PICAT: Uma Linguagem de Programação Multiparadigma

, Claudio Cesar de Sá, Miguel Alfredo Nunes, Jeferson L. R. Souza

miguel.nunes@edu.udesc.br

jeferson.souza@udesc.br

claudio.sa@udesc.br

Departamento de Ciência da Computação – DCC
Centro de Ciências e Tecnológias – CCT
Universidade do Estado de Santa Catarina – UDESC

22 de abril de 2019

Contribuições

- Alexandre Gonçalves;
- João Herique Faes Battisti;
- Paulo Victor de Aguiar;
- Rogério Eduardo da Silva;
- Hakan Kjellerstrand – (`http://www.hakank.org/picat/`)
- Neng-Fa Zhou – (`http://www.picat-lang.org/`)
- Outros anônimos que auxiliaram na produção deste documento;

- Em projetos de linguagens de programação há dois tipos verificação do tipo de dados: **estática** e **dinâmica**

Tipos de Dados e Variáveis – I

- Em projetos de linguagens de programação há dois tipos verificação do tipo de dados: **estática** e **dinâmica**
- A verificação de tipos dados estática em *tempo de compilação*.

Tipos de Dados e Variáveis – I

- Em projetos de linguagens de programação há dois tipos verificação do tipo de dados: **estática** e **dinâmica**
- A verificação de tipos dados estática em *tempo de compilação*.
- Enquanto a dinâmica em *tempo de execução*.

Tipos de Dados e Variáveis – I

- Em projetos de linguagens de programação há dois tipos verificação do tipo de dados: **estática** e **dinâmica**
- A verificação de tipos dados estática em *tempo de compilação*.
- Enquanto a dinâmica em *tempo de execução*.
- Linguagens fortemente tipadas, tais como C, Java e Pascal, exigem que o tipo do dado (conteudo) seja do mesmo tipo da variável ao qual este valor será atribuído. Tudo isto é pré-definido durante a fase da *compilação*.

- Nas linguagens interpretadas, com uma máquina virtual, esta definição é feita durante a *execução* do programa

- Nas linguagens interpretadas, com uma máquina virtual, esta definição é feita durante a *execução* do programa
- Prós e contras para o que é melhor, a discussão fica de lado neste momento

- Nas linguagens interpretadas, com uma máquina virtual, esta definição é feita durante a *execução* do programa
- Prós e contras para o que é melhor, a discussão fica de lado neste momento
- Picat até o momento tem a tipagem **dinâmica**

