CI1055: Algoritmos e Estruturas de Dados I

Prof. Dr. Marcos Castilho

Departamento de Informática/UFPR

10 de julho de 2020

Resumo

Expressões aritiméticas, booleanas e atribuições

Conceitos elementares de linguagens de programação

- o fluxo de execução de um programa
- os comandos que manipulam dados e permitem interação com o usuário
- as expressões aritméticas e booleanas
- o comando de atribuição
- os comandos que permitem alteração do fluxo de execução do programa

Expressões aritméticas

- permitem a realização de cálculos
- inicialmente veremos como integrar com os comandos que já aprendemos

Final da aula passada

```
program le_e_imprime;
var numero: integer;

begin
    read (numero);
    writeln (numero);
end.
```

- este program lê um valor inteiro do teclado
- depois imprime o valor lido na tela
- vejamos

Expressões aritméticas

Problema: ler um número do teclado e imprimir o dobro dele

```
program le_e_imprime;
var numero: integer;

begin
    read (numero);
    writeln (2 * numero);
end.
```

- observem a operação de multiplicação como argumento do comando de impressão
- a variável `numero` é multiplicada por dois
- usando-se o operador de multiplicação (*)

Cálculo das raízes de equação do segundo grau

- ler três valores do teclado, atribuindo respectivamente para `a, b, c`
- imprimir a primeira raiz pelo método de Bhaskara
- imprimir a segunda raiz pelo método de Bhaskara

Cálculo das raízes de equação do segundo grau

- raiz 1: $\frac{-b-\sqrt{b^2-4ac}}{2a}$
- raiz 2: $\frac{-b+\sqrt{b^2-4ac}}{2a}$
- desde que o discriminante não seja negativo
- vamos supor, agora, que não seja...
- o problema é como fazer estes cálculos

Solução para o problema

```
program bhaskara;
var a, b, c: real;

begin
      read (a, b, c);
      writeln ((-b - sqrt(b * b - 4 * a * c))/(2 * a));
      writeln ((-b + sqrt(b * b - 4 * a * c))/(2 * a));
end.
```

- o comando `write` imprime o resultado do cálculo da fórmula
- este cálculo codifica em *Pascal* a fórmula
- para isso segue algumas regras

Operadores e a função sqrt

- operadores
  - `+`: adição
  - `-`: subtração
  - `*`: multiplicação
  - `/`: divisão real
- a função `sqrt`: implementação nativa da raiz quadrada
- observem os parênteses

Prioridade dos operadores (princípios)

- `( )`: os parênteses têm precedência
- `* /`: operadores multiplicativos precedem os aditivos
- `+ -`: operadores aditivos vêm depois
- o `free pascal` escolhe qual faz primeiro dentro da mesma prioridade

Expressões aritméticas

- é importante ler o material complementar sobre como é a maneira correta de se escrever expressões aritméticas
- aqui só mostraremos o básico

Duas fórmulas diferentes

- $\frac{(6+4)}{2}$
- $6 + \frac{4}{2}$

A expressão abaixo calcula qual das duas acima?

```
6 + 4 / 2
```

- vejamos

E no caso de Bhaskara?

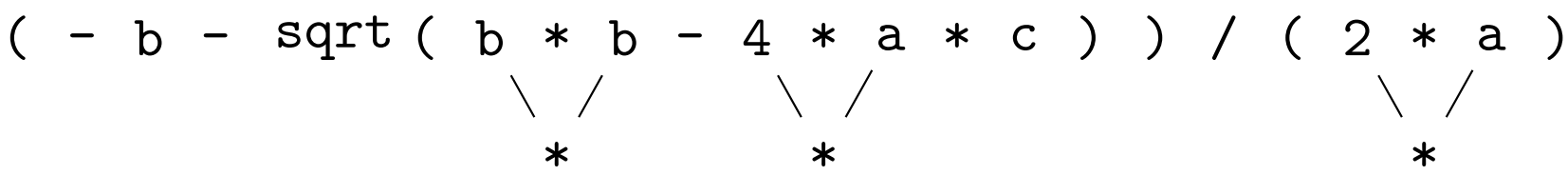
$\frac{-b-\sqrt{b^2-4ac}}{2a}$ `(-b - sqrt (b * b - 4 * a * c)) / (2 * a)`

