EXAMPLE PROGRAM SET INTERSECTION

1. Requirements

Develop an Ada program which, given a set of six positive numbers, determines the intersection of this set with the set `{1, 2, 3, 4, 5, 6}`. Note: the input set should contain no duplicates.

2. Design

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

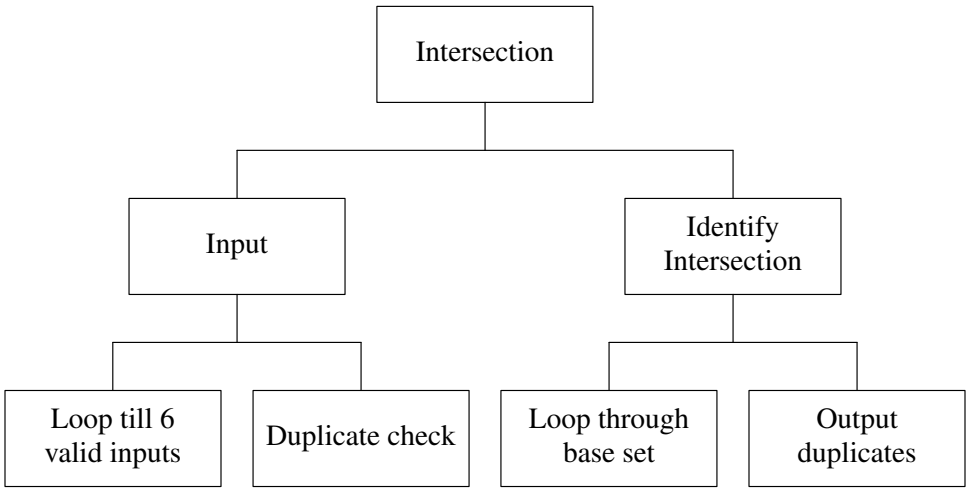


Table 1: *Top down analysis*

We will represent both the base set and the user defined sets as arrays using a user defined array type `SET` declared as a 6 element array of type `INTEGER`. We can then identify the following procedures/functions:

1) `INTERSECTION` (top level procedure): Calls to `INPUT_USER_SET` and `IDENTIFY_INTERSECTION`. Include type definition for `SET` and initialisation of base set. No data items.

2) `INPUT_USER_SET` (level 2 function): Loop till six valid numbers have been input. Call to `DUPLICATE_ELEMENT` to check inputs.

NAME	USAGE	TYPE	RANGE
USER_SET	Local variable (array)	SET	n.a.
SET_LENGTH	Local variable	INTEGER	Default
SET_ELEMENT	Local variable	INTEGER	Default

3) `IDENTIFY_INTERSECTION` (Level 2 procedure): Compare sets and output common elements.

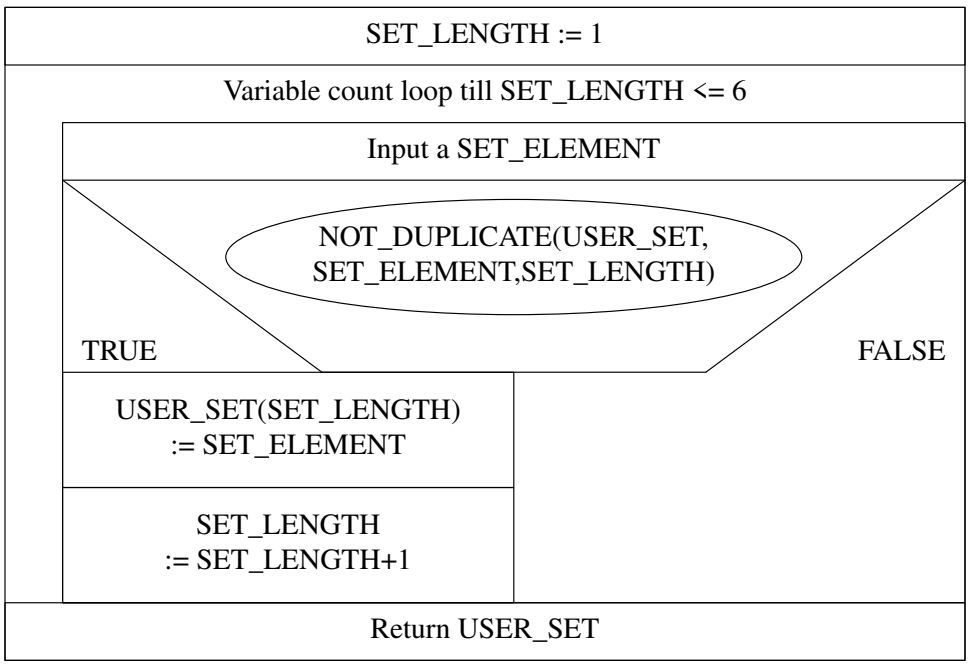
NAME	USAGE	TYPE	RANGE/VALUE
USER_SET	Formal parameter (array)	SET	n.a.
BASE_SET	Local constant (array)	SET	{1, 2, 3, 4, 5, 6}
USER_INDEX	Local variable	INTEGER	Default

