**Experiment No. : 2**

**Title: Implementation of Singly Linked List**

**Batch: B1 Roll No.: 1914078 Experiment No.:2**

**Aim:** Implementing Singly Linked List (SLL) supporting following operations using menu drivenprogram.

1. Insert at the Head
2. Insert after the specified existing node
3. Delete before the specified existing node
4. Display all elements

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Resources Used:** Turbo C/ C++/JAVA 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.

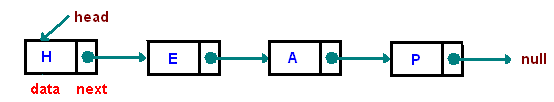


Fig 1.1 : Example of Singly Linked List

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. **create()** – 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 last element of the non-empty SLL, delete 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 the each element of the SLL till the end.

**Code:**

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

#include <ctype.h>

struct Node

{

    char name[15];

    int roll, age;

    float avg;

    struct Node \*next;

} \* head, \*currNode, \*temp, \*ptr;

int numberOfElements = 0;

void create();

void insert();

void addStart();

void addEnd();

void addIntermediate();

void delete ();

void deleteStart();

void deleteEnd();

void deleteIntermediate();

void display();

int main()

{

    head = NULL;

    int ch = 0;

    int num;

    printf("Enter number of prefilled items to be added: ");

    scanf("%d", &num);

    numberOfElements = num;

    for (int i = 0; i < num; i++)

    {

        create();

    }

    while (ch != 4)

    {

        printf("\n\n1. Insert");

        printf("\n2. Delete");

        printf("\n3. Display");

        printf("\n4. Quit");

        printf("\nEnter your choice from 1-4: ");

        scanf("%d", &ch);

        switch (ch)

        {

        case 1:

            insert();

            break;

        case 2:

            delete ();

            break;

        case 3:

            display();

            break;

        case 4:

            break;

        default:

            printf("\nEnter your choice from 1-4 only");

            continue;

        }

    }

    printf("Thank you");

    return 0;

}

void create()

{

    ptr = (struct Node \*)malloc(sizeof(struct Node));

    currNode = head;

    printf("\nEnter name: ");

    scanf("%s", ptr->name);

    printf("Enter roll no. : ");

    scanf("%d", &ptr->roll);

    printf("Enter age: ");

    scanf("%d", &ptr->age);

    printf("Enter average: ");

    scanf("%f", &ptr->avg);

    if (head == NULL)

    {

        head = ptr;

        ptr->next = NULL;

    }

    else

    {

        while (currNode->next != NULL)

        {

            currNode = currNode->next;

        }

        currNode->next = ptr;

        ptr->next = NULL;

    }

}

void insert()

{

    int ch = 0;

    while (ch != 4)

    {

        printf("\n\n1. Add at start");

        printf("\n2. Add at end");

        printf("\n3. Add at intermediate");

        printf("\n4. back");

        printf("\nEnter your choice from 1-4: ");

        scanf("%d", &ch);

        switch (ch)

        {

        case 1:

            addStart();

            break;

        case 2:

            addEnd();

            break;

        case 3:

            addIntermediate();

            break;

        case 4:

            break;

        default:

            printf("\nEnter your choice from 1-4 only");

            break;

        }

    }

}

void addStart()

{

    ptr = (struct Node \*)malloc(sizeof(struct Node));

    printf("\nEnter name: ");

    scanf("%s", ptr->name);

    printf("Enter roll no. : ");

    scanf("%d", &ptr->roll);

    printf("Enter age: ");

    scanf("%d", &ptr->age);

    printf("Enter average: ");

    scanf("%f", &ptr->avg);

    ptr->next = head;

    head = ptr;

    numberOfElements++;

    printf("\nAdded a new elemnt\n");

}

void addEnd()

{

    if (head != NULL)

    {

        ptr = (struct Node \*)malloc(sizeof(struct Node));

        currNode = head;

        printf("\nEnter name: ");

        scanf("%s", ptr->name);

        printf("Enter roll no. : ");

        scanf("%d", &ptr->roll);

        printf("Enter age: ");

        scanf("%d", &ptr->age);

        printf("Enter average: ");

        scanf("%f", &ptr->avg);

        while (currNode->next != NULL)