- primeiro: resolve cada um dos parênteses, os mais internos primeiro:
- depois: dentro de cada parênteses
  - resolve os operadores unários
  - resolve todas as operações multiplicativas
  - depois resolve as aditivas
- vejamos

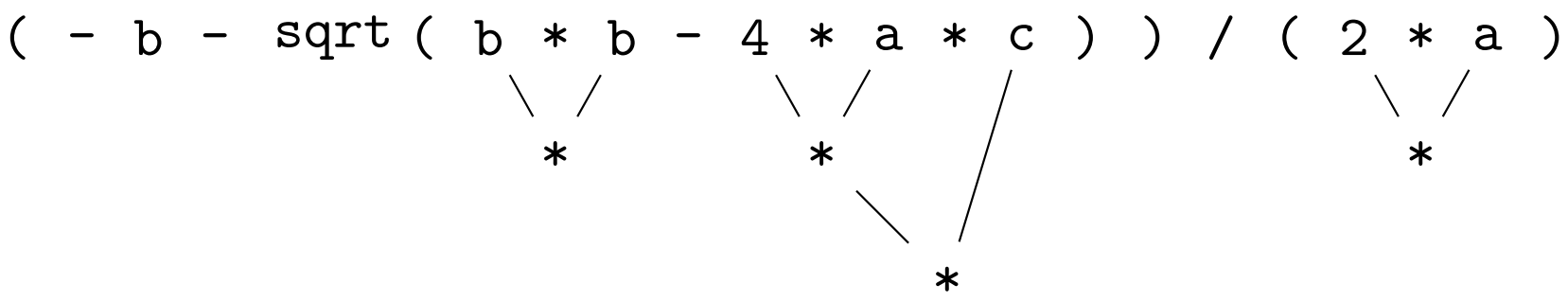
Árvore de operações (início)

```
( - b - sqrt ( b * b - 4 * a * c ) ) / ( 2 * a )
```

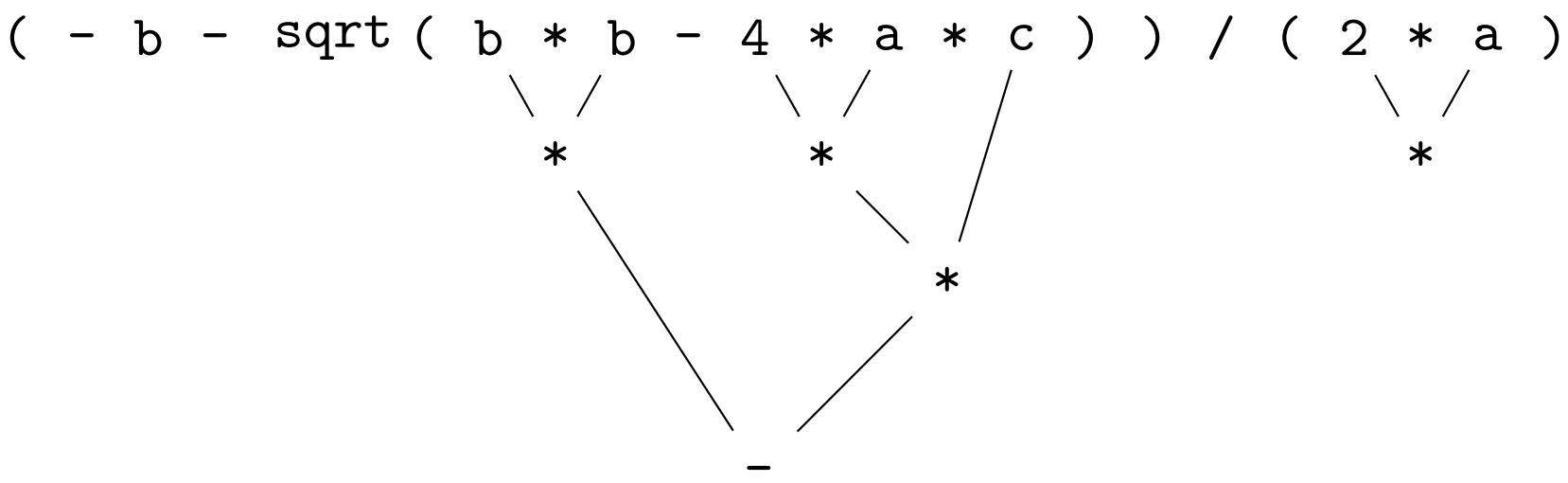
Árvore de operações (etapa 1)



