NO.

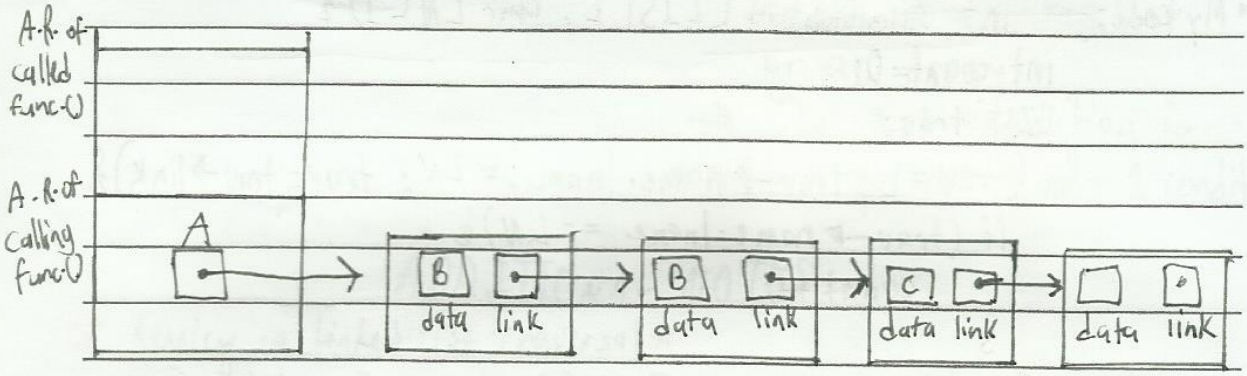
DATE 2/6/2023

PRACTICE EXERCISES & SIMULATIONS

- Delete all occurences of a given element.

data struct definition:

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



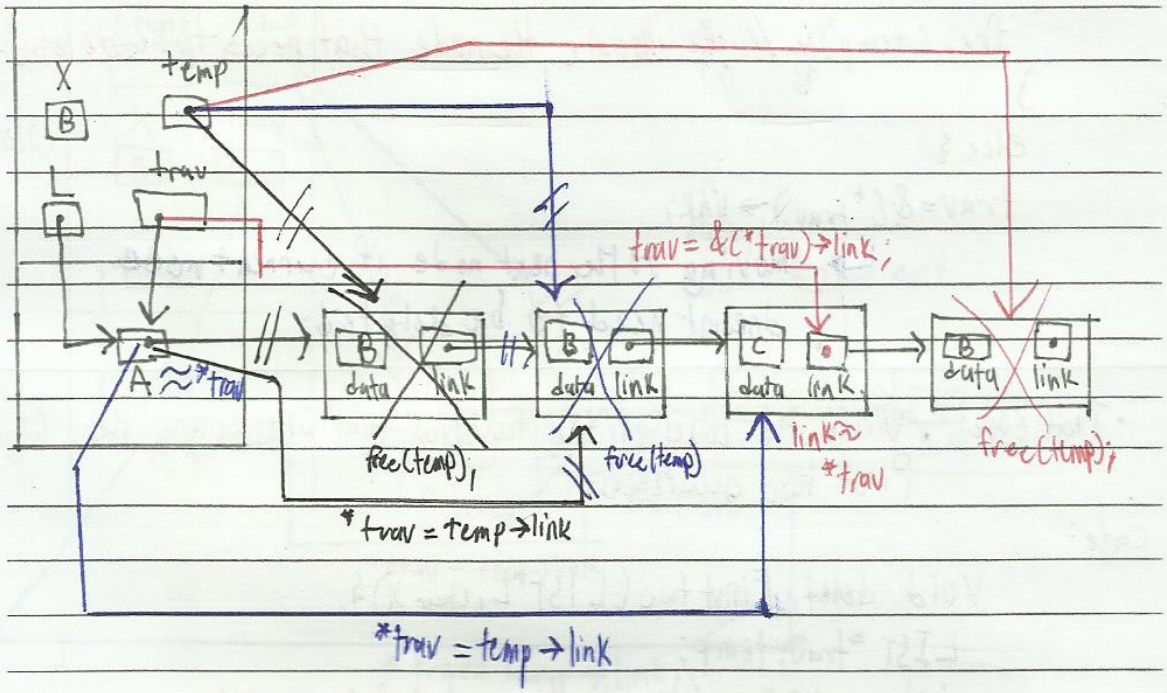
★ Code:

```
void deleteAllOccur(LIST *L, char x) {
    LIST *trav, temp;
    if(*trav != NULL) {
        for(trav = L; *trav != NULL; ) {
            if( (*trav)->data == x) {
                temp = *trav;
                *trav = temp->link;
                free(temp);
            } else {
                trav = &(*trav)->link;
            }
        }
    }
```

NO.

