CS 2124: DATA STRUCTURES
Spring 2024

6th Lecture

Topics: Advanced Linked Lists and Priority Queues

Quiz (22nd Feb, Thursday)

- Points: 5
- Date: 22nd Feb
- Quiz availability Time: 6:00 AM till End of day (11: 58 PM)
- Number of MCQ: 12 (Each MCQ Points vary based on difficulty)
- Once the Quiz starts students will have 24 Min to complete it.
- The quiz cannot be paused or stopped. It must be attempted in one sitting
- Kindly do not refresh or go back to the previous question (press back on the browser) as that is not allowed.
- One question will be visible at one time.
- Once you answer the question (submit) it cannot be changed

❖ Students with SDS approval only need to attempt the first 6 questions that they receive on Canvas.

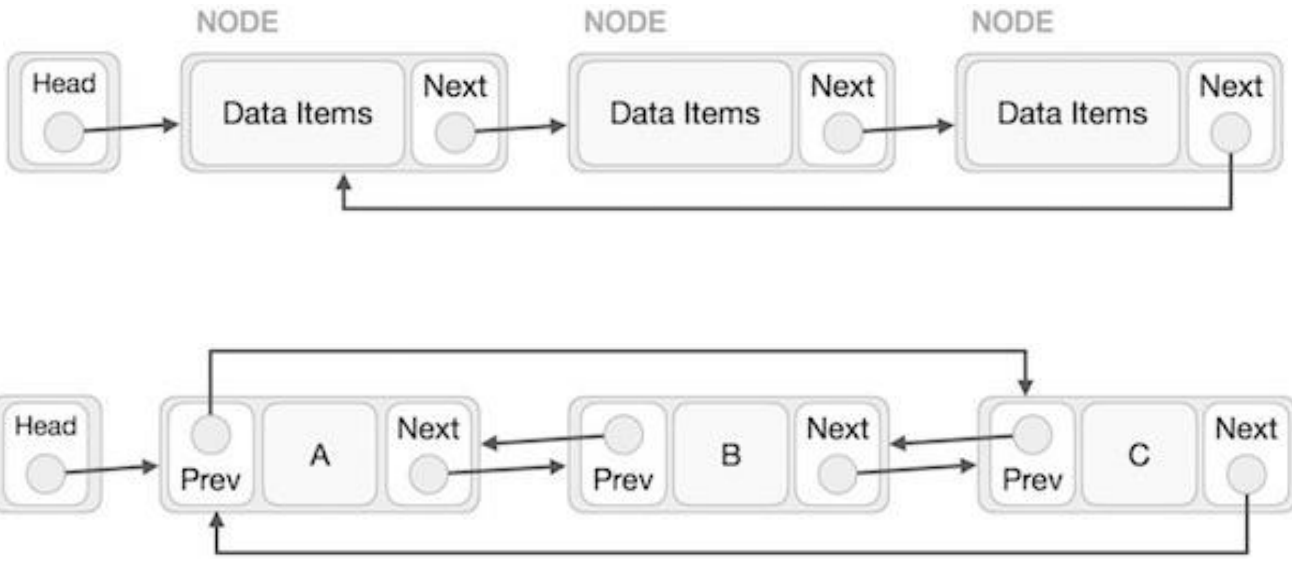
❖ After completion do email for grade scaling

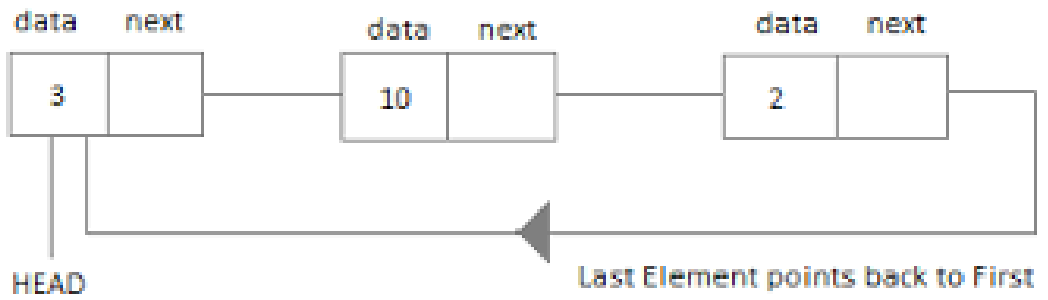
Topics

- Circular LLL (Linear Linked List)
- Singly LinkedList (L.L) as Circular L.L
  - Algorithm
  - Implementation
  - Operation - Insertion at Front
  - Operation - Insertion at Last
  - Operation - Delete First Element
  - Operation – Searching
  - Applications
- Dual LinkedList (DLL)
  - Memory Representation and Operations on a DLL
  - Insertion At Beginning Of DLL
  - Insertion At End Of DLL
  - Deletion At Beginning Of DLL
  - Deletion After A Specified Node
- Circular DLL
  - Implementation
- Priority Queues
  - Priority Queues – Characteristics
  - Priority Queues – Implementation

Circular LLL (Linear Linked List)

- Circular Linked List is a variation of Linked list in which the first element points to the last element and the last element points to the first element.
- Both **Singly Linked List** and **Doubly Linked List** can be made into a circular linked list.





