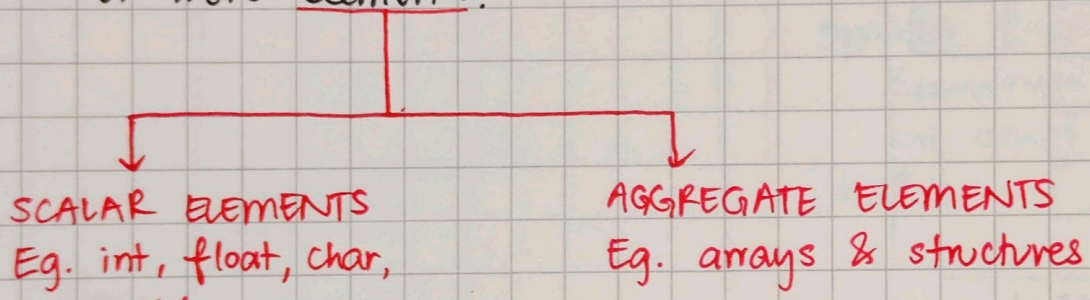
ADT - ABSTRACT DATATYPES

ADT ⟶ a mathematical model with a set of operations defined on the model.

Eg. • A LIST IS A SET OF INTEGERS WHOSE OPERATIONS ARE INTERSECTION, UNION & DIFFERENCE
• LIST, STACK & QUEUE IMPLEMENTATIONS ARE EXAMPLES OF ABSTRACT DATATYPES.

LIST

ADT LIST is a datatype that is a sequence of 0 or more elements.



OPERATIONS ON ADT LIST

- ☆ Initialize
- ☆ Insert (InsertFirst, insertLast, insertSorted, insertPosition)
- ☆ Delete
- ☆ isMember

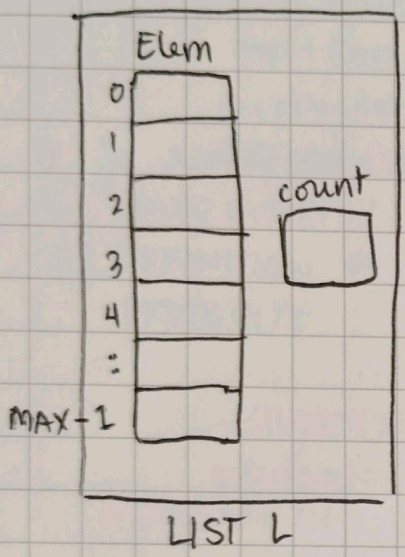
ADT LIST REPRESENTATIONS / IMPLEMENTATIONS

1. Array implementation ⟶ 4 versions.
2. Linked List
3. Cursor - Based

4 VERSIONS OF ARRAY IMPLEMENTATION

Version 1: LIST is a structure with an array & variable count (a variable that keeps track of the number of elements in the array list)

