Experiment No.6
Implement Singly Linked List ADT
Name:Umang Borse
Roll No:03
Date of Performance:
Date of Submission:
Marks:
Sign:

Experiment No. 6: Singly Linked List Operations

Aim: Implementation of Singly Linked List

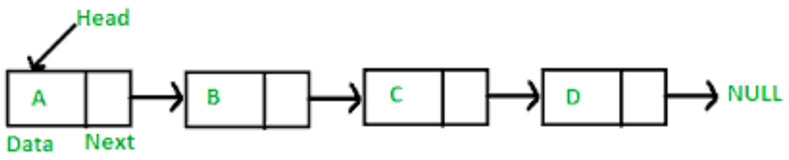
Objective:

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:

```
 #include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <malloc.h>
struct node
{
```

```
int data;

struct node *next;

};

struct node *start = NULL;

struct node *create_ll(struct node *);

struct node *display(struct node *);

struct node *insert_beg(struct node *);

struct node *insert_end(struct node *);

struct node *insert_before(struct node *);

struct node *insert_after(struct node *);

struct node *delete_beg(struct node *);

struct node *delete_end(struct node *);

struct node *delete_node(struct node *);

struct node *delete_after(struct node *);

struct node *delete_list(struct node *);

struct node *sort_list(struct node *);

int main(int argc, char *argv[]) {

int option;

do

{

 printf("\n\n ** IMPLEMENRTATION OF SINGLY LINDED LIST **");

 printf("\n 1: Create a list");

 printf("\n 2: Display the list");

 printf("\n 3: Add a node at the beginning");

 printf("\n 4: Add a node at the end");

 printf("\n 5: Add a node before a given node");

 printf("\n 6: Add a node after a given node");
 printf("\n 7: Delete a node from the beginning");
printf("\n 8: Delete a node from the end");
 printf("\n 9: Delete a given node");
 printf("\n 10: Delete a node after a given node");
 printf("\n 11: Delete the entire list");
 printf("\n 12: Sort the list");
 printf("\n 13: EXIT");
 printf("\n\n Enter your option : ");
 scanf("%d", &option);
 switch(option)
 {
 case 1: start = create_ll(start);
 printf("\n LINKED LIST CREATED");
 break;
 case 2: start = display(start);
 break;
 case 3: start = insert_beg(start);
 break;
 case 4: start = insert_end(start);
 break;
 case 5: start = insert_before(start);
 break;
 case 6: start = insert_after(start);
 break;
 case 7: start = delete_beg(start);
 break;
 case 8: start = delete_end(start);
```

```
 break;
 case 9: start = delete_node(start);
 break;
 case 10: start = delete_after(start);
 break;
 case 11: start = delete_list(start);
 printf("\n LINKED LIST DELETED");
 break;
 case 12: start = sort_list(start);
 break;
 }
}while(option !=13);
getch();
return 0;
}
struct node *create_ll(struct node *start)
{
struct node *new_node, *ptr;
int num;
printf("\n Enter -1 to end");
printf("\n Enter the data : ");
scanf("%d", &num);
while(num!=-1)
{
 new_node = (struct node*)malloc(sizeof(struct node));
 new_node -> data=num;
 if(start==NULL)
 {
 new_node -> next = NULL;
 start = new_node;
 }
 else
 {
 ptr=start;
while(ptr->next!=NULL)
 ptr=ptr->next;
 ptr->next = new_node;
 new_node->next=NULL;
 }
 printf("\n Enter the data : ");
 scanf("%d", &num);
}
return start;
}
struct node *display(struct node *start)
{
struct node *ptr;
ptr = start;
while(ptr != NULL)
{
 printf("\t %d", ptr -> data);
 ptr = ptr -> next;
}
return start;
```

