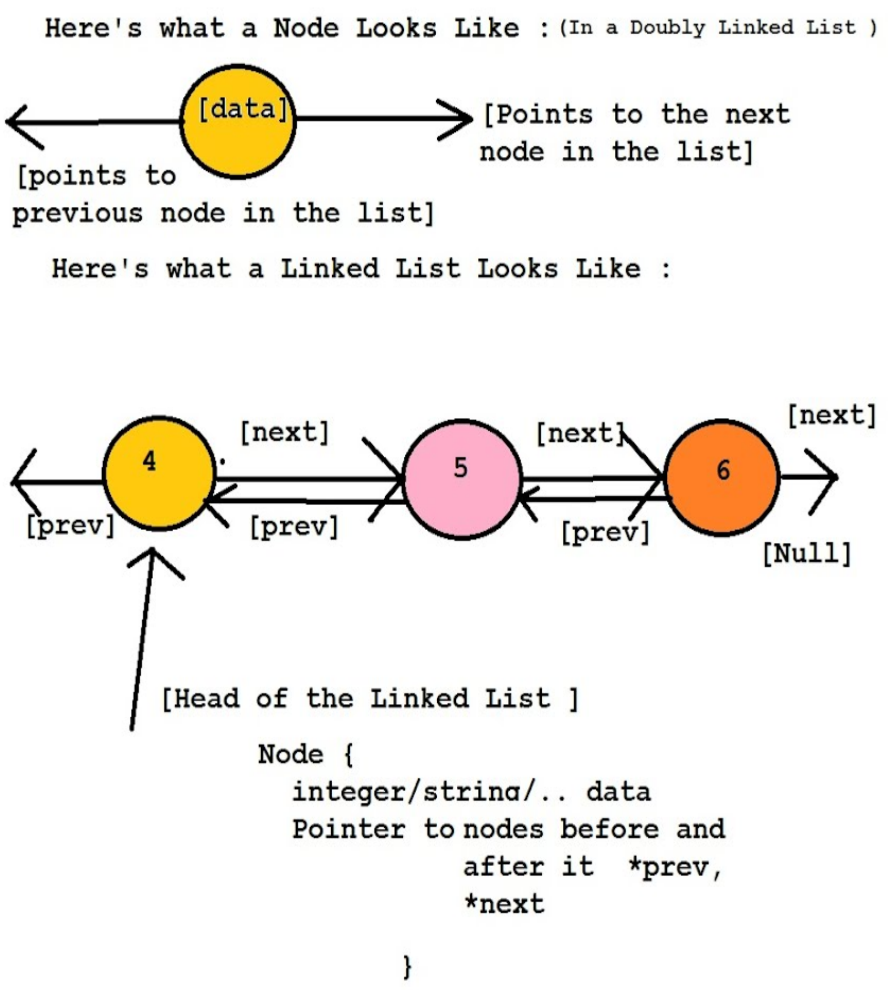
Data Structures: Doubly Linked List (with C Program source code)

Quick Links to Various Sections on the Main Page

- The Learning Point
- Computer Science
- Algorithms: An introduction to Dynamic Programming
- CS - Data Structures and Algorithms
- CS - Programming Interviews
- CS - Miscellaneous C Programs
- CS - Intro to DB Queries
- Electrical Science and Engineering
- Electrical Sci - DC Circuits
- Electrical Sci - Digital Electronics
- Mathematics
- Mathematics - Linear Algebra
- Mathematics - Geometry
- Mathematics - Single Variable Calculus
- Mathematics - Game Theory
- Mathematics - Operations Research
- Physics
- Physics - Basic Mechanics
- Physics - Electrostatics
- Sitemap
- Recent site activity

These are Quick Links to sections of the Main Page

2



To go through the C program / source-code, scroll down this page

Doubly Linked List

Doubly-linked list is a more sophisticated form of linked list data structure. Each node of the list contain two references (or links) – one to the previous node and other to the next node. The previous link of the first node and the next link of the last node points to NULL. In comparison to singly-linked list, doubly-linked list requires handling of more pointers but less information is required as one can use the previous links to observe the preceding element. It has a dynamic size, which can be determined only at run time.

Related Tutorials :

Single Linked List	A self referential data structure. A list of elements, with a head and a tail; each element points to another of its own kind.	Double Linked List	A self referential data structure. A list of elements, with a head and a tail; each element points to another of its own kind in front of it, as well as another of its own kind, which happens to be behind it in the sequence.	Circular Linked List	Linked list with no head and tail - elements point to each other in a circular fashion.

Performance

1. The advantage of a doubly linked list is that we don’t need to keep track of the previous node for traversal or no need of traversing the whole list for finding the previous node.
2. The disadvantage is that more pointers needs to be handled and more links need to updated.

