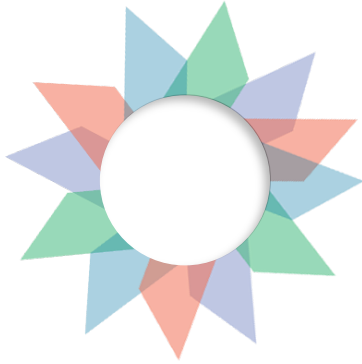


ALGORITHMS & DATA STRUCTURES

04 – Linked List



Learning Outcomes

Students are able to understand the concepts and methods of applying linked list



Outline

1

Dynamic Memory Allocation

2

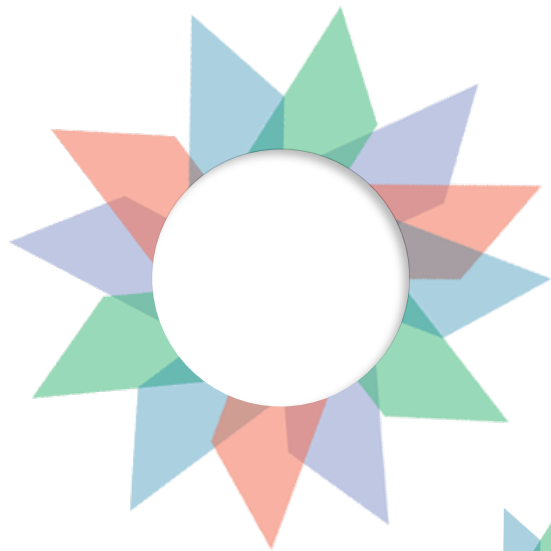
Single Linked List

3

Double Linked List

4

Circular Linked List



Dynamic Memory Allocation

- Stack vs Heap Memory
- Dynamic Memory Allocation

Stack vs Heap Memory



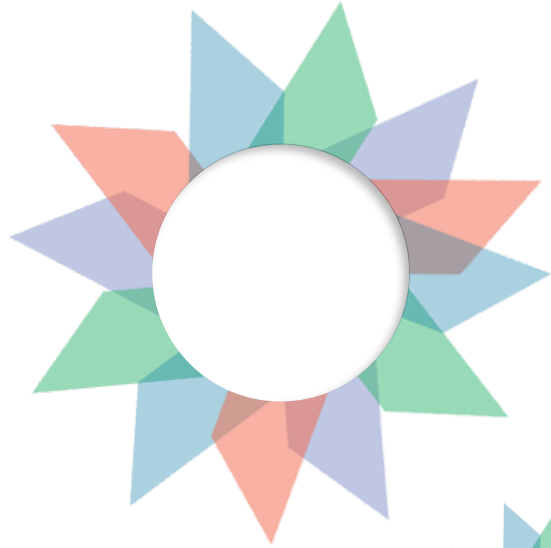
Stack Memory	Heap Memory
Static memory allocation	Dynamic memory allocation
Allocated when the program is compiled	Allocated at run time
Very fast access	Relatively slower access
The stack is always reserved in a LIFO order, the most recently reserved block is always the next block to be freed	A block can be allocated at any time and free it anytime
It really simple to keep track of the stack, freeing a block from the stack is nothing more than adjusting one pointer	It much more complex to keep track of which parts of the heap are allocated or free at any given time
Use the stack if you know exactly how much data you need to allocate before compile time and it is not too big	Use heap if you don't know exactly how much data you will need at runtime or if you need to allocate a lot of data

Dynamic Memory Allocation



- Creating and maintaining dynamic data structures that can grow and shrink as the program runs requires **dynamic memory allocation** - the ability for a program to obtain more memory space at execution time, and to release space no longer needed.
- Functions `malloc` and `free`, and operator `sizeof`, are essential to dynamic memory allocation.

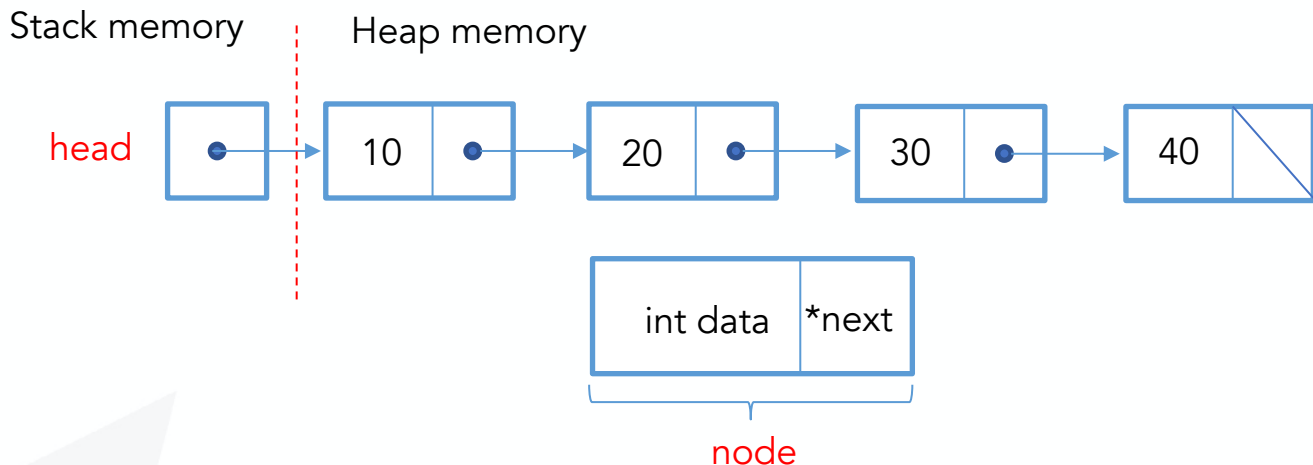
```
newPtr = malloc(sizeof(struct node));  
Free(newPtr);
```