4) `NOT_DUPLICATE` (Level 3 Boolean function): Check that element has not previously been included in the user defined set. If so return `TRUE`, `FALSE` otherwise.

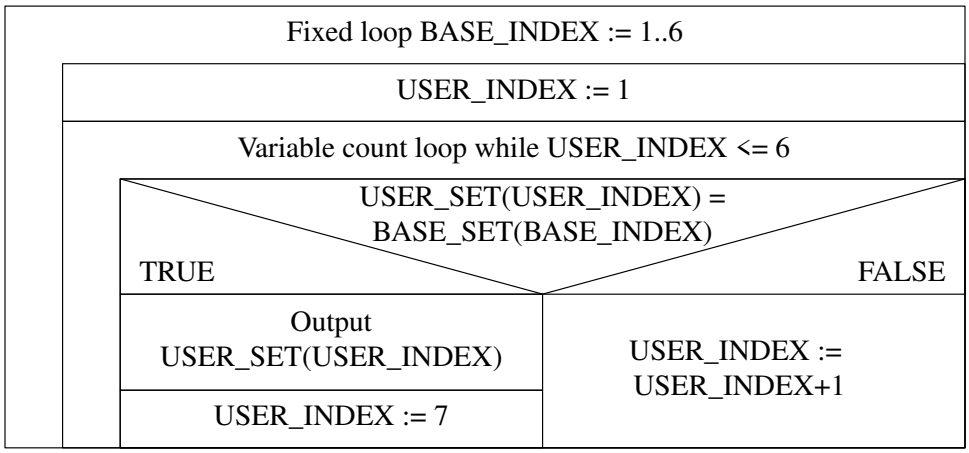
NAME	USAGE	TYPE	RANGE
USER_SET_SOFAR	Formal parameter (array)	SET	n.a.
INPUT_ELEMENT	Formal parameter	INTEGER	Default
LENGTH_SOFAR	Formal parameter	INTEGER	Default
SET_INDEX	Local variable	INTEGER	Default
RETURN_VALUE	Local variable	BOOLEAN	Default

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

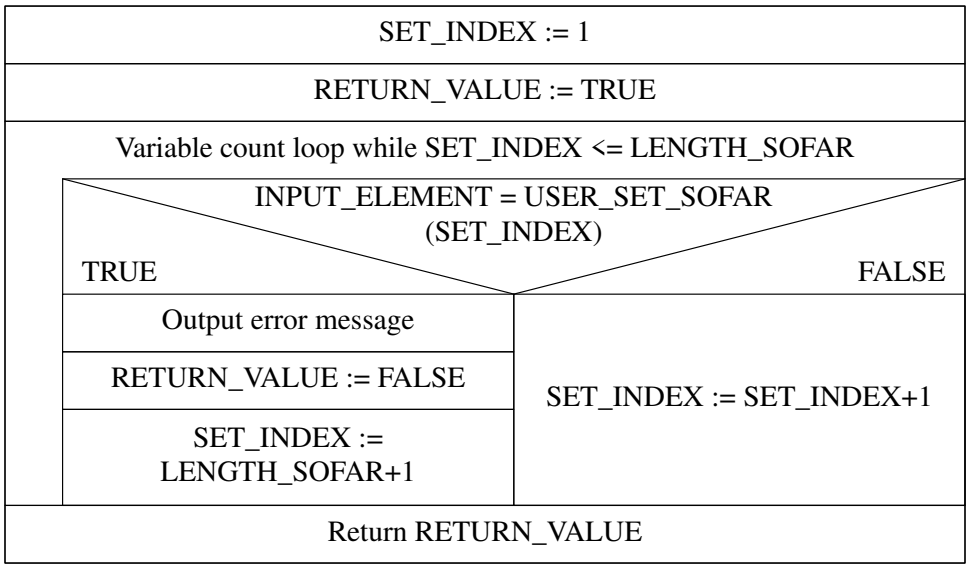
INTERSECTION
(Top Level Procedure)