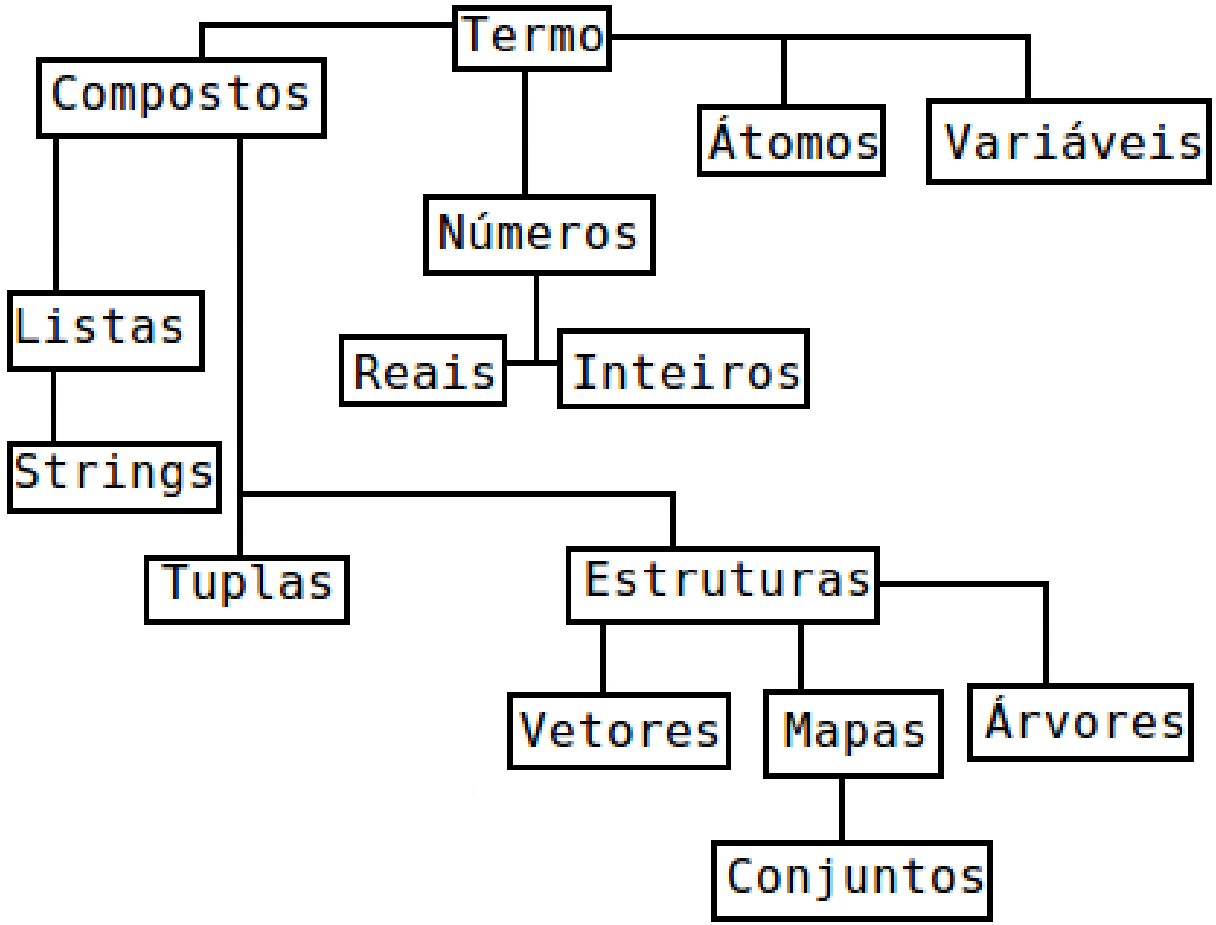

Figura 1: Hierarquia dos Tipos de Dados

Termos

- Em Picat, variáveis e valores são *genericamente* chamados de *termos*

Termos

- Em Picat, variáveis e valores são *genericamente* chamados de *termos*
- Os valores são subdivididos em duas categorias, números e valores compostos
- Os números, por suas vez, podem ser inteiros ou reais, e valores compostos podem ser listas e estruturas

Átomos

- Átomos são constantes simbólicas, podendo ser delimitados ou não, por aspas simples.
- Carácteres são representados por átomos de comprimento 1.
- Átomos não delimitados por aspas simples, **nunca** começam com uma letra maiúscula, nem número ou *underscore*.

Átomos

- Átomos são constantes simbólicas, podendo ser delimitados ou não, por aspas simples.
- Carácteres são representados por átomos de comprimento 1.
- Átomos não delimitados por aspas simples, **nunca** começam com uma letra maiúscula, nem número ou *underscore*.

Exemplos

```
x    x_1    '_'    '\\'    'a\'b\n'    '_ab'    '$%'
```

Números se dividem em:

- **Inteiro:** Inteiros podem ser representados por números binários, octais, decimais ou hexadecimais.

Exemplos

`12_345`	12345 em notação decimal, usando _ como separador
`0b100`	4 em notação binária
`0o73`	59 em notação octal
`0xf7`	247 em notação hexadecimal

O **underscore** é ignorado pelo compilador e o interpretador.

- **Real:** Números reais são compostos por um parte inteira, um ponto, seguido por uma fração decimal, ou um expoente.
- Se existe uma parte inteira em um número real então ela deve ser seguida por uma fração ou um expoente. Isso é necessário para distinguir um número real de um número inteiro.

Exemplos

```
12.345    0.123    12-e10    0.12E10
```

Compostos

- Termos compostos podem conter mais de um valor ao mesmo tempo.

Compostos

- Termos compostos podem conter mais de um valor ao mesmo tempo.
- Termos compostos são acessados pela notação de índice, começando a partir de 1 e indo até N, onde N é o tamanho deste termo.

Compostos

- Termos compostos podem conter mais de um valor ao mesmo tempo.
- Termos compostos são acessados pela notação de índice, começando a partir de 1 e indo até N, onde N é o tamanho deste termo.
- Se dividem em Listas e Estruturas.

