Experiment no 6:

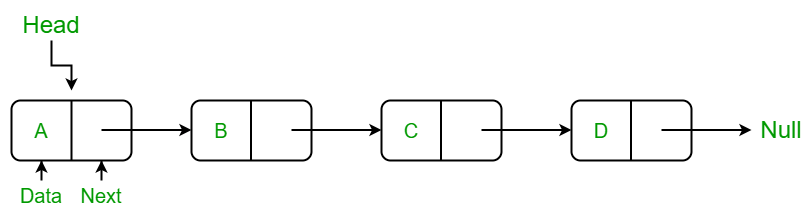
Name: Aryan Vanjari Roll no.: 60 Batch: C SEM:III

Aim: Implementation of Singly Linked List

Objective : It is used to implement stacks and queue which are linked needs throughout computer science .To prevent the Collision between the data in the Hash map.we use a singly Linked list

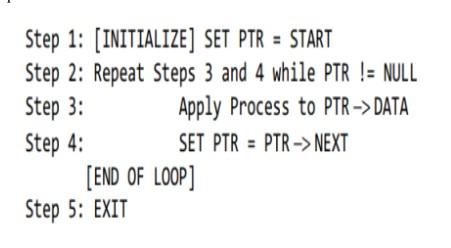
Theory:

A singly linked list is a linear data structure in which the elements are not stored in contiguous memory locations and each element is connected only to its next element using a pointer.



Algorithm:

Algorithm for traversing a linked list



Inserting a node at the beginning

Step 1: SET NEW\_NODE = PTR

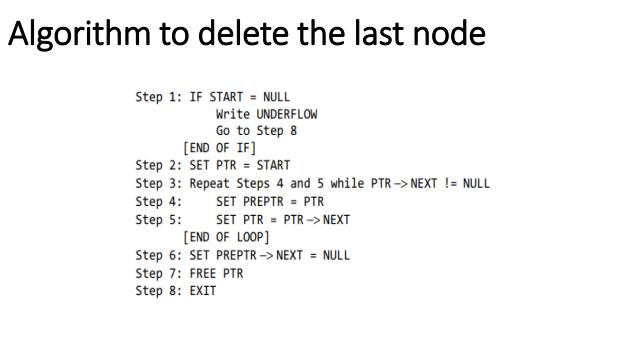
Step 2: SET PTR = PTR → NEXT

Step 3: SET NEW\_NODE → DATA = VAL

Step 4: SET NEW\_NODE → NEXT = HEAD

Step 5: SET HEAD = NEW\_NODE

Step 6: EXIT



Code:

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*head;

void beginsert ();

void lastinsert ();

void randominsert();

void begin\_delete();

void last\_delete();

void random\_delete();

void display();

void search();

void main ()

{

int choice =0;

while(choice != 9)

{

printf("\n1. Insert in begining\n2. Insert at last\n3. Delete from Beginning\n4. Delete from last\n5. Display\n6. Count\n7. Exit\n");

printf("\nEnter your choice?\n");

scanf("\n%d",&choice);

switch(choice)

{

case 1:

beginsert();

break;

case 2:

lastinsert();

break;

case 3:

begin\_delete();

break;

case 4:

last\_delete();

break;

case 5:

display();

break;

case 6:

count();

break;

case 7:

exit(0);

break;

default:

printf("Please enter valid choice..");

}

}

}

void beginsert()

{

struct node \*ptr;

int item;

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

if(ptr == NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter value\n");

scanf("%d",&item);

ptr->data = item;

ptr->next = head;

head = ptr;

printf("\nNode inserted");

}

}

void lastinsert()

{

struct node \*ptr,\*temp;

int item;

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

if(ptr == NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter value?\n");

scanf("%d",&item);

ptr->data = item;

if(head == NULL)

{

ptr -> next = NULL;

head = ptr;

printf("\nNode inserted");

}

else

{

temp = head;

while (temp -> next != NULL)

{

temp = temp -> next;

}

temp->next = ptr;

ptr->next = NULL;

printf("\nNode inserted");

}

}

}

void begin\_delete()

{

struct node \*ptr;

if(head == NULL)

{

printf("\nList is empty\n");

}

else

{

ptr = head;

head = ptr->next;

free(ptr);

printf("\nNode deleted from the begining ...\n");

}

}

void last\_delete()

{

struct node \*ptr,\*ptr1;

if(head == NULL)

{

printf("\nlist is empty");

}

else if(head -> next == NULL)

{

head = NULL;

free(head);

printf("\nOnly node of the list deleted ...\n");

