Structured Programming
- Linked List

Dr Donglong Chen

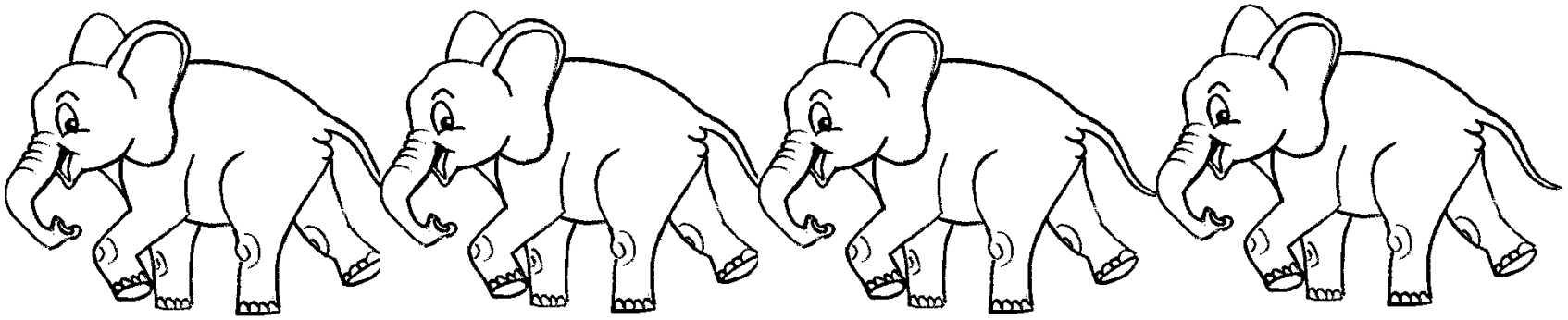
With Thanks to Dr. Xin Feng and Dr. Haipeng Guo

With courtesy to Dr. Judy Feng of UIC

Outline

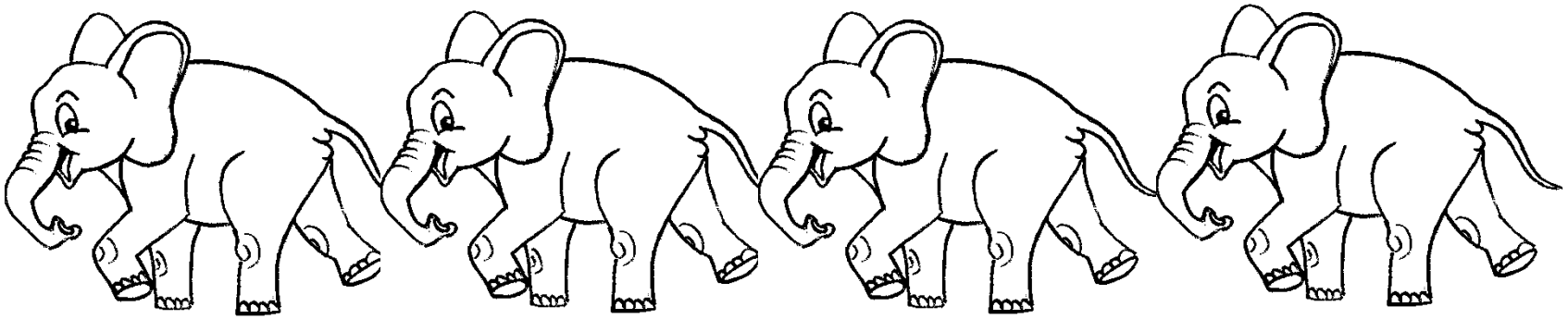
- Concept of linked list
- List operations
- Advantages and disadvantages

An Elephant Troop



What are the main features of this troop?

A Linked List



- A linked list consists of a sequence of data records such that in each record there is a field that contains a reference (i.e., a link) to the next record in the sequence.