Single Linked List

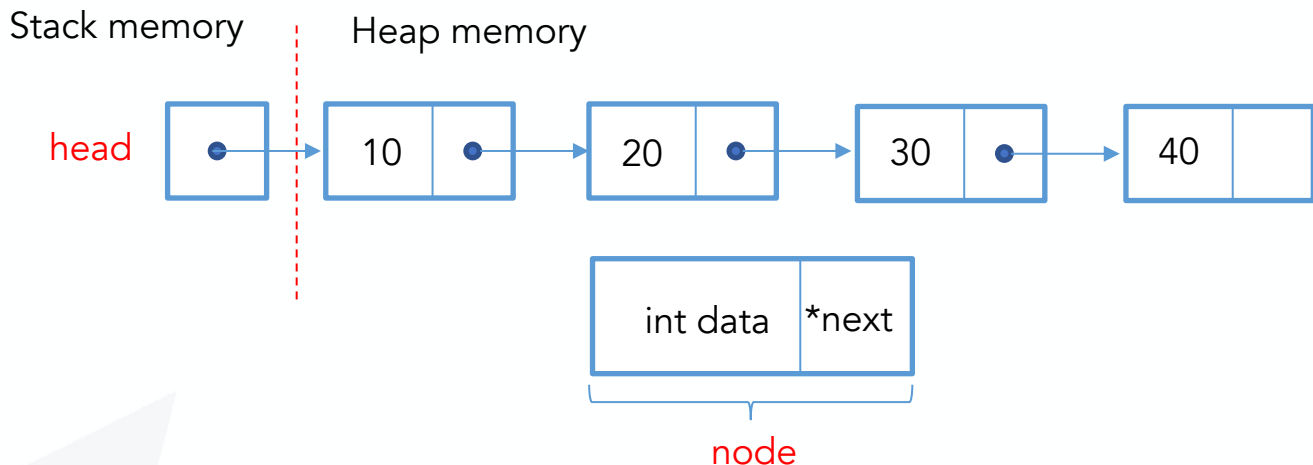
- Basic Terminologies
- Operations in Linked List

Basic Terminologies



- A linked list is a **linear collection of self-referential structures**, called **nodes**, connected by pointer **links**.
- Linked list is a data structure which in turn can be used to implement other data structures , such as stacks, queues, and their variations.

Basic Terminologies



- A linked list is accessed via a pointer to the first node of the list.
- Subsequent nodes are accessed via the link pointer member stored in each node.
- The **last node has a reference to null**.
- A node can contain data of any type including other structs.

Basic Terminologies



Array	Linked List
Linear data structure	Linear data structure
Number of data elements is predictable	Number of data elements is unpredictable
The size of an array created at compile time, however, can not be altered.	Dynamic, the length of a list can increase or decrease at execution time as necessary.
Arrays can become full	Linked lists become full only when the system has insufficient memory to satisfy dynamic storage allocation requests
Stores its members in consecutive memory locations	Does not store its nodes in consecutive memory locations
Insertions and deletions can be done at any point in the list in a constant time	Does not allow random access of data

Operations in Linked List



- **Declaration** – declare the data type and head of the list.
- **Insertion** – adds an element into the list.
- **Deletion** – deletes an element from the list.
- **Display** – displays the complete list.
- **Free** – returns the node to heap memory.

Declaration

```
typedef struct tnode {  
    int data;  
    struct tnode *next;  
};
```



node

```
int main()  
{  
    struct tnode *head, *node;  
    int bil;  
    head = NULL;  
    ...  
    return 0;  
}
```

head



node



Insertion

- Case 1: The new node is inserted at the beginning.
- Case 2: The new node is inserted at the end.
- Case 3: The new node is inserted after a given node.
- Case 4: The new node is inserted before a given node.

Insertion

Inserted at the beginning

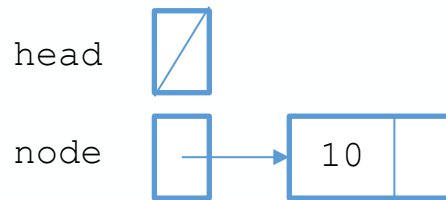
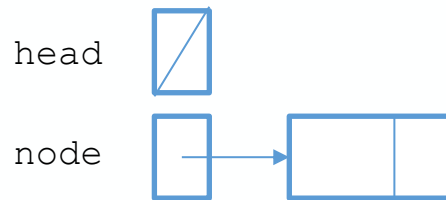
- Insert the first node

1. New node

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

2. Insert data to node

```
scanf("%d", &bil);  
node->data = bil;
```



Insertion

Inserted at the beginning

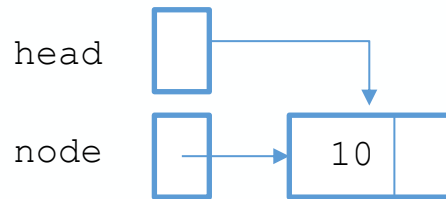
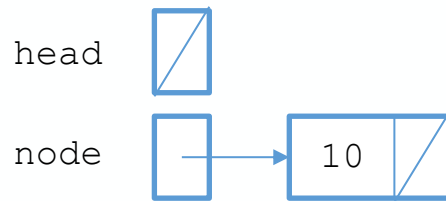
- Insert the first node

3. Set next to NULL

```
node->next = NULL;
```

4. Head refers to the first node

```
head = node;
```



Insertion

Inserted at the beginning

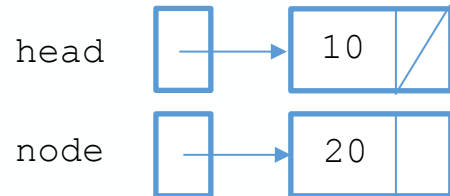
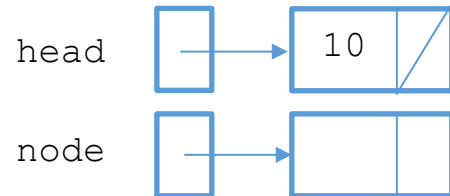
- Insert a new node