Listas são agrupamentos de valores quaisquer sem ordem e sem tamanho pré-definido. Seu tamanho não é armazenado na memória, sendo necessário recalcular sempre que necessário seu uso. Listas são encapsuladas por colchetes.

Exemplos

[1,2,3,4,5] [a,b,32,1.5,aaac] ["string",14,22]

Há uma seção dedicada a esta poderosa estrutura de dados!

Strings são listas especiais que contém somente carácteres. *Strings* podem ser inicializadas como uma sequência de carácteres encapsulados por aspas duplas, ou como uma sequência de carácteres dentro colchetes separados por vírgulas.

Exemplos

"Hello" "World!" "\n" [o,l,a," ",m,u,n,d,o]

Tuplas

- Tuplas é um conjunto de termos não-ordenados, podendo ser acessados por notação de índice assim como listas.
- Tuplas são estáticas, ou seja, os termos contidos em uma tupla não podem ser alterados, assim como não podem ser adicionados ou removidos termos de tuplas.
- Tuplas são encapsuladas por parênteses e seus termos são separados por vírgulas.

Exemplos

(1,2,3,4,5) (a,b,32,1.5,aaac) ("string",14,22)

Em geral, usamos as tuplas dentro de listas.

Estruturas são termos especiais que podem ser definidos pelo usuário. Estruturas tomam a seguinte forma:

$$\$\mathtt{s}(t_1, \ldots, t_n)$$

Onde *'s'* é um átomo que denomina a estrutura, cada *'t_i'* é um de seus termos, e *'n'* é a aridade ou tamanho da estrutura.

Exemplo

$ponto(1,2) $pessoa(jose, "123.456.789.00", "1.234.567")

Estruturas são termos especiais que podem ser definidos pelo usuário. Estruturas tomam a seguinte forma:

$$\$\mathtt{s}(t_1, \ldots, t_n)$$

Onde *'s'* é um átomo que denomina a estrutura, cada *'t_i'* é um de seus termos, e *'n'* é a aridade ou tamanho da estrutura.

Exemplo

$ponto(1,2) $pessoa(jose, "123.456.789.00", "1.234.567")

Temos 4 outras estruturas que não usam o símbolo $, são elas:

[fragile, allowframebreaks=0.9]
Vetores ou *arrays* são estruturas especiais do tipo:

$$\{t_1, \ldots, t_n\}$$

[fragile, allowframebreaks=0.9]
Vetores ou *arrays* são estruturas especiais do tipo:

$$\{t_1, \ldots, t_n\}$$

- Vetor é um conjunto ordenado de tamanho n, delimitado por '{}'.
- Vetores tem comportamentos análogo às listas, tanto é que quase todas as funções de listas são sobrecarregadas para vetores.
- A diferença entre vetores e listas é que vetores tem um tamanho constante.
- Vetores são muito práticos quando se manipula matrizes na entrada

Exemplos

{1,2,3,4,5} {a,b,32,1.5,aaac} {"string",14,22}

- **Mapas** são estruturas especiais que são conjuntos de relações do tipo `chave-valor`.
- **Conjuntos** são sub-tipos de mapas onde todos as chaves estão relacionadas com o átomo `not_a_value`.
- *Heaps* são árvores binárias completas representadas como vetores. Árvores podem ser do tipo *máximo*, onde o maior valor está na raiz, ou *mínimo*, onde o menor valor esta na raiz.

Variáveis I

- Picat é uma linguagem de Tipagem Dinâmica, ou seja, o tipo de uma variável é validado durante a execução do programa
- Isto é, quando uma variável é criada, seu tipo não é instanciado
- Variáveis são análogas as da matemática, são símbolos que *seguram* ou representam um valor
- Ao contrário de variáveis em linguagens imperativas, variáveis em Picat não são endereços simbólicos de locais na memória
- Uma variável é dita *livre* (*free*) se não contém nenhum valor, e dita *instanciada* (*bound*) se ela contém um valor
- Uma vez que uma variável é instanciada, ela permanece com este valor na execução atual
- Por isso, diz-se que variáveis em Picat são de *atribuição única*