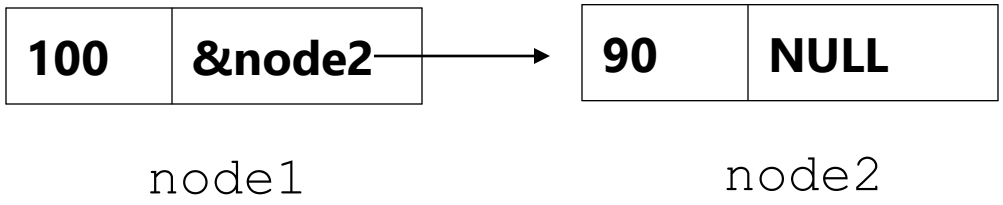
Declaration of A Node

```
struct node{
    int score;
    struct node *next;
};

struct node node1, node2;

node1.score = 100;
node2.score = 90;

node1.next = &node2;
node2.next = NULL;
```



An Example

```
struct node{
   char name[20];
   int score;
   struct node *next;
};
struct node *pnode, *head
pnode = (struct node*)malloc(sizeof(struct node));
strcpy(pnode -> name, "John");
pnode -> score = 100;
head = pnode;
pnode = (struct node*)malloc(sizeof(struct node);
strcpy(pnode -> name, "Tony");
pnode -> score = 90;

head -> next = pnode;
pnode -> next = NULL;
```

Difference Between . And ->

```
. is the member of a structure
-> is the member of a POINTED TO structure
```

So the `.` is used when there is a direct access to a variable in the structure. But the `->` is used when you are accessing a variable of a structure through a pointer to that structure.

Exercise #1

```
struct node{
   char name[20];
   int score;
   struct node *next;
};
struct node node1, node2;

strcpy(node1.name, "John");
node1.score = 100;

strcpy(node2.name, "Tony");
node2.score = 90;

node1.next = &node2;
node2.next = NULL;
```

What link list will be produced?

List Operations

- Create a list
- Delete a list
- Search a node
- Insert a node
- Delete a node

Create A List

- To create a list, we must
  - create each node
  - create links between nodes
  - designate the first (head) node
  - assign NULL to the link of the last node

Create A List

```
struct node *pnode1, *pnode2, *head;

pnode1 =(struct node*)malloc(sizeof(struct node))
pnode1 -> data = ...
head = pnode1;
```

desinagate the first (head) node

```
while (…){
  pnode2 = (struct node*)malloc ...
  pnode2 -> data = ...
  pnode1 -> next = pnode2;
  pnode1 = pnode2;
}
```

create each node

create links between nodes

```
pnode2 -> next = NULL;
```

assign NULL to the link of the last node

Create A List

```
struct node *pnode1, *pnode2, *head;

pnode1 =(struct node*)malloc(sizeof(struct node))
pnode1 -> data = ...
head = pnode1;

while (…){
  pnode2 = (struct node*)malloc ...
  pnode2 -> data = ...
  pnode1 -> next = pnode2
  pnode1 = pnode2;
}

pnode2 -> next = NULL;
```

What if we remove this statement?

Delete A List

- To delete a list, we must
  - delete from the head node
  - before a node is deleted, the link to the next node must be remembered
  - free the deleted node if the memory is dynamically allocated

Delete A List

```
void deleteList(struct node* head)
{
   struct node *next, *current;
   current = head;
   while (current != NULL) {
      next = current -> next;
      free(current);
      current = next;
    }
 }
```

delete from the first (head) node

remember the next node

free the node

Delete A List

```
void deleteList(struct node *head)
{
   struct node *next, *current;
   current = head;
   while (current != NULL) {
      next = current -> next;
      free(current);
      current = next;
    }
 }
```

What if this statement is missed?

What if this statement is missed?

Search A Node

- There are two kinds of search
  - search by index
  - search by element

Search by Index

```
struct node *searchByIndex(struct node *head, int id)
{
   struct node *current;
   int i = 1;
   current = head;
   while ((i < id) && (current != NULL)) {
     current = current -> next;
     i++;
    }
    return current;
 }
```

Search by Index

```
struct node *searchByIndex(struct node *head, int id)
{
    struct node *current;
    int i = 1;
    current = head;
    while ((i < id) && (current != NULL)) {
      current = current -> next;
      i++;
     }
     return current;
 }
```

What if we miss this?

