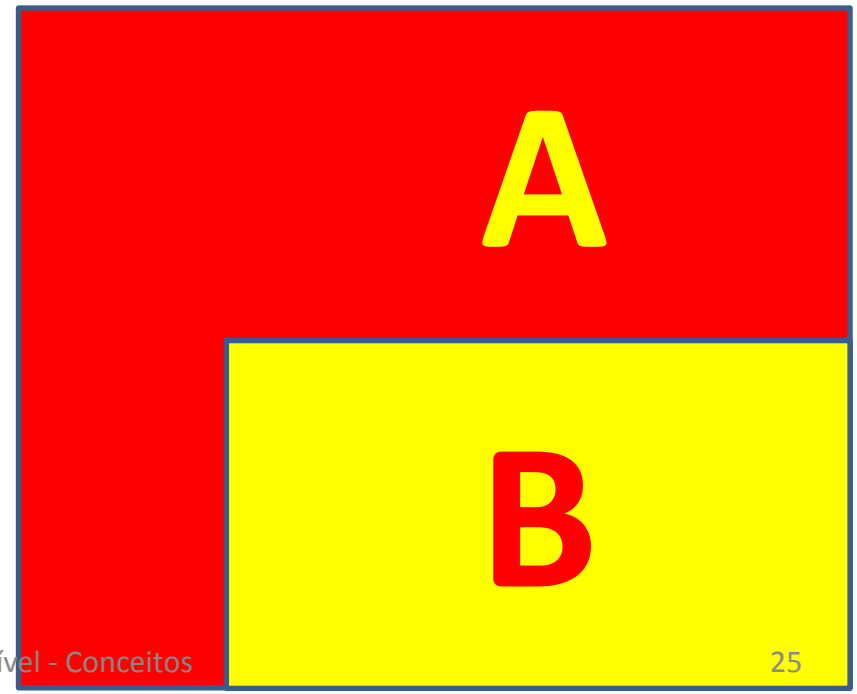
Decisões são blocos básicos com lógica condicional.
São representadas por um retângulo subdividido em três outros: uma **condição** (no retângulo superior) e dois retângulos justapostos horizontalmente, correspondendo a **duas ações** (mutuamente exclusivas) a serem executadas caso a condição seja respectivamente **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 um pouco mais 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 se baseia na utilização de procedimentos iterativos, apoiada na teoria das Máquinas de Turing e da Semântica Operacional.
  - 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 deste paradigma são construídos fundamentados na Lógica de primeira ordem e na 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 é uma formulação recursiva que constitui o modelo matemático de uma função que, aplicada aos dados de entrada, calcula a saída desejada.
  - 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 achando 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

- Neste paradigma, 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**.
- No paradigma lógico, 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, números 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 intensamente até os dias de hoje** 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
  - de **chamada de subrotina** – call

Estruturas de Controle

- Na década de 1970, a 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**.

- Após esse pequeno estudo panorâmico sobre programação e linguagens de programação, temos elementos objetivos para iniciar uma análise estrutural mais detalhada das 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).

- Nesta disciplina será desenvolvido um compilador para uma linguagem imperativa, a qual deve ser devidamente especificada formalmente para que seu reconhecedor sintático possa ser implementado de forma algorítmica.
- Para isso, é preciso obter uma gramática que descreva a sintaxe de cada um dos componentes da linguagem.
- Convém fazer antes um levantamento de tais componentes, a partir da observação de algumas linguagens imperativas.
- Pode-se então formalizar essa sintaxe através do uso de uma metalinguagem apropriada, construindo-se assim uma gramática formal para a linguagem desejada.
- Essa gramática poderá então ser empregada como ponto de partida para a construção de um processador (compilador ou interpretador) para essa linguagem.

Metalinguagens

- Como a etimologia sugere, metalinguagens são notações (ou seja, lingugens) apropriadas para serem utilizadas na descrição formal (no caso, gramáticas) de outras linguagens (no nosso caso, as linguagens de programação que estamos querendo especificar).
- São muito encontradas na literatura as gramáticas denotadas na metalinguagem BNF (Backus-Naur form).
- Nesta disciplina, além da BNF, usaremos também a Notação de Wirth, metainguagem proposta por Niklaus Wirth, derivada da BNF, que se mostra muito prática na construção de compiladores e interpretadores.

Gramáticas

- Gramáticas são textos, escritos usando alguma metalinguagem, cuja finalidade é a de exprimir fatos léxico-sintáticos das linguagens a que se referem.
- Essas gramáticas são coleções de regras, cada qual destinada a detalhar algum aspecto da sintaxe que descreve.
- Regras são compostas de duas partes:
  - Um nome, escrito geralmente à esquerda da regra
  - Um texto, à direita do nome.
- Uma regra inicial representa a linguagem como um todo, e o nome empregado nesta regra simbolizaa forma geral de um texto escrito nessa linguagem.
- Cada regra associa um nome ao texto correspondente, e nas demais regras o aparecimento do mesmo nome funciona como uma abreviatura do texto a ele associado (exatamente como acontece com as macros).
- Textos sintaticamente corretos podem ser obtidos por substituições sucessivas dos nomes pelos textos a eles associados nas regras correspondentes, até que não restem nomes no texto.

BNF (Backus-Naur Form)

- Pioneira, esta metalinguagem tem sido muito usada desde a década de 1960, quando foi historicamente criada e utilizada para a descrição formal da sintaxe livre de contexto da linguagem Algol 60, a primeira das linguagens estruturadas em blocos.
- A gramática do Algol 60 apresentada adiante foi escrita em BNF.
- Notação:
  - Nomes (de classes sintáticas) se escrevem como textos entre < e >.
  - O símbolo ::= apenas separa o nome, na regra, do texto associado.
  - Átomos da linguagem são denotados como textos livres.
  - Barras verticais significam “ou” (união de conjuntos).
  - Justaposição de elementos significa a concatenação dos conjuntos que esses elementos representam.
  - Nos exemplos apresentados, o símbolo # funciona como aspas, significando que o texto circundado por dois símbolos # deve ser interpretado literalmente.
  - A letra grega epsilon (ε) denota a cadeia vazia (uma sequência de zero símbolos)

Notação de Wirth

- A notação BNF só permite denotar repetições por meio de formulações recursivas. A Notação de Wirth oferece em adição uma notação específica para representar repetições, e permite evitar referências à cadeia vazia, necessárias em BNF.
- Essas características da Notação de Wirth a tornam particularmente atraente para a construção de processadores das linguagens que ela permite descrever.
- Isso se deve à correspondência existente entre essa notação e a estrutura dos reconhecedores sintáticos que, com facilidade, podem ser gerados a partir dela.
- Notação:
  - Nomes (de classes sintáticas) são denotados como textos livres.
  - O símbolo = separa o nome, na regra, do texto associado.
  - O símbolo . (ponto) demarca o final de uma regra.
  - A letra grega epsilon (ε) denota a cadeia vazia (sequência de zero símbolos)
  - Cadeias de um ou mais átomos são denotadas como textos entre aspas.
  - Barras verticais significam “ou” (união de conjuntos).
  - Parênteses ( ) simbolizam apenas o agrupamento de opções sintáticas.
  - Colchetes [ ] agrupam sintaxes opcionais (evitando o uso de epsilon)
  - Chaves { } agrupam sintaxes reinstanciáveis (evitando recursões)
  - Justaposição de elementos significa a concatenação dos conjuntos que esses elementos representam.

