Pseudocode (programmer defined identifiers preserve the Spanish words of the original cheat sheet) (date: 2018-10-18)

1. Predefined data types

```
boolean  {with values true and false; we write comments like this in curly brackets}
character
natural  {we include 0 in Natural numbers}
integer
real
string  {sequence of characters with arbitrary length (maybe empty)}
```

2. Constants definition

```
constants
  letraA = ‘A’; maxNum = 100  {we use semicolon’s to separate instructions}
  hoy = "martes"  {a non-empty string}
  pi = 3.1416
```

3. Definition of new data types

```
types
  mes = (ene,feb,mar,abr,may,jun,jul,ago,sep,oct,nov,dic)  {enumerated type}
  mesVerano = jul..sep  { subrange type of another discrete type }
  día = 1..31
  fecha = record   {record type (or structure), aggregation of fields}
            elDía: día
            elMes: mes
            elAño: natural
          endrec
  pluviometría = array[mes] of real    { array type }
  fiestas = array[1..maxNum] of fecha
  secFechas = file of fecha  {binary file of ‘fechas’}
  entrada = text file  {sequence of lines (a line is a sequence of characters)}
```

4. Variables declaration

```
variables
  contador: natural:= 0    {we can initialize the variable with its definition}
  éxito, error: boolean
  nombre: string
  cadenaVacía: string:= “”  {the variable is initialized with empty sequence, “”}
  cumpleaños,aniversario: fecha
  festivos,patronos: fiestas
```

5. Assignment statement

```
contador:= contador+1
error:= false; éxito:= true
éxito:= not error and (contador>3) or éxito
cumpleaños.elDía:= 13
nombre:= “Juan”
festivos[2].elMes:= successor(ene)  {successor and predecessor for enumerated types}
cumpleaños:=aniversario {assignment of a record: all its fields are assigned}
festivos:=patronos {assignment of an array: all its components are assigned}
```

6. String operations

```
variables apellido, resto: string
          i: natural
          letra: character
```

Comparisons (by alphabetical order, considering all the characters according with ASCII code and its extensions):

```
"Costa" = apellido       apellido ≠ "Po3$"     apellido < resto
apellido ≤ resto         apellido > resto      apellido ≥ resto
```

Concatenation:

```
resto:= resto + apellido  {resto takes the value of the concatenation of its previous
                           value with the value of apellido}
apellido:= "de " + apellido  {"de" string is preceded to previous value of apellido}
resto:= "hol" + "a"     {resto takes the value "hola"}
```

Length:

```
i:= length(aux)  {length returns the length of the string, that is, its number of
                  characters; the lenght of the empty string, "", is 0}
```

i-th carácter and substring:

```
letra:= apellido[i]  {i-th character of the string; 1 ≤ i ≤ length(apellido)}

letra:= apellido[1] {first character of the string, assuming the string is not empty}
resto:= apellido[2..length(apellido)] {substring of apellido from the 2nd character to
                                       the last one}
resto:= apellido[i..j] {substring from i-th character to j-th; 1≤i≤j≤length(apellido)}
```

7. Input/output statements

```
write("Introduzca su edad: ")  {writes the message in the standard output device
                                       (the screen, for instance)}
readLine(edad) {if edad is an Integer variable, it reads a value of type Integer from
                    the standard input device (the keyboard, for instante), that value
                    is assigned to the variable edad, and finally reads the end-of-
                    line mark}
writeLine("La edad introducida es ",edad) {writes the message and the value of edad
                                                and after that jumps to the next line}
writeLínea("Su nombre es:", nombre) {writes the message, then the value of the string
                                         nombre and finally jumps to the next line}
write("Introduce tres letras: ") {writes the message}
read(a,b,c)  {if a,b,c are character variables, it reads three characters and assigns
                    them to the variables a,b,c}
readLine {reads the enf-of-line mark}
```

8. Conditional statements

```
if <condition> then
  <sequence of statements>
endif

if <condition> then
  <sequence of statements>
else
  <sequence of statements>
endif

if <condition> then
  <sequence of statements>
elsif <condition> then
  <sequence of statements>
elsif <condition> then
  <sequence of statements>
...
```

```
else
  <sequence of statements>
endif

selection
  <condition 1>: <sequence of statements>;
  <condition 2>: <sequence of statements>;
  ...
  <condition n>: <sequence of statements>;
  [ otherwise: <sequence of statements> ]
endsel
{conditions must be in mutual exclusion;
 once the sequence of statements of the first true condition is executed, the
 selection statement ends}
```

9. Iterative statements

```
for <discrete_type_variable>:=<initial_value> to <final_value> do
  <sequence of statements>
endfor

for <discrete_type_variable>:=<initial_value> downto <final_value> do
  <sequence of statements>
endfor

while <condition> do
  <sequence of statements>
endwhile

repeat
  <sequence of statements>
until <condition>
```

