ADT Based on SETS

SET → collection of elements where each member is distinct, unique. No two elements that are the same are contained with a set.

SET vs. LIST

SET	vs.	LIST
• unique elements		• elements can be duplicated
• order is not significant (unless specified)		• order is important

"ADT's based on SETS" aka no duplicated elements in the ADTs
- ↳ ADT UID
- ↳ Dictionary
- ↳ Priority Queue (save for later)

SET UID –

SET ADT with operations
1. Set Union
2. Set Intersection
3. set Difference

IMPLEMENTATIONS

(a) Array
(b) Linked List - sorted or unsorted
(c) Cursor Based
(d) Bit-vector implementation

BIT VECTOR IMPLEMENTATION

- the set is represented by a bit vector ('bit' i.e binary digit - the component of a 1-D vector, 'vector' i.e a 1-D array) in which the n^{th} bit is true or '1' if n is part of the set.

Eg. SET A = {3, 7, 0, 4, 9}

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

$A[x] = 1$ if $x \in A$
$A[x] = 0$ if $x \notin A$

When to use Bit-vector implementation?

- when the elements are integers because they represent the indices of the set array
- when the universal set u is small because the size of the set is dependent on the size of the universal set.

BIT VECTOR IMPLEMENTATION PROS VS. CONS.

PROs.

★ isMember(), insert() & delete() can be performed with O(1) by directly addressing the appropriate bit.

★ Union(), intersection() & difference() can be performed in time proportional to the size of the universal set

★ if universal set is small & can fit a computer word, then UID operations can be performed with one logical operation

CONS.

★ Space needed for the set is proportional to the size of the universal set.
If universal set is large then more space is needed.

SET Union - set of all elements which belong to A or to B or to both. $A \cup B \rightarrow$ OR || |

SET Intersection - set of all elements that are common to A and to B. $A \cap B \rightarrow$ AND &&

SET Difference - all elements in A but no in B.
$A - B \rightarrow$ NOT ! ~

EXERCISE - WRITE THE CODE OF UNION

SET A

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

① Write the appropriate definition of datatype SET such that in the declaration SET A, A is an array of integers of size 10.

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

② Write the code of Union, given two sets C & D.

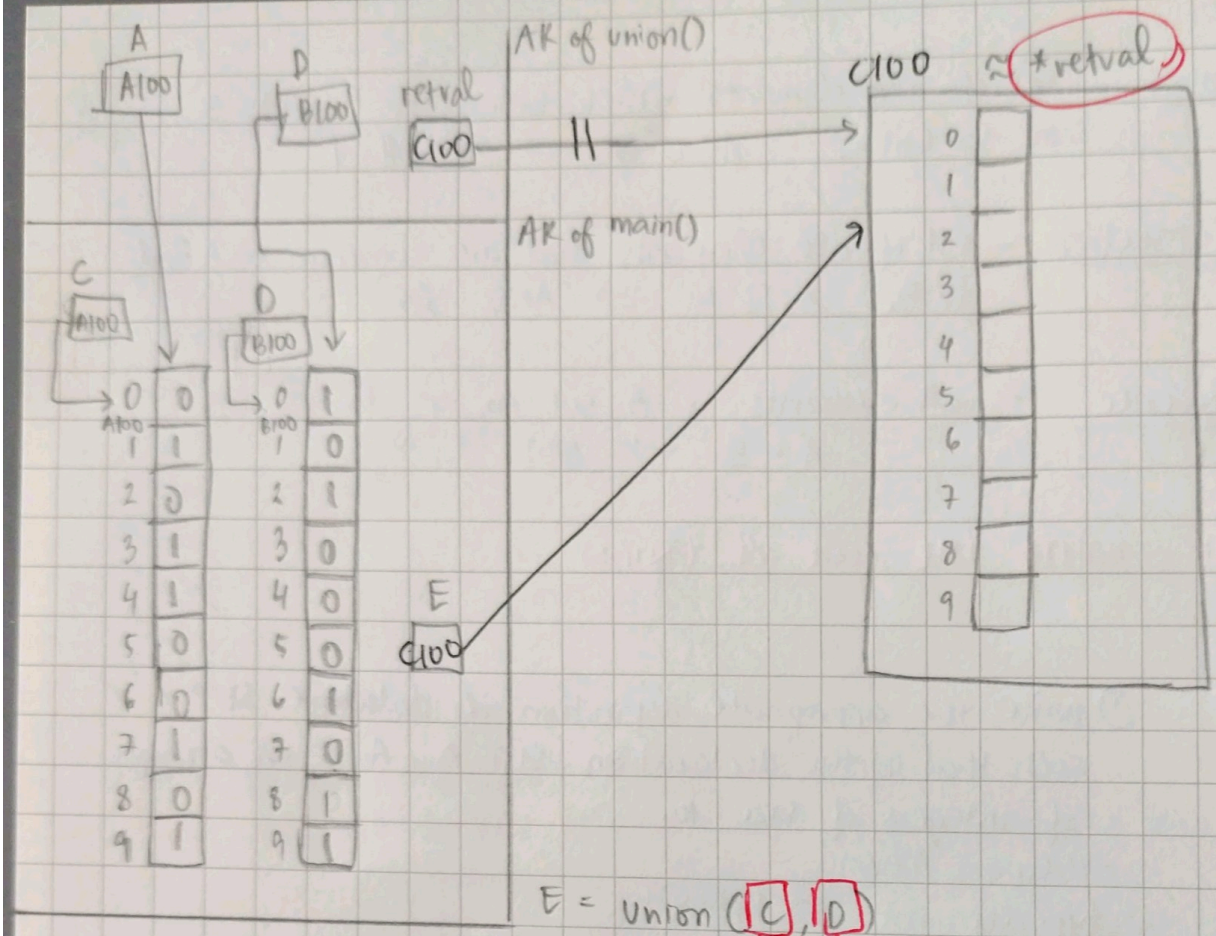
C = { 1, 3, 4, 7, 9 }
D = { 0, 2, 6, 8, 9 }

