Programming and Algorithms

COMP1038.PGA

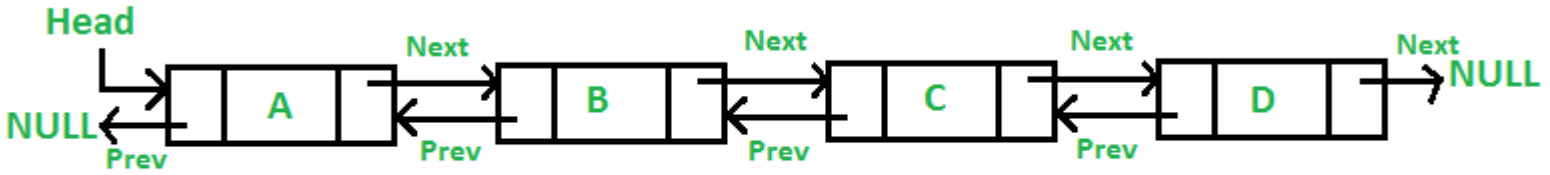
Session 16: Doubly linked list

Outline

- Doubly linked list
  - Introduction
  - Creation
  - Insertion
  - Deletion
  - Printing

Introduction

- Pointer to next element as with singly-linked list.
- Pointer to previous element as well.
- Can access previous element just by using previous pointer.
- More efficient navigation but more complex algorithms and larger storage requirements



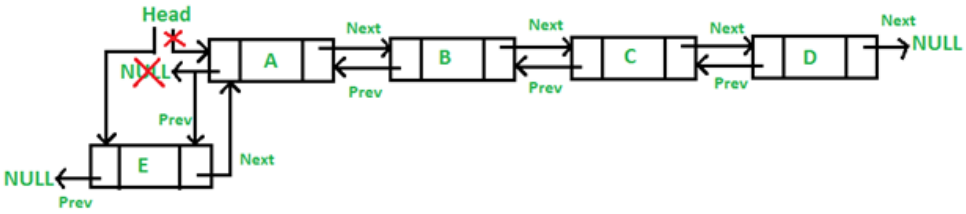
Creation

```
/* Node of a doubly linked list */
struct Node {
    int data;
    struct Node* next; // Pointer to next node in DLL
    struct Node* prev; // Pointer to previous node in DLL
};
```

Insertion

- **Add a node at the front:**

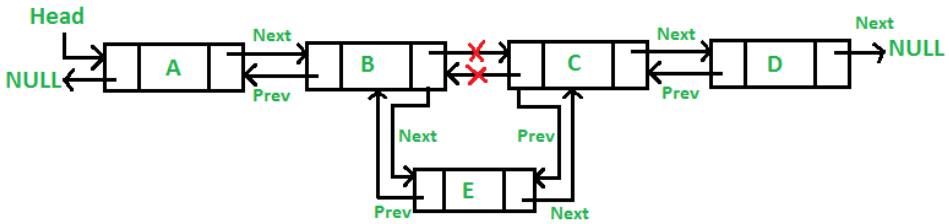
```
void insertBegining(struct Node** head_ref, int new_data)
{
   /* 1. allocate node */
   struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));

   /* 2. put in the data  */
   new_node->data = new_data;

   /* 3. Make next of new node as head and previous as NULL */
   new_node->next = (*head_ref);
   new_node->prev = NULL;

   /* 4. change prev of head node to new node */
   if ((*head_ref) != NULL)
      (*head_ref)->prev = new_node;

   /* 5. move the head to point to the new node */
   (*head_ref) = new_node;
}
```



