EXAMPLE PROBLEM - PASS WORD VERIFICATION

1. Requirements

Design a Ada program that takes a text string as input, analysis this text string and outputs whether it is a valid password or not (a similar operation, at least in part, to that performed by the `passwd` UNIX command). If the string is not a valid password output the reason why. A valid password displays the following attributes:

- Contain between 6 and 8 characters
- At least two characters must be letters (upper or lower case)
- At least one must be:
  - □ numeric (the digits 0 through 9), or
  - □ special (neither letter nor numeric - for example, -, _, @ or $).

If the string is not a valid password output the reason why. Note the following ASCII equivalents:

ASCII CODE	CHARACTER
33-47, 58-64, 91-96, 123-126	Special characters
48-57	Numeric characters
65-90	Upper case letters
97-122	Lower case letters

2. Design

A top down analysis of the problem is given in Table 1.

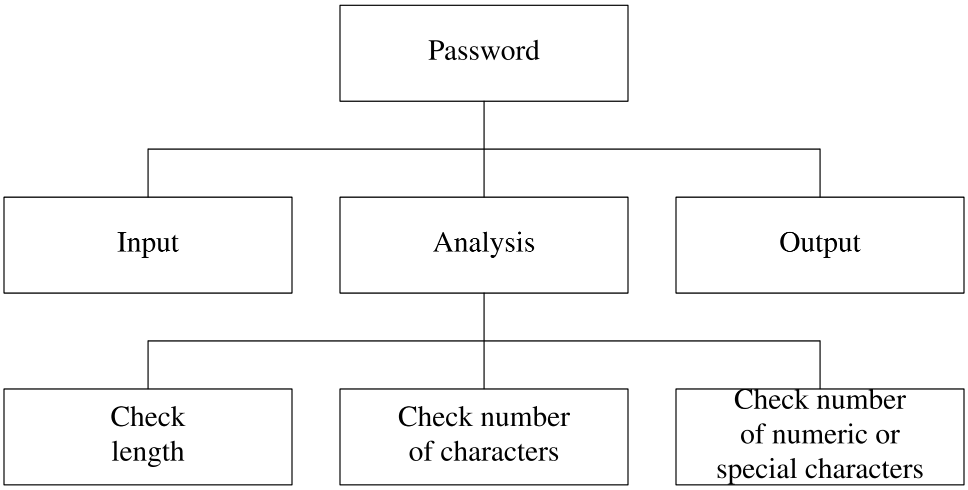


Table 1: *Top down analysis*

The following procedures and functions will be required:

1) `PASSWORD` (top level): Input string, calls to `CHECK_LENGTH`, `CHECK_CHARACTERS` and `CHECK_OTHERS`. Output "valid password" if functions all return `TRUE`.

Name	Description	Type	Range
INPUT_STRING	Input variable	STRING	Unconstrained
STRING_LENGTH	Global variable	INTEGER	Default
VALID_FLAG	Global variable	Boolean	TRUE or FALSE

2) `CHECK_LENGTH` (level 2 function): Check string length. Return `TRUE` if OK, `FALSE` otherwise.

