Data Structures: Lists & Linked Lists

Doan Nhat Quang

doan-nhat.quang@usth.edu.vn
University of Science and Technology of Hanoi
ICT department

Today Objectives

- Introduce the basis of Linked Lists: declaration, initialization, and use.
- Learn different functions, and operations with Linked Lists: add, remove, search, etc.
- Implement examples in C/C++.

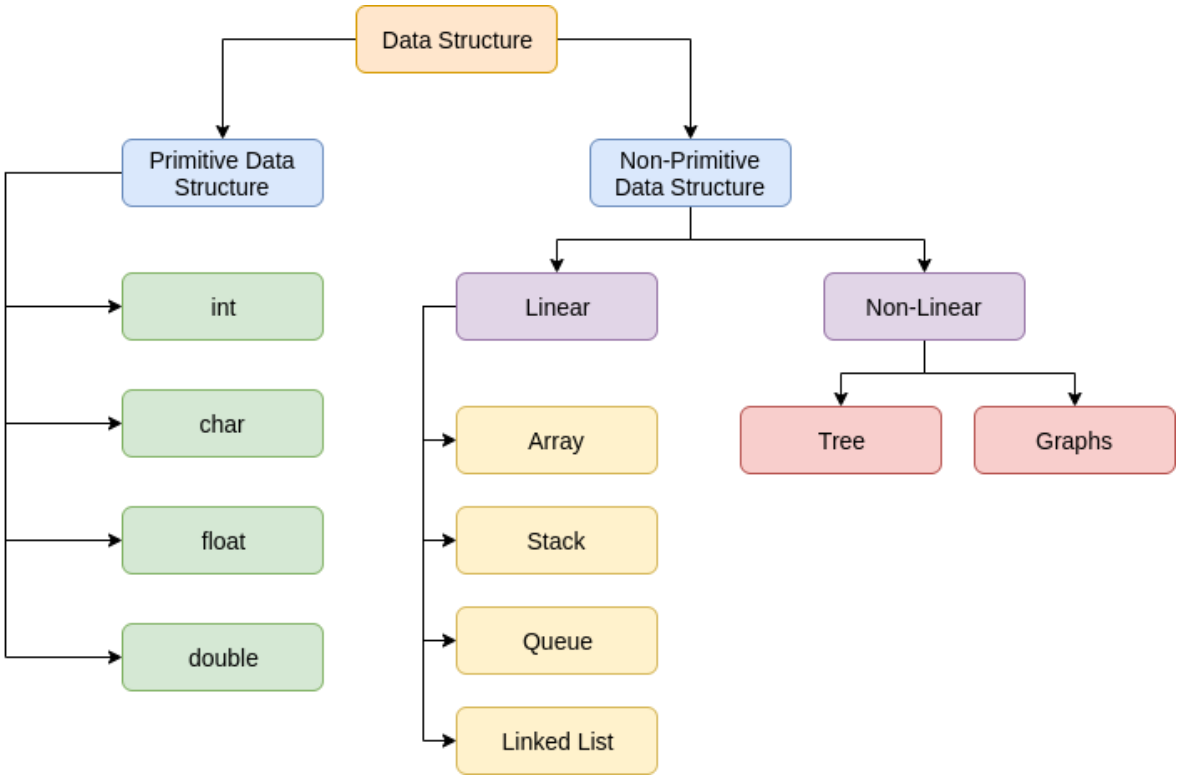
Data Structures

- Data structures study the organization of data in computers
  - the (abstract) data types (definition and representation)
  - relationship between elements of this type
  - operations on data types: retrieve, manipulate, compute, etc.
- Algorithms
  - methods to operate on data structures: efficiency and simplicity
  - **program = data structures + algorithm**

Data Structure

- A data structure consists of
  - a collection of data elements
  - a set of operations on these data elements
- Data structures in C:
  - **primitive or pre-defined**: any language defines a group of pre-defined data types such as int, char, float, double in C
  - **non-primitive or user-defined (Data structure)**: any language allows users to define their own (new) **data types** such as struct in C.

Data Structures



Data Structure

- Primitive data types:
  - type: int
  - elements: ...., -2, -1, 0, 1, 2, ...
  - operations: +, - , *, /, %, ...
- Data structures:
  - type: complex
  - elements: 1+3i, -15 + 93i, ...
  - operations: add, distance, square, etc.

Data Types vs Data Structures

Data Types	**Data Structures**
- Can hold values and not data; values of the given data type are assigned to a particular variable	- Data can be represented using an object
- Values can directly be assigned to the data type variables (using '=')	- Data is given to the data structure object using some set of algorithms and operations
- No problem with time complexity O(1)	- Time complexity comes into play when working with data structures

Structures

Structure is a user-defined data type available in C programming, which allows combining one or more variables, possibly of different types, grouped under a single name for convenient handling.

```
1 struct [structure tag]{
2     member definition;
3     member definition;
4     ...
5     member definition;
6 } [structure name];
```

Structures

- Container for related data for easy access
- May have empty gaps between members
- Useful in creating data structures such as linked lists, queues, trees, etc.

