|  |  |
| --- | --- |
|  | SVKM’s NMIMS  School of Technology Management & Engineering Navi Mumbai Campus |
| Department of Computer Engineering |

|  |  |
| --- | --- |
| Name: Yash Patil | SAP ID: 70472200140 |
| Semester: III | Year: II |
| Subject: Data Structures and Algorithm | Roll No.: A176 |
| Practical: 5 | Date: 28/08/2023 |
| Batch: 1 |  |

# Aim:–

To study and implement the concept of Linked list data structure.

# Theory:–

Linked List is a linear data structure, in which elements are not stored at a contiguous location, rather they are linked using pointers. Linked List forms a series of connected nodes, where each node stores the data and the address of the next node.

Node Structure: A node in a linked list typically consists of two components: Data: It holds the actual value or data associated with the node.

Next Pointer: It stores the memory address (reference) of the next node in the sequence.

Head and Tail: The linked list is accessed through the head node, which points to the first node in the list. The last node in the list points to NULL or nullptr, indicating the end of the list. This node is known as the tail node.

Why linked list data structure needed?

Here are a few advantages of a linked list that is listed below, it will help you understand why it is necessary to know.

Dynamic Data structure: The size of memory can be allocated or de-allocated at run time based on the operation insertion or deletion.

Ease of Insertion/Deletion: The insertion and deletion of elements are simpler than arrays since no elements need to be shifted after insertion and deletion, Just the address needed to be updated.

Efficient Memory Utilization: As we know Linked List is a dynamic data structure the size increases or decreases as per the requirement so this avoids the wastage of memory.

Implementation: Various advanced data structures can be implemented using a linked list like a stack, queue, graph, hash maps, etc.

[Types of linked lists](https://www.geeksforgeeks.org/types-of-linked-list/):

There are mainly three types of linked lists:

1. Single-linked list 2.Double linked list 3.Circular linked list

Operations on Linked Lists

-> [Insertion](https://www.geeksforgeeks.org/insertion-in-linked-list/): Adding a new node to a linked list involves adjusting the pointers of the existing nodes to maintain the proper sequence. Insertion can be performed at the beginning, end, or any position within the list

->[Deletion](https://www.geeksforgeeks.org/deletion-in-linked-list/): Removing a node from a linked list requires adjusting the pointers of the neighboring nodes to bridge the gap left by the deleted node. Deletion can be performed at the beginning, end, or any position within the list.

->[Searching](https://www.geeksforgeeks.org/search-an-element-in-a-linked-list-iterative-and-recursive/): Searching for a specific value in a linked list involves traversing the list from the head node until the value is found or the end of the list is reached.

Advantages of Linked Lists

* + Dynamic Size: Linked lists can grow or shrink dynamically, as memory allocation is done at runtime.
  + Insertion and Deletion: Adding or removing elements from a linked list is efficient, especially for large lists.
  + Flexibility: Linked lists can be easily reorganized and modified without requiring a contiguous block of memory.

Disadvantages of Linked Lists

* + Random Access: Unlike arrays, linked lists do not allow direct access to elements by index. Traversal is required to reach a specific node.
  + Extra Memory: Linked lists require additional memory for storing the pointers, compared to arrays.

# Code:-

**SINGLY LINKED LIST**

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

struct Node

{

int data;

struct Node \*next;

};

struct Node \*start=NULL;//intial list

void addBegining(int value1)

{

struct Node \*new1=(struct Node \*)malloc(sizeof(struct Node)); new1->data=value1;

new1->next=NULL;

if(start==NULL)

{

start = new1;

}

else

{

new1->next=start;//storing memory location of start in new node start = new1;//making statring node as node

}

}

void addEnd(int value2)

{

struct Node \*new2=(struct Node \*)malloc(sizeof(struct Node)); new2->data=value2;

new2->next=NULL;

if(start==NULL)

{

start = new2;

}

else

{

struct Node \*last = start; while(last->next != NULL)

{

last = last->next;

}

last->next = new2;

}

}

void addSpecific(int value3,int loc)

{

struct Node \*new3=(struct Node \*)malloc(sizeof(struct Node)); new3->data=value3;

new3->next=NULL;

if(start==NULL)

{

start = new3;

}

else

{

struct Node \*temp = start; for(int i=1;i<=loc-1;i++)

{

temp = temp->next;

}

if (temp==NULL)

{

printf("There are no sufficient nodes");

}

else

{

new3->next = temp->next; temp->next = new3;

}

}

}

void deleteB()

{

struct Node \*temp = start; start = start->next;

printf("Deleted Element:%d",temp->data); free(temp);

}

void deleteE()

{

struct Node \*temp = start; while(temp->next->next!=NULL)

{

temp = temp->next;

}

printf("Deleted Element:%d",temp->next->data); free(temp->next);

temp->next = NULL;

}

void deleteS(int loc)

{

struct Node \*temp1 = start; for(int i=1;i<=loc-1;i++)

{

temp1 = temp1->next;

}

struct Node \*temp2 = temp1->next; temp1->next = temp2->next;

printf("Deleted Element:%d",temp2->data); free(temp2);

}

void reverse()

{

struct Node \*curr = start; struct Node \*prev = NULL; struct Node \*next1 = NULL;

while(curr!=NULL)

{

next1 = curr->next; curr->next=prev; prev = curr;

curr = next1;

}

start = prev;

}

void search(int val)

{

struct Node \*temp = start; while(temp!=NULL)

{

if(val==temp->data)