Name	Description	Type	Range
LENGTH	Formal parameter	INTEGER	Default

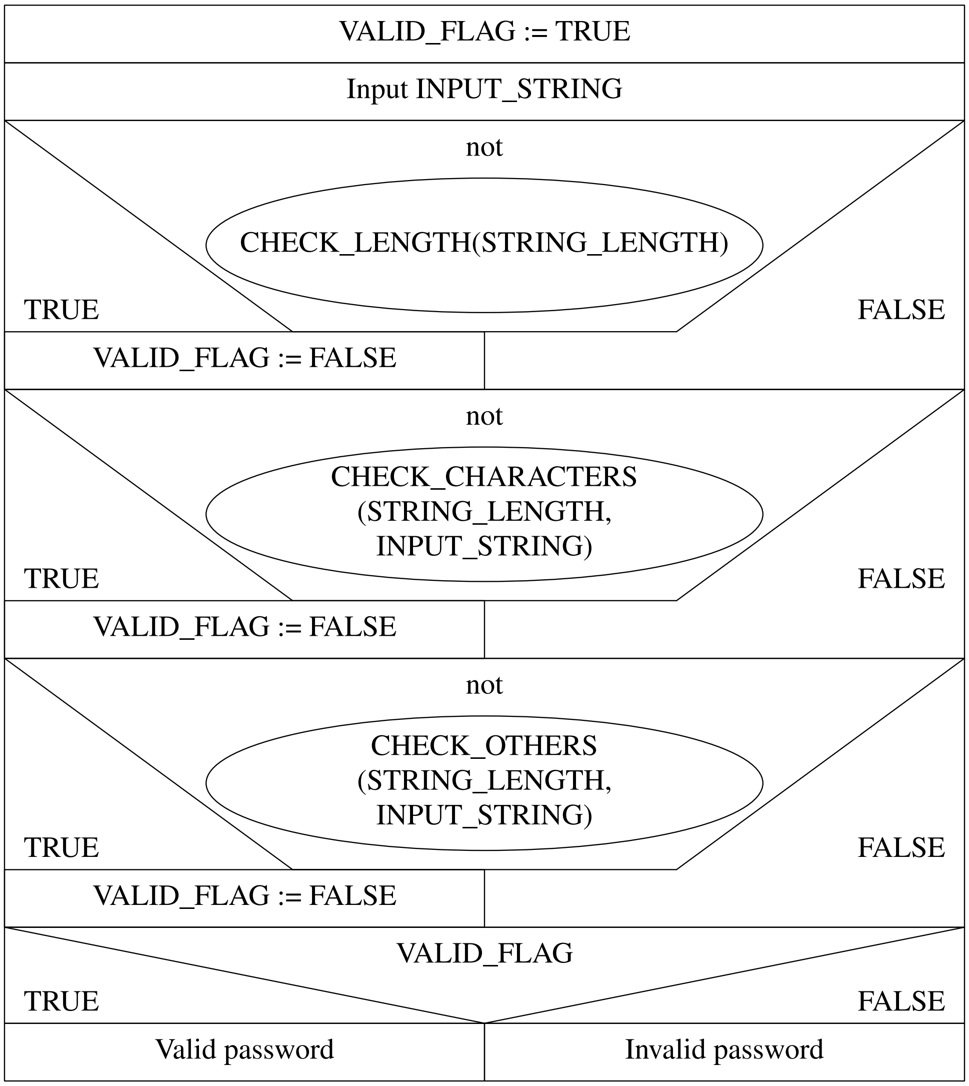
3) `CHECK_CHARACTERS` (level 2 function) : Check number of characters Return `TRUE` if OK, `FALSE` otherwise.

Name	Description	Type	Range
PS_WORD	Formal parameter	STRING	Unconstrained
LENGTH	Formal parameter	INTEGER	Default
CODE	Local variable	INTEGER	Default
COUNTER	Local variable	INTEGER	Default

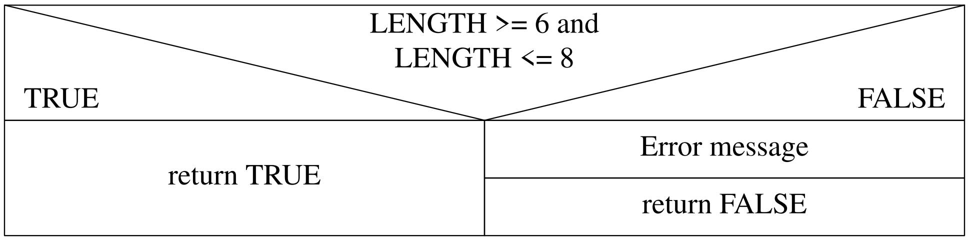
4) CHECK_OTHERS (level 2 function): Check for numeric or special character. Return `TRUE` if OK, `FALSE` otherwise.

Name	Description	Type	Range
PS_WORD	Formal parameter	STRING	Unconstrained
LENGTH	Formal parameter	INTEGER	Default
CODE	Local variable	INTEGER	Default

