C Programming Language

(8th class)

Dohyung Kim

Assistant Professor @ Department of Computer Science

Today …

- **Linked List Data Structure**

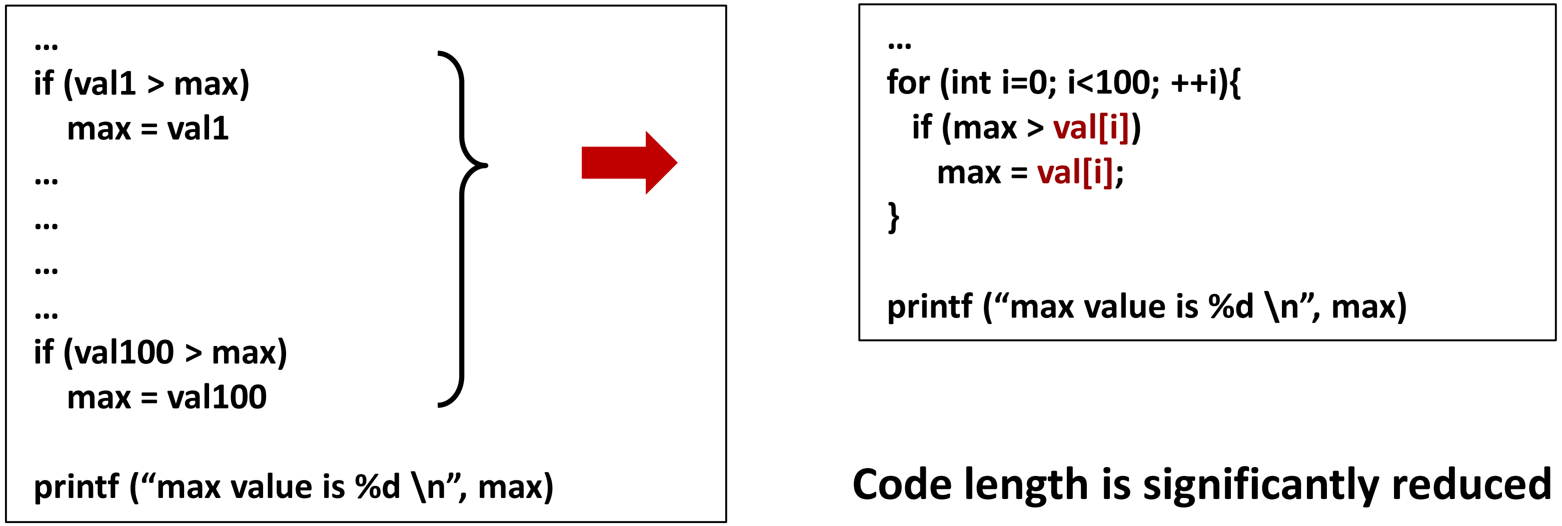
Problem

- Let us assume that we have to write a program that manipulates math scores of 100 students in the class. How shall we store those scores in your program?

```
int val[100];

for (int i=0; i<100; ++i){
    printf(“input the %d-th student score : ”);
    scanf(“%d”, &val[i]);
}
```

Array

- **A useful way to store a collection of the same type of data**
  - Removal of repeated codes
  - Easy manipulation of data with a single variable name and indexes



```
…
if (val1 > max)
   max = val1
…
…
…
…
if (val100 > max)
   max = val100

printf (“max value is %d \n”, max)
```

```
…
for (int i=0; i<100; ++i){
  if (max > val[i])
    max = val[i];
}

printf (“max value is %d \n”, max)
```

Code length is significantly reduced

- **But …**

Problem

- Let us assume that we have to write a program that manipulates math scores of 100 students in the class. How shall we store those scores in your program?

```
int val[100];
```

- **If a few students newly join the class, how can we add their scores to the existing array?**