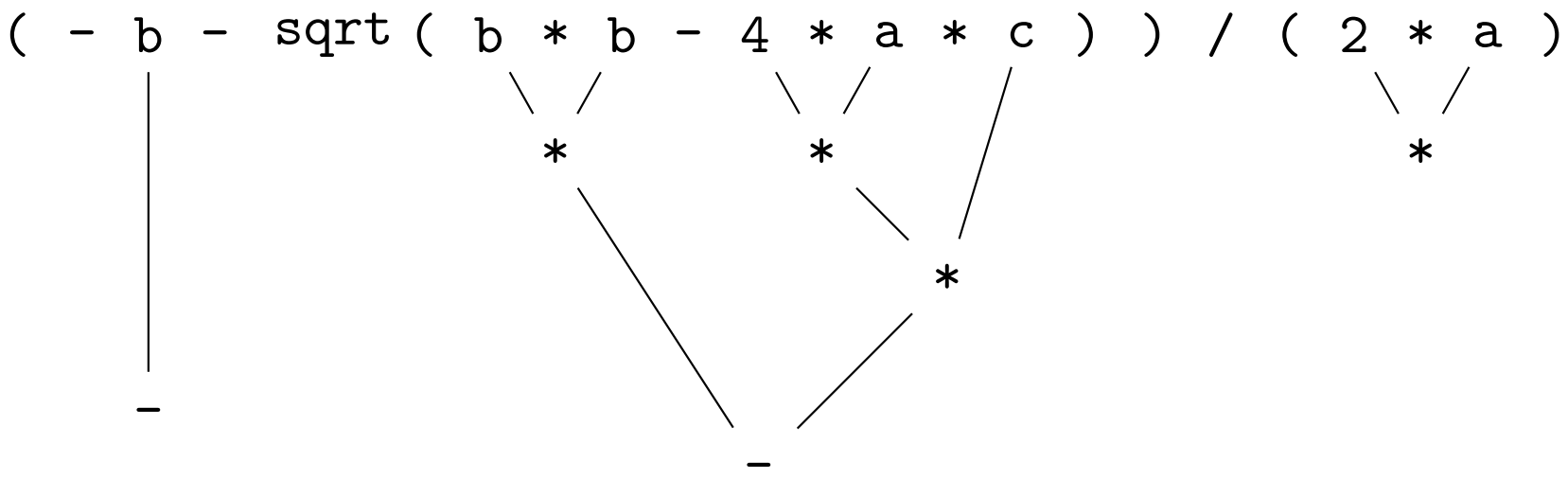
Árvore de operações (etapa 2)



