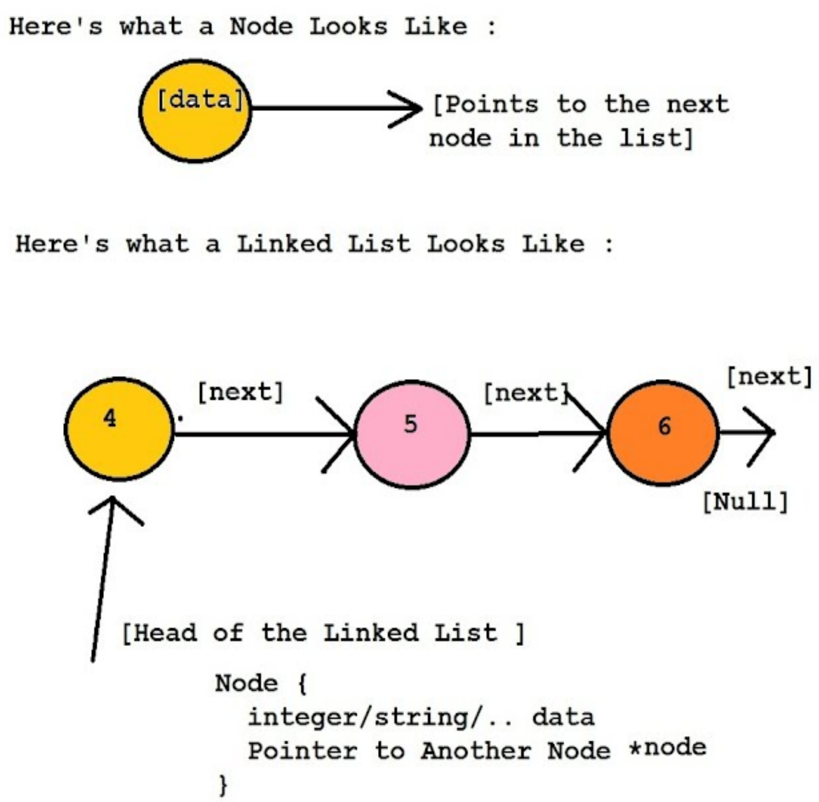
The Learning Point

Computer Science >

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

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

Singly Linked List



Singly linked list is the most basic linked data structure. In this the elements can be placed anywhere in the heap memory unlike array which uses contiguous locations. Nodes in a linked list are linked together using a next field, which stores the address of the next node in the next field of the previous node i.e. each node of the list refers to its successor and the last node contains the NULL reference. 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.

Basic operations of a singly-linked list are:

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

Functions

1. Insert – This function takes the start node and data to be inserted as arguments. New node is inserted at the end so, iterate through the list till we encounter the last node. Then, allocate memory for the new node and put data in it. Lastly, store the address in the next field of the new node as NULL.

2. Delete - This function takes the start node (as pointer) and data to be deleted as arguments. Firstly, go to the node for which the node next to it has to be deleted, If that node points to NULL (i.e. pointer->next=NULL) then the element to be deleted is not present in the list. Else, now pointer points to a node and the node next to it has to be removed, declare a temporary node (temp) which points to the node which has to be removed. Store the address of the node next to the temporary node in the next
field of the node pointer (pointer->next = temp->next). Thus, by breaking the link we removed the node which is next to the pointer (which is also temp). Because we deleted the node, we no longer require the memory used for it, free() will deallocate the memory.

3. Find - This function takes the start node (as pointer) and data value of the node (key) to be found as arguments. First node is dummy node so, start with the second node. Iterate through the entire linked list and search for the key. Until next field of the pointer is equal to NULL, check if pointer->data = key. If it is then the key is found else, move to the next node and search (pointer = pointer -> next). If key is not found return 0, else return 1.
4. Print - function takes the start node (as pointer) as an argument. If pointer = NULL, then there is no element in the list. Else, print the

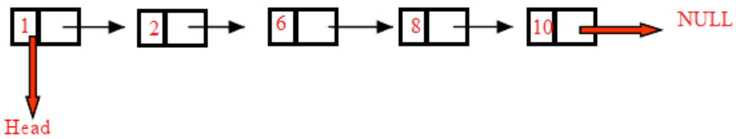
data value of the node (pointer->data) and move to the next node by recursively calling the print function with pointer->next sent as an argument.

Performance

1. The advantage of a singly linked list is its ability to expand to accept virtually unlimited number of nodes in a fragmented memory environment.
2. The disadvantage is its speed. Operations in a singly-linked list are slow as it uses sequential search to locate a node.