Insertion cont…

- **Add a node after a given index:**

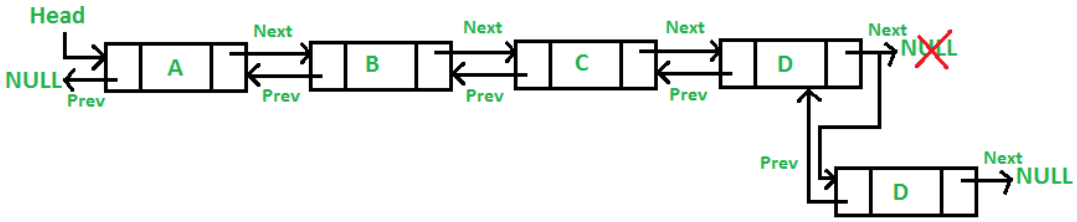
```
/* Given a reference (pointer to pointer) to the head
of a DLL and an int and index, inserts a new node after the index */
void insertAfterIndex(struct Node** head_ref, int new_data, int index)
{
    /* 1. allocate new node */
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));

    /* 2. if list in NULL or invalid position is given */
    if (*head_ref== NULL || index <= 0)
      return;

    struct Node* current = *head_ref;
    int i;

    /* 3. traverse up to the node at position 'index' from the beginning */
    for (i= 1; current != NULL && i< index; i++)
      current = current->next;

    /* 4. if 'index' is greater than the number of nodes in the doubly
          linked list */
    if (current == NULL)
      return;

    /* 5. put in the data */
    new_node->data = new_data;

    /* 6. Make next of new node as next of prev_node */
    new_node->next = current->next;

    /* 7. Make the next of prev_node as new_node */
    current->next = new_node;

    /* 8. Make prev_node as previous of new_node */
    new_node->prev = current;

    /* 9. Change previous of new_node's next node */
    if (new_node->next != NULL)
          new_node->next->prev = new_node;
}
```



Insertion cont…

- **Add a node at the end:**

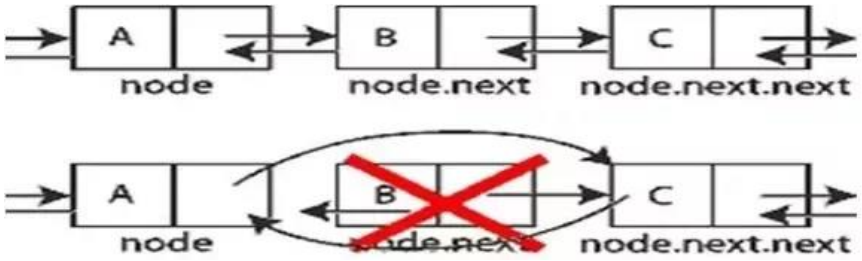
```
void addEnd(struct Node** head_ref, int new_data)
{   /* 1. allocate node */
   struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));

   struct Node* last = *head_ref; /* used in step 5*/

   /* 2. put in the data  */
   new_node->data = new_data;

   /* 3. This new node is going to be the last node, so
        make next of it as NULL*/
   new_node->next = NULL;

   /* 4. If the Linked List is empty, then make the new
        node as head */
   if (*head_ref == NULL) {
      new_node->prev = NULL;
      *head_ref = new_node;
      return;
   }
   /* 5. Else traverse till the last node */
   while (last->next != NULL)
      last = last->next;

   /* 6. Change the next of last node */
   last->next = new_node;

   /* 7. Make last node as previous of new node */
   new_node->prev = last;

   return;
}
```



Deletion

- **Deletion at the at the given node position:**

```
/* Function to delete a node in a Doubly Linked List.
  head_ref --> pointer to head node pointer.
  del  -->  pointer to node to be deleted. */
void deleteNode(struct Node** head_ref, struct Node* del)
{
  /* base case */
  if (*head_ref == NULL || del == NULL)
    return;

  /* If node to be deleted is head node */
  if (*head_ref == del)
    *head_ref = del->next;

  /* Change next only if node to be deleted is NOT the last node */
  if (del->next != NULL)
    del->next->prev = del->prev;

  /* Change prev only if node to be deleted is NOT the first node */
  if (del->prev != NULL)
    del->prev->next = del->next;

  /* Finally, free the memory occupied by del*/
  free(del);
  return;
}
```



Deletion cont…

- **Deletion at the given node index:**

```
/* Function to delete the node at the given node index
  in the doubly linked list */
void deleteNodeAtGivenIndex(struct Node** head_ref, int n)
{
   /* if list in NULL or invalid position is given */
   if (*head_ref == NULL || n <= 0)
      return;

   struct Node* current = *head_ref;
   int i;

   /* traverse up to the node at position 'n' from
     the beginning */
   for (int i = 1; current != NULL && i < n; i++)
      current = current->next;

   /* if 'n' is greater than the number of nodes
     in the doubly linked list */
   if (current == NULL)
      return;

   /* delete the node pointed to by 'current' */
   deleteNode(head_ref, current);
}
```

Printing

```
void printList(struct Node* node)
{
   while (node != NULL) {
      printf ("%d " node->data);
      node = node->next;
   }
}
```

Applications

- Doubly linked list can be used in navigation systems where both front and back navigation is required.
- It is used by browsers to implement backward and forward navigation of visited web pages i.e. back and forward button.
- It is also used by various application to implement Undo and Redo functionality.

Thank you