Structures

```
1  struct Student{
2      int age;
3      char name[50];
4      unsigned char gender;
5  };
6  struct Student s1, s2;
```

Lists

Definition

A list is a collection with a finite number of data objects that has the following properties:

- It is homogeneous, i.e., the elements are all of the same types.
- It has a finite length.
- Its elements are arranged in sequential (linear) order.

Lists

- Polynomial functions
  - $13 + 4x^2 + 79x^3 + 100x^{10} + 16x^{58}$
- Unbounded Integers
  - 456913018892132115479910488793126548976l3
- Text
  - "This is a sample of text!!"

Lists

Guild Wars 2 : heart of thorns par NCsoft
Jeu informatique
Il ne reste plus que 4 exemplaire(s) en stock.
Vendu par Century
Options de cadeau pas disponibles. En savoir plus
Supprimer | Mettre de côté

EUR 44,50

Starcraft 2 : legacy of the void par Blizzard
Jeu vidéo
En stock
Éligible à l'expédition GRATUITE
Ceci sera un cadeau En savoir plus
Supprimer | Mettre de côté

EUR 29,99

Kindle Paperwhite, Écran Haute Résolution 6" (15 cm) 300 ppp avec éclairage intégré et Wi-Fi - Avec offres spéciales par Amazon
En stock
Éligible à l'expédition GRATUITE
Ceci sera un cadeau En savoir plus
Supprimer | Mettre de côté

EUR 129,99

Sony DSC-RX100.CEE8 Appareil photo numérique 20,2 Mpix Zoom optique 3,6x Noir par Sony
En stock
Éligible à l'expédition GRATUITE
Ceci sera un cadeau En savoir plus
Supprimer | Mettre de côté

EUR 345,24

Samsung Galaxy Grand Prime Value Edition Gold par Samsung
En stock
Vendu par Tecnosell
Options de cadeau pas disponibles. En savoir plus
Supprimer | Mettre de côté

EUR 136,00

Lists

A **data structure** is a group of **data** elements grouped together under one name. These **data** elements, known as members, can have different types and different lengths. **Data structures** can be declared in C++ using the following syntax: struct type_name {

Data structures - C++ Tutorials - Cplusplus.com
www.cplusplus.com/doc/tutorial/**structures**/
You visited this page.

About this result • Feedback

Data structure - Wikipedia, the free encyclopedia
https://en.wikipedia.org/wiki/**Data_structure** ▾ Wikipedia ▾
In computer science, a **data structure** is a particular way of organizing **data** in a computer so that it can be used efficiently.
List of data structures · Abstract data type · Linked · Array

Data Structure and Algorithms (DSA) Tutorial
www.tutorialspoint.com/**data_structures**_algorithms/ ▾
Data Structures are the programmatic way of storing **data** so that **data** can be used efficiently. Almost every enterprise application uses various types of **data structures** in one or other way.

Data structures - C++ Tutorials - Cplusplus.com
www.cplusplus.com/doc/tutorial/**structures**/ ▾
A **data structure** is a group of data elements grouped together under one name. These data elements, known as members, can have different types and different ...
You visited this page.

Data Structures Archives - GeeksforGeeks
www.geeksforgeeks.org/**data-structures**/ ▾
Average time Taken to Cover All this post. Beginners - 8 MONTHS.(Learnt already C/C++/Java.) Intermediate - 4 Months(Read and Execute Mode). Intermediate ...

Data structure

In computer science, a data structure is a particular way of organizing data in a computer so that it can be used efficiently. Wikipedia

Feedback

See results about

Algorithms + Data Structures = Programs (Book by Nikl...
Originally published: 1976
Author: Niklaus Wirth

Lists

Common different approaches for implementing a List data structure

- Static array: arrays can be simply used to manipulate collections of elements.
- Dynamic array: using **malloc()** is capable of representing a list to avoid the fixed-size list
- Linked list: A very flexible mechanism for dynamic memory management is provided by pointers.

Lists

Basic operations are often defined

1. init(): create an empty list
2. isEmpty(): check if the list is empty
3. insert(): add new item in a list
4. remove(): remove an item from a list

Lists

Basic operations are often defined

1. init(): create an empty list
2. isEmpty(): check if the list is empty
3. insert(): add new item in a list
4. remove(): remove an item from a list

Some other possible operations can be used:

1. length(): return the length of a list
2. search(): search a specific element in a list
3. display(): display a list
4. sort(): sort all items in a list

Static Array-based Lists

The idea is to store the list in a fixed-size static array.

```
1  struct _List {
2      int size;
3      int data[CAPACITY];
4  };
5  typedef struct _List List;
```

Static Array-based Lists

- Before use, a list must have to be initialized with no element.
- The below init() requires an initialization of l in the main() using (List*)malloc(sizeof(List));.

```
1  void init (List *l) {
2      l->size = 0;
3  }
```

Static Array-based Lists

- If l is initialized and gets malloc() inside the function, you should return l to the actual argument.

```
1  List * init(List *l, int N) {
2      l = (List*)malloc(sizeof(List));
3      l->size = 0;
4      return l;
5  }
```

Static Array-based Lists

- Several operations require to verify whether a list is not empty or else further operations will be treated.

```
1  int isEmpty(List *l) {
2    return (l->size==0);
3  }
```

8	13	2	1	30	15	91	24	7	10

Static Array-based Lists

- This function provides basic information about a list.

