NO:

DATE:

- ADTs based on SETS -

* SET

- unique elements
- order DOESN'T MATTER (unless specified)

* LIST

- can have duplicate elements
- order is IMPORTANT
  - ↳ like in Array list: shifting of index

* Types

1.) ADT UID (Union, Intersection, Difference)

2.) Dictionary

3.) Prionty Queve

* Implementations

1.) Linked List

2.) Array Implementation

3.) Cursor-based

4.) Bit-vector Implementation

* Operations

1.) Union

2.) Intersection

3.) Difference

4.) MAKENULL

5.) INITIALIZE

	A	B
0	1	1
1	0	0
2	0	0
3	1	0
4	1	1
5	0	1
6	0	0
7	1	0
8	0	0
9	1	0

Set A = {3, 7, 0, 4, 9}

Set B = {0, 4, 5}

U = {0, 1, 2, ..., 9}

*List range will be the entirety of the universal set, regardless if the elements are present or not.

NO:

DATE:

- BIT-VECTOR IMPLEMENTATION

* Bit-vector Implementation

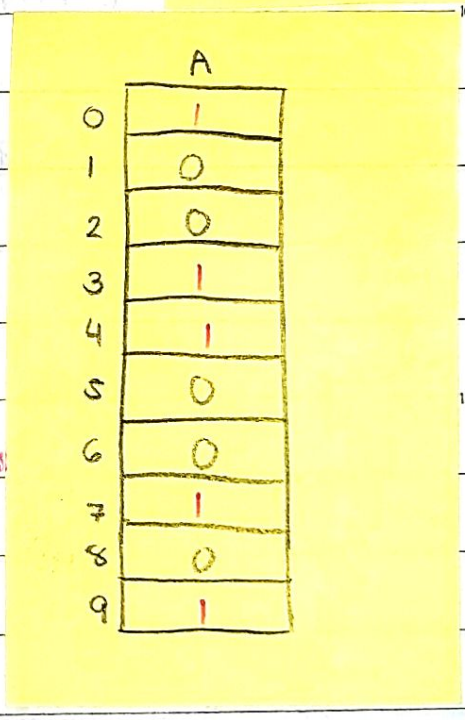
- also known as Boolean array implementation (T, F)
- size of set is dependent on Universal Set
  - ↳ all given sets are SUBSETS
- best used:
  - > when Universal set U is SMALL
  - > elements are INTEGERS (or can be mapped to integers)

Example:

set A = { 3, 7, 0, 4, 9 }

A[x] = 1 if $x \in A$

A[x] = 0 if $x \notin A$



* Place a 1 for every elements that
  = to index in set A.
* Assign a UNIQUE integer number for each non-integer element!

* Advantages

1.) member (), insert (), delete () = O(1)

2.) union (), intersection (), difference() = O(N)

3.) If > U is sufficiently small & = UID can be performed in one logical operation
  - > can fit in one computer word

* Disadvantages

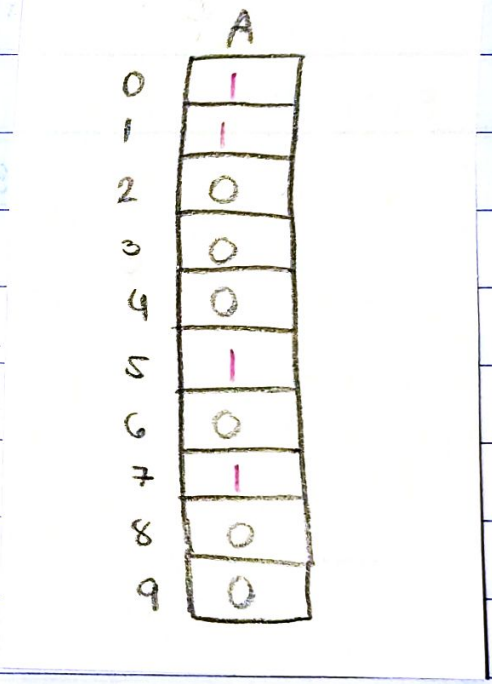
1.) Space needed = size of U

If Universal Set U is relatively big, then size of set will also be big.