- **Can we reduce the memory space when more than 50 students leave the class?**

Disadvantage of arrays

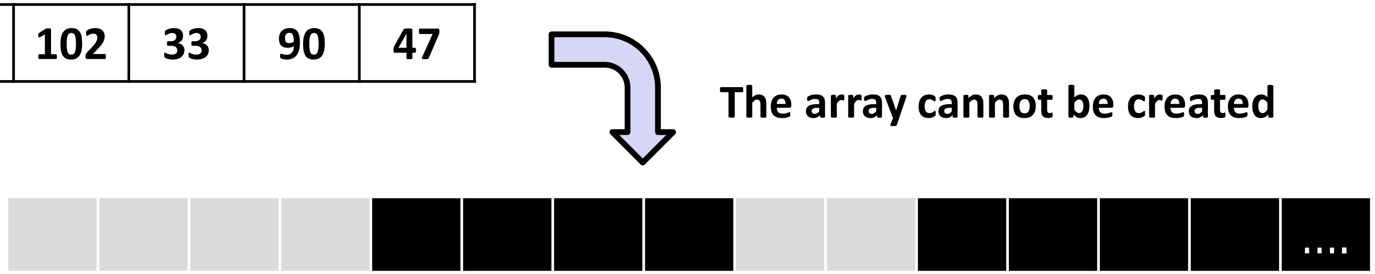
- **Arrays are static and of the fixed size**
  - Cannot be easily extended or shrunk
  - What if a new data should be added?

21	**102**	**33**	**90**	**….**	**41**	**67**	**52**

- Programmers allocate arrays which seem "large enough"
- Memory space could be wasted

- **Contiguous memory space is required**
  - All elements should be allocated in one block of memory

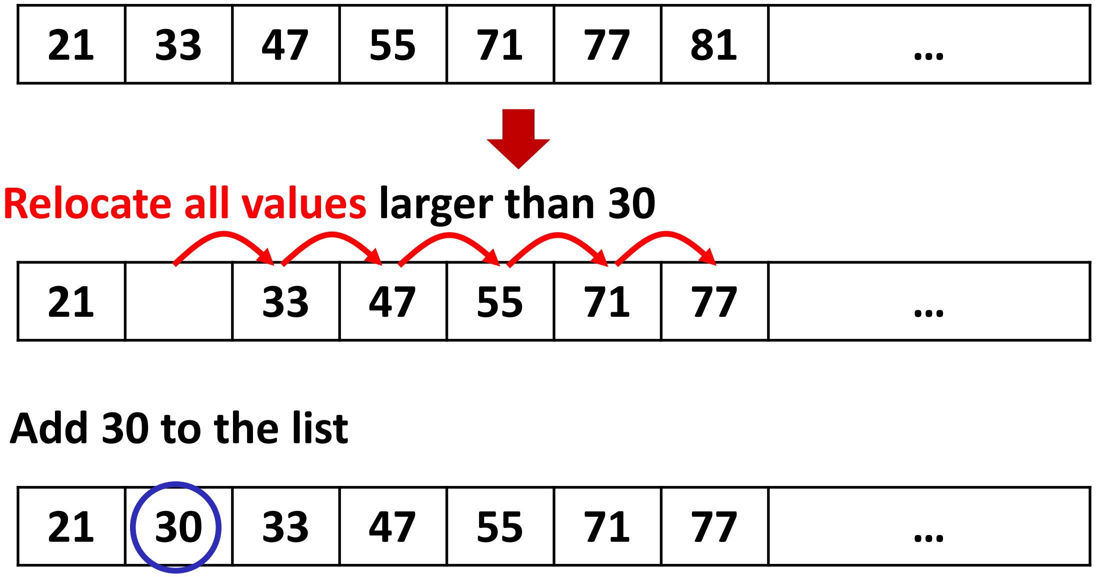
21	**102**	**33**	**90**	**47**



The array cannot be created

Disadvantage of arrays

- **Expensive insertion or deletion**
  - Insertion into the middle of an array
    - Let’s add 30 in the sorted array



