NO.

DATE 01/18/2023

T.I.L:

- Reviewed topics related in PROG01 & PROG02
  - ↳ Variable - A location in memory

Attributes & Variables

can consist of:

a. name
b. value
c. datatype → This needs to be declared
d. address
e. Scope & Lifetime → It exists but not accessible, the text of the program
   ↳ Only during program execution (i.e., start of a program, within parameters of a function, etc.)

NOTE: The programer has control over all of these EXCEPT address

* Additional Learnings / Understanding:

1.) Linked List + Array Based = CURSOR BASED
2.) Drawing helps in Visualizing Codes

Sterling

NO.

DATE 01/18/2023

Drawing Execution Stacks

Note: A.R. = Activation Record

Exercise 1

Problem Specification: The function will exchange the values of a given integer

Function Code:

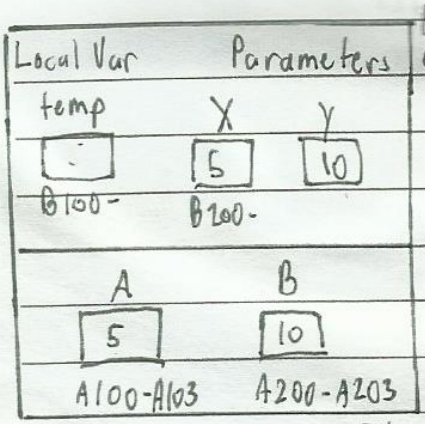
```
void exchange(int x, int y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}
```

Function Call in main():

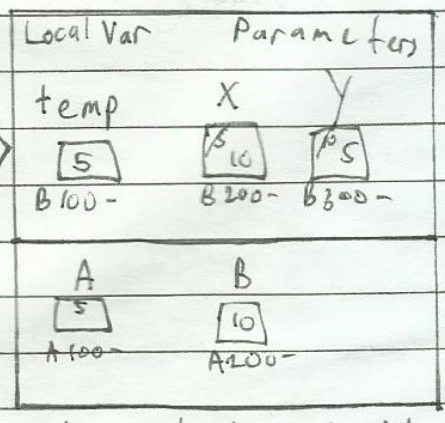
```
int A = 5;
int B = 10;
exchange(A, B);
```

Using an execution stack, show that the function will or will not work based on the function specification & Explain.

(1)



(2) (upon simulation)



Expl: This function does not work due to it only exchanging the values. There is no access for main() for the variables. No changes in address.

NO.

DATE 1/18/2023

Exercise 2

- Since the previous function does not work due to main() having no access to the variables, the solution is to use Pass by address.

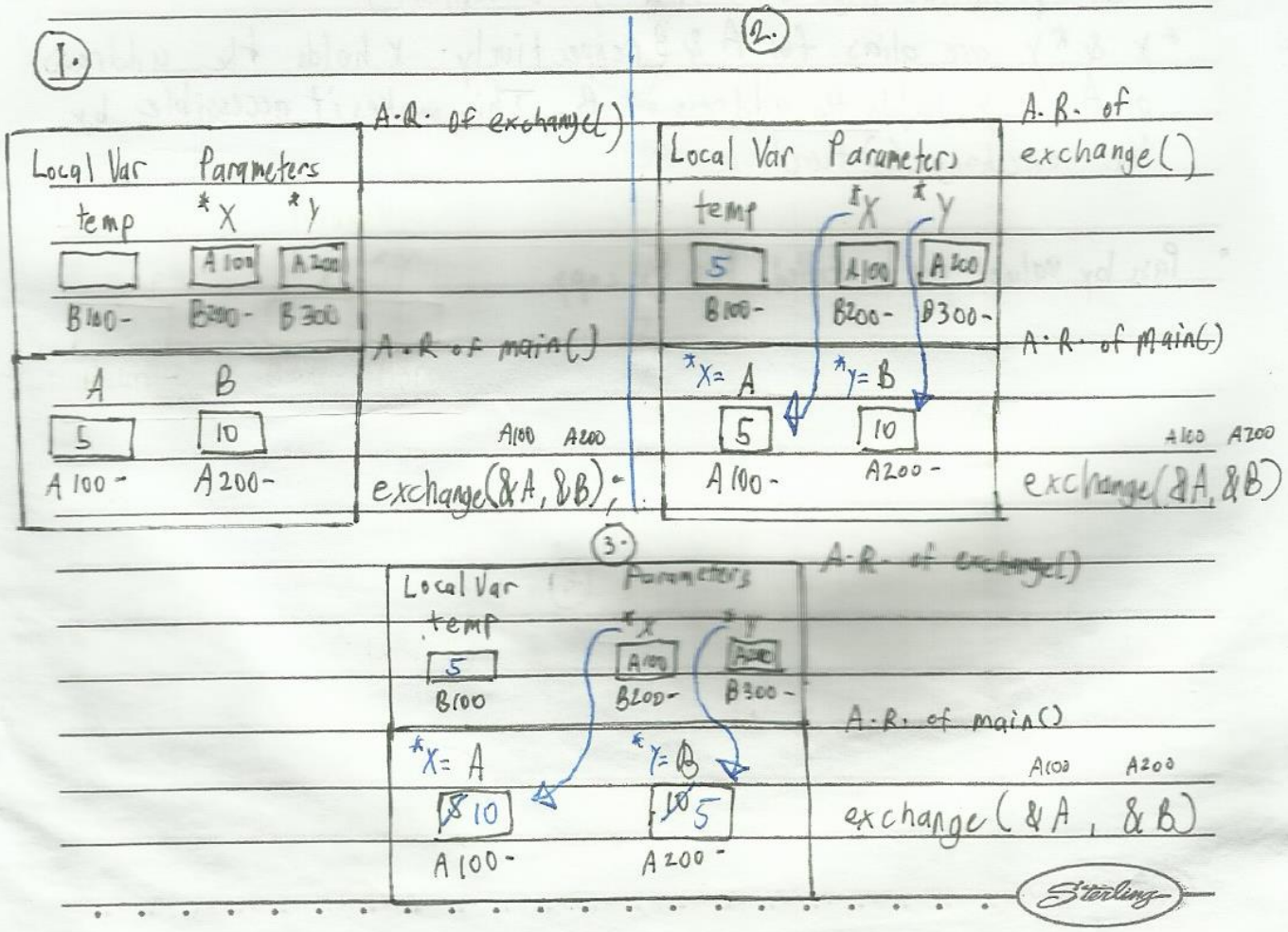
SOLUTION:

a.) Function Header → void exchange(int *x, int *y)

b.) Write an appropriate function call. Declare & initialize the variables used in the call, before the call

```
int A = 5;
int B = 10;
exchange (&A, &B);
```

c.) Draw the execution stack that illustrates the func. call assuming that the call is in main()



Sterling

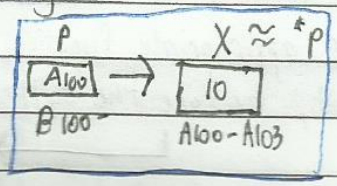
NO.

DATE 01/18/2023

* SUPPORTING STUFF *

- int x = 10; *Write a C statement that:
  int *p; 1.) will let p point to x
  ✓Code: ✓Drawing:

  p = &x;



- ⊛ = to dereference the variable, get the value of the address its pointing to.

