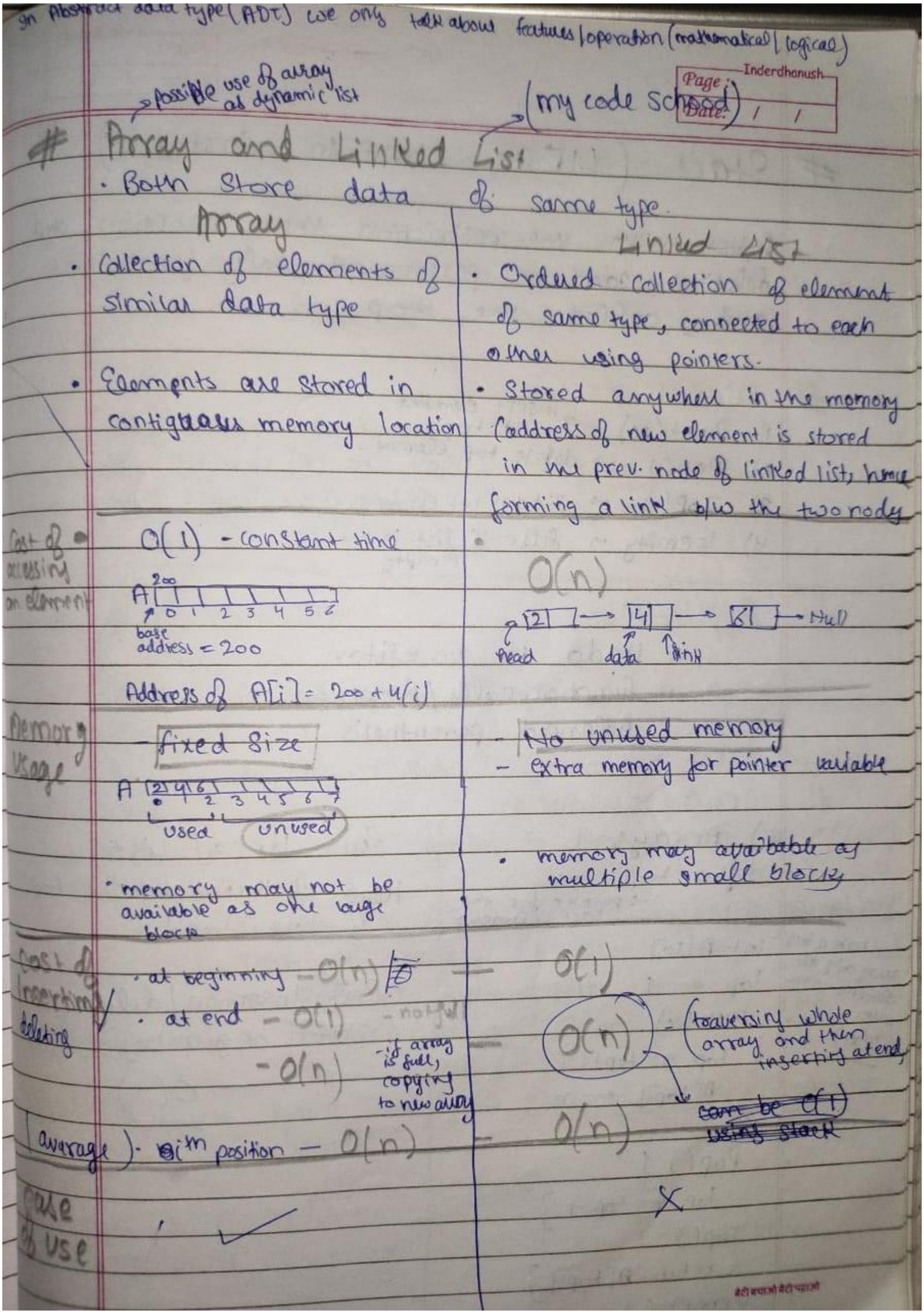
//Important===Floyd's cycle approach



In Abstract data type (ADT) we only talk about features/operation (mathematical/logical)

Inderdhanush
Page:
Date: / /

Array and Linked List

possible use of array as dynamic list

(my code school)