10. Procedures and functions

```
procedure <name>(in <parameters_1>:<type_1>;
                 out <parameters_2>:<type_2>;
                 i/o <parameters_3>:<type_3> ... )
{in,out,i/o means in, out or in and out parameter, respectively}
<local declaration of constants, types, variables, procedures, functions...>
begin
  <sequence of statements>
end
{a procedure is a virtual action or statement;
 the main program is a procedure without parameters}

function <name>(<param_1>:<type_1>; <param_2>:<type_2> ...) returns <type_fun>
<local declaration of constants, types, variables, procedures, functions...>
begin
  <sequence of statements>
  returns <value_of_type_fun>   {tras devolver el valor la función termina}
end
{a function is a virtual value, i.e., it is evaluated inside an expression and the
result is a value}
```

11. Modules

```
module tablas
import <list of other modules that are needed by this one>
export
  {public part: constants, type names, procedures and functions headings...}
  ...
implementation
  {private part: definition of types named in public part, other (private) types,
   implementation of procedures and functions...}
  ...
end
```

12. Generic modules

```
module generic listasGenéricas
parameters
  type elemento
  with function “<”(e1,e2:elemento) returns boolean
export
  type lista
  procedure crear(out l: lista)
  procedure añadirÚltimo(i/o l:lista; in e:elemento)
  ...
implementation
  ...
end
```

13. Use of generic modules

a. To define another generic type:

```
module generic pilasGenéricas
import listasGenéricas
parameters
  type elemento
export
  type pila
   . . .
implementation
  type pila = listasGenéricas.lista {the type exported by listasGenéricas}

  procedure crear(out p:pila)
  begin
    listasGenéricas.crear(p)
  end
  . . .
end {of the module}
```

b. Instantiate and use a generic module:

```
. . .
import pilasGenéricas;
module pila_naturales = pilasGenéricas(natural); {it exports type pila}
. . .
{in order to declare variables:}
p:pila;  {or: p:pila_naturales.pila}
{in order to use its operations:}
crear(p);  {or: pila_naturales.crear(p);}
. . .
```

14. Pointers

```
types pila = ↑unDato {pointer type (or reference) to unDato}
      unDato = record
                 dato:natural
                 siguente:pila
               endrec
variables p,q:pila
```

Especial value (constant) of any pointer type: `nil` (“no address”)

```
p:=nil
```

Statements with pointers:

```
newData(p);
```

```
p↑.dato:=3
q:=p
dispose(p)

if p=q then ...
```

15. Statements for creation and use of files

In the following, a basic scheme of how files are used is shown, including the basic statements for their manipulation.

- **Text files**

```
variables
   f:text file;
   d:character;
   nombre:string
...
begin
   {To associate variable f with external file named “Leeme.txt”}
   associate(f,“Leeme.txt“);
   {Initialize the file for writing: empty external file is created}
   initiateWrite(f);
   {We use the same writing statements used for writing in the screen or standard
   output, adding a text file variable as the first parameter. Example:
   write(f,d); {writes character d at the end of file f}
   write(f,nombre); {writes string nombre at the end of file f}
   . . .
   {Initialize the file associated with f for reading, reading position is set at the
   beginning of f, the first data to read will be the first of the file}
   initiateRead(f);

   {To know if there are more data in the file to read, or not, we have the function:
   endOfFile(f) (returns a boolean)}
   while not endOfFile(f) do
      {We use the same reading statements used for reading from the keyboard or
      standard input, adding a text file variable as the first parameter. Example:}
      read(f,d); {reads the next character from file f, assigns it to d, and f is
                 ready to read the next character}
      . . .
   endwhile;
   {Dissociate f and the external file}
   dissociate(f);
   ...
end
```

- **Binary files**

  Similar to text files, but now the minimal unit of information to read or to write is a data of the specified type (and, of course, there is not a line structure).

```
variables
   ff:file of fecha  {binary file type of fechas}
   día:fecha

...
begin
   {To associate variable ff with external file named “misDatos.dat”}
   associate(ff,“misDatos.dat“);
   {Initialize the file for writing: empty external file is created}
   initiateWrite(ff);
   write(ff,día); {writes data dia at the end of file ff}
   . . .
   initiateRead(ff); {Initialize the file associated with ff for reading, reading
   position is set at the beginning of ff, the first data to read will be the first of
   the file}
   {To know if there are more data in the file to read, or not, we have the function:
   endOfFile(f) (returns a boolean)}
   while not endOfFile(ff) do
      {To read a data from the binary file:}
      read(ff,día); {reads the next data of type fecha from file ff, assigns it
                       to día, and ff is ready to read the next data }
   endwhile;
   {Dissociate f and the external file}
   dissociate(ff);
   ...

end
```