NO.

DATE 01/30/2023

* Abstract Data Types - also known as ADT
  - A mathematical model together w/ a set of operations defined on the model

NOTE
  - OOP languages are able to do abstraction, C is different.

- ADT List → A collection of elements
  → Sequence of 0 or more elements
  - Scalar (int, char, float)
  - Aggregate (structures, Arrays)

* Operations Used:
  a.) initialize();
  b.) insert();
    - can also be insertFirst();, insertLast();, insertSorted();, insertPosition();
  c.) delete();
  d.) isMember();

A.D.T. List Representation / Implementations
1.) Array Implementation
2.) Linked List Implementation
3.) Cursor-based Implementation

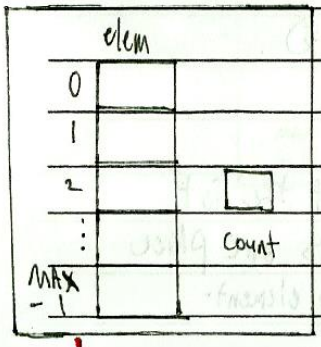
NO.

DATE 01/30/2023

1) Array Implementation

* Version 1A → using Scalar element

LIST L



→ Allocated 44 bytes (MAX * elem + count)

1.) Write the def. of datatype LIST

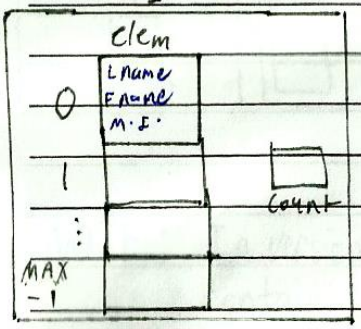
→
```
# define MAX 10
typedef struct {
int elem[MAX];
int count;
} LIST;
```

2.) Write the code for func. InitList(). The function will make the LIST Empty.

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

* Version 1B → using Aggregate

LIST L



1.) write the definition:

→
```
# define MAX 10
typedef struct {
char lname[16];
char fname[16];
char mI;
} Nametype;

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

NOTE: In version 1: List is a structure containing an array and a variable count.

Sterling

NO.

DATE 01/30/2023

SUPPORT PAGE HERE

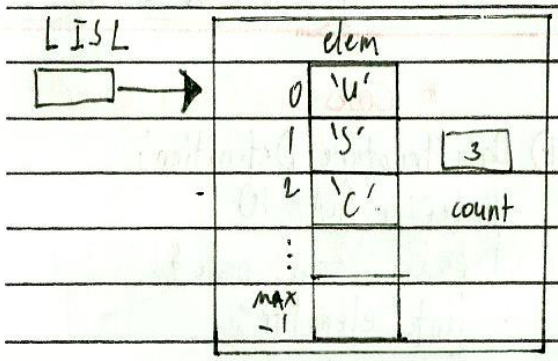
- Advantages:
  - ★ New elements can be appended readily to the tail of the list.
  - ★ List is easily traversed (i.e. using the index)

- Disadvantages
  - ★ Inserting an element into the middle of the list requires shifting all the following elements one place in the array to make room for the new element.
  - ★ Deleting any element except the element at last position also requires shifting of elements to close the gap.

NO.

DATE 01/30/2023

* Version 2 → Scalar



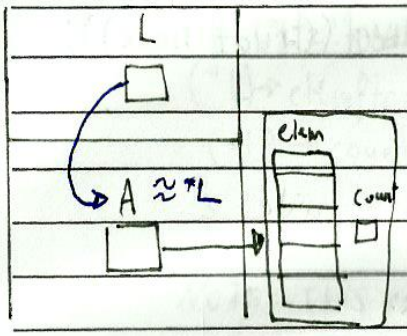
1.) Write an appropriate def. of datatype LIST

→
```
# define MAX 10
typedef struct node {
  char elem [MAX];
  int count;
} *LIST;
```

2.) Write the code of initList():

The function will prepare the list so that it can be used for the first time.

Drawing:



Code:

```
LIST A; //in main()

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

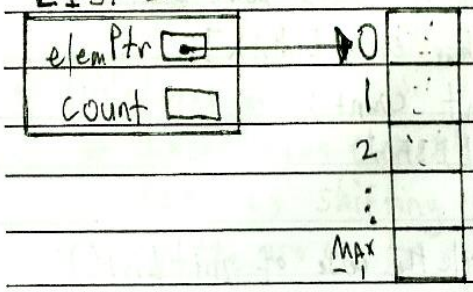
NOTE: In version 2, List is a pointer to a structure containing an array and a variable count.

NO.

DATE 01/30/2023

- Version 3- List is a structure containing variable count and a pointer the first element of a dynamically allocated array.

* Drawing/ExecStack:



* Codes!

1.) Data Structure Definition:

```
#define MAX 10
typedef struct node {
    int * elemPtr;
    int count;
} LIST;
```

2.) function Definition of initList().

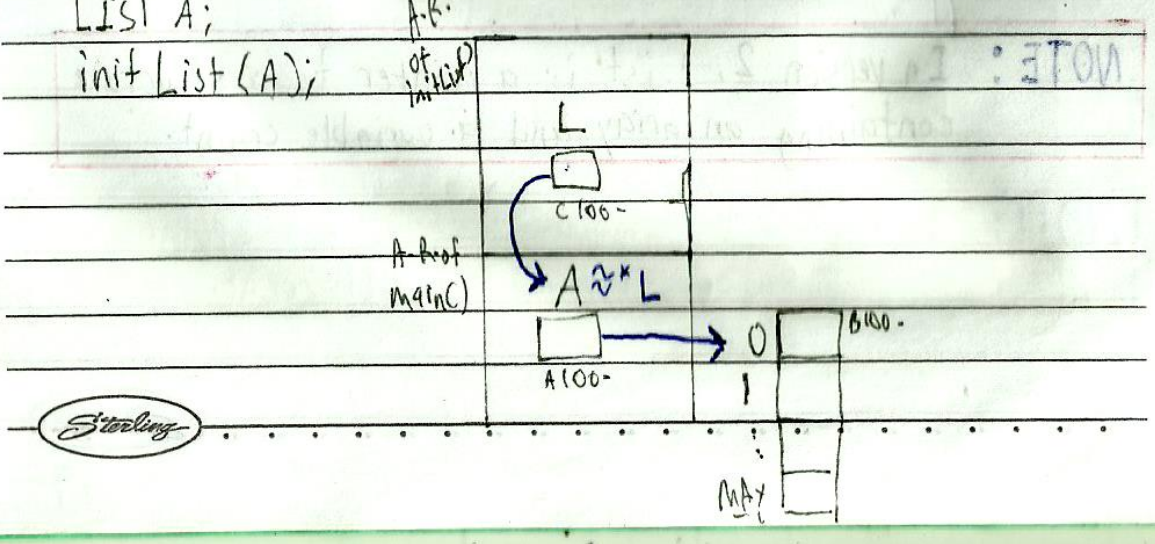
```
void initList (LIST *L) {
    L->elemPtr = (LIST) malloc (sizeof (struct node));
    L->count = 0;
}
```

- Execution Stack & func. call in main as a guide

//func. call in main():

```
LIST A;
initList (A);
```

//Execstack:

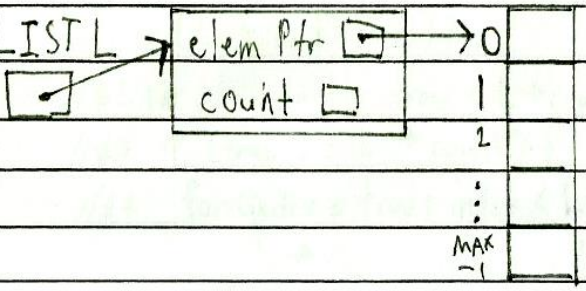


NO.

DATE 01/30/2023

•Version 4- List is a pointer to a structure containing variable count and a pointer the first element of a dynamically allocated array.

*Drawing/Exec Stack:



*Codes:

1.) Data Struct Definition:

```
#define MAX 10
typedef struct node {
    int *elemPtr;
    int count;
}*LIST;
```

2.) Func- Definition of initList().

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

NOTE: This version is S.T.C.

★ Problem Specification: Given the List, element, & position, write the code of the function insert() if there is space and position is valid.

NOTE: 1.) Check if there is space
2.) Make room for the new element (to shift)
3.) insert the new element & update count.

→

Sterling

NO.

Writer's note:
WHAT IS HAPPENNING

DATE 01/30/2023

• NOTE: This code is S.T.C., this is using version 1

Drawing & simulation:

• How it going to work:

elem [3] = elem [2]
elem [2] = elem [1]
elem [1] = elem [0]

• Data Struct Def (for ref):

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

• Code:

```
void insert (LIST *L, char data, Position pos) {
  Position index;
  if (L != NULL && L -> count < MAX) {
    for(index = pos-1; index < pos ; index--) {
      L -> elem[index+1] = L -> elem[index-1];
    }
    L -> elem[pos] = data;
    L -> count++;
  }
```

Stabilo

NO.

DATE 10/30/2023

Problem specification: Given the list, & element, write the code that will delete the element from the list & update count.
(Note: this is also using version 1)

My code:

```
Void delete (LIST L, char data) {
int index, trav;
//#1 if (L→count != 0) {
//#2 for(trav = 0; trav < L→count && L→elem [trav] != data; trav++) {}
//#3 if (trav != L→count) {
//#4 for(index = trav; index < L→count -1; index++) {
        L→elem [index] = L→elem [index + 1];
    }
} else {
    printf ("element not found");
}
//#5 L→count--;
}
```

Simulation:

1.) Check if list is empty e.g. count = 4;
2.) Traverse the list, look for element
3.) if trav has not reached count (or end) proceed, else no elem found
4.) shift???
5.) update count

Sterling

NO.

DATE 2/1/2023

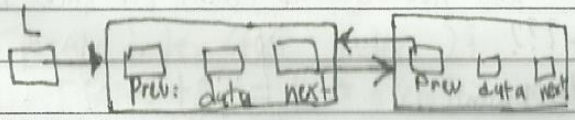
Recall:

ADT List - A sequence of 0 or more elements
- A representation in memory

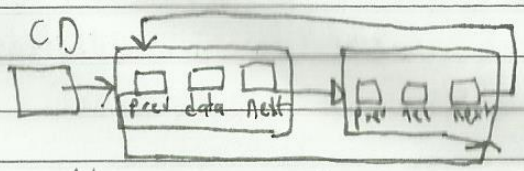
1.) Array Implementation
↳ size tells us how far we can access
NOTE: for string datatype, no need to pass the size as it knows where to end
↳ 4 versions of Array Implementation
↳ Characteristics:
1.) Has physical order of the elements
2.) Finite space (Limited space)
3.) elements are stored in contiguous cells of the array
↳ insertion: shifting downwards
↳ deletion: shifting upwards
• Has a running time of O(n)

★ 2) LINKED LIST IMPLEMENTATION ★

• 2 Types of Linked Lists •
↳ Simply Linked List - one-directional



↳ Doubly Linked List - circular-based doubly linked



NOTE: Logical order may not be the same as the physical order

NO.

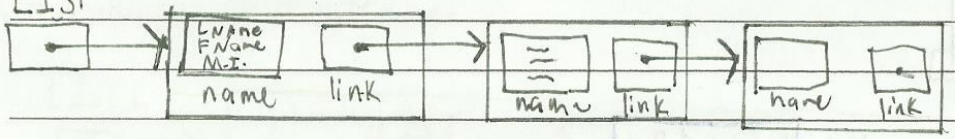
DATE 2/1/2023

★ When to use either of the implementations/representations ★

↳ Array Implementation - if the # of elements are known

↳ Linked List Implementation - for insertion & deletion, etc.

