LINKED LISTS

Array Data structure

Operations for Searching an Unsorted array

- If there are 1024 elements it would take max 1024 searches
- If there are 2048 elements it would take max 2048 searches

- If there are 1600 elements it would take max 1600 searches

Binary Search in a sorted array

- 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
- [108 121 237 288 332 370 460 482 515 555 578 626 747 874 930 995]
- Search for 125
- (0+15)/2 = 7 :: 482 NO
- (0+6)/2 = 3 :: 288 NO
- (0+2)/2 = 1 :: 121 NO
- (2+2)/2 = 2 :: 237 NO

- Any search would take max 4 operations.
- $(2)^4$ = 16

Operations for Searching a sorted array

- If there are 1024 elements it would take max 10 searches
- $(2)^{10}$ = 1024
- If there are 2048 elements it would take max 11 searches
- $(2)^{11}$ = 2048
- If there are 1600 elements it would take max 11 searches

- $(2)^{10}$ = 1024 and $(2)^{11}$ = 2048
- If there are 2000 elements it would take max 11 searches

$\cancel{2^x = 1600}$

$x = ?$

Search, Delete, Insert in a sorted array

- Sorted arrays are very good to store information
- Searching is very fast
- What about Deletion and insertion?

Storing items stocked by each dealer

- There are 16 dealers. Each dealer has stock of 5 different items.
- Each dealer has an ID
- One array holds the IDs of 16 dealers
- This array is sorted
- 5 more arrays hold the stock (number of items) of each variety corresponding to each dealer

Stock-data stored in an array sorted on ID

- 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
- [108 121 237 288 332 370 460 482 515 555 578 626 747 874 930 995]
- [6 5 0 2 5 7 9 3 8 2 5 4 7 2 5 8]
- [2 1 7 3 0 2 1 7 4 7 0 2 1 6 3 8]
- [6 5 8 1 5 3 9 9 0 1 5 8 0 2 0 2]
- [9 1 3 7 3 4 0 7 6 7 2 0 1 0 3 0]
- [0 0 0 1 5 1 2 9 3 9 0 1 3 1 0 6]

- Dealer with ID 930 closes his store.
- We want to remove related information from our array.

Delete records for ID 930

- 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
- [108 121 237 288 332 370 460 482 515 555 578 626 747 874 930 995]
- [6 5 0 2 5 7 9 3 8 2 5 4 7 2 5 8]
- [2 1 7 3 0 2 1 7 4 7 0 2 1 6 3 8]
- [6 5 8 1 5 3 9 9 0 1 5 8 0 2 0 2]
- [9 1 3 7 3 4 0 7 6 7 2 0 1 0 3 0]
- [0 0 0 1 5 1 2 9 3 9 0 1 3 1 0 6]

Delete records for Id 930

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
[	108	121	237	288	332	370	460	482	515	555	578	626	747	874	930	995	]
[	6	5	0	2	5	7	9	3	8	2	5	4	7	2	5	8	]
[	2	1	7	3	0	2	1	7	4	7	0	2	1	6	3	8	]
[	6	5	8	1	5	3	9	9	0	1	5	8	0	2	0	2	]
[	9	1	3	7	3	4	0	7	6	7	2	0	1	0	3	0	]
[	0	0	0	1	5	1	2	9	3	9	0	1	3	1	0	6	]

- You cannot leave array values blank

Delete records for Id 930

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
[	108	121	237	288	332	370	460	482	515	555	578	626	747	874	995	]
[	6	5	0	2	5	7	9	3	8	2	5	4	7	2	8	]
[	2	1	7	3	0	2	1	7	4	7	0	2	1	6	8	]
[	6	5	8	1	5	3	9	9	0	1	5	8	0	2	2	]
[	9	1	3	7	3	4	0	7	6	7	2	0	1	0	0	]
[	0	0	0	1	5	1	2	9	3	9	0	1	3	1	6	]

- 6 records shifted

Consider original Stock data again

- 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
- [108 121 237 288 332 370 460 482 515 555 578 626 747 874 930 995]
- [6 5 0 2 5 7 9 3 8 2 5 4 7 2 5 8]
- [2 1 7 3 0 2 1 7 4 7 0 2 1 6 3 8]
- [6 5 8 1 5 3 9 9 0 1 5 8 0 2 0 2]
- [9 1 3 7 3 4 0 7 6 7 2 0 1 0 3 0]
- [0 0 0 1 5 1 2 9 3 9 0 1 3 1 0 6]

Now Delete records for ID 121

- 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
- [108 121 237 288 332 370 460 482 515 555 578 626 747 874 930 995]
- [6 5 0 2 5 7 9 3 8 2 5 4 7 2 5 8]
- [2 1 7 3 0 2 1 7 4 7 0 2 1 6 3 8]
- [6 5 8 1 5 3 9 9 0 1 5 8 0 2 0 2]
- [9 1 3 7 3 4 0 7 6 7 2 0 1 0 3 0]
- [0 0 0 1 5 1 2 9 3 9 0 1 3 1 0 6]

- You can not leave an element blank

Now Delete records for ID 121

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
[	108	121	237	288	332	370	460	482	515	555	578	626	747	874	930	995	]
[	6	5	0	2	5	7	9	3	8	2	5	4	7	2	5	8	]
[	2	1	7	3	0	2	1	7	4	7	0	2	1	6	3	8	]
[	6	5	8	1	5	3	9	9	0	1	5	8	0	2	0	2	]
[	9	1	3	7	3	4	0	7	6	7	2	0	1	0	3	0	]
[	0	0	0	1	5	1	2	9	3	9	0	1	3	1	0	6	]

