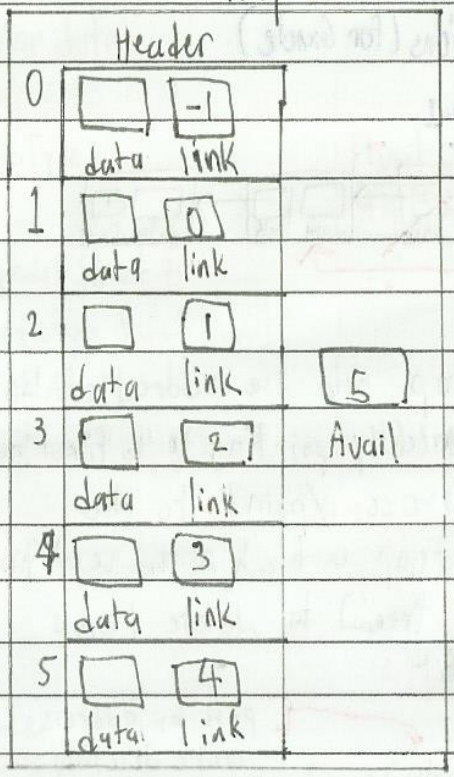
NO.

DATE 2/13/2023

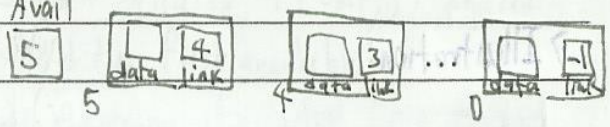
Recall:

SIZE : 6

Virtual Heap VH



Other way of illustration:



• 3 virtual Heap Management Routines Recalled:

1.) initVirtualHeap()

2.) deleteFirst() - Equivalent to allocSpace. Given the VirtualHeap, the function will delete the 1st available node from the list of available nodes & return the index (of the deleted node) to the calling. If the list of available nodes is empty, -1 is returned.

3.) insertFirst() - Equivalent to deallocSpace - Given the VirtualHeap & the index to a node / assuming to be non-existing in the list of available nodes, the function will insert at the 1st position the node in the list of available nodes.

Sterling

NO.

DATE 2/13/2022

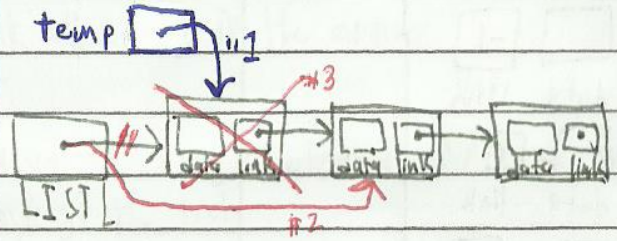
1.) deleteFirst() ← creates & gives space
Also known as allocSpace() in Cursor Based, acts as malloc in Cursor-Based

- Linked-List Version (for comparison)

> Data Struct Definition:

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

> Illustrations (for Guide)



1.) Let temp hold the address of L, which is currently pointing to the first node.
2.) Let *L access/point to the link of temp, which is the next node.
3.) use free() to delete temp.
4.) Profit

> Code:

← pass by address since we're deleting stuff

```
void deleteFirst (LIST *L){
    LIST temp;

    if ( (*L) != NULL) { //check if LIST is not empty
        temp = *L;          //#1
        *L = temp->link;    //#2
        free(temp);         //#3
    }
}
```

- NOTE: No traversal since we're deleting the first node

Sterling

NO.

DATE 2/13/2023

★ Cursor-based Implementation Version

#define MAX 6

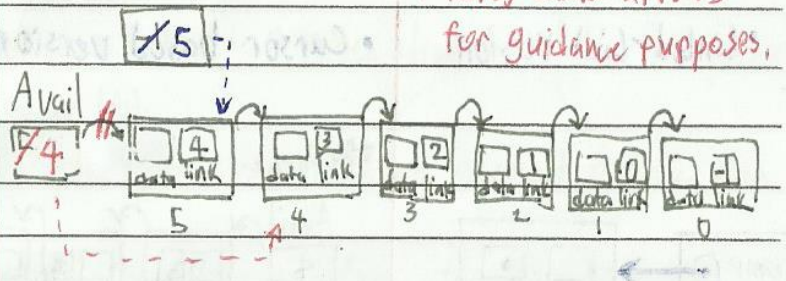
> Data Struct. def:

```
typedef struct {
    char data;
    int link;
} nodeType;

typedef struct {
    nodeType Nodes[MAX];
    int Avail;
} VirtualHeap;

typedef int cbList;
```