• Illustration:



• Data Structure Definition:

data type not needed — data type needed

```
typedef struct {
char lname[16];
char fname[24];
char MI;
} Person

typedef struct node {
Person name;
struct node* link;
} *LIST, nodetype
```

A self-reference structure

* ADDITIONAL NOTES *

• struct node ≈ nodetype
nodetype* ≈ *LIST
struct node ≈ LIST

• Internet Versions Comparisons:

↳ internet ver:

```
struct node {
struct Name name;
struct node* link;
};
```

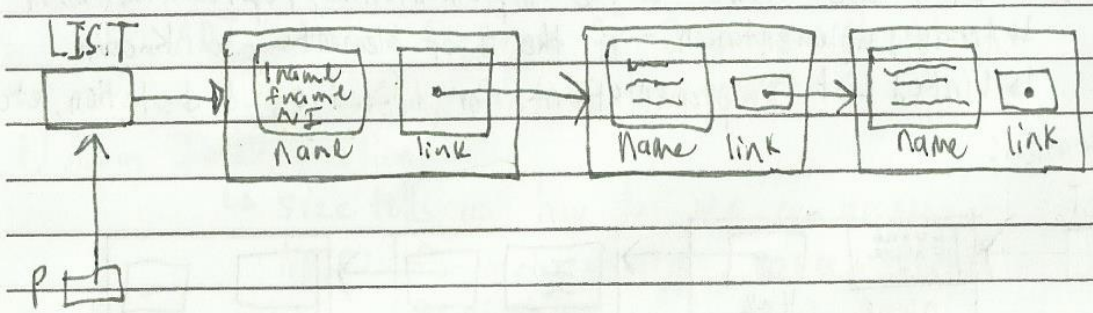
↳ Pass the list by copy: void displayList(LIST A);
internet version: void displayList(struct node* A);

↳ Pass the list by address:
• void insertFirst(LIST *A, Nametype N);
• net version: void insertFirst(struct node* A, struct Name N);

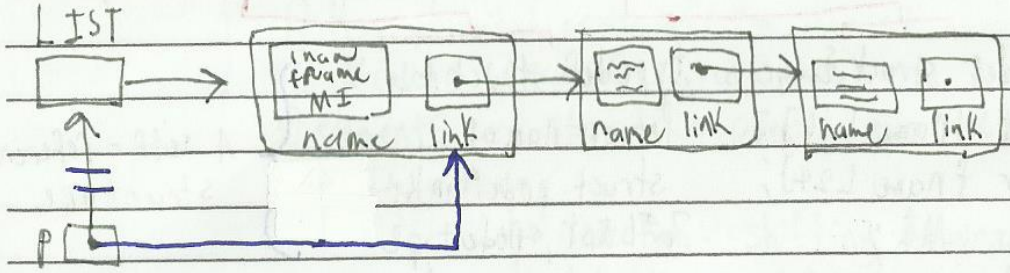
NO.

DATE 2/1/2023

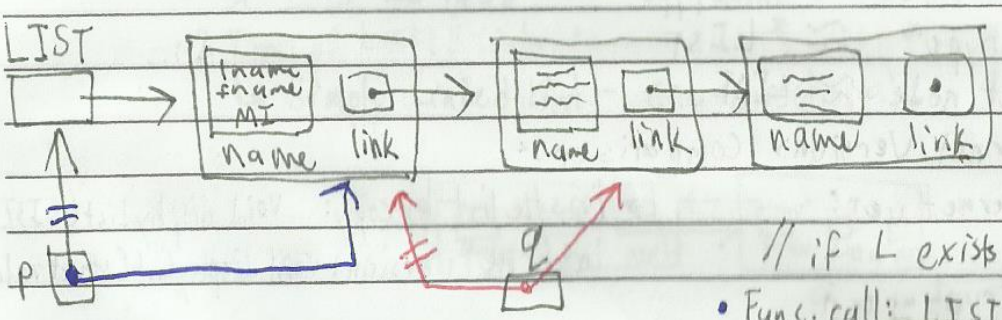
writers note: I am having a bad sugar rush send help



- func. call: `LIST *P;`




```
LIST *P; //does *P exist? (no)
P = &L;  // when will *P exist? when P has a value.
```



// if L exists (wdym past me?)