* Insert

U = {0, 1, 2, ..., 9}

A = {1, 0, 5}



• Insert (A, 7)

A = {1, 0, 5, 7}

* Union

- combine the sets

A = {1, 0, 5, 7}

B = {0, 4, 5}

∴ Union (A, B)

A ∪ B = {0, 1, 5, 4, 7}

DATE:

* Set Operations

1.) UNION ()

- belongs to A or B // BOTH / COMBINE
- A ∪ B

2.) INTERSECTION ()

- common to A and B //
- A ∩ B

3.) DIFFERENCE ()

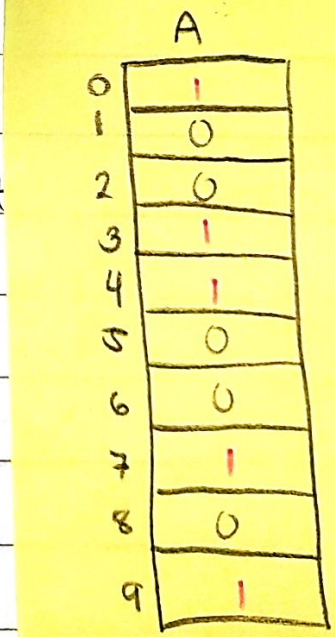
- element in A, but not in B
- A - B

- SET OPERATIONS : UNION - (BVI)

// Exercise 1 - Union ()

- Definition of datatype SET such that A is an array of integers of size 10.

```
#define SIZE 10
typedef int SET [SIZE];

SET A; → = A[10]
```



A = {0, 3, 4, 7, 9}

- UID OPERATIONS - VISUALIZATION

* UID

- **Union** - combine all elements

SET A = {7, 0, 4, 2}
B = {0, 1, 5}

A∪B = {0, 1, 5, 2, 4, 7}

	7	6	5	4	3	2	1	0	
	1	0	0	1	0	1	0	1	= A
OR (|)	0	0	1	0	0	0	1	1	= B
	1	0	1	1	0	1	1	1	= {0, 1, 2, 4, 5, 7}

- **Intersection** - common elements

SET A = {7, 0, 4, 1}
B = {0, 1, 5}

A∩B = {0, 1}

	7	6	5	4	3	2	1	0	
	1	0	0	1	0	0	1	1	= A
AND (&)	0	0	1	0	0	0	1	1	= B
	0	0	0	0	0	0	1	1	= {0, 1}

- **Difference** - element in A, NOT in B

SET A = {7, 0, 4, 1}
B = {0, 1, 5}
A - B = {7, 4}

* get complement of B FIRST.

0 0 1 0 0 0 1 1 → (~ COMPLEMENT) → 1 1 0 1 1 1 0 0

* perform AND

	7	6	5	4	3	2	1	0	
	1	0	0	1	0	0	1	1	= A
(&)	1	1	0	1	1	1	0	0	= B
	1	0	0	1	0	0	0	0	= {7, 4}

• **Subset** - if set's every element is in another set

SET A = {7, 0, 4, 1} $B \subseteq A$

B = {0, 1, 4}

```
  7 6 5 4   3 2 1 0
  1 0 0 1   0 0 1 1   = A
& 0 0 0 1   0 0 1 1   = B
-----------------------
  0 0 0 1   0 0 1 1   != 0   ✓ B is a subset of A.
```

- SET OPERATIONS -
VISUALIZATION

//member () - returns 1 / non-zero if element is member
0 if not member

SET A = { 2, 4, 5 }

X = 5

*5 is a mask value that was shifted to that position from 1.

```
7654 3210
0011 0100  = A
0010 0000  = 5
---------
0010 0000  != 0 ✓
```

MEMBER

```
0001 0100  = A
0010 0000  = 5
---------
0000 0000  == 0 ✗
```

NOT MEMBER

mask value = 0000 0001 << X

//insert () - insert element at appropriate position matching bit position

SET A= { 7, 0, 4, 1 }

X = 5

OR

```
7654 3210
1001 0011  = A
0010 0000  = 5
---------
1011 0011  = { 7, 0, 4, 1, 5 }
```

//delete () - delete element at appropriate position matching bit position

SET A = { 7, 0, 4, 1, 5 }

X = 5

*get complement of X
*perform AND

```
7654 3210
1011 0011  = A
~0010 0000 = 5
```

```
1011 0011  = A
&1101 1111 = 5
---------
1001 0011  = { 7, 0, 4, 1 }
```

• Function Header : Union() - will return new set C w/ elements found in given sets.