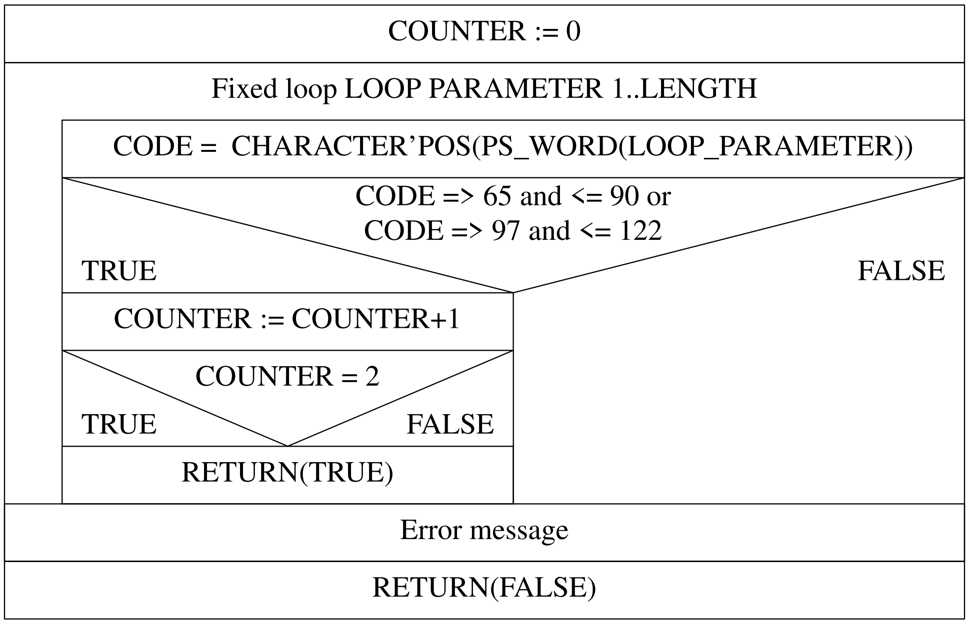
A complete Nassi-Shneiderman design for the above is given in Table 2.



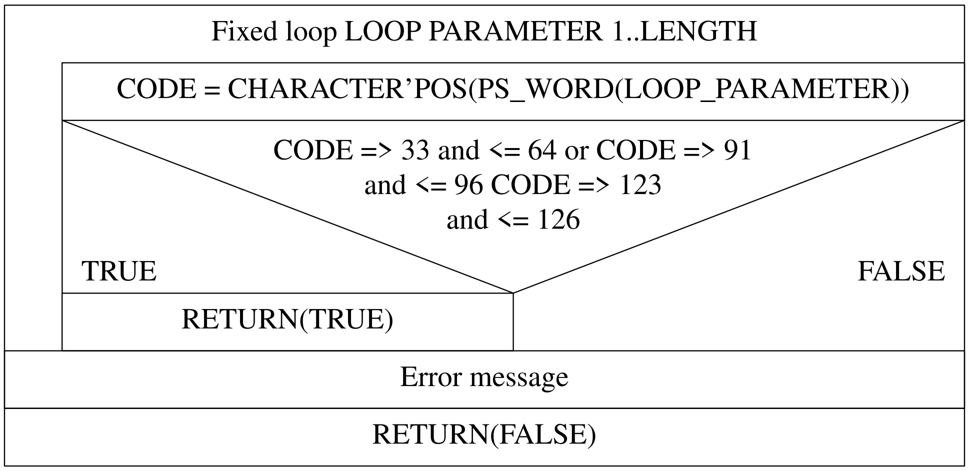
PASSWORD
(Top level procedure)



STRING_LENGTH
(Level 2 function)



CHECK_CHARACTERS
(Level 2 function)



CHECK_OTHERS
(Level 2 function)

Table 2: *Nassi-Shneiderman chart*

A flow chart indicating the flow of control through this sequence of procedures/functions is given in Table 3