Doubly-Linked List

Doubly-linked list is a more sophisticated form of linked list data structure. Each node of the list contain two references (or links) – one to the previous node and other to the next node. The previous link of the first node and the next link of the last node points to NULL. In comparison to singly-linked list, doubly-linked list requires handling of more pointers but less information is required as one can use the previous links to observe the preceding element. It has a dynamic size, which can be determined only at run time.

Basic operations of a singly-linked list are:

1. Insert – Inserts a new element at the end of the list.
2. Delete – Deletes any node from the list.
3. Find – Finds any node in the list.
4. Print – Prints the list.

Algorithm:

The node of a linked list is a structure with fields data (which stored the value of the node), *previous (which is a pointer of type node that stores the address of the previous node) and *next (which is a pointer of type node that stores the address of the next node).

Two nodes *start (which always points to the first node of the linked list) and *temp (which

Double Linked List - C Program source code

```
#include<stdio.h>
#include<stdlib.h>
typedef struct Node
```

```
{
        int data;
        struct Node *next;
        struct Node *prev;
}node;
void insert(node *pointer, int data)
{
        /* Iterate through the list till we encounter the last node.*/
        while(pointer->next!=NULL)
        {
                pointer = pointer -> next;
        }
        /* Allocate memory for the new node and put data in it.*/
        pointer->next = (node *)malloc(sizeof(node));
        (pointer->next)->prev = pointer;
        pointer = pointer->next;
        pointer->data = data;
        pointer->next = NULL;
}
int find(node *pointer, int key)
{
        pointer =  pointer -> next; //First node is dummy node.
        /* Iterate through the entire linked list and search for the key. */
        while(pointer!=NULL)
        {
                if(pointer->data == key) //key is found.
                {
                        return 1;
                }
                pointer = pointer -> next;//Search in the next node.
        }
        /*Key is not found */
        return 0;
}
void delete(node *pointer, int data)
{
        /* Go to the node for which the node next to it has to be deleted */
        while(pointer->next!=NULL && (pointer->next)->data != data)
        {
                pointer = pointer -> next;
        }
        if(pointer->next==NULL)
        {
                printf("Element %d is not present in the list\n",data);
                return;
        }
        /* Now pointer points to a node and the node next to it has to be removed */
        node *temp;
        temp = pointer -> next;
        /*temp points to the node which has to be removed*/
        pointer->next = temp->next;
        temp->prev =  pointer;
        /*We removed the node which is next to the pointer (which is also temp) */
        free(temp);
        /* Beacuse we deleted the node, we no longer require the memory used for it .
           free() will deallocate the memory.
         */
        return;
}
void print(node *pointer)
{
        if(pointer==NULL)
        {
                return;
        }
        printf("%d ",pointer->data);
        print(pointer->next);
}
int main()
{
        /* start always points to the first node of the linked list.
           temp is used to point to the last node of the linked list.*/
        node *start,*temp;
        start = (node *)malloc(sizeof(node));
        temp = start;
        temp -> next = NULL;
        temp -> prev = NULL;
```

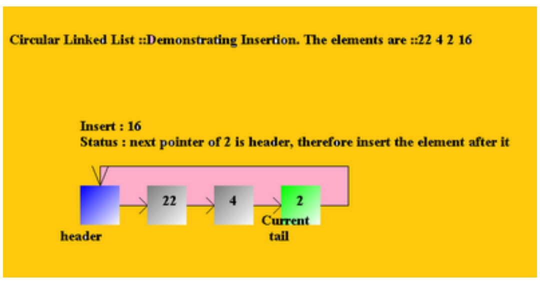
```
        /* Here in this code, we take the first node as a dummy node.
           The first node does not contain data, but it used because to avoid handling special cases
           in insert and delete functions.
         */
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Print\n");
        printf("4. Find\n");
        while(1)
        {
                int query;
                scanf("%d",&query);
                if(query==1)
                {
                        int data;
                        scanf("%d",&data);
                        insert(start,data);
                }
                else if(query==2)
                {
                        int data;
                        scanf("%d",&data);
                        delete(start,data);
                }
                else if(query==3)
                {
                        printf("The list is ");
                        print(start->next);
                        printf("\n");
                }
                else if(query==4)
                {
                        int data;
                        scanf("%d",&data);
                        int status = find(start,data);
                        if(status)
                        {
                                printf("Element Found\n");
                        }
                        else
                        {
                                printf("Element Not Found\n");

                        }
                }
        }


}
```

Related Visualizations (Java Applet Visualizations for different kinds of Linked Lists) :

Lists : Linear data structures, contain elements, each of which point to the "next" in the sequence as demonstrated in the examples below (Simple, Circular and Double Linked Lists are some common kinds of lists) . Additions and removals can be made at any point in the list - in this way it differs from stacks and queues.