Initialize & declare the variables used in the function call & write the function call.
Draw the execution stack of the function call.

```
SET C = {0,1,0,1,1,0,0,1,0,1}
SET D = {1,0,1,0,0,0,1,0,1,1}
SET * E;
  E = Union (C, D);
```

→ to initialize an (or any aggregate variable) array, always declare it on the same line

→ do not initialize a function call on the same line as declaration. Do it separately.



E = Union (C, D)

because SET is an array,
we return a pointer to the array

the name of an array is
a pointer to the first component

```
SET * Union (SET A, SET B){// A and B are pointers to the first component
                           of their SET
  SET *retval = (SET *) malloc (sizeof (SET));
  catcher variable, retval is a pointer to an entire array, not the
    first component
  int ndx;
  for (ndx = 0; ndx < MAX; ndx ++){
    (*retval) [ndx] = A [ndx] || B [ndx];
  }
  return retval;
}
```

x	y	x || y
0	0	0
0	1	1
1	0	1
1	1	1

EXERCISE - WRITE THE CODE OF BIT VECTOR IMPLEMENTATION OPERATIONS INTERSECTION & SET DIFFERENCE

A] INTERSECTION

```
SET * intersect (SET A, SET B) {
    SET * retval = (SET *) malloc (sizeof(SET));
    int ndx;
    for (ndx = 0; ndx < MAX; ndx++) {
        (*retval)[ndx] = A[ndx] && B[ndx];
    }
    return retval;
}
```

B] SET DIFFERENCE

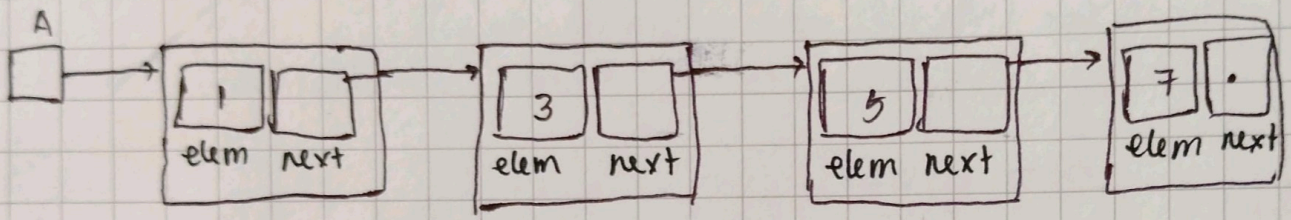
```
SET * Difference (SET A, SET B) {
    SET * retval = (SET *) malloc (sizeof (SET));
    int ndx;
    for (ndx = 0; ndx < MAX; ndx++) {
        (*retval)[ndx] = A[ndx] && ! B[ndx];
    }
    return retval;
}
```