DATE 02/6/2023

* Drawing w/ simulation



Resulting List:

Code Explanation:

```
void deleteAllOccur(LIST *L, char x){
```

deletion involved, pass by address

```
LIST *trav, temp;
```

pointer-node, will hold the address to be deleted

PPN

```
for(trav = L; *trav != NULL){
```

only one condition statement because we expect to move through the entire list. "delete ALL occurences".

Note:

In exec stack, trav & L hold the same value; they are both pointers to pointer. trav = L, not trav = &L.

```
if((*trav)->data == x){ //checking if data == x, value to be deleted.
    temp = *trav; // let temp hold the address of the node to be deleted.
    *trav = temp->link; } let it point to the node after the current node
```

(the current node is the one that needs to be deleted.

the link pointing to the current node

cont. →

NO.

DATE 2/6/2023 *continuation*

```
free(temp); // deallocate the node that needs to be deleted.
}
else{
trav=&(*trav)->link;
```

↳ moving to the next node if current node doesnt need to be deleted.

- Test case: Write the code of the function that will delete the first two occurences.

Code:

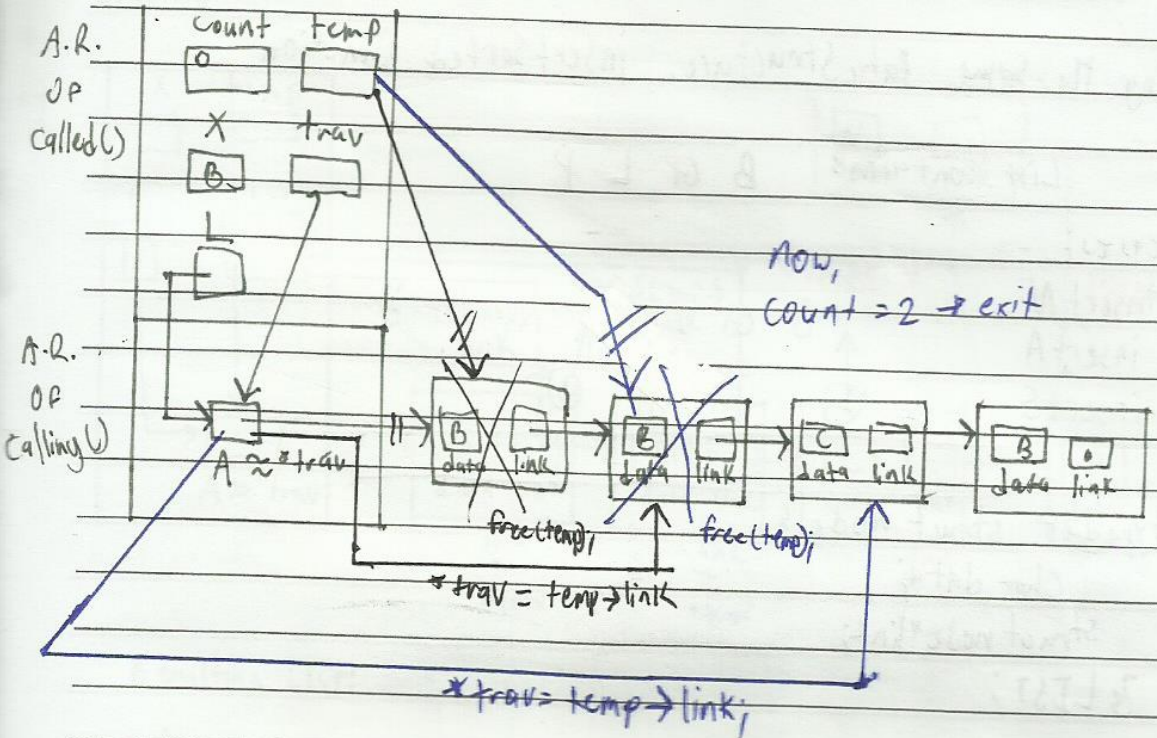
```
void deleteFirsttwo(LIST*L, char x){
LIST *trav, temp;
int count=0; // counter to count # of occurences
for(trav=L; *trav!=NULL && count!=2;){
    if((*trav)->data==x){
      temp=*trav;
      *trav=temp->link;
       free(temp);
       count++; // increment count if occurence is spotted
    }else
      trav=&(*trav)->link;
  }
}
```

Deletion Code

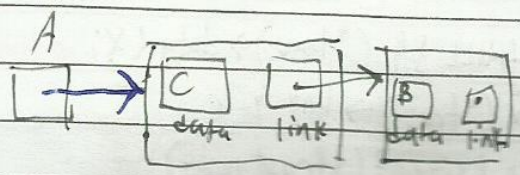
↳ two conditions cuz we dont expect to traverse the whole list.