21	**33**	**47**	**55**	**71**	**77**	**81**	**…**

Relocate all values larger than 30

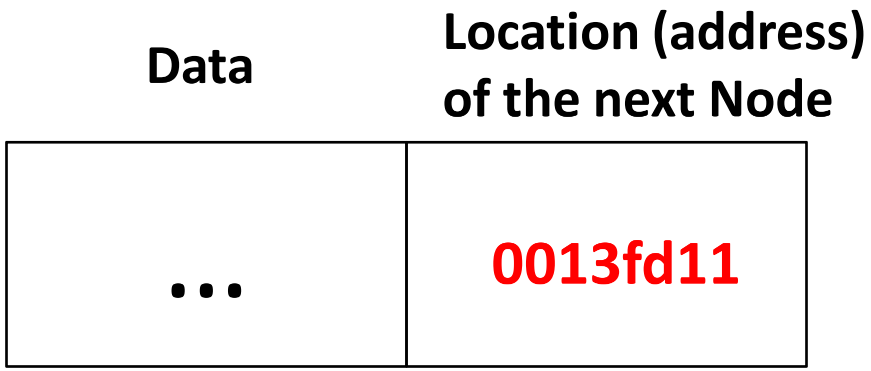
21		**33**	**47**	**55**	**71**	**77**	**…**

Add 30 to the list

21	**30**	**33**	**47**	**55**	**71**	**77**	**…**

Linked List

- Similar to arrays, linked lists store collections of data
- A linked list is composed of nodes
- Node



Data

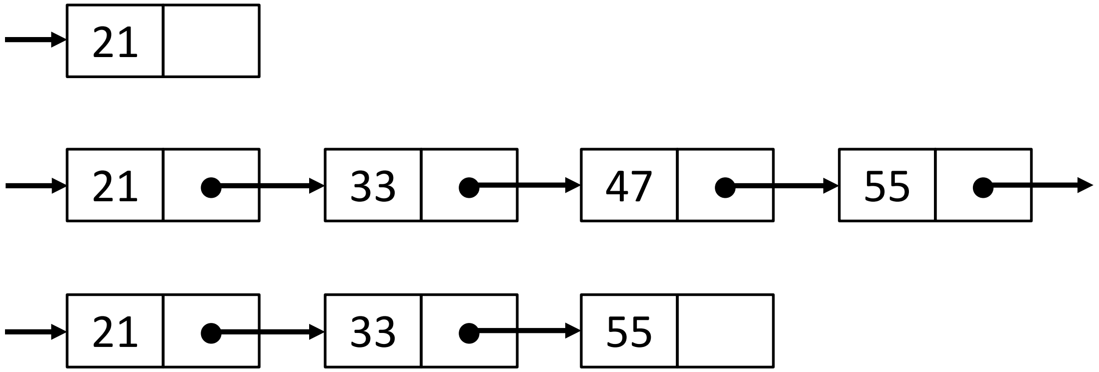
Location (address) of the next Node

…

0013fd11

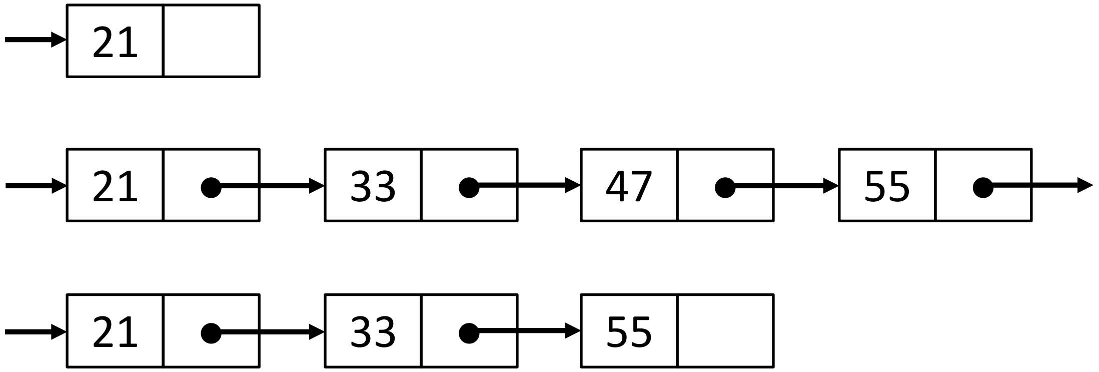
Linked List

- **Linked Lists are dynamic**
  - The length of a list can increase or decrease as necessary



Linked List

- **Linked Lists are dynamic**
  - The length of a list can increase or decrease as necessary