Now Delete records for Id 121

- 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
- [108 237 288 332 370 460 482 515 555 578 626 747 874 930 995]
- [6 0 2 5 7 9 3 8 2 5 4 7 2 5 8]
- [2 7 3 0 2 1 7 4 7 0 2 1 6 3 8]
- [6 8 1 5 3 9 9 0 1 5 8 0 2 0 2]
- [9 3 7 3 4 0 7 6 7 2 0 1 0 3 0]
- [0 0 1 5 1 2 9 3 9 0 1 3 1 0 6]

- 84 Records shifted

Search, Deletions and Insertions

- So while *Searching* is very fast for N dealers with M items, *Deletion* of one dealer records may involve upto MxN shift of data, as you cannot leave holes in the arrays. Same thing holds true for *Insertion* of a new dealer. Firstly, all elements would need to be shifted right to make place for the elements at this position.

- So we need to look for other data structures.

Issues with Arrays

- Consider a university student record with 4000 students, and 200 items stored for each student.
- If array is Sorted, *Searching* for a record would need around 12 searches
-

Issues with Arrays

- *Insertion operation* for a new student
  - create a hole, as all elements are in consecutive locations in memory – shift number of elements
  - number of operations , best case last element 1 operation
  - worst case: first student : 4000x200 elements to be shifted
- *Deletion operation* of an existing student
  - close a hole – shift number of elements , best case: last student
  - worst case: first student : 4000x200 elements to be shifted

Issues with Arrays

- Sorted arrays: O(log n) time to search

- Inserting a new element of size m
  - create a hole – shift large number of elements
  - large number of operations, worst case: O(nm)

- Deleting a new element
  - close a hole – shift large number of elements
  - large number of operations, worst case: O(nm)

- Also, the array size has to be declared in the beginning.

Passengers sitting in some particular order

- Suppose every passenger has 3 times more luggage
- How about allowing a new passenger to join (in sorted order)
- How about removing a passenger? How many people and luggage would need to shift?

Handling Polynomials

- Addition of Two polynomials
- $2x^3 + 4x^2 - 6x + 8$
- $7x^2 + 3x + 10$

- Addition of Two polynomials
- $2x^3 + 4x^2 - 6x + 8$
- $\qquad 7x^2 + 3x + 10$

- $2y^3 + 4y^2 - 6y + 8$
- $\qquad 7y^2 + 3y + 10$
- Use an array whose index relate to the power, and contents denote coefficients

only coefficients need to be filled

$2x^3 + 4x^2 - 6x + 8$

6	5	4	3	2	1	0
			2	4	-6	8

$7x^2 + 3x + 10$

6	5	4	3	2	1	0
				7	3	10

6	5	4	3	2	1	0
			2	11	-3	18

$2x^3 + 11x^2 - 3x + 18$

- But $\boxed{2 \;\cdot\; - \; - \; - \; - \; - \;\cdot\; 1 \;\cdot\; -6 \quad 8}$
- $2x^{30} + x^{12} - 6x + 8$
- $x^{22} + 4x^{2} - 16x$

$$2x^3 + 4x^2 - 6x + 8$$

$$7x^2 + 3x + 10$$

6	5	4	3	2	1	0
			2	4	-6	8

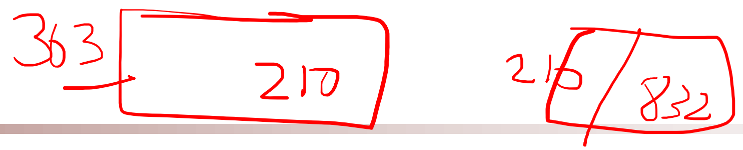
6	5	4	3	2	1	0
				7	3	10

6	5	4	3	2	1	0
			2	11	-3	18

$$2x^{30} + x^{12} - 6x + 8$$

Can we handle this with sorted list of passengers

A new approach



- How about giving some arbitrary number to each passenger.
- Put all the numbers in some order (363, 210, 832, 410,….)
- Ask each passenger to have a chit showing his number, and his successor
- 832 -- 410
- When his turn comes, he simply calls out to his successor to be ready
- Passenger can lounge around in any place on the platform
- List: 126 – 363 – 210 – 832 – 410 – 666 – 154

linked list in computer memory

Memory Address	List contents	Next link
126	Ramesh	363
832	Ashok	410
363	Munish	210
410	Ratna	666
154	Sumati	XXX
210	Piyush	832
666	Ankit	154

Adding and removing passengers

- Suppose you wanted to add another person after 210 with number 960
- Simply give passenger 210 a new chit 210 – 960
- and give a chit to new passenger 960 – 832
- New list
- List: 126 – 363 – 210 – 960 – 832 – 410 – 666 – 154
- Same thing if a passenger needs to be removed from the list

Modified linked list in computer memory

Memory Address	List contents	Next link
126	Ramesh	363
832	Ashok	410
363	Munish	210
410	Ratna	666
154	Sumati	XXX
210	Piyush	960
666	Ankit	154
960	Suman	832

A new approach

- You need to handle only the IDs, NOT their records. They can be anywhere.
- Once ID is gone, rest need not be physically removed.

