PES UNIVERSITY

CELEBRATING 50 YEARS

Data Structures

Dilip Kumar Maripuri

Computer Applications

Data Structures

Session : Delete Operations: At Pos, By Content

Dilip Kumar Maripuri

Computer Applications

Data Structures
Operations on Lists

- **Delete a Node at a position in a Singly Linked List**

Algorithm Delete_Position(Head, pos):

```
1. if Head = NULL then
    1.1 Print "Empty List. Cannot delete."
    1.2 return NULL
2. end if
3. if pos = 0 then                         // Special case: deleting the Head node
    3.1 Set temp ← Head
    3.2 Set Head ← Head.link
    3.3 Print "Deleted Node temp.data"
    3.4 Free the memory allocated to temp
    3.5 return Head
4. end if
```

Operations on Lists

- **Delete a Node at a position in a Singly Linked List**

Algorithm Delete_Position(Head, pos):

```
5. Set curr ← Head
6. for i ← 0 to pos - 1 do
     6.1 if curr.link = NULL then
         6.1.1 exit for loop                    // Stop if reached the end of the list
     6.2 Set curr ← curr.link
7. end for
```

- **Delete a Node at a position in a Singly Linked List**

Algorithm Delete_Position(Head, pos):

```
8.  if $curr.link$ = NULL then
        8.1  Print "Out of range. Node not found."
        8.2  return Head
9.  end if
10. Set $temp \leftarrow curr.link$
11. Set $curr.link \leftarrow temp.link$
12. Print "Deleted Node $temp.data$"
13. Free the memory allocated to $temp$
14. return Head
```

End Algorithm

```c
NODE delete_position(NODE Head, int pos) {

    if (Head == NULL) {
        printf("\n\t\tEmpty List. Cannot delete.");
        return NULL;
    }

    if (pos == 0) {
        NODE temp = Head;
        Head = Head->link;
        printf("\n\tDeleted Node %d", temp->data);
        free(temp);
        return Head;
    }
```

```c
NODE delete_position(NODE Head, int pos) {
    \\ Continued .....
    NODE curr = Head;
    for (int i = 0; curr->link != NULL && i < pos - 1; i++)
        curr = curr->link;

    if (curr->link == NULL) {
        printf("\n\tOut of range. Node not found.");
        return Head;
    }

    NODE temp = curr->link;
    curr->link = temp->link;
    printf("\n\tDeleted Node %d", temp->data);
    free(temp);

    return Head;
}
```

Operations on Lists

- **Delete a Node by Content Singly Linked List**
- Can we modularize and have functions as :
  - Search function : Can we generalize the function in a way that it returns the address of the previous node if found else NULL???
  - Use the search function and then delete the required node ??

- **Delete a Node by Content Singly Linked List**

Algorithm Search_And_Return_Previous(Head, key):

```
1.  if Head = NULL or Head.data = key then
      1.1  return NULL
2.  end if
3.  Set $curr \leftarrow$ Head
4.  while $curr.link \neq$ NULL and $curr.link.data \neq$ key do
      4.1  Set $curr \leftarrow curr.link$
5.  end while
6.  if $curr.link$ = NULL then
      6.1  return NULL                      // Key not found in the list
7.  else
      7.1  return $curr$                    // Return the previous node
8.  end if
```

End Algorithm

- **Delete a Node by Content Singly Linked List**

Algorithm Delete_By_Content(Head, key):

```
1. if Head = NULL then
    1.1 Print "Empty List. Cannot delete."
    1.2 return NULL
2. end if
3. if Head.data = key then                // Special case: deleting the Head node
    3.1 Set temp ← Head
    3.2 Set Head ← Head.link
    3.3 Print "Deleted Node temp.data"
    3.4 Free the memory allocated to temp
    3.5 return Head
4. end if
```

- **Delete a Node by Content Singly Linked List**

Algorithm Delete_By_Content(Head, key):

```
1. Set $prev \leftarrow$ Search_And_Return_Previous(Head, key)
2. if $prev$ = NULL or $prev.link$ = NULL then
    2.1 Print "Node with key $key$ not found."
    2.2 return Head
3. end if
4. Set $temp \leftarrow prev.link$
5. Set $prev.link \leftarrow temp.link$
6. Print "Deleted Node $temp.data$"
7. Free the memory allocated to $temp$
8. return Head
```

End Algorithm

```
NODE search_and_return_previous(NODE Head, int
   key) {
    if (Head == NULL || Head->data == key)
        return NULL;

    NODE curr = Head;
    while (curr->link != NULL && curr->link->
       data != key)
        curr = curr->link;

     return (if (curr->link == NULL) ? NULL :
        curr);
}
```

```c
NODE delete_by_content(NODE Head, int key) {
    if (Head == NULL) {
        printf("\n\t\tEmpty List. Cannot delete.");
        return NULL;
    }

    if (Head->data == key) {
        NODE temp = Head;
        Head = Head->link;
        printf("\n\tDeleted Node %d", temp->data);
        free(temp);
        return Head;
    }
```

```
NODE delete_by_content(NODE Head, int key) {
    \\ Continued ....
    NODE prev = search_and_return_previous(Head,
        key);

    if (prev == NULL) {
        printf("\n\tNode with key %d not found."
            , key);
        return Head;
    }

    NODE temp = prev->link;
    prev->link = temp->link;
    printf("\n\tDeleted Node %d", temp->data);
    free(temp);
    return Head;
}
```

PES
UNIVERSITY
CELEBRATING 50 YEARS

Thank You

Dilip Kumar Maripuri
Associate Professor
Department of Computer Applications

dilip.maripuri@pes.edu
8073212026