Singly Linked List

Singly linked list is the most basic linked data structure. In this the elements can be placed anywhere in the heap memory unlike array which uses contiguous locations. Nodes in a linked list are linked together using a next field, which stores the address of the next node in the next field of the previous node i.e. each node of the list refers to its successor and the last node contains the NULL reference. 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) 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 is used to point to the last node of the linked list) are initialized. Initially temp = start and temp-

Single Linked List - C Program source code

```
#include<stdio.h>
#include<stdlib.h>
typedef struct Node
{
        int data;
        struct Node *next;
}node;
void insert(node *pointer, int data)
{
        /* Iterate through the list till we encounter the last node.*/
        while(pointer->next!=NULL)
        {
                pointer = pointer -> next;
        }
```

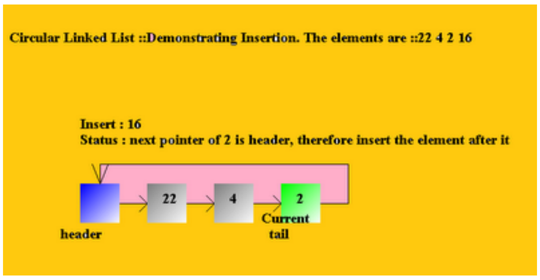
```
        /* Allocate memory for the new node and put data in it.*/
        pointer->next = (node *)malloc(sizeof(node));
        pointer = pointer->next;
        pointer->data = data;
        pointer->next = NULL;
}
int find(node *pointer, int key)
{
        pointer =  pointer -> next; //First node 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;
        /*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;
        /* 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");

                        }
                }
        }


}
```

Books from Amazon which might interest you !

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

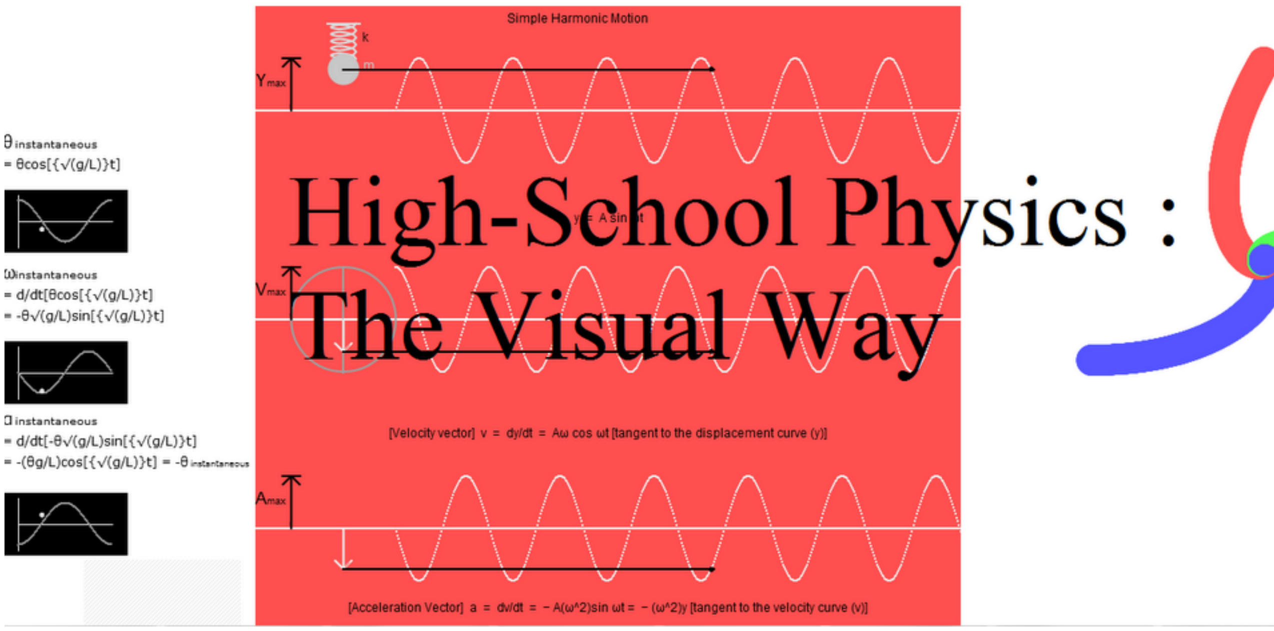
Some Important Data Structures and Algorithms, at a glance:

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:	Minumum Spanning Trees:

		Kruskal Algorithm	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 !



7 comments

Raghu Ambati · Tpist
sir I need insert element at compail time using single linked list.
Reply · Like · September 2 at 2:44am

Niraj Kumar Saxena · Tezpur Central University
gdd
Reply · Like · August 23 at 5:35am

Swapnil Raut · MET Bhujbal Knowledge City
Thnk u sir...
Reply · Like · September 4 at 8:24am

Rashmi Rayee Majumder · Member at NSEC-Phonix
Sir I want the programs of Linked List...
Reply · Like · September 3 at 11:19am

View 3 mor

Facebook social plugin