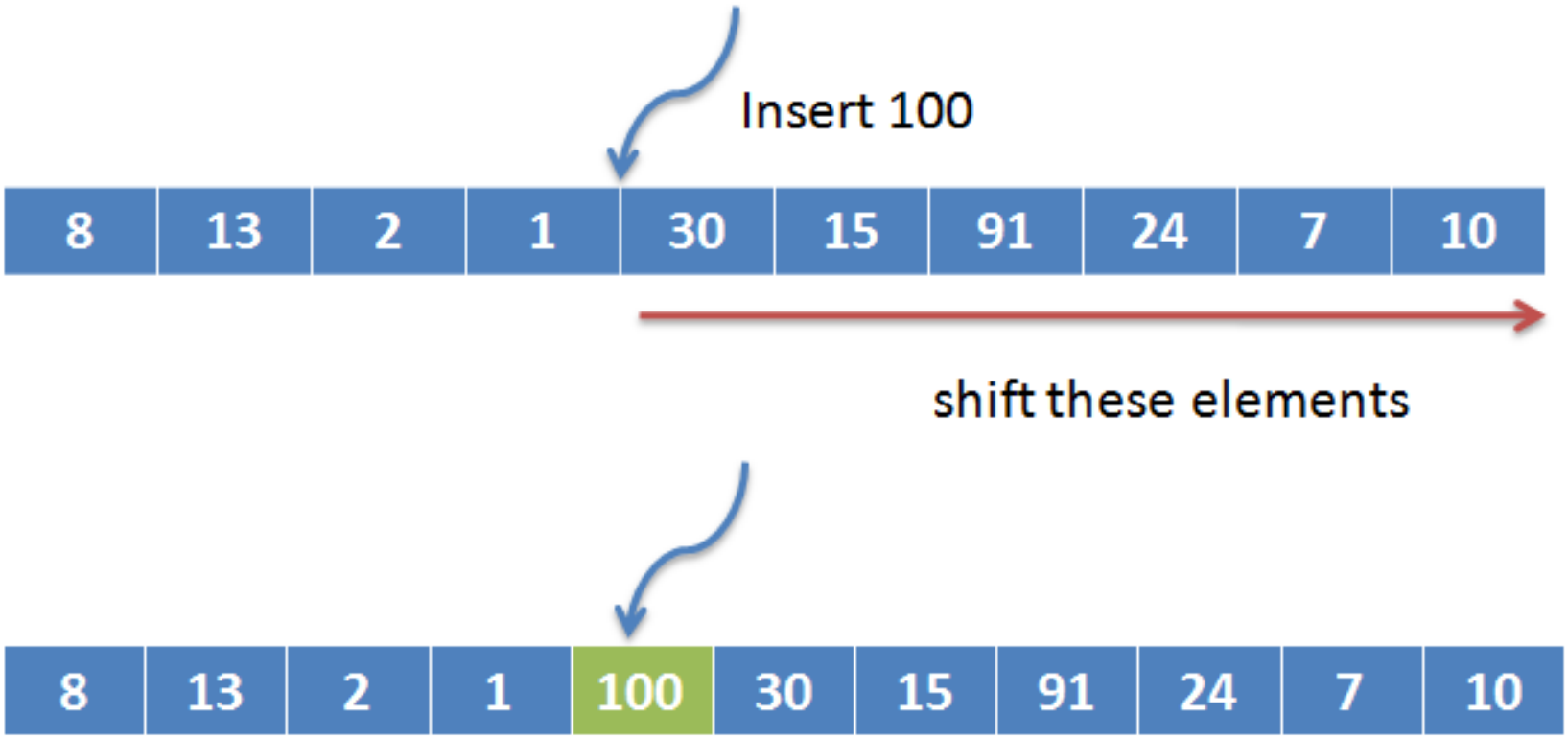
```
1  int length(List *l) {
2    return l->size;
3  }
```

- An array-based list allows access directly to the index of its elements. display() functions can be implemented to provide its information.

```
1  int display(List *l){
2       for (int i=0; i<length(l); i++){
3            printf(''Element %d: %d'',i,l->data[i]);
4       }
5  }
```

Static Array-based Lists

Before the insertion, several verifications on the index should be done.
Elements next to the inserted index should be shifted to the right.