> Illustration



1.) if Avail is not equal to -1 (empty), proceed

2.) Let temp hold the first available node since we will be returning its index from the heap

3.) Set Avail to the index of the next available node (Basically updating avail)

4.) return the temp which holds the index

> Code:

```
int deleteFirst (VirtualHeap *VH) {
    int temp = -1; // initialize temp as -1. -1 is equivalent to NULL
                   // if the list of available nodes is empty
//#1 if (VH->Avail != -1) { //check if avail is empty or not
//#2     temp = VH->Avail;
//#3     VH->Avail = VH->Nodes[temp].link;
    }
//#4 return temp;
}
```

return dataType

Sterling

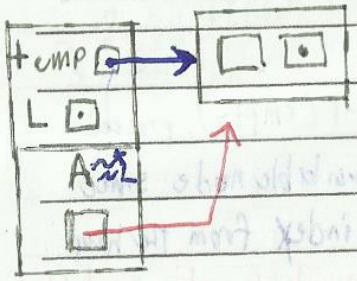
NO.

DATE 2/13/2022

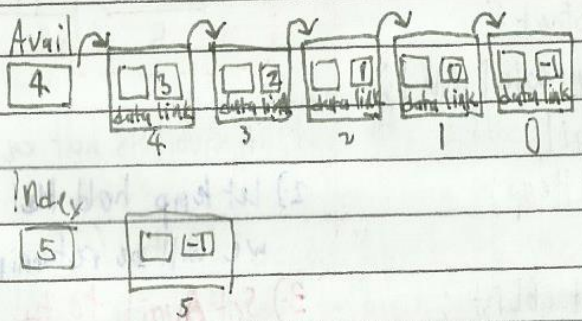
- insertFirst() - also known as deallocSpace in Cursor-based ▽ equivalent to free
  - eg

> Illustrations:

- Linked-List version

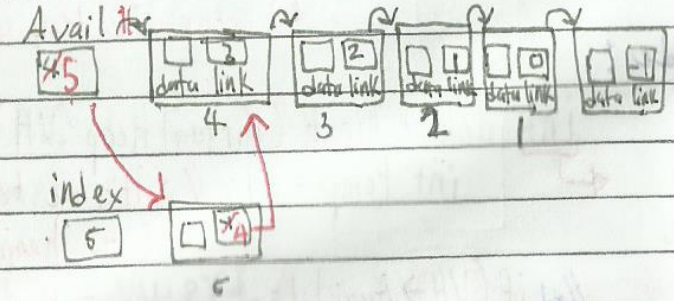


- Cursor-based version



- Linked-List code & comparison

```
void insertFirst(LIST *L, char elem){
LIST temp = (LIST) malloc(sizeof(struct node));
    temp->data = elem;
//1 temp->link = *L;
//2 *L = temp;
}
```



> Code:

```
void insertFirst(VirtualHeap *VH, int index){
//1  VH->Nodes[index].link = VH->Avail;
//2  VH->Avail = index;
}
```

NOTE: Uses the same logic of insertFirst in Linked-List

Sterling

NO.

DATE 2/15/2023

★ ADT List Functions (in Cursor-Based) ★

1.) initList() - initializes the list

2.0.) insertLast() - The function will insert a given element at the last portion of the given list

2.1.) insertFirst() - The function will insert a given element at the first portion of the given list.

3.) deleteElem() - The function will delete the given element from the given list.

4.) displayList() - displays the elements from the given list

• Comparisons between LL & CB •

> insert()	(LL)	(CB)
1.) Allocate space for a new node	• temp=(LIST) malloc(sizeof (struct node));	• temp = allocSpace(VH);
2.) Assign the data to the node	• temp→data = elem;	• VH→Nodes[temp].data = elem;
3.) Link the node to the remainder of the list	• temp→link = *trav;	• VH→Nodes[temp].link = *trav;
4.) Link the area of insertion to the new node	• *trav = temp;	• *trav = temp;

> delete()	(LL)	(CB)
1.) Let temp hold the node to be removed	• temp = *trav;	•
2.) Link the node before temp to the node after temp	• *trav = temp→link;	•
3.) Free the memory space consumed by temp	• free(temp);	•

Sterling

NO.