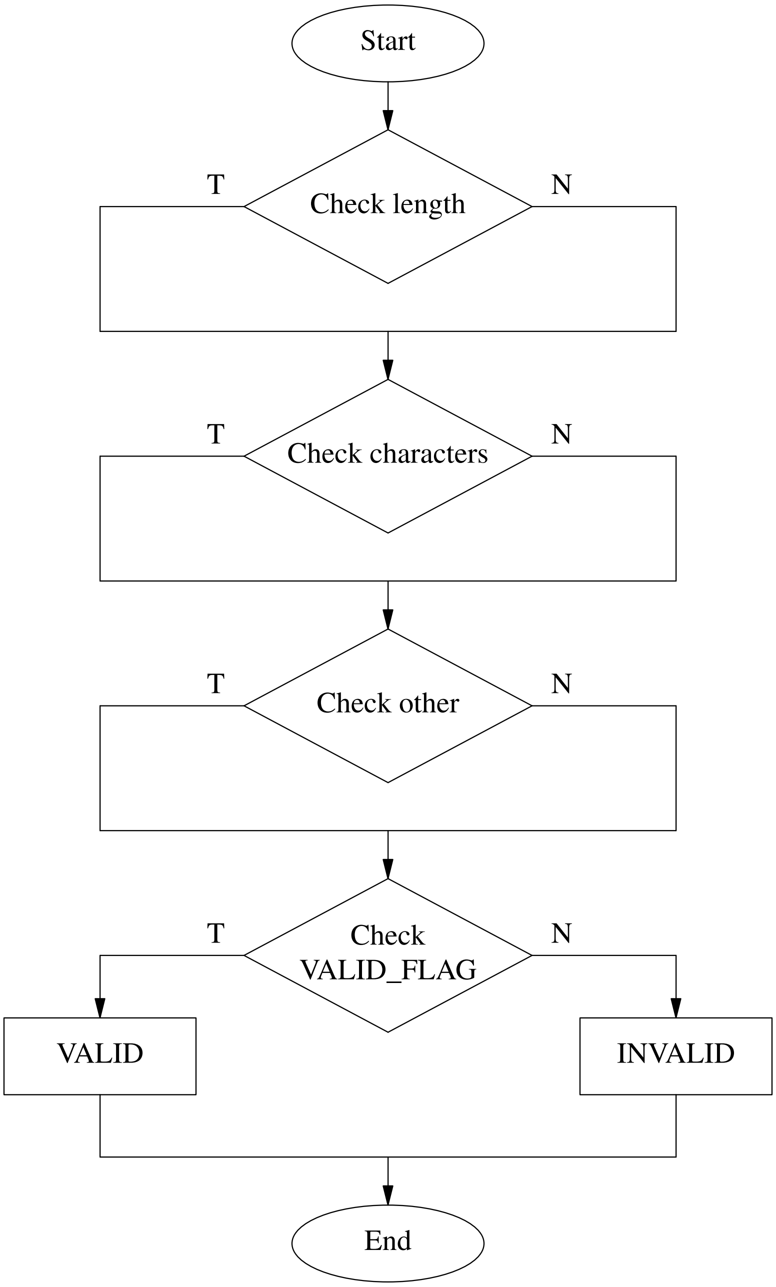


Table 3: *Control flow chart*

3. Implementation

The encoding of the above design is given in Table 4.

```
-- PASSWORD
-- 1 October 1997
-- Frans Coenen
-- Dept Computer Science, University of Liverpool

with TEXT_IO;
use TEXT_IO;

procedure PASSWORD is
    INPUT_STRING  : STRING(1..20);
    STRING_LENGTH : INTEGER;
    VALID_FLAG    : BOOLEAN := TRUE;

    ----------------------------------------------------------------------
    -- CHECK LENGTH
    function CHECK_LENGTH(LENGTH: INTEGER) return BOOLEAN is
    begin
        if (LENGTH >= 6 and LENGTH <= 8) then
            RETURN(TRUE);
        else
            PUT_LINE("Password must comprise 6 to 8 characters");
            RETURN(FALSE);
        end if;
    end CHECK_LENGTH;
    ----------------------------------------------------------------------
    -- CHECK CHARACTERS
    function CHECK_CHARACTERS(PS_WORD: STRING; LENGTH: INTEGER) return
            BOOLEAN is
        COUNTER     : INTEGER := 0;
        CODE    : INTEGER;
    begin
        for LOOP_PARAMETER in 1..LENGTH loop
            CODE := CHARACTER'POS(PS_WORD(LOOP_PARAMETER));
            if (CODE >= 65 and CODE <= 90) or
                    (CODE >= 97 and CODE <= 122) then
                COUNTER := COUNTER+1;
                if COUNTER = 2 then
                    RETURN(TRUE);
                end if;
            end if;
        end loop;
        PUT("Password must comprise at least two alphabetic ");
        PUT_LINE("characters");
        RETURN(FALSE);
    end CHECK_CHARACTERS;
    ----------------------------------------------------------------------
```