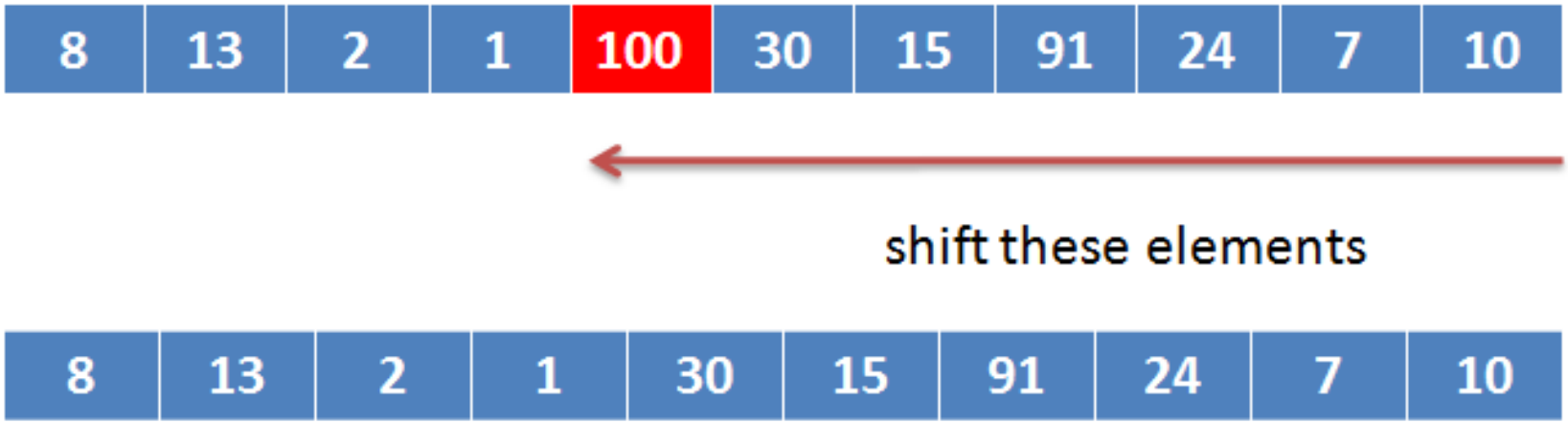


Static Array-based Lists

```
1  void add(List *l, int index, int _val){
2      if (length(l) == CAPACITY){
3          printf("Cannot add more items to the list!");
4          return;
5      }
6      if ((index < 0) || (index> length(l))){
7          printf("Illegal index!"); return;
8      }
9      else{
10         for (int i=length(l);i>index;i--)
11             l->val[i] = l->val[i-1];
12         l->val[index] = _val;
13         l->size++;
14     }
15 }
```

Static Array-based Lists

Removing an item with a specific index from a list requires shifting elements to the left.



Static Array-based Lists

```
1  void remove(List *l, int index){
2       if (isEmpty(l) == 1){
3           printf("List is empty!");
4           return;
5       }
6       if ((index < 0) || (index> length(l))){
7           printf("Illegal index!");
8           return;
9       }
10      else{
11          for (int i=index-1;i<length(l);i++)
12              l->val[i] = l->val[i+1];
13          l->size--;
14      }
15 }
```

Dynamic Array-based Lists

- We often don't know the size of the memory in the problem in advance.
- Instead of Static Lists, Dynamic Lists can be used.

```
1 typedef struct _List{
2         int size;
3         int capacity;
4         int  *val;
5 }List;
```

Dynamic Array-based Lists

- The initialization will be implemented with the use of the malloc function.
- The below init() requires an initialization of l in the main() using (List*)malloc(sizeof(List));.

```
1 void init (List *l, int N) {
2     l->size = 0;
3     l->capacity = N;
4     l->val = (int *)malloc(l->capacity);
5 }
```

- All other operations are the same as the ones of Static Array-based Lists.

Dynamic Array-based Lists

- If l is initialized and gets malloc() inside the function, you should return l to the actual argument.

```
1  List * init(List *l, int N) {
2      l =  (List*)malloc(sizeof(List));
3      l->size = 0;
4      l->capacity = N;
5      l->val = (int *)malloc(l->capacity);
6      return l;
7  }
```

Array-based Lists

Array-based list implementation:

Pros

- easy to understand and simple to implement.
- able to access to any element.

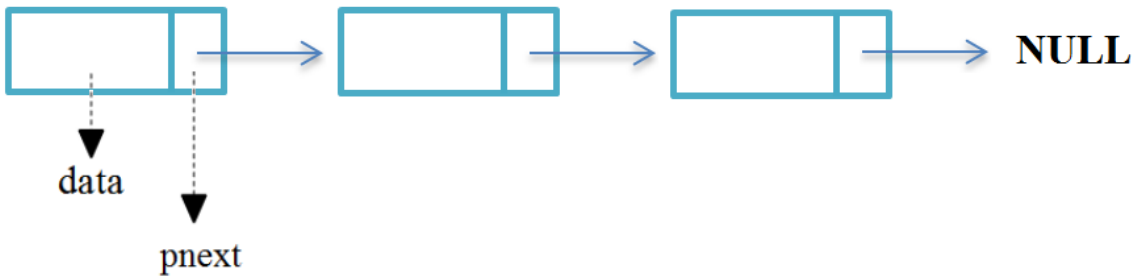
Cons

- the size has to be fixed beforehand.
- inserting or deleting an element is very difficult because we have to shift the position of many elements.