A different approach for handling lists

Handling Lists

- Consider a list of integers
- { 16, 8, 10, 2, 34, 20, 12, 32, 18, 9, 3 }
- It can be thought of as an element 16 followed by another list
- $\boxed{16}$ --{ $\underline{8, 10, 2, 34, 20, 12, 32, 18, 9, 3}$ }

- 16 --{some list}

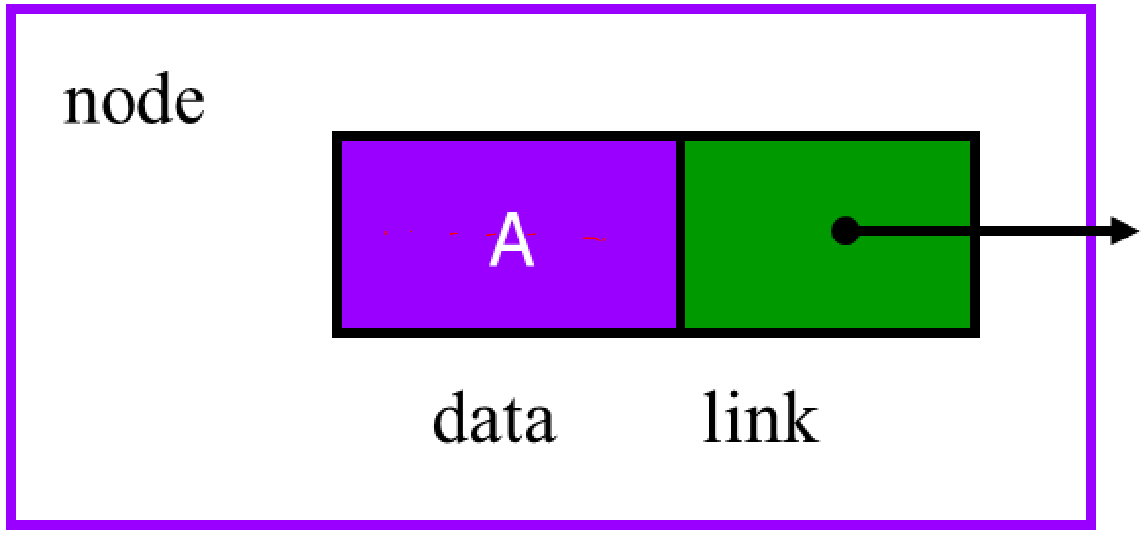
A List is an element followed by Lists

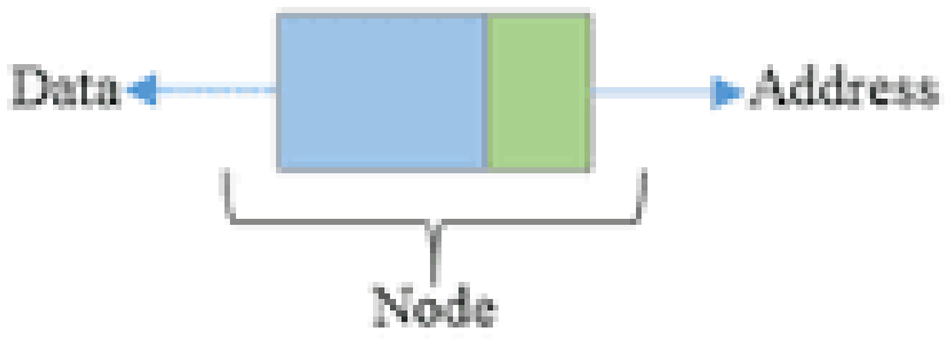
- The second list can also be thought of an element 8 followed by a list
- 16--8-- { $\underline{10, 2, 34, 20, 12, 32, 18, 9, 3}$ }
- 16--8--{another list}

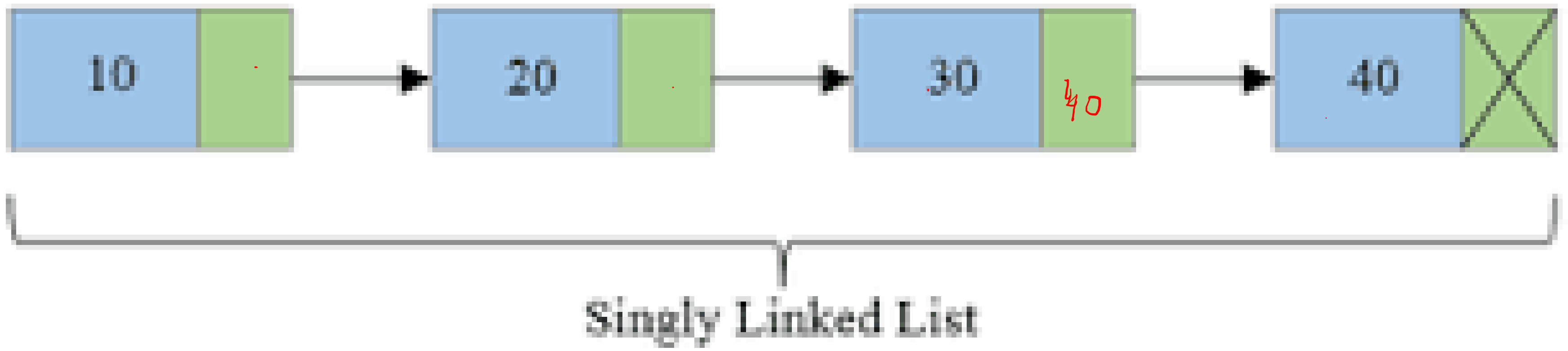
- Thus any list can be thought of as an element followed by a list
- We can come up with a recursive definition of a linked list where each element is linked to the next one

