Generating the Subsets of a Set.

(i.e. Generate all the elements in the Powerset)

For ease of presentation let us write the procedure in pseudo-Eiffel. This will allow us to include any mathematical notation that suits our purpose. In generating subsets we will need the operations Union and Difference.

Given a set of N elements, we want a procedure that will generate all the 2^N subsets.

The critical procedure, Gen_Subsets can be written as

```
Gen_Subsets(i,N : INTEGER) is
    do
        if i > N then
            Print_Set
        else
            S := S / {i}  -- exclude i
            Gen_Subsets(i+1, N)
            S := S ∪ {i} -- include i
            Gen_Subsets(i+1, N)
        end
    end -- Gen_Subsets
```

Let us implement Sets by a zero-one or 'bit' array,

i.e. s : ARRAY[INTEGER]

where s.item(i) = 1 iff i ε s

We can rewrite Gen_Subsets as

```
Gen_Subsets(i,N : INTEGER) is
    local
        bit : INTEGER
    do
        if i > N then
            Print_Set
        else
            from
                bit := 0
            until
                bit > 1
            loop
                s.put(bit,i)
                Gen_Subsets(i+1, N)
                bit := bit+1
            end
    end -- Gen_Subsets
```

In the following class, we use a boolean array to represent a set, as above.

```
class GEN_SETS
creation
    make
feature

    s : ARRAY[BOOLEAN]
-- Integer Set:  s.item(i) iff i ε s

    make is
        local
            N : INTEGER
        do
            io.put_string("%N Enter Size of Set : ")
            io.read_integer
            N := io.last_integer
            !!s.make(1,N)
            io.put_string("%nThe subsets  are %n")
            Gen_Subsets(1,N)
        end -- make

    Gen_Subsets(i,N : INTEGER) is
        do
            if i > N then
                Print_Set
            else
                s.put(False,i) -- exclude i
                Gen_Subsets(i+1,N)
                s.put(True,i)  -- include i
                Gen_Subsets(i+1,N)
            end

        end -- Gen_Subsets
    Print_Set is
        local
            i,k : INTEGER
        do
            from
                io.putchar('{')
                i := 1
                k := 0
            until
                i > s.count
            loop
                if s.item(i) then
                    if k = 0 then
                        io.put_integer(i)
                    else
                        io.putchar(',')
                        io.put_integer(i)
                    end
                    k := k+1
                end
                i := i+1
            end
            io.putchar('}')
            io.new_line
        end -- Print_Set


end --GEN_SETS
```

Another Version for Generating Combinations

To generate all the combinations of k numbers from N numbers, generate all the subsets of {1..N} and output only those of size k.

```
class GEN_COMB
creation
    make
feature
    s : ARRAY[BOOLEAN]
    All_Combs(i,N,k : INTEGER) is
        do
            if i > N  then
                if Setsize = k  then
                    Print_Set
                end
            else
                s.put(True,i) -- include i
                All_Combs(i+1,N,k)
                s.put(False,i)  -- exclude i
                All_Combs(i+1,N,k)
            end
        end -- All_Combs
```

```
Setsize: INTEGER is
    local
        i,counter : INTEGER
    do
        from
            i := 1
            counter := 0
        until
            i > s.count
        loop
            if s.item(i) then
                counter := counter+1
            end
            i := i+1
        end
        result := counter
    end -- Setsize

make is
    local
        N,k : INTEGER
    do
        io.put_string("%N N := ")
        io.read_integer
        N := io.last_integer
        io.put_string("%N k := ")
        io.read_integer
        k := io.last_integer
        !!s.make(1,N)
        io.put_string("%N The Combinations are: %N")
        All_Combs(1,N,k)
    end -- make
```

--The routine Print_Set is the same as above in GEN_SETS

```
    Print_Set is -- as above for GEN_SETS
        local
            i,k : INTEGER
        do
            from
                io.putchar('{')
                i := 1
                k := 0
            until
                i > s.count
            loop
                if s.item(i) then
                    if k = 0 then
                        io.put_integer(i)
                    else
                        io.putchar(',')
                         io.put_integer(i)
                    end
                     k := k+1
                end
                i := i+1
            end
            io.putchar('}')
            io.new_line
        end -- Print_Set

end --GEN_COMB
```