Definition

In Linked Lists, each item is placed together with the link to the next item, resulting in a simple component called a node:

- A data part stores an element value of the list.
- A next part contains a link (or pointer) that indicates the node's location containing the next list element. If a link does not point to a node, its value is set to NULL (a special C++ constant in stdlib.h).



Linked Lists

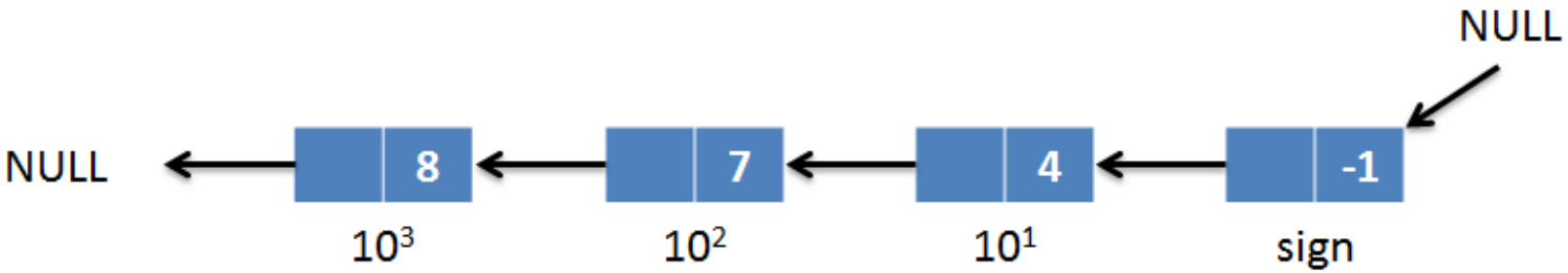
- An array is a static data structure. This means the array's length cannot be altered at run time. While a linked list is a dynamic data structure.
- In an array, all the elements are kept at consecutive memory locations, while in a linked list, the elements (or nodes) may be kept at any location but are still connected.

Linked Lists

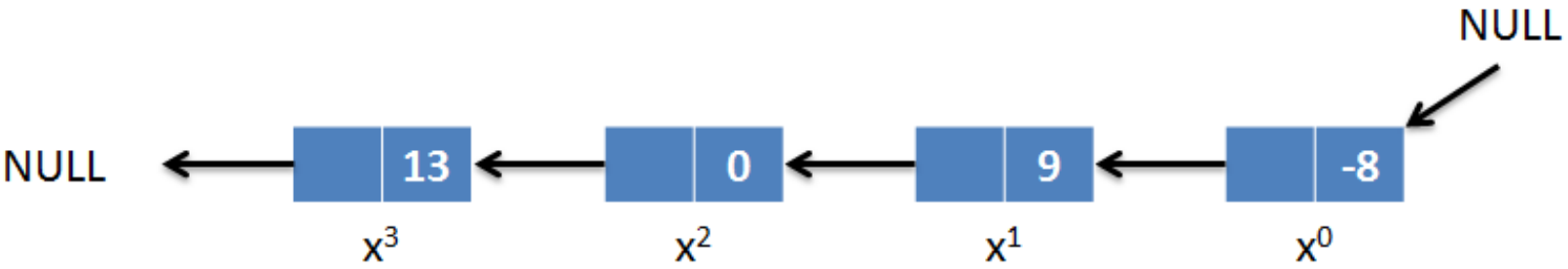
- Successive elements are linked by pointers, and the last element points to NULL.
- The size grows or shrinks during the execution of a program; it does not waste memory space like Array-based List data structure.
- Linked lists provide flexibility in allowing the items to be rearranged efficiently for inserting or deleting an element.

Linked Lists applications

- To represent unbounded integers: -874



- To represent polynomial functions: $13x^3 + 9x^1 - 8$



Linked Lists applications

- Blockchain adopts the same principle as Linked List.
  - Block in blockchain has the hash function/number; Nodes in a linked list have a pointer.
  - Blockchain's data structure is Linked List, but Blockchain is a cryptography protocol.
- Linked list principle is applied in Stack, Queue, Tree, Graph.

Linked Lists

Basic operations are often defined

1. init(): create an empty list
2. isEmpty(): check if the list is empty
3. insert(): add new item in a list
4. remove(): remove an item from a list