Search by Element

```
struct node{
    int score;
    struct node *next;
}
struct node *searchByElement(struct node *head, int score)
{
  struct node *current;
  current = head;
  while ((current != NULL)&&(current -> score != score))
      current = current -> next;
  return current;
 }
```

Insert A Node

- We can insert
  - before a node
    - the node is the head
    - the node is not the head
  - after a node

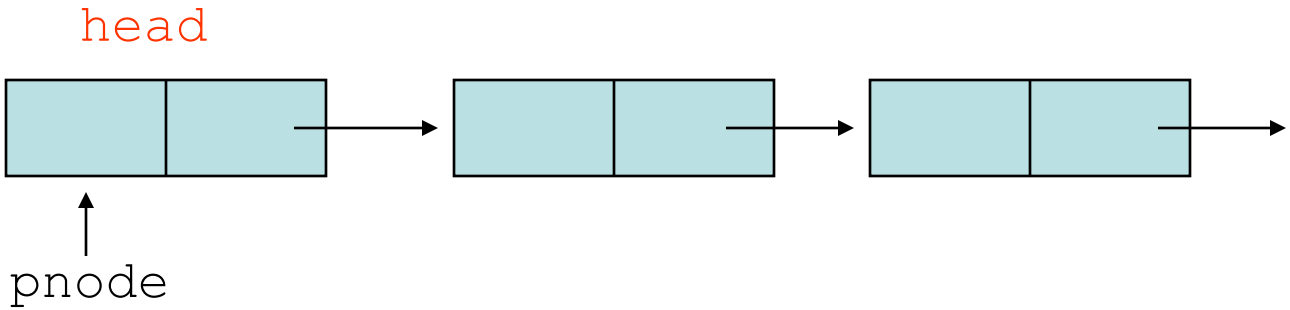
In the following slides, Node is defined as follows.

```
typedef struct node{
    …
    struct node *next;
} Node;
```

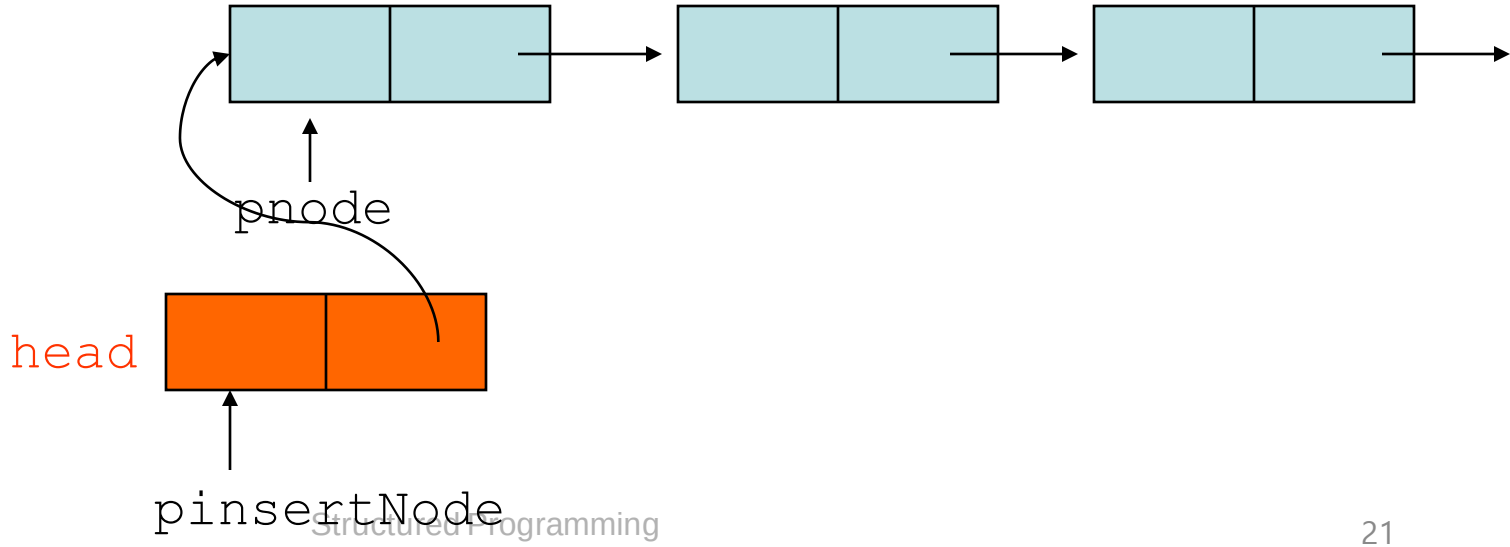
Insert Before A Node

```
insertBefore(Node *head, Node *pnode, Node *pinsertNode)
```