1. New node

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

2. Insert data to node

```
scanf("%d", &bil);  
node->data = bil;
```



Insertion

Inserted at the beginning

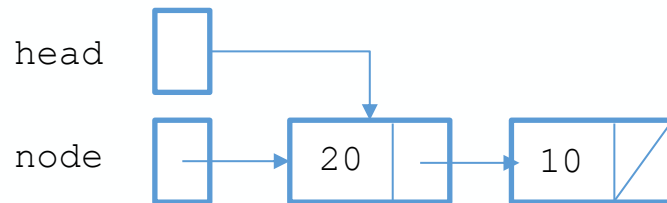
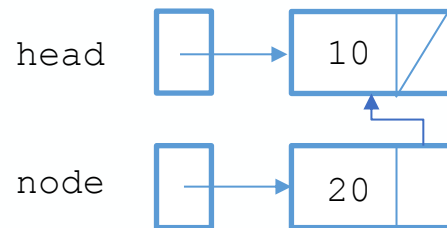
- Insert a new node

3. Pointer next of new node refers to head

```
node->next = head;
```

4. Head refers to the first node

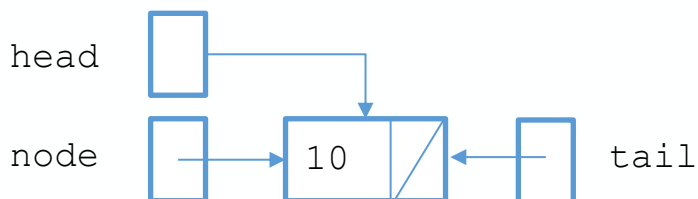
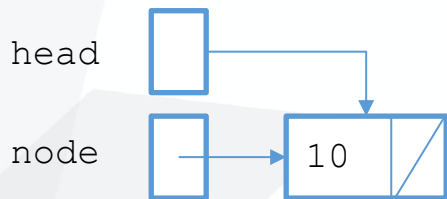
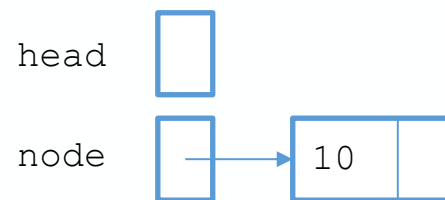
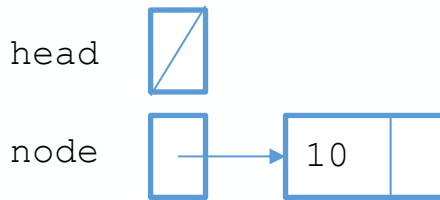
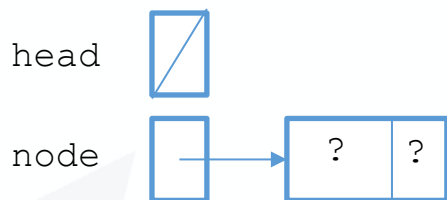
```
head = node;
```



Insertion

Inserted at the end

- Insert the first node: one pointer (tail) needed



Insertion

Inserted at the end

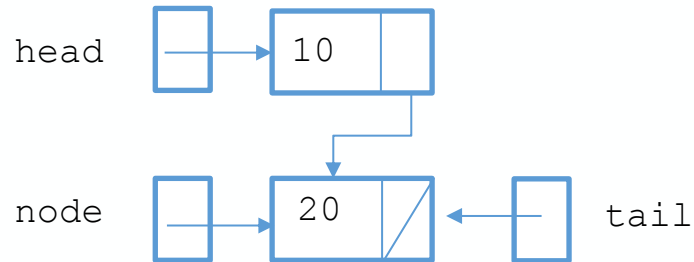
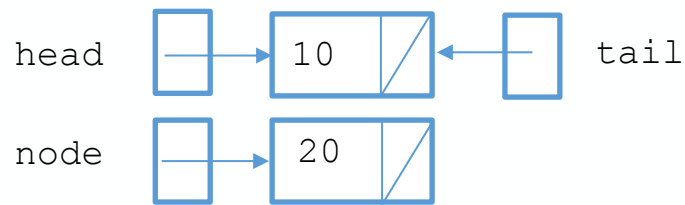
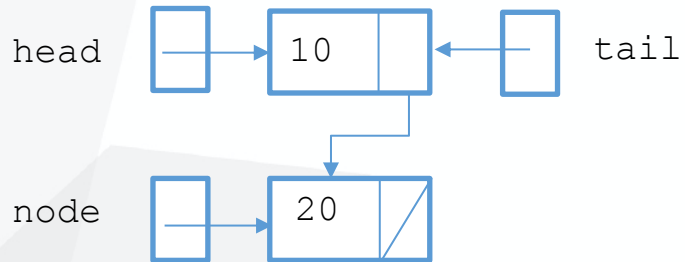
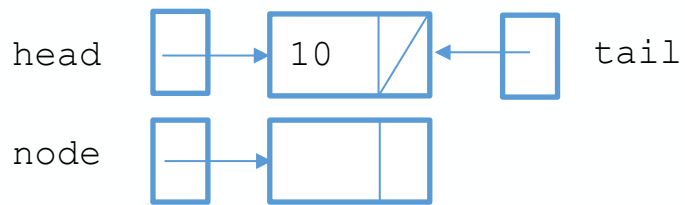
- Insert the first node: one pointer (tail) needed

```
node = (struct tnode*) malloc(sizeof(struct tnode));
scanf("%d", &bil);
node->data = bil;
node->next = NULL;
if(head == NULL)
{
    head = node;
    tail = node;
}
```