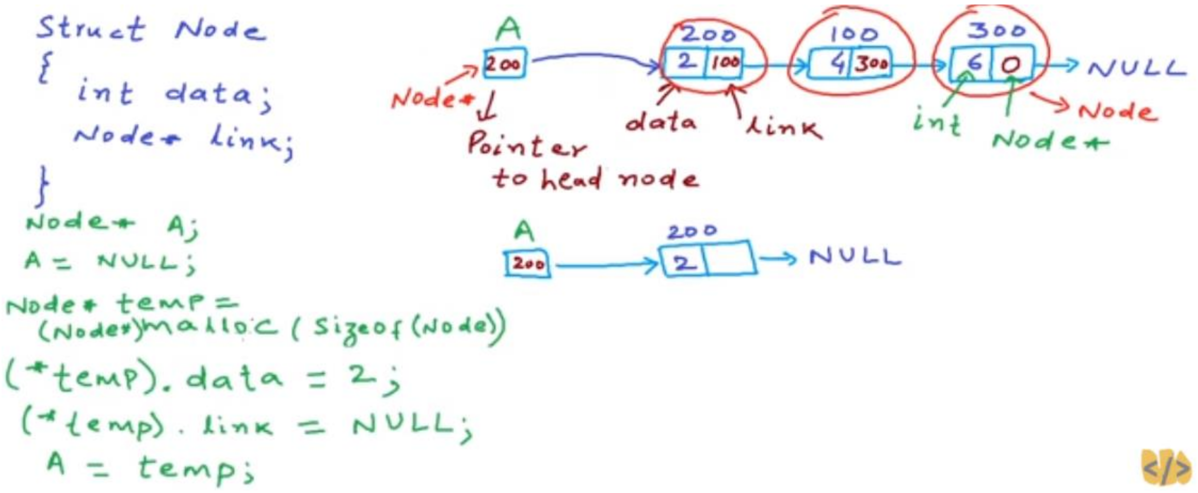
- Both Store data of same type.

	Array	Linked List
	• Collection of elements of similar data type • Elements are stored in contiguous memory location	• Ordered collection of element of same type, connected to each other using pointers. • Stored anywhere in the memory (address of new element is stored in the prev. node of linked list, hence forming a link b/w the two nodes
Cost of accessing an element	O(1) - constant time A [0 1 2 3 4 5 6] 200 base address = 200 Address of A[i] = 200 + 4(i)	O(n) head → [2] → [4] → [8] → Null data link
Memory usage	- fixed Size A [2 4 6 _ _ _ _ _] 0 1 2 3 4 5 6 7 used unused • memory may not be available as one large block	- No unused memory - extra memory for pointer variable • memory may available as multiple small blocks

Cost of inserting / deleting	• at beginning - O(n)
	• at end - O(1) - not full
	- O(n) - if array is full, copying to new array
	(average) • ith position - O(n)
		O(1)
		O(n) - (traversing whole array and then inserting at end,
		~~can be O(1) using stack~~
		O(n)

Case of use	✓	X

LINKED LIST BASIC(CREATING A LINK LIST):



```
Struct Node
{
  int data;
  Node* link;
}
Node* A;
A = NULL;
Node* temp =
  (Node*)malloc ( Sizeof (Node))
(*temp). data = 2;
(*temp) . link = NULL;
  A = temp;
```