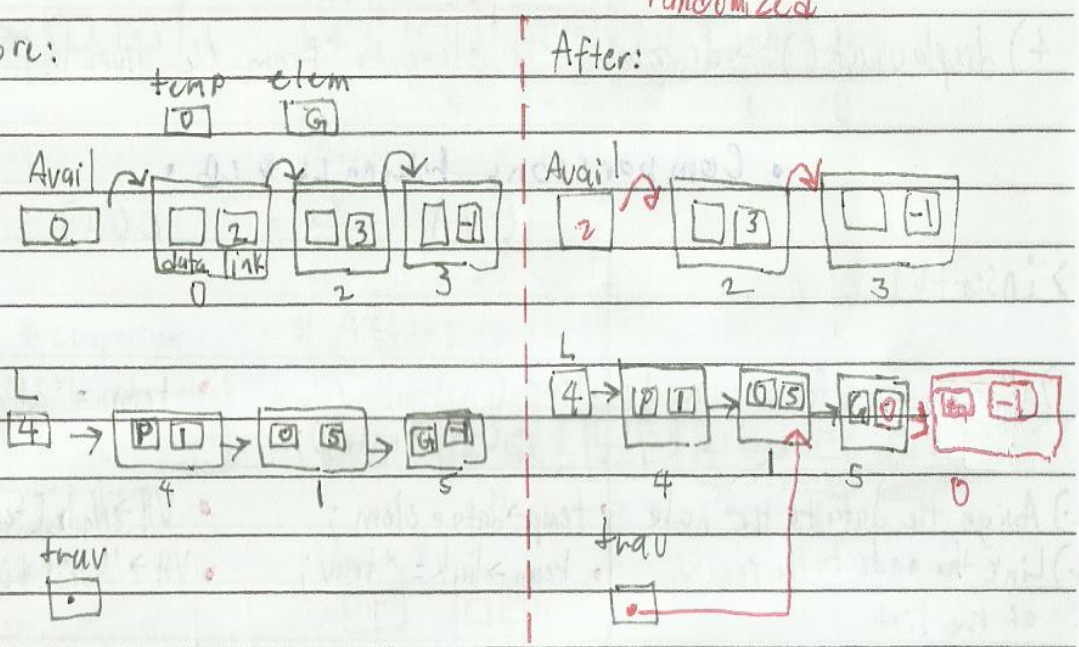
DATE 2/15/2023

- insertLast() – uses PPN, pass by address

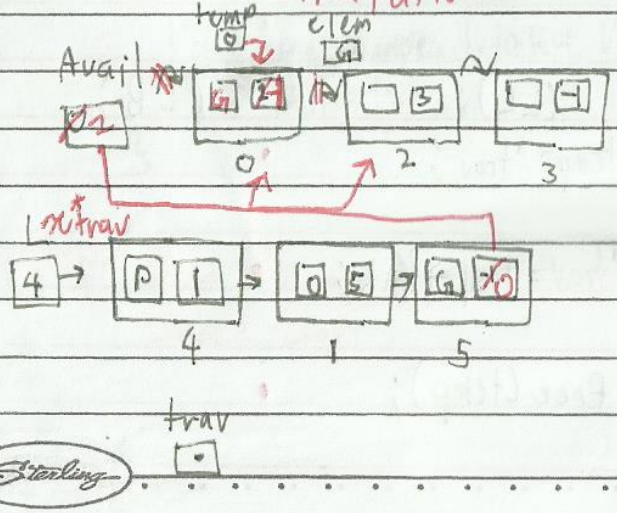
> General steps:

1.) Dynamically allocate space for new node
2.) Assign the data to the node
3.) Link the node to the remainder of the List
4.) Link the area of isolation to the new node

> Illustration: (in cursor-based) //what if it was randomized



Simulation:



Sterling

NO.