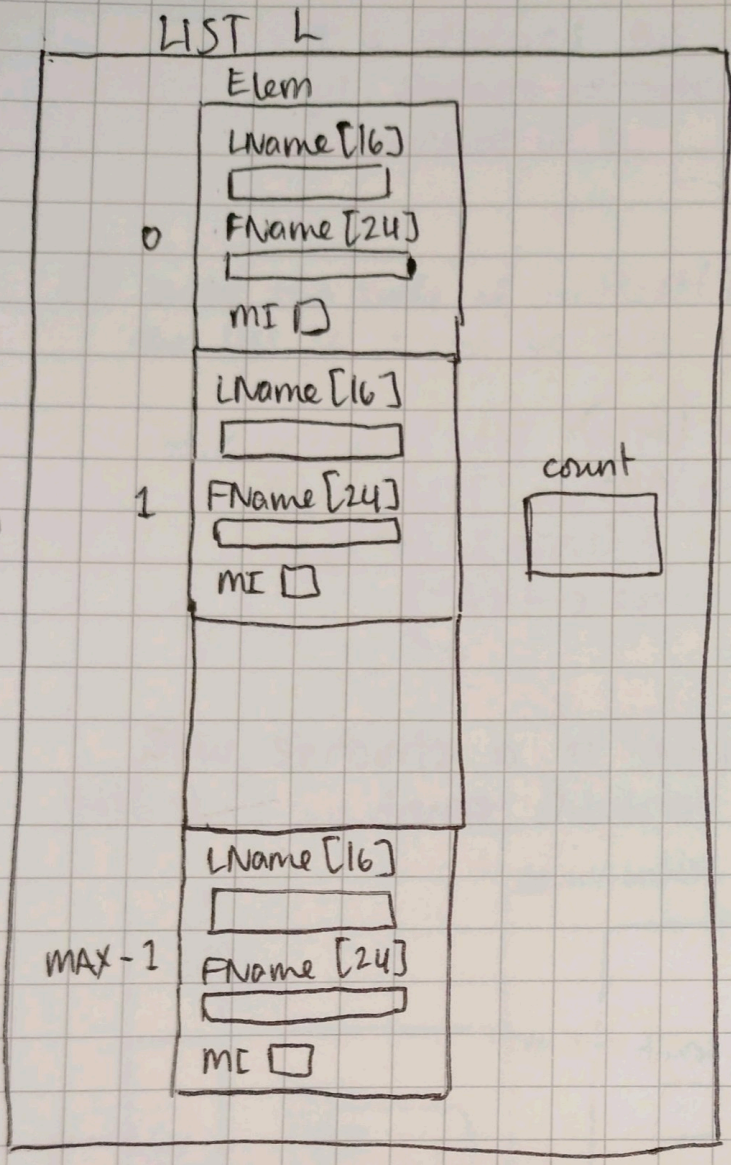
Scenario: The value stored at each index of the array is a scalar value.



① Write the appropriate definition of datatype LIST

```
#define MAX 5
typedef struct {
        int Elem [MAX];
        int count;
} LIST;
```

Scenario: The value stored at each index of the array is an aggregate value.



① Write the appropriate definition of datatype LIST

```
#define MAX 5
typedef struct {
    char LName [16];
    char FName [24];
    char MI;
} Nametype

typedef struct {
    Nametype Elem [MAX];
    int count;
} LIST;
```

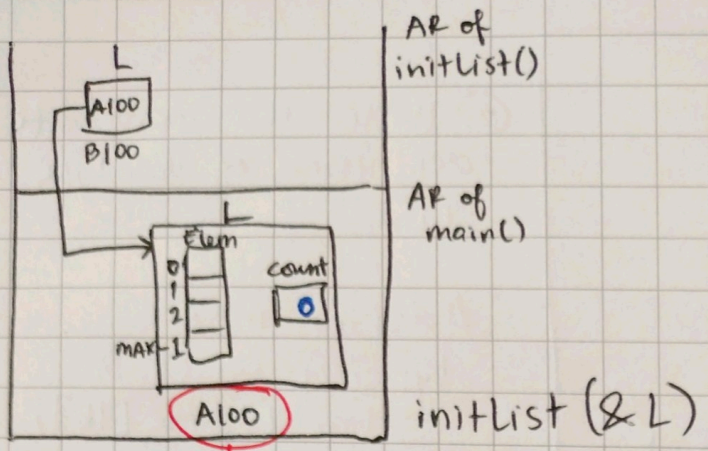
→ datatype Nametype was defined because each value stored inside the array are aggregate datatypes (a structure that holds information of student)

② Write the code of the function initList(). The function will make the LIST empty.