NO.

DATE 26/2023



Resulting List:



NO.

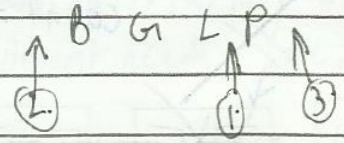
DATE 2/6/2023

- Using the same data structure, insertSorted function

List contains: B G L P

Test cases:

1. insert N
2. insert A
3. insert S




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

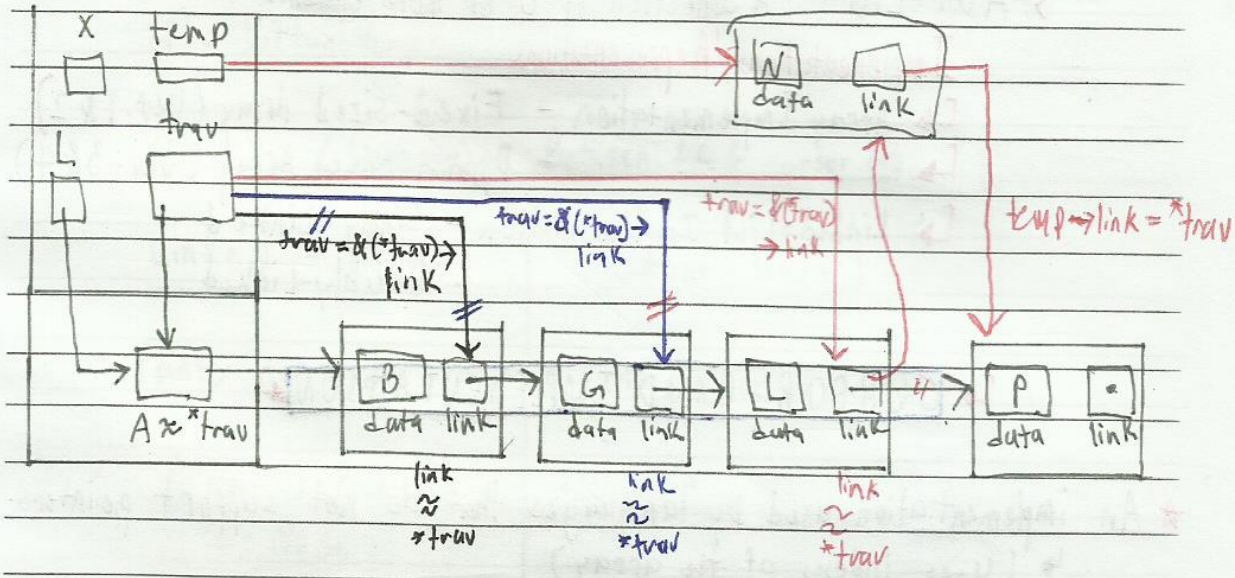
Code:

```
void insertSorted(LIST *L, char x) {
LIST, temp, *trav;
for(trav=L; *trav != NULL && (*trav)->data < x;
        trav = &(*trav)->link );
temp = (LIST) malloc (sizeof(struct node));
if (temp != NULL) {
    temp->data = x;
    temp->link = *trav;
  *trav = temp;
  }
}
```

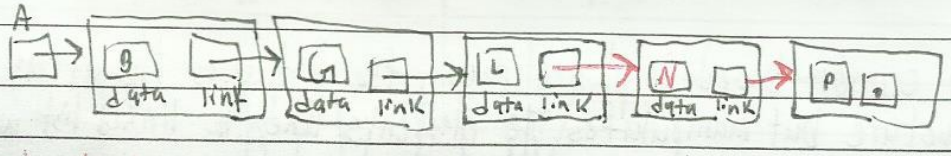
NO.

DATE

Drawing



Resulting List:



Code Explanation:

```
void insertSorted(LIST *L, char x){
    LIST temp, *trav;                                   // compares ABC() values
    for(trav=L; *trav!=NULL && (*trav)->data<x; trav=&(*trav)->link){}
```

two conditions, not expected to traverse the whole list everytime

Butterfly notation

loop, once the conditions are no longer met, loop exists

```
    temp=(LIST) malloc(sizeof(struct node));  //dynamically allocate so it points to new node
    if(temp!=NULL){ //check if allocation was successful
        temp->data=x; //assign values to new node
        temp->link=*trav; //connects new node to the next node in the LIST
                          (or NULL if last node).
        *trav=temp; //connects previous node to new node
    }
}
```

Sterling