Linked Lists

- A linked list is a linear data structure where each element is a separate object.
- Each element of such a list needs to comprise of two items
  - the data and
  - a reference to the next node.
- **node :** object that holds the data and refers to the next element
- **data** component may actually consist of several fields
- For example in a linked list of students, the data could be student’s name and marks in 6 subjects

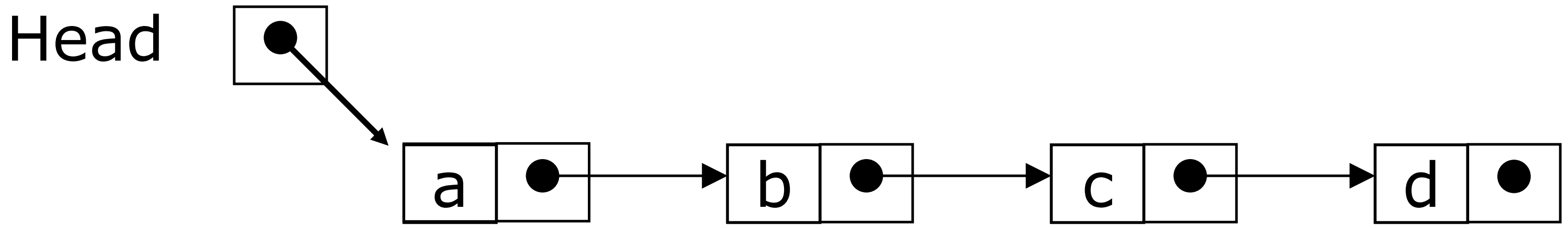

node
A
data
link


Data
Address
Node


10
20
30
40
40
Singly Linked List

Anatomy of a linked list

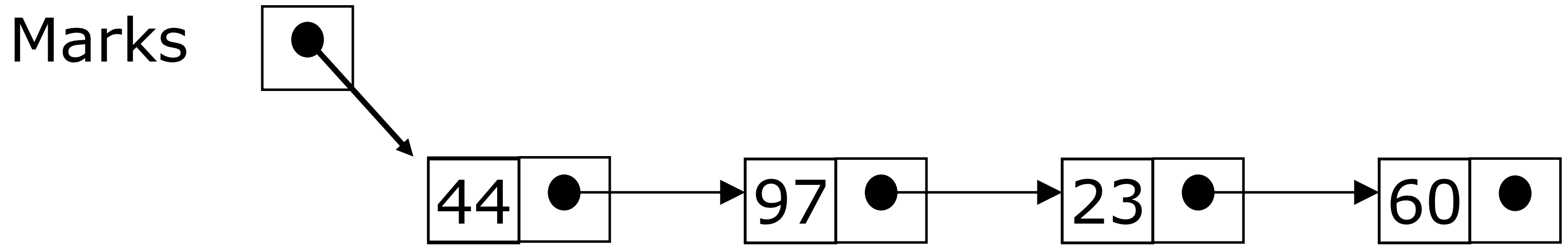
- A linked list consists of:
  - A sequence of nodes



Each node contains data (a value)
and a link (pointer) to the following node

The last node contains a null link

The list is referred to by pointer to first node, called the head/front


Marks
44
97
23
60

In C language, a linked list can be implemented using structure and pointers .

```
struct LinkedList{
        int data;
        struct LinkedList *next;
};
```

How Linked List are arranged in memory?

```
struct node
{
        int val;
        struct node  *next;
};
```

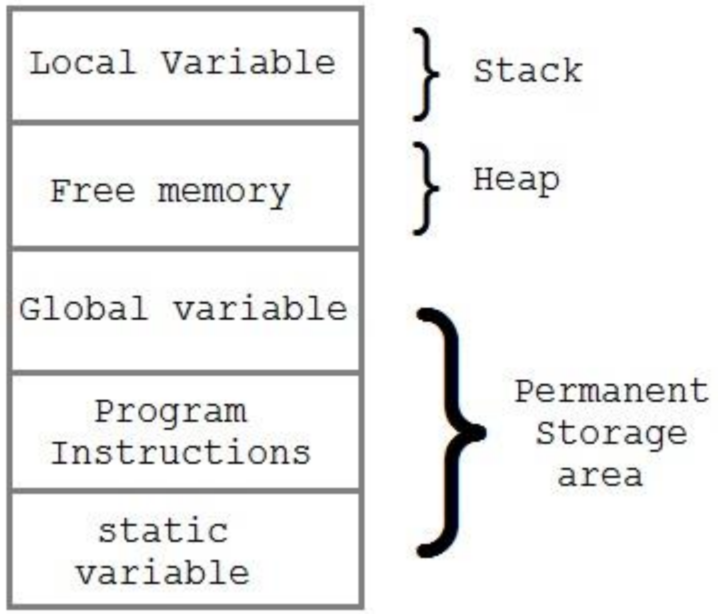
```
typedef struct node
{
        int val;
        struct node * next;
} node;
```

This structure contains a value ‘val’ and a pointer to a structure of same type. The value ‘val’ can be any value (depending upon the data that the linked list is holding) while the pointer ‘next’ contains the address of next block of this linked list. So linked list traversal is made possible through these ‘next’ pointers that contain address of the next node. The ‘next’ pointer of the last node (or for a single node linked list) would contain a NULL.

