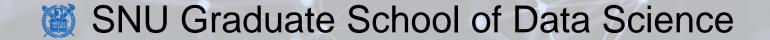
Review

- Structures
 - Declaration
 - typedef
 - Arrays and Pointers
- Dynamic Memory Allocation
 - Arrays vs. Linked Lists
 - Stack vs. Heap
 - malloc

Linked Lists in C - Basics

Lecture 35-1

Hyung-Sin Kim



Let's revisit linked lists, but in C!

Single Linked Lists in Python (no sentinel)

```
class LinkedNode():

def __init__(self, x):

self.val = x

self.next = None
```

```
class SLList():
      def init (self, x: int) -> None:
           self.first = None
           self.size = 0
      def addFirst(self, x: int) -> None:
           newFirst = LinkedNode(x)
           newFirst.next = self.first
           self.first = newFirst
           self.size += 1
       def getFirst(self) -> int:
           if self.first:
               return self.first.val
           return None
       def getSize(self) -> int:
           return self.size
```

Single Linked Lists in C (no sentinel)

```
typedef struct nodeType LinkedNode;
struct nodeType {
  int val;
  LinkedNode *next:
};
LinkedNode *createNode(int x) {
  LinkedNode *newNode;
  newNode = (LinkedNode *) malloc(sizeof(LinkedNode);
  newNode->val = x;
  newNode->next = NULL;
  return newNode;
```

Don't have the __init__ method in this case.

Actually... there is no method at all since it is not a class but a structure.

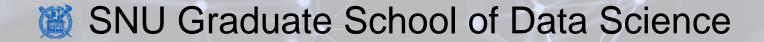
We should define necessary functions outside

Create a new node with a new memory block on heap

Linked Lists in C – Useful Functions

Lecture 35-2

Hyung-Sin Kim



Single Linked Lists in C (no sentinel)

```
typedef struct listType SLList;
    struct listType {
      LinkedNode *first;
      int size;
    };
    int main(void) {
      SLList myLL = \{NULL, 0\};
      addFirst(&myLL, 10);
      printf("%d\n", getFirst(&myLL));
      getSize(&myLL);
      printSLList(&myLL);
```

We need to define these functions

Single Linked Lists in C (no sentinel)

```
void addFirst(SLList *LL, int x) {
  LinkedNode *newFirst:
  newFirst = createNode(x);
  newFirst->next = LL->first;
  LL->first = newFirst;
  LL->size++;
int getFirst(SLList *LL) {
  if (LL->first != NULL)
     return LL->first->val;
  return 0;
```

```
int getSize(SLList *LL) {
     return LL->size;
void printSLList(SLList *LL) {
     LinkedNode *curr = LL->first;
     printf("size: %d, firstVal: %d, allVals: ", getSize(LL), getFirst(LL));
     while (curr != NULL) {
        printf("%d->", curr->val);
        curr = curr->next;
     printf("END\n");
```

We can add nodes to a linked list. Great!

Then? You will experience memory error sometimes later since there is no deallocation!

Let's practice ©

Delete a node from SLList

- For simplicity, we assume that every node has a unique integer
- Step 1) Make a search function
 - LinkedNode *searchNode(SLList *LL, int x) { /* Your code */ }

- Step 2) Make a delete function that is a bit more complex than **searchNode** since you need to reorganize next pointers and decrease list size
 - void deleteNode(SLList *LL, int x) { /* Your code */ }

Delete a node from SLList

- What do you see on your screen when you write the main function as that on the right side?
 - I mean... if your code ever works... ©

```
int main (void) {
    SLList myLL = {NULL, 0};
    printSLList(&myLL);
    addFirst(&myLL, 10);
    printSLList(&myLL);
    addFirst(&myLL, 20);
    printSLList(&myLL);
    addFirst(&myLL, 30);
    printSLList(&myLL);
    deleteNode(&myLL, 20);
    printSLList(&myLL);
    deleteNode(&myLL, 30);
    printSLList(&myLL);
    return 0;
```

Delete a node from SLList

- What do you see on your screen when you write the main function as that on the right side?
 - I mean... if your code ever works... ©

```
size: 0, firstVal: 0, allVals: END
size: 1, firstVal: 10, allVals: 10->END
size: 2, firstVal: 20, allVals: 20->10->END
size: 3, firstVal: 30, allVals: 30->20->10->END
size: 2, firstVal: 30, allVals: 30->10->END
size: 1, firstVal: 10, allVals: 10->END
```

```
int main (void) {
    SLList myLL = {NULL, 0};
    printSLList(&myLL);
    addFirst(&myLL, 10);
    printSLList(&myLL);
    addFirst(&myLL, 20);
    printSLList(&myLL);
    addFirst(&myLL, 30);
    printSLList(&myLL);
    deleteNode(&myLL, 20);
    printSLList(&myLL);
    deleteNode(&myLL, 30);
    printSLList(&myLL);
    return 0;
```

Delete a node from SLList – Solution

searchNode implementation

```
LinkedNode *searchNode(SLList *LL, int x) {
    LinkedNode *curr = LL->first;
    while (curr != NULL) {
        if (curr->val == x)
            return curr;
        curr = curr->next;
    return NULL;
```

Delete a node from SLList – Solution

deleteNode implementation

```
void deleteNode(SLList *LL, int x) {
   LinkedNode *curr = LL->first;
   LinkedNode *prev = NULL;
   while (curr != NULL) {
       if (curr->val == x) {
           if (curr == LL->first) {
                LL->first = LL->first->next;
            else {
                prev->next = curr->next;
           free(curr);
            LL->size--;
            return;
        else {
            prev = curr;
            curr = curr->next;
```

Summary

- Single Linked Lists
 - Basic structure
 - Create
 - Add
 - Search
 - Delete

Thanks!