Question:

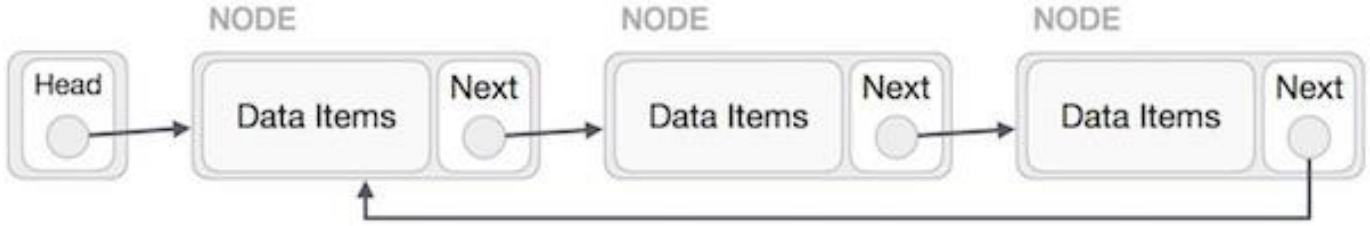
Which Approach is better ?

Separate Head pointer or imbedded?

Circular LLL (Linear Linked List)

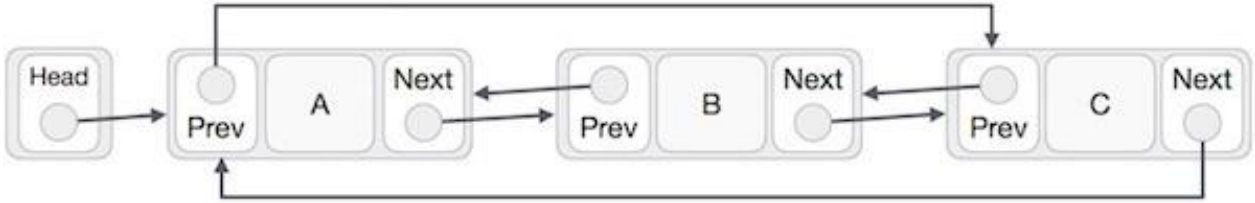
- **Singly** Linked List as Circular: In singly linked list, the next pointer of the last node points to the first node.

Figure 1 =



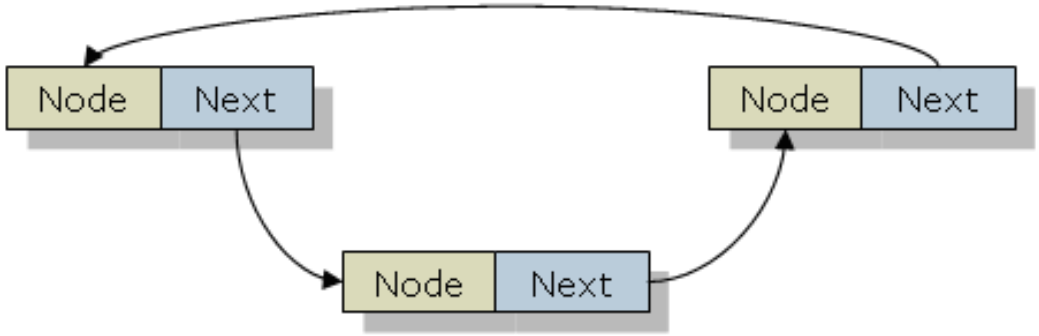
- **Doubly** Linked List as Circular: In doubly linked list, the next pointer of the last node points to the first node and the previous pointer of the first node points to the last node making the circular in both directions.

Figure 2 =



- The last link's next points to the first link of the list in both cases of singly as well as doubly linked list.
- The first link's previous points to the last of the list in case of doubly linked list.

Singly LinkedList as Circular (Algorithm)



1. Create a node
   I. Data
   II. Pointer to point next node
2. If first node, create node and place data with null as pointer (as it's the only node)
   I. Data
   II. Pointer = null
3. Else last node contains the reference of the new node and new node contains the reference of the previous/first node
   I. Data
   II. Last node pointer = next node address
   III. New node pointer = previous/first node address *(insertion at end)*

LinkedList

(Single LinkedList – Previous Lecture)

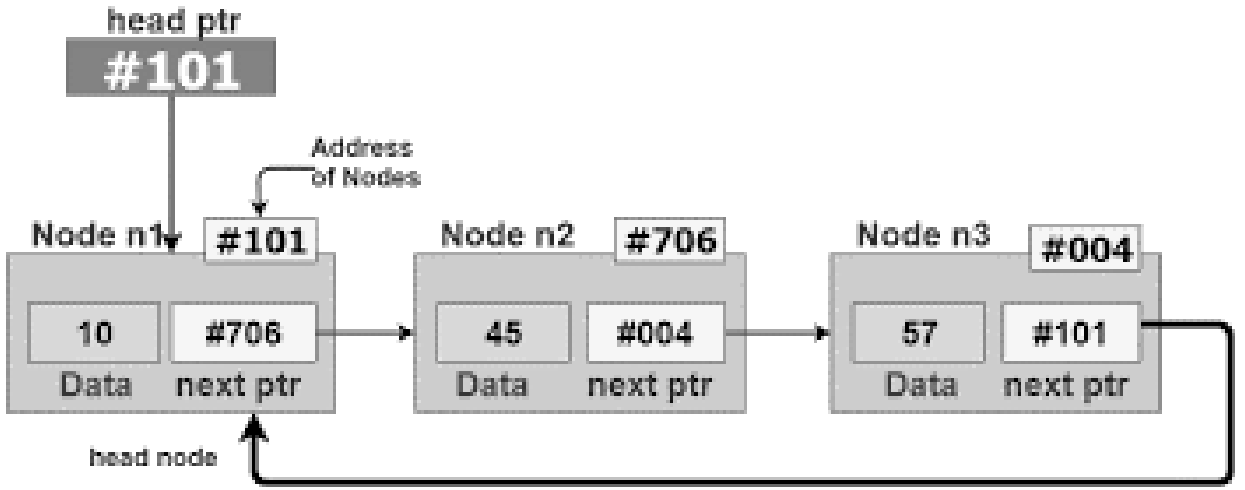
```c
#include <stdio.h>
#include <stdlib.h>
// Creating a node
struct node {
  int value;
  struct node *next;
};
```