- (check previous page w/ *x & *y for context).
  *x & *y are alias for A & B respectively. x holds the address of A & y holds the address of B. This makes it accessible by the exchange() function.

- Pass by value can be called Pass by copy

Sterling

NO.

DATE 01/18/2023

4.) Write the code of the function (w/ func. header & body)

```
void exchange (int *x, int *y)
{
   int temp;

   temp = *x;
   *x = *y;
   *y = temp;
}
```

NO.

DATE

* SUPPORT PAGE *

NOTES:

- Memory that is dynamically allocated is allocated to the heap portion of the computer memory.

NO.

DATE 01/18/2023

* Exercise 3 *

Data Structure Definition

```
typedef struct node {
        int data;
        struct node *link;
} *LIST;
```

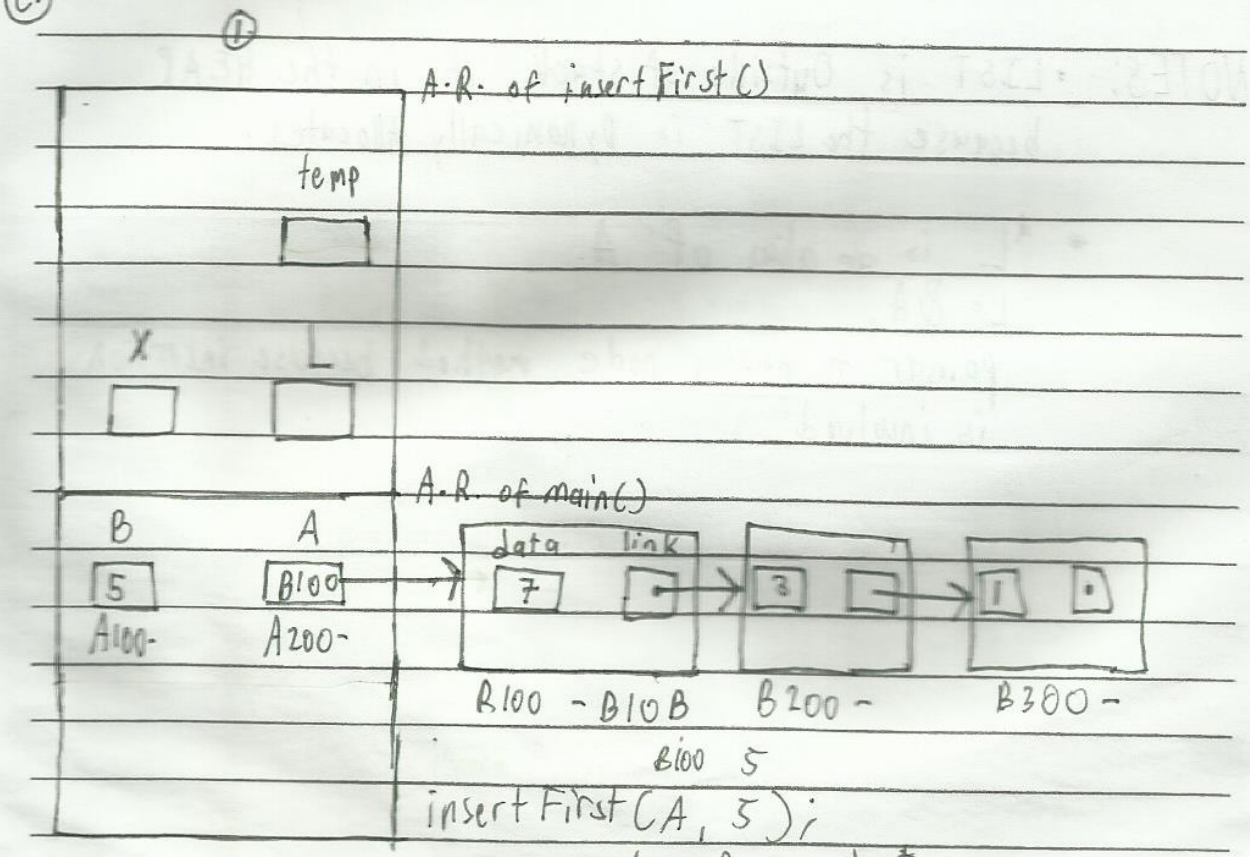
Problem specification: The function will insert an element at the 1st position of the given list.

a.) write the func. header
b.) Write the func. call
c.) Draw the exec stack illustrating the func. call
d.) Write the code of the func.

(a.) `void insertFirst (LIST L, int x)`

(b.)
```
LIST A = NULL; //assume that LIST A is populated w/ 5 elems.
int B = 5;
insertFirst (A, B); // assume that the func. call is in main()
```

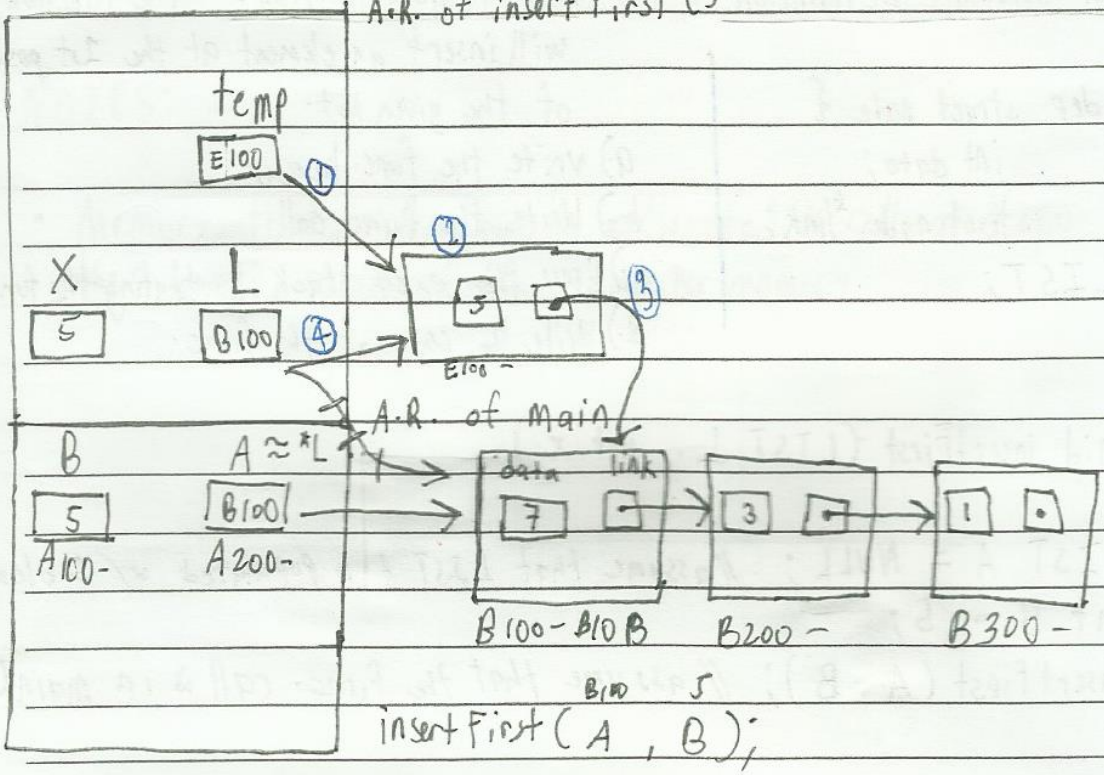
(c.)