pointer to locally declared array

SET * Union (SET C, SET D);

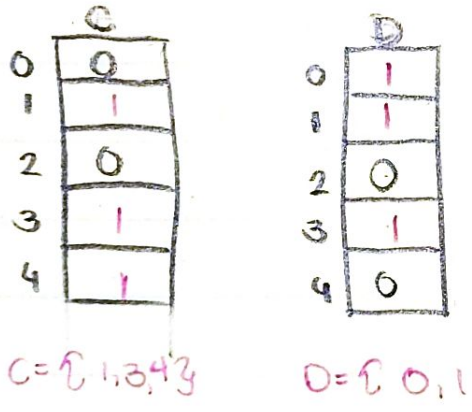
• Function Call (w/ variable declaration & initialization)

// Assumption: U = {0,1,2,3,4} D = {0,1,3}

C = {1,3,4} CUD = {0,1,3,4}

* C&D in main()
A&B in Union()

* shortened to SIZE = 5 for convenience



C = {1,3,4} D = {0,1,3}

- Declaration & Initialization!

SET C = {0,1,0,1,1}

SET D = {1,1,0,1,0}

REMEMBER, keep declaration & initialization TOGETHER!

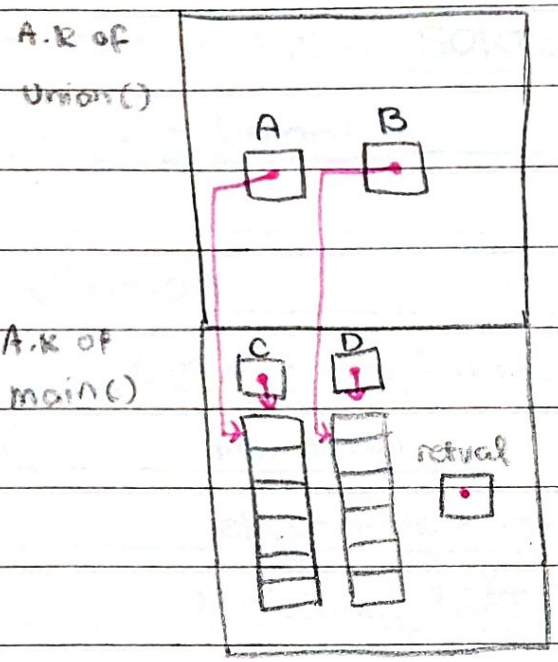
can be omitted

SET * retval = Union(C, D);

* Make a catcher variable bcs you're returning a new set.
* its a pointer bcs it will point to the locally created dynamically allocated array in the function.

DATE:

• Execution Stack



* The name of an array is a POINTER to first component.
* A & B will get the address of C[0] & D[0]
* Array must be dynamically allocated when returned.

```
#define SIZE 5
typedef int SET [SIZE];
```

• Code

```
SET * Union (SET A, SET B) {
    SET *C = (SET *) calloc (MAX, sizeof (SET));
    int x;

    if (C != NULL) {
      for ( x=0; x<SITE; x++) {
    (*C)[x] = (A[x] == 1 || B[x] == 1) ? 1 : 0 ;
      }        OR:
    }          (*C)[x] = A[x] || B[x];
}
```

→ initializes array to be all 0 so just change it to 1; eliminates need for else.

EXECUTION STACK (after)

- SET OPERATIONS: UNION - (LINKED LIST)

DATE:

- uses size of set represented, not size of Universal set
- better if list is SORTED

// Exercise 2 - Union()