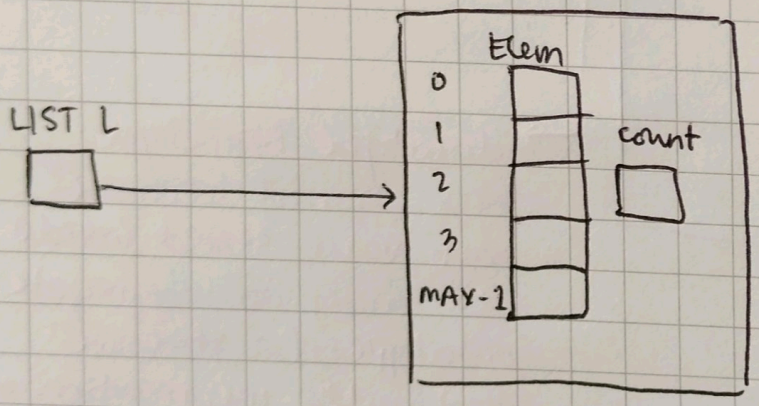
```
void initList (LIST * L) {
    L → count = 0;
}
```

→ setting count = 0 because the array is accessed by using count

initList() SIMULATION



Version 2: LIST is a pointer to a structure with an array & variable count.



① Write the appropriate definition of datatype LIST.

```
# define MAX 5
typedef struct node {
        char Elem [MAX];
        int count;
} *LIST;
```

tag name of the structure

pointer to structure datatype defined here

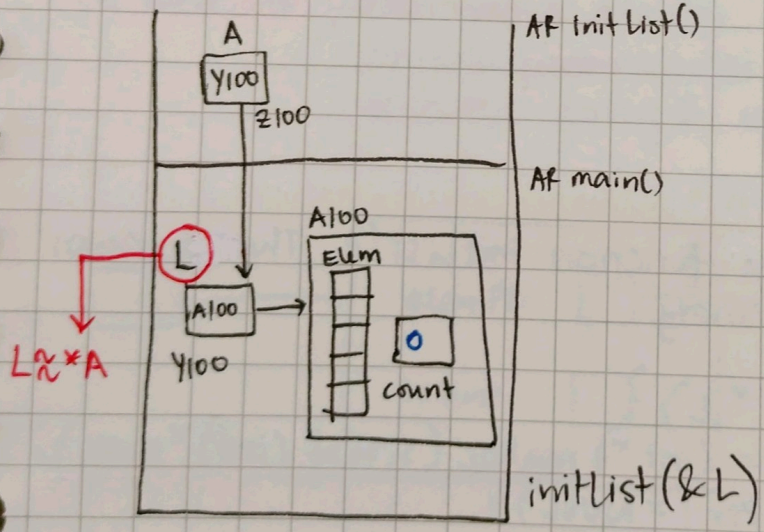
★ Note: Definition of a structure holds no storage until it is declared. But if it is not declared yet, storage allocated to the definition of a structure is 0.

② Write the code of initList(). The function will prepare the list so it can be used for the first time.

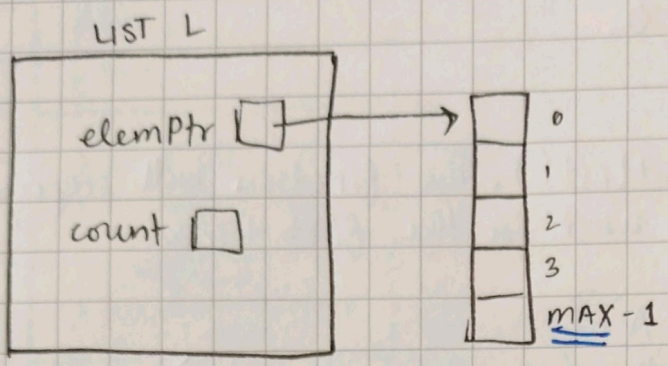
```
void initList ( LIST *A) {
    *A = (LIST) malloc (sizeof (struct node));
    if ((*A) != NULL) {
        (*A) → count = 0;
}
```

L ≈ *A, so the code is similar to version 1

initList() SIMULATION



version 3: LIST is a structure that contains an elemPtr & has a count variable



① Write the appropriate datatype definition of datatype LIST

```
typedef struct {
    int * elemPtr;
    int count;
} LIST;
```

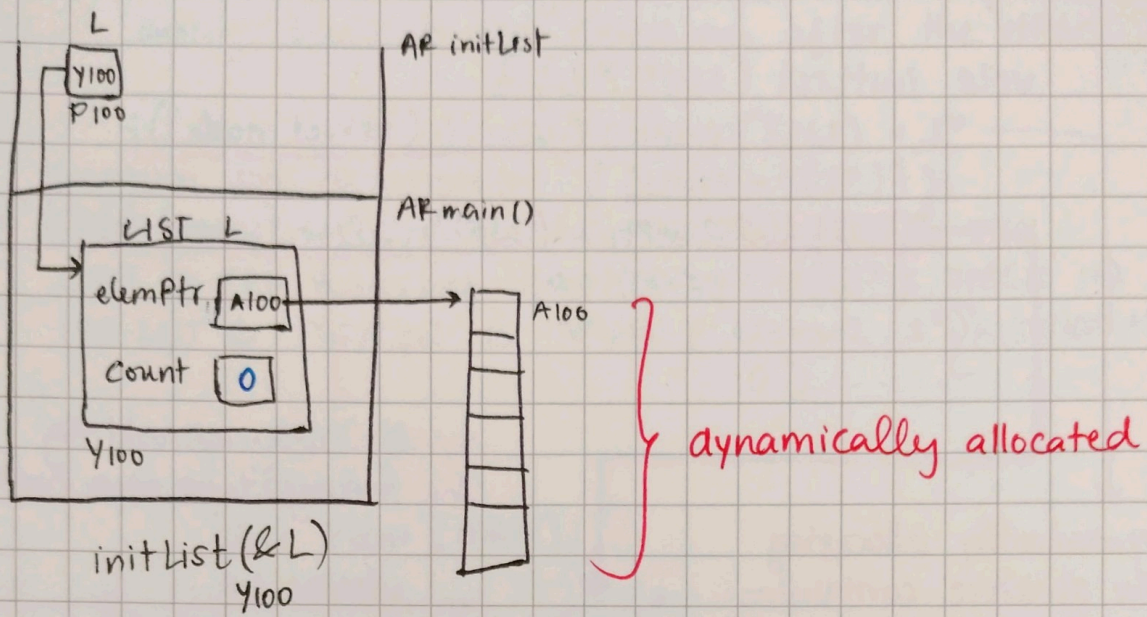
→ pointer to create & access a dynamically allocated array.

② Write the code of the function initList(). The function will make the LIST empty.

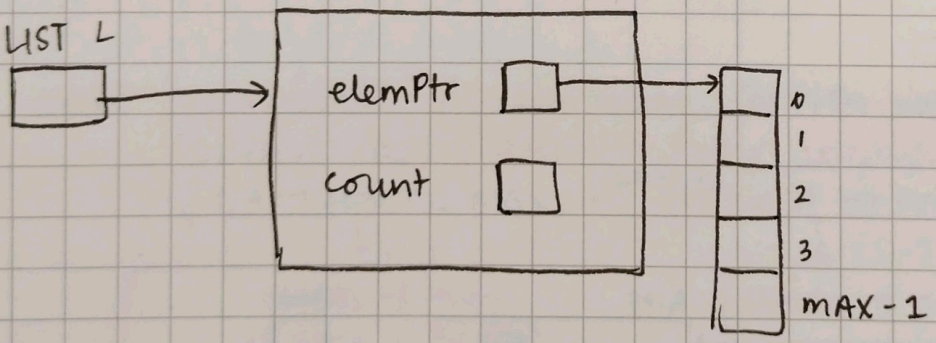
```
void initList(LIST * L) {
    L → elemPtr = (int *) malloc (sizeof (int) * MAX);
    if (L → elemPtr != NULL) {
        L → count = 0;
    }
}
```

→ dynamically allocating the array of integers of MAX size (0 - MAX-1 index)

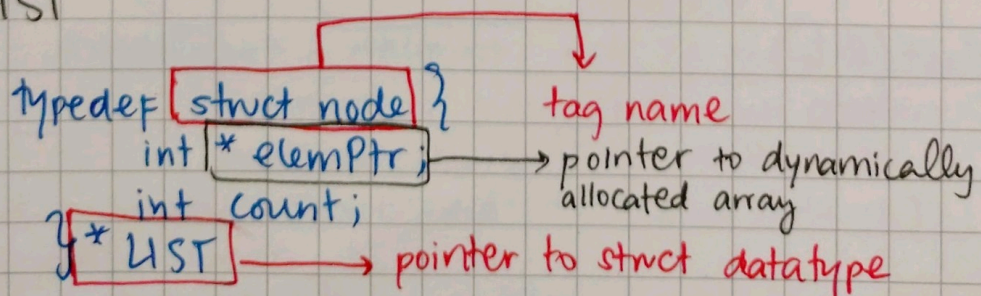
initList() SIMULATION



Version 4: LIST is a pointer to a structure that contains a pointer to a dynamically allocated array & variable count.



① Write the appropriate datatype definition of datatype LIST



② Write the code of the function initList(). The function will make the LIST empty.

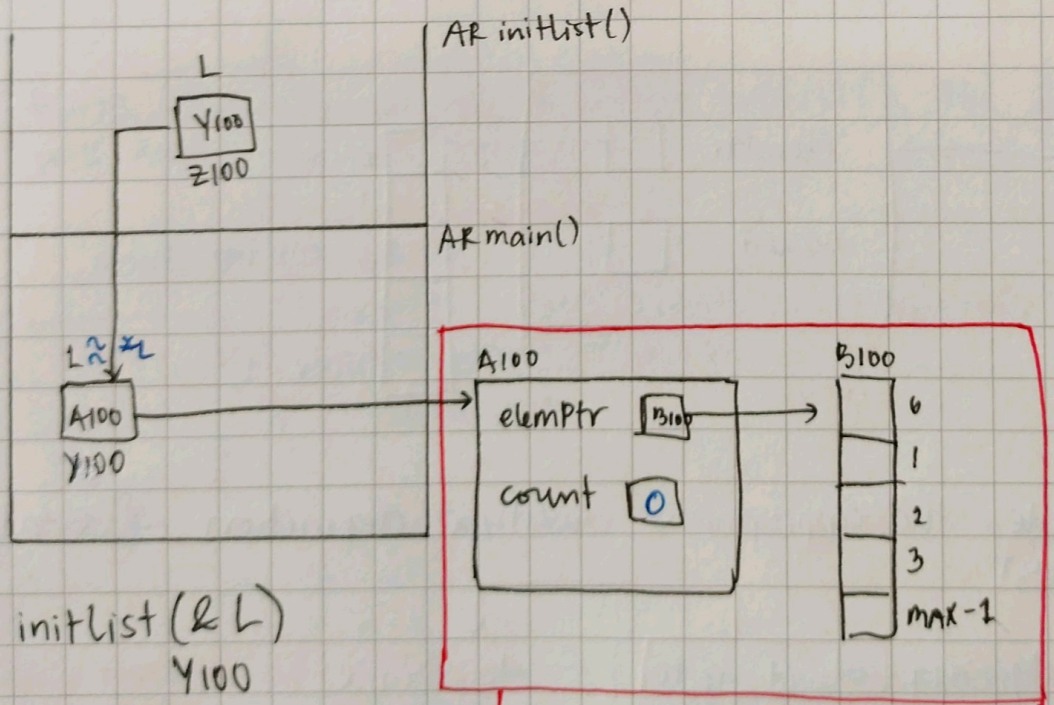
```
void initList (LIST *L) {
    *L = (LIST) malloc (sizeof (struct node));
    if ((*L) != NULL) {
        (*L) → elemPtr = (LIST) malloc (sizeof (int) * MAX);
        (*L) → count = 0;
    }
}
```

dynamically allocating the array data structure

dynamically allocating the structure containing elemPtr & count

"(*L) →" parentheses precedence.

initList() SIMULATION



→ both structures are dynamically allocated, storage in the heap

★ Note: **ARRAY IMPLEMENTATION**

Elements are stored in contiguous cells of the array. Elements are stored one after the other.

Q. PROBLEM SPECIFICATION:
GIVEN THE LIST, ELEMENT & POSITION, WRITE THE CODE OF THE FUNCTION insert() if THERE IS SPACE IN THE LIST & POSITION IS VALID. (ASSUME 1st POSITION IS 0)

```
#define MAX 10
typedef struct {
        char Elem[MAX];
        int count;
} LIST;

typedef int Position;
```

(because I'm not smart bruh)

ANS. In problem solving, come up with subtasks
1. Check if there is space & if position is valid
2. Make room for the new element by shifting
3. Insert new element & update count

```
void insert(LIST *A, char elem, Position pos) {
    Position ndx; // declaring index variable
    if (A -> count < MAX && pos < A -> count + 1) {
```

checks if there is space in the list

checks if the position to be inserted into is a valid one. e.g positions 1, 2, 3, 0 can be inserted into a list with count = 3

```
        for (ndx = A -> count; ndx >= pos; ndx--) {
```

↳ initializes index to the last element & decrements until the position pos is reached

shifting of elements

```
            A -> Elem[ndx] = A -> Elem[ndx - 1];
        }
        A -> Elem[ndx] = elem; // assigning elem to the position
        A -> count++; // incrementing count.
    }
}
```

ASSIGNMENT:

Q. Write the code given the list & the element. The function will delete given element from the list.

```
#define MAX 10

typedef struct {
        char Elem[MAX];
        int count;
}LIST;
```

ANS. Subtasks (for a dummy like me)

1. Traverse the list & find the element
2. Perform shifting of elements
3. Decrement count;

```
void delete(LIST *L, char elem){
    int x, y;
    if (L->count != 0) { // checking if LIST is empty
      for(x=0; x < L->count && L->Elem[x] != elem; x++ {}
```

↳ traversing through the list until elem is found

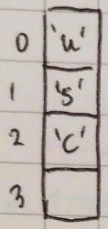
```
      if( x < L->count) { // condition when elem is found
          for(y=x; y < L->count; y++) {
              L->Elem[y] = L->Elem[y+1];
          }
shifting  L->count--; // decrement number
                      of elements because
                      deletion (duh)
    }
}
```

ADT LIST is a data structure that has 0 or more elements.
(Recall)
ADT LIST REPRESENTATION IN MEMORY

★ ARRAY IMPLEMENTATION (4 VERSIONS)
Characteristics :-

① physical order of elements is the same as the logical order

Eg. the illustration below shows the logical representation of the array.



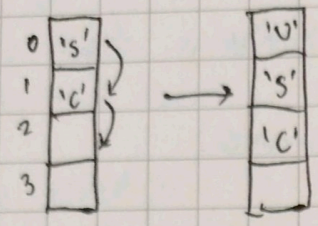
In the computer's memory, the elements are stored in the same manner physically i.e contiguously.

② Provides space that is finite. Space allocated for an array data structure is limited to the max number of elements declared for the array.

③ Operation Running Time = O(n) where n is the number of elements.
Because elements are stored in contiguous cells of the array...

↳ INSERTION
Elements need to be shifted down in order to make room for the inserted element.

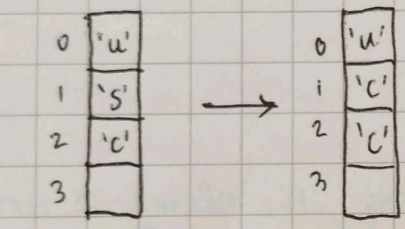
Eg. Insert 'u' in index 0.



↳ DELETION

Elements need to be shifted up in order to overwrite the element that needs to be deleted.

E.g delete 's'



count = 3 count = 2

What happened here?

Remember → count is used to access the array. When the array is accessed, it will only display 'u' and 'c'

↳ TRAVERSAL

Begins from low index (usually 0) to high index OR high index to low index.

(Side) Note: Array is passed to functions with its size included as a parameter because that is how we know where the array ends.

String → array of characters. No need to pass the size because it ends with '\0' indicator.