Variáveis II

- O nome de variáveis devem sempre ser iniciado com letras **maiúsculas** ou com o caráctere ***underscore (_)***, porém;
  - Variáveis cujo nome é unicamente um caractere _ são chamadas de *variáveis anônimas*.
  - As *variáveis anônimas* podem receber qualquer valor não os guardam durante a execução do programa;
  - Num mesmo programa, podem existir diversas variáveis anônimas, instanciadas durante a execução do mesmo

Há dois modos de definir valores às variáveis:

Unficação e Atribuição

Há dois modos de definir valores às variáveis:

- Unificação usa o operador '='
- Atribuição usa o operador ':='

Unificação

- A **Unificação** é uma operação que instancia uma variável a um termo, substituindo toda ocorrência dessa variável pelo valor

Unificação

- A **Unificação** é uma operação que instancia uma variável a um termo, substituindo toda ocorrência dessa variável pelo valor
- Caso ocorra uma instância que não falhe nenhuma situação a variável é unificada à este termo ou padrão.

Unificação

- A **Unificação** é uma operação que instancia uma variável a um termo, substituindo toda ocorrência dessa variável pelo valor
- Caso ocorra uma instância que não falhe nenhuma situação a variável é unificada à este termo ou padrão.
- Uma instanciação é indefinida até que se encontre um valor que possa ser unificada a uma variável.

Unificação

- A **Unificação** é uma operação que instancia uma variável a um termo, substituindo toda ocorrência dessa variável pelo valor
- Caso ocorra uma instância que não falhe nenhuma situação a variável é unificada à este termo ou padrão.
- Uma instanciação é indefinida até que se encontre um valor que possa ser unificada a uma variável.
- Termos são ditos unificáveis se são idênticos ou podem ser tornados idênticos instanciado variáveis nos termos.

Exemplo

```
Picat> X = 1
X = 1
Picat> $f(a,b) = $f(a,b)
yes
Picat> [H|T] = [a,b,c]
H = a
T = [b,c]
Picat> $f(X,b) = $f(a,Y)
X = a
Y = b
Picat> bind_vars({X,Y,Z},a)
Picat> X = $f(X)
```

Cuidar neste último caso, há um laço infinito nesta chamada!

Atribuição

- A **atribuição** simula a atribuição em linguagens imperativas

Atribuição

- A **atribuição** simula a atribuição em linguagens imperativas
- Permite que variáveis assumam novos valores durante a execução do programa

Atribuição

- A **atribuição** simula a atribuição em linguagens imperativas
- Permite que variáveis assumam novos valores durante a execução do programa
- O escopo da atribuição da variável é local e volátil

Atribuição

- A **atribuição** simula a atribuição em linguagens imperativas
- Permite que variáveis assumam novos valores durante a execução do programa
- O escopo da atribuição da variável é local e volátil
- Na unificação, uma nova variável temporária é criada afim de substituir um valor atribuído ou outra variável.

Exemplo

```
teste => X = 0, X := X + 1,  X := X + 2, write(X).
```

- Neste exemplo X é unificado a 0.
- Em seguida, há uma atribuição X a $X + 1$, porém X já foi unificado a um termo.
- Então, outras operações devem ser feitas para que esta atribuição seja possível.
- Nesse caso, o compilador cria uma variável temporária, $X1$ por exemplo, e unifica com $X + 1$. Cada vez que X for instanciado, o compilador/programa atualiza em $X1$.
- O mesmo ocorre na atribuição $X1 := X1 + 2$, neste caso uma outra variável temporária é criada, por exemplo $X2$, e o processo se repetido.

Exemplo

Portanto, estas atribuições sucessivas são compiladas como:

Exemplo

Portanto, estas atribuições sucessivas são compiladas como:

```
test => X = 0, X1 = X + 1, X2 = X1 + 2, write(X2).
```

Exemplos de Variáveis Válidas

X1	_	_ab
X	A	Variavel
_invalido	_correto	_aa

⇒ Relembrando, um nome de variável é válido se começa com letra **maiúscula** ou _