        {

            currNode = currNode->next;

        }

        currNode->next = ptr;

        ptr->next = NULL;

        numberOfElements++;

        printf("\nAdded a new elemnt\n");

    }

    else

    {

        addStart();

    }

}

void addIntermediate()

{

    ptr = (struct Node \*)malloc(sizeof(struct Node));

    currNode = head;

    int pos, count = 1;

    printf("\nEnter position where you want to add the element[starts from 1]: ");

    scanf("%d", &pos);

    if (pos > 1)

    {

        while (count != (pos - 1) && currNode->next != NULL)

        {

            currNode = currNode->next;

            count++;

        }

        if (count == pos - 1)

        {

            printf("\nEnter name: ");

            scanf("%s", ptr->name);

            printf("Enter roll no. : ");

            scanf("%d", &ptr->roll);

            printf("Enter age: ");

            scanf("%d", &ptr->age);

            printf("Enter average: ");

            scanf("%f", &ptr->avg);

            ptr->next = currNode->next;

            currNode->next = ptr;

            numberOfElements++;

            printf("\nAdded a new elemnt\n");

        }

        else

        {

            printf("\nPosition not found, please enter a valid position\n");

        }

    }

    else if (pos == 1)

    {

        addStart();

    }

    else

    {

        printf("\nPosition not found, please enter a valid position\n");

    }

}

// ------------------Delete Functions------------------------

void delete ()

{

    int ch = 0;

    while (ch != 4)

    {

        printf("\n\n1. Delete at start");

        printf("\n2. Delete at end");

        printf("\n3. Delete at intermediate");

        printf("\n4. back");

        printf("\nEnter your choice from 1-4: ");

        scanf("%d", &ch);

        switch (ch)

        {

        case 1:

            deleteStart();

            break;

        case 2:

            deleteEnd();

            break;

        case 3:

            deleteIntermediate();

            break;

        case 4:

            break;

        default:

            printf("\nEnter your choice from 1-4 only");

            break;

        }

    }

}

void deleteStart()

{

    if (head != NULL)

    {

        currNode = head;

        head = currNode->next;

        currNode->next = NULL;

        printf("\nDeleted the elemnt\n");

        printf("%s\n", currNode->name);

        printf("%d\n", currNode->roll);

        printf("%d\n", currNode->age);

        printf("%f\n\n", currNode->avg);

        free(currNode);

        numberOfElements--;

    }

    else

    {

        printf("\nEmpty Linked List\n");

    }

}

void deleteEnd()

{

    if (head != NULL)

    {

        if (numberOfElements != 1)

        {

            currNode = head;

            while (currNode->next->next != NULL)

            {

                currNode = currNode->next;

            }

            ptr = currNode->next;

            currNode->next = NULL;

            printf("\nDeleted the elemnt\n");

            printf("%s\n", ptr->name);

            printf("%d\n", ptr->roll);

            printf("%d\n", ptr->age);

            printf("%f\n\n", ptr->avg);

            free(ptr);

            numberOfElements--;

        }

        else

        {

            deleteStart();

        }

    }

    else

    {

        printf("\nEmpty Linked List\n");

    }

}

void deleteIntermediate()

{

    currNode = head;

    int pos, count = 1;

    printf("\nEnter position where you want to delete the element: ");

    scanf("%d", &pos);

    if (pos > numberOfElements)

    {

        printf("\nPosition not found, please enter a valid position\n");

    }

    else if (pos > 1)

    {

        printf("273");

        while (count != (pos - 1))

        {

            currNode = currNode->next;

            count++;

        }

        if (pos > numberOfElements)

        {

            printf("\nPosition not found, please enter a valid position\n");

        }

        else if (count == (pos - 1))

        {

            ptr = currNode->next;

            currNode->next = ptr->next;

            ptr->next = NULL;

            printf("\nDeleted the elemnt\n");

            printf("%s\n", ptr->name);

            printf("%d\n", ptr->roll);

            printf("%d\n", ptr->age);

            printf("%f\n\n", ptr->avg);

            free(ptr);

            numberOfElements--;