* check nextpg for cont. *

NO.

DATE 01/18/2023 *Continuation of prev pg.*



NOTES:
- LIST is Outside of stack, its in the HEAP because the LIST is Dynamically Allocated.
- *L is an alias of A.
  L = &A.
  pointer to pointer node method because insertion is involved

NO.

DATE 01/18/2023

(d.)

```
void insertFirst (LIST *L, int x) //Func. header
{
    LIST temp; // declaration of temp
    temp = (LIST) malloc (size of (struct node);
                //dynamically allocating a new node

    if (temp != NULL) // check if allocation was successful
    {
        temp→data = x; //assigning data to be inserted
        temp→link = *L; //Linking new node to next one
        *L = temp; // Linking head to new node
    }
}
```

Sterling

NO.

DATE [01/23/2023] *Continuation of Linked List review*

- Steps on making efficient code and/or functions -

1. Make a function header
2. Make the function call
3. Draw the execution stack
4. Write the code of the
5. Simulate the code
   ↳ Make test cases
   [i.e. test case 1: List has 3 elements
   test case 2: List is empty.]

{Review on previous exercise}

Data Structure Definition:

```
typedef struct node {
        int data;          → 4 bytes
struct node *link;         → 8 bytes
3 *List;                   12 bytes w/o padding
                           (16 bytes w/ padding)
```

- Func. header:

```
void insertFirst(List *L, int x)
```

- Func. call:

```
//assumption: List has 3 elems
List A;
int elem = 4;
insertFirst(&A, elem);
```

NO.

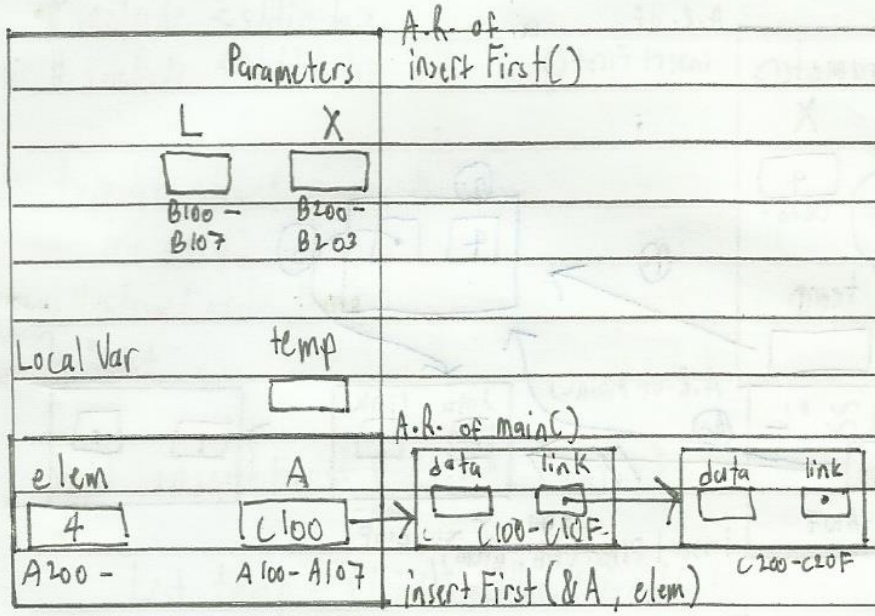
DATE 01/23/2023

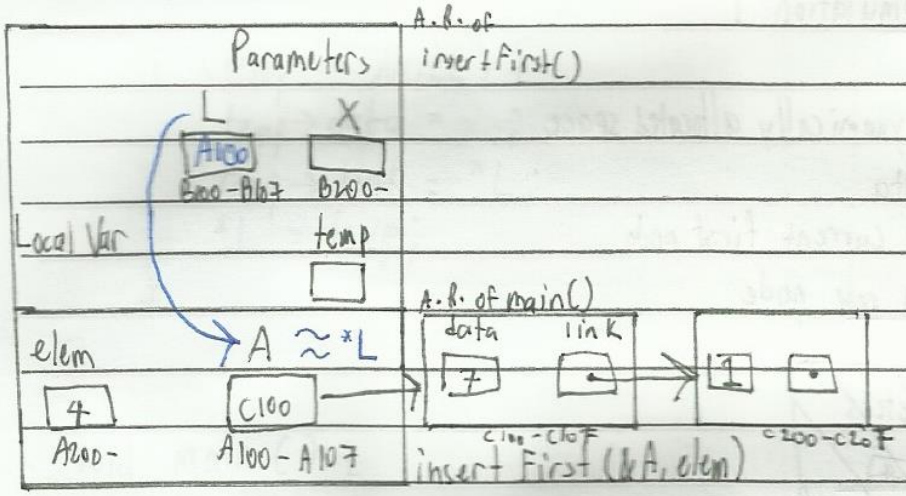
• Execution Stacks •

// Case 1: List has 2 elems

(1.)

Before Func. call & simulation





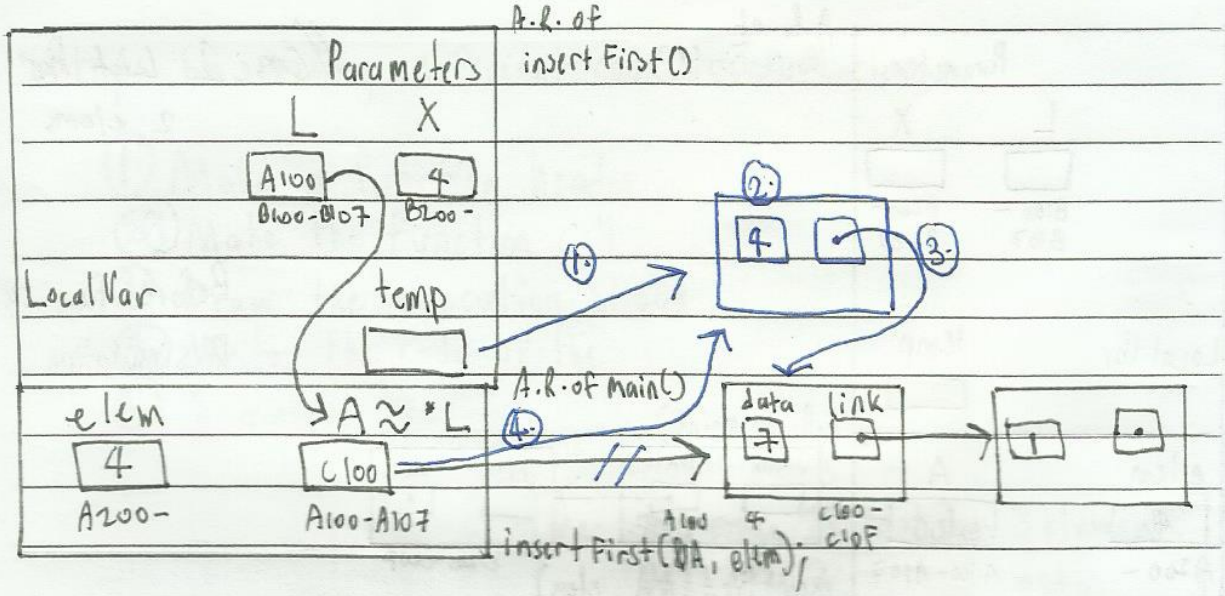
(2.)