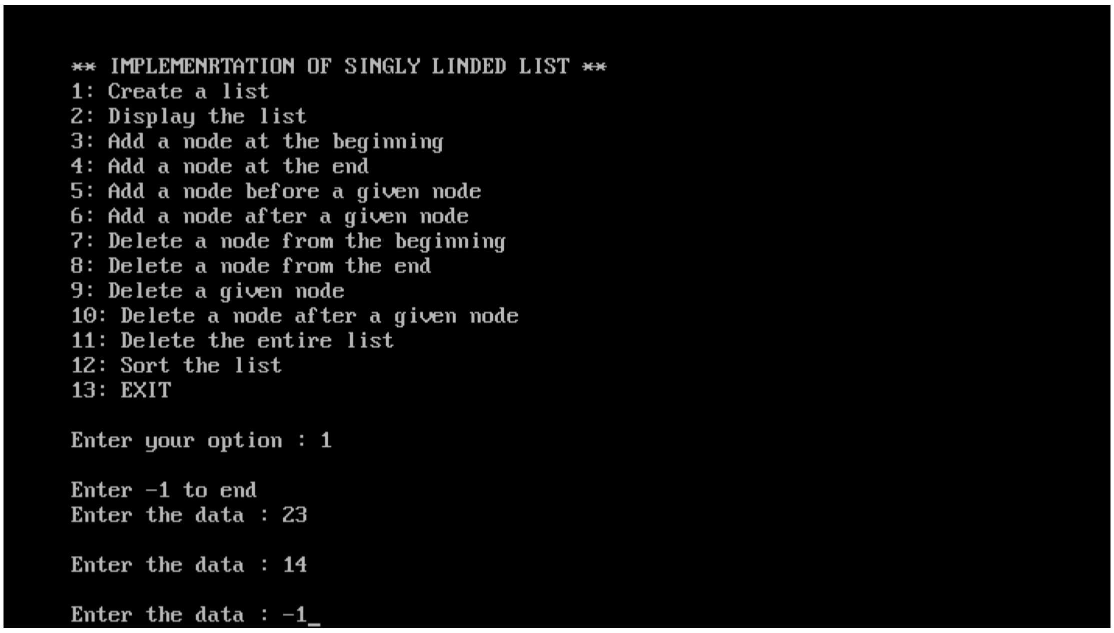
```
}

struct node *insert_beg(struct node *start)

{

struct node *new_node;

int num;

printf("\n Enter the data : ");

scanf("%d", &num);

new_node = (struct node *)malloc(sizeof(struct node));

new_node -> data = num;

new_node -> next = start;

start = new_node;

return start;

}

struct node *insert_end(struct node *start)

{

struct node *ptr, *new_node;

int num;

printf("\n Enter the data : ");

scanf("%d", &num);

new_node = (struct node *)malloc(sizeof(struct node));

new_node -> data = num;

new_node -> next = NULL;

ptr = start;

while(ptr -> next != NULL)

ptr = ptr -> next;

ptr -> next = new_node;

return start;
```

```
}

struct node *insert_before(struct node *start)

{

struct node *new_node, *ptr, *preptr;

int num, val;

printf("\n Enter the data : ");

scanf("%d", &num);

printf("\n Enter the value before which the data has to be inserted : ");

scanf("%d", &val);

new_node = (struct node *)malloc(sizeof(struct node));

new_node -> data = num;

ptr = start;

while(ptr -> data != val)

{

preptr = ptr;

 ptr = ptr -> next;

}

preptr -> next = new_node;

new_node -> next = ptr;

return start;

}

struct node *insert_after(struct node *start)

{

struct node *new_node, *ptr, *preptr;

int num, val;

printf("\n Enter the data : ");

scanf("%d", &num);
```