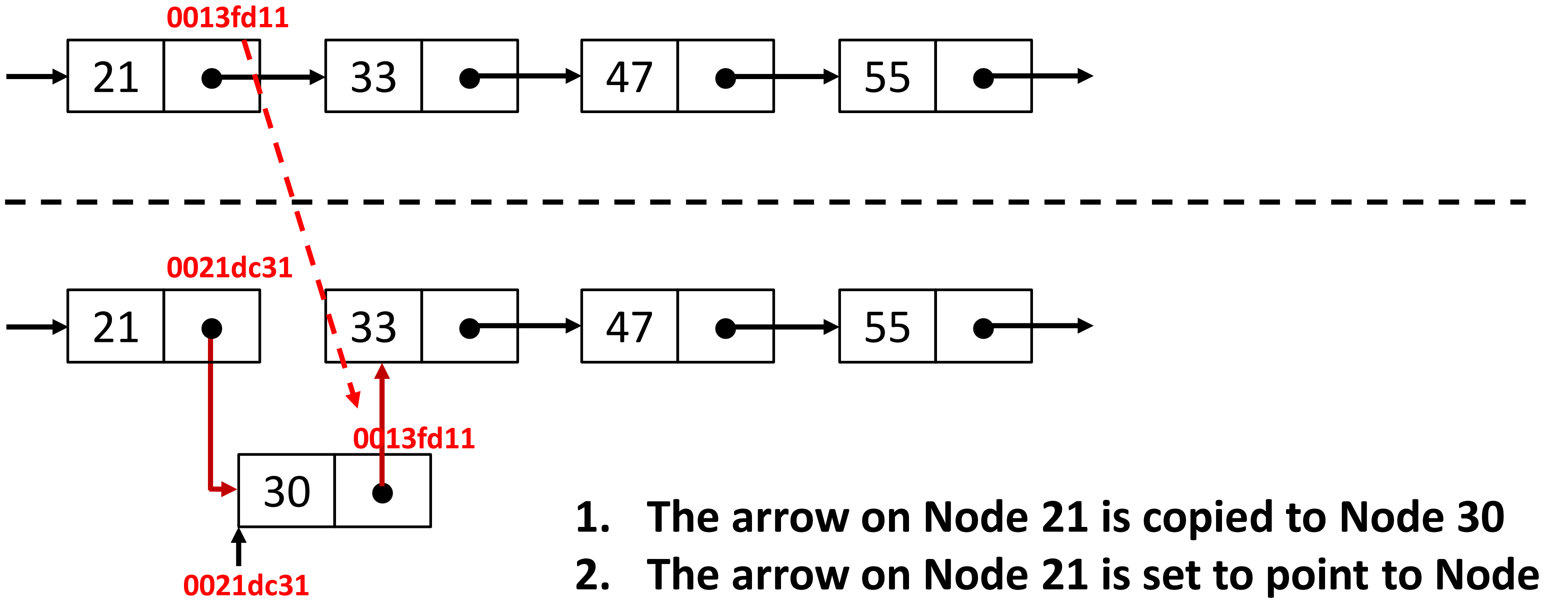
- **Physical memory space for each element can be separated.**
  - An arrow points to the following memory space that may not be physically continuous

Linked List

- **Easy maintenance**
  - Insertion and deletion are simple

Linked List

- **Easy maintenance**
  - Insertion and deletion are simple
  - Let’s add 30 to a sorted linked list



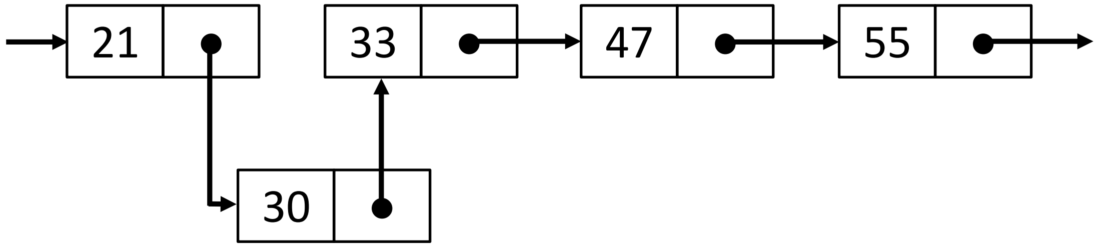
1. **The arrow on Node 21 is copied to Node 30**
2. **The arrow on Node 21 is set to point to Node 30**