After Func. call

Note: continuation at the back

NO.

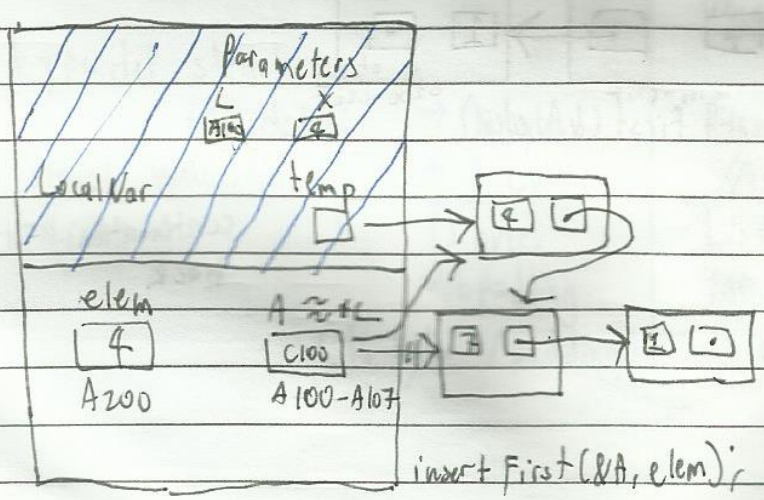
DATE 01/23/2023



3[Function Simulation]

Exp:

1. Point to dynamically allocated space
2. Assign data
3. point to current first node
4. point to new node



4[Function Terminates]

- When the function terminates, the variables within the func- no longer exist.

Sterling

Full Code:

NO. credits to Bessq

DATE 01/23/2023

```
#include <stdio.h>
#include <stdlib.h>

typedef struct node {
    int data;
    struct node *link;
} *List;

void insertFirst ( List *L, int x)
{
    List temp = (List) malloc (sizeof(struct node));

    if (temp != NULL) {
        temp->data = x;
        temp->link = *L;
        *L = temp;
    }
}

void main () {
    List A;
    int elem = 4;
    insertFirst (&A, x);
}
```

NO.

DATE 01/23/2023

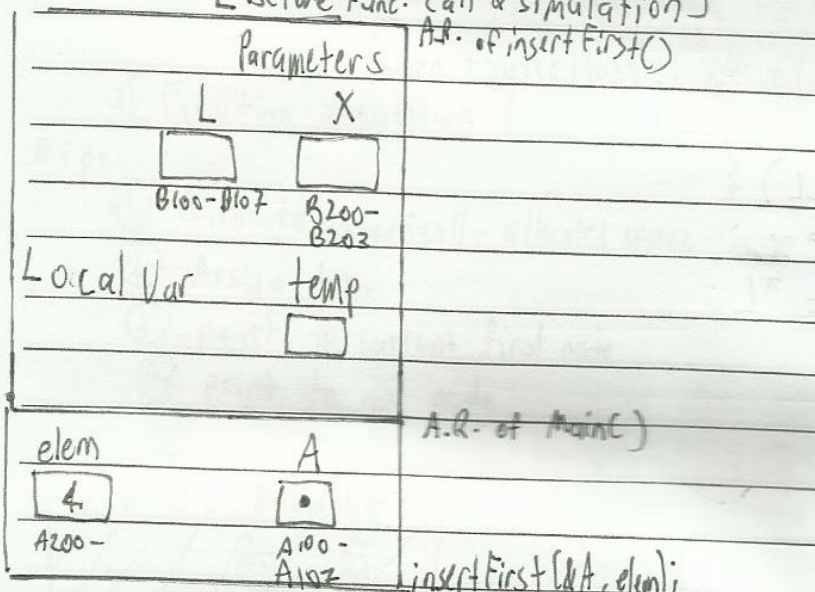
// Test case 2: if List is empty

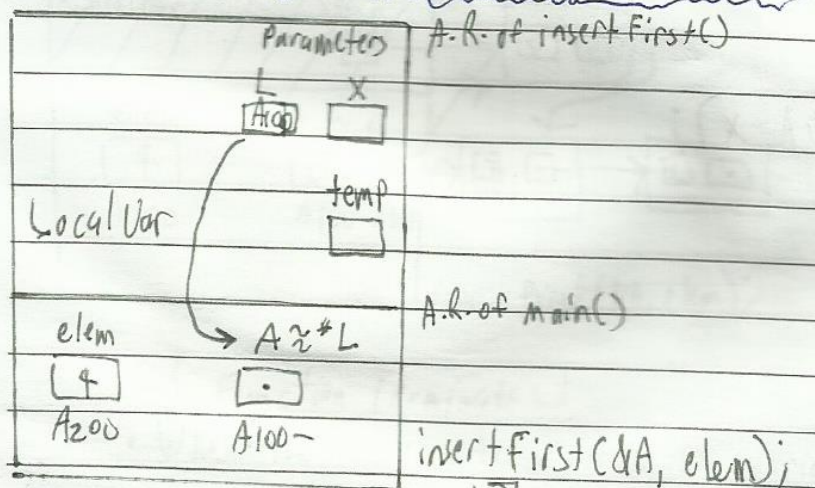
(a) Func. header: void insertFirst(List *L, int x)

(b) Func. call: List A = NULL;
int elem = 4;
insertFirst(&A, elem);

(c) Execution Stack

• [Before Func. call & simulation]





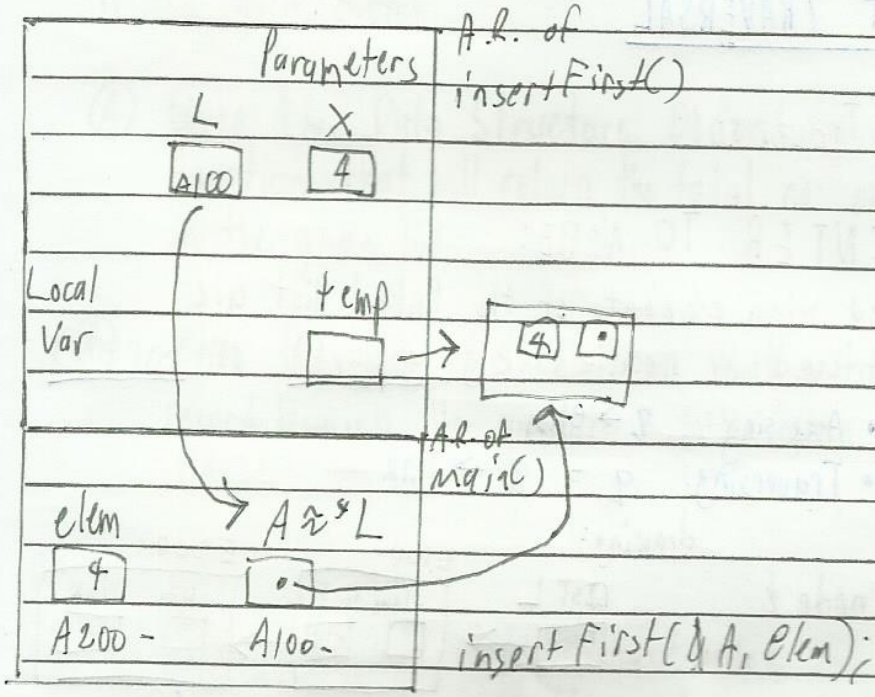
• [Func. call]

Sterling

continu →

NO.