}

else

{

ptr = head;

while(ptr->next != NULL)

{

ptr1 = ptr;

ptr = ptr ->next;

}

ptr1->next = NULL;

free(ptr);

printf("\nDeleted Node from the last ...\n");

}

}

void display()

{

struct node \*ptr;

ptr = head;

if(ptr == NULL)

{

printf("Nothing to print");

}

else

{

printf("\nprinting values . . . . .\n");

while (ptr!=NULL)

{

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

ptr = ptr -> next;

}

}

}

void count()

{

int count=0;

struct node \*ptr;

ptr = head;

if(ptr == NULL)

{

printf("Nothing to count");

}

else

{

while (ptr!=NULL)

{

ptr = ptr -> next;

count++;

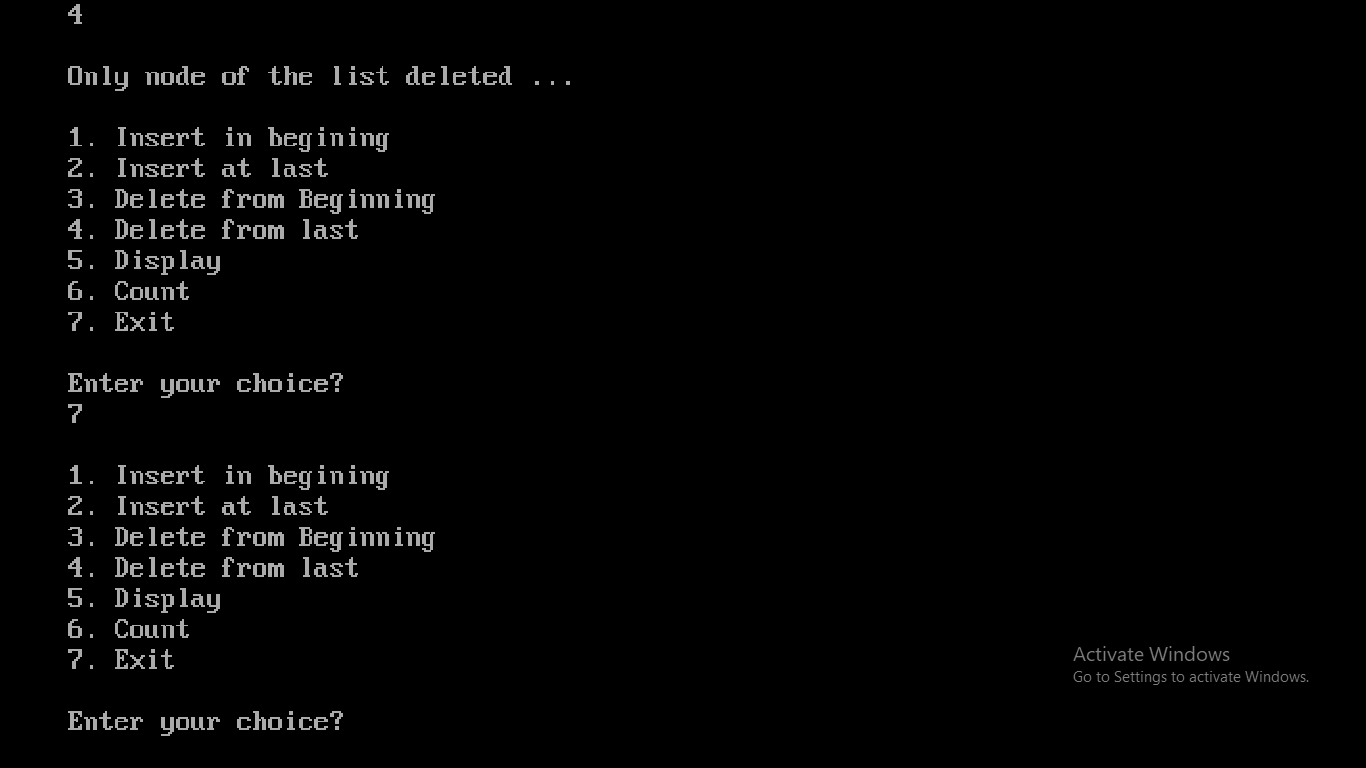
}

printf("The count is %d", count);

}

}

Output:



Conclusion : Therefore, clearly it has the beginning and the end. the main problem which comes with this list is that we cannot access the predecessor of the node from the current node. therefore, we can say that a singly linked list is a dynamic data structure because it may shrink or grow. hence, the shrinking and growing depending on the operation made.