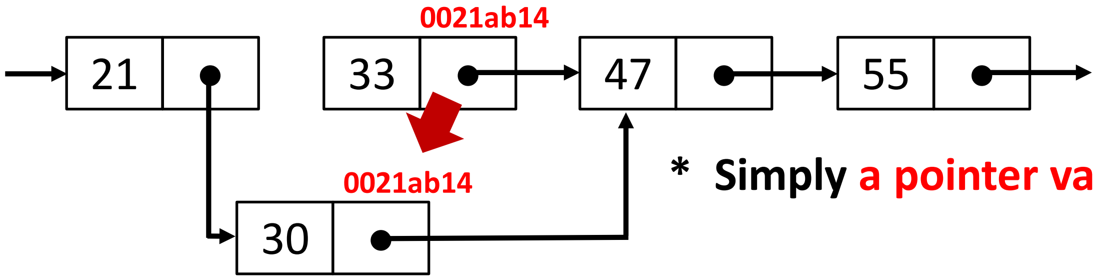
* **Simply the pointer values (arrows) are updated to insert a new value**

* **No additional copies are required to relocate data**

Linked List

- **Easy maintenance**
  - Insertion and deletion are simple
  - Let’s add 30 to a sorted linked list
  - Let’s remove 33 from the list

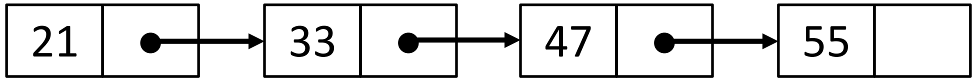




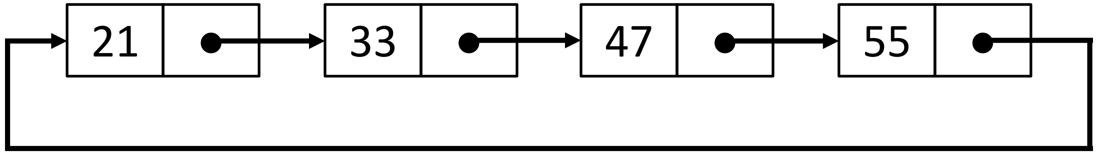
* **Simply a pointer value (arrow) is copied**

Types of linked lists

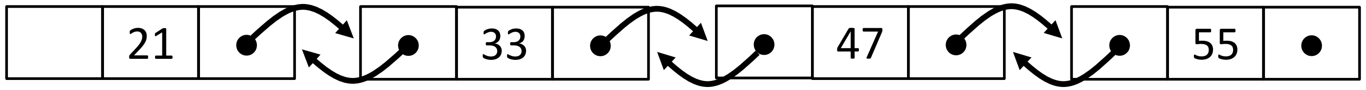
- **Singly linked list**



- **Circular linked list**



- **Doubly linked list**



Linked List in C

- node structure that comprises a linked list

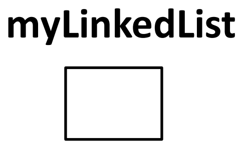
```
struct node {
   int val;
   struct node *nextNode;
};
```

data

pointer (address, arrow)

- Creation of an empty linked list

```
struct node *myLinkedList = NULL;
```



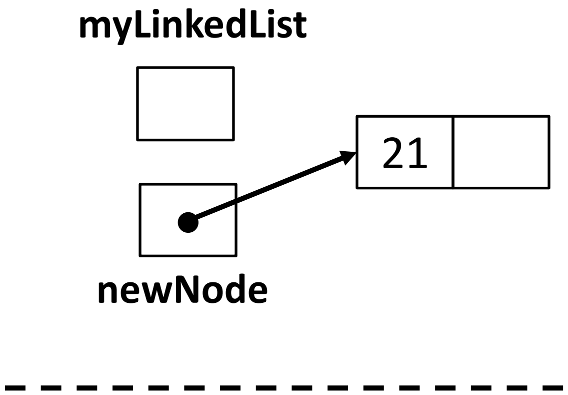
Linked List in C

- Create a new node for a new data

```
struct node *newNode = (struct node*) malloc ( sizeof ( struct node));

newNode->val = 21;
newNode->nextNode = NULL;
```

create a memory space for a new node

data and pointer values are set



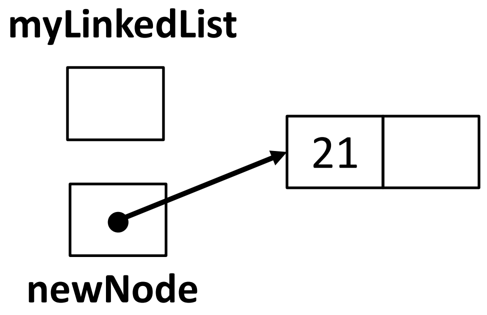
Linked List in C

- Create a new node for a new data

```
struct node *newNode = (struct node*) malloc ( sizeof ( struct node));

newNode->val = 21;
newNode->nextNode = NULL;
```