Case 1:



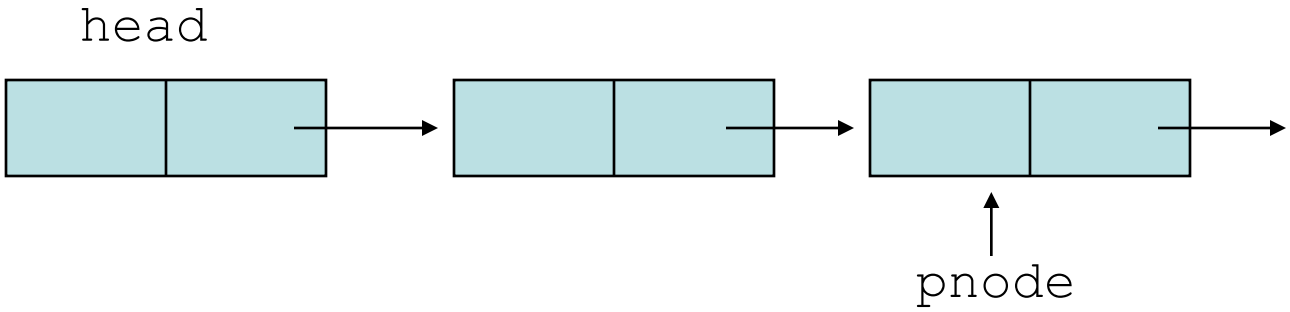
After insertion:



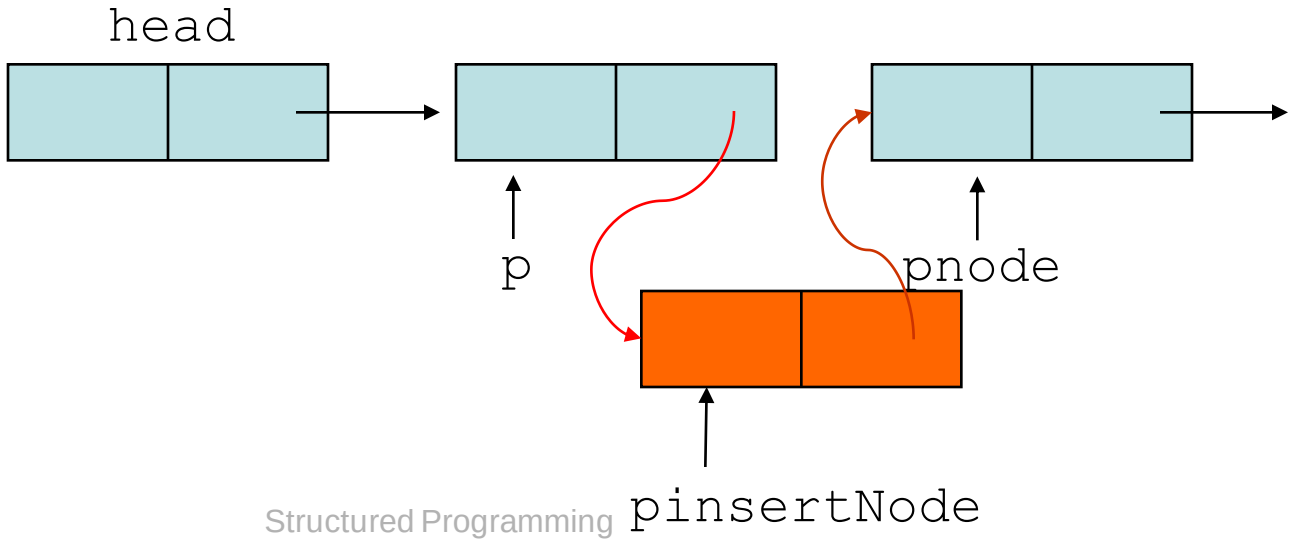
Insert Before A Node

```
insertBefore(Node *head, Node *pnode, Node *pinsertNode)
```

