Lab 03 Advanced Topics in Singly Linked List Implementation

Objectives:

- Learn to insert nodes in a linked list.
- Learn to delete nodes from the linked list.
- Learn to store/load database to/from a file.

Pre-Lab

Reading Task 1:

Read linked lists node insertion and deletion topics (page 167 to 180) from the book "*Data Structures using C*" by Reema Thareja, 2nd Edition. Some excerpts from the book are listed here for convenience.

Inserting Nodes in a Linked List:

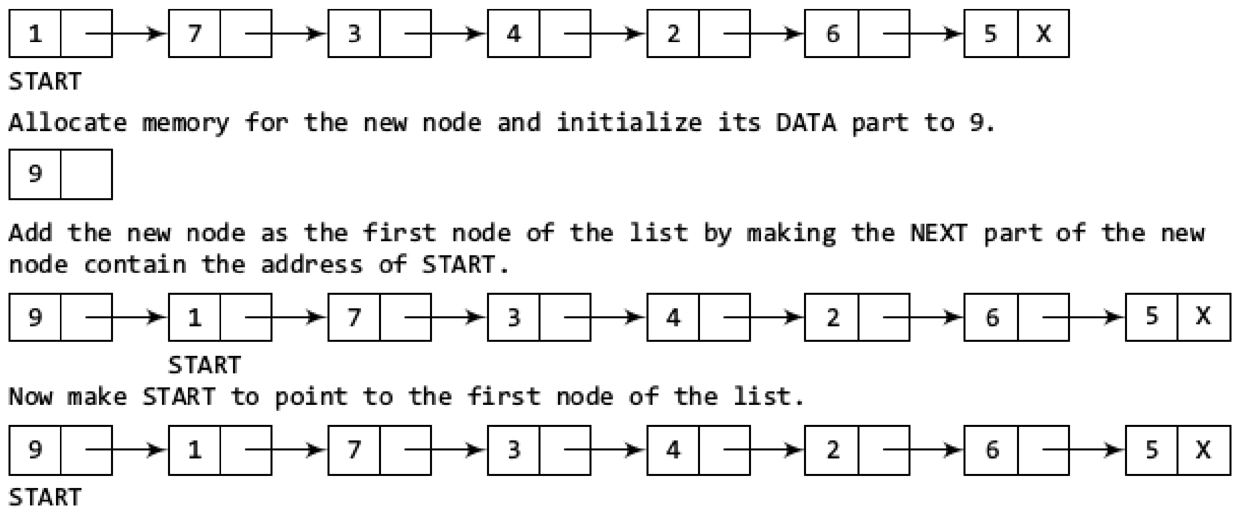
New nodes can be inserted in a linked list in a variety of ways. Some of the cases are listed below:

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.

Case 1: The new node is inserted at the beginning:

If a new node is to be inserted at the beginning of a linked list, following steps should be performed:

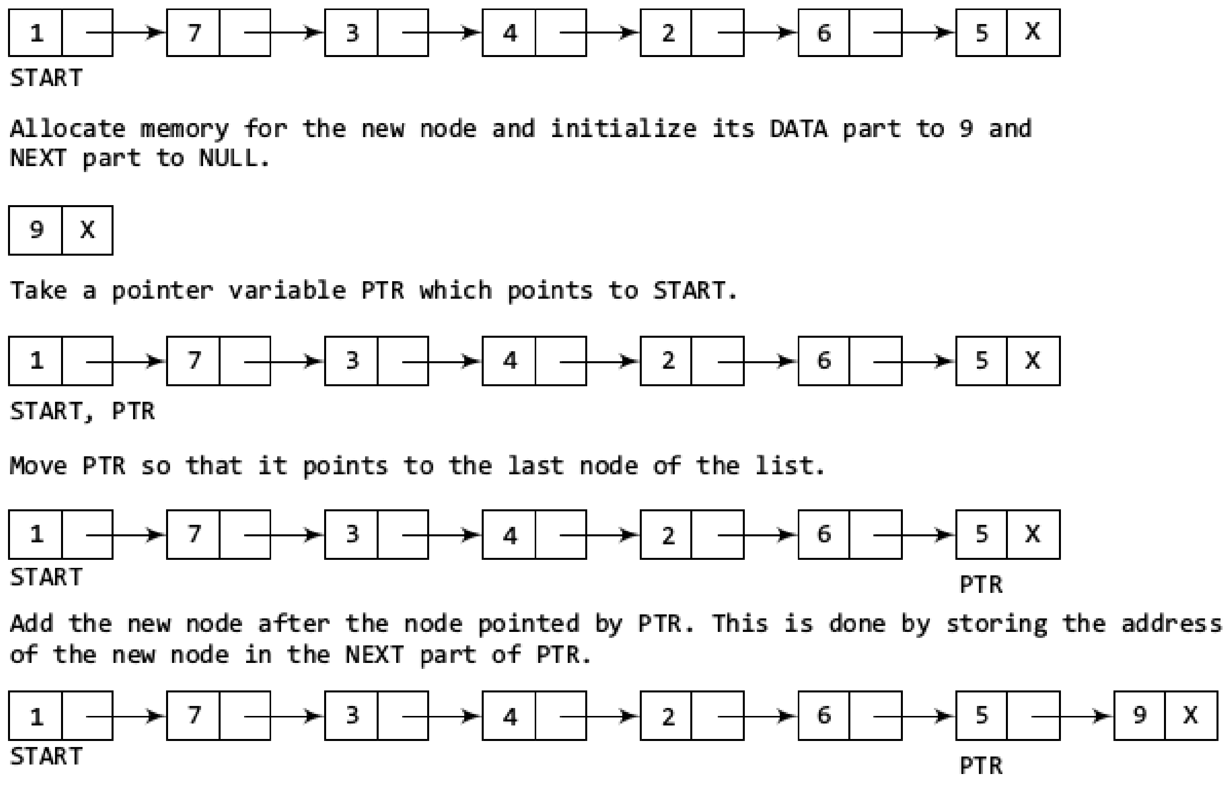
- Allocate space for new node (unless system runs out of memory) and get its data.
- Let the 'next' part of new node contain the address of 'start'
- Make 'start' point to the new node (which is now the first node in the list).