Insertion

Inserted at the end

- Insert a new node



Insertion

Inserted at the end

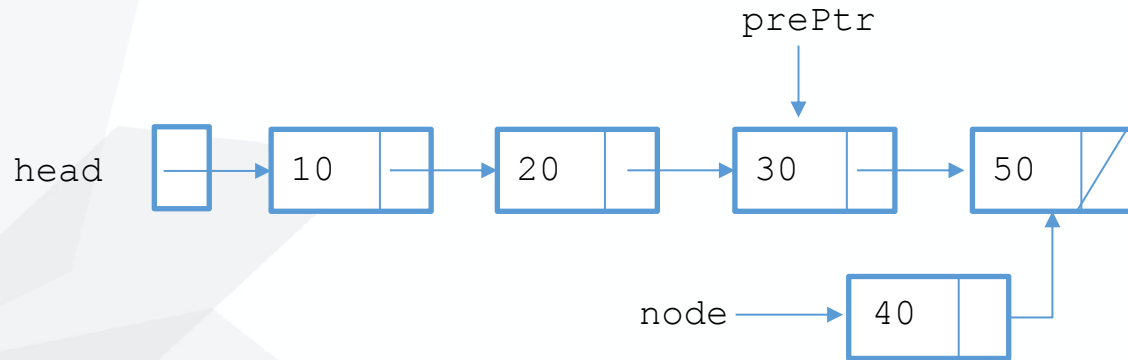
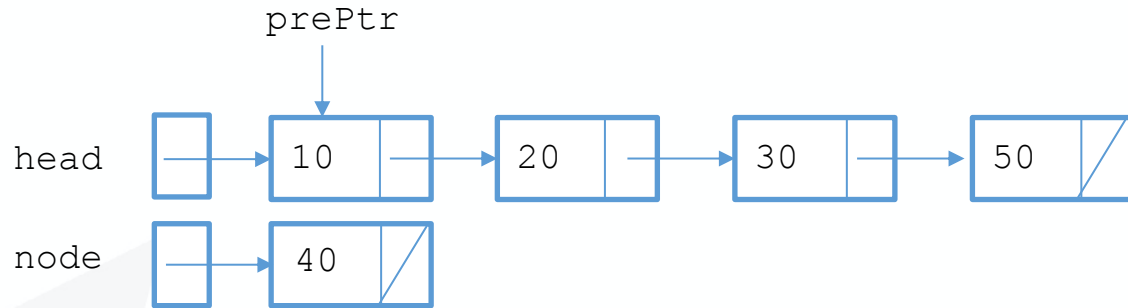
- Insert a new node

```
node = (struct tnode*) malloc(sizeof(struct tnode));
scanf("%d", &bil);
node->data = bil;
node->next = NULL;
if(head != NULL)
{
    tail->next = node;
    tail = node;
}
```

Insertion

Inserted after a given node

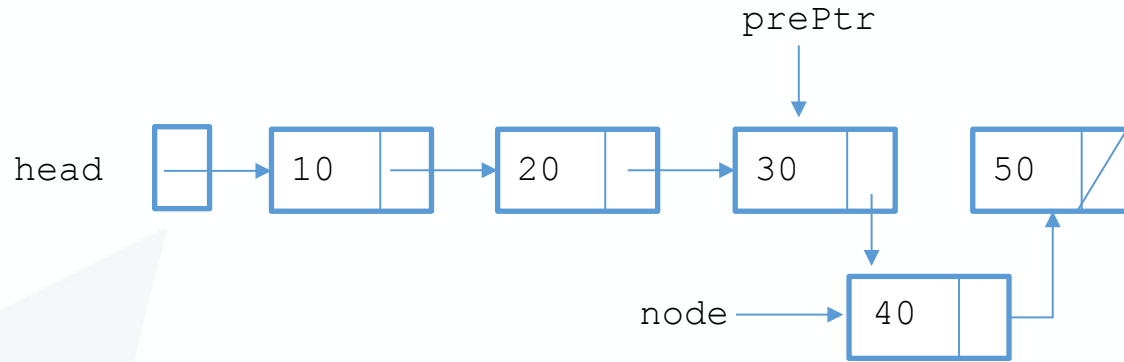
- Insert a new node



Insertion

Inserted after a given node

- Insert a new node



Insertion

Inserted after a given node

- Insert a new node

```
node = (struct tnode*) malloc(sizeof(struct tnode));
scanf("%d", &bil);
node->data = bil;
node->next = NULL;
preNode = 30;

prePtr = head;
while(prePtr->data != preNode)
{
    prePtr = prePtr->next;
}

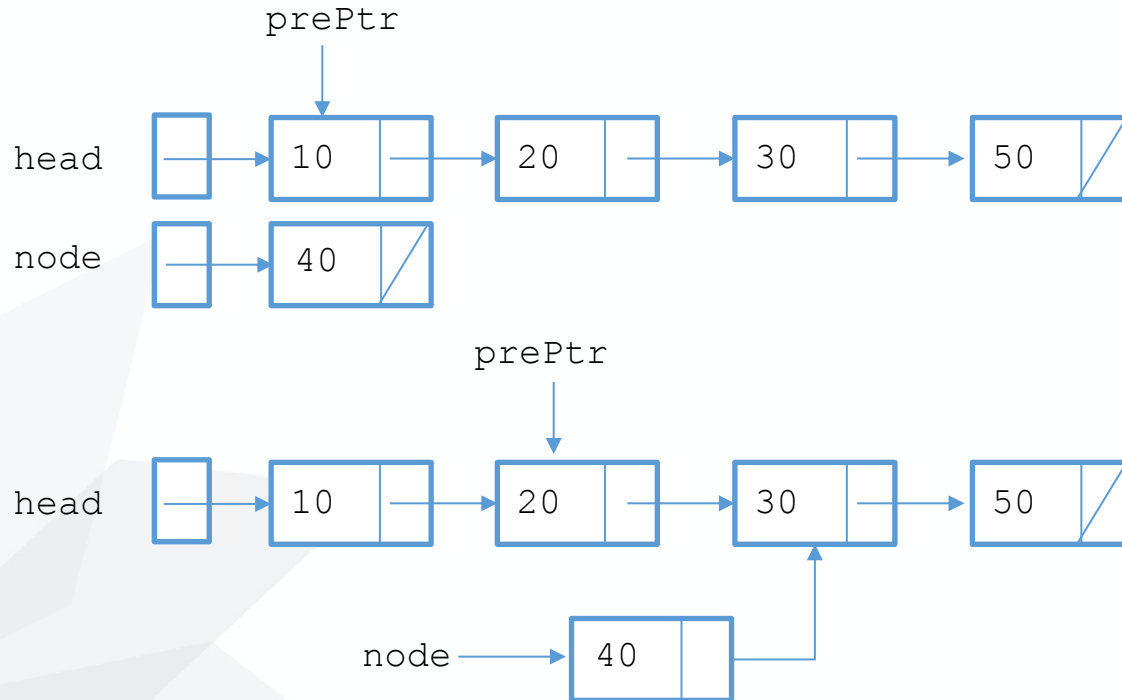
node->next = prePtr->next;
prePtr->next = node;
```




