Experiment No.6
Implement Singly Linked List ADT
Name: Sharvari Anand Bhondekar
Roll No: 06
Date of Performance:
Date of Submission:
Marks:
Sign:

Experiment No. 6: Singly Linked List Operations

Aim: Implementation of Singly Linked List

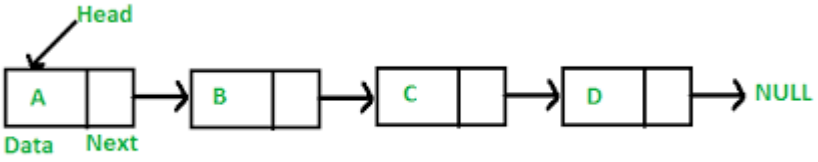
Objective:

It is used to implement stacks and queues which are like fundamental needs throughout computer science. To prevent the collision between the data in the hash map, we use a singly linked list.

Theory:

A linked list is an ordered collection of elements, known as nodes. Each node has two fields: one for data (information) and another to store the address of the next element in the list. The address field of the last node is null, indicating the end of the list. Unlike arrays, linked list elements are not stored in contiguous memory locations; instead, they are connected by explicit links, allowing for dynamic and non-contiguous memory allocation.

The structure of linked list is as shown below



Header is a node containing null in its information field and an next address field contains the address of the first data node in the list. Various operations can be performed on singly linked lists like insertion at front, end, after a given node, before a given node deletion at front, at end and after a given node.

Algorithm

Algorithm to insert a new node at the beginning

Step 1: IF AVAIL = NULL

Write OVERFLOW

Go to Step 7 [END OF IF]

Step 2: SET NEW_NODE = AVAIL

Step 3: SET AVAIL = AVAIL NEXT

Step 4: SET DATA = VAL

Step 5: SET NEW_NODE -->NEXT = START

Step 6: SET START = NEW_NODE

Step 7: EXIT

Algorithm to insert a new node at the end

Step 1: IF AVAIL = NULL

Write OVERFLOW

Go to Step 1 [END OF IF]

Step 2: SET = AVAIL

Step 3: SET AVAIL = AVAIL NEXT

Step 4: SET DATA = VAL

Step 5: SET NEW_NODE = NULL

Step 6: SET PTR = START

Step 7: Repeat Step 8 while PTR NEXT != NULL

Step 8: SET PTR = PTR NEXT [END OF LOOP]

Step 9: SET PTR--> NEXT = New_Node

Step 10: EXIT

Algorithm to insert a new node after a node that has value NUM

Step 1: IF AVAIL = NULL

Write OVERFLOW

Go to Step 12 [END OF IF]

Step 2: SET = AVAIL

Step 3: SET AVAIL = AVAIL-->NEXT

Step 4: SET DATA = VAL

Step 5: SET PTR = START

Step 6: SET PREPTR = PTR

Step 7: Repeat Steps 8 and 9 while != NUM

Step 8: SET PREPTR = PTR

Step 9: SET PTR = PTR -->NEXT

[END OF LOOP]

Step 10 : PREPTR--> NEXT = NEW_NODE

Step 11: SET NEW_NODE NEXT = PTR

Step 12: EXIT

Algorithm to insert a new node before a node that has value NUM

Step 1: IF AVAIL = NULL

Write OVERFLOW

Go to Step 12 [END OF IF]

Step 2: SET = AVAIL

Step 3: SET AVAIL = AVAIL-->NEXT

Step 4: SET DATA = VAL

Step 5: SET PTR = START

Step 6: SET PREPTR = PTR

Step 7: Repeat Steps 8 and 9 while PTR DATA != NUM

Step 8: SET PREPTR = PTR

Step 9: SET PTR = PTR -->NEXT

[END OF LOOP]

Step 10: PREPTR-->NEXT = NEW_NODE

Step 11: SET NEXT = PTR

Step 12: EXIT

Algorithm to delete the first node

Step 1: IF START = NULL

Write UNDERFLOW

Go to Step 5 [END OF IF]

Step 2: SET PTR = START

Step 3: SET START = START -->NEXT

Step 4: FREE PTR

Step 5: EXIT

Algorithm to delete the last node

Step 1: IF START = NULL

Write UNDERFLOW

Go to Step 8 [END OF IF]

Step 2: SET PTR = START

Step 3: Repeat Steps 4 and 5 while PTR NEXT != NULL

Step 4: SET PREPTR = PTR

Step 5: SET PTR = PTR -->NEXT [END OF LOOP]