Elementos de uma linguagem imperativa

- Conforme levantamento realizado na última aula, os variados elementos que devem ser observados nas linguagens de programação podem ser classificados em:
  - Elementos básicos (rótulos, nomes, números, etc)
  - Expressões (aritméticas, booleanas, etc)
  - Estruturas de blocos (para definição de escopos)
  - Declarações (de variáveis, funções, estruturas de dados, etc)
  - Estruturas de controle (sequenciais, condicionais, *loops*, etc)
  - Estruturas de dados (matrizes, vetores, variáveis, registros, etc)
  - Comandos imperativos (atribuições, desvios, entrada e saída, etc)
- Usando, como base para consulta, manuais, tutoriais ou gramáticas de algumas das linguagens de programação imperativas usuais, quais são as formas mais encontradas de cada um dos elementos enumerados, e de outros similares?
- Na sequência, ilustramos este estudo com a linguagem Algol 60, pioneira das linguagens estruturadas em blocos, e que aqui está sendo definida por meio de uma gramática livre de contexto.
- Esta linguagem tem uma complexidade muito maior do que precisamos, portanto devemos usar sua gramática apenas para colher ideias e avaliar a dificuldade inerente aos seus diversos recursos sintáticos.

Elementos básicos

- No nível mais elementar, os programas são constituídos de sequência de caracteres ASCII, que se agrupam formando átomos (embora formados de vários caracteres, os átomos de uma linguagem constituem elementos que perdem seus significados se forem divididos.
- Usualmente observam-se os seguintes elementos básicos (átomos) em uma linguagem típica de programação:
  - Identificadores (nomes de objetos a que o programa se refere)
  - Palavras reservadas, palavras-chave, nomes predefinidos
  - Constantes (numéricas, lógicas, cadeias, etc)
  - Sinais de operação (mais, menos, etc)
  - Sinais de pontuação (vírgula, ponto e vírgula, etc)
  - Agrupadores (parênteses, chaves etc)
- Além disso, devem ser considerados:
  - Espaçadores (espaços, linhas em branco, tabulações, etc)
  - Comentários
- Localize algumas variantes de cada um dos elementos indicados acima, e entre elas, escolha uma que seja do seu agrado, mas não exageradamente complexa, para ser adotada no seu projeto. Percorra os demais tópicos deste levantamento antes de assumir uma escolha definitiva.

Elementos Básicos da Linguagem Algol 60

```
<empty> ::=

<basic symbol>::= <letter> | <digit> | <logical value> | <special symbol> | <delimiter>

<letter> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | Y | W | X | Y | Z
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<logical value> ::= #TRUE# | #FALSE#
<special symbol> ::= <any symbol in CDC 64-character set>
<delimiter> ::= <operator> | <separator> | <bracket> | <declarator> | <specificator >

<operator> ::= <arithmetic operator> | <relational operator> | <logical operator > | <sequential operator>
<arithmetic operator> ::= + | - | * | / | // | ** | ^
<relational operator> ::= < | #le# | = | #ge# | > | ~=
<logical operator> ::= #EQUIV# | #IMPL# | #and# | #OR# | ~
<sequential operator> ::= #GO TO# | #IF# | #THEN# | #ELSE# | #FOR# | #DO#

<separator> ::= # | , | . | : | ; | := | #STEP# | #UNTIL# | #WHILE# | #COMMENT# | #CODE# | #ALGOL# | #FORTRAN# | #RJ#
<bracket> ::= ) | ( | ] | [ | #(# | #)# | #BEGIN# | #END#
<declarator> ::= #OWN# | #BOOLEAN# | #INTEGER# | #REAL# | #ARRAY# | #SWITCH# | #PROCEDURE#
<specificator> ::= #STRING# | #LABEL# | #VALUE# | #VARIABLE# | #SIMPLE# | #FORMAT# | #LIST#

<identifier> ::= <letter> | <identifier><letter> | < identifier><digit>
<ld> ::= <letter> | <digit>
<tail> ::= <ld> | <ld><ld> | <ld><ld><ld> | <ld><ld><ld><ld> | <ld><ld><ld><ld><ld> | <ld><ld><ld><ld><ld><ld>
<external identifier> ::= <letter> | <letter><tail>

<unsigned integer> ::= <digit> | <unsigned integer><digit>
<integer> ::= <unsigned integer> | + <unsigned integer> | - <unsigned integer>
<decimal fraction> ::= .<unsigned integer>
<exponent part> ::= #<integer>

<decimal number> ::= <unsigned integer> | <decimal fraction> | <unsigned integer><decimal fraction>
<unsigned number> ::= <decimal number> | <exponent part> | <decimal number><exponent part>
<number> ::= <unsigned number> | + <unsigned number > | - <unsigned number >

<proper string> ::= <any sequence of characters not containing #(# or #)#> | <empty>
<open string> ::= <proper string> | <proper string>#(#< open string>#)#<open string>
<string> ::= #(#<open string>#)# | #(#<open string>#)#<string>
```

Expressões

- Dentre as construções sintáticas típicas das linguagens de programação imperativas, talvez as mais utilizadas e conhecidas sejam as **expressões**.
- Expressões exprimem composições de operações realizadas sobre os dados referenciados nos programas, de acordo com sua natureza.
- Há dois tipos de expressões que são muito importantes na codificação de programas imperativos:
  - as **expressões aritméticas**, envolvendo constantes e variáveis aritméticas, chamadas de funções aritméticas, e operações também aritméticas
  - as **expressões lógicas** ou **booleanas**, formadas por constantes e variáveis lógicas, chamadas de funções booleanas, comparações entre expressões lógicas ou entre expressões aritméticas
  - Em adição, há **expressões envolvendo cadeias** (strings), chamadas de funções que retornam cadeias, e operações entre cadeias.

Componentes de Expressões no Algol 60

```
<variable identifier> ::= <identifier>
<simple variable> ::= <variable identifier>

<subscript expression> ::= <arithmetic expression>
<subscript list> ::= <subscript expression> | <subscript list>, <subscript expression>

<array identifier> ::= <identifier>
<subscripted variable> ::= <array identifier> [ <subscript list> ]
<variable> ::= <simple variable> | <subscripted variable>

<procedure identifier> ::= < identifier>
<actual parameter> ::= <string> | <expression> | <array identifier> | <switch identifier> | <procedure identifier>
<letter string> ::= <letter> | <letter string><letter>
<parameter delimiter> ::= , | )<letter string>:(
<actual parameter list> ::= <actual parameter> | <actual parameter list> <parameter delimiter> <actual parameter>
<actual parameter part> ::= <empty> | <actual parameter list>
<function designator> ::= <procedure identifier> <actual parameter part>
```