Case 2:



After insertion:

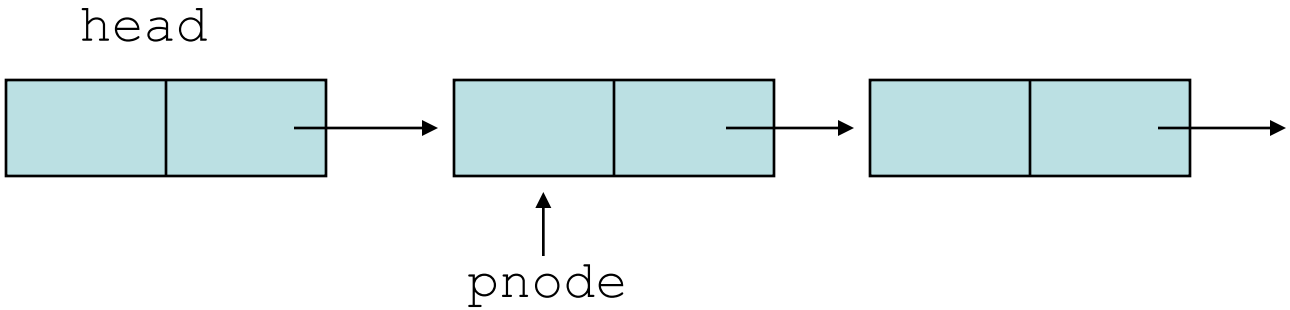


Insert Before A Node

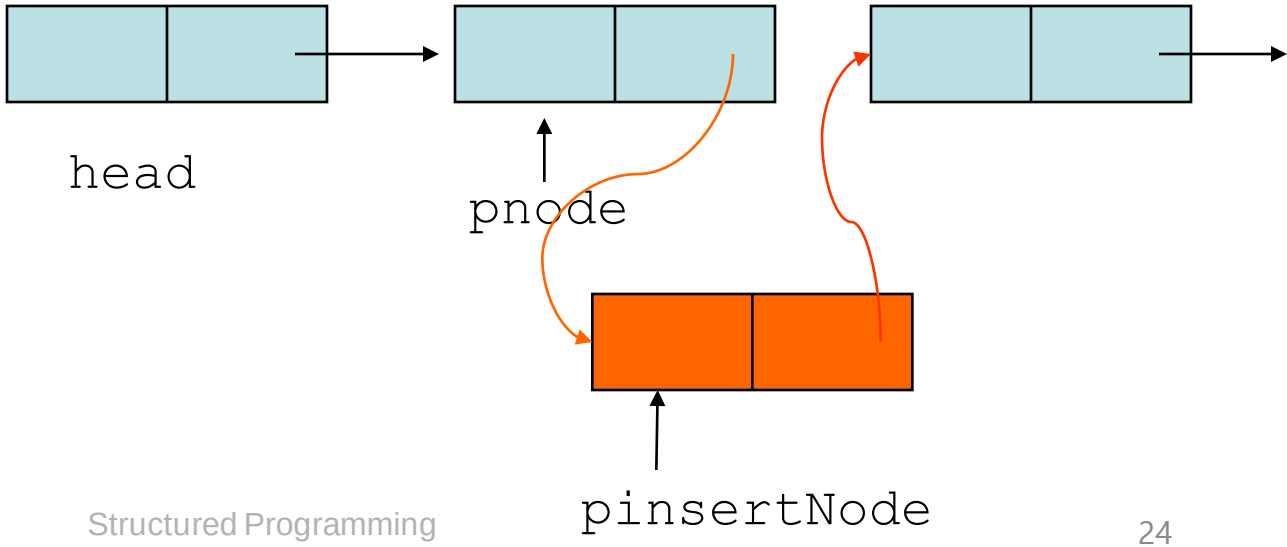
```
Node *insertBefore(Node *head, Node *pnode, Node *pinsertNode)
{
   if (pnode == head){
     pinsertNode -> next = head; head = pinsertNode;
   }
   else{
     Node *p = head;
     while((p != NULL)&&(p -> next != pnode)) p = p -> next;
     if (p!= NULL){
         p -> next = pinsertNode;
         pinsertNode -> next = pnode;
     }
  }
   return head;
}
```