1. Simple Linked Lists - A Java Applet Visualization

2. Circular Linked Lists - A Java Applet Visualization

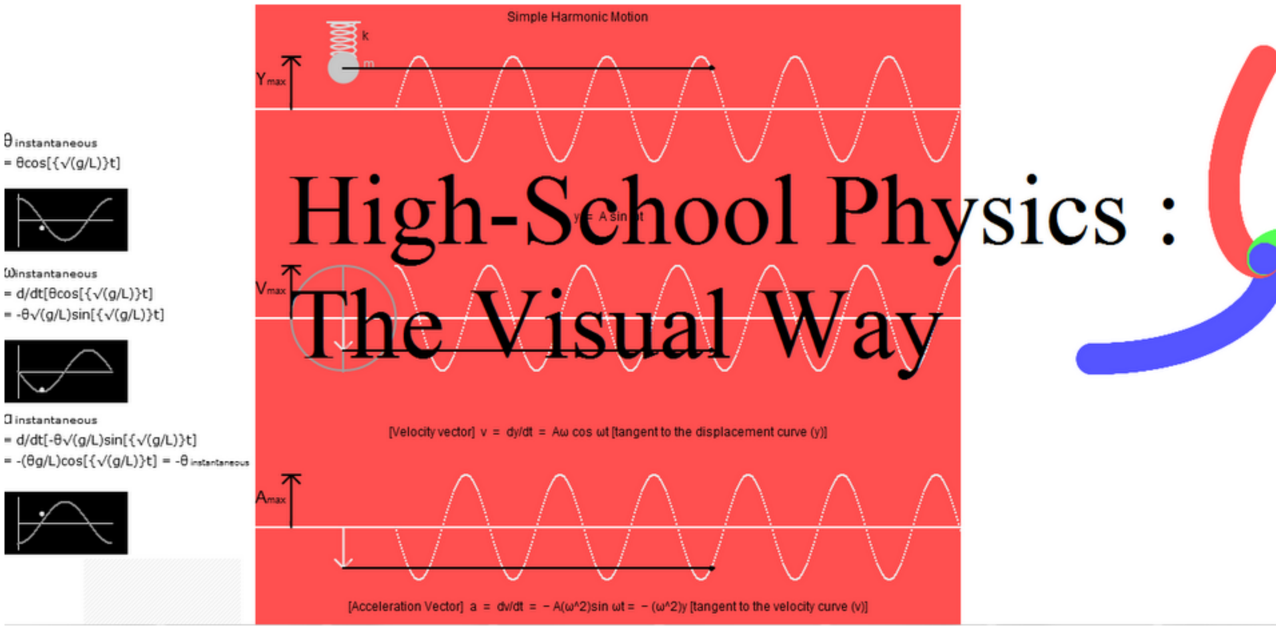
3. Double Linked Lists - A Java Applet Visualization

Some Important Data Structures and Algorithms, at a glance:

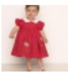
Arrays : Popular Sorting and Searching Algorithms			
Bubble Sort	Insertion Sort	Selection Sort	Shell Sort
Merge Sort	Quick Sort	Heap Sort	Binary Search Algorithm
Basic Data Structures and Operations on them			
Stacks	Queues	Single Linked List	Double Linked List
Circular Linked List	1.		

Tree Data Structures			
Binary Search Trees	Heaps	Height Balanced Trees	
Graphs and Graph Algorithms			
Depth First Search	Breadth First Search	Minimum Spanning Trees: Kruskal Algorithm	Minumum Spanning Trees: Prim's Algorithm
Dijkstra Algorithm for Shortest Paths	Floyd Warshall Algorithm for Shortest Paths	Bellman Ford Algorithm	
Popular Algorithms in Dynamic Programming			
Dynamic Programming	Integer Knapsack problem	Matrix Chain Multiplication	Longest Common Subsequence
Greedy Algorithms			
Elementary cases : Fractional Knapsack Problem, Task Scheduling	Data Compression using Huffman Trees		

Click below if you'd like to check out our recent additions in the Physics section !



Add a comment...



Pramela Jayanthi · Tamilnadu College of Engineering, Coimbatore

nice

Reply · 1 · Like · September 30 at 8:53am

Judah Joel Devapriyan · Musician at NLAG

oh yeah! I had to finish my data-structure assignment late night... assignment soon! :-)

Reply · 1 · Like · August 22 at 10:49am