[Skip to main content](https://courses.iitbombayx.in/courses/course-v1:IITBombayX+CS101.2x+2022_T1/courseware/a71234edfd4f465e8d3e7e0082d04540/7565e9f168914c6cab4d54f9ba443bca/1?activate_block_id=block-v1%3AIITBombayX%2BCS101.2x%2B2022_T1%2Btype%40vertical%2Bblock%403b3b4160b29f4c6190a9d743897575c0#main)

**[IITBombayX Home Page](http://iitbombayx.in/)**

**IITBombayX: CS101.2xObject-Oriented Programming**

[Courses](http://iitbombayx.in/courses)

[Help](https://iitbombayx.in/contact_us)

[[https://courses.iitbombayx.in/static/images/profiles/default_50.3455a6581573.png](https://courses.iitbombayx.in/dashboard) Dashboard for:Paramhans\_Singh](https://courses.iitbombayx.in/dashboard)

1. [Course, current location](https://courses.iitbombayx.in/courses/course-v1:IITBombayX+CS101.2x+2022_T1/course/)

1. [Discussion](https://courses.iitbombayx.in/courses/course-v1:IITBombayX+CS101.2x+2022_T1/discussion/forum/)

1. [Progress](https://courses.iitbombayx.in/courses/course-v1:IITBombayX+CS101.2x+2022_T1/progress)

1. [Grading Policy](https://courses.iitbombayx.in/courses/course-v1:IITBombayX+CS101.2x+2022_T1/487375f5363a49a492641b63d5e814ec/)

1. [Course Slides](https://courses.iitbombayx.in/courses/course-v1:IITBombayX+CS101.2x+2022_T1/pdfbook/0/)

1. [Practice Programming](https://courses.iitbombayx.in/courses/course-v1:IITBombayX+CS101.2x+2022_T1/b5c2b3240ef9416ebb0ea5b2087cb4ee/)

1. [FAQs](https://courses.iitbombayx.in/courses/course-v1:IITBombayX+CS101.2x+2022_T1/f2e9c32b35ae4ef083ed716b4981ff0d/)

[Course](https://courses.iitbombayx.in/courses/course-v1:IITBombayX+CS101.2x+2022_T1/course/)  [Topic 2: Review of Pointers and Structures](https://courses.iitbombayx.in/courses/course-v1:IITBombayX+CS101.2x+2022_T1/course/#block-v1:IITBombayX+CS101.2x+2022_T1+type@chapter+block@a71234edfd4f465e8d3e7e0082d04540)  [Session 11: Programming Using Structures](https://courses.iitbombayx.in/courses/course-v1:IITBombayX+CS101.2x+2022_T1/course/#block-v1:IITBombayX+CS101.2x+2022_T1+type@sequential+block@7565e9f168914c6cab4d54f9ba443bca)  Walkthrough

 PreviousNext

1. video Programming Using Structures
2. other Slides: Programming Using Structures
3. other Walkthrough

**Walkthrough**

 Bookmark this page

**Self Referential Structures and Linked Lists**

**OBJECTIVE:**

This walkthrough will help you get introduced to one of the most important data structures "Linked Lists" used for storing data when the size of the data can dynamically increase (is not known prior). You will come to know how simple concepts of pointers and structures can be used to make something so useful and flexible.

**Let's Start :**

**Self Referential Structures :**

A Self Referential Structure (we would call it SRS from now) is a special structure which contains a member variable that points to structure of its own kind. Let's take an example to be more clear.

struct name{

     member 1;

     member 2;

     //Some other members

     struct name\* pointer;

}

**The Linked Lists :**

So, let's begin the main topic - The Linked lists.

So, you all have learnt about arrays specifically arrays of structure objects. But one of the major disadvantages these arrays suffer from is that they are static in nature. Let me explain it to you:

You have been given a task to maintain a list of books available in a library. What you have been provided with is the structure of the book element. Let's say that name of that struct is book.

Now what should be expected from you :

Updating the list when a new book gets added.

Deleting element from the list when the book is stolen or lost.

The problem in using static arrays here is that we need a size of array to be defined before declaring the array. But we can't come up with a number and say that this is the maximum number of books library will have. Maybe we can initialise a giant array of suppose size  1 million but if there are too few books in the library then the `array would be too sparse and unoptimized.  Let's see how Linked lists solve this problem.

