ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO
Departamento de Engenharia de Computação e Sistemas Digitais

PCS3110- Algoritmos e Estruturas de Dados para Engenharia Elétrica

Pseudocódigo

Os algoritmos apresentados em PCS3110 seguem o pseudocódigo do livro do Cormen[1], que é o livro texto da disciplina. Os comandos são similares aos de linguagens de programação como C, C++, Java e Python.

Este documento descreve a sintaxe e a semântica desse pseudocódigo.

Regras gerais

- Não é necessário colocar um caractere de fim de comando (o ";" em C).
    - O ";" pode ser usado para permitir que mais de um comando seja colocado em uma mesma linha (ou seja, como um *separador*). Por exemplo, na linha 2 do código faz-se duas atribuições: 2 à variável "b" e 3 à variável "c".

```
1  a = 1
2  b = 2; c = 3
```

- A indentação indica a estrutura de bloco. Ou seja, a tabulação indica que um comando está *aninhado* a um outro comando – não existe um símbolo de início de bloco (o "{" em C) e um símbolo de fim de bloco (o "}" em C). Por exemplo, no código a seguir o corpo da condição na linha 1 compreende as linhas 2 e 3, mas não a linha 4 (ela está fora do `if`).

```
1  if x ≥ 5
2    y = 3
3    z = 4
4  print(x)
```

Variáveis

- Variáveis são locais a uma função.
- Variáveis não precisam ser declaradas.
    - O tipo da variável é definido quando um valor é atribuído a ela[2].

[1] Cormen, T.; Leiserson, C.; Rivest, R.; Stein, C. *Algoritmos: Teoria e Prática*, tradução da 3ª. Edição, Elsevier - Campus Ed., 2013.

- O "=" é usado para atribuição.
    - É possível fazer uma atribuição múltipla, como no código abaixo: nele o valor 1 é atribuído tanto à variável "x" quando à "y".

```
1  x = y = 1
```

Condição

- Uma condição tem a sintaxe:

```
if <expressão>
  <comandos1>
else <comandos2>
```

  Onde:
    - `<expressão>`: é uma expressão que deve ser avaliada verdadeira ou falsa. É um erro sintático se a expressão for um número, um texto ou qualquer outra coisa diferente de verdadeiro ou falso.
    - `<comandos1>`: os comandos que devem ser executados caso `<expressão>` seja avaliada verdadeira.
    - `<comandos2>`: os comandos que devem ser executados caso `<expressão>` seja avaliada falsa.

  O "else" é opcional. Um exemplo de condição é apresentado no código abaixo. Se "x" for maior que 5 então 2 será atribuído a "a" e 3 a "b". Caso contrário, 10 será atribuído a "a" e 17 a "b"

```
1  if x > 5
2    a = 2
3    b = 3
4  else a = 10
5    b = 17
```

  Um exemplo com aninhamento de condições é apresentado a seguir. Se "x" for maior ou igual a 5, então 2 será atribuído a "a". Caso não seja maior ou igual a 5 e seja maior que 0 (ou seja, se for 1, 2, 3 ou 4), então 3 será atributo a "a". Caso contrário 5 será atribuído a "a" (ou seja, se for menor ou igual a 0).

```
1  if x ≥ 5
2    a = 2
3  else if x > 0
4    a = 3
5  else
6    a = 5
```

- Expressões usadas em condições e laços podem usar os seguintes operadores condicionais:

---

[2] A princípio o tipo de uma variável pode mudar, mas evitaremos escrever códigos que fazem isso.

Operador	Representação
“e” lógico	and
“ou” lógico	or
Negação	not

- Operadores condicionais binários (“e” e “ou”) têm avaliação de “curto-circuito”, ou seja, a segunda expressão é avaliada condicionalmente. No código a seguir, por exemplo, como a expressão `x == 0` é falsa, a expressão do `if` será obrigatoriamente avaliada como falsa e, portanto, a expressão `y == 1` não é avaliada (falso “e” qualquer coisa é sempre falso).

```
1 x = 1
2 y = 2
3 if x == 0 and y == 1
4   print(“Condição verdadeira”)
```

- A expressão pode usar os seguintes operadores relacionais:

Operador	Representação
Maior	>
Maior ou igual	≥
Menor	<
Menor ou igual	≤
Igual	==
Diferente	≠

Laço

- Existem 3 tipos de laço: `while`, `repeat-until` e `for`.
- O `while` permite executar comandos zero ou mais vezes. A sintaxe dele é:

```
while <expressão>
  <comandos>
```

Onde:

- `<expressão>`: é uma expressão que deve ser avaliada verdadeira ou falsa.
- `<comandos>`: os comandos que devem ser executados *enquanto* a expressão for avaliada verdadeira.

O código a seguir apresenta um laço executado enquanto “x” for maior ou igual a `0`. Como ele é inicialmente `1` e é decrementado em `1` em cada iteração, o texto “ok” será impresso 2 vezes.

```
1  x = 1
2  while x ≥ 0
3    print(“ok”)
```

```
4    x = x - 1
```

- O `repeat-until` permite executar comandos uma ou mais vezes. A sintaxe dele é:

```
repeat
  <comandos>
until <expressão>
```

Onde:

- `<expressão>`: é uma expressão que deve ser avaliada verdadeira ou falsa.
- `<comandos>`: os comandos que devem ser executados *até que* a expressão seja avaliada verdadeira.

O código a seguir apresenta um laço executado até que "x" seja igual a `0`. Como ele é inicialmente `2` e é decrementado em `1` em cada iteração, o texto "ok" será impresso `2` vezes.

```
1  x = 2
2  repeat
3    print("ok")
4    x = x - 1
5  until x == 0
```

- O `for` permite iterar de um valor inicial até um valor final. Ele é bastante usado para iterar sobre vetores. A sintaxe *básica* dele é:

```
for <variavel> = <início> to <fim>
  <comandos>
```

Onde:

- `<variável>`: variável que terá o valor atribuído em cada iteração do laço. O laço será executado de `<início>` até `<fim>` (inclusive). Portanto na primeira iteração `<variável>` terá o valor `<início>` atribuído. A cada iteração o valor em `<variável>` será incrementado em `1`, até `<fim>`. Ou seja, a última iteração executada é quando `<variável> == <fim>`. Quando o laço termina, a `<variável>` terá valor `<fim> + 1`.
- `<comandos>`: os comandos que devem ser executados.

O código a seguir apresenta um `for` executado de `1` a `2`. Na primeira iteração a variável "i" terá o valor `1`, portanto sendo impresso "1". A variável será incrementada em `1`, ficando com o valor 2. O comando da linha `2` será executado novamente, imprimindo "2". A variável "i" será novamente incrementada, ficando em 3. Como 3 é maior que 2, o laço então termina.

```
1  for i = 1 to 2
2    print(i)
```

Esse `for` pode ser expresso como um `while`, apresentado abaixo.

```
1  i = 1
2  while i ≤ 2
3    print(i)
4    i = i + 1
```

- Um `for` pode decrementar o valor da variável de controle. Para isso deve-se escrever "`downto`" ao invés de "`to`" como no exemplo abaixo. Nesse caso será impresso, em ordem, "2" e "1".

```
1  for i = 2 downto 1
2    print(i)
```

Funções

- Uma função tem a seguinte sintaxe:

  <Nome da função>(<lista de parâmetros>)
  <comandos>

  Onde:
  - <Nome da função>: o nome da função.
  - <lista de parâmetros>: uma lista (que pode ser vazia) com os parâmetros da função. Caso a lista possua mais de um parâmetro, eles devem ser separados por vírgulas. Na lista de parâmetros não é preciso informar o tipo da variável.

  Por exemplo, abaixo é apresentada uma função que retorna a soma de três números, "a", "b" e "c".

```
   Soma(a, b, c)
1    return a + b + c
2
```

- O comando `return` transfere imediatamente o controle para quem chamou a função. Esse comando pode retornar `0` ou mais valores para quem chamou a função. Por exemplo, no código abaixo a função "Vizinhos" retorna dois valores: "a – 1" e "a + 1"; na função "Sucessor" é retornado apenas 1 valor: "a + 1".

```
   Vizinhos(a)
1    return a - 1, a + 1
```

```
   Sucessor(a)
1    return a + 1
```

  - Caso a função retorne mais de um valor, deve-se usar a ", " para separar as variáveis que receberão o valor. Por exemplo, no código abaixo se atribui a "a" o primeiro valor retornado por "Vizinhos" (no caso 2) e a "b" o segundo valor (no caso 4).

```
1  a, b = Vizinhos(3)
```

- Caso a função não tenha retorno, não é necessário colocar "`return`" na última linha. Ele é implícito.
- Parâmetros são passados por valor, ou seja, a função recebe uma *cópia* do valor. Quando dados compostos são passados como parâmetros, uma cópia do ponteiro é passada. Portanto no exemplo abaixo, a linha 3 do código à esquerda imprimirá "Jose" na saída.

```
1  Seja P uma nova pessoa
2  X(P)
3  print(P.nome)
```

```
X(P)
1    P.nome = "Jose"
```

Comentários

- "//" é usado para comentar o restante da linha. Por exemplo no código a seguir a linha 2 está comentada.

```
1  if i == 0
2    // Acabou o processamento
3    print(i)
```

Dados compostos

- Dados compostos são organizados em objetos, os quais possuem atributos. Atributos são variáveis internas ao objeto e são acessíveis com o "." (similar à muitas linguagens orientadas a objetos). Por exemplo, no código a seguir o valor do atributo "nome" do objeto em "A" está sendo atribuído à variável "x", enquanto que o valor em "y" está sendo atribuído ao "endereco" do objeto em "E".

```
1  x = A.nome
2  E.endereco = y
```

  - Objetos podem ter como atributo outros dados compostos. Dessa forma é possível *cascatear* o acesso a atributos. Por exemplo, no código abaixo o objeto em "P" possui um atributo "endereço" e esse atributo "endereço" possui um atributo "rua". Portanto "Luciano Gualberto" está sendo atribuído à "rua" do "endereço" do objeto em "P".

```
1  P.endereço.rua = "Luciano Gualberto"
```

- A criação de um objeto possui uma sintaxe específica:

  `seja <variáveis> um novo <tipo do objeto>`

  Onde:
  - `<variáveis>`: nome das variáveis que guardarão os objetos criados.
  - `<tipo do objeto>`: é o tipo do objeto criado. Por simplicidade não há uma definição formal do tipo do objeto.

Por exemplo, no código a seguir a linha 1 cria um novo objeto lista enquanto que na linha 2 são criadas duas pessoas (colocadas em P e Q).

```
1  seja L uma nova lista
2  seja P e Q novas pessoas
```

- Variáveis representando dados compostos são ponteiros para o objeto. Portanto, no exemplo do código abaixo a atribuição da linha 3 faz com que "P" e "Q" apontem para o mesmo objeto pessoa. Como na linha 4 o nome do objeto é alterado para "Antonio", na linha 5 é impresso "Antonio".

```
1  seja P uma nova pessoa
2  P.nome = "Jose"
3  Q = P
4  Q.nome = "Antonio"
5  print(P.nome)
```

- Para representar que uma variável não aponta para um objeto usaremos o valor `NIL`.

Vetores

- Vetores são dados compostos. O tamanho de um vetor pode obtido pelo atributo "tamanho". Por exemplo, o código a seguir imprime todos os elementos de um vetor "V" (é impresso o elemento em 1 até, inclusive, o elemento em `V.tamanho`).

```
1  for i = 1 to V.tamanho
2    print(V[i])
```

- Vetores *normalmente* se iniciam na posição 1.
- Não é possível redimensionar o tamanho de um vetor.
- A criação de um vetor é feita com a seguinte sintaxe:

```
seja <variável>[<inicio>..<fim>] um novo vetor
```

O `inicio` e `fim` representam os índices inicial e final do vetor, os quais devem ser separados por ".." na declaração. Por exemplo, no código abaixo são criados dois vetores "A" e "B". O vetor "A" tem índice inicial 1 e final 3 – portanto, ele pode ter valores atribuídos aos índices 1, 2 e 3. O vetor "B" tem índice inicial 1 e final 5.

```
1  sejam A[1..3] e B[1..5] novos vetores
2  A[1] = 2
3  A[2] = 13
4  A[3] = 99
```

- Elementos do vetor podem ser acessados ao colocar o índice entre "[" e "]", como nas linhas 2, 3 e 4 do código acima. Elementos do vetor podem ser usados como variáveis normais.

Conjuntos

- Conjuntos são dados compostos que possuem alguns operadores e comandos específicos.
  - Um conjunto pode ser definido como uma sequência de valores entre "{" e "}" separados por vírgula.
  - O símbolo "∪" é usado para unir dois conjuntos.
  - O símbolo "∈" é usado para avaliar se um valor pertence a um conjunto. Portanto o resultado desse operador binário é um valor verdadeiro ou falso.
  - O símbolo "∉" é usado para avaliar se um valor não pertence a um conjunto.
  - O símbolo "–" é usado para obter o conjunto com a diferença entre dois conjuntos.
  - O símbolo "∅" representa o conjunto vazio.

  Um exemplo de uso de conjuntos é apresentado a seguir. Na linha 2 é atribuído o conjunto com valores 1, 2 e 3 à variável "A"; na linha 3 o conjunto com valores 3 e 4 é atribuído a variável "B". Na linha 4 é feita a união dos conjuntos em "A" e "B", atribuindo o resultado a "C". Com isso "C" terá "{1, 2, 3, 4}". A linha 4 atribui a "C" a diferença entre "C" e o conjunto com "1", resultando em "{2, 3, 4}". Por fim, a linha 5 verificação se 1 pertence ao conjunto em "A" – o que é verdadeiro.

```
1  A = {1, 2, 3}
2  B = {3, 4}
3  C = A ∪ B
4  C = C – {1}
5  if 1 ∈ A
6    print("pertence")
```

- Um `for` também pode iterar sobre um conjunto. Nesse caso, em cada iteração um dos valores do conjunto é colocada na variável do `for`. A sintaxe é a seguinte:

```
for each <variavel> ∈ <conjunto>
  <comandos>
```

  Onde:

  - `<variável>`: A variável que guardará em cada iteração um dos valores do conjunto.
  - `<conjunto>`: o conjunto que se deseja iterar.
  - `<comandos>`: os comandos que devem ser executados em cada iteração.

  Por exemplo, o código a seguir imprime "1", "2" e "3".

```
1  N = {1, 2, 3}
2  for each i ∈ N
3    print(i)
```

Par

- Um par é um dado composto que possui duas informações. Ele é representado com os valores entre parênteses. Por exemplo, no código abaixo o conjunto "A" é formado por pares e se está unindo a "A" o par `(1, 2)`.

```
1  A = A ∪ {(1, 2)}
```

Outros detalhes

- A tabela a seguir apresenta os operadores disponíveis para números.

Descrição	Representação
Soma	+
Subtração	-
Multiplicação	*
Divisão	/
Módulo	mod
Exponenciação	**

  - No caso da divisão é possível especificar se é desejado obter o piso (entre "⌊" e "⌋") ou o teto da divisão (entre "⌈" e "⌉"). Se não especificado o resultado da divisão é truncado. Por exemplo, no código abaixo é guardado em "x" o teto da divisão de 5 por 2 (que é igual a 3)

```
1  x = ⌈5 / 2⌉
```

- A tabela a seguir apresenta alguns valores especiais.

Descrição	Representação
Verdadeiro	TRUE
Falso	FALSE
Ponteiro para nenhum valor	NIL
Número infinito	∞

- Existem alguns comandos e funções especiais:

Comando / Função	Descrição	Exemplo
`print(<texto>)`	Função que imprime algum texto na saída	`print("Saida: " i)`
`error <mensagem>`	Indica a ocorrência de um erro, o qual é descrito em uma mensagem (uma string)	`error "Valor inválido"`

- A concatenação de strings é feita sem um operador específico. Valores das variáveis são convertidos automaticamente para string para ser impresso. Por exemplo, no código abaixo é impressa uma mensagem ao usuário concatenando 3 strings.

```
1  i = 5
2  print(“A resposta correta é” i “.”)
```