INPUT_USER_SET
(Level 2 function)



IDENTIFY_INTERSECTION
(Level 2 Procedure)



DUPLICATE_ELEMENT
(Level 3 function)

Table 2*: Nassi-Shneiderman chart*

3. Implementation

The encoding of the above design is given in Table 3.

Table 3: *Implementation*

```
    -- IDENTIFY_INTERSECTION
    procedure IDENTIFY_INTERSECTION(USER_SET : SET_T) is
        BASE_SET   : constant SET_T := (1, 2, 3, 4, 5, 6);
        USER_INDEX : INTEGER;
    begin
        PUT("Intersection = { ");
        for BASE_INDEX in SET_T'FIRST..SET_T'LAST loop
            USER_INDEX := START_INDEX;
            while USER_INDEX <= END_INDEX loop
                if USER_SET(USER_INDEX) = BASE_SET(BASE_INDEX) then
                    PUT(USER_SET(USER_INDEX), width=>1);
                    PUT(" ");
                    USER_INDEX := END_INDEX+1;
                else
                    USER_INDEX := USER_INDEX+1;
                end if;
            end loop;
        end loop;
        PUT_LINE(" }");
    end IDENTIFY_INTERSECTION;
    ------------------------------------------------------------------------


-- INTERSECTION top level procedure
begin
    IDENTIFY_INTERSECTION(INPUT_USER_SET);
end SET_INTERSECTION;
```

Table 3 continued

4. Testing

4.1. Black box testing

BVA, Limit and arithmetic testing: Inputs may range from the maximum to the minimum default range for the integer type. We should also include a zero test value. A suitable set of black box test cases is given in Table 4. The result is given in Table 5.

TEST CASE USER_SET	EXPECTED RESULT OUTPUT
-2147483649 ... 2147483648 ... -2147483648 -2147483647 0 2147483646 2147483647 1	DATA ERROR DATA ERROR 1

Table 4: Black box test cases

```
kuban-154 $ set_intersection
Input a set of 6 (integer) elements: -2147483649

Ada-runtime: Exception DATA_ERROR propagated out of main.
kuban-155 $ set_intersection
Input a set of 6 (integer) elements: 2147483648

Ada-runtime: Exception DATA_ERROR propagated out of main.
kuban-156 $ set_intersection
Input a set of 6 (integer) elements: -2147483648 -2147483647 0 2147483646
2147483647 1
Intersection = { 1  }
```

Table 5: Black box test result

4.2. White box testing

Path Testing: We should test each path through the code. More particularly we should test where:

1) Input is a duplicate and is not a duplicate of previous inputs.

2) Input is and is not a member of the `BASE_SET`.

A suitable test `USER_SET` would then be of the form given in Table 6. The result is given in Table 7.

TEST CASE USER_SET	EXPECTED RESULT OUTPUT
0 2 3 0 1 7 -2	2 3 1

Table 6: White box test cases

```
kuban-157 $ set_intersection
Input a set of 6 (integer) elements: 0 2 3 0 1 7
Duplicate input (0)
-2
Intersection = { 1 2 3  }
```

Table 7: White box test result

4.3. Data Validation Testing

Results of some spurious data validation testing are given in Table 8.

```
kuban-158 $ set_intersection
Input a set of 6 (integer) elements: frans
Ada-runtime: Exception DATA_ERROR propagated out of main.

kuban-159 $ rans
ksh: rans:  not found

kuban-160 $ set_intersection
Input a set of 6 (integer) elements: 1 2 3 4.5
Ada-runtime: Exception DATA_ERROR propagated out of main.

kuban-161 $ 5
ksh: 5:  not found
```

Table 8: Data validation testing results