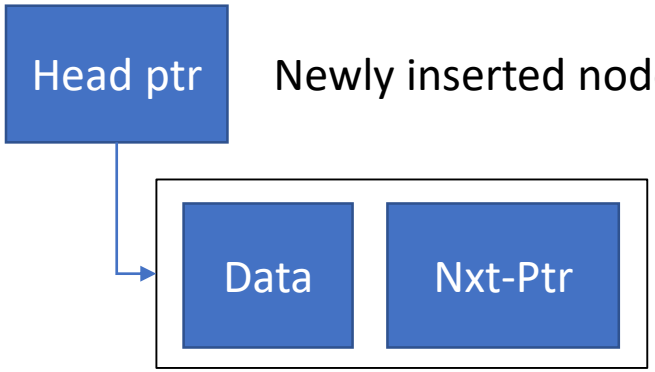
```c
// print the linked list value & address
void printLinkedlist(struct node *p) {
  while (p != NULL) {
    printf("Value: %d , Add: %p \n ", p->value, &p->value);
    p = p->next;
  } }
```

```c
int main() {
 // Initialize nodes
  struct node *head;
  struct node *one = NULL;
  struct node *two = NULL;
  struct node *three = NULL;
  struct node *four = NULL;
 // Allocate memory
  one = malloc(sizeof(struct node));
  two = malloc(sizeof(struct node));
  three = malloc(sizeof(struct node));
  four = malloc(sizeof(struct node));
// Assign value values
  one->value = 2;
  two->value = 0;
  three->value = 2;
  four->value = 4;
 // Connect nodes
  one->next = two;
  two->next = three;
  three->next = four;
  four->next = NULL;
 // printing node-value
  printf("<Name, abc123, SP24>\n");
  head = one;
  printLinkedlist(head);
}
```

LinkedList
(Single Circular LinkedList – Insertion at Front Part 1/3)

```c
#include <stdio.h>
#include <stdlib.h>
// Structure of a linked list node
struct node
{
    int info;
    struct node* next;
};
// Pointer to last node in the list
struct node* last = NULL;
// Function to insert a node in the starting of the list
void insertAtFront(int data)
{
    // Initialize a new node
    struct node* temp;
    temp = (struct node*)malloc(sizeof(struct node));
    // If the new node is the only node in the list
    if (last == NULL)
    {
        temp->info = data;
        temp->next = temp;
        last = temp;
        printf("\n Only Node Data = %d", data);
        printf("\n Only Node Add: %p", temp);
    }
```





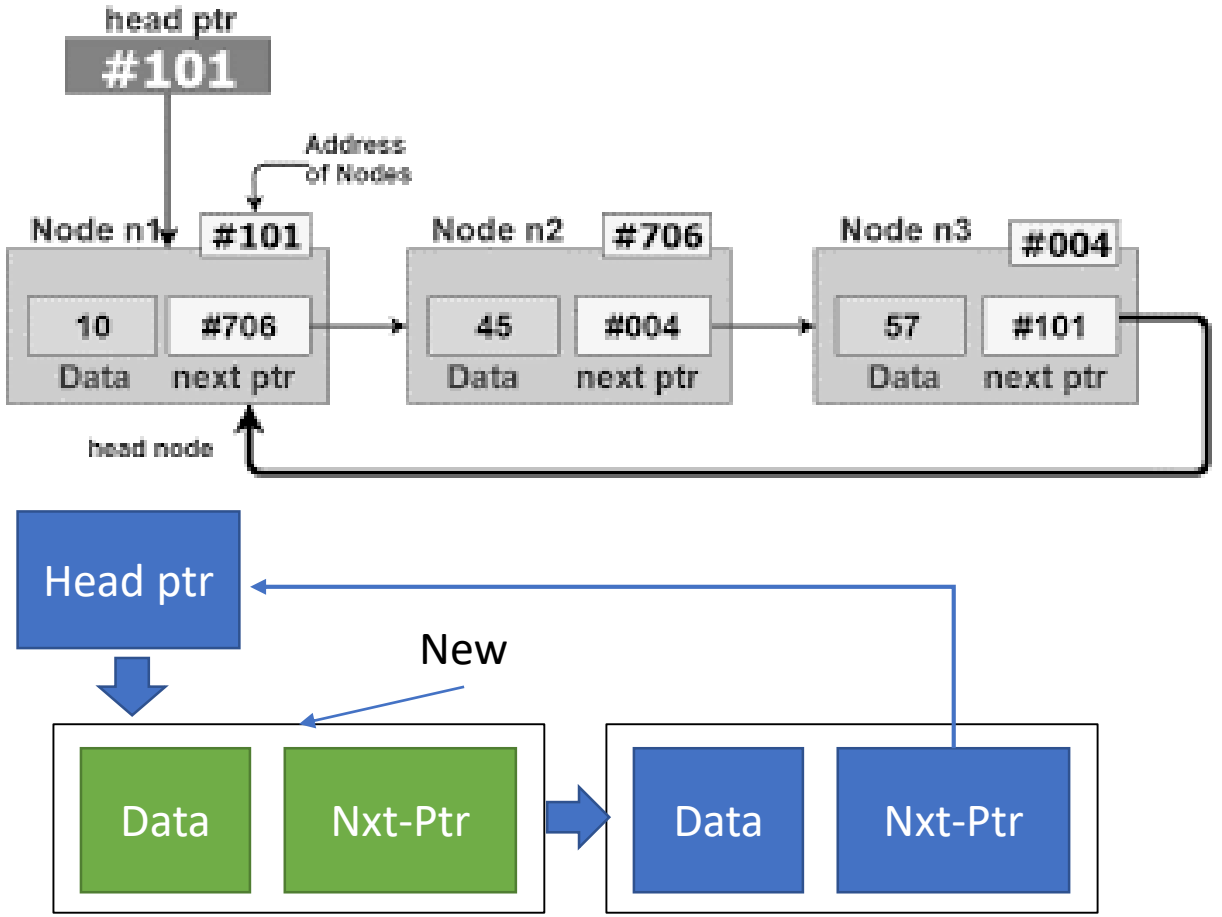
Newly inserted node is the first node

Continue >>

LinkedList

(Single Circular LinkedList – Insertion at Front Part 2/3)