DATE 01/23/2023



[During simulation]

NO.

DATE 01/23/2023

LINKED LIST TRAVERSAL

2 types of List Traversal:

(a) POINTER TO NODE:

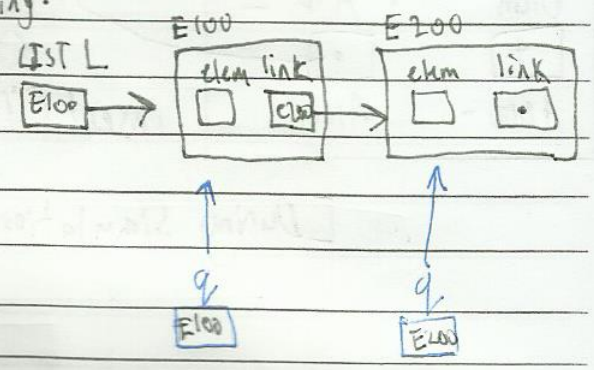
↳ used when elements of the linked list are retrieved or modified e.g. display() or sort()

- Accessing: q → elem
- Traversing: q = q → link

definition:

```
typedef struct node {
    int elem;
    struct node *link;
} *LIST;
```

drawing:



q = q → link

(b) POINTER TO POINTER TO NODE

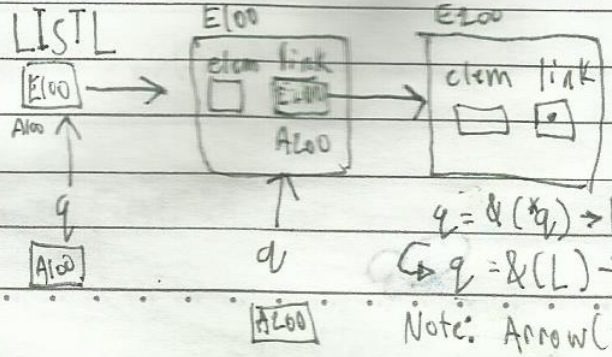
↳ used when inserting or deleting
ie. when a change is expecting in the original list

- Accessing: (*q) → elem
- Traversal: q = &(*q) → link

def:

```
typedef struct node {
    int elem;
    struct node *link;
} *LIST;
```

drawing:



q = &(*q) → link
↳ q = &(L) → link

Note: Arrow (→) takes precedence over &

NO.

DATE 01/23/2023

Problem Specification:

(A.) Given the Data Structure Definition, write the code of the function that will return the total no. of 1st year students in the given list.

(B.) Write the code of the function that will return TRUE if the record bearing the given ID is in the given list, otherwise return FALSE.

Data Struct Def:

```
typedef struct {
    int ID;
    char name[30];
    chore course[8];
    int yearlevel;
} studtype;
```

```
typedef struct node{
    studtype stud;
    struct node *link;
} *List;

typedef enum {TRUE, FALSE} boolean;
```

(A.)

1.) Func. header: `int totalFirstYrs(List L)`

pass by copy only cuz no change is expected in the original list, only reading the data in the list

2.) Func. call:
```
List A;
totalFirstYrs(A);
```

cont. →

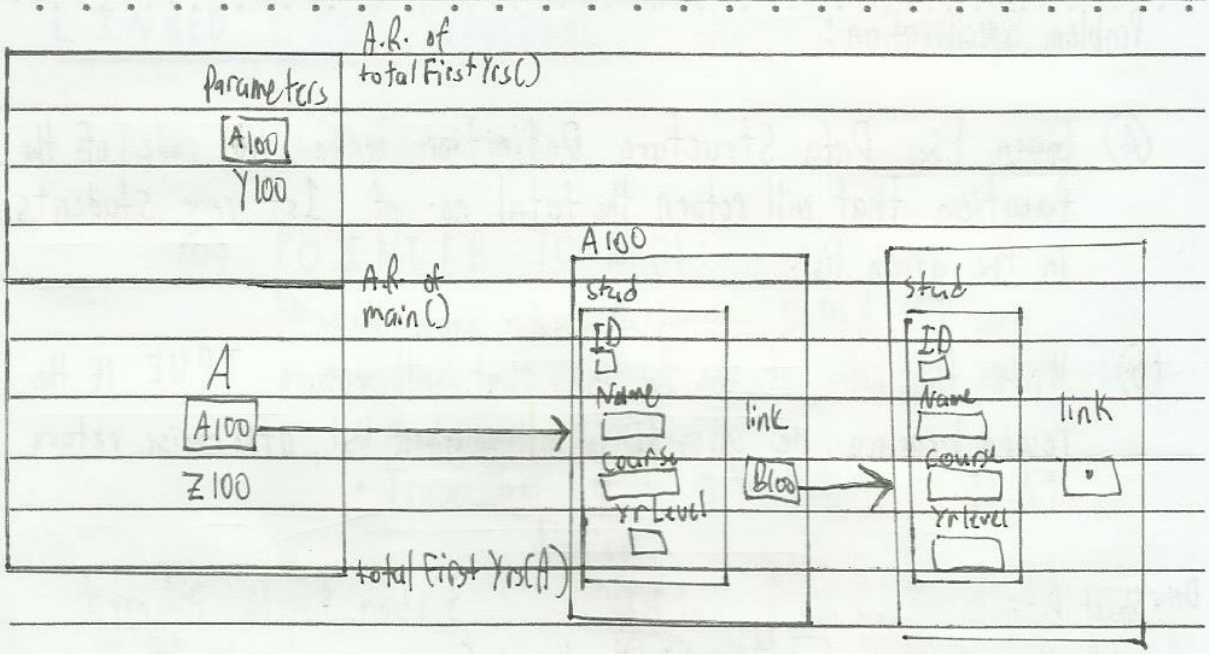
[redacted]

Sterling

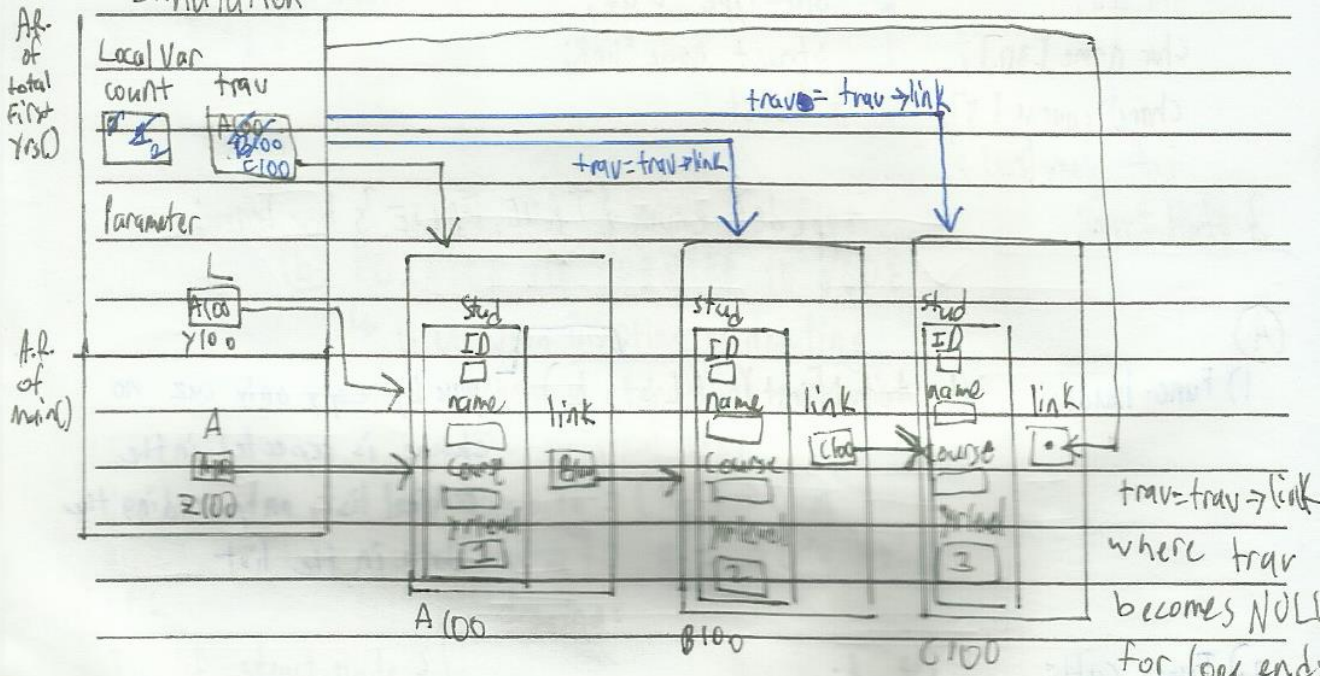
NO.

DATE 01/23/2023

3) Execution Stack



Simulation:



NO.

DATE 01/25/2023

4) Code of the Function

```
int totalFirstYrs (List L){
  int count = 0;
  List trav;

  for (trav = L; trav != NULL; trav = trav->link) {
    if (trav->stud.yearlevel == 1) {
      count ++;
    }
  return count;
}
```

traversal

Assignment: "Using the same Data Struct Definition, Write the code of the function that will return TRUE if the record bearing ID is in the given list; otherwise, return FALSE;

1.) Func. header: boolean checkID (List L, int elem)

2.) Func. call: List A;
int studID;
checkID (A, studID);

3.) Execution Stack:

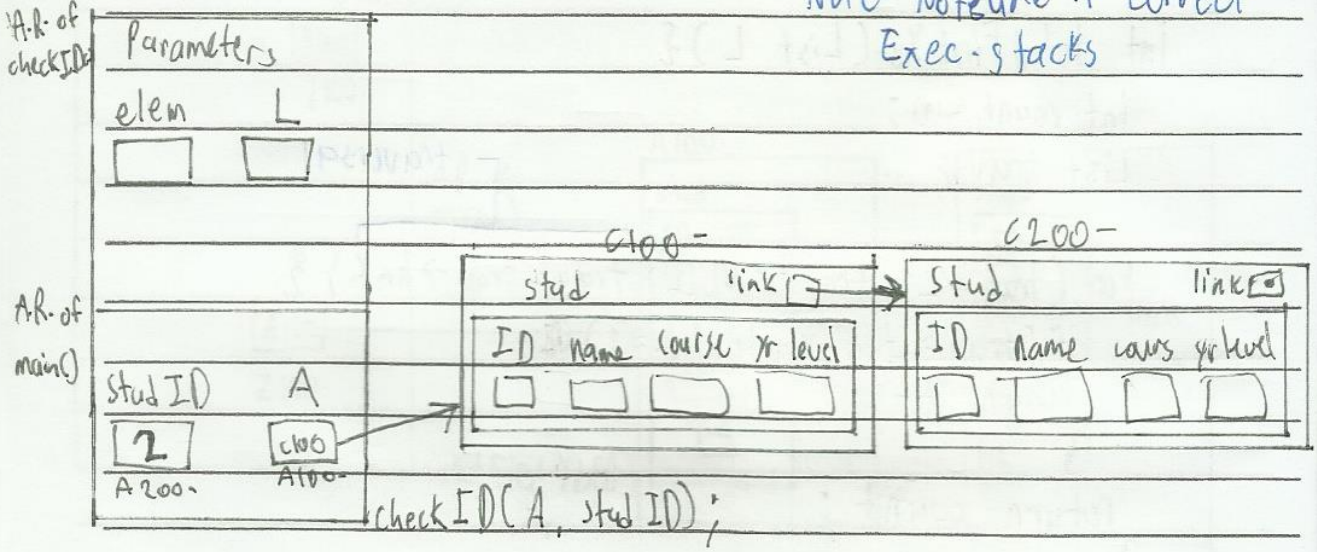
Check next Page

Sterling

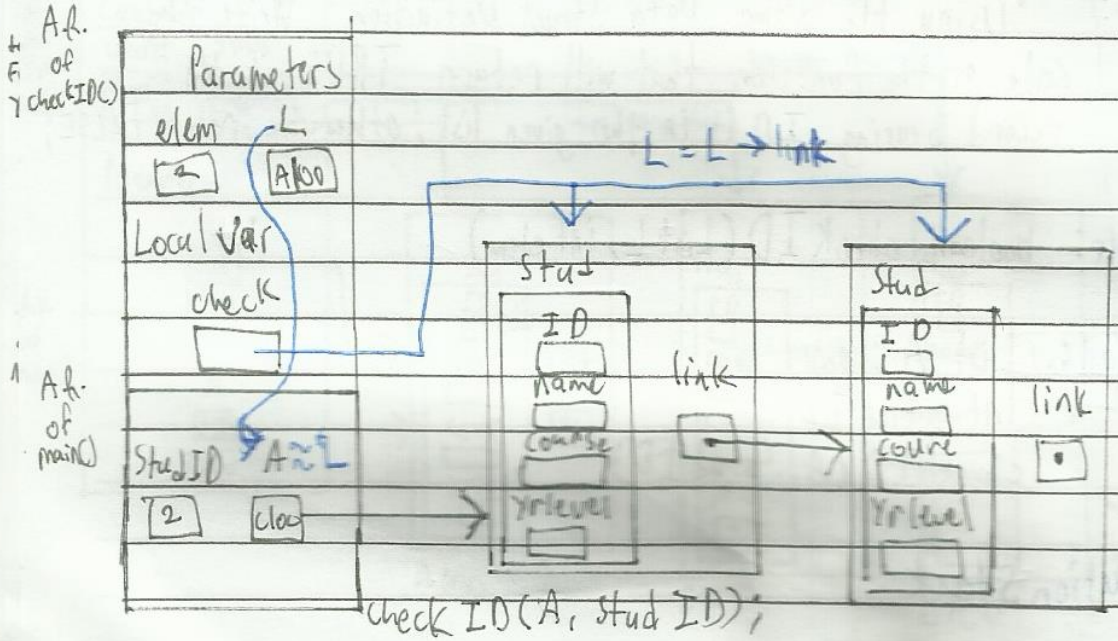
NO.