Insertion

Inserted before a given node

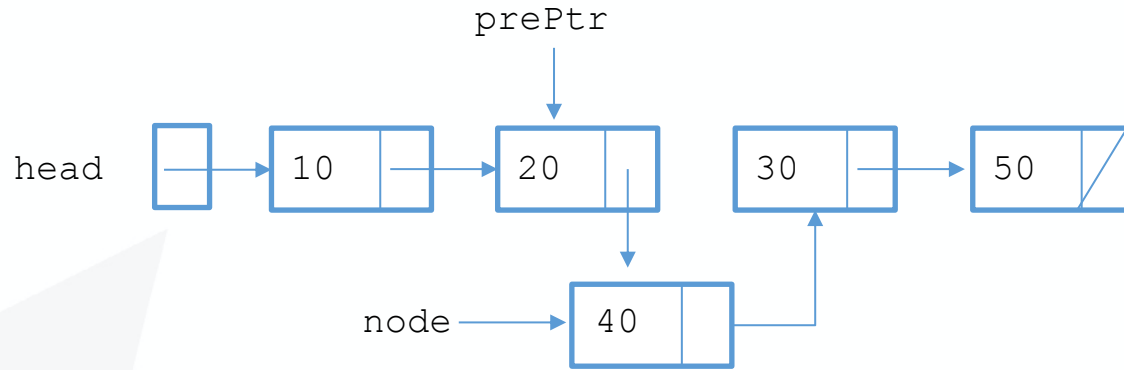
- Insert a new node



Insertion

Inserted before a given node

- Insert a new node



Insertion

Inserted before a given node

- Insert a new node

```
node = (struct tnode*) malloc(sizeof(struct tnode));
scanf("%d", &bil);
node->data = bil;
node->next = NULL;
preNode = 30;

prePtr = head;
while(prePtr->next->data != preNode)
{
    prePtr = prePtr->next;
}

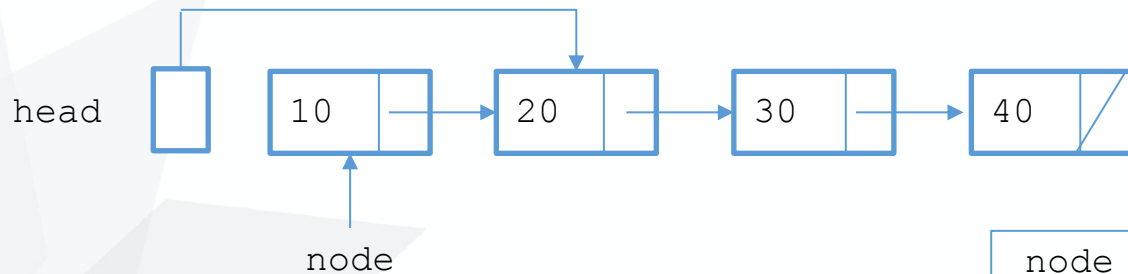
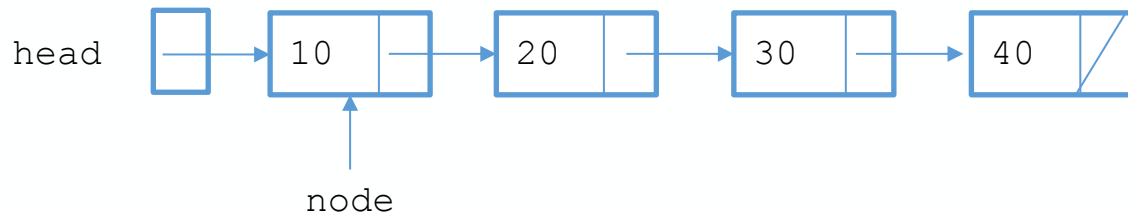
node->next = prePtr->next;
prePtr->next = node;
```

Deletion

- Case 1: The first node is deleted.
- Case 2: The last node is deleted.
- Case 3: The node after a given node is deleted.

Deletion

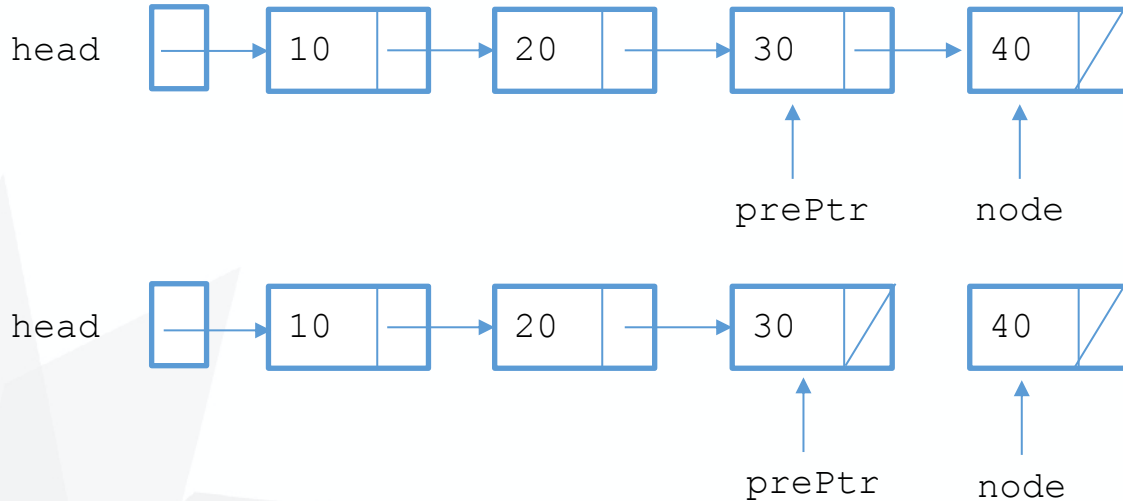
The first node is deleted



```
node = head;  
head = node->next;  
node->next = NULL; //(optional)  
free(node);
```