Expressões no Algol 60

```
<expression> ::= <arithmetic expression> | <Boolean expression> | <designational expression>
<adding operator> ::= + | -
<multiplying operator> ::= * | / | //

<primary> ::= <unsigned number> | <variable> | <function designator> | (<arithmetic expression>)
<factor> ::= <primary> | <factor> ** <primary> | <factor> ^ <primary>
<term> ::= <factor> | <term> <multiplying operator> <factor>
<simple arithmetic> ::= <term> | <adding operator > <term> | <simple arithmetic> <adding operator> <term>

<if clause> ::= #IF# <Boolean expression> #THEN#

<arithmetic expression> ::= <simple arithmetic> | <if clause> <simple arithmetic> #ELSE# <arithmetic expression>
<relational operator> ::= < | #le# | = | #ge# | > | ~=
<relation> ::= <simple arithmetic expression> <relational operator> <simple arithmetic expression>

<Boolean primary>::= <logical value> | <variable> | <function designator> | <relation> | (<Boolean expression>)
<Boolean secondary> ::= <Boolean primary> | ~<Boolean primary>
<Boolean factor> ::= <Boolean secondary> | <Boolean factor> #or# <Boolean secondary>
<Boolean term>::= <Boolean factor> | <Boolean term> #and# <Boolean factor>
<implication> ::= <Boolean term> | <implication> #impl# <Boolean term>
<simple Boolean> ::= <implication> | <simple Boolean> #equiv# <implication>
<Boolean expression> ::= <simple Boolean> | <if clause> <simple Boolean> #ELSE# <Boolean expression>

<label> ::= <identifier>
<switch identifier> ::= <identifier>
<switch designator> ::= <switch identifier>[<subscript expression>]

<subscript expression> ::= <arithmetic expression>
<simple designational> ::= <label> | <switch designator> | (<designational expression>)
<designational expression>::=<simple designational>|<if clause><simple designational>#ELSE#<designational expression>
```

Estrutura de blocos

- Linguagens com estrutura de blocos se caracterizam por apresentarem construções sintáticas que permitem delimitar, nos programas, **escopos** para os nomes dos elementos declarados, ou por eles referenciados.
- Em geral, a **demarcação das fronteiras dos escopos** é feita posicionando-se delimitadores, que indiquem os limites físicos dos blocos aninháveis que compõem os programas.
- Os limites de cada bloco estabelecem o **perímetro de atuação dos identificadores** neles declarados.
- **Regras de escopo** costumam determinar a pertinência, ao escopo de um bloco, dos identificadores declarados em tal bloco e dos identificadores não homônimos declarados em blocos nos quais ele esteja aninhado, e exclui blocos paralelos ou externos.

Estrutura de Blocos no Algol 60

```
<compound tail> ::= <statement> #END#
                  | <statement> ; <compound tail>
<block head> ::= #BEGIN# <declaration>
               | <block head> ; <declaration>
<unlabeled compound> ::= #BEGIN# <compound tail>
<unlabeled block> ::= <block head>; <compound tail>
<compound statement> ::= <unlabeled compound>
                       | <label> : <compound statement>
<block> ::= <unlabeled block> | <label> : <block>
<program> ::= <block> | <compound statement>
```

Declarações

- São usadas para atribuir nomes a variáveis, matrizes, outros dados, rótulos e procedimentos utilizados no programa, e associar a eles os correspondentes tipos.
- Utilizando a estrutura de blocos do programa, é possível estabelecer o escopo (conjunto de regiões do programa) em que os nomes declarados são reconhecidos e podem ser utilizados.
- Declarações de dados costumam permitir que o programador especifique escalares, vetores, matrizes e estruturas, e especificar suas dimensões, número de elementos, tipos e outros atributos, conforme o caso.
- Declarações de procedimentos informam ao compilador o nome, os parâmetros, e as ações associadas aos procedimentos declarados, bem como os tipos dos dados que devem resultar de sua execução.

Declarações no Algol 60

<declaration> ::= <type declaration> | <array declaration> | <switch declaration> | <procedure declaration>
<type list> ::= <simple variable> | <simple variable>, <type list>
<type> ::= #REAL# | #INTEGER# | #BOOLEAN#
<local or own> ::= <empty> | #OWN#
<type declaration> ::= <local or own> <type> <type list>
<lower bound> ::= <arithmetic expression>
<upper bound> ::= <arithmetic expression>
<bound pair>::= <lower bound>: <upper bound>
<bound pair list> ::= <bound pair> | <bound pair list>, <bound pair>
<array segment> ::= <array identifier>[<bound pair list>] | <array identifier> , <array segment>
<array list> ::= <array segment> | <array list>, <array segment>
<array declarer> ::= < type> #ARRAY# | #ARRAY#
<array declaration> ::= <local or own><array declarer> <array list>
<switch list> ::= <designational expression> | <switch list> <designational expression>
<switch declaration> ::= #SWITCH# <switch identifier> := <switch list>
<formal parameter> ::= <identifier>
<formal parameter list> ::= <formal parameter> | <formal parameter list> <parameter delimiter> <formal parameter>
<formal parameter part> ::= <empty> | (<formal parameter list>)
<identifier list> ::= <identifier> | <identifier list>, <identifier>
<value part> ::= #VALUE# <identifier list>; | <empty>
<separate specifier>::=#STRING# | <type> | <array declarer> | #LABEL# | #SWITCH# | #PROCEDURE# | <type>#PROCEDURE# | #VARIABLE# |#SIMPLE# | #FORMAT# | #LIST#
<separate specification part> ::= <empty> | <separate specifier><identifier list>; | <separate specification part> <separate specifier> <identifier list>;
<separate procedure heading> ::= <procedure identifier> <formal parameter part>; <value part> <separate specification part>
<code number> ::= <digit> | <digit><digit> | <digit><digit> <digit> | <digit><digit><digit><digit> | <digit><digit><digit><digit><digit>
<code identifier> ::= <empty> | <code number> | <external identifier>
<code specifier> ::= #RJ# | <empty>
<code> ::= #CODE# <code specifier> <code identifier> | #ALGOL#<code specifier><code identifier> | #FORTRAN#<code identifier>
<separate procedure body> ::= <code>
<separate procedure declaration> ::= #PROCEDURE# <separate procedure heading> <separate procedure body> | <type>#PROCEDURE# <separate procedure heading> <separate procedure body>
<specifier> ::= #STRING# | <type> | <array declarer> | #LABEL# | #SWITCH# | #PROCEDURE# | <type> #PROCEDURE#
<specification part> ::= <empty> | <specifier> <identifier list>; | <specification part> <specifier> <identifier list>;
<procedure heading> ::= <procedure identifier> <formal parameter part>; <value part> <specification part>
<procedure body> ::= <statement>
<procedure declaration>::= #PROCEDURE# <procedure heading> <procedure body> | <type> #PROCEDURE# <procedure heading> <procedure body>

Estruturas de dados

- **Estruturas de dados** são definidas e utilizadas nas linguagens de programação imperativas como elementos de armazenamento de dados, e permitem que estes sejam organizados de uma forma adequada à aplicação.
- Costumam ser classificadas em **homogêneas** (vetores, matrizes) e **heterogêneas** (registros, estruturas), conforme os tipos associados aos seus elementos **componentes** (inteiros, reais, etc).
- Ao contrário dos dados **escalares**, que são **referenciados** simplesmente por seu **nome**, os dados **agregados**, como estruturas homogêneas ou heterogêneas, devem ter seus **elementos referenciados** através de uma seleção, usando-se para isso construções sintáticas conhecidas como **seletores**.
- Por exemplo, o **seletor** para **matrizes** costuma ser denotado pelo **nome da matriz** seguido pela indicação de uma **lista de expressões** a serem usadas como **índices**.