DATE 01/23/2023

Note: Not sure if correct Exec. stacks



Simulation:



NO.

DATE

4:) code of the func.:

```
boolean checkID(List L, int elem){
while (L!=NULL){
    (L→stud.ID == elem)? return TRUE : L=L→link;
  }
return FALSE;
}
```

OR

```
boolean checkID(List L, int elem){
boolean check;
    for(; L!=NULL; L=L→link){
    if(L→stud.ID == elem)? check=TRUE : check=FALSE;
  }
return check;
}
```

bessq suggestion:

```
check=(L→stud.ID == elem)? TRUE: FALSE;
```

Sterling

NO.

DATE 1/25/2023

*Support Notes:

⊙ Scalar Structures: int, float, char & Aggregate Structures: Structures

⊙ I.P.O
- Input → to the function through parameters
- Output, return the data to the calling function

⊙ Traversal → to start from the first element and to travel through the list until the last element.

* Arrays, Array Traversal & Pointer Arithmetic *

Array Declaration: datatype arrayName [arraySize];
ex: char name [8];

NOTE: size & type of an array cannot be changed once declared

How to initialize: int arr [5] = {3, 6, 9}

0	1	2	3	4
3	6	9	0	0

int arr [] = {3, 6, 9}

0	1	2
3	6	9

The name of the array is the address of the first component. i.e. pointer to the first component. Use pointer arithmetic as another way to traverse through an array.

E.g. int arr [5];

0	3	A100 - A103
1	1	A104 - A107
2	9	A108 - A10B
3	7	A10C - A10F
4	8	A110 - A113

arr = &arr[0]

arr + 1 = &arr[0] + (1) = &arr[1] ← increments by 4 bytes, the size of 1 int.

*(arr) = 3

*(arr+1) = 1

~~*(arr+2) = 9~~

*(arr+3) = 7

(*)(arr+4) = 8

dereferences &arr[0] + N where N is the number of increments. accesses the value stored inside

Sterling

NO.

DATE 1/25/2023

- PROBLEM SPECIFICATION:

Given the Array of Integers, The size of the array, and an element x, the function will return TRUE if x is in the array, otherwise return FALSE.

typedef enum {TRUE, FALSE} boolean;

① Function Header: boolean isMember(int arr[], int size, int x)

OR int *arr but [] is preferrable.

② Function Call: int A[5] = {1, 2, 3};

0	1	2	3	4
1	2	3	0	0

int B = 5; // size

int C = 3; // element x to find

boolean check = isMember(A, B, C);

to catch the return value of the function.

isMember(A, B, C)

array A where the name of the array is the address of the first component. Passes an address.

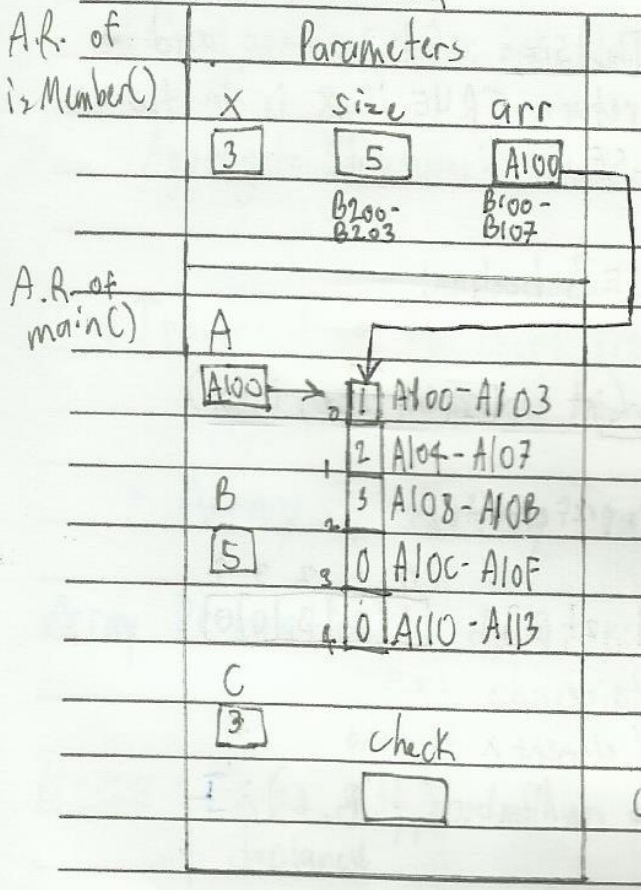
passes size of the array (int)

passes the value element x which is to be searched for

NO.

DATE 01/25/2023

3.) Execution stack



A100, 5, 3

check = isMember(A, B, C);

4.) Code:

Method 1

```
boolean isMember(int arr[], int size, int X){
    boolean check = TRUE;
    int index;
    for(index = 0; index < size && arr[index] != X; index++){
    if (index == size){
        check = FALSE;
    }
    return check;
}
```

NO.

DATE 01/25/2023

METHOD 2

```
boolean isMember (int arr[], int size, int x) {
    int index;
    for(index = 0; index<size && arr[index] != x; index++){}
    return index<size? TRUE:FALSE;
}  //is concise
```

or →

```
return check = (n==size)? FALSE : TRUE; //n is index
return (n==size) ? FALSE :TRUE;
```

- Problem Specification:

Function inputArray() allows user to input from the keyboard the total # of integers N, to be stored in the newly created array and puts N at index 0 of the new array. The values from 1 to N will also be inputted from the keyboard. In addition, the newly created array will be returned to the calling func.

* SUBTASKS * (for dumb dumb like me)

1. Input value of N
2. Allocate the Array Dynamically (malloc or calloc)
3. Initialize index 0 of the new array
4. Input N number of integers
5. Return the Array

- Function Header:

```
int * inputArray()
```

→ use * when returning an ARRAY from a function. because its a POINTER that was DYNAMICALLY ALLOCATED so it does not expire on function termination

Sterling

NO.

DATE 01/25/2023 Code: Method 1

```
int *InputArray() {
    int N, index;
    N= scanf("%d",&N); //1.

    int *arr = (int*) malloc (sizeof(int) *(N+1)); //2.

    if (arr != NULL){ // check if allocation was successful
        arr[0]=N; //3.
        for(index = 1; index <N; index++) { //4.
            scanf("%d", &arr[index]);
        }
    return arr; //5.
    }
}
```

Subtasks

1. input the value of N.
2. Dynamically Allocate memory (malloc or calloc)
3. initialize index 0 of New Array
4. input N number of integers
5. return the array

NO.