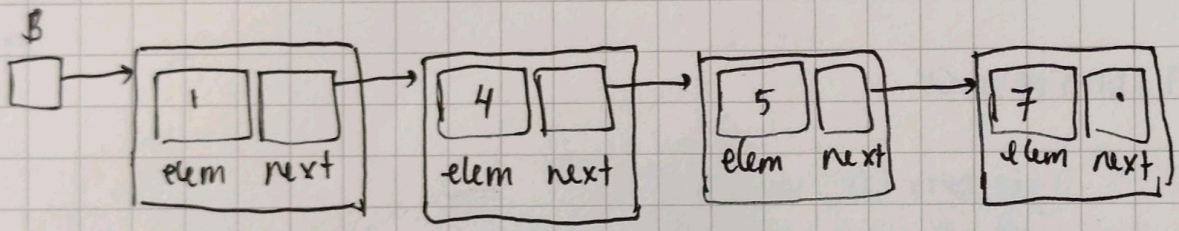
LINKED LIST (POINTER) IMPLEMENTATION)

```
typedef struct cell {
        int elem;
        struct cell * next;
    } ctype, * SET;
```

GIVEN:

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





Write the codes For Union, Intersection, Difference & is Subset.

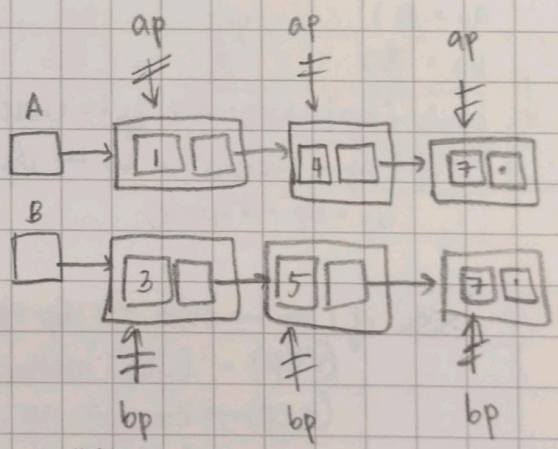
UNION

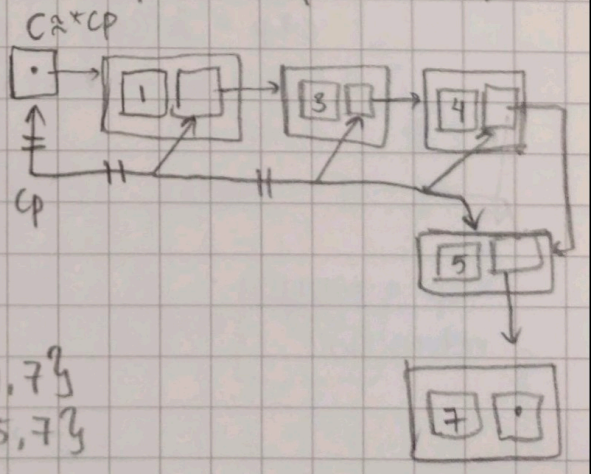
Subtasks:

1. Assign pointers to the First node For the 2 given set
2. Initialize a 3rd set (the list that contains the union) & assign a pointer to the head.
3. Start traversing the sets & compare the elems. If one elem is lesser than the other, copy the data into the new list.
4. Assign the pointers to point to the next node. For the set to be returned, set up the pointer to point to the next Field.
5. If one list reaches null, copy the rest of the remaining data into the new linked list.
6. NULL the "next" field of the new linked list.
7. Return to the calling function.