Linked Lists

Basic operations are often defined

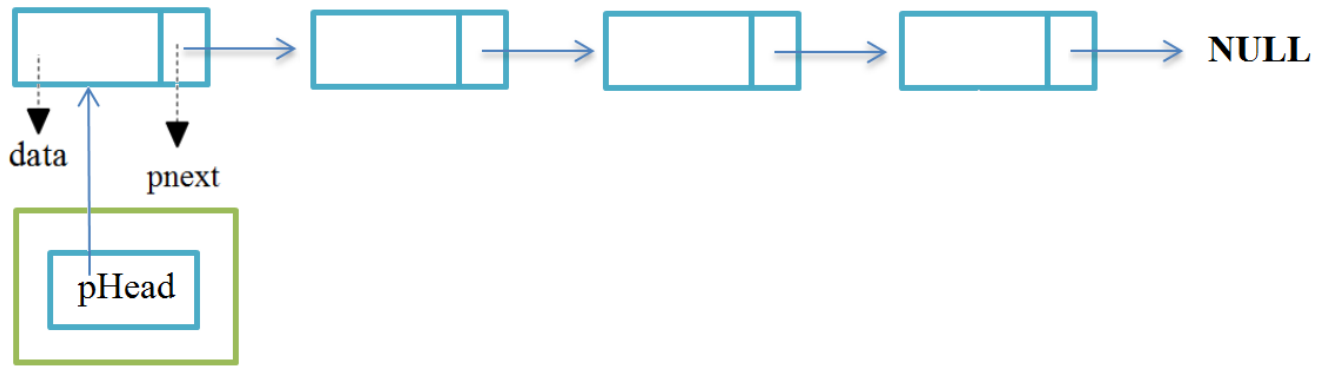
1. init(): create an empty list
2. isEmpty(): check if the list is empty
3. insert(): add new item in a list
4. remove(): remove an item from a list

Some other possible operations can be used:

1. length(): return the length of a list
2. search(): search a specific element in a list
3. display(): display a list
4. sort(): sort all items in a list

Linked Lists

```
1  typedef struct _Node {
2      int data;
3      struct _Node *pnext;
4  } Node;
5  typedef struct _List {
6      int size;
7      Node  *pHead;
8  } List;
```



Linked Lists

- A linked list must be initialized with the head node, a NULL pointer.

```
1  void init(List *l){
2    // l gets malloc() in the main() function
3    //memset(l,0,sizeof(List));
4    // older compilers may require this function
5    l->size=0;
6    l->pHead=NULL;
7  }
8  Node* initNode(int val){
9    Node *node = (Node *) malloc(sizeof *node);
10   node->data = val;
11   return node;
12 }
```

- A list is empty if there is no element or the head node is NULL.

```
1  int isEmpty(List *l){
2   return (l->size == 0);
3  }
```

Linked Lists

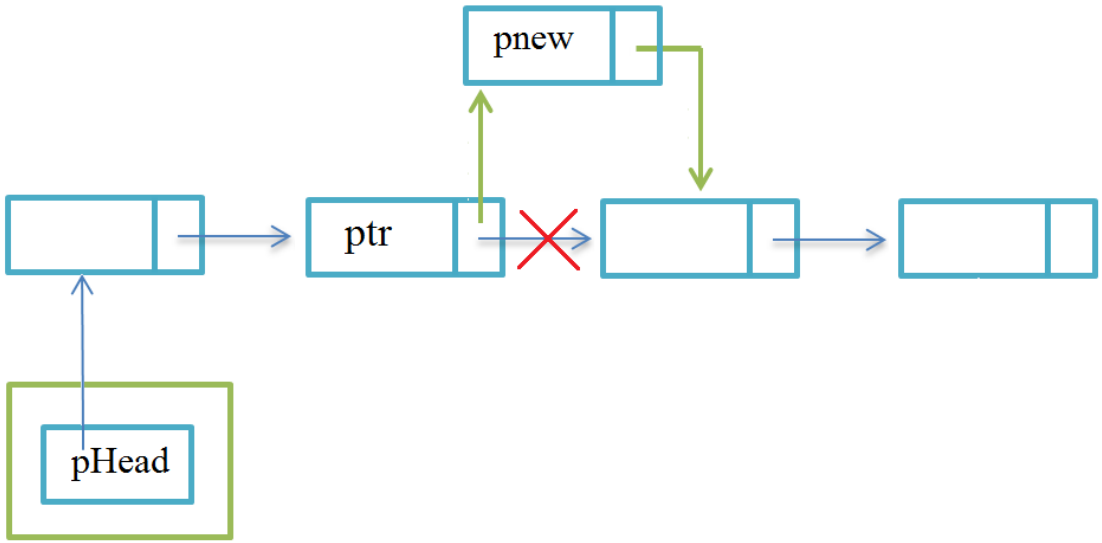
There are two cases while inserting new elements into a list:

1. If the list is empty, its first element will be added to the list head.

```
1 void insertFirst(Node *pnew, List *l){
2     if (isEmpty(l)){
3         l->size ++;
4         l->pHead = pnew;
5     }
6 }
```