Comandos

- São os elementos sintáticos que representam as atividades básicas que podem ser acionadas de forma imperativa pelo programa.
- Em geral, as linguagem oferecem um repertório variado de **comandos** ao programador, porém há alguns tipos desses comandos que estão presentes praticamente em todas as linguagens usuais:
  - **Imperativos**: entrada e saída, atribuições, indexações, seleção de elementos de estruturas de dados, etc.
  - **Controle de fluxo**: sequências, loops, decisões, desvios, chamadas de procedimentos, etc.
  - **Estruturais**: estrutura de blocos, módulos, etc.

Comandos do Algol 60

```
<unlabelled basic statement>::=<assignment statement> | <go to statement> | <dummy statement> | <procedure statement>
<basic statement> ::= <unlabelled basic statement> | <label> : <basic statement>
<destination>::= <variable> | <procedure identifier>
<left part> ::= <destination> :=
<left part list> ::= <left part> | <left part list><left part>
<assignment statement> ::= <left part list> <arithmetic expression> | <left part list> <Boolean expression>
<go to statement> ::= #GO TO# <designational expression>
<dummy statement> ::= <empty>
<procedure identifier> ::= <identifier>
<actual parameter> ::= <string> | <expression> | <array identifier > | <switch identifier> | <procedure identifier>
<letter string> ::= <letter> | <letter string><letter>
<parameter delimiter> ::= , | )<letter string>:(
<actual parameter list> ::= <actual parameter> | <actual parameter list > <parameter delimiter> <actual parameter>
<actual parameter part> ::= <empty> | ( <actual parameter list> )
<procedure statement> ::= <procedure identifier> <actual parameter part>
<statement> ::= <unconditional statement> | <conditional statement> | <for statement>
<for list element> ::= <arithmetic expression> | <arithmetic expression> #STEP# <arithmetic expression> #UNTIL#
    <arithmetic expression> | <arithmetic expression> #WHILE#< Boolean expression>
<for list> ::= <for list element> | <for list>, <for list element>
<for clause> ::= #FOR# <variable identifier> := <for list> #DO#
<for statement> ::= <for clause><statement> | <label> : <for statement>
<if clause> ::= #IF# <Boolean expression> #THEN#
<unconditional statement> ::= <basic statement> | <compound statement> | <block>
<if statement> ::= <if clause> <unconditional statement>
<conditional statement> ::= <if statement> | <if statement> #ELSE# <statement> | <if clause> <for statement> |
    <label>: <conditional statement>
```

Estruturas de controle

- Em particular, as estruturas de controle permitem ao programador estabelecer a sequência na qual devem ser executadas as diversas partes do seu programa, representadas pelos comandos da linguagem.
- O teorema fundamental da programação estruturada (Teorema de Boehm-Jacopini) estabelece que, para que possa ser representada em uma linguagem de programação a lógica descrita por fluxogramas arbitrários, é condição suficiente que ela ofereça ao programador no mínimo os recursos que lhe permitam a especificação de sequências, decisões e *loops*.
- Naturalmente, para complementar a linguagem em termos da representação de outras funcionalidades, as correspondentes operações imperativas devem também estar representadas na linguagem, como por exemplo, atribuições de valor, entrada e saída de dados, chamadas de subrotinas e funções, comandos de seleção e manipulação de dados, etc.

Mini-linguagens

- Na sequência desta apresentação, é mostrado um conjunto de slides, no qual se esboça, de forma ampla embora incompleta, um passo inicial em direção à especificação conceitual de uma linguagem de programação imperativa simples.
- Trata-se da proposta qualitativa da escolha dos elementos constituintes de uma linguagem particular bastante simples, inspirada na inspeção de outras linguagens, cuja especificação tenha sido objeto de análise prévia.
- Um material detalhado acerca de recursos típicos, encontrados em linguagens de programação usuais, é o livro ***The programming language landscape***, de Ledgard e Marcotty, principal fonte de informação para a elaboração do exemplo adiante apresentado.
- No site da disciplina, estão disponíveis especificações em notação de Wirth para todas as mini-linguagens que tal obra utiliza, assim como explicações do funcionamento dos conceitos e construtos sintáticos ali definidos. Recomenda-se o estudo dessas linguagens e a consulta ao livro para a aquisição de detalhes suficientes para completar a contento esta tarefa.

Uma Mini-linguagem

Descrição informal e incompleta de uma pequena linguagem de programação, adaptada de

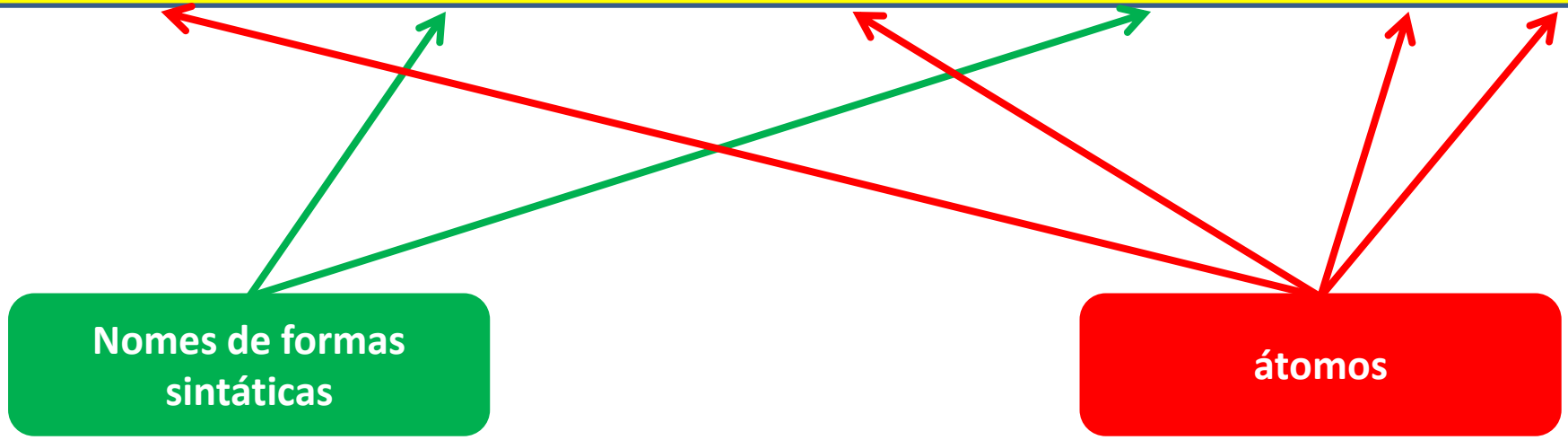
Ledgard & Marcotty , cap.2

The Programming Language Landscape

programa

- consiste da sequência:

“program” *declarações* “begin” *comandos* “end” “;”



