COMPUTER WORD IMPLEMENTATION

COMPUTER WORD → UNIT OF DATA OF DEFINED BIT LENGTH
i.e char is a unit of data worth 8 bits (1 byte)
int is a unit of data worth 32 bits (4 bytes)

Bit vector & computer word
→ most effective running time
→ O(1) running time
→ uses less storage vs. other implementations

Eg.

SET A = 31 → this is represented by a series of bits

	128	64	32	16	8	4	2	1
written like this	0	0	0	1	1	1	1	1

SET X = -25 → when given is negative, do 2's complement.

	128	64	32	16	8	4	2	1
25:	0	0	0	1	1	0	0	1
-25:	1	1	1	0	0	1	1	1

2's complement

To access & manipulate data in computer word, utilize bitwise operators.

INSERT, INITIALIZE, DISPLAY, DELETE

Initialize - just set the whole thing to 0.

Eg. Initialize SET A = 31

	128	64	32	16	8	4	2	1
	0	0	0	1	1	1	1	1
SET A = 0	0	0	0	0	0	0	0	0

Insertion - assign a mask 1 then shift to appropriate position.

Eg SET A = 0, insert [3] in A → $2^{(3)}$

128	64	32	16	[8]	4	2	1
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0

→ mask (1 << 3)

☆ Note: If elements of a set in bit-vector are represented by the indices of the array, in computer word they are represented by the exponents of 2.
(Why 2? Binary 1's & 0's baby)

Display - To display, go bit-by-bit. Use For loop & shifting to check if member exists. We only want to display bits > 0

Eg Display SET A = 31

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	0	0	1	1	1	1	1

← 1

5 bits in total to display, loop starts at 1.

Shifting the mask and performing AND to print the bits that are greater than 0.

Deletion - get the complement and perform AND operation.

Eg. SET A = 171, delete 7.

[2^7]	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	0	1	0	1	0	1	1
1	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1

→ mask for bit in 7th place

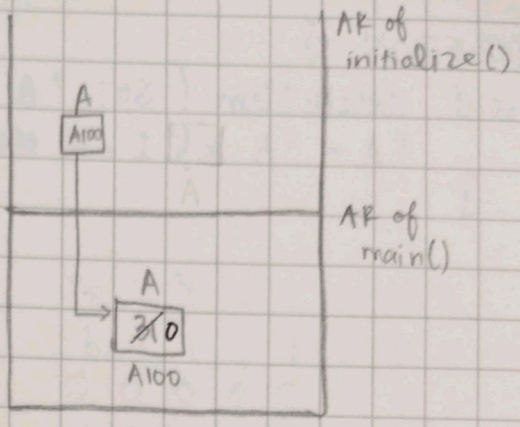
→ complement of mask

171:	1	0	1	0	1	0	1	1
complement of mask:	0	1	1	1	1	1	1	1
	0	0	1	0	1	0	1	1

→ 171 with 7 deleted.

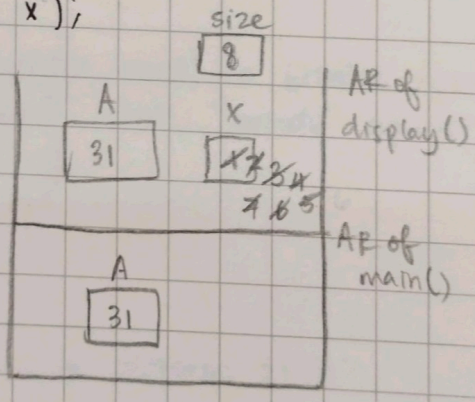
✧ Initialize ✧

```
void initialize (SET *A) {
        *A = 0;
}
```



✧ Display ✧

```
void display (SET A) {
    int x = 1; // making mask shifting counter
    int size = sizeof(datatype) * 8; // datatype e.g char, int, etc.
                                     *8 because 1 byte = 8 bits
    for(x = 1; x <=size; x++) {
        if((A & 1<<x) > 0) {
            printf("%d", x);
        }
    }
}
```