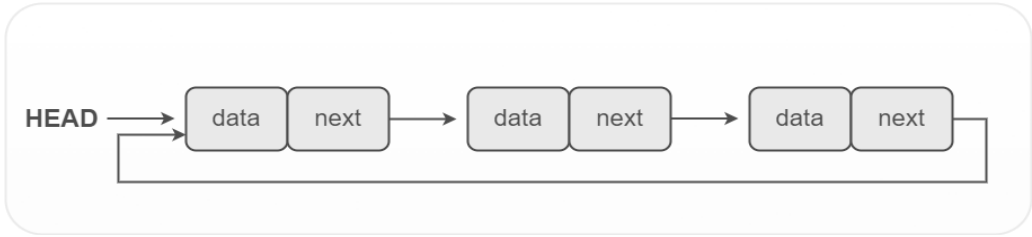
```
26  // Else last node contains the reference of the new node and
27  // new node contains the reference of the previous first node
28      else {
29          printf("\n  New Node Data = %d", data);
30          temp->info = data;
31          temp->next = last->next;
32          // last node now has reference of the new node temp
33          last->next = temp;
34          printf("\n Node Data = %d", data);
35          printf("\n Node Add: %p", temp); }}
36  void viewList() // Function to print the list
37  {   // If list is empty
38      if (last == NULL)
39          printf("\nList is empty\n");
40      else { // Else print the list
41          struct node* temp;
42          temp = last->next;
43  // While first node is not reached again, print,
44          // since the list is circular
45          do {
46              printf("\nData = %d", temp->info);
47              printf("\n Node Address: %p", temp);
48              temp = temp->next;
49              printf("\n Next Node Address: %p", temp);
50                      }
51                      while (temp != last->next);
52  } }
```



Continue >>

LinkedList

(Single Circular LinkedList – Insertion at Front) Part 3/3)




```
// Else last node contains the reference of the new node and
// new node contains the reference of the previous first node
    else {
        printf("\n  New Node Data = %d", data);
        temp->info = data;
        temp->next = last->next;
        // last node now has reference of the new node temp
        last->next = temp;
        printf("\n Node Data = %d", data);
        printf("\n Node Add: %p", temp); }}
void viewList() // Function to print the list
{   // If list is empty
    if (last == NULL)
        printf("\nList is empty\n");
    else { // Else print the list
        struct node* temp;
        temp = last->next;
// While first node is not reached again, print,
        // since the list is circular
        do {
            printf("\nData = %d", temp->info);
            printf("\n Node Address: %p", temp);
            temp = temp->next;
            printf("\n Next Node Address: %p", temp);
                    }
                    while (temp != last->next);
} }
```

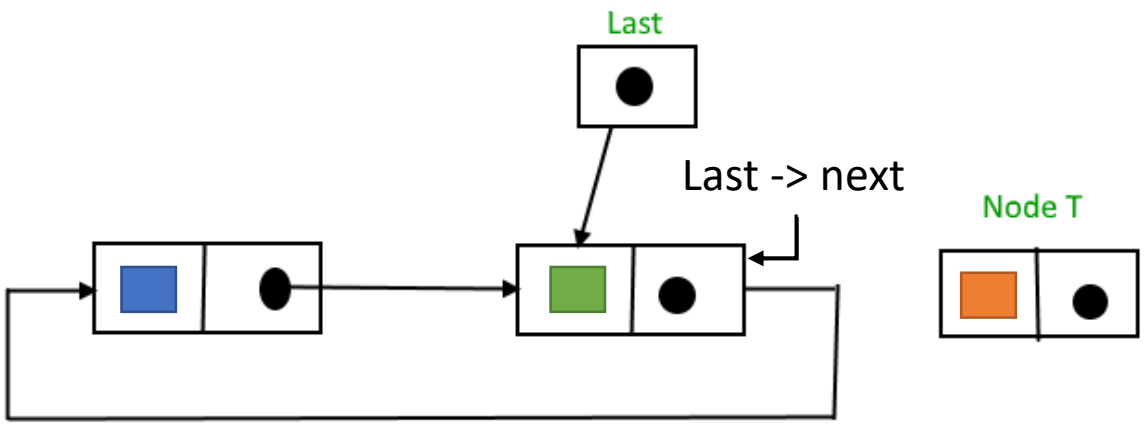
```
void insertAtFront(int data)
{
    // Initialize a new node
    struct node* temp;
    temp = (struct node*)malloc(sizeof(struct node));
    // If the new node is the only node in the list
    if (last == NULL)
    {
        temp->info = data;
        temp->next = temp;
        last = temp;
        printf("\n Only Node Data = %d", data);
        printf("\n Only Node Add: %p", temp);
    }
```

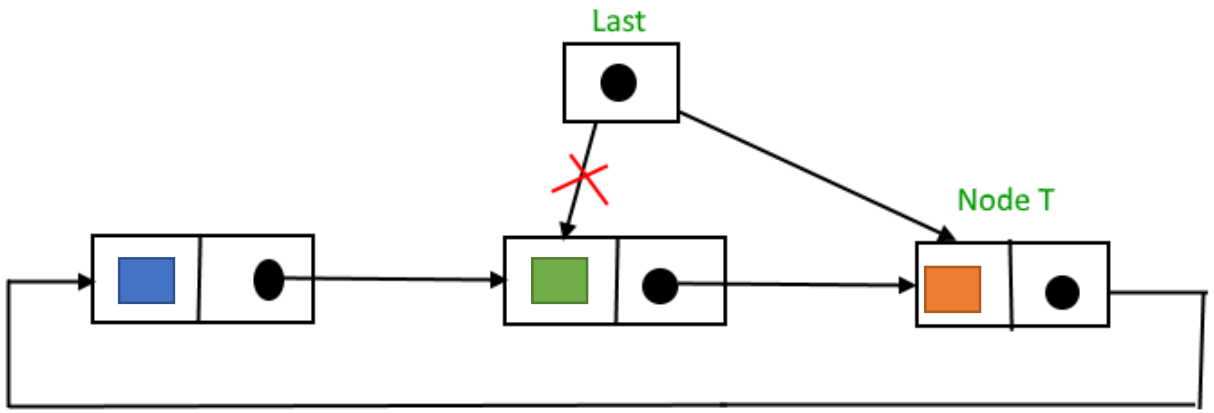
```
// Driver Code
int main()
{
    // Function Call
    insertAtFront(10);
    insertAtFront(20);
    insertAtFront(30);
    // Print list
    viewList();
    return 0;
}
```

Continue >>

LinkedList (Single Circular LinkedList – Insertion at Last)

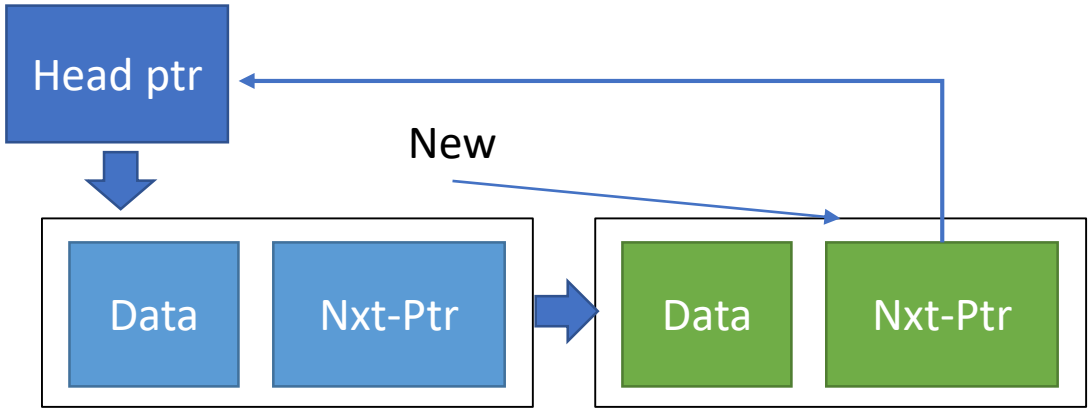
- **Insertion at the end of the list**
  - To insert a node at the end of the list, follow these steps:
    - Create a node, say T
    - Make T -> next = last -> next
    - last -> next = T
    - last = T





LinkedList (Single Circular LinkedList – Insertion at Last – Part 1/3)

```
12  void addatlast(int data)
13  {
14      // Initialize a new node
15      struct node* temp;
16      temp = (struct node*)malloc(sizeof(struct node));
17      // If the new node is the only node in the list
18      if (last == NULL)
19      {
20          temp->info = data;
21          temp->next = temp;
22          last = temp;
23          printf("\n Only Node Data = %d", data);
24          printf("\n Only Node Add: %p", temp);
25      }
26      // Else last node contains the reference of the new node and
27      // new node contains the reference of the previous first node
28      else {
29          printf("\n New Node Data = %d", data);
30          temp->info = data;
31          temp->next = last->next;
32          // last node now has reference of the new node temp
33          last->next = temp;
34          last = temp; //Add at the last
35          printf("\n Node Data = %d", data);
36          printf("\n Node Add: %p", temp);
37      }
38  }
```



Using the same base code as Insertion at beginning.
Only replacing the node add function in **ELSE statement**

LinkedList (Single Circular LinkedList – Insertion at Last – Part 2/3)

```c
void addatlast(int data)
{
    // Initialize a new node
    struct node* temp;
    temp = (struct node*)malloc(sizeof(struct node));
    // If the new node is the only node in the list
    if (last == NULL)
    {
        temp->info = data;
        temp->next = temp;
        last = temp;
        printf("\n Only Node Data = %d", data);
        printf("\n Only Node Add: %p", temp);
    }
    // Else last node contains the reference of the new node and
    // new node contains the reference of the previous first node
    else {
        printf("\n New Node Data = %d", data);
        temp->info = data;
        temp->next = last->next;
        // last node now has reference of the new node temp
        last->next = temp;
        last = temp; //Add at the last
        printf("\n Node Data = %d", data);
        printf("\n Node Add: %p", temp);
    }
}
```

```c
// Function to print the list
void viewList()
{
    // If list is empty
    if (last == NULL)
        printf("\nList is empty\n");
    // Else print the list
    else {
        struct node* temp;
        temp = last->next;
        // While first node is not
        // reached again, print,
        // since the list is circular
        do {
        printf("\nData = %d", temp->info);
        printf("\n Node Address: %p", temp);
        temp = temp->next;
        printf("\n Next Node Address: %p", temp);
                }
        while (temp != last->next);
        }
}
```