Table 4: *Implementation*

```
    -- CHECK OTHERS
    function CHECK_OTHERS(PS_WORD: STRING; LENGTH: INTEGER) return
             BOOLEAN is
        CODE    : INTEGER;
    begin
        for LOOP_PARAMETER in 1..LENGTH loop
            CODE := CHARACTER'POS(PS_WORD(LOOP_PARAMETER));
            if (CODE >= 33 and CODE <= 64) or
                    (CODE >= 91 and CODE <= 96) or
                    (CODE >= 123 and CODE <= 126) then
                RETURN(TRUE);
            end if;
        end loop;
        PUT("Password must comprise at least one numeric or ");
        PUT_LINE("special character");
        RETURN(FALSE);
    end CHECK_OTHERS;
    -----------------------------------------------------------------------

begin
    PUT_LINE("Input a proposed password");
    GET_LINE(INPUT_STRING,STRING_LENGTH);
-- Check length
    if not CHECK_LENGTH(STRING_LENGTH) then
        VALID_FLAG := FALSE;
    end if;
-- Check number of alphabetic characters
    if not CHECK_CHARACTERS(INPUT_STRING,STRING_LENGTH) then
        VALID_FLAG := FALSE;
    end if;
-- Check number of other characters
    if not CHECK_OTHERS(INPUT_STRING,STRING_LENGTH) then
        VALID_FLAG := FALSE;
    end if;
-- Output result
    if VALID_FLAG then
        PUT_LINE("Valid password");
    else
        PUT_LINE("Invalid password");
    end if;
end PASSWORD;
```

Table 4 Continued

4. Testing

4.1. Black box testing

The input string variable should be tested at limits, somewhere in middle and outside limits. A set of test cases is presented in Table 5 (and results in Table 6).

TEST CASE INPUT_STRING	EXPECTED RESULT OUTPUT
aVeryLongPasswordIndeed	invalid
a	invalid
password_With_20Char	invalid
myPass1	valid

Table 5: Input Test Cases

```
kuban-197 $ password
Input a proposed password
aVeryLongPasswordIndeed
Password must comprise of 6 to 8 characters
Password must comprise at least one numeric or special character
Invalid password
kuban-198 $ eed
ksh: eed:  not found

kuban-199 $ password
Input a proposed password
a
Password must comprise of 6 to 8 characters
Password must comprise at least two alphabetic characters
Password must comprise at least one numeric or special character
Invalid password

kuban-200 $ password
Input a proposed password
password_With_20Char
Password must comprise of 6 to 8 characters
Invalid password

kuban-201 $ password
Input a proposed password
myPass1
Valid password
```

Table 6: Black box testing result

4.2. White box testing

Path testing. The flow chart given previously identifies the paths through the system. The path selected is dependent on a number of tests. These selection statements involve compound Boolean expression incorporating both disjunctions and conjunctions. There are number of these selection statements. An appropriate set of test cases is given in the following tables.

`STRING_LENGTH` function

We have a conjuncted Boolean expression here (A and B). We should therefore test the situations where both A and B are true, both false and all combinations of true and false. However, given the nature of the input it is not possible for both A and B to be false (string less than 6 character and greater than 8 characters).

TEST CASE INPUT_STRING	EXPECTED RESULT OUTPUT
word777	valid
word99999	invalid
word5	invalid

`CHECK_CHARACTERS` function.

Expression here is of the form (A and B) or (C and D), but we are expecting one of a sequence of numbers there fore we should test the limits of the ranges and numbers outside of the ranges as shown below. We should also test for the situation where no alphabetic characters occur, one occurs, two occurs or more than two occur.

TEST CASE		EXPECTED RESULT
ASCII_CODE	INPUT_STRING	OUTPUT
60	<	invalid
65	A	invalid
78	NN	invalid
90	ZZZ	invalid
95	_	invalid
97	a	invalid
110	nn	invalid
122	zzz	invalid
125	}	invalid

`CHECK_OTHERS` function

In a similar manner to the `CHECK_CHARACTERS` function we should test the limits of the ranges and numbers outside of the ranges included in the compound Boolean expression `(CODE >= 33 and CODE <= 64) or (CODE >= 91 and CODE <= 96) or (CODE >= 123 and CODE <= 126)`

TEST CASE		EXPECTED RESULT
ASCII_CODE	INPUT_STRING	OUTPUT
32	(space)	invalid
33	!	invalid
50	2	invalid
64	@	invalid
78	N	invalid
91	[	invalid
94	ˆ	invalid
96	‘	invalid
110	n	invalid
123	{	invalid
124	|	invalid
126	˜	invalid

Due to the difficulty encountered when attempting to input the ASCII character 127 a test above 126 has not been included.