4. Linked Lists

24 April 2024 08:43

ARRAY LIST IMPLEMENTATION

Define some `last = -1.` → always stores index of last element of the array

Append (insert last)

Add element to array and increment last

Delete last

Decrement last; previous element at last is now considered out of bounds and junk

Insert front / Insert at position

Elements have to be moved to create space for new element to be inserted

Delete front / Delete at position

Same as above but elements moved towards front

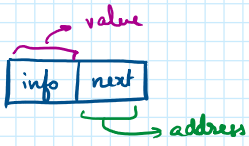
Disadvantage of Array List: Random insertions and deletions are time-consuming operations
You cannot change the size of array once defined.

LINKED LISTS

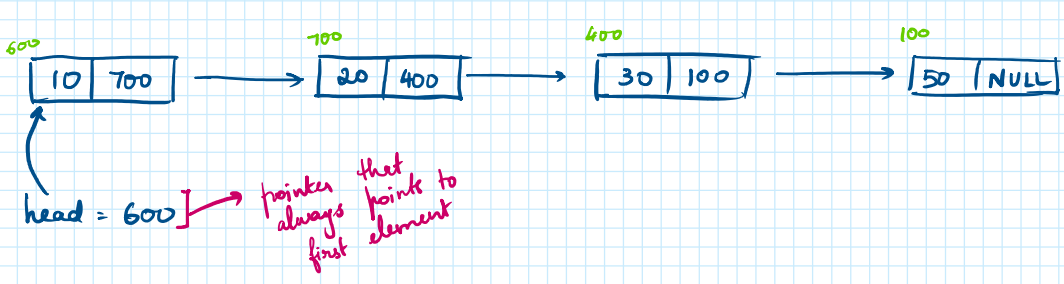
```
typedef struct node
{
    int info;
    struct node * next;
} NODE;
```

} Used to dynamically allocate memory for data

Each node:



We first allocate NULL to the "next" parameter because we do not know if there will be a successive element.



SINGLY LINKED LIST

```
typedef struct node
```

```
{
    int info;
    struct node * next;
} NODE;
```

function declarations

```
int main ()
{
    NODE * head = NULL;


}

NODE * insertFront ( NODE * head, int ele)
{
    NODE * newNode = malloc ( sizeof (NODE)); // NOTE: malloc may not always succeed; it is good to check
    newNode -> info = ele;
    newNode -> next = head;
    head = newNode;
    return head;
}

NODE * deleteFront ( NODE * head, int * pele)
{
    if (head == NULL)                    // accounting for empty linked list
        return head;
    NODE *p = head;
    *pele = head -> info;
    head = head -> next;
    free (p);
    return head;
}

void display ( NODE * head)
{
   while (head)
   {
```

```
    while (head)
    {
        printf ("%d", head -> info);
        head = head -> next;
    }
}
NODE * destroylist (NODE * head)
{
    while (head)
    {
        NODE *p = head;
        head = head -> next;
        free(p);
    }
    return head;
}
```