Exemplos de Variáveis Inválidas

1_Var	variável	valida
23	"correto	‘termo
!numero	$valor	#comum

Tabela 1: Operadores Aritméticos em Ordem de Precedência

X `**` *Y*	Potenciação
X `*` *Y*	Multiplicação
X `/` *Y*	Divisão, resulta em um real
X `//` *Y*	Divisão de Inteiros, resulta em um inteiro
X `mod` *Y*	Resto da Divisão
X `+` *Y*	Adição
X `-` *Y*	Subtração
Inicio `..` *Passo* `..` *Fim*	Uma série (lista) de números com um passo
Inicio `..` *Fim*	Uma série (lista) de números com passo 1

Tabela 2: Tabela de Operadores Completa em Ordem de Precedência

Ops Aritméticos	Ver Tabela **??**
`++`	Concatenação de Listas/Vetores
`= :=`	Unificação e Atribuição
`== =:=`	Equivalência e Equivalência Numérica
`!= !==`	Não Unificável e Diferença
`< =< <=`	Menor que
`> >=`	Maior que
`in`	Contido em
`not`	Negação Lógica
`, &&`	Conjunção Lógica
`; ||`	Disjunção Lógica

Operadores de Termos Não-Compostos

- **Equivalência**(==): compara se dois termos são iguais.
  No caso de termos compostos, eles são ditos equivalentes se todos os termos contidos em si são equivalentes. O compilador considera termos de tipos diferentes como totalmente diferentes, portanto a comparação $1.0 == 1$ seria avaliada como falsa, mesmo que os valores sejam iguais. Nesses casos, usa-se a *Equivalência Numérica*.

Operadores de Termos Não-Compostos

- **Equivalência**(==): compara se dois termos são iguais.
  No caso de termos compostos, eles são ditos equivalentes se todos os termos contidos em si são equivalentes. O compilador considera termos de tipos diferentes como totalmente diferentes, portanto a comparação $1.0 == 1$ seria avaliada como falsa, mesmo que os valores sejam iguais. Nesses casos, usa-se a *Equivalência Numérica*.
- **Equivalência Numérica**(=:=): Compara se dois números são o mesmo valor. Deve ser usada com termos que sejam números.

Operadores de Termos Não-Compostos

- **Diferença**(!==): compara se dois termos são diferentes, isto é, a negação da equivalência.

Operadores de Termos Não-Compostos

- **Diferença**(!==): compara se dois termos são diferentes, isto é, a negação da equivalência.
- **Não-Unificável**(!=): Verifica se dois termos não são unificáveis. Termos são ditos unificáveis se são idênticos ou podem ser tornados idênticos instanciando variáveis destes termos.

Exemplos

1. $a == a, \quad [1,2,3] == [1,2,3], \quad Var1 == Var2$

Exemplos

1. $a == a$, $[1, 2, 3] == [1, 2, 3]$, $Var1 == Var2$
   yes, *yes*, Depende dos valores (padrão *no*)

Exemplos

1. $a == a$, $[1, 2, 3] == [1, 2, 3]$, $Var1 == Var2$
   yes, *yes*, Depende dos valores (padrão *no*)
2. $1.0 == 1$

Exemplos

1. $a == a$, $[1, 2, 3] == [1, 2, 3]$, $Var1 == Var2$
   yes, *yes*, Depende dos valores (padrão *no*)
2. $1.0 == 1$
   no

Exemplos

1. $a == a$, $[1,2,3] == [1,2,3]$, $Var1 == Var2$
   yes, *yes*, Depende dos valores (padrão *no*)
2. $1.0 == 1$
   no
3. $1.0 =:= 1$, $1.2 =:= 1$

Exemplos

1. $a == a$, $[1,2,3] == [1,2,3]$, $Var1 == Var2$
   yes, *yes*, Depende dos valores (padrão *no*)
2. $1.0 == 1$
   no
3. $1.0 =:= 1$, $1.2 =:= 1$
   yes, *no*

Exemplos

1. $a == a$, $[1,2,3] == [1,2,3]$, $Var1 == Var2$
   yes, *yes*, Depende dos valores (padrão *no*)
2. $1.0 == 1$
   no
3. $1.0 =:= 1$, $1.2 =:= 1$
   yes, *no*
4. $1.0 \; !== 1$, $Var3 \; !== Var4$