Function	Use of Function
malloc()	Allocates requested size of bytes and returns a pointer first byte of allocated space
calloc()	Allocates space for an array elements, initializes to zero and then returns a pointer to memory
free()	deallocate the previously allocated space
realloc()	Change the size of previously allocated space

Memory Allocation Process

Global variables, static variables and program instructions get their memory in permanent storage area whereas local variables are stored in area called Stack. The memory space between these two region is known as Heap area. This region is used for dynamic memory allocation during execution of the program. The size of heap keeps changing.



Basic Operations on a List

- Creating a list
- Inserting an item in the list
  - Insert at the beginning of the list
  - Insert at the end of the list
  - Insert at a given location in the list
- Deleting an item from the list
  - Deletion from the beginning
  - Deletion at the end of the list
  - Delete from a given location in the list
- Traversing the list

Pseudo-code for creating a list

```
typedef struct node{
   int data;
   struct node * next;
   }node;

node *CreateList()
{
          node * head=NULL;
          return head;
}
```

Pseudo-code for inserting an item at the beginning of the list

```
node *insert_beg(int element, node *head)
{
        node *tmp = (node *) malloc(sizeof(node));
        tmp->data = element;
        tmp->next = head;
        head = tmp;
        return head;
}
```

Pseudo-code for inserting an item at the end of the list

```
node *insert_end(int element, node *head)
{       if(!Is_Empty)
        {
                node *tmp = (node *) malloc(sizeof(node));
                tmp->data = element;
                tmp->next = NULL;
                node *crt = head;
                while(crt->next != NULL)
                        crt = crt->next;
                crt->next = tmp;
                return head;
        }       }
```

Pseudo-code to check if the list is empty or not

```
bool Is_Empty(node *head)
{
        if(head == NULL)
                return 1;
        else
                return 0;
}
```

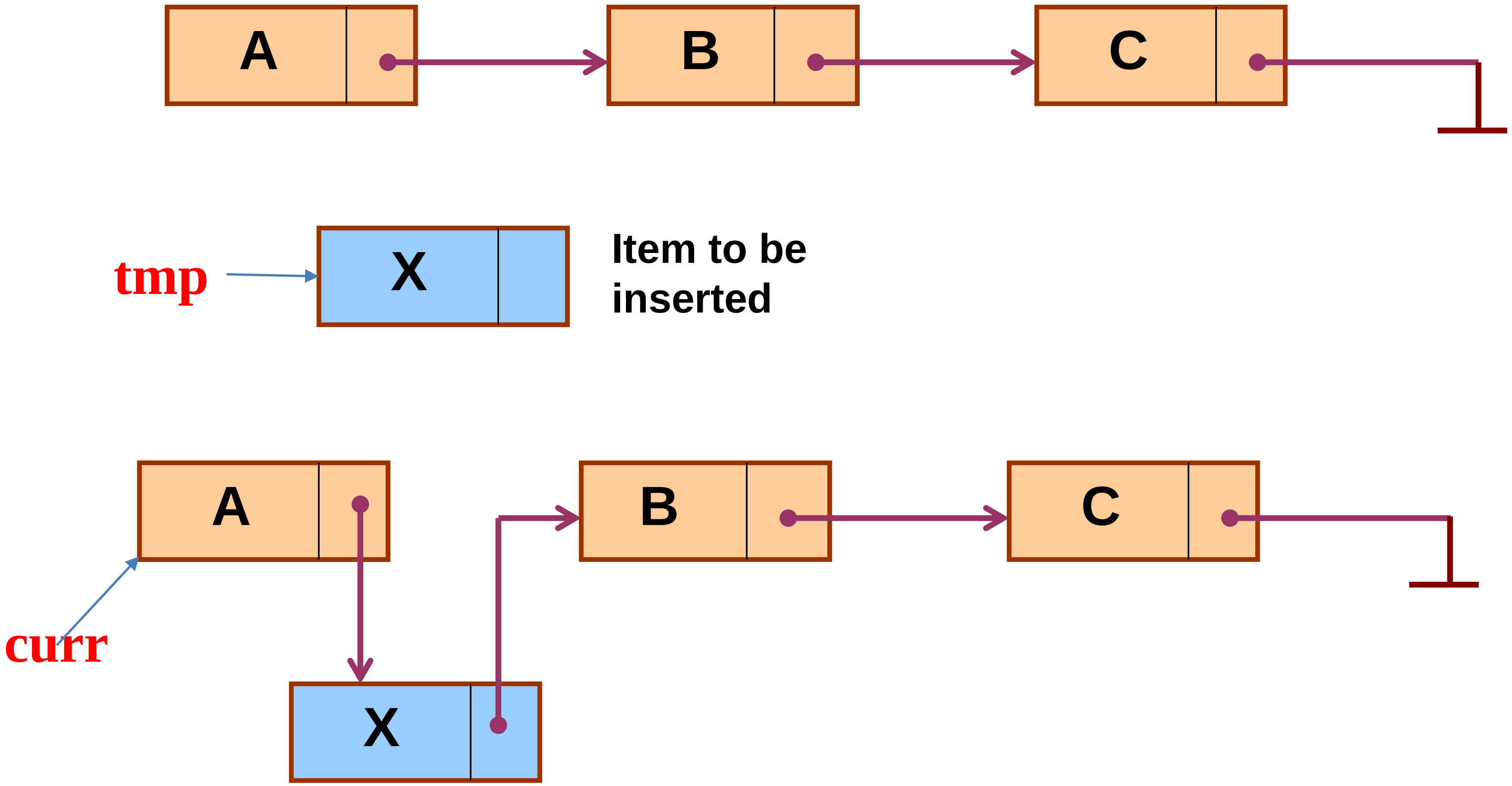
Pseudo-code to delete entire list

```
node *del_list(node *head)
{
        node *crt, *ptr;
        if(!Is_Empty)
        {
                node *crt=head;
                while(crt!=NULL)
                {
                        ptr=crt->next;
                        free(crt);
                        crt=ptr;
                }
                head=NULL;
                return head;
        }
}
```