DATE 01/25/2023

method 2: Bessal's code

```
int *inputArray(){ → no parameters cuz question did not specify & does not need parameters

int N, ndx, *retval ← catches return value

scanf("%d", &N);
retval = (int*) malloc (sizeof(int)*(N+1)); → new array, as stated in the question, will hold the value of N + the N number of values i.e. N+1 number of values in the resulting array.

if (retval != NULL){ → checking if succesful allocation
    retval[0]=N; → initializing index[0] to be assigned "N"
  for (ndx=1; ndx<=N; ndx++){
      scanf("%d", &retval[ndx]);
                  OR *(retval+ndx)
  }
}
return retval; → returning the pointer to the dynamically allocated resulting array.
}
```

NO.

DATE 01/25/2023

* Malloc() & Calloc() - Explained

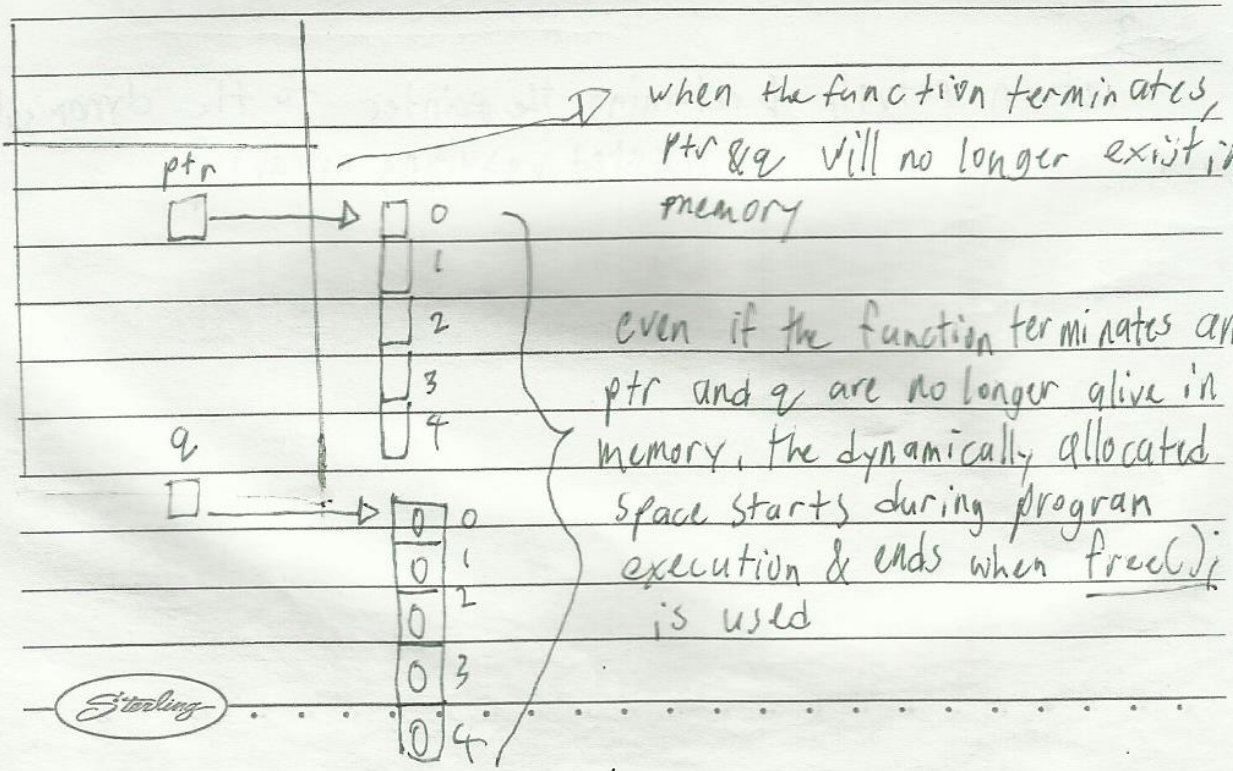
- Syntax of malloc() - void *malloc (size_t size)
  (datatype variable)

NOTE: size_t is not a keyword in C but is a datatype defined in a header file

Also NOTE: when writing code, base it on syntax, not example. Notice how void *malloc (size_t size), the correct syntax of malloc() is written like a function header, whereas something like int *ptr = (int*) malloc (5 * sizeof(int)), is an example

- Syntax of calloc() - void *calloc (size_t NoOfItems, size_t ItemSize)
  (NoOfItems: element count; ItemSize: item size)

E.g. int *q = (int*) calloc (5, sizeof (int));



when the function terminates, ptr & q will no longer exist in memory

even if the function terminates and ptr and q are no longer alive in memory, the dynamically allocated space starts during program execution & ends when free(); is used

Sterling

e

NO.

DATE

Assignment Program

- Problem Specification:

Function displayArray() will accept as parameters the array of integers & the size of the array. It will display the elements of the given array.

PROGRAM STRUCTURE:

(A) Include Files

(B) Function Prototypes

inputArray()
displayArray()
isMember()

(C) Main Function

↳ Declaration of Variables

- printf("Task 1: ")

//call inputArray() & displayArray()

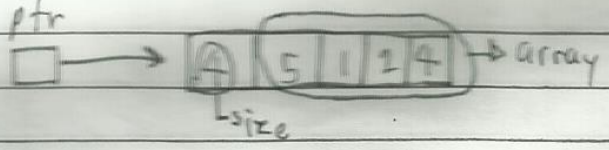
- Printf("Task 2: ")

// Call isMember() twice (test cases)

1st call → result: Element is found
2nd call → result: Element is not found

(D) Definition of the functions listed in the function prototype

- NOTE: According to the question.



displayArray() should print 5, 1, 2, 4, the elements of the Array

NO.

DATE

```
#include <stdio.h> //task A
#include <stdlib.h>

typedef enum{FALSE, TRUE} boolean;

//task B function prototypes
int *inputArray();
void displayArray(int arr[], int size);
boolean isMember(int arr[], int size, int x);

//task C Main function
void main(){
 int *myArr;
 int checkMem, checkMem2;
 printf("\n Task 1: \n"); // to call inputArray() & displayArray()
 myArr = inputArray();
 displayArray(myArr, myArr[0]);

 printf("\n Task 2: \n"); //to call isMember() twice
 checkMem = isMember(myArr, myArr[0], 4); //if element is found
 printf("\n %d", checkMem);
 checkMem2 = isMember(myArr, myArr[0], 15); //if not found
 printf("\n %d", checkMem2);
}
```

NO.

DATE

// Task D definition of the func. Prototypes

```
int *inputArray() {
    int N, index;
    int *arr;
    scanf("%d", &N);
    arr = (int *) malloc(sizeof(int) * (N+1));

    if (arr != NULL) {
        arr[0] = N;
        for (index = 1; index <= N; index++) {
            scanf("%d", &arr[index]);
        }
    }
    return arr; }
```

```
void displayArray (int arr[], int size) {
    int index;
    for (index = 1; index <= size; index++) {
        printf("%d ", arr[index]);
    }
}
```

```
boolean isMember(int arr[], int size, int x) {
    boolean check;
    int index;
    for(index = 1; index < size && arr[index] != x; index++) { }
    return (index == size) ? FALSE : TRUE;
}
```