Exemplos

1. $a == a$, $[1,2,3] == [1,2,3]$, $Var1 == Var2$
   yes, *yes*, Depende dos valores (padrão *no*)
2. $1.0 == 1$
   no
3. $1.0 =:= 1$, $1.2 =:= 1$
   yes, *no*
4. $1.0 \; !== 1$, $Var3 \; !== Var4$
   yes, Depende dos valores (padrão *yes*)

Exemplos

1. $a == a$, $[1,2,3] == [1,2,3]$, $Var1 == Var2$
   yes, *yes*, Depende dos valores (padrão *no*)
2. $1.0 == 1$
   no
3. $1.0 =:= 1$, $1.2 =:= 1$
   yes, *no*
4. $1.0 \; !== 1$, $Var3 \; !== Var4$
   yes, Depende dos valores (padrão *yes*)
5. $1.0 \; != 1$, $aa \; ! = bb$, $Var1 \; != Var5$

Exemplos

1. $a == a$, $[1,2,3] == [1,2,3]$, $Var1 == Var2$
   yes, *yes*, Depende dos valores (padrão *no*)
2. $1.0 == 1$
   no
3. $1.0 =:= 1$, $1.2 =:= 1$
   yes, *no*
4. $1.0 \; !== 1$, $Var3 \; !== Var4$
   yes, Depende dos valores (padrão *yes*)
5. $1.0 \; != 1$, $aa \; ! = bb$, $Var1 \; != Var5$
   yes, *yes*, *no*

Alguns Operadores de Termos Compostos

- **Concatenação** (++): concatena duas listas ou vetores. O termo da esquerda é a primeira parte lista e a segundo a parte final da lista resultante.

Alguns Operadores de Termos Compostos

- **Concatenação** (++): concatena duas listas ou vetores. O termo da esquerda é a primeira parte lista e a segundo a parte final da lista resultante.
- **Separador** (`H | T`): separa uma lista L em seu primeiro termo H, chamado de *cabeça* (em inglês *Head*), e o resto da lista T, chamado de *cauda* (em inglês *Tail*).

Na seção de listas este assunto é retomado

Alguns Operadores de Termos Compostos

- **Iterador** (`X in L`): itera X no termo composto L, instanciando um termo não-composto X aos termos contidos em L.

Alguns Operadores de Termos Compostos

- **Iterador** (`X in L`): itera X no termo composto L, instanciando um termo não-composto X aos termos contidos em L.
- **Sequência** (`Inicio..Passo..Fim`): gera uma lista ou vetor, começando (inclusivamente) em *Inicio* incrementando por *Passo* e parando (inclusivamente) em *Fim*. Se *Passo* for omitido, este é automaticamente atribuído 1.

Na seção de listas este assunto é retomado

Operadores de Termos Compostos – Exemplos

1. $[1,2,3] ++ [4,5,6]$, $[] ++ [1,2,3]$, $[] ++ []$

Operadores de Termos Compostos – Exemplos

1. $[1,2,3] ++ [4,5,6]$, $[] ++ [1,2,3]$, $[] ++ []$
   $[1,2,3,4,5,6]$, $[1,2,3]$, $[]$

Operadores de Termos Compostos – Exemplos

1. $[1,2,3] ++ [4,5,6]$, $[] ++ [1,2,3]$, $[] ++ []$
   $[1,2,3,4,5,6]$, $[1,2,3]$, $[]$
2. $L = [1,2,3]$, $[H|T] = L$

Operadores de Termos Compostos – Exemplos

1. $[1,2,3] ++ [4,5,6]$, $[] ++ [1,2,3]$, $[] ++ []$
   $[1,2,3,4,5,6]$, $[1,2,3]$, $[]$
2. $L = [1,2,3]$, $[H|T] = L$
   $L = [1,2,3]$

Operadores de Termos Compostos – Exemplos

1. $[1,2,3] ++ [4,5,6]$, $[] ++ [1,2,3]$, $[] ++ []$
   $[1,2,3,4,5,6]$, $[1,2,3]$, $[]$
2. $L = [1,2,3]$, $[H|T] = L$
   $L = [1,2,3]$
   $H = 1$

Operadores de Termos Compostos – Exemplos

1. $[1,2,3] ++ [4,5,6]$, $[] ++ [1,2,3]$, $[] ++ []$
   $[1,2,3,4,5,6]$, $[1,2,3]$, $[]$
2. $L = [1,2,3]$, $[H|T] = L$
   $L = [1,2,3]$
   $H = 1$
   $T = [2,3]$

