Experiment No. : 2

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

Batch: B1 **Roll No.: 16010422234** **Name: Chandana Ramesh Galgali**

Experiment No.:2

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

1. Insert at the Beginning
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.

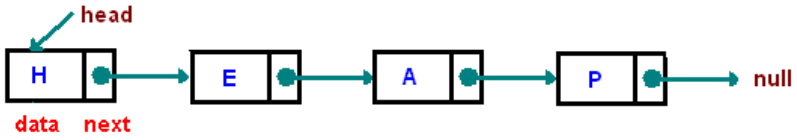

Fig 1.1 : Example of Singly Linked List

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 :

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

struct node{
   int data;
   struct node *next;
}n;

struct node* head = NULL;

struct node* createSLL(struct node *head)
{
   struct node *new_node, *ptr;
   int i=0, n, num;
   printf("\nEnter the number of nodes in the list: ");
```

```
    scanf("%d", &n);
    while(i < n)
    {
        printf("\nEnter the data: ");
        scanf("%d", &num);
        if(head == NULL)
        {
            new_node = (struct node*)malloc(sizeof(struct node));
            new_node -> data = num;
            new_node -> next = NULL;
            head = new_node;
        }
        else
        {
            ptr = head;
            new_node = (struct node*)malloc(sizeof(struct node));
            new_node -> data = num;
            while(ptr -> next != NULL)
            {
                ptr = ptr -> next;
            }
                ptr -> next = new_node;
                new_node -> next = NULL;
        }
        i++;
    }
    return head;
}

struct node* insertBegin(struct node *head, int new_element)
{
    struct node* new_node;
    new_node = (struct node*)malloc(sizeof(struct node));
    new_node -> data = new_element;
    new_node -> next = head;
    head = new_node;
    return head;
}

struct node* insertAfter(struct node *head, int new_element, int existing_element)
{
    struct node* new_node, *ptr, *prev;
```

```
    new_node = (struct node*)malloc(sizeof(struct node));
    new_node -> data = new_element;
    new_node -> next = NULL;
    ptr = head;
    prev = ptr;
    while (prev -> data != existing_element)
    {
       prev = ptr;
       ptr = ptr -> next;
    }
    prev -> next = new_node;
    new_node -> next = ptr;
    return head;
}

struct node* deleteBefore(struct node* head, int existing_element)
{
    int deleted_element;
    struct node *ptr, *preptr, *prev;
    ptr = head;
    preptr = ptr;
    prev = preptr;
    while(ptr -> data != existing_element)
    {
       prev = preptr;
       preptr = ptr;
       ptr = ptr -> next;
    }
    prev -> next = ptr;
    deleted_element = preptr -> data;
    printf("\nThe deleted element is: %d", deleted_element);
    free(preptr);
}

void display(struct node *head)
{
    struct node *prev, *ptr;
    ptr = head;
    prev = ptr;
    printf("\nData Part\tAddress of the next node");
    printf("\n\t\t%p", ptr);
    ptr = ptr -> next;
```

```
    while(ptr != NULL)
    {
        printf("\n%d\t\t%p", prev -> data, ptr);
        prev = ptr;
        ptr = ptr -> next;
    }
    printf("\n%d\t\t%p", prev -> data, ptr);
    printf("\n");
}

int main()
{
    int choice, new_element_b, new_element_a, existing_element_a, existing_element_b;
    printf("\n1.Create a Singly Linked List\n2.Insert at the Beginning\n3.Insert after the specified existing node\n4.Delete before the specified existing node\n5.Display all elements in tabular form\n6.Exit");
    printf("\nChoose an option: ");
    scanf("%d",&choice);
    switch(choice)
    {
    case 1:
        head = createSLL(head);
        main();
        break;
    case 2:
        printf("\nEnter the data to be inserted at the beginning: ");
        scanf("%d", &new_element_b);
        head = insertBegin(head, new_element_b);
        main();
        break;
    case 3:
        printf("\nEnter the data to be inserted: ");
        scanf("%d", &new_element_a);
        printf("\nEnter the existing element after which the data has to be inserted: ");
        scanf("%d", &existing_element_a);
        head = insertAfter(head, new_element_a, existing_element_a);
        main();
        break;
    case 4:
        printf("\nEnter the existing_element before which the node has to deleted: ");
        scanf("%d", &existing_element_b);
        deleteBefore(head, existing_element_b);
```