Step 6: SET PREPTR-->NEXT = NULL

Step 7: FREE PTR

Step 8: EXIT

Algorithm to delete the node after a given node

Step 1: IF START = NULL

Write UNDERFLOW

Go to Step 1 [END OF IF]

Step 2: SET PTR = START

Step 3: SET PREPTR = PTR

Step 4: Repeat Steps 5 and 6 while PREPTR DATA != NUM

Step 5: SET PREPTR = PTR

Step 6: SET PTR = PTR--> NEXT

[END OF LOOP]

Step 7: SET TEMP = PTR

Step 8: SET PREPTR -->NEXT = PTR--> NEXT

Step 9: FREE TEMP

Step 10: EXIT

Code:

1.Insert at beginning

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

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

void insertAtBeginning(struct Node** head, int newData)
{
struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
if (newNode == NULL)
{
printf("Memory allocation failed.\n");
return;
}
newNode->data = newData;
newNode->next = *head;
*head = newNode;
printf("Node with data %d inserted at the beginning.\n",
```

```
newData);
}

void displayList(struct Node* node)
{
if (node == NULL) {
printf("List is empty.\n");
return;
}
printf("Linked List: ");
while (node != NULL) {
printf("%d -> ", node->data);
node = node->next;
}
printf("NULL\n");
}

void freeList(struct Node* head) {
struct Node* temp;
while (head != NULL) {
temp = head;
head = head->next;
free(temp);
}
}
```

```
int main() {
struct Node* head = NULL;
int value, choice;
while (1) {
printf("\n1. Insert at Beginning\n");
printf("2. Display List\n");
printf("3. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
printf("Enter value to insert: ");
scanf("%d", &value);
insertAtBeginning(&head, value);
break;
case 2:
displayList(head);
break;
case 3:
freeList(head);
return 0;
default:
printf("Invalid choice! Please try again.\n");
}
}
return 0;
```

}

Output:

```
1. Insert at Beginning
2. Display List
3. Exit
Enter your choice: 1
Enter value to insert: 50
Node with data 50 inserted at the beginning.

1. Insert at Beginning
2. Display List
3. Exit
Enter your choice: 1
Enter value to insert: 52
Node with data 52 inserted at the beginning.

1. Insert at Beginning
2. Display List
3. Exit
Enter your choice: 1
Enter value to insert: 54
Node with data 54 inserted at the beginning.

1. Insert at Beginning
2. Display List
3. Exit
Enter your choice: 1
Enter value to insert: 56
Node with data 56 inserted at the beginning.

1. Insert at Beginning
2. Display List
3. Exit
Enter your choice: 2
Linked List: 56 -> 54 -> 52 -> 50 -> NULL

1. Insert at Beginning
2. Display List
3. Exit
Enter your choice: 3
```

2. Insertion at end

code:

#include <stdio.h>

```
#include <stdlib.h>
#include <conio.h>
struct Node {
int data;
struct Node* next;
};
void insertAtEnd(struct Node** head, int newData) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
struct Node* last = *head;
newNode->data = newData;
newNode->next = NULL;
if (*head == NULL) {
*head = newNode;
return;
}
while (last->next != NULL)
last = last->next;
last->next = newNode;
printf("Node with data %d inserted at the end.\n",
newData);
}
void displayList(struct Node* node) {
if (node == NULL) {
printf("List is empty.\n");
return;
```

```
}
printf("Linked List: ");
while (node != NULL) {
printf("%d -> ", node->data);
node = node->next;
}
printf("NULL\n");
}
int main() {
struct Node* head = NULL;
int value;
while (1)
{
printf("\nEnter value to insert at the end (or -1 to exit): ");
scanf("%d", &value);
if (value == -1) {
break;
}
insertAtEnd(&head, value);
displayList(head);
}
return 0;
}
```

OUTPUT:

```
Enter value to insert at the end (or -1 to exit): 25
Linked List: 25 -> NULL

Enter value to insert at the end (or -1 to exit): 24
Node with data 24 inserted at the end.
Linked List: 25 -> 24 -> NULL

Enter value to insert at the end (or -1 to exit): 26
Node with data 26 inserted at the end.
Linked List: 25 -> 24 -> 26 -> NULL

Enter value to insert at the end (or -1 to exit): 82
Node with data 82 inserted at the end.
Linked List: 25 -> 24 -> 26 -> 82 -> NULL

Enter value to insert at the end (or -1 to exit): -1
```

3.Insertion in between

code:

```
#include<stdio.h>

#include<stdlib.h>



struct node {

    int data;

    struct node *next;