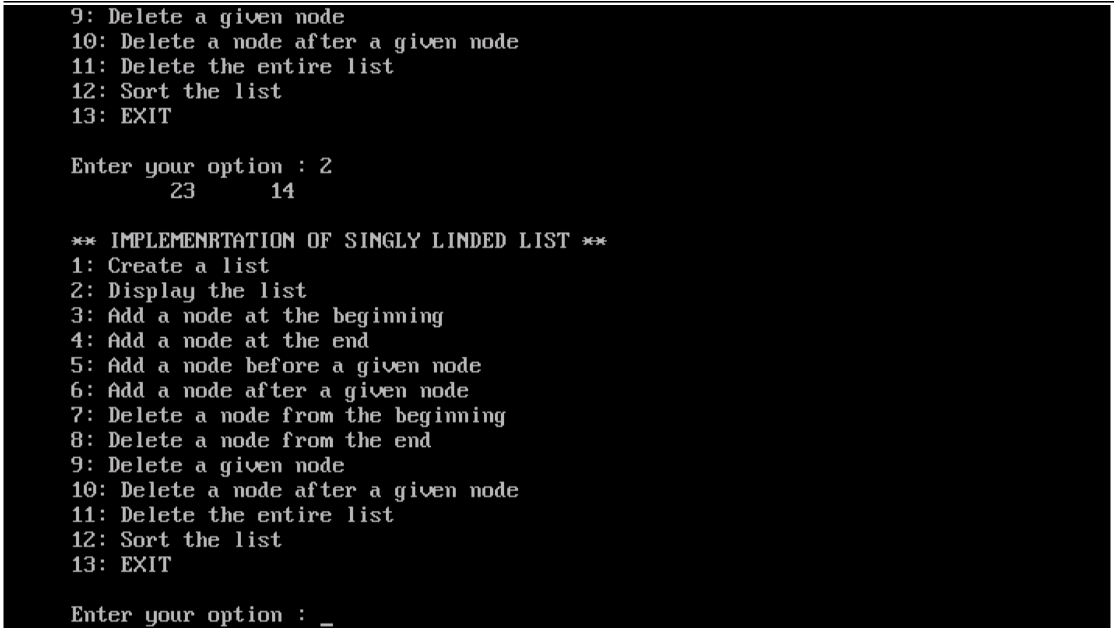
```
printf("\n Enter the value after which the data has to be inserted : ");

scanf("%d", &val);

new_node = (struct node *)malloc(sizeof(struct node));

new_node -> data = num;

ptr = start;

preptr = ptr;

while(preptr -> data != val)

{

 preptr = ptr;

 ptr = ptr -> next;

}

preptr -> next=new_node;

new_node -> next = ptr;

return start;

}

struct node *delete_beg(struct node *start)

{

struct node *ptr;

ptr = start;

start = start -> next;

free(ptr);

return start;

}

struct node *delete_end(struct node *start)

{

struct node *ptr, *preptr;

ptr = start;
```

```
while(ptr -> next != NULL)
{
 preptr = ptr;
 ptr = ptr -> next;
}
preptr -> next = NULL;
free(ptr);
return start;
}
struct node *delete_node(struct node *start)
{
struct node *ptr, *preptr;
int val;
printf("\n Enter the value of the node which has to be deleted : ");
scanf("%d", &val);
ptr = start;
if(ptr -> data == val)
{
 start = delete_beg(start);
 return start;
}
else
{
while(ptr -> data != val)
 {
 preptr = ptr;
 ptr = ptr -> next;
 }
 preptr -> next = ptr -> next;
 free(ptr);
 return start;
}
}
struct node *delete_after(struct node *start)
{
struct node *ptr, *preptr;
int val;
printf("\n Enter the value after which the node has to deleted : ");
scanf("%d", &val);
ptr = start;
preptr = ptr;
while(preptr -> data != val)
{
 preptr = ptr;
 ptr = ptr -> next;
}
preptr -> next=ptr -> next;
free(ptr);
return start;
}
struct node *delete_list(struct node *start)
{
        struct node *ptr; // Lines 252-254 were modified from original code to fix
unresposiveness in output window
if(start!=NULL){
 ptr=start;
 while(ptr != NULL)
 {
 printf("\n %d is to be deleted next", ptr -> data);
 start = delete_beg(ptr);
ptr = start;
 }
}
return start;
}
struct node *sort_list(struct node *start)
{
struct node *ptr1, *ptr2;
int temp;
ptr1 = start;
while(ptr1 -> next != NULL)
{
 ptr2 = ptr1 -> next;
 while(ptr2 != NULL)
 {
 if(ptr1 -> data > ptr2 -> data)
 {
 temp = ptr1 -> data;
 ptr1 -> data = ptr2 -> data;
 ptr2 -> data = temp;
 }
 ptr2 = ptr2 -> next;
 }

 ptr1 = ptr1 -> next;

}

return start;

 }
```