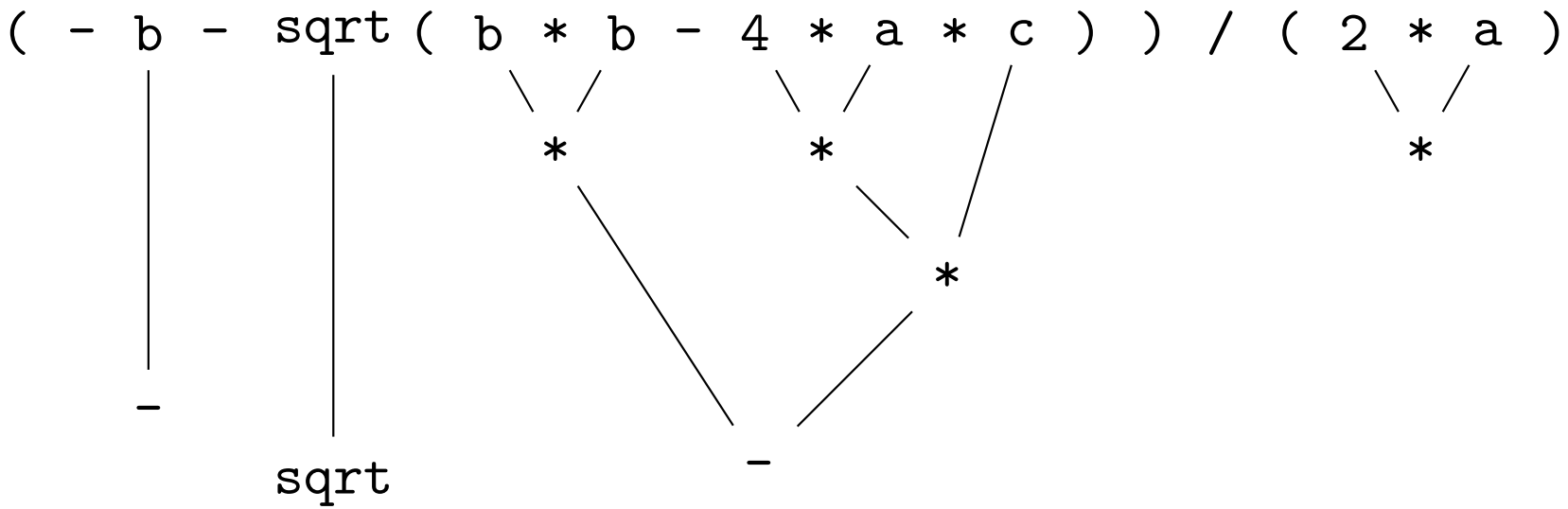
Árvore de operações (etapa 3)



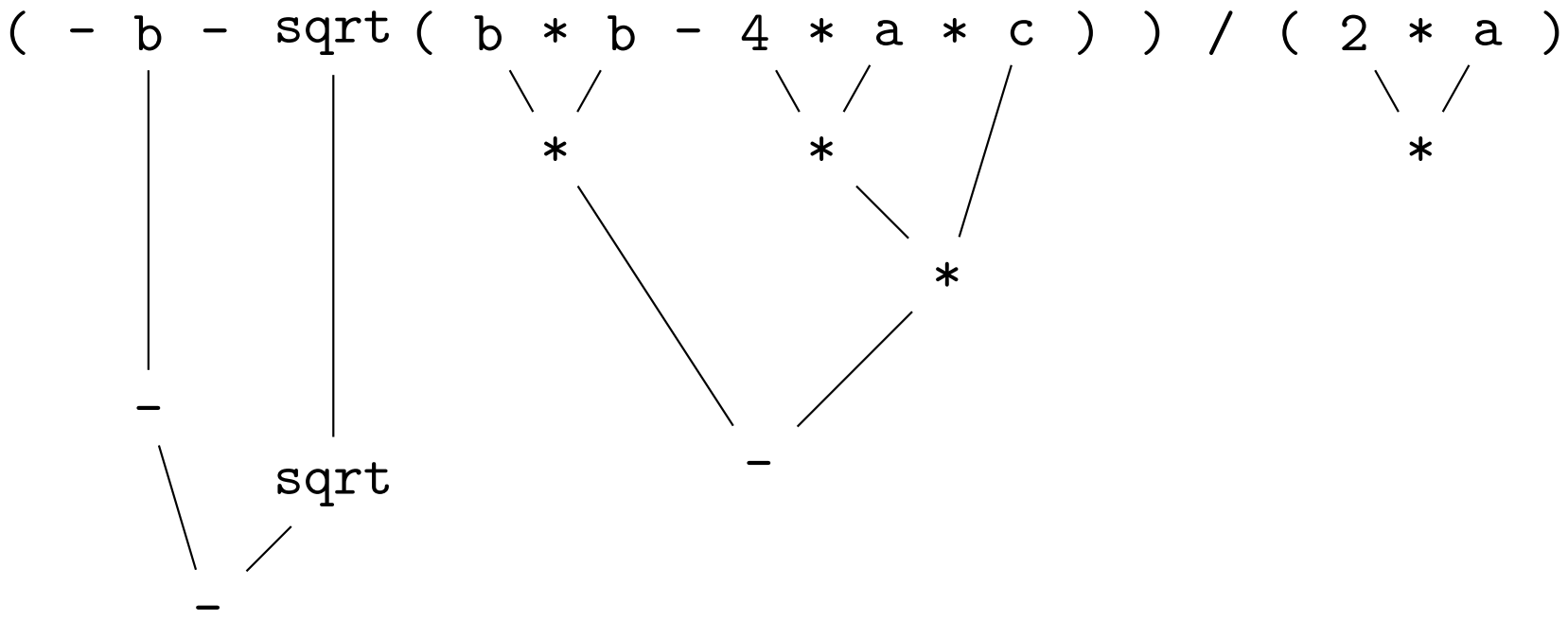
Árvore de operações (etapa 4)