· Definition

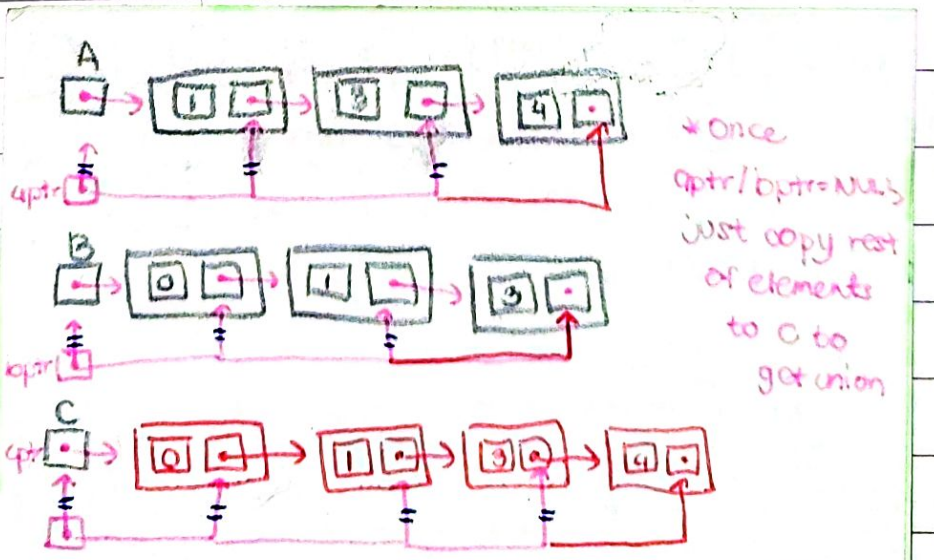
```
typedef struct node{
    int data;
    struct node * link;
} Nodetype, *SET;
```

```
SET A; // {1,3,4}
SET B; // {0,1,3}
        A∪B = {0,1,3,4}
SET C;
C = union(A,B); //pass by copy
```

· Code

```
SET Union(SET A, SET B) {
SET C, *cptr, aptr, bptr;
C=NULL;  cptr = &C;
aptr = A;  bptr = B;

while (aptr != NULL && bptr != NULL) {
  if (aptr->data < bptr->data) { // A<B
    *cptr = (SET) malloc(sizeof(struct node));
    (*cptr)->data = aptr->data;
    aptr = aptr->link;
  }
  else if(bptr->data < aptr->data) { // B<A
    (*cptr)->data = bptr->data;
    bptr = bptr->link;
  }
```

```
else {                                  // EQUAL
        *cptr -> elem = aptr -> elem;
        aptr = aptr -> link;
        bptr = bptr -> link;
    }
    cptr = &(*cptr) -> link;
}

if(aptr == NULL) {
    aptr = bptr;
}

for(; aptr != NULL; aptr = aptr -> link, cptr = &(*cptr) -> link) {
    *cptr = malloc(sizeof(struct node));
    (*cptr) -> elem = aptr -> elem;
}

*cp = NULL;
return C;
```



1. Declare aptr, bptr (PtN) bcs just traversing
   - aptr (PPN) bcs inserting
   - C (new list/set)

2. Loop checking process until either ptr becomes NULL.

3. Check if a < b (bcs sorted), if a is smaller, insert first to C.
   - update *aptr & aptr

NO:

DATE:

- COMPUTER WORD IMPLEMENTATION

* Computer Word

* manipulating BITS.

- unit of data of defined bit length
- can be moved & addressed
- can be:

char	8 bits
short int	16 bits
int	32 bits

Example:

char ch = 'A'

char x = -25

ch = _ _ _ _ _ _ _ _ 8 bits
128 64 32 16 8 4 2 1

'A' = 65 (ASCII)

$-\frac{1}{64}\ \frac{}{}\ \frac{}{}\ \frac{}{}\ \frac{1}{1} = 65$

$25 = \frac{0001}{16\ 8}\ \frac{1001}{4\ 2\ 1} = 25$

* Operations

1.) UID
2.) member ()
3.) insert()
4.) delete()

↓ COMPLEMENT (to get -25)

= 0001 1001 → leave first 0 & 1 unchanged (none, so skip)

1110 0110 → reverse bits

= 1110 0111 → add 1

-25 = 1110 0111 (in computer's memory)

NO:

DATE:

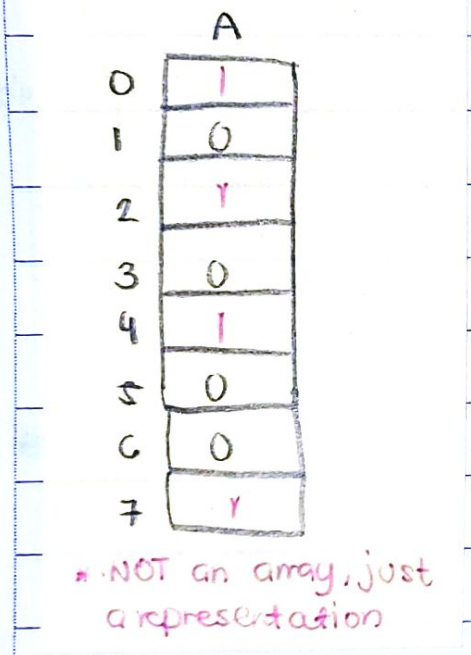
//display Bit Pattern

Write the code of the fn() that will display bit pattern of a given integer.