Pseudo-code for inserting an item after a given node

```
node* insert_between(int element, int search_key, node *head)
{
        node *tmp = (node *)malloc(sizeof(node));
        tmp->data = element;
        node *crt = head;
        while(crt->data != search_key)
                crt = crt->next;
        tmp->next = crt->next;
        crt->next =tmp;
        return head;
}
```

Illustration: Insertion



Pseudo-code for inserting an item at position ‘p’

```
node* insert_pos(int element, int p, node *head)
{
        int counter=1;
        node *crt=head;
        node *tmp=(node *)malloc(sizeof(node));
        tmp->data=element;
        if(p = = 1)
                insert_beg(element, head);
        else
        {
                while(counter!=p-1)
                {
                        crt=crt->next;
                        counter++;
                }
                tmp->next=crt->next;
                crt->next=tmp;
        }       }
```

Deletion in Linked Lists

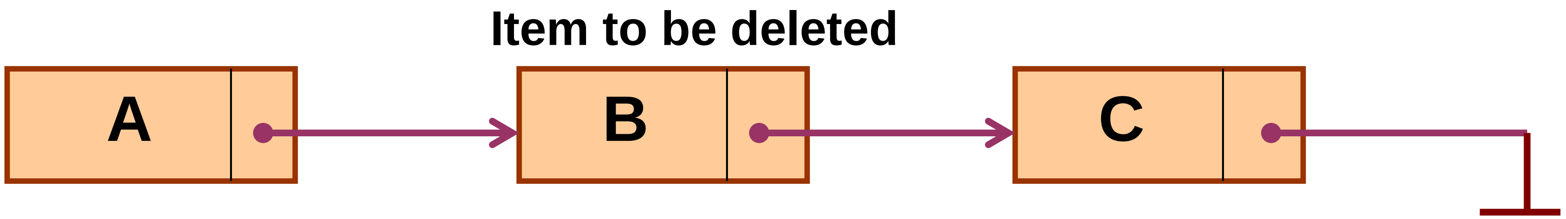
Pseudo-code for deleting an item from the beginning of the list

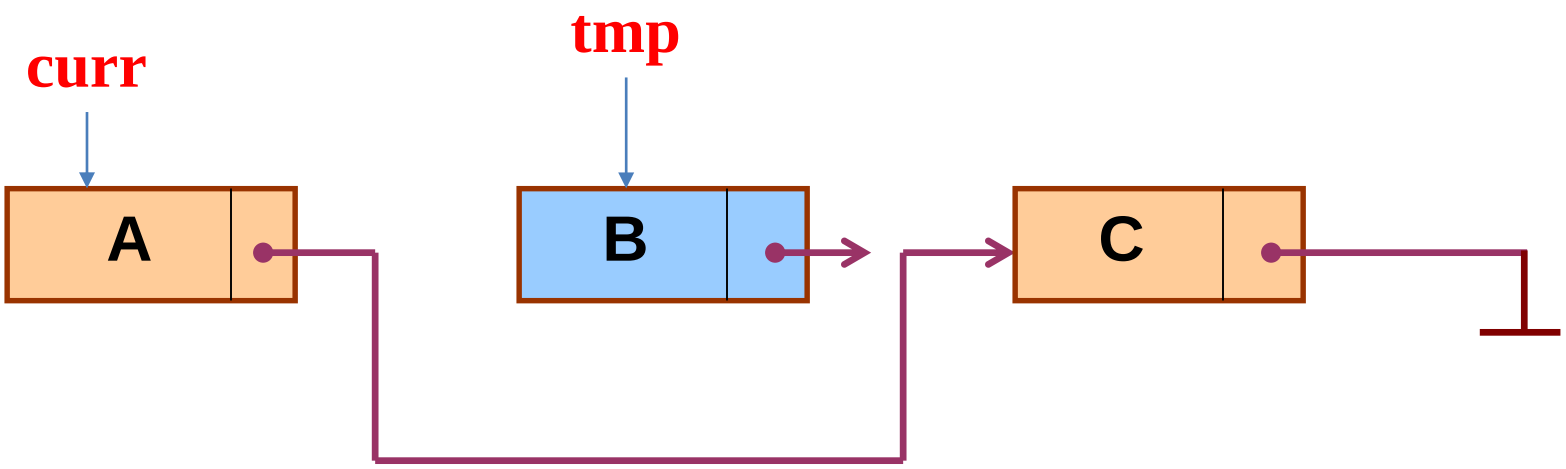
```
node *del_beg(node *head)
{
        if(!Is_Empty( ) )
        {
                node *tmp = head;
                head=head->next;
                free(tmp);
                return head;
        }
}
```

Pseudo-code for deleting the last item in the list

```
node *del_end(node *head)
{
        node *crt = head;
        if(!Is_Empty( ))
        {
                if(head->next = = NULL)
                {
                        head = NULL;
                        free(crt);
                }
                else
                {
                        while(crt->next ->next != NULL)
                                crt = crt->next;
                        free(crt->next);
                        crt->next = NULL;
                }
        }
        return head;    }
```

Illustration: Deletion





Pseudo-code for deleting a particular item from the list

```
node *del_item(node *head, int item)
{
        node *crt=head;
        if(head->data = =item)
                head=del_beg(node *head);
        else
        {
                while(crt->next->data != item)
                        crt = crt->next;
                node *tmp = crt->next;
                crt->next =tmp->next;
                free(tmp);
        }
        return head;
}
```

In essence ...