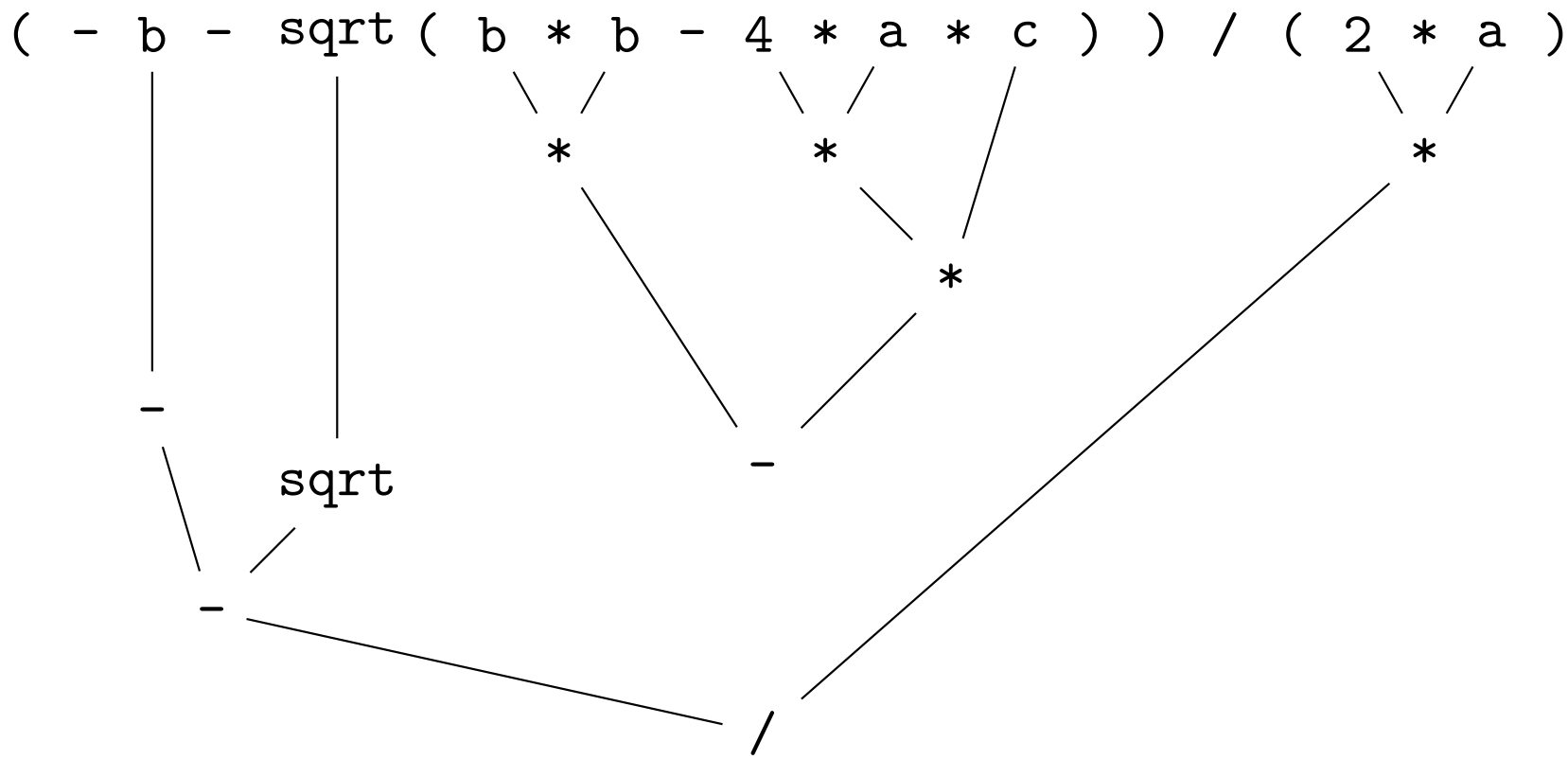
Árvore de operações (etapa 5)