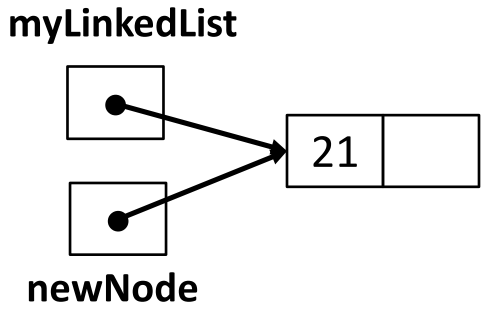
create a memory space for a new node

data and pointer values are set



- Add the first new node to the link

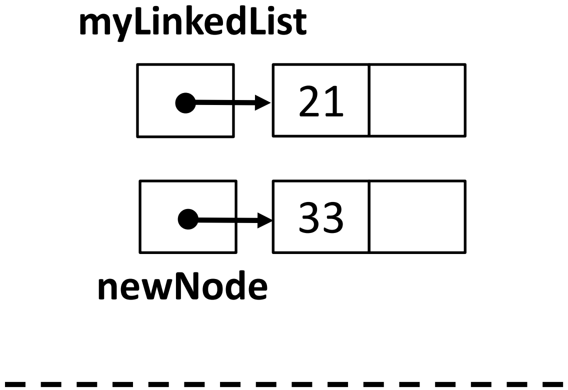
```
myLinkedList = newNode;
```



Linked List in C

- **Create another new node for the value of 33**

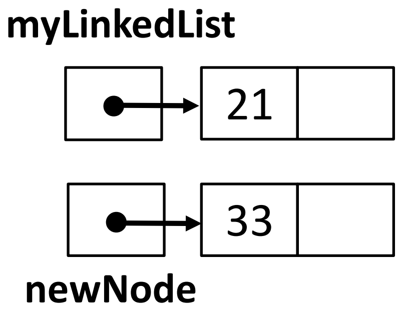
```
struct node *newNode = (struct node*) malloc ( sizeof ( struct node));

newNode->val = 33;
newNode->nextNode = NULL;
```



Linked List in C

- Create another new node for the value of 33

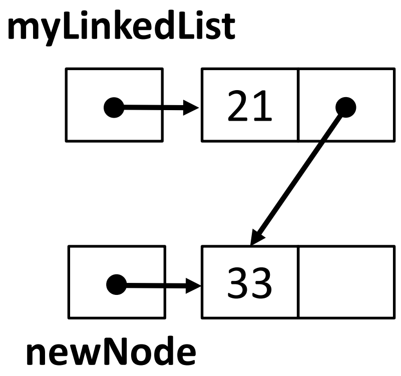
```
struct node *newNode = (struct node*) malloc ( sizeof ( struct node));

newNode->val = 33;
newNode->nextNode = NULL;
```



- Add the new node to the end of the list

```
struct node *curPos = NULL;
curPos = myLinkedList;
while (curPos->next != NULL)
   curPos = curPos->next;
curPos->next = newNode;
```

curPos points to the end node in the list, and the new node connects behind that end node.



Question

- **What are disadvantages of linkedlists compared to arrays?**

Question

- **What are disadvantages of linkedlists compared to arrays?**
  - We need to store arrows (pointers) as well as data
  - Data should be read from the beginning
  - Data is stored incontiguously, greatly increasing the time to access individual data

Remarks

- **Compared to arrays, linked lists can**
  - Manage memory space more dynamically
  - Add or remove data in a simpler way
- **Linked List consists of NODES with data and location information of the adjacent node (address).**
- **Storing the addresses, nodes can be connected in a chain**

Q and A