*RECALL

In Bit-vector, implementation,

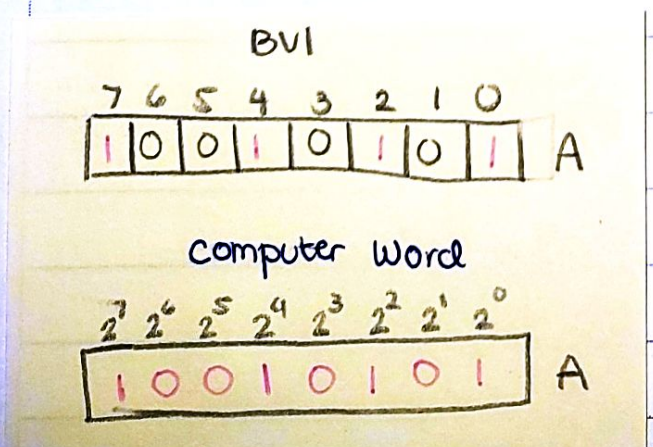
$U = \{0,1,2,3,4,5,6,7,8,9\}$ $A = \{7,0,4,2\}$



* NOT an array, just a representation

- **Facts**
  - contents are bits (0,1)
  - instead of a "bit" datatype, we use char/int
  - elements of Universal Set are indices of array
  - can FIT INSIDE char since char = 8 bits

* BVI to computer Word



* Elements are INDEX.

* Elements are EXPONENTS.

- If $x \in A$ then bit position w/ place value 2^x is 1.

Ex: $4 \in A$
bit position: 2^4 is 1

NO:

DATE:

· Code

```
void displayBitPattern (SET A) {
    int x;
    int SIZE = sizeof (SET) * 8;

    for (x=1; x<SIZE; x++) {
        if ( A & 1<<x) > 0) {
            printf(" %d", x);
        }
    }
}
```

bitwise AND

// to print what numbers are IN the set.

OR

```
if (A & 1 << x) > 0) {
    printf("%d", 1);
} else
    printf("%d", 0);
}
```

// to print 0000 1111 pattern of the set.

① set x = 1 → MASK VALUE
fn. will shift from here

② calculate size of datatype.
-remember, must be platform-independt

③ Check bit pattern of A against
1 << x. (shift 1 place, 2 places)

④ Print.

NO:

DATE:

- COMPUTER WORD OPERATIONS (VISUALIZATION)

```
typedef unsigned
char SET;
```

$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

_ _ _ _ _ _ _ _ | A

// member() - VISUALIZATION

- must EXTRACT to retrieve element → mask value
- checks if element (exponent) is a MEMBER of set (bits)

Example: A = { 5, 4, 2 }

7 6 5 4 3 2 1 0

= 0011 0100

AND operator

```
  0011 0100
& 0000 0100   → mask value
-----------
  0000 0100 != 0
  MEMBER
```

```
  0011 0000
& 0000 0100
-----------
  0000 0000 == 0
  NOT MEMBER
```

mask value: 0000 0001 << 1

* Move/SHIFT the mask value & use it to check against the elements in set A to check if elements are present.
* Mask value should always start at 1 (0000 0001) & shift LEFT! Function will check against each bit per iteration.
* If resulting bit pattern != 0 (means it found an element in that position, so 2^2 exists)
  == 0 (means element does not exist in that bit position

NO:

DATE:

- COMPUTER WORD OPERATIONS - CODE

typedef unsigned char SET;

*UID

// Union (A, B) - returns A∪B

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

// Intersection (A, B) - returns A∩B

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

// Difference (A, B) - returns A - B

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

// Subset. (A, B) - returns 1 /nonzero if B ⊆ A ; 0 otherwise

```
int subset (SET A, SET B) {
    return (A & B) == B;
}
```

DATE:

*Set Functions

// initialize (*A)

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

// member (A, x)

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

// insert (*A, x)

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

// delete (*A, x)

```
SET delete (SET *A, int x) {
        *A = *A & ~(1<<x);
}
```