Insert After A Node

```
insertAfter(Node *pnode, Node *pinsertNode)
```



After insertion:



Insert After A Node

```
void insertAfter(Node *pnode, Node *pinsertNode)
{
   if (pnode == NULL)
     return;
   pinsertNode -> next = pnode -> next;
   pnode -> next = pinsertNode;
}
```

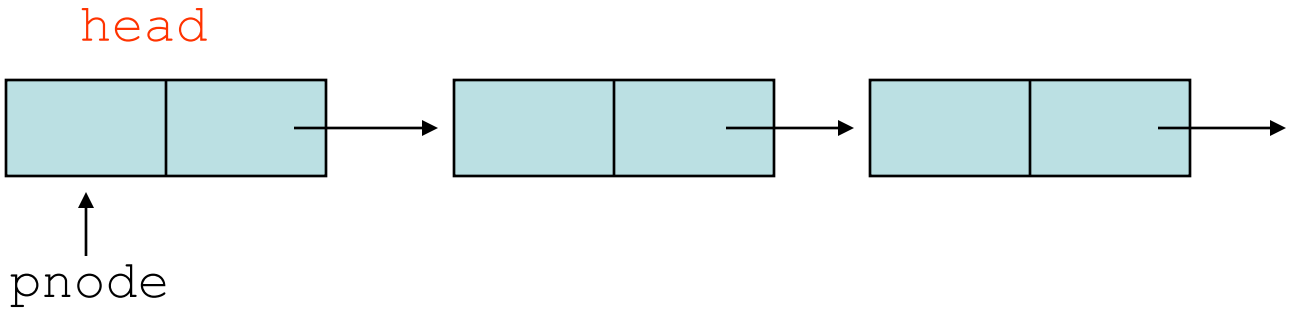
Delete A Node

- The deleted node can be
  - head
  - any of other nodes

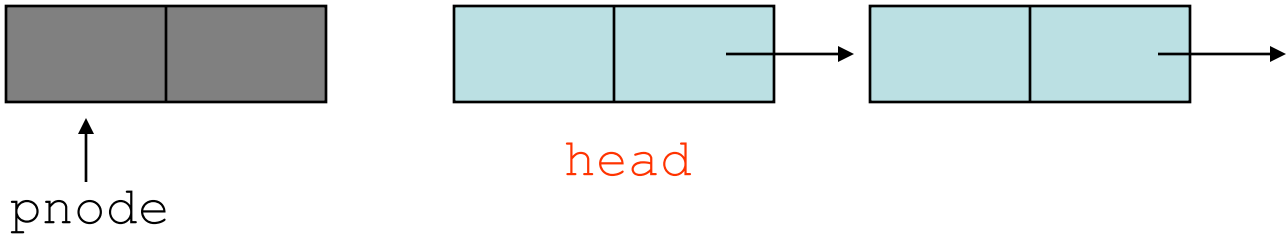
Delete A Node

```
deleteNode(Node *head, Node *pnode)
```