        }

        else

        {

            printf("\nPosition not found, please enter a valid position\n");

        }

    }

    else if (pos == 1)

    {

        deleteStart();

    }

    else

    {

        printf("\nPosition not found, please enter a valid position\n");

    }

}

// -----------------------------------------------------------

void display()

{

    if (head != NULL)

    {

        currNode = head;

        // printf(numberOfElements);

        if (currNode->next == NULL)

        {

            printf("\n%-10s\t", currNode->name);

            printf("%-10d\t", currNode->roll);

            printf("%-10d\t", currNode->age);

            printf("%-10.2f\t", currNode->avg);

        }

        else

        {

            while (currNode != NULL)

            {

                printf("\n%-10s\t", currNode->name);

                printf("%-10d\t", currNode->roll);

                printf("%-10d\t", currNode->age);

                printf("%-10.2f\t", currNode->avg);

                currNode = currNode->next;

            }

        }

    }

    else

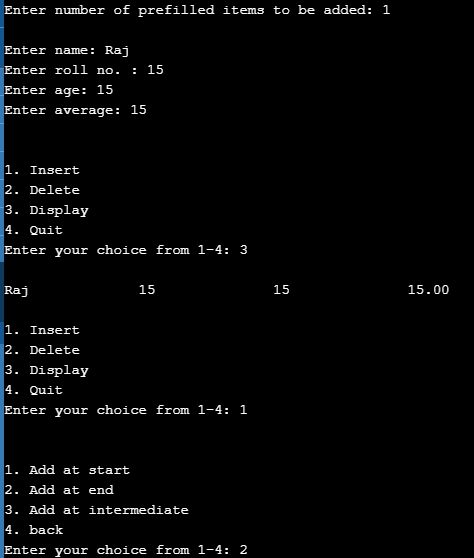
    {

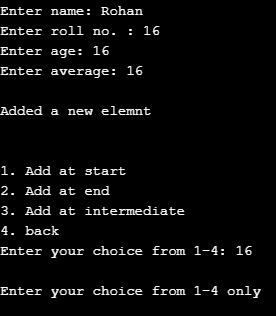
        printf("\nEmpty Linked List\n");

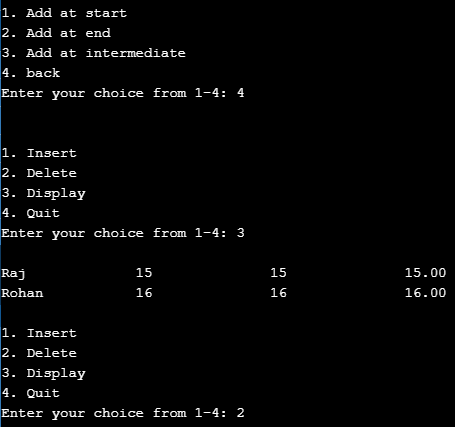
    }

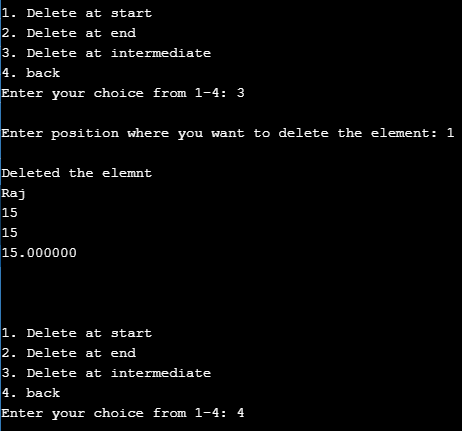
}

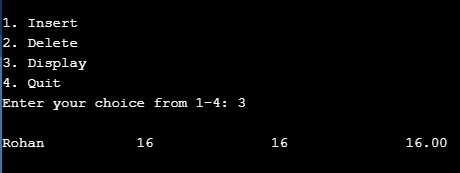
**Output:**

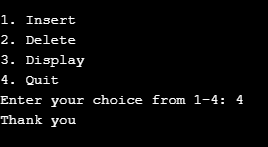
****

****

****

****

****

****

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

**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