1 7 3 4 2 6 5 X
START
Allocate memory for the new node and initialize its DATA part to 9.
9
Add the new node as the first node of the list by making the NEXT part of the new node contain the address of START.
9 1 7 3 4 2 6 5 X
START
Now make START to point to the first node of the list.
9 1 7 3 4 2 6 5 X
START
Figure 1: Inserting a new node at the beginning of the list

Case 2: The new node is inserted at the end:

If a new node is to be inserted at the end of a linked list, following steps should be performed:

- Allocate space for new node (unless system runs out of memory) and get its data.
- Take a pointer which is pointing to the start of the list.
- Move this pointer to the last node of the list.
- Store the address of the new node in the ‘next’ part of this pointer.
- Now this new node is the last node in the list.
- Assign the ‘next’ pointer of the las node as NULL.



Allocate memory for the new node and initialize its DATA part to 9 and NEXT part to NULL.

Take a pointer variable PTR which points to START.

Move PTR so that it points to the last node of the list.

Add the new node after the node pointed by PTR. This is done by storing the address of the new node in the NEXT part of PTR.

Figure 2: Inserting new node to the end of a linked list

Read through rest of the section for the remaining two cases of node insertion in the linked list.

Deleting Nodes from a Linked List:

Nodes from an existing list can be deleted in a number of ways. Following are the most common cases when deleting nodes from a linked list.

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

Case 1: The first node is to be deleted.

When a the first node is to be deleted from the list following steps should be performed.

- Make ‘start’ point to the next node in the list. (but first copy its address in pointer).
- De-allocate the memory for the previous first node (using the above mentioned pointer).

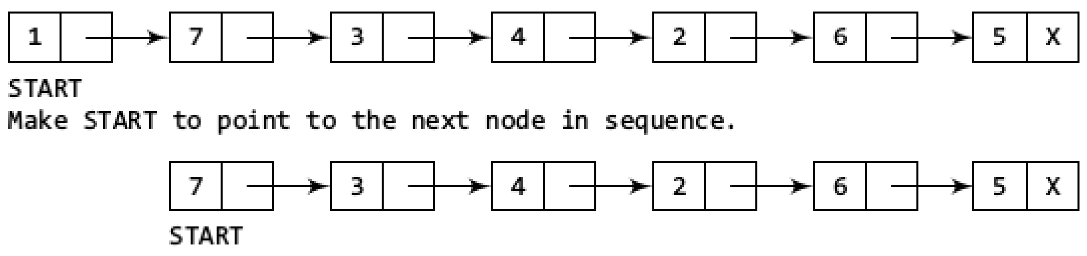


Figure 3: Deleting first node from the list

Case 2: The last node is to be deleted.

When the last node is to be deleted from the list following steps should be performed.

- Declare an pointer ‘ptr’ pointing to the first node in the list.
- Move this pointer to the second-last node in the array.
- De-allocate the memory for the last node.
- Make the ‘next’ pointer of the previous second-last node equal to NULL.