Deletion

The last node is deleted

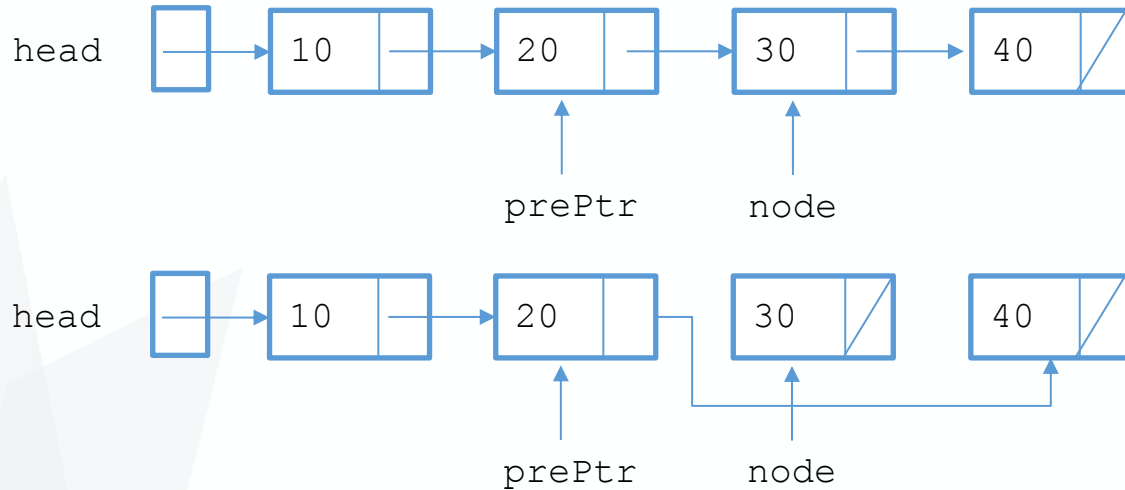


```
node = prePtr = head;
while(node->next != NULL)
    node = node->next;
while(prePtr->next != node)
    prePtr = prePtr->next;

prePtr->next = NULL;
free(node);
```

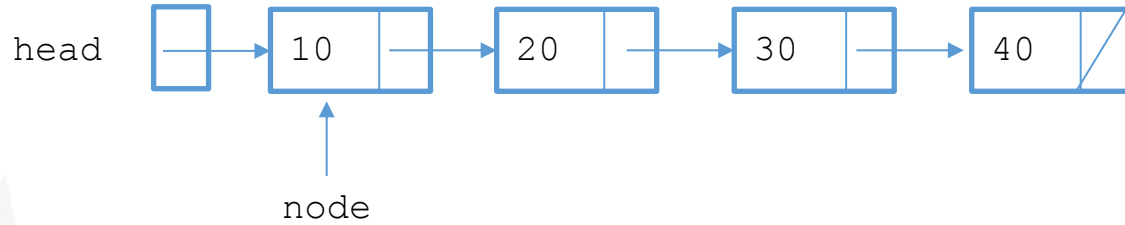
Deletion

The node after a given node is deleted



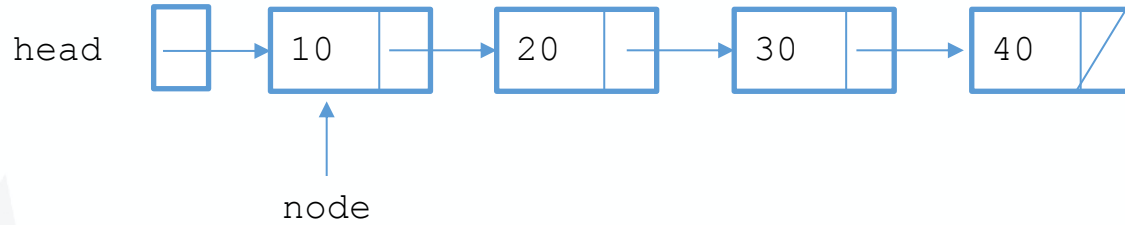
```
prePtr = head;
while(prePtr->data != 20)
    prePtr = prePtr->next;
node = prePtr->next;
prePtr->next = node->next;
node->next = NULL;
free(node);
```

Display



```
node = head;
while(node != NULL)
{
    printf("%d", node->data);
    node = node->next;
}
```


Free



```
while(head != NULL)
{
    node = head;
    head = head->next;
    free(node);
}
```

```
#include<stdio.h>
#include<malloc.h>

struct tnode
{
    int data;
    struct tnode *next;
};

int main()
{
    struct tnode *head, *node;
    int num;

    head = NULL;
    scanf("%d", &num);

    while(num != 0)
    {
        //insert at the beginning
        node = (struct tnode*) malloc(sizeof(struct tnode));
        node->data = num;
        node->next = NULL;

        if(head == NULL)
            node->next = NULL;
        else
            node->next = head;

        head = node;

        scanf("%d", &num);
    }
}
```


```
while(node != NULL)
{
    printf("%d ", node->data);
    node = node->next;
}

while(head != NULL)
{
    node = head;
    head = head->next;
    free(node);
}

return 0;
}
```

```
10
20
30
0
30 20 10
```

...Program finished with exit code 0
Press ENTER to exit console.



```
#include<stdio.h>
#include<malloc.h>

struct tnode
{
    int data;
    struct tnode *next;
};

int main()
{
    struct tnode *head, *node, *tail;
    int num;

    head = NULL;
    scanf("%d", &num);

    while(num != 0)
    {
        node = (struct tnode*) malloc(sizeof(struct tnode));
        node->data = num;
        node->next = NULL;

        if(head == NULL)
            head = node;
        else
            tail->next = node;

        tail = node;

        scanf("%d", &num);
    }
```