LinkedList (Single Circular LinkedList – Insertion at Last – Part 3/3)

```
39  // Function to print the list
40  void viewList()
41  {
42      // If list is empty
43      if (last == NULL)
44          printf("\nList is empty\n");
45      // Else print the list
46      else {
47          struct node* temp;
48          temp = last->next;
49          // While first node is not
50          // reached again, print,
51          // since the list is circular
52          do {
53          printf("\nData = %d", temp->info);
54          printf("\n Node Address: %p", temp);
55          temp = temp->next;
56          printf("\n Next Node Address: %p", temp);
57                  }
58          while (temp != last->next);
59          }
60  }
```

```
62  int main()
63  {
64      // Function Call
65      addatlast(10);
66      addatlast(20);
67      addatlast(30);
68      // Print list
69      viewList();
70      return 0;
71  }
```

LinkedList (Single Circular LinkedList – Insertion in L.L)

- Insert a new node in between the list.
  - If the list is empty, both head and tail will point to new node.
  - If the list is not empty, then.
    - We will define two nodes
      - Current (current will point to the node previous to temp), and
      - Temp (temp will point to head).
    - We iterate through the list till desired-point is reached (i.e. incrementing temp to temp.next)
      - Then, insert the new node in between current and temp.
    - Current -> next node will be new and the new -> next node will be temp.

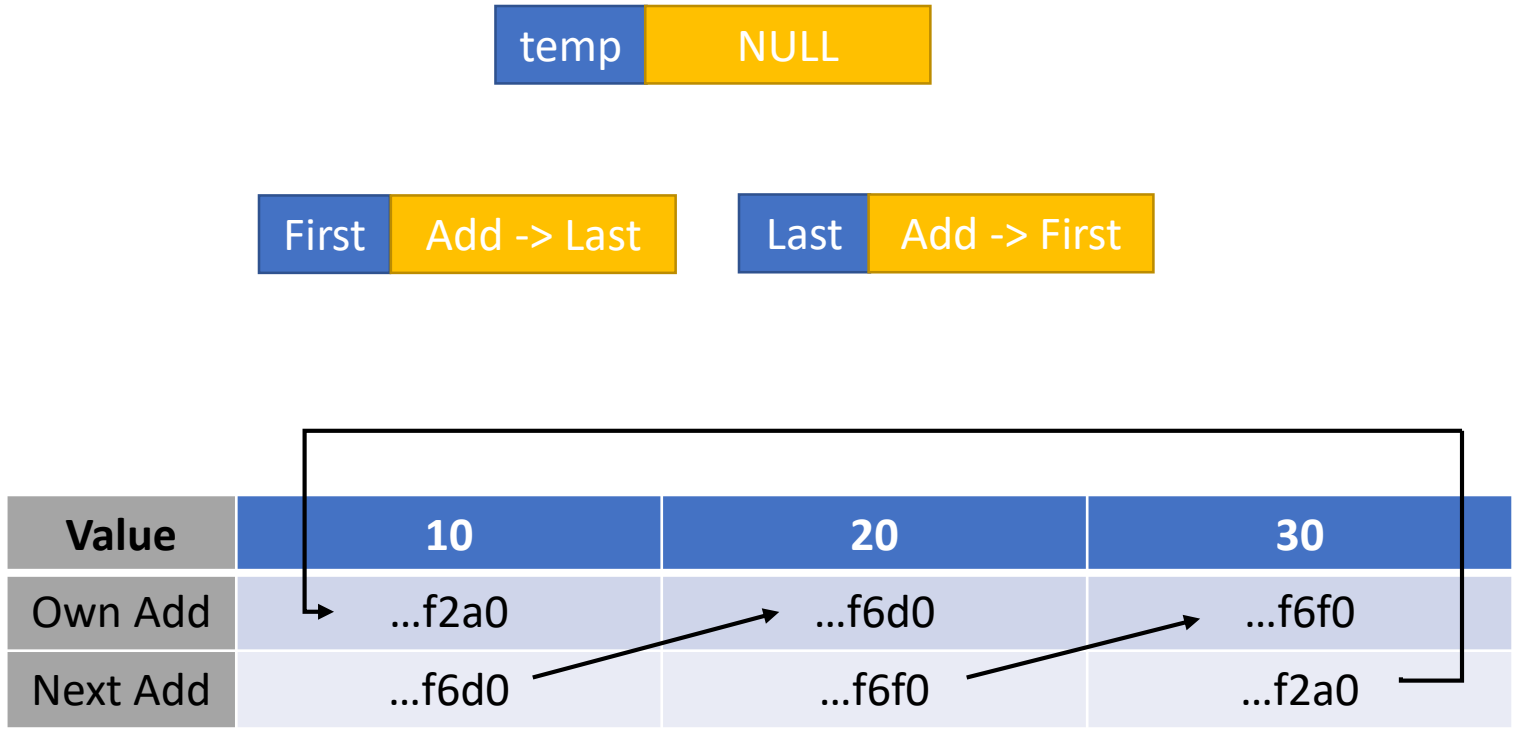
Try to code this by your self

Additional Source: Link

LinkedList

(Single Circular LinkedList – Insertion at Last and Delete First Element)

```
40  void deletefirst()
41  {
42      struct node* temp;
43      // If list is empty
44      if (last == NULL)
45          printf("\nList is empty.\n");
46       // Else last node now contains
47      // reference of the second node
48      // in the list because the
49      // list is circular
50      else {
51          temp = last->next;
52          last->next = temp->next;
53          free(temp);
54      }
55  }
```



Using the same base code as Insertion at beginning & end.