2. A new node is linked after a specific node.

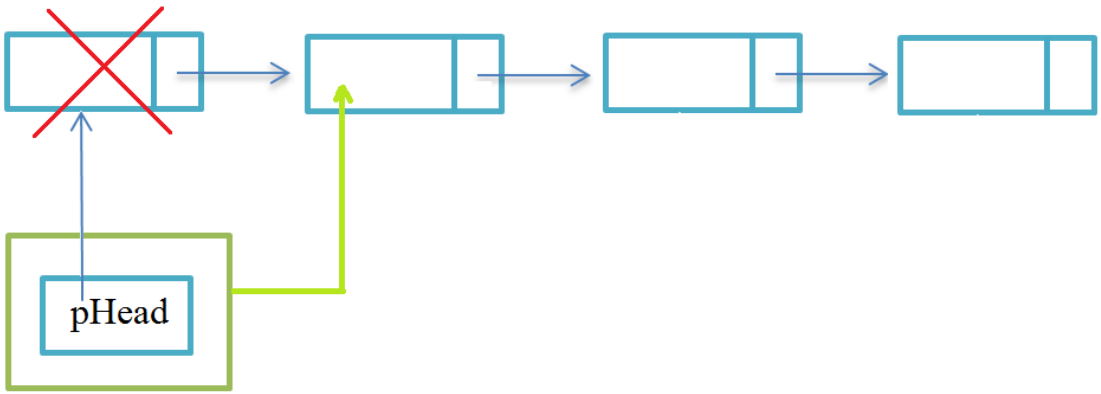
```
1  void insert(Node *pnew, Node *ptr, List *l){
2    pnew->pnext = ptr->pnext;
3    ptr->pnext=pnew;
4    l->size++;
5  }
```



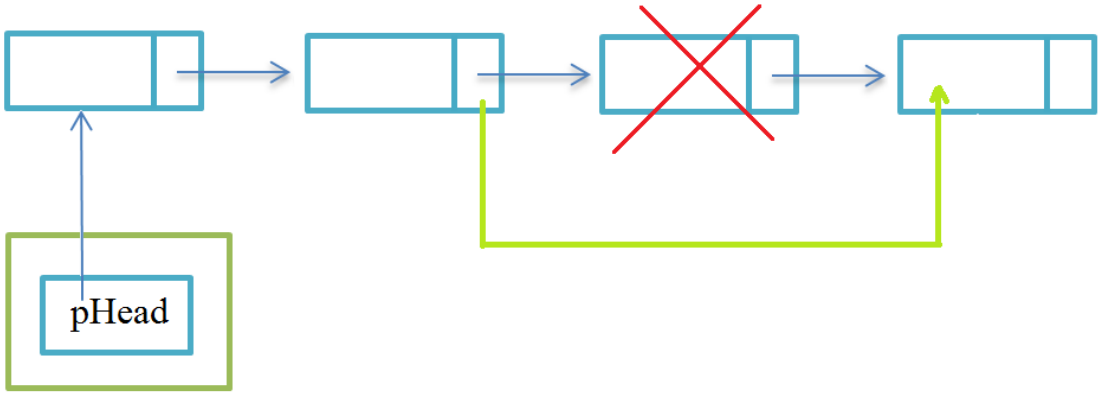
Linked Lists

There are two cases while removing an element from a list:

1. If the first element is removed, the head should point to the next node of the first element.



2. In another case, any node is removed, and a new link is created between the previous node and the next node.



free() is used to free the allocated memory.

```
1  void remove(List *l, int val){
2       Node *p = l->pHead;
3       if (p->data == val){
4            l->pHead = p->pnext;
5            free(p);
6            l->size--;
7            return;
8       }
9       Node *q = p->pnext;
10      int cnt = 1;
11      while ((p->data != val) && (cnt <length(l))){
12           q = p;
13           p = p->pnext;
14      }
15      if (p != NULL){
16         q->pnext = p->pnext;
17         l->size--;
18         free(p);
19      }
20 }
```

Comparisons

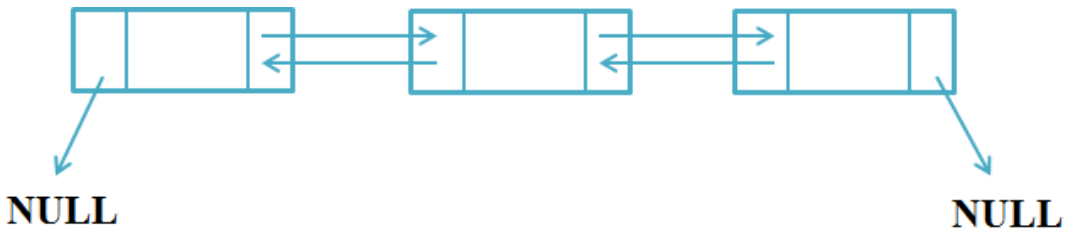
Comparisons of complexity for different list implementations

	Linked list	Array
Indexing	O(n)	O(1)
Insert/delete at the beginning	O(1)	O(n)
Insert/delete at the end	O(n)	O(1)
Insert/detele at the middle	searching time	shifting time

Doubly Linked Lists

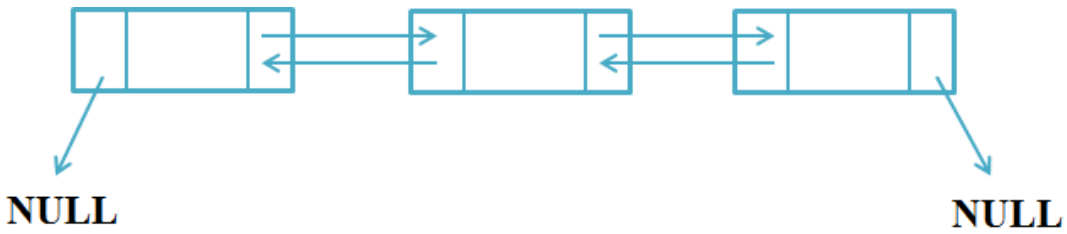
Doubly Linked Lists is a variation of Linked List in which navigation is possible in both ways either forward and backward.