```
        main();
        break;
    case 5:
        display(head);
        main();
        break;
    case 6:
        break;
    default:
        printf("\nChoose a valid option!");
        break;
    }
    return 0;
}
```

Output :

```
1.Create a Singly Linked List
2.Insert at the Beginning
3.Insert after the specified existing node
4.Delete before the specified existing node
5.Display all elements in tabular form
6.Exit
Choose an option: 1

Enter the number of nodes in the list: 5

Enter the data: 20

Enter the data: 30

Enter the data: 60

Enter the data: 40

Enter the data: 60
```

```
1.Create a Singly Linked List
2.Insert at the Beginning
3.Insert after the specified existing node
4.Delete before the specified existing node
5.Display all elements in tabular form
6.Exit
Choose an option: 5

Data Part       Address of the next node
                0000000000BD6C40
20              0000000000BD6C60
30              0000000000BD6C80
60              0000000000BD6CA0
40              0000000000BD6CC0
60              0000000000000000
```

```
1.Create a Singly Linked List
2.Insert at the Beginning
3.Insert after the specified existing node
4.Delete before the specified existing node
5.Display all elements in tabular form
6.Exit
Choose an option: 2

Enter the data to be inserted at the beginning: 10

1.Create a Singly Linked List
2.Insert at the Beginning
3.Insert after the specified existing node
4.Delete before the specified existing node
5.Display all elements in tabular form
6.Exit
Choose an option: 5

Data Part       Address of the next node
                0000000000BD6CE0
10              0000000000BD6C40
20              0000000000BD6C60
30              0000000000BD6C80
60              0000000000BD6CA0
40              0000000000BD6CC0
60              0000000000000000
```

```
1.Create a Singly Linked List
2.Insert at the Beginning
3.Insert after the specified existing node
4.Delete before the specified existing node
5.Display all elements in tabular form
6.Exit
Choose an option: 3

Enter the data to be inserted: 50

Enter the existing element after which the data has to be inserted: 40

1.Create a Singly Linked List
2.Insert at the Beginning
3.Insert after the specified existing node
4.Delete before the specified existing node
5.Display all elements in tabular form
6.Exit
Choose an option: 5

Data Part       Address of the next node
                0000000000BD6CE0
10              0000000000BD6C40
20              0000000000BD6C60
30              0000000000BD6C80
60              0000000000BD6CA0
40              0000000000BD6D00
50              0000000000BD6CC0
60              0000000000000000
```



```
1.Create a Singly Linked List
2.Insert at the Beginning
3.Insert after the specified existing node
4.Delete before the specified existing node
5.Display all elements in tabular form
6.Exit
Choose an option: 4

Enter the existing_element before which the node has to deleted: 40

The deleted element is: 60
1.Create a Singly Linked List
2.Insert at the Beginning
3.Insert after the specified existing node
4.Delete before the specified existing node
5.Display all elements in tabular form
6.Exit
Choose an option: 5

Data Part       Address of the next node
                0000000000BD6CE0
10              0000000000BD6C40
20              0000000000BD6C60
30              0000000000BD6CA0
40              0000000000BD6D00
50              0000000000BD6CC0
60              0000000000000000
```

```
1.Create a Singly Linked List
2.Insert at the Beginning
3.Insert after the specified existing node
4.Delete before the specified existing node
5.Display all elements in tabular form
6.Exit
Choose an option: 6

Process returned 0 (0x0)   execution time : 57.174 s
Press any key to continue.
```

Conclusion:

The experiment successfully achieves the aim of implementing a Singly Linked List (SLL) supporting the mentioned operations using a menu-driven program. The Singly Linked List data structure proves to be a useful tool for managing and manipulating a collection of elements efficiently.

Outcomes achieved:

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