Operadores de Termos Compostos – Exemplos

1. $[1,2,3] ++ [4,5,6]$, $[] ++ [1,2,3]$, $[] ++ []$
   $[1,2,3,4,5,6]$, $[1,2,3]$, $[]$
2. $L = [1,2,3]$, $[H|T] = L$
   $L = [1,2,3]$
   $H = 1$
   $T = [2,3]$
3. $\mathit{foreach}(X \ \mathit{in}\ [1,2,3])\ \ \mathit{printf}("\%w ", X)\ \ \mathit{end}$

Operadores de Termos Compostos – Exemplos

1. $[1,2,3] ++ [4,5,6]$, $[] ++ [1,2,3]$, $[] ++ []$
   $[1,2,3,4,5,6]$, $[1,2,3]$, $[]$
2. $L = [1,2,3]$, $[H|T] = L$
   $L = [1,2,3]$
   $H = 1$
   $T = [2,3]$
3. $\mathit{foreach}(X \ \mathit{in}\ [1,2,3])\ \ \mathit{printf}("\%w ", X)\ \ \mathit{end}$
   1 2 3

Operadores de Termos Compostos – Exemplos

1. $[1,2,3] ++ [4,5,6]$, $[] ++ [1,2,3]$, $[] ++ []$
   $[1,2,3,4,5,6]$, $[1,2,3]$, $[]$
2. $L = [1,2,3]$, $[H|T] = L$
   $L = [1,2,3]$
   $H = 1$
   $T = [2,3]$
3. $foreach(X\ in\ [1,2,3])\ printf("\%w", X)\ end$
   1 2 3
4. $X = 1..10$, $Y = 0..2..20$, $Z = 10..-1..1$

Operadores de Termos Compostos – Exemplos

1. $[1,2,3] ++ [4,5,6], \quad [] ++ [1,2,3], \quad [] ++ []$
   $[1,2,3,4,5,6], \quad [1,2,3], \quad []$
2. $L = [1,2,3], \ [H|T] = L$
   $L = [1,2,3]$
   $H = 1$
   $T = [2,3]$
3. $foreach(X \ in \ [1,2,3]) \ \ printf("\%w", X) \ \ end$
   1 2 3
4. $X = 1..10, \ Y = 0..2..20, \ Z = 10..-1..1$
   $X = [1,2,3,4,5,6,7,8,9,10]$

Operadores de Termos Compostos – Exemplos

1. $[1,2,3] ++ [4,5,6], \quad [] ++ [1,2,3], \quad [] ++ []$
   $[1,2,3,4,5,6], \quad [1,2,3], \quad []$
2. $L = [1,2,3], \ [H|T] = L$
   $L = [1,2,3]$
   $H = 1$
   $T = [2,3]$
3. $foreach(X \ in \ [1,2,3]) \ \ printf("\%w", X) \ \ end$
   1 2 3
4. $X = 1..10, \ Y = 0..2..20, \ Z = 10..-1..1$
   $X = [1,2,3,4,5,6,7,8,9,10]$
   $Y = [0,2,4,6,8,10,12,14,16,18,20]$

Operadores de Termos Compostos – Exemplos

1. $[1,2,3] ++ [4,5,6], \quad [] ++ [1,2,3], \quad [] ++ []$
   $[1,2,3,4,5,6], \quad [1,2,3], \quad []$
2. $L = [1,2,3], \ [H|T] = L$
   $L = [1,2,3]$
   $H = 1$
   $T = [2,3]$
3. $foreach(X \ in \ [1,2,3]) \ \ printf("\%w", X) \ \ end$
   1 2 3
4. $X = 1..10, \ Y = 0..2..20, \ Z = 10..-1..1$
   $X = [1,2,3,4,5,6,7,8,9,10]$
   $Y = [0,2,4,6,8,10,12,14,16,18,20]$
   $Z = [10,9,8,7,6,5,4,3,2,1]$

Reflexões

- O conteúdo desta parte do curso pode ser complementado com a **Videoaula 02: Tipos de Dados do PICAT**
  `https://www.youtube.com/watch?v=7fPKPd0ZDnc`