EXEMPLO : *programa*

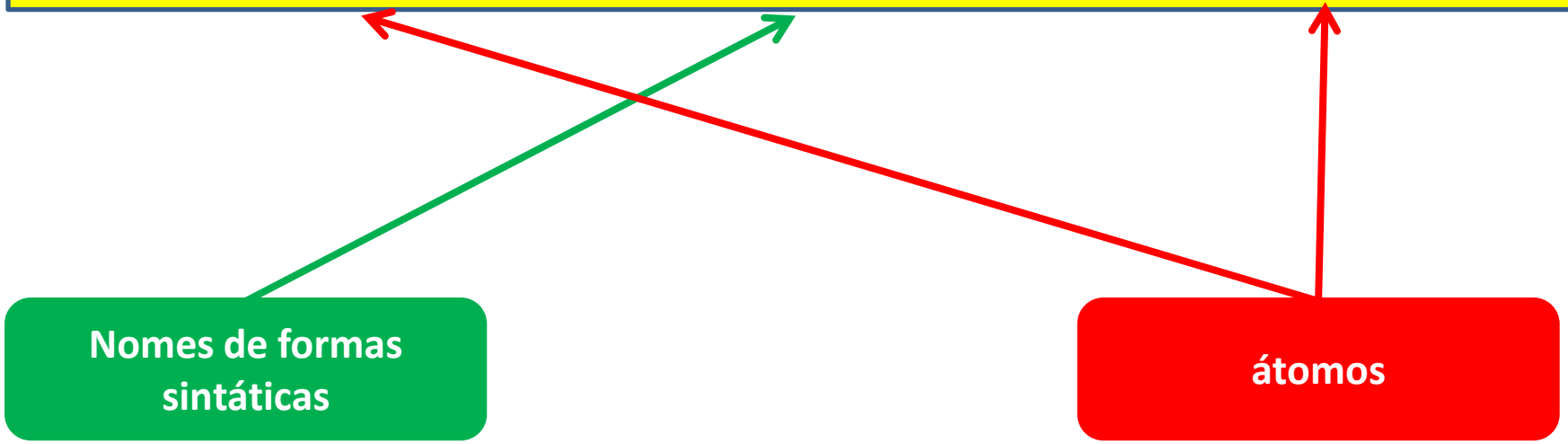
program declare *i, j, k* ;

begin *i:=0; j:=i+1; k:=j-i* end ;

declarações

- É uma sequência de ocorrências da forma

“declare” *lista-de-identificadores* “;”

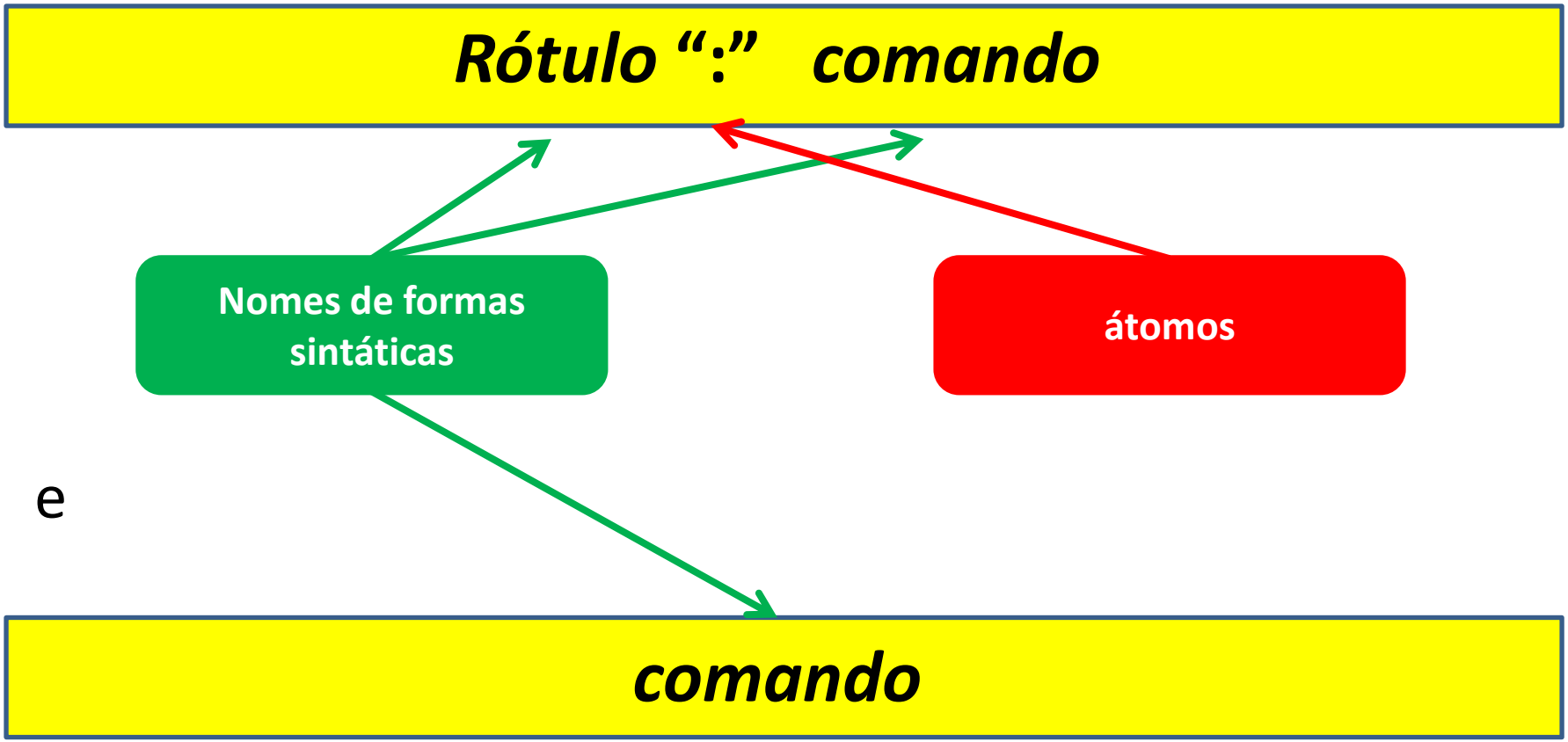


EXEMPLO : *declarações*

declare *i, j, k* ;

comandos

- É uma sequência de ocorrências das formas:

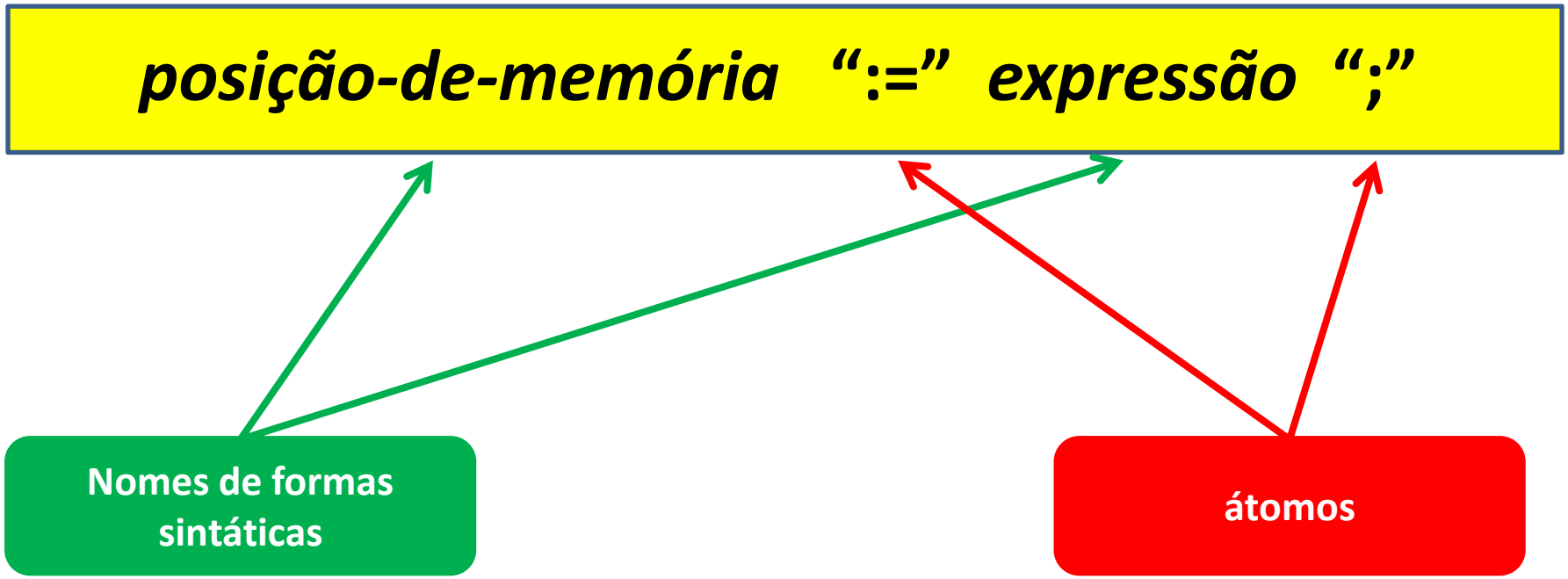


EXEMPLO : *comandos*

```
loop: a := a+b ; go to loop;
```

```
a := a+b ; go to loop;
```