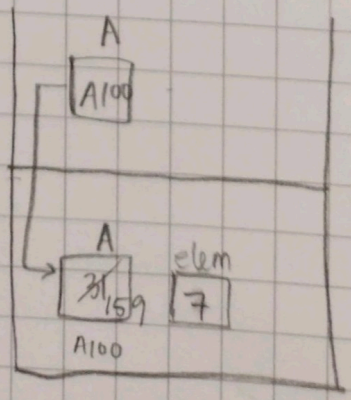


2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	1	1	1	1	1
1	1	1	1	1	1	✗	✗
0	0	0	1	1	1	1	1

Given: SET A = 31 = {0, 1, 2, 3, 4, 0, 0, 0}
Result: {0, 1, 2, 3, 4, 0, 0, 0}

✧ Insertion ✧

```
void insert (SET *A, int elem) {
    *A = *A | 1 << elem;
}
```



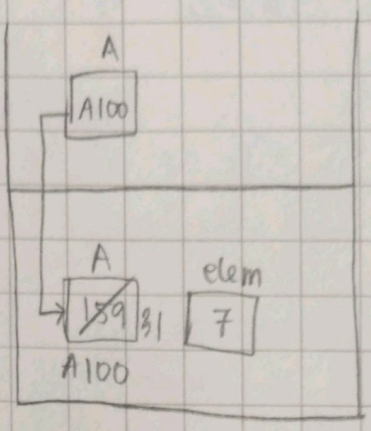
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
31:	0	0	0	1	1	1	1	1
	1	0	0	0	0	0	0	0
159:	1	0	0	1	1	1	1	1

128
31
159

✧ DELETION ✧

```
void deleteElem ( SET *A, int elem) {
    *A = *A &!( 1 << elem);
}
```

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
159:	1	0	0	1	1	1	1	1
	1	0	0	0	0	0	0	0
	0	1	1	1	1	1	1	1
31 :	0	0	0	1	1	1	1	1



✧ isMember ✧

```
int isMember (SET A, int elem) {
    return ( A & 1 << elem) > 0;
}
```

COMPUTER WORD BIT OPERATIONS

Eg. GIVEN:

SET A = { 1, 3, 5, 7 }

SET B = { 3, 4, 5 }

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
A:	0	0	1	0	1	0	1	0
B:	0	0	1	1	1	0	0	0

✧ UNION ✧

	7	6	5	4	3	2	1	0
A:	1	0	1	0	1	0	1	0
B:	0	0	1	1	1	0	0	0
	1	0	1	1	1	0	1	0

→ perform OR

$A \cup B = \{ 1, 3, 4, 5, 7 \}$ ✓

```
SET setUnion (SET A, SET B) {
    return A | B;
}
```

↳ since computer word, the bitwise operator will compare whole "words" against each other

✧ INTERSECTION ✧

	7	6	5	4	3	2	1	0
A:	1	0	1	0	1	0	1	0
B:	0	0	1	1	1	0	0	0
	0	0	1	0	1	0	0	0

→ perform AND

$A \cap B = \{ 3, 5 \}$ ✓

```
SET setIntersection (SET A, SET B) {
    return A & B;
}
```

SET A = { 1, 3, 5, 7 }
SET B = { 3, 4, 5 }

```
   7 6 5 4 3 2 1 0
A: 1 0 1 0 1 0 1 0
B: 0 0 1 1 1 0 0 0
```

✧ DIFFERENCE ✧

* A − B

get B's complement : 1 1 0 0 0 1 1 1

```
    7 6 5 4 3 2 1 0
A:  1 0 1 0 1 0 1 0   ⟶ perform AND
~B: 1 1 0 0 0 1 1 1
   ---------------
    1 0 0 0 0 0 1 0
```

A − B = { 1, 7 } ✓

```
SET setdifference (SET A, SET B) {
    return A & ~ B;
}
```

✧ SUBSET ✧

```
   7 6 5 4 3 2 1 0
A: 1 0 1 0 1 0 1 0   ⟶ Intersect the two sets.
B: 0 0 1 1 1 0 0 0     If it results in the smaller
   ---------------     set, then it is a subset of
   0 0 1 0 1 0 0 0     the larger set.
```

B is not a subset of A.

```
                    * is B subset of A?
int isSubset (SET A, SET B) {
    return (A & B) == B
}
```