LinkedList (Single Circular L.L – Insertion at Last and Delete First Element) – Part 1/3

```c
#include <stdio.h>
#include <stdlib.h>
// Structure of a linked list node
struct node
{
    int info;
    struct node* next;
};
// Pointer to last node in the list
struct node* last = NULL;
// Function to insert a node in the starting of the list
void addatlast(int data)
{
    // Initialize a new node
    struct node* temp;
    temp = (struct node*)malloc(sizeof(struct node));
    // If the new node is the only node in the list
    if (last == NULL)
    {
        temp->info = data;
        temp->next = temp;
        last = temp;
        printf("\n Only Node Data = %d", data);
        printf("\n Only Node Address: %p", temp);
    }
    // Else last node contains the reference of the new node and
    // new node contains the reference of the previous first node
    else {
        temp->info = data;
        temp->next = last->next;
        // last node now has reference of the new node temp
        last->next = temp;
        last = temp; //Add at the last
        printf("\n New Node Data = %d", data);
        printf("\n New Node Add: %p", temp);
    }
}
// Function to delete the first
// element of the list
void deletefirst()
{
    struct node* temp;
    // If list is empty
    if (last == NULL)
        printf("\nList is empty.\n");
     // Else last node now contains
    // reference of the second node
    // in the list because the
    // list is circular
    else {
        temp = last->next;
        last->next = temp->next;
        free(temp);
    }
}
```

LinkedList (Single Circular L.L – Insertion at Last and Delete First Element) – Part 2/3

```
57  void viewList()
58  {
59      // If list is empty
60      if (last == NULL)
61          printf("\nList is empty\n");
62      // Else print the list
63      else {
64          struct node* temp;
65          temp = last->next;
66          // While first node is not
67          // reached again, print,
68          // since the list is circular
69          do {
70              printf("\nData = %d", temp->info);
71              printf("\n Node Address: %p", temp);
72              temp = temp->next;
73              printf("\n Next Node Address: %p", temp);
74                      }
75                      while (temp != last->next);
76                      }
77  }
78  // Driver Code
79  int main()
80  {   // Function Call
81      addatlast(10);
82      addatlast(20);
83      addatlast(30);
84      viewList(); // Print list
85      deletefirst(); // Function Call
86      printf("\n\nAfter deletion:\n");
87      viewList();
88  }
```

LinkedList (Single Circular LinkedList – Insertion at Last and Delete Last Element)

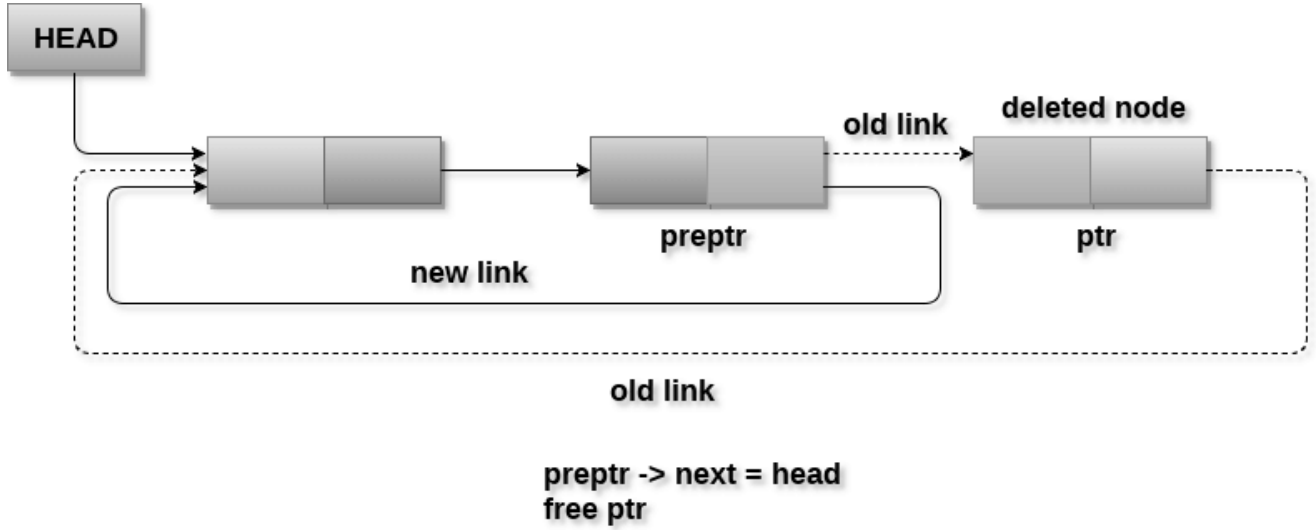
- **Scenario** (the list contains single element)
- If the list contains single node then, the condition head → next == head will become true. In this case, we need to delete the entire list and make the head pointer free. This will be done by using the following statements.

```
if(head->next == head)
{
    head = NULL;
    free(head);
}
```

LinkedList (Single Circular LinkedList – Insertion at Last and Delete Last Element)

- **Scenario** (the list contains more than one element)
- If the list contains more than one element, then in order to delete the last element, we need to reach the last node.
- We also need to keep track of the second last node of the list. For this purpose, the two pointers ptr and preptr are defined. The following sequence of code is used for this purpose.

```
ptr = head;
        while(ptr ->next != head)
        {
            preptr=ptr;
            ptr = ptr->next;
        }
preptr->next = ptr -> next;
        free(ptr);
```



- We need to make just one more pointer adjustment. We need to make the next pointer of preptr point to the next of ptr (i.e. head) and then make pointer ptr free.