- A data part stores an element value of the list.
- A next part contains a link indicating the next element.
- A previous part contains a link indicating the previous element.

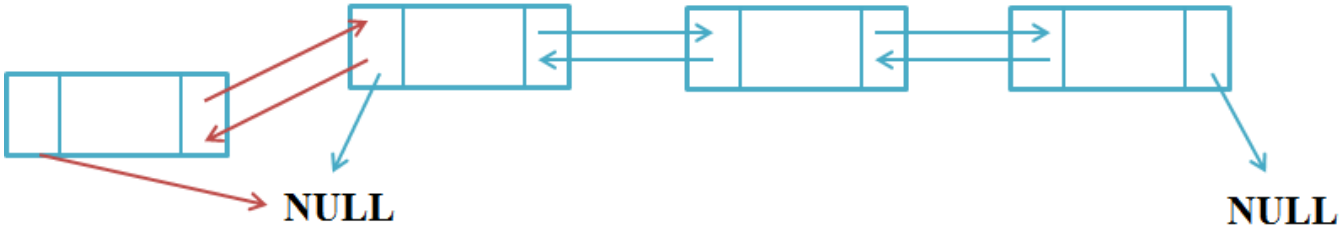


Doubly Linked Lists

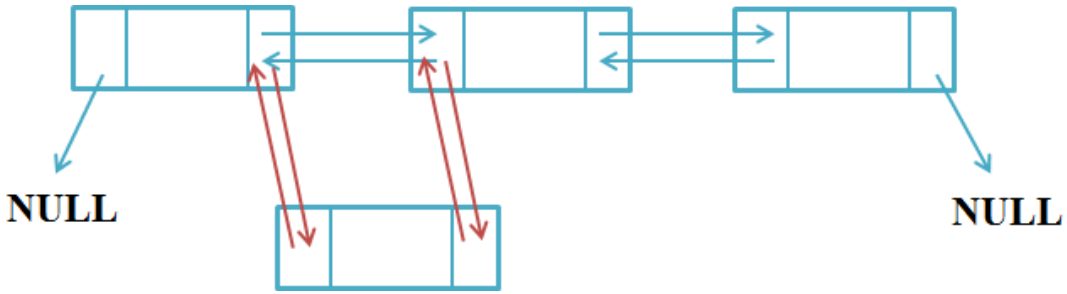
```
1  typedef struct _Node {
2      int data;
3      struct _Node *pnext;
4      struct _Node *pres;
5  } Node;
6  typedef struct _List {
7      int size;
8      Node  *pHead;
9      Node *pTail;
10 } List;
```



Doubly Linked Lists



Insertion at the head of Double Linked Lists



Insertion at the middle of Double Linked Lists

Doubly Linked Lists

Basic operations are often defined

1. init(): create an empty list
2. isEmpty(): check if the list is empty
3. insert(): add new item in a list. It is possible to add new elements to the head or the tail.
4. remove(): remove an item from a list

Doubly Linked Lists

Basic operations are often defined

1. init(): create an empty list
2. isEmpty(): check if the list is empty
3. insert(): add new item in a list. It is possible to add new elements to the head or the tail.
4. remove(): remove an item from a list

Some other possible operations can be used:

1. length(): return the length of a list
2. search(): search a specific element in a list
3. display(): display a list from left to right or from right to left.
4. sort(): sort all items in a list

Doubly Linked Lists

Advantages

- Linked lists are a dynamic data structure that can grow and be pruned, allocating and deallocating memory while the program runs.
- Insertion and deletion node operations are easily implemented in a linked list.
- Linear data structures such as stacks and queues are easily executed with a linked list (next chapter).

Doubly Linked Lists

Disadvantages

- Nodes in a linked list must be read in order from the beginning as linked lists are inherently sequential.
- Nodes are stored contiguously, greatly increasing the time required to access individual elements within the list.
- Difficulties arise in linked lists when it comes to reverse traversing. For instance, singly linked lists are cumbersome to navigate backward, and while doubly linked lists are somewhat easier to read, memory is wasted in allocating space for a back-pointer.

Linked List Applications

Applications

- Great use of the doubly linked list is in navigation systems, as it needs front and back navigation.
- Linked lists can be implemented to represent structured data, i.e., graphs, trees in the web domain, social networks, etc.
- Multiply linked lists, Circular Linked lists are available to deal with specific issues.