Árvore de operações (etapa 6)



Árvore de operações (etapa 7)



Como seria sem os parênteses?

- `-b - sqrt(b) * b - 4 * a * c / 2 * a`

$$-b - \sqrt{b}b - 4a\frac{c}{2}a$$

Isto é:

$$-b - b\sqrt{b} - \frac{4a^2c}{2}$$

Árvore de operações

```
- b - sqrt ( b ) * b - 4 * a * c / 2 * a
```

Atribuições

```
program bhaskara;
var a, b, c: real;

begin
      read (a, b, c);
      writeln ((-b - sqrt(b * b - 4 * a * c))/(2 * a));
      writeln ((-b + sqrt(b * b - 4 * a * c))/(2 * a));
end.
```

- a operação $\sqrt{b^2 - 4ac}$ é realizada duas vezes!
- para evitar pode-se salvar o primeiro cálculo em uma variável
- usa-se o comando de atribuição `:=`

Atribuições

```
program bhaskara_v2;
var a, b, c, delta: real;

begin
      read (a, b, c);
      delta:= sqrt(b * b - 4 * a * c);
      writeln ((-b - delta)/(2 * a));
      writeln ((-b + delta)/(2 * a));
end.
```

Atribuições

- os símbolos `:=` representam um comando de atribuição em *Pascal*
- a forma geral é: `variavel := expressao_aritmetica`
- funciona assim:
  - primeiro: resolve-se a expressão aritmética que está no lado direito dos símbolos `:=`
  - depois: atribui-se este resultado, que é um número, para a variável cujo nome está do lado esquerdo do símbolo `:=`
- como toda variável, ela deve ser declarada no cabeçalho e ter um tipo **compatível** com os operadores

Outro exemplo com Bhaskara

```
program bhaskara_v3;
var a, b, c, delta, dois_a: real;

begin
      read (a, b, c);
      delta:= sqrt(b * b - 4 * a * c);
      dois_a:= 2 * a;
      writeln ((-b - delta) / dois_a);
      writeln ((-b + delta) / dois_a);
end.
```

Uma atribuição simples e útil

- uma das operações mais utilizadas em algoritmos é o incremento do valor de uma variável inteira de uma unidade
- o comando é assim:
  - `i:= i + 1;`
- suponha que o valor de `i`, antes do comando, tenha o valor 2
- quando o comando é executado, primeiro se determina o valor da expressão do lado esquerdo, que é `i + 1`
- como `i` vale 2, a expressão resulta em 3
- finalmente, o valor 3 é atribuído à variável `i`

Exemplo de incremento

```
program incremento;
var i: longint;

begin
    i:= 2;
    writeln ('i valia antes do incremento: ',i);
    i:= i + 1;
    writeln ('i vale depois do incremento: ',i);
end.
```

Execução deste programa

```
marcos@tuco ~ $ fpc incremento.pas
Free Pascal Compiler version 3.0.4+dfsg-18ubuntu2 [2018/08/29] for x86_64
Copyright (c) 1993-2017 by Florian Klaempfl and others
Target OS: Linux for x86-64
Compiling incremento.pas
Linking incremento
/usr/bin/ld.bfd: aviso: link.res contém seções de saída; você se esqueceu de -T
9 lines compiled, 0.1 sec
marcos@tuco ~ $ ./incremento
i valia antes do incremento: 2
i vale depois do incremento: 3
marcos@tuco ~ $ cd ..
```