- For insertion:
  - A record is created holding the new item.
  - The next pointer of the new record is set to link it to the item which is to follow it in the list.
  - The next pointer of the item which is to precede it must be modified to point to the new item.
- For deletion:
  - The next pointer of the item immediately preceding the one to be deleted is altered, and made to point to the item following the deleted item.

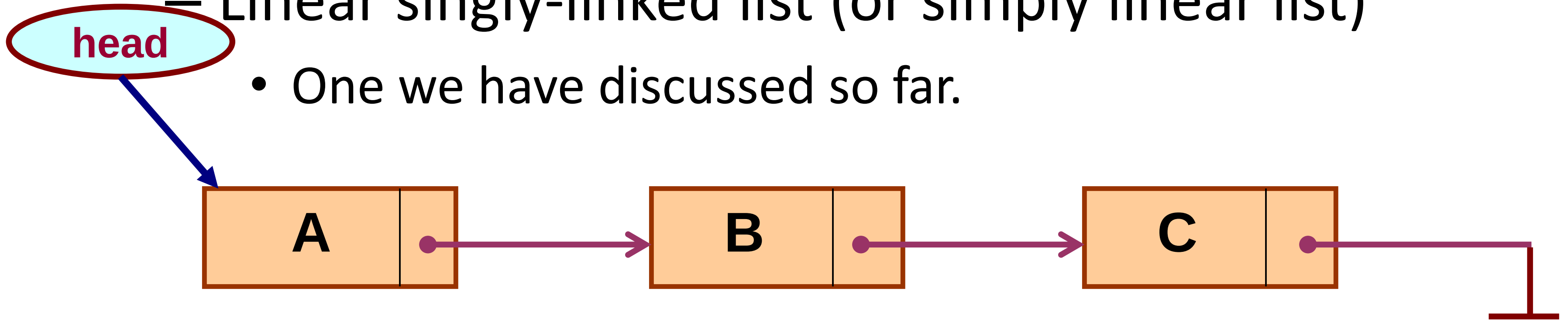
Array versus Linked Lists

- Arrays are suitable for:
  - Inserting/deleting an element at the end.
  - Randomly accessing any element.
  - Searching the list for a particular value.
- Linked lists are suitable for:
  - Inserting an element at any position.
  - Deleting an element from any position.
  - Applications where sequential access is required.
  - In situations where the number of elements cannot be predicted beforehand.

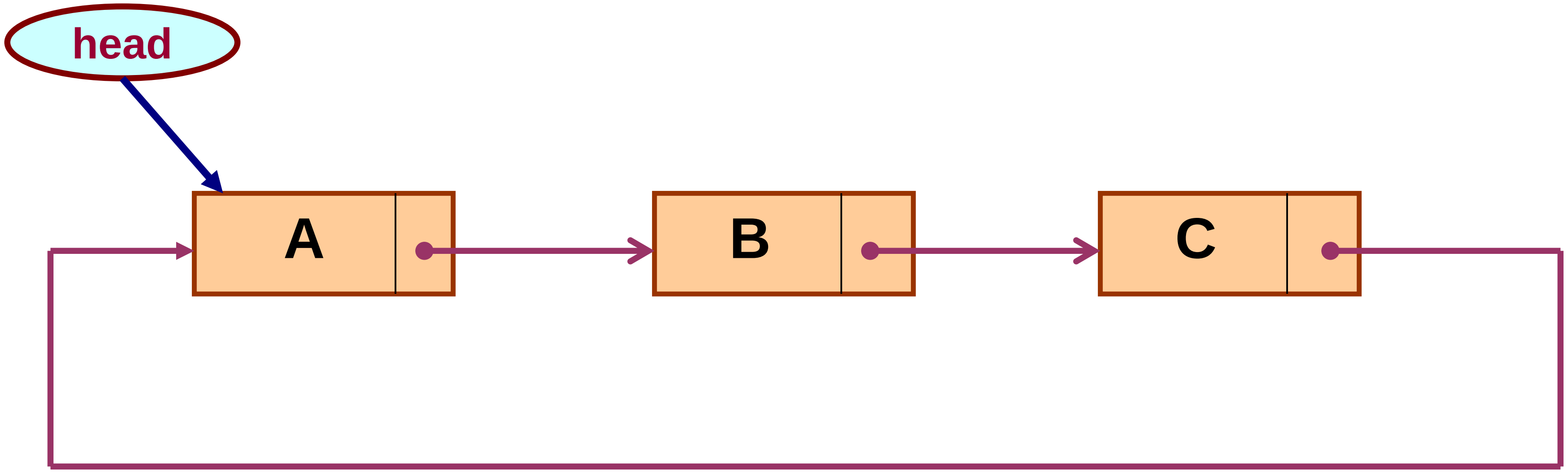
Types of Lists

- Depending on the way in which the links are used to maintain adjacency, several different types of linked lists are possible.