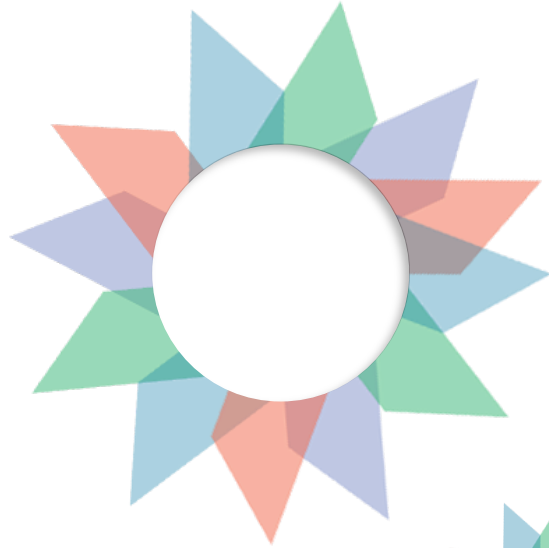
```
node = head;
while(node != NULL)
{
    printf("%d ", node->data);
    node = node->next;
}

while(head != NULL)
{
    node = head;
    head = head->next;
    free(node);
}

return 0;
}
```

```
10
20
30
0
10 20 30
```

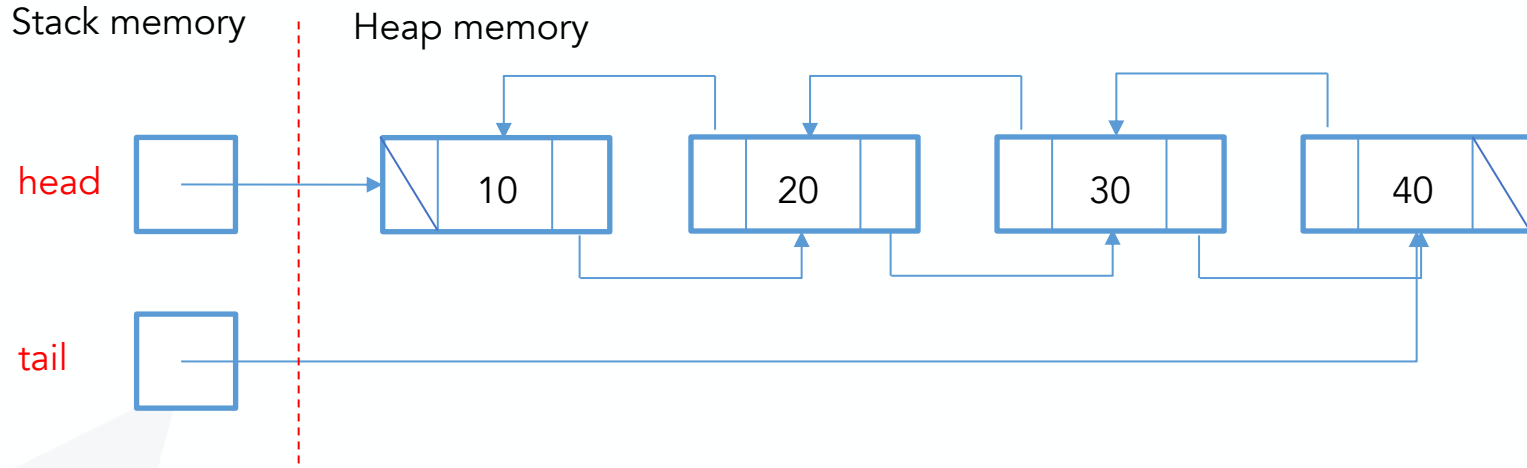
```
...Program finished with exit code 0
Press ENTER to exit console. 
```



Double Linked List

- Basic Terminologies
- Operations in Linked List

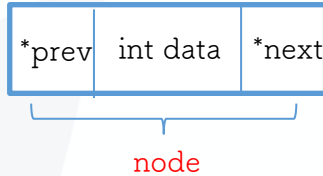
Basic Terminologies



node

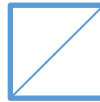
Declaration

```
typedef struct tnode {  
    int data;  
    struct tnode *next, *prev  
;  
};
```

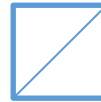


```
int main()  
{  
    struct tnode *head, *node;  
    int bil;  
    head = tail = NULL;  
    ...  
    return 0;  
}
```

head



node



Insertion

Inserted at the beginning

- Insert the first node

```
node = (struct tnode*)malloc(sizeof(struct tnode));
scanf("%d", &bil);
node->data = bil;

if(head == NULL)
{
    node->next = NULL;
    tail = node;
    head = node;
    head->prev = NULL;
}
```

Insertion

Inserted at the beginning

- Insert a new node

```
node = (struct tnode*)malloc(sizeof(struct tnode));
scanf("%d", &bil);
node->data = bil;

if(head != NULL)
{
    head->prev = node;
    node->next = head;
    head = node;
    head->prev = NULL;
}
```


Insertion

Inserted at the end

- Insert the first node

```
node = (struct tnode*)malloc(sizeof(struct tnode));
scanf("%d", &bil);
node->data = bil;

if(head == NULL)
{
    node->prev = NULL;
    head = node;
    tail = node;
    tail->next = NULL;
}
```

Insertion

Inserted at the end

- Insert the a new node

```
node = (struct tnode*)malloc(sizeof(struct tnode));
scanf("%d", &bil);
node->data = bil;

if(head != NULL)
{
    tail->next = node;
    node->prev = tail;
    tail = node;
    tail->next = NULL;
}
```

Deletion

The first node is deleted

```
node = head;  
head = node->next;  
head->prev = NULL;  
node->next = node->tail = NULL; //(optional)  
free(node);
```

Deletion

The last node is deleted

```
node = tail;  
tail = node->prev;  
tail->next = NULL;  
node->next = node->tail = NULL; //(optional)  
free(node);
```

Display



```
/* Head to Tail */
node = head;
while(node != NULL)
{
    printf("%d", node->data);
    node = node->next;
}
```

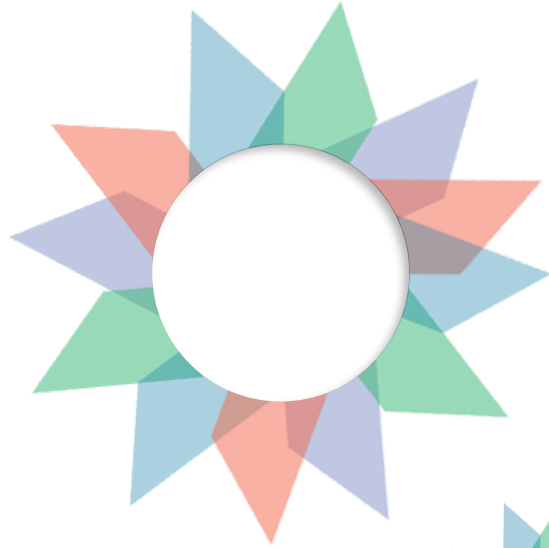
```
/* Tail to Head */
node = tail;
while(node != NULL)
{
    printf("%d", node->data);
    node = node->prev;
}
```