UNION

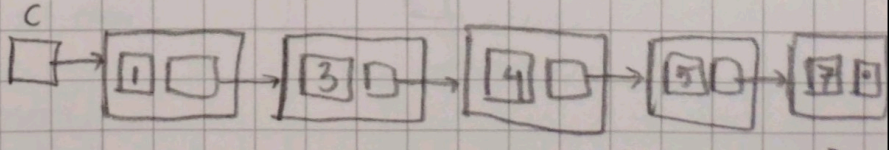
```
SET Union (SET A, SET B) {
    SET C, *cp, ap, bp;
    C = NULL;
    cp = &C;
    ap = A;
    bp = B;
    while (ap != NULL && bp != NULL) {
        (*cp) = (SET) malloc (sizeof (ctype));
        if (ap -> elem < bp -> elem) {
            (*cp) -> elem = ap -> elem;
            ap = ap -> next;
        }
        else {
            if (ap -> elem == bp -> elem) {
                ap = ap -> next;
            }
            (*cp) -> elem = bp -> elem;
            bp = bp -> next;
        }
        cp = &(*cp) -> next;
    }
    if (bp != NULL) { // A came to NULL
        ap = bp;
    }
    while (ap != NULL) {
        (*cp) -> elem = ap -> elem;
        ap = ap -> next;
        cp = &(*cp) -> next;
    }
    *cp = NULL;
    return C;
}
```





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

RESULT:

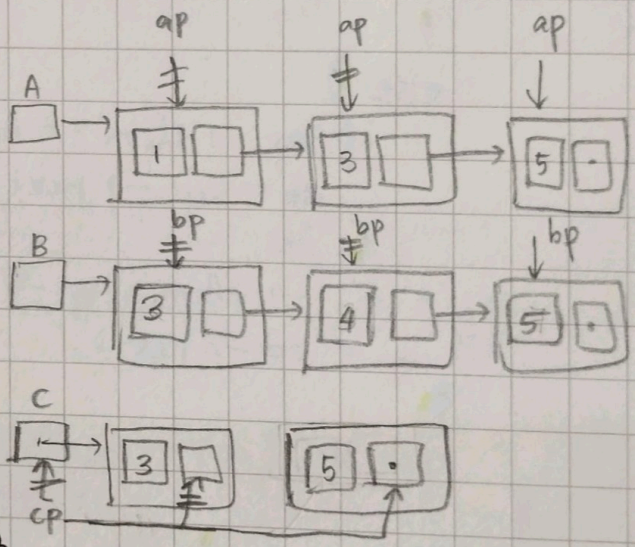


INTERSECTION

Subtasks:
1. Assign pointers to the first node for the 2 given sets
2. Initialize a 3rd set (the intersection set i.e set to be returned) & assign pointer to the head.
3. Start comparing elements in both sets. If one element is greater than the other, traverse to the next element.
4. Else if both elements are equal, copy the data into the new list. Traverse both sets.
5. When one set reaches NULL, exit loop
6. NULL the "next" field of the new linked list
7. Return to the calling function.

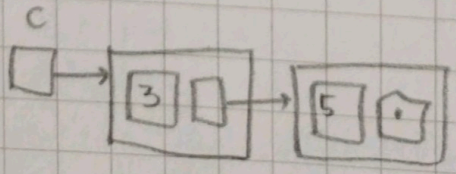
INTERSECTION

```
SET Intersection (SET A, SET B) {
    SET C, *cp, ap, bp;
    C = NULL;
    cp = &C;
    ap = A;
    bp = B;
    while (ap != NULL && bp != NULL) {
        if (ap → elem > bp → elem) {
            bp = bp → next;
        }
        ~~else~~ if (ap → elem == bp → elem) {
            (*cp) = (SET) malloc (sizeof (ctype));
            (*cp) → elem = ap → elem;
            cp = &(*cp) → next;
            bp = bp → next;
        }
        ap = ap → next;
    }
    *cp = NULL;
    return C;
}
```



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

RESULT:



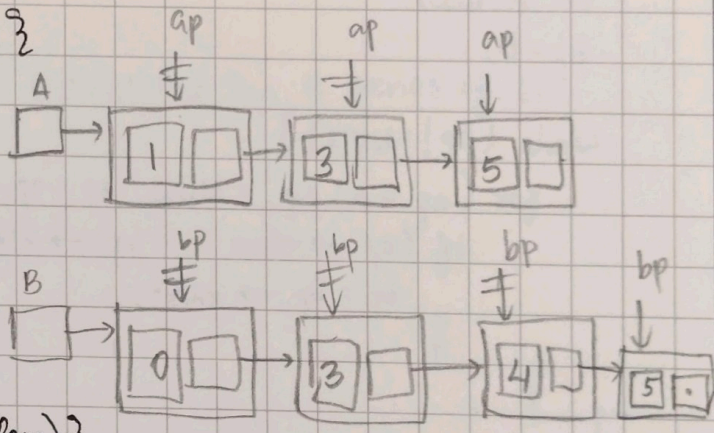
DIFFERENCE

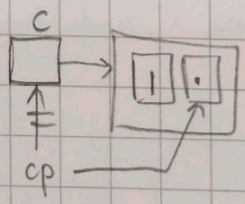
Subtasks :
1. Assign pointers to the first node for the 2 given sets
2. Initialize a 3rd set (the difference set i.e set to be returned) & assign pointer to the head.
3. start comparing elements in both sets. If both elements are the same, traverse both sets.
4. If one element is lesser than the other, traverse one set. If greater, copy the data into the new list.
5. When one set reaches NULL, copy the rest of the remaining data into the new list.
6. NULL the "next" field of the new linked list
7. Return to the calling function.

DIFFERENCE A−B

```
SET Difference (SET A, SET B) {
    SET C, *cp, ap, bp;
    C = NULL;
    cp = &C;
    ap = A;
    bp = B;
    while (ap != NULL) {
        while (bp != NULL) {
        if (ap → elem == bp → elem) {
            ap = ap → next;
            bp = bp → next;
        }
        if (ap → elem > bp → elem) {
            bp = bp → next;
        }
        else {
            (*cp) = (SET) malloc (sizeof (ctype));
            (*cp) → elem = ap → elem;
            ap = ap → next;
            cp = &(*cp) → next;
        }
    }
    *cp = (SET) malloc (sizeof(ctype));
    (*cp) → elem = ap → elem;
     ap = ap → next;
     cp = &(*cp) → next;
}
        *continue*
```

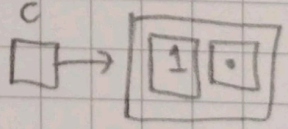
```
    *cp = NULL;
    return C;
}
```





A = {1, 3, 5}
B = {0, 3, 4, 5}

RESULT:

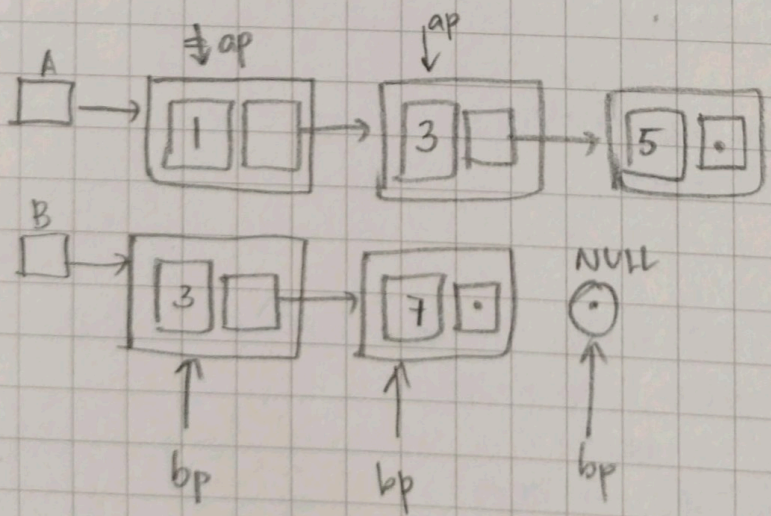


Subtasks:
1. Traverse through one set until the end
2. If element is found in the second set, move pointer to next node
3. If 2nd pointer reaches NULL, the set is a subset of the larger set.
4. Return 1 or 0

SUBSET is B subset of A?

```
int isSubset(SET A, SET B){
    SET ap, bp;
    int retval;
    bp = B;

    while (bp != NULL){
        for (ap = A; ap != NULL; ap = ap → next){
            if (ap → elem == bp → elem){
                bp = bp → next;
            }
        }
    }
    return (bp == NULL ? 1 : 0);
}
```



A = {1, 3, 5}
B = {3, 5}

RESULT: 1, B is a subset of A