Output:



```
** IMPLEMENRTATION OF SINGLY LINDED LIST **
1: Create a list
2: Display the list
3: Add a node at the beginning
4: Add a node at the end
5: Add a node before a given node
6: Add a node after a given node
7: Delete a node from the beginning
8: Delete a node from the end
9: Delete a given node
10: Delete a node after a given node
11: Delete the entire list
12: Sort the list
13: EXIT

Enter your option : 1

Enter -1 to end
Enter the data : 23

Enter the data : 14

Enter the data : -1_
```



```
    9: Delete a given node
    10: Delete a node after a given node
    11: Delete the entire list
    12: Sort the list
    13: EXIT

    Enter your option : 2
            23      14

    ** IMPLEMENRTATION OF SINGLY LINDED LIST **
    1: Create a list
    2: Display the list
    3: Add a node at the beginning
    4: Add a node at the end
    5: Add a node before a given node
    6: Add a node after a given node
    7: Delete a node from the beginning
    8: Delete a node from the end
    9: Delete a given node
    10: Delete a node after a given node
    11: Delete the entire list
    12: Sort the list
    13: EXIT

    Enter your option : _
```

Conclusion:

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

Stack Implementation Using a Singly Linked List

```
#include <stdio.h>

#include <stdlib.h>

// Define the structure for a node in the singly linked list

struct Node {

   int data;

   struct Node* next;

};

// Define the structure for the stack

struct Stack {
    struct Node* top;

};

// Initialize an empty stack

void initStack(struct Stack* stack) {

    stack->top = NULL;

}

// Check if the stack is empty

int isEmpty(struct Stack* stack) {

    return (stack->top == NULL);

}

// Push an element onto the stack

void push(struct Stack* stack, int data) {

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

    newNode->data = data;

    newNode->next = stack->top;

    stack->top = newNode;

}

// Pop an element from the stack

int pop(struct Stack* stack) {

    if (isEmpty(stack)) {

        printf("Stack underflow\n");
        exit(1);
    }
    int data = stack->top->data;
    struct Node* temp = stack->top;
    stack->top = stack->top->next;
    free(temp);
    return data;
}
// Peek at the top element of the stack without popping
int peek(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty\n");
        exit(1);
    }
    return stack->top->data;
}
int main() {
    struct Stack stack;
    initStack(&stack);
    push(&stack, 10);
    push(&stack, 20);
    push(&stack, 30);

    printf("Top element: %d\n", peek(&stack));

    printf("Popped element: %d\n", pop(&stack));

    printf("Popped element: %d\n", pop(&stack));

    printf("Top element: %d\n", peek(&stack));

    return 0;

}
```

<u>Queue Implementation Using a Singly Linked List</u>

```
#include <stdio.h>

#include <stdlib.h>

// Define the structure for a node in the singly linked list

struct Node {

    int data;

    struct Node* next;

};

// Define the structure for the queue

struct Queue {

    struct Node* front;

    struct Node* rear;

};

// Initialize an empty queue
```

```
void initQueue(struct Queue* queue) {

    queue->front = queue->rear = NULL;

}

// Check if the queue is empty

int isEmpty(struct Queue* queue) {

    return (queue->front == NULL);

}

// Enqueue (add) an element to the rear of the queue

void enqueue(struct Queue* queue, int data) {

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

    newNode->data = data;

    newNode->next = NULL;

    if (isEmpty(queue)) {

        queue->front = queue->rear = newNode;

    } else {

        queue->rear->next = newNode;

        queue->rear = newNode;

    }

}

// Dequeue (remove) an element from the front of the queue

int dequeue(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty\n");
        exit(1);
    }
    int data = queue->front->data;
    struct Node* temp = queue->front;
    queue->front = queue->front->next;
    free(temp);
    if (queue->front == NULL) {
        queue->rear = NULL;
    }
    return data;
}
int main() {
    struct Queue queue;
    initQueue(&queue);
    enqueue(&queue, 10);
    enqueue(&queue, 20);
    enqueue(&queue, 30);
    printf("Dequeued element: %d\n", dequeue(&queue));
    printf("Dequeued element: %d\n", dequeue(&queue));
    enqueue(&queue, 40);

    printf("Dequeued element: %d\n", dequeue(&queue));

    printf("Dequeued element: %d\n", dequeue(&queue));

    return 0;

}
```

These examples demonstrate the implementation of a stack and a queue using a singly linked list in C. The stack follows the Last-In-First-Out (LIFO) principle, while the queue follows the First-In-First-Out (FIFO) principle.