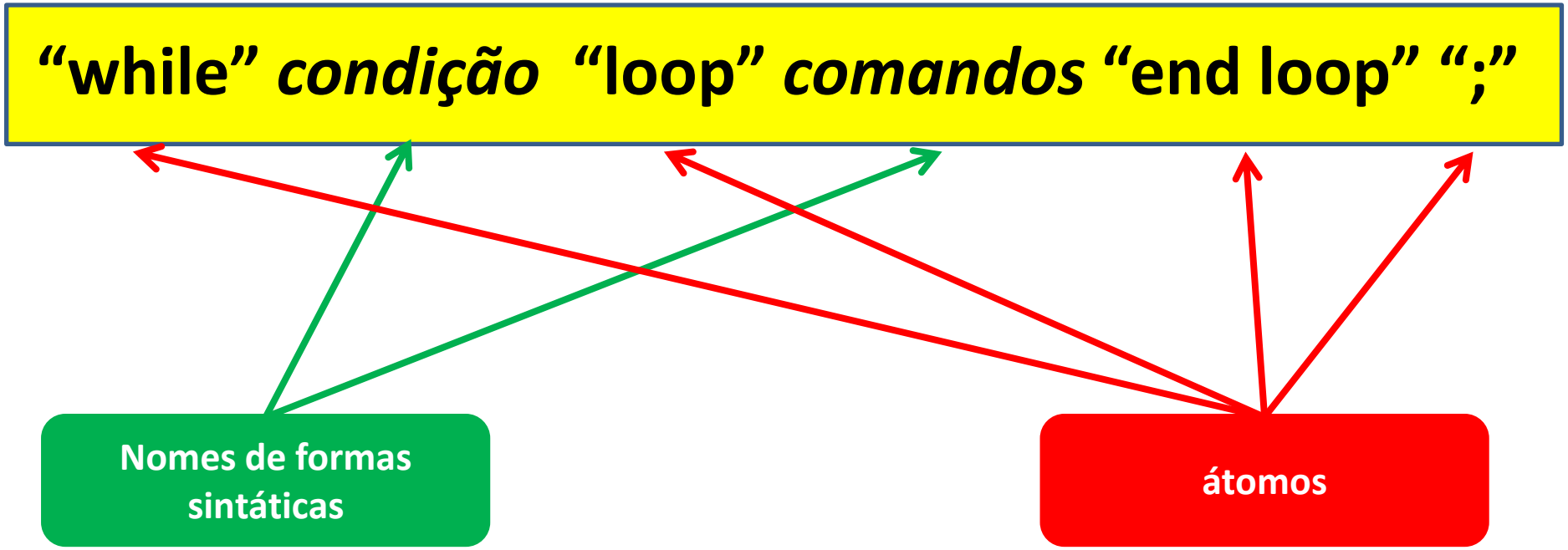
comando-de-atribuição



EXEMPLO : *comando-de-atribuição*

```
x := i+2*(j+x-1);
```

comando-iterativo



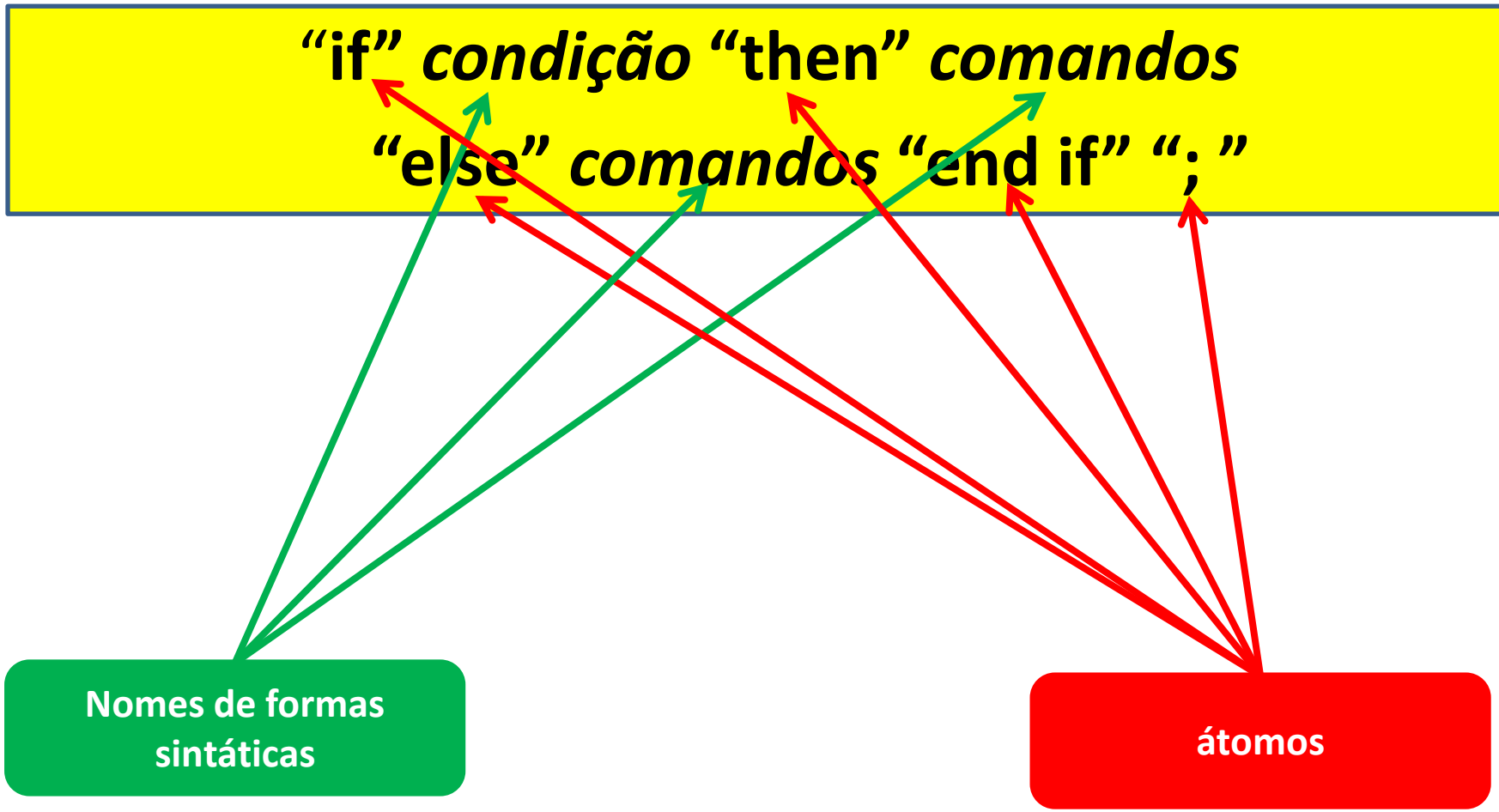
EXEMPLO : *comando-iterativo*

while *i<2*j* loop *i:=i-1; j:=j+1* end loop ;

comando-condicional

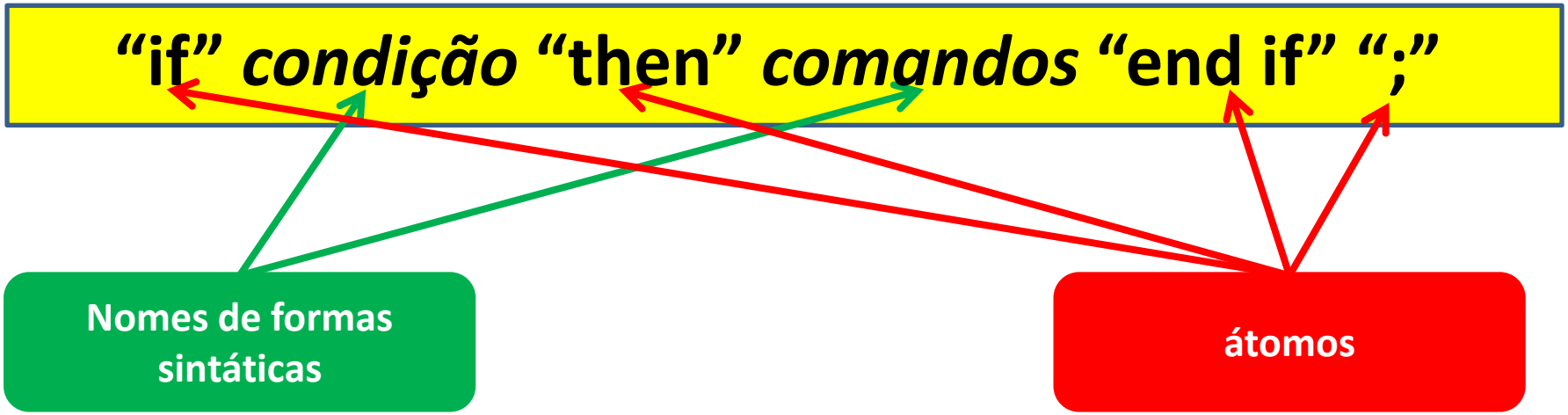
- primeira forma (if..then..else):

“if” *condição* “then” *comandos*
“else” *comandos* “end if” “;”



comando-condicional

- segunda forma (if..then):



EXEMPLO : *comando-condicional*

- primeira forma (if..then..else):

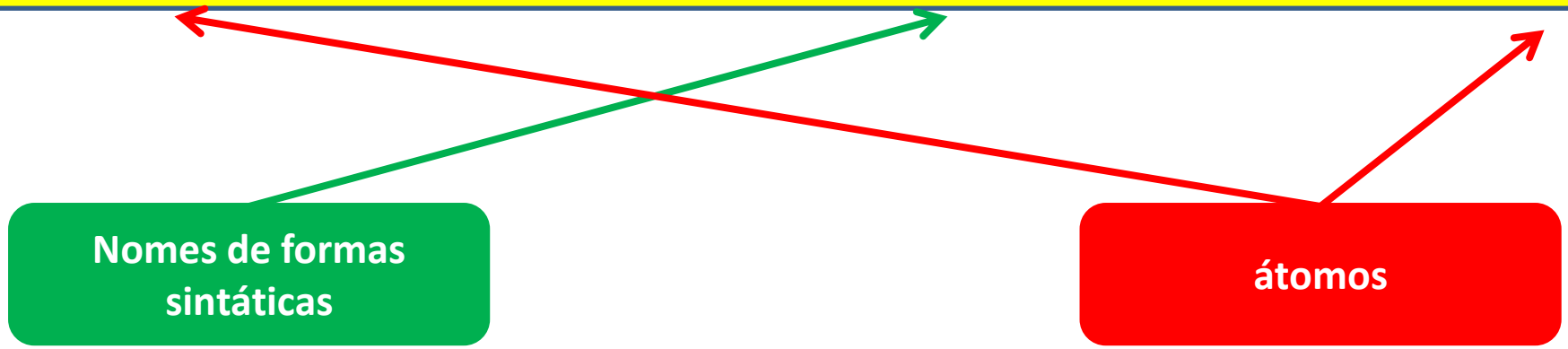
if *x>2* then *i:=5* else *i:=0; j:=j+1* end if;

- segunda forma (if..then):

if *x=0 and j<10* then *x:=x*j-5; j:=0* end if ;

comando-de-entrada

“input” ***lista-de-posições-de-memória “;”***

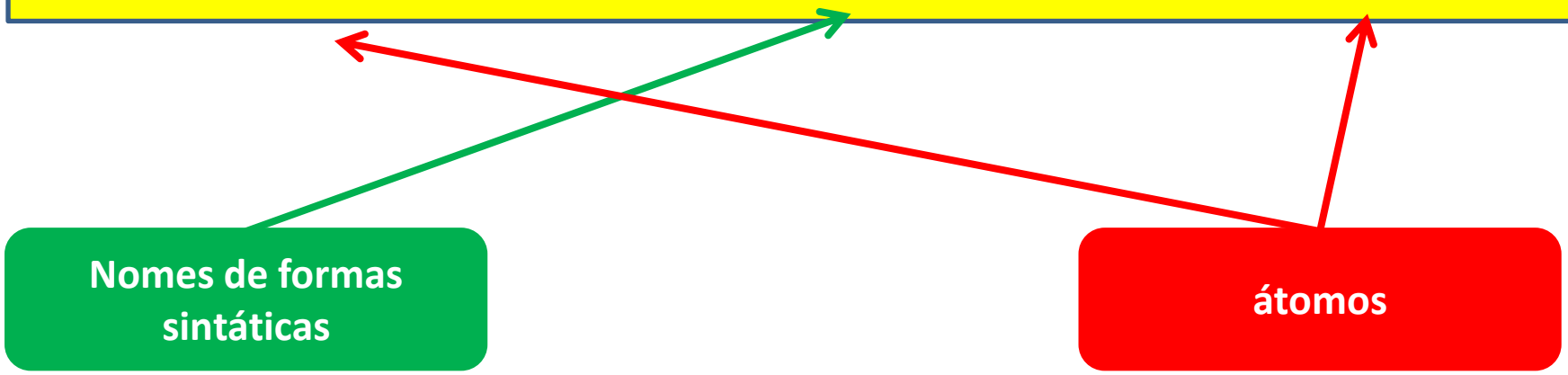


EXEMPLO : *comando-de-entrada*

input *x, y, z* ;

comando-de-saída

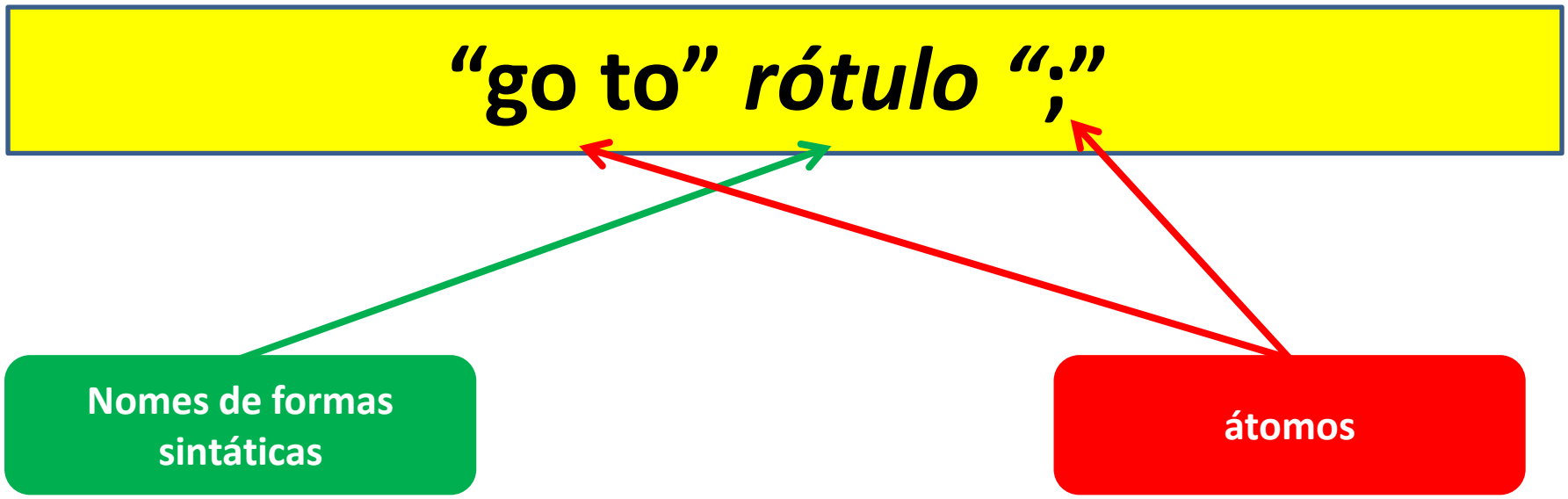
“output” ***lista-de-expressões*** **“;”**



EXEMPLO : *comando-de-saída*

output *2+a , b*(c-5), 2*a+b-1* ;

comando-de-desvio



EXEMPLO : *comando-de-desvio*

go to *loop* ;

Atividades

- Identifique, em alguma linguagem de programação imperativa à sua escolha, e com a qual esteja bem familiarizado, (por exemplo, C, Pascal, Java, Python, Basic, Fortran, Algol, Cobol, etc), cada um dos conceitos mencionados anteriormente.

Formas gerais dos comandos

- Procure, através de exemplos, determinar as possíveis formas diferenciadas que cada um dos comandos indicados pode assumir.
  Por exemplo, os comandos **if** costumam assumir uma das duas formas: com **else** e sem **else.**
  Sempre que for o caso, estabeleça, ainda que de maneira aproximada, alguma “forma geral” para cada comando, de modo que todas as suas variantes possam ser consideradas como casos particulares dessa forma geral.

Descrição informal da linguagem

- Complete uma descrição informal da linguagem que você está elaborando, procurando definir, da maneira que achar mais conveniente, todas as formas léxicas e sintáticas adicionais, que julgar estarem faltando no conjunto de exemplos utilizado.

Formas sintáticas suplementares

- Que outras formas sintáticas adicionais costumam ser encontradas nas outras linguagens de programação que você conhece?
  Por que razões essas formas sintáticas não foram incluídas na mini-linguagem que você propõe?
  Quais das formas sintáticas incluídas na sua minilinguagem você considera que sejam mais relevantes para a programação? Por quê?

Recursos mínimos

- Quais dessas formas sintáticas propostas você considera essenciais, a ponto de não ser possível abrir mão delas, sempre que esta decisão estiver sob seu controle?
  Justifique e mostre qual deve ser a ação de um programador se tiver que utilizar linguagens que não disponham de tais recursos linguísticos.
  Lembre-se de que a maior parte dos recursos podem ser simulados a partir de outros que estejam disponiveis.

Teorema de Boehm-Jacopini

- A mini-linguagem que você está propondo pode ser considerada suficiente para elaborar a descrição da estrutura de controle de qualquer de programa estruturado, de acordo com o que preconiza o teorema de Boehm-Jacopini?
  Por quê?

Sintaxe essencial

- Caso a resposta seja negativa, corrija sua proposta, modificando-a para que tal limitação seja revertida.
  Justifique as alterações realizadas.

Programas elementares

- Escreva pequenos programas que efetuem operações de entradas de dados, algum processamento simples e a saída de resultados, usando esta linguagem.
  Muita atenção à pontuação e aos outros aspectos sintáticos da linguagem proposta, pois considerar com cuidado esses detalhes será muito importante para o bom funcionamento do compilador que você irá desenvolver para essa linguagem.

Programa mínimo

- Apresente um programa sintaticamente correto que seja o mais curto possível (e naturalmente, que tenha sentido) que se pode escrever por intermédio desta linguagem.

Aplicativo simples

- Construa, usando a mini-linguagem proposta, um programa simples que efetue a leitura de duas matrizes quadradas, cuja dimensão é fornecida ao programa como dado, e que calcule a soma das duas matrizes, produzindo uma saída impressa em que são apresentadas as duas matrizes originais e o resultado do cálculo realizado.

Programação estruturada

Represente o aplicativo construído no item anterior na forma de **diagramas de Nassi-Schneidermann**.
Um programa, nesta notação, é representado por um retângulo.
Sequências representam-se justapondo retângulos na vertical.
Decisões são representadas por um retângulo subdividido em três partes: uma condição (retângulo superior) e dois retângulos justapostos horizontalmente, correspondendo às ações (mutuamente exclusivas) a serem executadas caso a condição seja respectivamente verdadeira (retângulo esquerdo) ou falsa (retângulo direito).
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, e o retângulo inferior representa a ação a repetir enquanto a condição for verdadeira.
Qualquer retângulo nesta notação pode ser subdividido em outros, de acordo com as formas básicas: **sequência, decisão, repetição**.

Formalização

- Uma vez obtida uma descrição informal abrangente da linguagem proposta, resta formalizá-la através de uma gramática que a descreva de forma rigorosa e completa.
- A metalinguagem a ser utilizada para isso deverá ser a Notação de Wirth, visto que todos os métodos a serem utilizados para a construção do processador dessa linguagem pressupõem que a gramática esteja expressa nessa notação.