DATE 2/15/2023

> CODE:

```
void insertLast(VirtualHeap *VH, CbList *L, char elem){
    //1 int temp = allocSpace(VH);   -> equivalent to malloc, also called deleteFirstL)
      int *trav;
for(trav=L; (*trav)!=-1; trav=&VH->Nodes[*trav].link){}
    if(temp!=-1){
    //2 VH->Nodes[temp].data = elem;
    //3 VH->Nodes[temp].link = -1;
    //4 VH->Nodes[*trav].link = temp;
}
```

Linked-List ver. for comparing

```
void insertLast(List *A, char elem){
  List *trav
  List temp;

  for(trav=L; *trav!=NULL; trav=&(*trav)->link){}
//1 temp = (List)malloc(sizeof(struct node));
    if(temp!=NULL){
    //2 temp->data = elem;
    //3 temp->link = *trav;
    //4 *trav = temp;
    }
```

Sterling

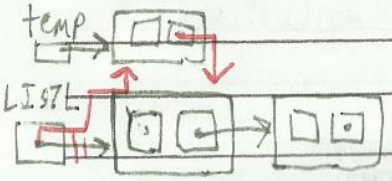
DATE 2/15/2023

• insertFirst() — pass by address, PV

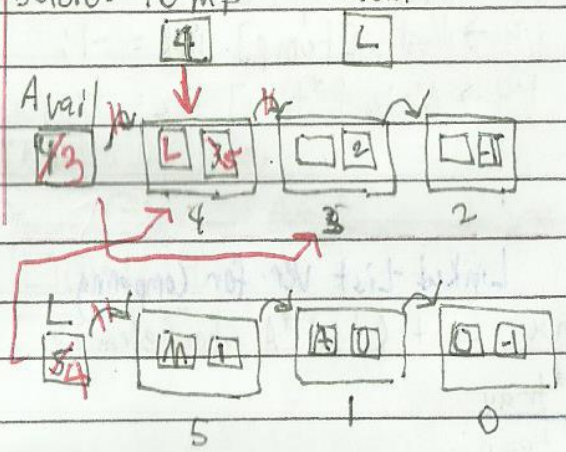
Note: Steps are generally the same

> Illustration:

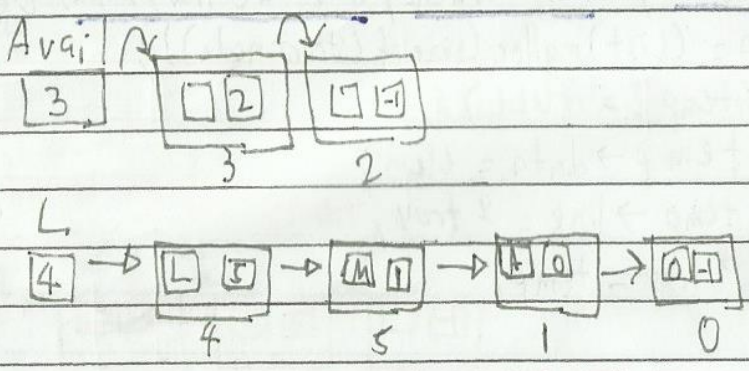
• Linked List ver:



• Cursor Based Ver:



After simulation:



help

Sterling

NO.

DATE 2/15/2023

> Code:

```
Void insertFirst (VirtualHeap *VH, CBList *L, char elem) {
    CBList temp;
    temp = allocSpace (VH);

    VH -> Nodes [temp].data = elem;
    VH -> Nodes [temp].link = *L;
    *L = temp;
}
```

Linked-List version:

```
Void insertFirst (LIST *L, char elem) {
    List temp;
    temp = (LIST) malloc (sizeof (struct node));

    temp -> data = elem;
    temp -> link = *L;
    *L = temp;
}
```

NO.

DATE 2/15/2023

- deleteElem() - uses PPN, Pass by address

> General steps:

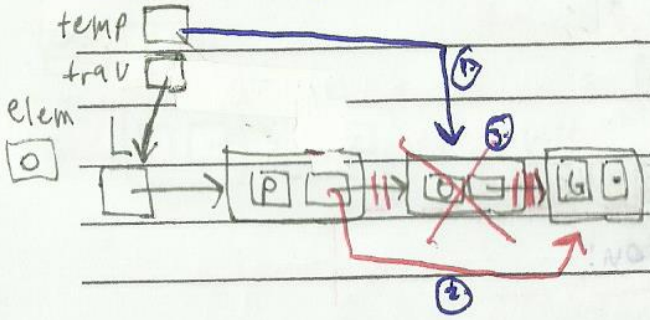
1.) Let temp hold the node to be removed