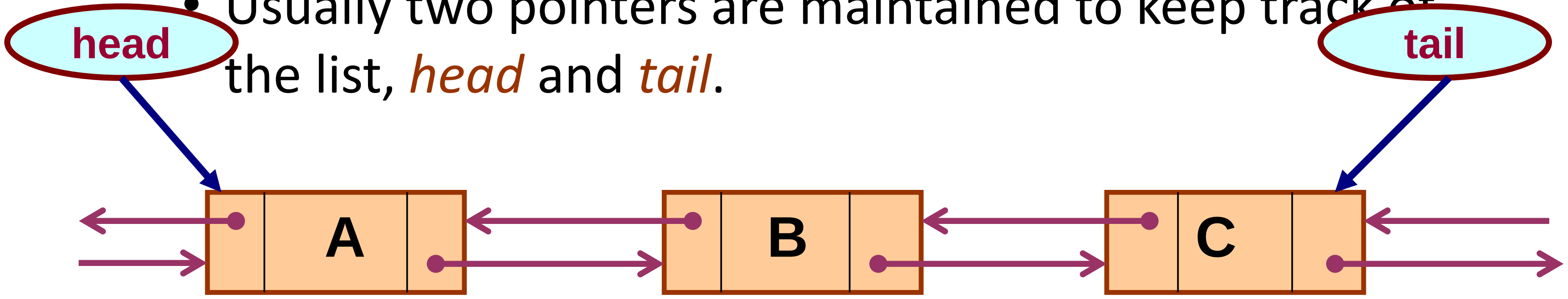
  – Linear singly-linked list (or simply linear list)
    - One we have discussed so far.



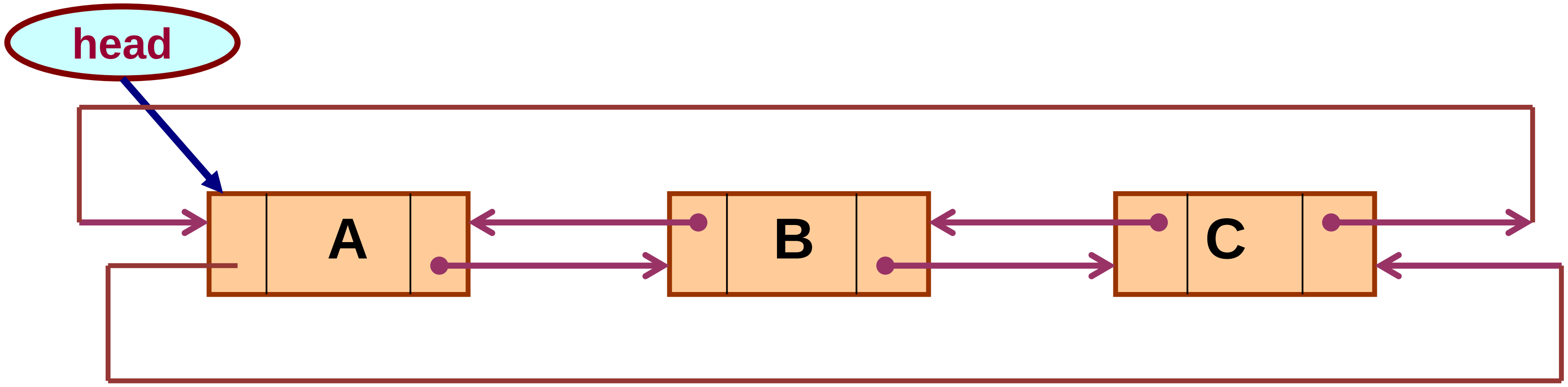
– Circular linked list
  - The pointer from the last element in the list points back to the first element.



- Doubly linked list
  - Pointers exist between adjacent nodes in both directions.
  - The list can be traversed either forward or backward.
  - Usually two pointers are maintained to keep track of the list, *head* and *tail*.



– Circular Doubly linked list
  - Pointers exist between adjacent nodes in both directions.
  - The list can be traversed either forward or backward.



List is an Abstract Data Type

- What is an abstract data type?
  - It is a data type defined by the user.
  - Typically more complex than simple data types like *int*, *float*, etc.
- Why abstract?
  - Because details of the implementation are hidden.
  - When you do some operation on the list, say insert an element, you just call a function.
  - Details of how the list is implemented or how the insert function is written is no longer required.

Case Study

Case 1: Address Book

- Problem Definition
  - A useful Address Book
  - Name
  - Phone Number
  - Email Address
- It includes two functions
  - Print address book
  - Add Entry in address book

Source Code (1/4)

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

typedef struct node{
        char name[20];
        long int ph_nbr;
        char email[20];
        struct node *next;
}node;

node *start_ptr = NULL;
```

Source Code (2/4)

```
void main()
{
        int choice;
        do
        {
                printf(“Enter your choice\n”);
                printf(“1. New Address data\n”);
                printf(“2. Display Address Book\n”);
                printf(“3. Exit\n”);
                scanf(“%d”, &choice);
                switch(choice)
                {
                        case 1: new_data();
                                        break;
                        case 2: output_data();
                                        break;
                        case 3: exit(0);
        }while(choice != 3);
}
```