Experiment No.6
Aim: Implementation of Singly Linked List
Name: Nikhil Kamalaji Shingade
Roll no: 57
Date of Performance:
Date of Submission:

Experiment No. 6: Singly Linked List Operations

Aim: Implementation of Singly Linked List

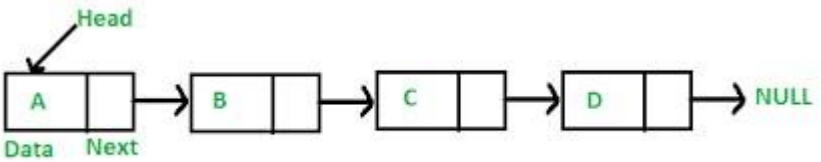
Objective:

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<malloc.h>

struct node
{
int data;
struct node *next;
};
void insertAB(struct node *head)
{
int num;
struct node *temp,*p;
temp=(struct node*)malloc(sizeof(struct node));
printf("enter a number");
scanf("%d",&num);
temp->data=num;
temp->next=head;
head=temp;
printf("content of Linked list\n");
p=head;
while(p!=NULL)
{
printf("\n%d",p->data);
p=p->next;
}
}
```

```
void insertAtEnd(struct node *head)
{
   int num;
   struct node *p=head;
   struct node *temp;
   temp=(struct node *)malloc(sizeof(struct node));
   printf("\nenter number");
   scanf("%d",&num);
   temp->data=num;
   temp->next=NULL;
   while(p->next!=NULL)
   {
      p=p->next;

   }

      p->next=temp;


printf("content of Linked list\n");
struct node *q;
q=head;
while(q!=NULL)
{
printf("\n%d",q->data);
q=q->next;
}
}
void search(struct node *head)
{
 int num;
 struct node *p=head;
 int r=0;
 printf("\nenter number to be searched");
 scanf("\n%d",&num);
 while(p!=NULL)
 {
    if(p->data==num)
    {
       r=1;
       break;
    }
    else
    {p=p->next;

    }
    }
 if(r==0)
```

```
 {
    printf("no is absent");

 }
 else
 printf("no is present");
}
void InBetween(struct node *head)
{
   int num,pos,i;
   struct node *temp,*p;
    temp=(struct node *)malloc(sizeof(struct node));
   printf("\nenter number");
   scanf("%d",&num);
   temp->data=num;
   printf("enter the pos");
   scanf("%d",&pos);
   p=head;
   for(i=1;i<pos-1;i++)
   {
      p=p->next;

   }
   temp->next=p->next;
   p->next=temp;
   printf("content of Linked list\n");
p=head;
while(p!=NULL)
{
printf("\n%d",p->data);
p=p->next;
}
}
 void main()
 {
 struct node n3={30,NULL};
 struct node n2={20,&n3};
 struct node n1={10,&n2};
 struct node *head=&n1;

insertAtEnd(head);
search(head);
InBetween(head);
}
```

Output:

```
Output

/tmp/sL4xFNT2YI.o

enter number40
content of Linked list

10
20
30
40
enter number to be searched20
no is present
enter number50
enter the pos4
content of Linked list

10
20
30
50
40

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

Conclusion:

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

Stack Implementation

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

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

struct Stack {
```

```
    struct Node* top;
};

struct Stack* createStack() {
    struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
    stack->top = NULL;
    return stack;
}

int isEmpty(struct Stack* stack) {
    return stack->top == NULL;
}

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;
}

int pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack Underflow\n");
        return -1;
    }
    struct Node* temp = stack->top;
    stack->top = stack->top->next;
    int poppedData = temp->data;
    free(temp);
    return poppedData;
}

int peek(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty\n");
        return -1;
    }
    return stack->top->data;
}

int main() {
    struct Stack* stack = createStack();
    push(stack, 1);
    push(stack, 2);
    printf("Popped: %d\n", pop(stack));
    printf("Top element: %d\n", peek(stack));
    return 0;
}
```

Queue Implementation

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

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

struct Queue {
   struct Node *front, *rear;
};

struct Queue* createQueue() {
   struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
   queue->front = queue->rear = NULL;
   return queue;
}

int isEmpty(struct Queue* queue) {
   return queue->front == NULL;
}

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

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

```
int main() {
    struct Queue* queue = createQueue();
    enqueue(queue, 1);
    enqueue(queue, 2);
    printf("Dequeued: %d\n", dequeue(queue));
    printf("Dequeued: %d\n", dequeue(queue));
    return 0;
}
```