2.) Link the node before temp to the node after temp

3.) Free the memory space

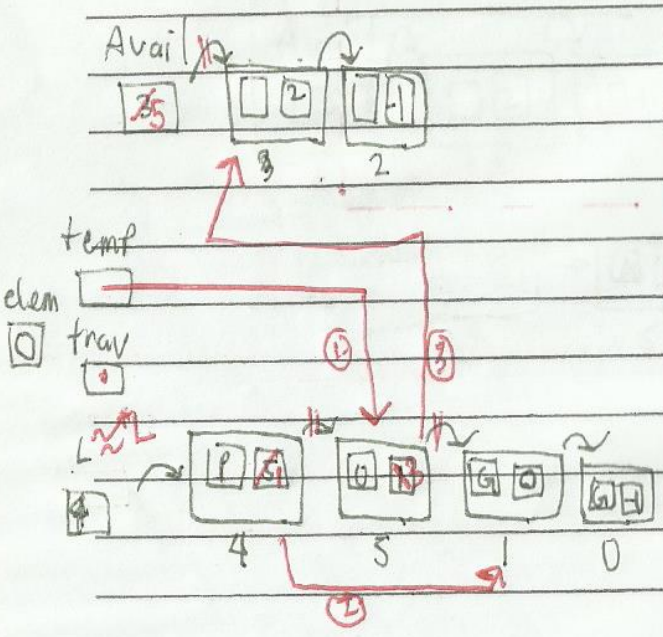
★ ILLUSTRATIONS ★

- Linked-List version:

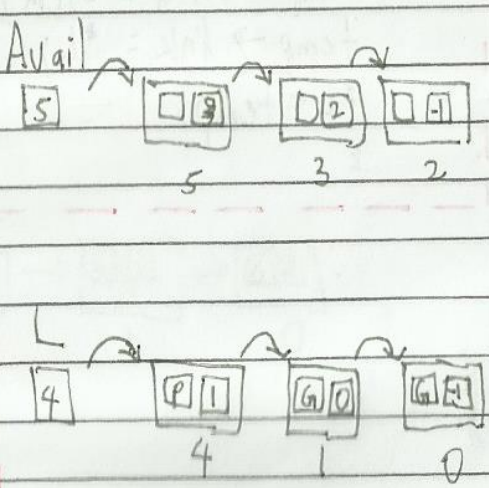


- Cursor-Based Version

Before & during Simulation:



After:



Sterling

//send help
idk if its correct

NO.

DATE 2/15/2023

> CODE:

```
Void deleteElem (VirtualHeap *VH, CBList *L, char elem) {
    CBList *trav, temp;

    For (trav = L; *trav != -1; ) {
        if (VH->Nodes[*trav].data == elem) {
    //1   temp = *trav;
    //2   *trav = VH->Nodes[temp].link;
    //3   deallocSpace(VH, temp);   -> equivalent to free()
                                     also called as insertFirstC)
        } else {
            trav = &VH->Nodes[*trav].link;
        }
    }
}
```

Linked-List Comparison

```
Void deleteElem (LIST *L, char elem) {
    LIST *trav, temp;

    for (trav=L; *trav != NULL;) {
        if ((*trav)->data == elem) {
    //1   temp = *trav;
    //2   *trav = temp -> link;
    //3   free(temp);
        } else {
            trav = &(*trav)->link;
        }
    }
}
```

Sterling

NO.

DATE 2/15/2023

• displayList()

↳ is pass by copy
since we're not changing anything

> Code:

```
Void displayList (CBList L, VirtualHeap VH) {
  CBList trav;

for (trav = L; trav != -1; trav = VH.Nodes[trav].link) {
    printf("%c ", VH.Nodes[trav].data);
}
```

Linked-List version

```
Void displayList (LIST L) {
  LIST trav;
for (trav = L; trav != NULL; trav = trav->link) {
    printf("%c ", trav->data);
}
```

NO.

DATE 2/15/2023

- initList() → has 2 versions: Void (ptr) & 1 o call/ declared

- Version 1:

```
void initList(CBList *L){
    (*L) = -1;
}
```

- Version 2:

```
CBList initList (){
    CBList L;
    return L=-1;
}
```

Note: can forego declaring a variable & just return -1 since its an integer