REC. Nomenclàtor de pseudocodi

Introducció

1. Llegenda

2. Comentari

3. Algorisme

4. Tipus bàsics de dades

- **4.1. Valors de constants booleanes**
- **4.2. Definició de variables**
- **4.3. Definició de constants**
- **4.4. Operador d'assignació**
- **4.5. Operadors lògics**
- **4.6. Operadors de comparació**
- **4.7. Funcions de canvi de tipus**

5. Funcions i accions d'entrada i sortida

- **5.1. Variables**
- **5.2. Fitxers**

6. Altres tipus de variables

- **6.1. Definició de vectors**
- **6.2. Definició de tuples**
- **6.3. Definició de punters**

7. Estructures de control

- **7.1. Estructura condicional**
- **7.2. Estructura iterativa**

8. Funció

9. Acció

10. Classes de paràmetres d'una acció

Introducció

El nomenclàtor és una guia ràpida sobre com s'usa el pseudocodi. Ara que tot just comença el curs, potser no enteneu gaire bé el que es detalla aquí, però, a mesura que anem introduint els conceptes, aquest nomenclàtor us servirà per escriure pseudocodi amb el format adequat.

Quan aparegui pseudocodi en aquests materials, estarà escrit en `aquesta font` amb fons gris. Així, d'un cop d'ull, podrem diferenciar el codi C del pseudocodi. Hem triat aquesta font perquè sembla que està escrita a mà. El pseudocodi el podeu escriure a mà, en una llibreta, en Word..., com vulgueu, però, si està escrit a mà, no és un programa.

El pseudocodi ens ajuda a ordenar les idees sobre com abordar la solució d'un problema, però no és el programa. Ens sembla que amb aquest format aquesta idea queda ben clara de seguida.

Recordeu que el nomenclàtor no s'aplica al codi C: CodeLite us ajudarà a escriure el codi C amb un format adequat. A més, per tenir més detalls de com fer-ho, podeu revisar la **REC. Guia d'estil de programació en C (https://aula.uoc.edu/courses/78741/pages/rec-guia-destil-de-programacio-en-c-2-2)** .

1. Llegenda

- El format **negreta** s'utilitza per a les paraules clau del llenguatge.
- Aconsellem utilitzar l'anglès per al desenvolupament dels algorismes o programes. No sabem qui utilitzarà el codi que hem creat amb les eines que existeixen per compartir codi. Per això, us animem a utilitzar l'anglès quan creeu un codi: en el nom de les variables i també en els comentaris que expliquin i documentin el codi.

2. Comentari

```
{ comment }
```

3. Algorisme

```
algorithm algorithmName
   {Define variables}
   var
      intVar: integer;
   end var

   {Initialize variables}
   intVar := 0;

   {Add as many instructions as you need}

end algorithm
```

4. Tipus bàsics de dades

```
integer
char
real
boolean
```

```
string
```

4.1. Valors de constants booleanes

```
true
false
```

4.2. Definició de variables

```
var
  intVar: integer;
  charVar: char;
  realVar: real;
  booleanVar: boolean;
  stringVar: string;
end var
```

4.3. Definició de constants

```
const
  INTEGER_CONST: integer = 20;
  CHAR_CONST: char = 'a';
  REAL_CONST: real = 12.9;
  BOOLEAN_CONST: boolean = true;
  STRING_CONST: string = "Hello!";
end const
```

4.4. Operador d'assignació

:=

4.5. Operadors lògics

and

or

not

4.6. Operadors de comparació

=

≠

<

>

≤

≥

4.7. Funcions de canvi de tipus

function integerToReal(*varName*: **integer**): **real**;

function realToInteger(*varName*: **real**): **integer**;

function charToCode(*varName*: **char**): **integer**;

```
function codeToChar(varName: integer): char;
```

5. Funcions i accions d'entrada i sortida

5.1. Variables

```
function readInteger(): integer;
function readReal(): real;
function readChar(): char;
function readString(): string;
function readBoolean(): boolean;
action writeInteger(in varName: integer);
action writeReal(in varName: real);
action writeChar(in varName: char);
action writeString(in varName: string);
action writeBoolean(in varName: boolean);
```

5.2. Fitxers

```
function openfile(fileName: string): file;
action closeFile(inout fileName: string);
function readIntegerFromFile(fileToRead: file): integer;
function readRealFromFile(fileToRead: file): real;
function readCharFromFile(fileToRead: file): char;
function readStringFromFile(fileToRead: file): string;
function readBooleanFromFile(fileToRead: file): boolean;
function isEndOfFile(fileToRead: file): boolean;
```

```
action writeIntegerToFile(inout fileToWrite: file, in varName: integer);
action writeRealToFile(inout fileToWrite: file, in varName: real);
action writeCharToFile(inout fileToWrite: file, in varName: char);
action writeStringToFile(inout fileToWrite: file, in varName: string);
action writeBooleanToFile(inout fileToWrite: file, in varName: boolean);
```

6. Altres tipus de variables

6.1. Definició de vectors

```
const
  NUM_ELEMENTS: integer = 10;
end const
var
  integerVector: vector[NUM_ELEMENTS] of integer;
  charVector: vector[NUM_ELEMENTS] of char;
  realVector: vector[NUM_ELEMENTS] of real;
  booleanVector: vector[NUM_ELEMENTS] of boolean;
  stringVector: vector[NUM_ELEMENTS] of string;
end var
```

6.2. Definició de tuples

```
type
  tupleName = record
    realField: real;
    {You can define any other fields of any type following each var or type definition scheme}
  end record
end type
```

6.3. Definició de punters

```
var
  pointerToIntName: pointer to integer;
  pointerToRealName: pointer to real;
end var
```

7. Estructures de control

7.1. Estructura condicional

```
if i=j then {change "i=j" with any other expression you need}
    i:=j+2; {Add as many instructions as you need here}
end if
```

o

```
if i = j then {change "i=j" with any other expression you need}
    i := j+2; {Add as many instructions as you need here}
else
    i := j-2; {Add as many instructions as you need here}
end if
```

```
switch variable
    case variable = value1 then
        {Add as many instructions as you need here}
    end case

    case variable = value2 then
        {Add as many instructions as you need here}
    end case

    { ... }

    case default then
        {Add as many instructions as you need here}
    end case
end switch
```

7.2. Estructura iterativa

```
while i<j do {change "i=j" with any other expression you need}
```

```
    value1 := value2 + i ; {Add as many instructions as you need here}
    i := i + 1 ; {Always remember to change the value of one of the values in the expression to avoid infinite loops!}
end while
```

```
for i := initialValue to finalValue [step increment/decrement value] do
    value1 := value2+i;
    {Add as many instructions as you need here}
end for


for i := initialValue to finalValue step 2 do {step can be a positive or negative number, or a variable}
    value1 := value2+i;
    {Add as many instructions as you need here}
end for
```

8. Funció

```
function functionName (parName1: integer, parName2: integer): integer
    return parName1 + parName2;
end function
```

9. Acció

```
action actionName (in parName1: integer, out parName2: real, inout parName3, bool)
    parName2 = 2.0 * integerToReal(parName1);
    parName3 := not parName3;
end action
```

10. Classes de paràmetres d'una acció

in

out

inout