Problema: ler dois números e imprimir a soma deles

- apresentamos três programas que resolvem este problema

Somando dois, versão 1

```
program soma2;
var a, b: longint;

begin
    read (a,b);
    writeln (a+b);
end.
```

Somando dois, versão 2

```
program soma2_v2;
var a, b: longint;

begin
    write ('entre com o valor de a: ');
    read (a);
    write ('entre com o valor de b: ');
    read (b);
    writeln (a,' + ', b,' = ', a+b);
end.
```

Somando dois, versão 3

```
program soma2_v3;
var a, b, soma: longint;

begin
     read (a,b);
     soma:= a + b;
     write (soma);
     writeln;
end.
```

Expressões booleanas

- permitem a realização de cálculos que resultam em um valor verdadeiro ou falso
- existe o tipo `boolean` para variáveis que podem receber valores booleanos
- por exemplo:
  - `var OK, NotOk: boolean;`
  - `OK:= true;`
  - `NotOK:= false;`
- as expressões booleanas servem para permitir que se façam comparações ou testes que resultam em valor verdadeiro ou falso.

Tipos de operadores booleanos

- operadores relacionais
  - >: estritamente maior
  - >=: maior ou igual
  - <: estritamente menor
  - <=: menor ou igual
  - =: igual
  - <>: diferente
- operadores lógicos
  - `AND`: é o E lógico, operador binário
  - `OR`: é o OU lógico operador binário
  - `NOT`: é o NÃO lógico, operador unário

Exemplos

- `a > 0`
- `a + b = x + y`
- `sqrt(b*b + 4 * a * c) >= 0`
- `(a >= 3) AND (a <= 5)`
- `NOT ((a >= 3) AND (a <= 5))`

Prioridade dos operadores booleanos

- `( )`: os parênteses têm precedência maior
- `NOT`: os operadores unários vêm depois (igual a do - unário)
- `AND`: operadores multiplicativos vêm depois (igual a do *)
- `OR`: operadores aditivos vêm depois (igual a do +)
- `> >= < <= = <>`: operadores relacionais vêm por último
- o `free pascal` escolhe qual faz primeiro dentro da mesma prioridade

Expressões booleanas

- é importante ler o material complementar sobre como é a maneira correta de se escrever expressões booleanas
- aqui só mostramos o básico

Exemplo: computar $3 \leq a \leq 5$

- Correto: `(a >= 3) AND (a <= 5)`
- Errado: `3 <= a <= 5`
- Errado também: `a >= 3 AND a <= 5`
  - a maior prioridade é o AND
  - o computador vai tentar computar primeiro: `3 AND a`
  - mas os operandos do `AND` devem ser booleanos

Exemplo de atribuição booleana

```
program exemplo_atribuicao_booleana;
var a, b, c: real;
    existe_solucao_real: boolean;

begin
    read (a, b, c);
    existe_solucao_real:= b*b - 4 * a * c >= 0;
end.
```

- sabemos que uma equação do segundo grau tem soluções reais quando o discriminante é positivo ou nulo
- a variável `existe_solucao_real` valerá `true` se a comparação >= resultar em verdadeiro
- valerá `false` em caso contrário
- nas próximas aulas veremos como usar isso melhor

Exercícios

- fazer os exercícios de 1 a 12 do livro [1] contidos na seção 5.10
- fazer a leitura dos capítulos 2 e 3 do livro [1], é importante como motivação e explicação de alguns conceitos relevantes
- opcionalmente, fazer a leitura do capítulo 4 do livro, ajuda bastante a entender a noção de variável, além de mostrar como o computador funciona em um nível mais próximo do real, eliminando-se completamente os aspectos de hardware

`http://www.inf.ufpr.br/cursos/ci055/livro_alg1.pdf`

Fim do segundo tópico

- o conteúdo desta aula está no livro no capítulo 5, seções de 5.5 e 5.6
- na próxima aula veremos outros comandos importantes

Licença

- Slides feitos em LaTeX usando beamer
- Licença

  Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.`http://creativecommons.org/licenses/by-nc-nd/2.5/br/`

  Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.`http://creativecommons.org/licenses/by-nc-nd/2.5/br/`