Free



```
/* Head to Tail */
while(head != NULL)
{
    node = head;
    head = head->next;
    free(node);
}
tail = NULL;
```

```
/* Tail to Head */
while(tail != NULL)
{
    node = tail;
    tail = tail->prev;
    free(node);
}
head = NULL;
```



Circular Linked List

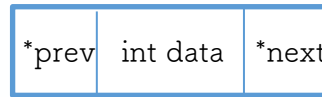
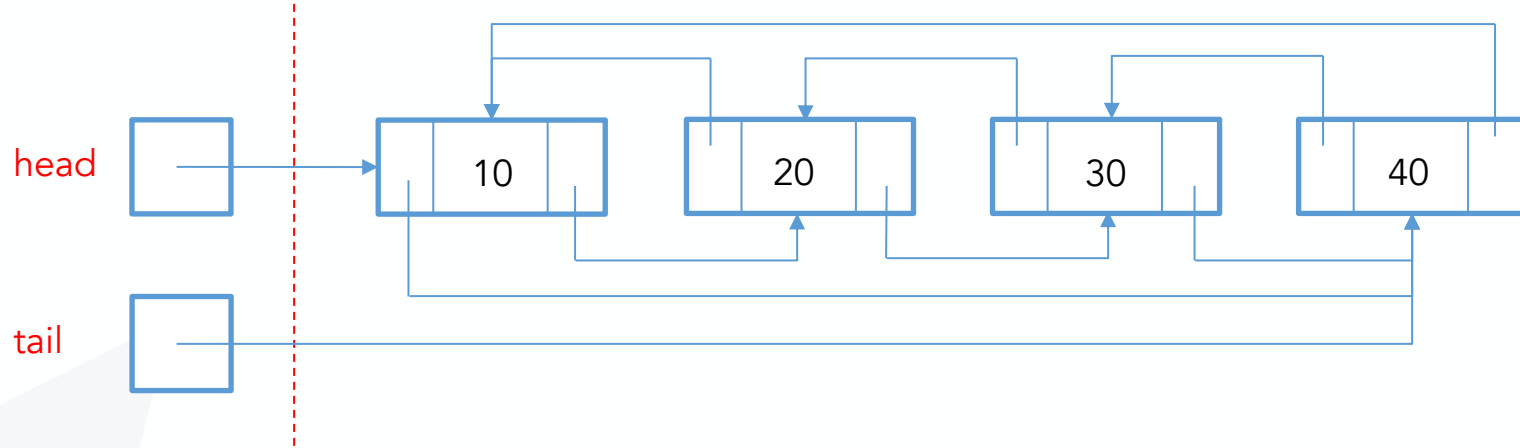
- Basic Terminologies

Basic Terminologies

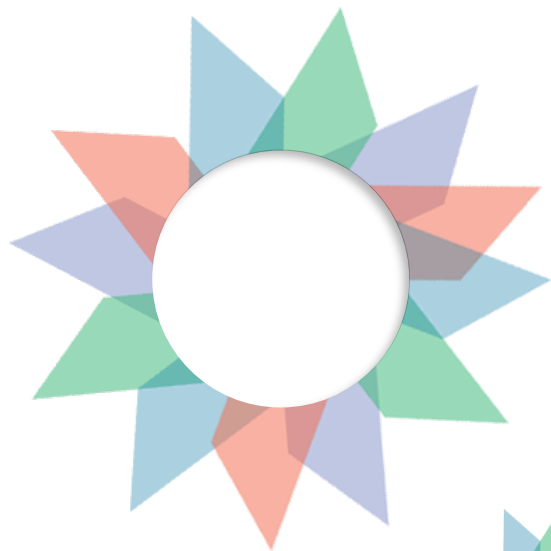


Stack memory

Heap memory



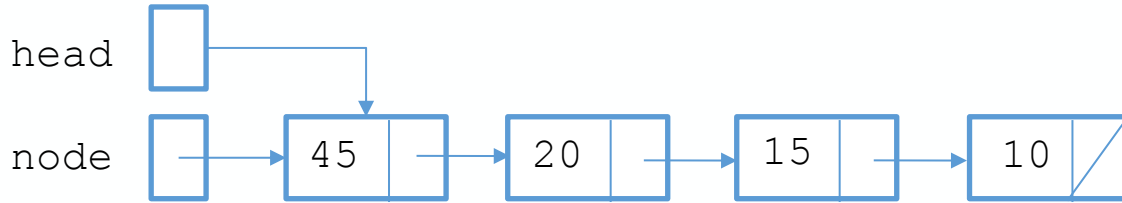
node



Practice



Practice 1



What is the output of the following code?

1. `printf("%d ", node->data);`
2. `printf("%d ", node->next->data);`
3. `printf("%d ", node->next->next->data);`

Practice 2



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

- Write a code to initialize SLL of 10 elements. The data is a number from 1 – 10.
- Write a code to sum up all the elements.
- Write a code to get the maximum value of all elements.

Practice 3



```
struct node
{
    int data;
    struct node *next, *prev;
};
```

- Write a code to initialize DLL of 10 elements. The data could be any positive number.
- Write a code that removes all nodes that have duplicate value.
- Write a code to delete a node of a given value.

References

1. Paul Deitel and Harvey Deitel. 2016. *C How to Program: with an introduction to C++*, 8th Edition, Global Edition. Great Britain: Pearson Education.
2. Reema Thareja, 2014. *Data Structures Using C*, 2nd Edition. India: Oxford University Press.

Next Outline

- Stack Representation
- Operations on Stack
- Stack Application
- Queue Representation
- Operations on Queue
- Queue Application



Thank you