{

printf("\n node found."); break;

}

temp = temp->next;

}

if(temp==NULL) printf("\n not found");

}

void display()

{

struct Node \*temp = start; printf("\nLINKED LIST START=>\n"); while(temp!=NULL)

{

printf("%d ==> ",temp->data); temp=temp->next;

}

printf("NULL");

}

void main()

{

int ch,value1,value2,value3,loc1,loc2,val; do

{

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

printf("\n1.ADD AT BEGINNING\n2.ADD AT END\n3.ADD AT SPECIFIC\n4.DELETE FROM BEGINNING\n5.DELETE FROM END\n6.DELETE FROM SPECIFIC LOCATION\n7.SEARCH\n8.REVERSE\n9.DISPLAY\n");

scanf("%d",&ch); switch(ch)

{

case 1:

printf("\nEnter Value=\t\n"); scanf("%d",&value1); addBegining(value1);

break;

case 2:

printf("\nEnter Value=\t\n"); scanf("%d",&value2); addEnd(value2);

break;

case 3:

printf("Enter location after which you want to attach node:"); scanf("%d",&loc1);

printf("\nEnter Value=\t\n"); scanf("%d",&value3); addSpecific(value3,loc1); break;

case 4:

deleteB(); break;

case 5:

deleteE(); break;

case 6:

printf("Enter location after which you want to attach node:"); scanf("%d",&loc2);

deleteS(loc2); break;

case 7:

printf("Enter value:"); scanf("%d",&val); search(val);

break;

case 8:

reverse(); break;

case 9:

display(); break;

}

}while(ch<=9);

}

# DOUBLY LINKED LIST

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

struct Node

{

int data;

struct Node \*next;

struct Node \*prev;

};

struct Node \*start=NULL;//intial list

void addBegining(int value1)

{

struct Node \*new1 = (struct Node \*)malloc(sizeof(struct Node)); new1->data = value1;

new1->next = NULL; new1->prev = NULL;

if(start==NULL)

{

start = new1;

}

else

{

new1->next = start;//storing memory location of start in new node start->prev = new1;

start = new1;//making statring node as node

}

}

void addEnd(int value2)

{

struct Node \*new2=(struct Node \*)malloc(sizeof(struct Node)); new2->data=value2;

new2->next=NULL;

new2->prev=NULL;

if(start==NULL)

{

start = new2;

}

else

{

struct Node \*temp = start; while(temp->next != NULL)

{

temp = temp->next;

}

new2->prev = temp; temp->next = new2;

}

}

void addSpecific(int value3,int loc)

{

struct Node \*new3=(struct Node \*)malloc(sizeof(struct Node)); new3->data=value3;

new3->next=NULL;

if(start==NULL)

{

start = new3;

}

else

{

struct Node \*temp = start; while(loc-2!=0)

{

temp = temp->next; loc--;

}

if (temp==NULL)

{

printf("There are no sufficient nodes");

}

else

{

new3->prev = temp;

new3->next = temp->next; temp->next->prev=new3; temp->next = new3;

}

}

}

void deleteB()

{

if(start==NULL)

{

printf("List is empty!!");

}

else

{

start = start->next;

printf("Deleted Element:%d",start->prev->data); free(start);

start->prev = NULL;

}

}

void deleteE()

{

if(start==NULL)

{

printf("List is empty!!");

}

struct Node \*temp = start; while(temp->next->next!=NULL)

{

temp = temp->next;

}

printf("Deleted Element:%d",temp->next->data); free(temp->next);

temp->next = NULL;

}

void deleteS(int loc)

{

struct Node \*temp1 = start; for(int i=1;i<loc-1;i++)

{

temp1 = temp1->next;

}

struct Node \*temp2 = temp1->next; temp1->next = temp2->next;

printf("Deleted Element:%d",temp2->data); free(temp2);

}

void display()

{

struct Node \*temp = start; printf("\nLINKED LIST START=>\n"); while(temp!=NULL)

{

printf("%d ==> ",temp->data); temp=temp->next;

}

printf("NULL");

}

void main()

{

int ch,value1,value2,value3,loc1,loc2; do

{

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

printf("\n1.ADD AT BEGINNING\n2.ADD AT END\n3.ADD AT SPECIFIC\n4.DELETE FROM BEGINNING\n5.DELETE FROM END\n6.DELETE FROM SPECIFIC LOCATION\n7.DISPLAY\n");

scanf("%d",&ch); switch(ch)

{

case 1:

printf("\nEnter Value=\t\n"); scanf("%d",&value1); addBegining(value1); break;

case 2:

printf("\nEnter Value=\t\n"); scanf("%d",&value2); addEnd(value2);

break;

case 3:

printf("Enter location after which you want to attach node:"); scanf("%d",&loc1);

printf("\nEnter Value=\t\n"); scanf("%d",&value3);

addSpecific(value3,loc1); break;

case 4:

deleteB(); break;

case 5:

deleteE(); break;

case 6:

printf("Enter location after which you want to attach node:"); scanf("%d",&loc2);

deleteS(loc2); break;

case 7:

display(); break;

}

}while(ch<=7);

}

# Conclusion:-

Linked lists are versatile data structures that provide dynamic memory allocation and efficient insertion and deletion operations. Understanding the basics of linked lists is essential for any programmer or computer science enthusiast. With this knowledge, you can implement linked lists to solve various problems and expand your understanding of data structures and algorithms.

**Outcome: -** Identified and applied appropriate linked data structure for the given problem