- Func. call: `LIST *P, q;`

`q = L; //what is happening`

- traversal of q : `q = q -> link;`
- Let p point to the next pointer : `p = &(*P) -> link;`

Sterling

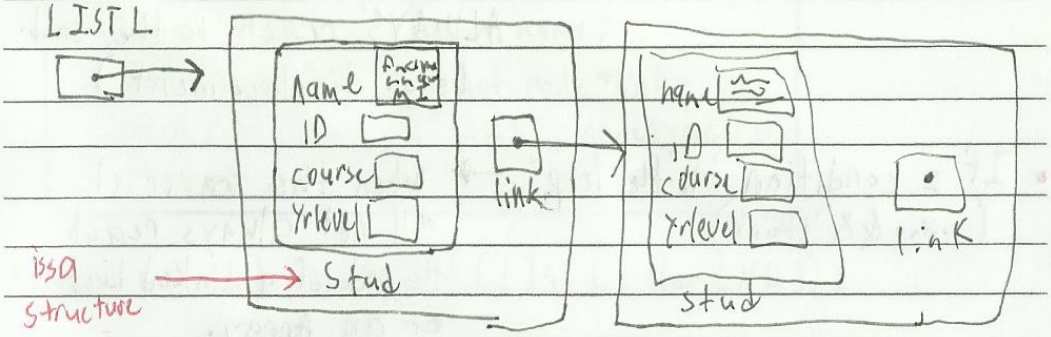
NO.

DATE 2/1/2023

• Write the statement that will access M.I.

1.) via L	2.) via q	3.) via p
↳ L → name . MI;	↳ q → name . MI;	↳ (*p) → name . MI;

★ Another example of accessing:



• How to access:

1.) L → stud . name . MI; //to access MI

2.) L → stud . name lname[0]; //to access first name of //lname

★ When is it one or two conditions for traversal? ★

Traversal for:

↳ Arrays: Low-Index (0) to High Index
High Index to Low-Index

↳ Linked Lists: 1st node to last node
↑ (if simply linked list)

Continuation:

Sterling

NO.

DATE 2/1/2023

- If 1 condition in the loop? → For every call, the traversal reaches to the end.

  → For every array passed to the function, the traversal will ALWAYS reach to the end. (e.g. displacement)

- If 2 condition in the loop? (using && i think) → When the traversal will not always reach the end of a Linked List or an Array.

  → The 1st condition is similar to that if just 1 condition.

ADDITIONAL NOTES

★ Review on Linked-List Traversal ★

1.) Pointer-To-Node (PN) Traversal → used when not inserting / deleting

2.) Pointer-To-Pointer-TO-NODE (PPN) → used when inserting and deleting (other than 1st elem/position)

NO.

DATE 2/1/2023

Problem Specification: Write the code of the function that will return the # of students bearing the given last name in the given list.

using Pointer-to-node

for reference:

```
typedef struct{          typedef struct node{
char lname[16];          Person name;
char fname[24];          struct node *link;
char MI;                 }*LIST, nodetype;
} Person;
```

- My code:

```
int numOfStud(LIST L, char LN[]){
    int count=0;
    LIST trav;
    for(trav=L; trav->name.lname != LN; trav=trav->link){
        if(trav->name.lname == LN){
            count++;
        }
    }
    return count;
}
```

OR

```
int numOfStud(LIST L, char LN[]){
    int count=0;
    for(; L->name.lname != LN && L != NULL; L=L->link){
        if(strcmp(L->name.lname, LN) == 0){
            count++;
        }
    } return count;
}
```