|

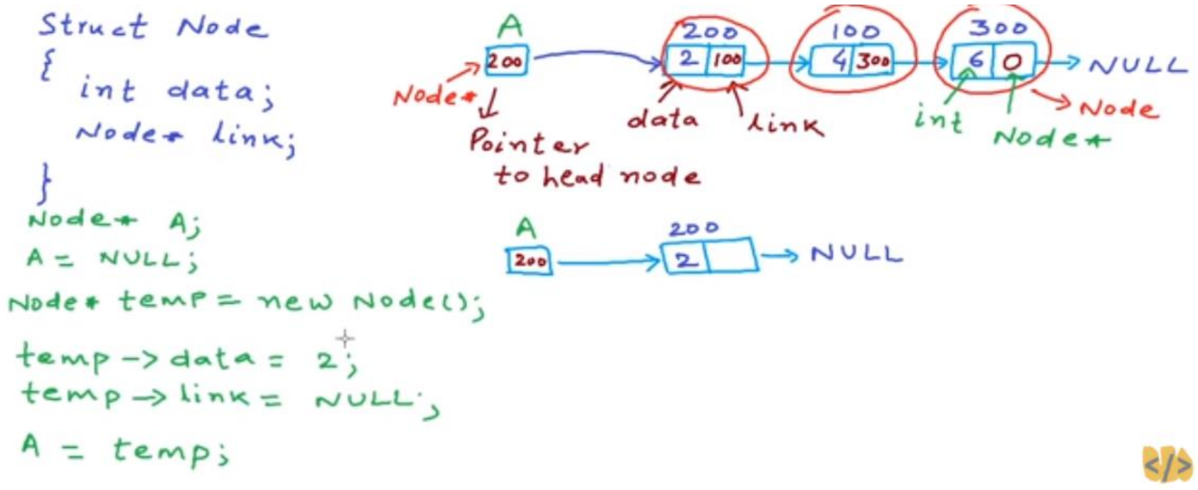
|

|

|

\./

NOW IN CPP WE DO IT LIKE THIS



```
Struct Node
{
  int data;
  Node* link;
}
Node* A;
A = NULL;
Node* temp = new Node();

temp -> data = 2;
temp -> link = NULL;

A = temp;
```

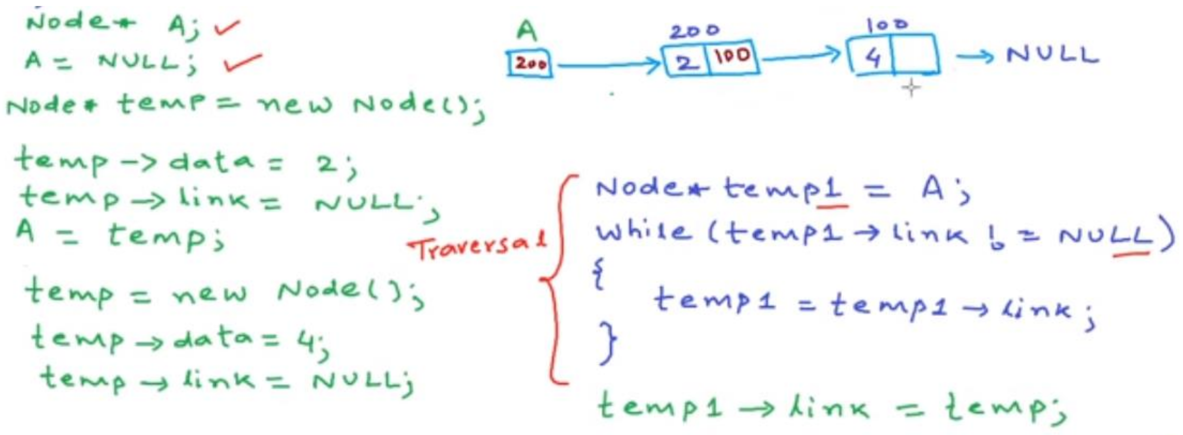
|

|

|

\./

ADDING ELEMENTS AT THE END:

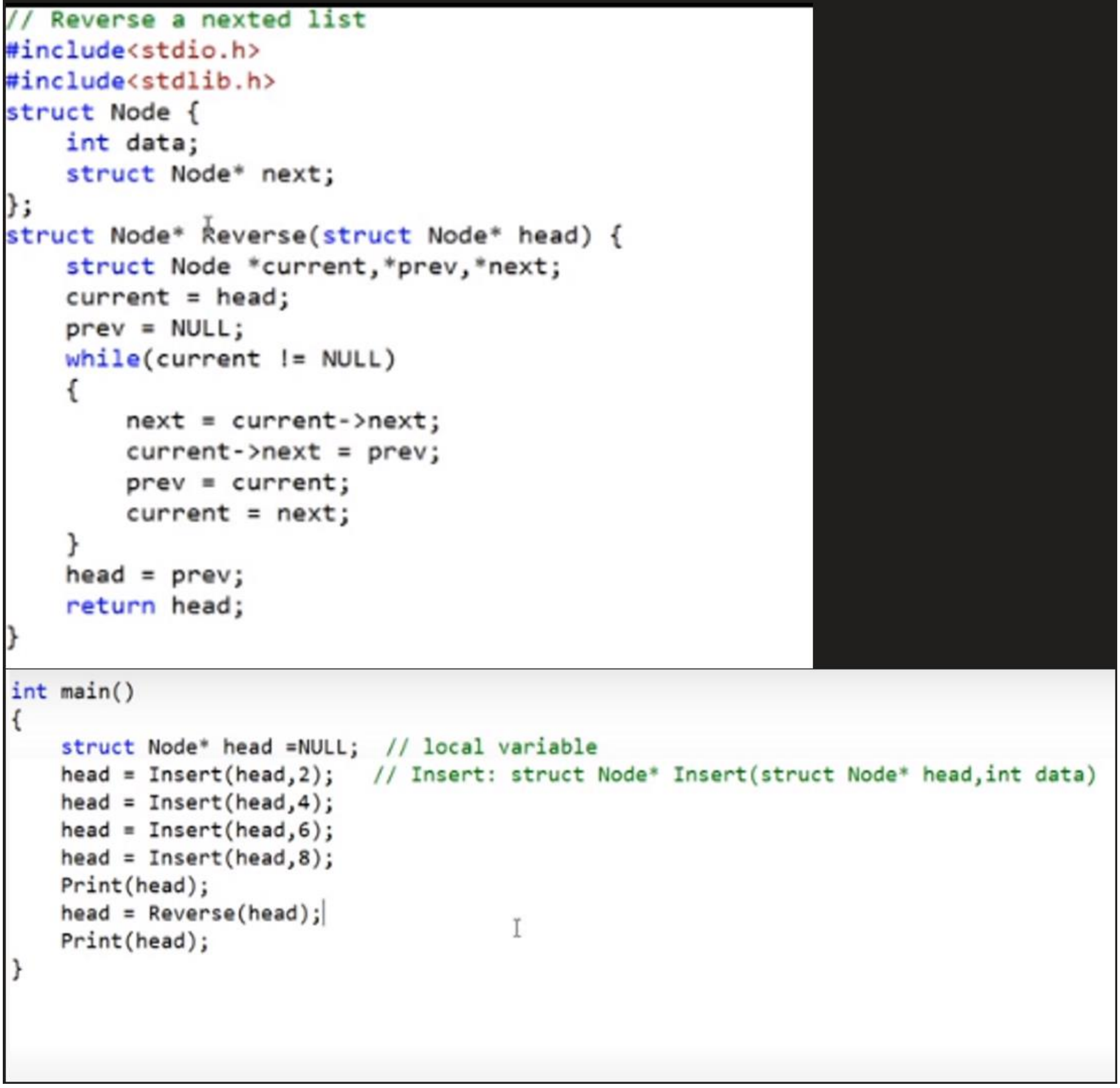


```
Node* A; ✓
A = NULL; ✓
Node* temp = new Node();
temp -> data = 2;
temp -> link = NULL;
A = temp;
temp = new Node();
temp -> data = 4;
temp -> link = NULL;
```

Traversal

```
Node* temp1 = A;
while (temp1 -> link != NULL)
{
    temp1 = temp1 -> link;
}
temp1 -> link = temp;
```

Reversing a Linked list:



```
// Reverse a nexted list
#include<stdio.h>
#include<stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* Reverse(struct Node* head) {
    struct Node *current,*prev,*next;
    current = head;
    prev = NULL;
    while(current != NULL)
    {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    head = prev;
    return head;
}
```

```
int main()
{
    struct Node* head =NULL;  // local variable
    head = Insert(head,2);   // Insert: struct Node* Insert(struct Node* head,int data)
    head = Insert(head,4);
    head = Insert(head,6);
    head = Insert(head,8);
    Print(head);
    head = Reverse(head);
    Print(head);
}
```