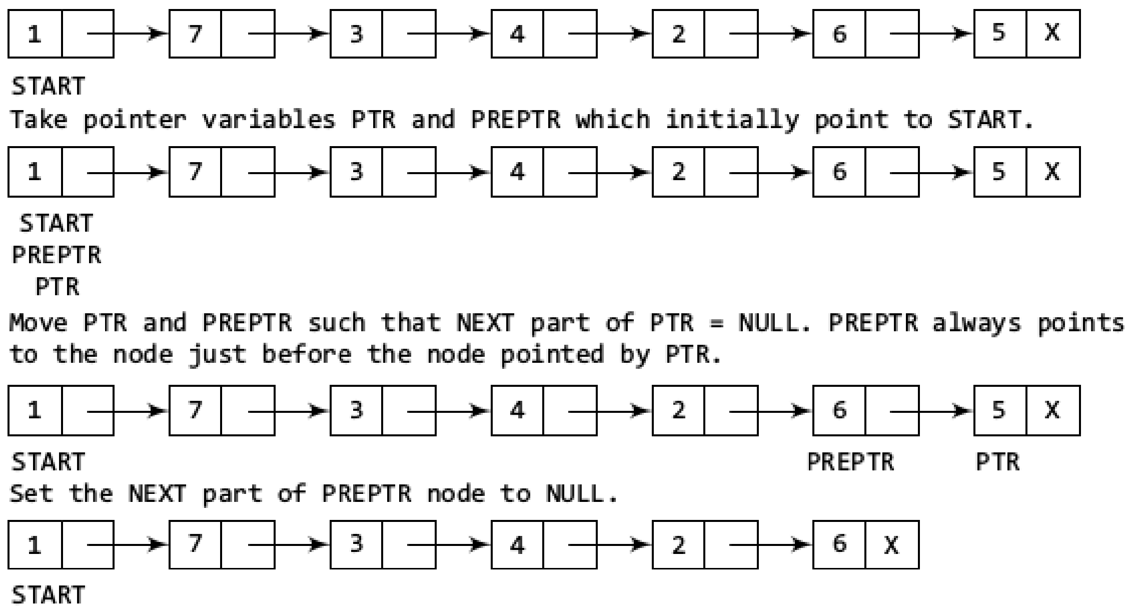

1 7 3 4 2 6 5 X
START
Take pointer variables PTR and PREPTR which initially point to START.
1 7 3 4 2 6 5 X
START
PREPTR
PTR
Move PTR and PREPTR such that NEXT part of PTR = NULL. PREPTR always points to the node just before the node pointed by PTR.
1 7 3 4 2 6 5 X
START
PREPTR
PTR
Set the NEXT part of PREPTR node to NULL.
1 7 3 4 2 6 X
START

Figure 4: Deleting the last node from the list

Read rest of the description of deleting nodes from the book.

Storing Data to file:

For writing in file, it is easy to write string or int to file using ***fprintf*** and ***putc***, but you might have faced difficulty when writing contents of struct. ***fwrite*** and ***fread*** make task easier when you want to write and read blocks of data. Following is a sample C program used to write structures to file using ***'fwrite'*** function.

```
// a struct to read and write
struct person
{
    int id;
    char fname[20];
    char lname[20];
};

int main ()
{
    FILE *fptr;

    // open file for writing
    fptr = fopen ("person.dat", "wb");
    if (outfile == NULL)
    {
        printf("\nError opening file!!\n");
        exit (1);
    }

    struct person input1 = {1, "Kamran", "Asghar"};
    struct person input2 = {2, "Jameel", "Ahmed"};

    // write struct to file
    fwrite (&input1, sizeof(struct person), 1, fptr);
    fwrite (&input2, sizeof(struct person), 1, fptr);

    if(fwrite != 0)
        printf("contents to file written successfully !\n");
    else
        printf("Error writing to file !\n");

    // close file
    fclose (fptr);

    return 0;
}
```

CodeListing 1: Example program for writing structures to file.

Further explanation can be found at the following link:

https://www.geeksforgeeks.org/readwrite-structure-file-c/

In-Lab Tasks:

You are given the following four files for this lab;

1. ‘main.c’ // This file contains the main application (menu based)
2. ‘SinglyLinkedList.c’ // This file contains the functions to implement linked list.
3. ‘SinglyLinkedList.h’ // This file contains the prototypes of functions.
4. ‘Node.h’ // This file contains the structure definitions for the linked list.

As you would know that each node in a singly linked list consists of a data part and a pointer to the next node in the list. Our implementation defines the node as follows (in file ***‘Node.h’***).

```
struct employee
{
   char name[50];
   int age;
   float bs;
};

struct node
{

    struct employee data;

    struct node * next;
};
```

You would notice that we have made the data part as a separate structure. This will allow us to write generic functions for linked list modifications even if there are changes in the ‘data’ part. Moreover it will allow us to store/load our database to/from a file much more conveniently.

In-Lab Task 1:

‘Inserting nodes at the end’ and ‘inserting node after a given node’ are already implemented in ***‘SinglyLinkedList.c’***. Your task is to implement ‘***insert at the beginning***’ and ‘***insert before***’ functions in the file ***‘SinglyLinkedList.c’***.

In-Lab Task 2:

Deleting a node from the end is already implemented in ***‘SinglyLinkedList.c’*** your task is to implement ‘***delete from beginning***’ and ‘***delete after***’ a given node.

Post-Lab Task:

Reading database from a file on the hard disk is already implemented. Your first task is to study and understand this implementation. Then you will have to implement the write to file function ***‘saveListToFile()’.*** Submit a report on your implementation.

********** End of Document *********