LinkedList (Single Circular LinkedList – Searching)

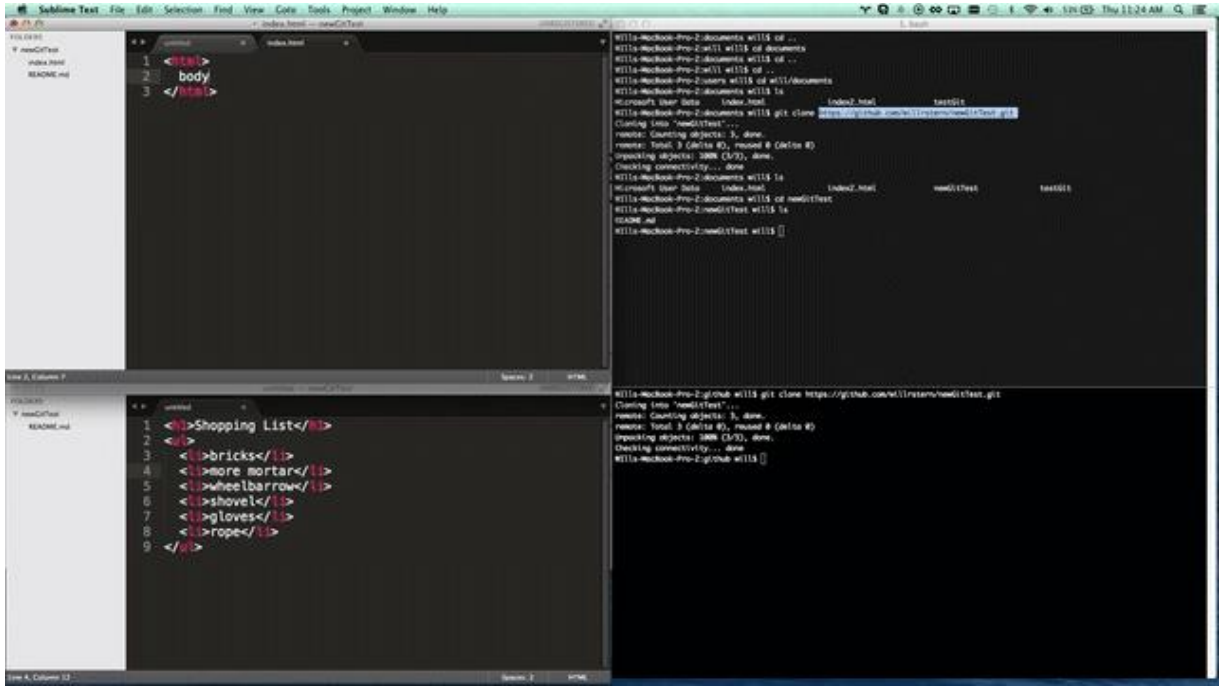
- Searching in circular singly linked list needs traversing across the list.
- The item which is to be searched in the list is matched with each node data of the list once and if the match found then the location of that item is returned otherwise -1 is returned.

1. Step 1: SET PTR = HEAD
2. Step 2: Set I = 0
3. STEP 3: IF PTR = NULL
    1. WRITE "EMPTY LIST"
    2. GOTO STEP 8
    3. END OF IF
4. STEP 4: IF HEAD → DATA = ITEM
    1. WRITE i+1 RETURN [END OF IF]
5. STEP 5: REPEAT STEP 5 TO 7 UNTIL PTR->next != head
6. STEP 6: if ptr → data = item
    1. write i+1
    2. RETURN
    3. End of IF
7. STEP 7: I = I + 1
8. STEP 8: PTR = PTR → NEXT [END OF LOOP]
9. STEP 9: EXIT

Circular LLL (Linear Linked List)

- Circular Linked List Applications:

1. It is used in multiplayer games to give a chance to each player to play the game.
2. Multiple running applications can be placed in a circular linked list on an operating system. The OS keeps on iterating over these applications.



LinkedList (Dual LinkedList)

Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence. Therefore, in a doubly linked list, a node consists of three parts: node data, pointer to the next node in sequence (next pointer) , pointer to the previous node (previous pointer).

Node

Doubly Linked List

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

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

```
struct DLLNode {
int info;
struct node *left, *right;
};
```

The prev part of the first node and the next part of the last node will always contain null indicating end in each direction.

LinkedList (Dual LinkedList)

```c
// Linked list implementation in C
#include <stdio.h>
#include <stdlib.h>
// Creating a node
struct node {
  int value;
  struct node *next;
  struct node *pre;
};
int main() {
  // Initialize nodes
  struct node *head;
  struct node *one = NULL;
  struct node *two = NULL;
  struct node *three = NULL;
  struct node *four = NULL;
  // Allocate memory
  one = malloc(sizeof(struct node));
  two = malloc(sizeof(struct node));
  three = malloc(sizeof(struct node));
  four = malloc(sizeof(struct node));
  // Assign value values
  one->value = 2;
  two->value = 0;
  three->value = 2;
  four->value = 3;
```

```c
  // Connect nodes
  one->next = two;
  one->pre = NULL;
  two->next = three;
  two->pre = one;
  three->next = four;
  three->pre = two;
  four->next = NULL;
  four->pre = three;
  printf("%d, %p \n", one->value, &one->value);
    printf("Last: %p \n", one->pre);
    printf("Next: %p \n", one->next);
    printf("%d, %p \n", two->value, &two->value);
    printf("Last: %p \n", two->pre);
    printf("Next: %p \n", two->next);
    printf("%d, %p \n", three->value, &three->value);
    printf("Last: %p \n", three->pre);
    printf("Next: %p \n", three->next);
    printf("%d, %p \n", four->value, &four->value);
    printf("Last: %p \n", four->pre);
    printf("Next: %p \n", four->next);
}
```

Is this going to work ??

LinkedList (Dual LinkedList)

```
36    printf("%d, %p \n", one->value, &one->value);
37      printf("Last: %p \n", one->pre);
38      printf("Next: %p \n", one->next);
39      printf("%d, %p \n", two->value, &two->value);
40      printf("Last: %p \n", two->pre);
41      printf("Next: %p \n", two->next);
42      printf("%d, %p \n", three->value, &three->value);
43      printf("Last: %p \n", three->pre);
44      printf("Next: %p \n", three->next);
45      printf("%d, %p \n", four->value, &four->value);
46      printf("Last: %p \n", four->pre);
47      printf("Next: %p \n", four->next);
48  }
```