Case 1:



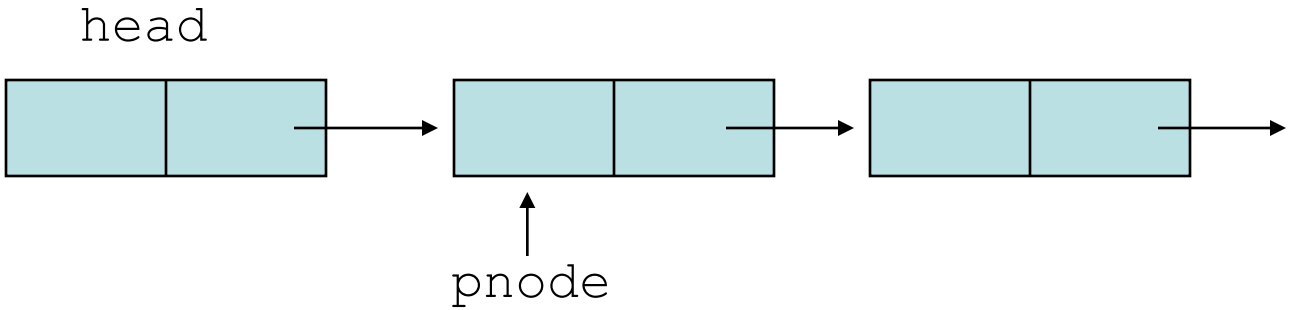
After deletion:



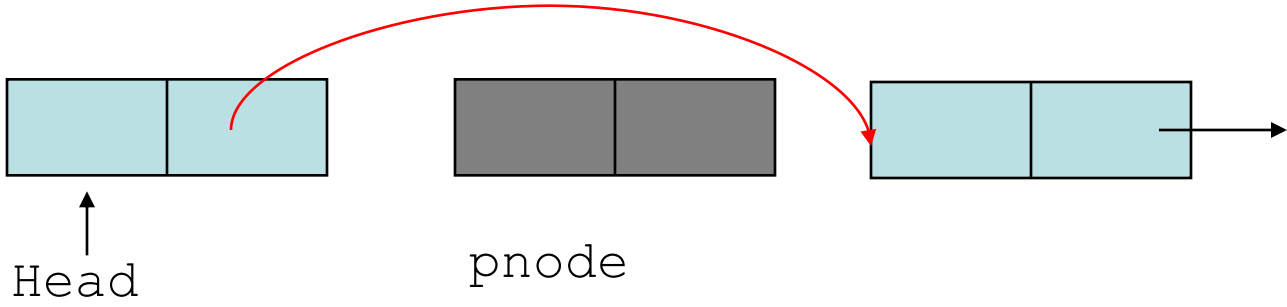
Delete A Node

```
deleteNode(Node *head, Node *pnode)
```

Case 2:



After deletion:



Delete A Node

```
Node *deleteNode(Node *head, Node *pnode)
{
   if (pnode == NULL) return head;
   else if (pnode == head)
     head = head -> next;
   else {
     Node *p = head;
     while ((p != NULL) && (p -> next != pnode))
       p = p -> next;
     if (p != NULL)
       p -> next = p -> next -> next; //pnode -> next
   }
   free(pnode);
   return head;
}
```

Advantages and Disadvantages

- Compare linked list with array from the following points
  - memory space
  - search a node
  - insert a node
  - delete a node

Exercise #2

What are the advantages and disadvantages of linked list over array?

https://www.thecrazyprogrammer.com/2016/11/advantages-disadvantages-linked-list.html

Summary

- Linked list can be used to store a collection of data
- We can search, insert or delete a node in a linked list
- Linked list, compared with array, has advantages and disadvantages