};



void insertInBetween(struct node* head) {

    int num, pos, i;

    struct node *temp, *p;


    temp = (struct node*)malloc(sizeof(struct node));
```

```
if (temp == NULL) {
    printf("Memory allocation failed!\n");
    return;
}

printf("Enter the value to be inserted: ");
scanf("%d", &num);
temp->data = num;

printf("Enter the position at which the element is to be inserted: ");
scanf("%d", &pos);

if (pos <= 0) {
    printf("Invalid position!\n");
    free(temp);
    return;
}

p = head;


for (i = 0; i < pos - 1 && p != NULL; i++) {
    p = p->next;
}
```

```
    if (p == NULL) {
        printf("Position out of bounds!\n");
        free(temp);
        return;
    }

    temp->next = p->next;
    p->next = temp;

    printf("Node inserted successfully!\n");
}

void displayList(struct node* head) {
    struct node *p = head;

    printf("The linked list is: ");
    while (p != NULL) {
        printf("%d -> ", p->data);
        p = p->next;
    }
    printf("NULL\n");
}

int main() {
```

```
struct node *one, *two, *three, *head;



one = (struct node*)malloc(sizeof(struct node));
two = (struct node*)malloc(sizeof(struct node));
three = (struct node*)malloc(sizeof(struct node));



one->data = 10;
one->next = two;

two->data = 20;
two->next = three;

three->data = 30;
three->next = NULL;

head = one;



displayList(head);



insertInBetween(head);
```

```
    displayList(head);



    free(one);
    free(two);
    free(three);

    return 0;
}
```

OUTPUT:

```
The linked list is: 10 -> 20 -> 30 -> NULL
Enter the value to be inserted: 78
Enter the position at which the element is to be inserted: 2
Node inserted successfully!
The linked list is: 10 -> 20 -> 78 -> 30 -> NULL
```

Conclusion:

Write an example of stack and queue implementation using singly linked list?

➔ **Stack Implementation**

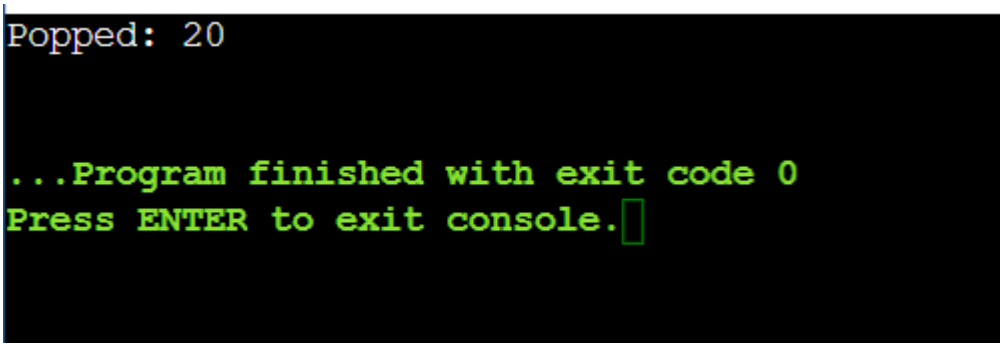
code:

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

struct Node {
    int data;
```

```
    struct Node* next;
};

struct Node* top = NULL; // top of the stack

void push(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = top;
    top = newNode;
}

int pop() {
    if (top == NULL) {
        printf("Stack is empty\n");
        return -1;
    }
    int data = top->data;
    struct Node* temp = top;
    top = top->next;
    free(temp);
    return data;
}
```

```
int isEmpty() {
    return top == NULL;
}

int main() {
    push(10);
    push(20);
    printf("Popped: %d\n", pop()); // Output: 20
    return 0;
}
```

OUTPUT:



Queue Implementation

code:

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

struct Node {
```

```
    int data;
    struct Node* next;
};

struct Node *front = NULL, *rear = NULL; // front and rear of the queue

void enqueue(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    if (rear == NULL) {
        front = rear = newNode;
        return;
    }
    rear->next = newNode;
    rear = newNode;
}

int dequeue() {
    if (front == NULL) {
        printf("Queue is empty\n");
        return -1;
    }
    int data = front->data;
    struct Node* temp = front;
    front = front->next;
    if (front == NULL) rear = NULL;
```

```
    free(temp);
    return data;
}

int isEmpty() {
    return front == NULL;
}

int main() {
    enqueue(10);
    enqueue(20);
    printf("Dequeued: %d\n", dequeue()); // Output: 10
    return 0;
}
```

OUTPUT: