Example 01: Palindrome filter

In this appendix we illustrate the CLiP style of literate programming by a program to filter palindromic lines from an input file.

Two files are involved: the program module (`PALINDROME.PAS`) and a test file (`TEST-DATA.IN`).

A.1. Specification

A palindrome is a sentence with the property that the letters from left to right, read the same as from right to left. In the comparison uppercase and lowercase letters are considered to be equivalent and all other characters are simply ignored. Hence an empty sentence is a palindrome. Other examples are:

```
(*************  #file "TESTDATA.IN"  #comment off  **************)
Ada
1234567
doremifasolosafimerod
(*******                Testdata                #multiple  *******)
(*****************  End of TESTDATA.PAS  ***********************)
```

The following sentences do not qualify as a palindrome.

```
(*******                Testdata                   #quick  *******)
Mr. Clinton won the elections over Mr. Bush.
This line is not palindromic.
Aabbccdd Aabbccdd Aabbccdd Aabbccdd
Abracadabra
supercalafragilisiticexpielecdosia
```

The following are examples of more sophisticated palindromes.

```
(*******                Testdata                   #quick  *******)
Able was I, ere I saw Elba.
A man, a plan, a canal, Panama.
Norma is as selfless as I Am, Ron.
```

Note that the famous Dutch sentence

```
(*******                Testdata                   #quick  *******)
Koos Eekfeen keek maar door rood kerkraam, maar krek door rood raam
keek neef Kees ook.
```

will not be recognized as a plindrome since it occupies two lines.

The program `PALINDROME` reads an input file, filters the lines that are palindromic and writes them to an output file.

A.2. Communication with the outside world

The program conforms to the general template of a Pascal program. We introduce the files `IN_FILE` and `OUT_FILE` to define its communication with the outside world. The actual files have to be specified at run-time. Thus we have

```
(****************  #file "PALINDRO.PAS"  *********************)
(************************************************************)
(* Program: Palindrome filter program.                      *)
(* Purpose: To filter the palindromic lines from a given input  *)
(*          file to a specified output file.                 *)
(************************************************************)
PROGRAM PALINDROME (INPUT, OUTPUT, IN_FILE, OUT_FILE);

(*******    Palindrome constants  #multiple #comment off  *******)
(*******    Palindrome types      #multiple #comment off  *******)
VAR
    IN_FILE, OUT_FILE: TEXT;
(*******    Palindrome variables  #multiple #comment off  *******)

BEGIN
    ASSIGN  (IN_FILE,  'TESTDATA.IN');  RESET   (IN_FILE);
    ASSIGN  (OUT_FILE, 'TESTDATA.OUT'); REWRITE (OUT_FILE);

    (****************  Palindrome (body)  ********************)
    (** Copy the lines of the IN_FILE that are palindromic to  **)
    (** the OUT_FILE.                                         **)
    (*******************************************************)

END (*PALINDROME*).
(******************  End of PALINDROME.PAS  ******************)
```

To prepare the module for future declarations of constants and types we have

```
(*******        Palindrome constants      #leader #quick  *******)
CONST
(*******        Palindrome types          #leader #quick  *******)
TYPE
```

A.3. Processing of the files

The program processes `IN_FILE` line by line. The idea is to buffer an exact copy of the current line in `IN_LINE`, while at the same time its letters are buffered in `LETTERS`. So `LETTERS` will be empty if the line holds no letters at all, in which case the line is considered to be palindromic by definition.

We choose the buffers `IN_LINE` and `LETTERS` to be of the same type, `TEXT_LINE`, which we will not specify in detail right now. For this purpose we introduce a type `ABSTRACT`.

```
(*******        Palindrome types                  #quick  *******)
    ABSTRACT  = (DEFINED, UNDEFINED);
```

`TEXT_LINE` will temporarily be declared `ABSTRACT` and its details will be defined later. Thus the declaration of `TEXT_LINE`

```
(*******                Palindrome types                 *******)
(*******            Declaration of TEXT_LINE             *******)
(****************  End of Palindrome types  ******************)
```

is temporarily satisfied with the type `ABSTRACT`.

```
(*******    Declaration of TEXT_LINE     #quick #default  *******)
    TEXT_LINE = ABSTRACT;
```

The declaration for the variables `IN_LINE` and `LETTERS` becomes

```
(*******    Palindrome variables                    #quick  *******)
    IN_LINE,
    LETTERS:         TEXT_LINE;
```

We have to test LETTERS in order to decide whether or not IN_LINE contains a palindrome. The result of this test is flagged by IS_PALINDROME, for which we introduce the declaration

```
(*******    Palindrome variables                    #quick  *******)
    IS_PALINDROME:  BOOLEAN;
```

Now the body of the Palindrome filter may be expanded as

```
    (*****************  Palindrome (body)  *********************)
    (** Copy the lines of the IN_FILE that are palindromic to  **)
    (** the OUT_FILE.                                          **)
    WHILE NOT EOF (IN_FILE) DO
    BEGIN
        (*****************  Palindrome (1)  ********************)
        (** Read a line from IN_FILE into IN_LINE. The letters **)
        (** of this line are copied to LETTERS.                **)
        (*******************************************************)

        READLN (IN_FILE);

        (*****************  Palindrome (2)  ********************)
        (** Test palindromicity of LETTERS. Set IS_PALINDROME  **)
        (** to reflect the result of the test.                 **)
        (*******************************************************)


        IF IS_PALINDROME THEN
        BEGIN
            (*****************  Palindrome (3)  ****************)
            (** Write IN_LINE to OUT_FILE.                     **)
            (***************************************************)

            WRITELN (OUT_FILE);
        END (*IF*);
    END (*WHILE*);
    (*************  End of Palindrome (body)  ******************)
```

A.4. Choosing the structure of IN_LINE and LETTERS

Before we can proceed we need to establish a structure for the objects IN_LINE and LETTERS. Thus we define TEXT_LINE as a structure with two components. The first component is an array, CHARS, which contains the characters to be buffered. The second component, LENGTH, indicates which part of the array is actually occupied. The maximum number of characters that can be buffered by the structure is determined by the length, MAX_L, of the array. MAX_L serves as an implementation parameter.

```
(*******    Palindrome constants                    #quick  *******)
    MAX_L = 132;

(*******        Declaration of TEXT_LINE           #quick  *******)
    TEXT_LINE =     RECORD
                        CHARS: ARRAY[1..MAX_L] OF CHAR;
                        LENGTH: 0..MAX_L;
                    END (*RECORD*);
```

A.5. Reading a line

For efficiency reasons we fill IN_LINE and LETTERS simultaneously. Therefore we buffer every character that is read from IN_FILE in the variable IN_CHAR.

```
(*******    Palindrome variables                    #quick  *******)
   IN_CHAR:    CHAR;
```

Only when IN_CHAR turns out to be a letter it is copied to LETTERS. Since this process is crucial for the overall operation, we make provisions for some debugging code here.

```
(*****************  Palindrome (1)  ********************)
(** Read a line from IN_FILE into IN_LINE. The letters **)
(** of this line are copied to LETTERS.                **)
IN_LINE.LENGTH := 0;
LETTERS.LENGTH := 0;
WITH IN_LINE DO
WHILE NOT EOLN (IN_FILE) DO
BEGIN
    READ (IN_FILE, IN_CHAR);
    LENGTH := LENGTH + 1;
    CHARS[LENGTH] := IN_CHAR;
    IF IN_CHAR IN ['A'..'Z', 'a'..'z'] THEN
    WITH LETTERS DO
    BEGIN
        LENGTH := LENGTH + 1;
        CHARS[LENGTH] := IN_CHAR;
    END (*WITH/IF*);
END (*WHILE/WITH*);

(*********************  Palindrome (test)  **************)
(** Check contents of IN_LINE and LETTERS.  #optional  **)
(*******************************************************)

(*****************  End of Palindrome (1)  **************)
```

A.6. Testing for palindromicity

We test the palindromicity of LETTERS in two steps. First we transform the contents of LETTERS to uppercase and then we compare the characters of LETTERS pairwise. The comparison is done starting with the most outside characters and progressing inward. The string is assumed a palindrome until the opposite is proven through a pair of different characters. With the local counter

```
(*******    Palindrome variables                    #quick  *******)
   I:        INTEGER;
```

we keep track of the comparing process. Now Palindrome (2) can be expanded as

```
(*****************  Palindrome (2)  ********************)
(** Test palindromicity of LETTERS. Set IS_PALINDROME  **)
(** to reflect the result of the test.                 **)
WITH LETTERS DO
BEGIN
    (* Transform lowercase to uppercase.                  *)
    FOR I := 1 TO LENGTH DO
    IF CHARS[I] IN ['a'..'z']
    THEN CHARS[I] :=
         CHR(ORD(CHARS[I]) - ORD('a') + ORD('A'));

    (* Perform the palindromicity test.                   *)
    IS_PALINDROME := TRUE;
    I := 1;
    WHILE IS_PALINDROME AND (I <= LENGTH DIV 2) DO
    IF CHARS[I] = CHARS[LENGTH-I+1] THEN
        I := I + 1
    ELSE
        IS_PALINDROME := FALSE;
END (*WITH*);
(*****************  End of Palindrome (2)  **************)
```

A.7. Writing the palindrome

The only remaining action is to write the contents of `IN_LINE`. Again we need a local counter

```
(*******    Palindrome variables                    #quick  *******)
   J:         INTEGER;
```

The writing proceeds straight forward.

```
(*****************  Palindrome (3)  *****************)
(** Write IN_LINE to OUT_FILE.                      **)
WITH IN_LINE DO
BEGIN
    FOR J := 1 TO LENGTH DO
        WRITE (OUT_FILE, CHARS[J]);
END (*WITH*);
(*************  End of Palindrome (3)  **************)
```