Experiment No. : 2

Title: Demonstrate the use of structures and pointer / class and objects to implement Singly Linked List (SLL).

Batch: SY_IT (B2) **Roll No.: 16010423076** **Experiment No.:2**

Aim: Implementing Singly Linked List (SLL) supporting following operations using menu driven program.

1. Insert at the Begin
2. Insert after the specified existing node
3. Delete before the specified existing node
4. Display all elements in tabular form.

Resources Used: Turbo C/ C++ editor and compiler (online or offline).

Theory:

Singly Linked List :-

Singly Linked Lists are a type of data structure. It is a type of list. In a singly linked list each node in the list stores the contents of the node and a pointer or reference to the next node in the list. It does not store any pointer or reference to the previous node. It is called a singly linked list because each node only has a single link to another node. To store a single linked list, you only need to store a reference or pointer to the first node in that list. The last node has a null pointer to indicate that it is the last node.

A linked list is a linear data structure where each element is a separate object.

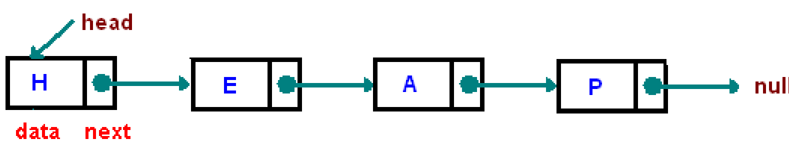

Fig 1.1 : Example of Singly Linked List

Each element (we will call it a node) of a list is comprising of two items - the data and a reference to the next node. The last node has a reference to null. The entry point into a linked list is called the head of the list. It should be noted that head is not a separate node, but the reference to the first node. If the list is empty then the head is a null reference.

A linked list is a dynamic data structure. The number of nodes in a list is not fixed and can grow and shrink on demand. Any application which has to deal with an unknown number of objects will need to use a linked list.

One disadvantage of a linked list against an array is that it does not allow direct access to the individual elements. If you want to access a particular item then you have to start at the head and follow the references until you get to that item.

Another disadvantage is that a linked list uses more memory compare with an array - we extra 4 bytes (on 32-bit CPU) to store a reference to the next node.

Algorithm :

Program should implement the specified operations strictly in the following manner. Also implement a support method isempty() and make use of it at appropriate places.

1. **createSLL()** – This void function should create a START/HEAD pointer with NULL value as empty SLL.
2. **insertBegin(typedef newelement)** – This void function should take a newelement as an argument to be inserted on an existing SLL and insert it before the element pointed by the START/HEAD pointer.
3. **insertAfter(typedef newelement, typedef existingelement)** – This void function should take two arguments. The function should search for an existingelement on non-empty SLL and insert newelement after this element.
4. **typedef deleteBefore(typedef existingelement)** – This function should search for the existing element passed to the function in the non-empty SLL, delete the node siting before it and return the deleted element.
5. **display( )** – This is a void function which should go through non- empty SLL starting from START/HEAD pointer and display each element of the SLL till the end.

NOTE : All functions should be able to handle boundary(exceptional) conditions.

Program : (copy-paste code here)

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

typedef struct Node {
   int data;
   struct Node* next;
} Node;

Node* START = NULL;

int isempty() {
   return START == NULL;
}

void createSLL() {
   START = NULL;
}

void insertBegin(int newele) {
```

```
    Node* newnode = (Node*)malloc(sizeof(Node));
    newnode->data = newele;
    newnode->next = START;
    START = newnode;
}

void insertAfter(int newele, int existingele) {
    Node* newnode = (Node*)malloc(sizeof(Node));
    newnode->data = newele;

    Node* now = START;
    while (now != NULL && now->data != existingele) {
        now = now->next;
    }

    if (now != NULL) {
        newnode->next = now->next;
        now->next = newnode;
    }
}

int deleteBefore(int existingele) {
    if (isempty()) {
        printf("List is empty. Cannot delete before %d.\n", existingele);
        return -1;
    }
    if (START->next == NULL || START->data == existingele) {
        printf("No element exists before %d to delete.\n", existingele);
        return -1;
    }
    Node* prev = NULL;
    Node* now = START;
    while (now->next != NULL && now->next->data != existingele) {
        prev = now;
        now = now->next;
    }
    if (now->next == NULL) {
        printf("Element %d not found in the list.\n", existingele);
        return -1;
    }
    int deletedData;
    if (prev == NULL) {
```

```
        Node* temp = START;
        START = START->next;
        deletedData = temp->data;
        free(temp);
    } else {
        prev->next = now->next;
        deletedData = now->data;
        free(now);
    }
    return deletedData;
}

void display() {
    if (isempty()) {
        printf("List is empty.\n");
        return;
    }
    Node* now = START;
    while (now != NULL) {
        printf("%d > ", now->data);
        now = now->next;
    }
    printf("NULL\n");
}

int main() {
    createSLL();
    insertBegin(30);
    insertBegin(25);
    insertBegin(20);
    insertBegin(15);
    insertBegin(10);
    insertBegin(5);
    insertBegin(0);
    display();
    insertAfter(24, 20);
    display();
    deleteBefore(15);

    display();
    return 0;
}
```

Output :

```
0 > 5 > 10 > 15 > 20 > 25 > 30 > NULL
0 > 5 > 10 > 15 > 20 > 24 > 25 > 30 > NULL
0 > 5 > 15 > 20 > 24 > 25 > 30 > NULL


=== Code Execution Successful ===
```

Conclusion :

I learned from the program how a basic singly linked list could be implemented in C. I covered all the major operations: the creation of an empty list, the insertion of nodes at the beginning and after some elements, and finally, the display of the content. In this code, I used structures, pointers, and dynamic memory allocation to manipulate the linked list. I have also learned how to manipulate a linked list for inserting a node after a specific element.

Outcomes achieved: (refer exp list)

CO1. Comprehend the different data structures used in problem solving.

CO2. Apply linear and non-linear data structure in application development.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

- Y. Langsam, M. Augenstin and A. Tannenbaum, “Data Structures using C”, Pearson Education Asia, 1st Edition, 2002.
- E. Horowitz, S. Sahni, S.Anderson-freed, “Fundamentals of Data Structures in C”, 2nd Edition, University Press