So the linked list is an array of structure objects where the structure needs to be self referential and the ith element in the list points to the (i+1)th element through its member pointer and the pointer of the last element of the list would point to NULL. The starting element will be called as the head node.

Let's take the example of the book structure. The structure of the book given to us might be like this (after adding the self referencing pointer to the structure) :

struct book{

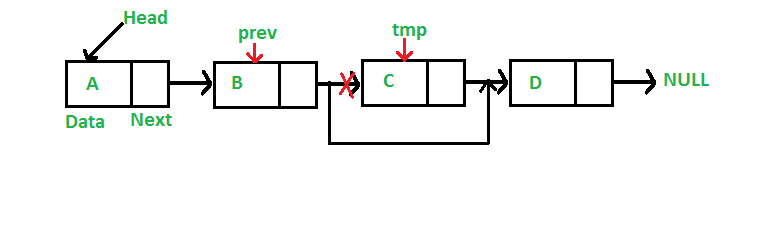
     member 1;

     member 2;

     // Other Members

     struct book\* next;

}



Well, first we make a head pointer.

struct book\* head = NULL;

This head pointer will serve as the starting point of the linked list.

**Basic Operations on Linked Lists :**

The functions we need to make will be of the following type:

void insert (struct book\* book\_new);

void delete\_book (int book\_id);

Here's how we implement these functions :

void insert (struct book\* book\_new){

     if (head == NULL){                                                /\*no element in the list.\*/

          head = new book; /\* Allocating space to head from the Heap memory \*/

          /\*Copy each member from book\_new to head\*/

          head->next = NULL;

     }

     else{        /\* there are already some elements in the list . Insert the new element at the start of the list.\*/

          book\* new\_node = new book;

          /\*Copy each member from book\_new to temp\*/

          temp->next = head;

          head = temp;

     }

}

Now comes the next function. The delete\_book() function:

void delete\_book (int book\_id){

     if(head == NULL){                                      /\* The list is empty \*/

          cout << "No book to delete. Exiting the function"<<endl;

          return;

     }

     else {        /\* When the list is not empty \*/

          /\* prev points to the element preceding the element pointed by temp\*/

          book\* temp = head;

          book\* prev = NULL;

          while (temp != NULL) {

               if(temp->id == book\_id) {

                    if(prev == NULL){            /\* We found the book at the head \*/

                         /\* Make the next node as the head pointer \*/

                         head = temp->next;

                         delete temp;

                         return;

                    }

                    else {  /\* Book\_id found somewhere else except the head \*/

                         prev->next = temp->next;

                         delete temp;

                         return;

                    }

               }

               prev = temp;

               temp = temp->next;

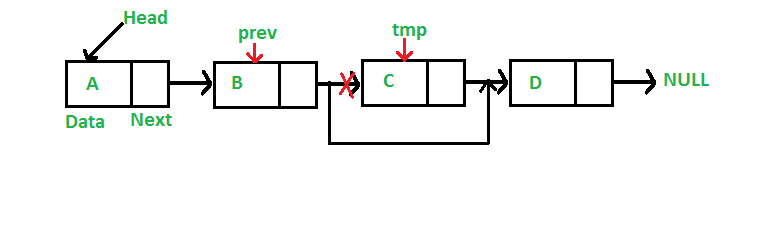
          }

          cout << "No book found with the matching id"<<endl;

          /\* No book found if we reach till here \*/

     }

}



**Conclusion:**

We briefly saw the use of linked lists for making dynamically growing data structures. No doubt, there are lots of variations in linked lists like *doubly linked lists*, *skip lists, etc.*I suggest you should go through them and explore more about it.

**Image Credits** : [geeksforgeeks.org](http://geeksforgeeks.org/)

 PreviousNext

© All Rights Reserved

[IITBombayX Home Page](http://iitbombayx.in/)

1. [About](http://iitbombayx.in/about)
2. [Research](http://iitbombayx.in/research)
3. [FAQs](http://iitbombayx.in/faq)
4. [Contact Us](http://iitbombayx.in/contact_us)

* [MOOCs Copyright](http://iitbombayx.in/mooc-copyright-guidelines)
* [Terms of Service and Honour Code](http://iitbombayx.in/honor)
* [Authorization Document](http://iitbombayx.in/authorization)
* [Privacy Policy](http://iitbombayx.in/privacy)

[Powered by Drupal](https://www.drupal.org/)

[Creative Common License](https://creativecommons.org/)

[](http://open.edx.org/)