Lecture-01

Date: 06-02-2020

Shuvo Sir

Book Reference:

Theory and Problems of Data structures.
— seymor Lipschutz

Data Structure: The logical or mathematical model of a particular organization of data.

Two Factor Dependency:

(i) Must be rich enough to mirror the actual relationship of data in the real world.

(ii) structure should be simple enough so that anyone can ~~eas~~ effectively process the data

Major Operations:	Linear DS	Non-Linear DS
① Traverse	Array	Tree
② searching	Linked List	Graph
③ Insertion	stack	Tables
④ Deletion	Ques	sets

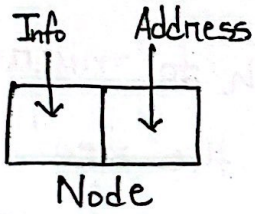
Complexity:

① Time Complexity ② Space Complexity

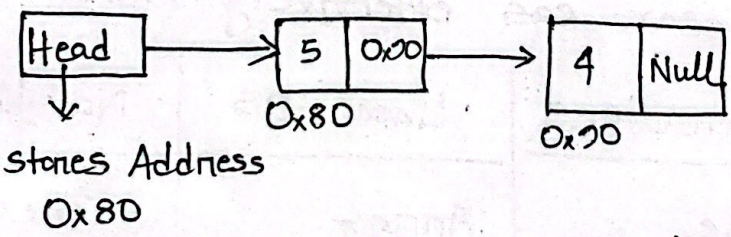
Lecture-2

Date: 09-02-2020

Linked List



Linked List is collection of Nodes.



Array vs Linked List

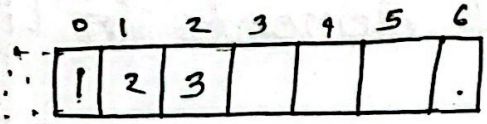
① Cost of accessing an Element:

Array	Linked List
O(1)	O(N)
Constant Time	Linear Time
Big-Oh	

(ii) Memory Requirement:

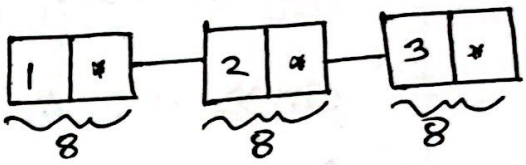
Array

A[7];



7x4 = 28 bytes

Linked List



8x3 = 24 bytes

(iii) Cost of Inserting an Element:

Array

1st Insert O(N)

Last Insert O(1)

Linked List

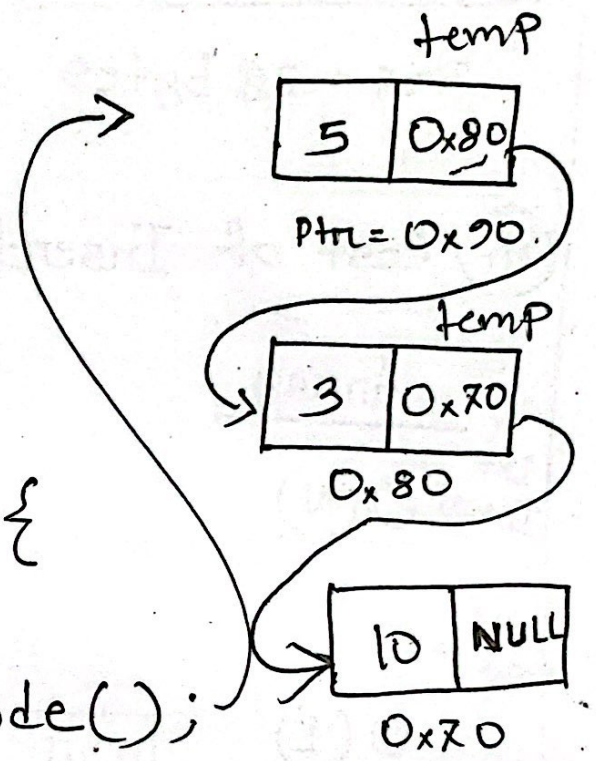
1st Insert O(1)

Last Insert O(N)

Node structure:

```
class node {
    public:
        int info;
        node *link;
};

node * Head, *temp = NULL;
```

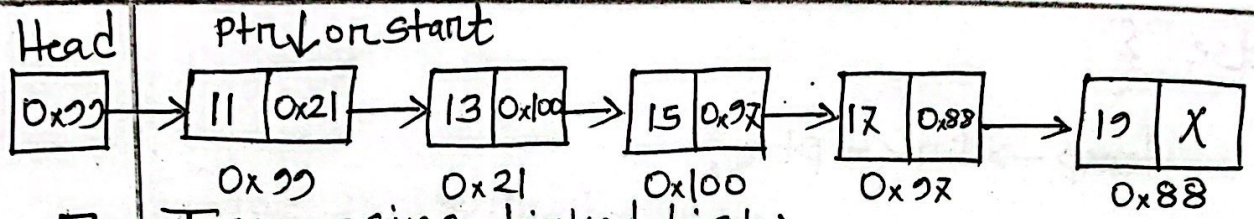
```
int main()
{
  int n, item i;
  cin>> n; // number of elements in List
  for(i=0; i<n; ++i) {
      cin>> item;
      creation (item);
  }
void creation (int data) {
        node *ptr;
        ptr = new node();
        ptr-> info = data;
        ptr-> link = NULL;
        if (Head == NULL) {
            Head = Ptr;
            temp = Ptr;
        }
}
```



```
else {
    temp -> link = ptr;
    temp = ptr;
  }
}
```

Lecture-03

Date: 13-02-2020



Traversing Linked List:

```
int traverse() {

    node *ptr;
    for(ptr=Head;;) {
        if (ptr->link != NULL) {
            cout << ptr->data;
            ptr = ptr->link;
        }
        else {
            cout << ptr->data;
            break;
        }
    }
}
```

Insertion in Linked List:

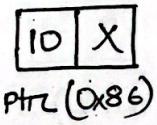
i. First Insertion

ii. Last Insertion

iii. Before a Given Item

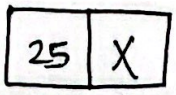
iv. After a Given Item

i. First Insertion:




```
void firstInsert () {
    int data;
    cin>> data;
    node *ptr;
    ptr = new node();
    ptr-> data= data;
    ptr-> link = Head;
    Head = ptr;
}
```

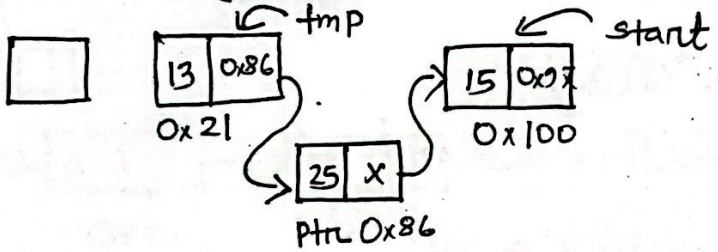
ii. Last Insertion:



Ptr 0x86

```
Void lastInsert () {
    int data;
    cin >> data;
    node *ptr;
    ptr = new node();
    ptr -> data = data;
    ptr -> link = NULL;
    node *start;
    for (start = Head; ;) {
        if (start -> link == NULL) {
            start -> link = ptr;
            break;
        }
        else {
            start = start -> link;
        }
    }
```

iii. Before a given Node or Data Insertion:



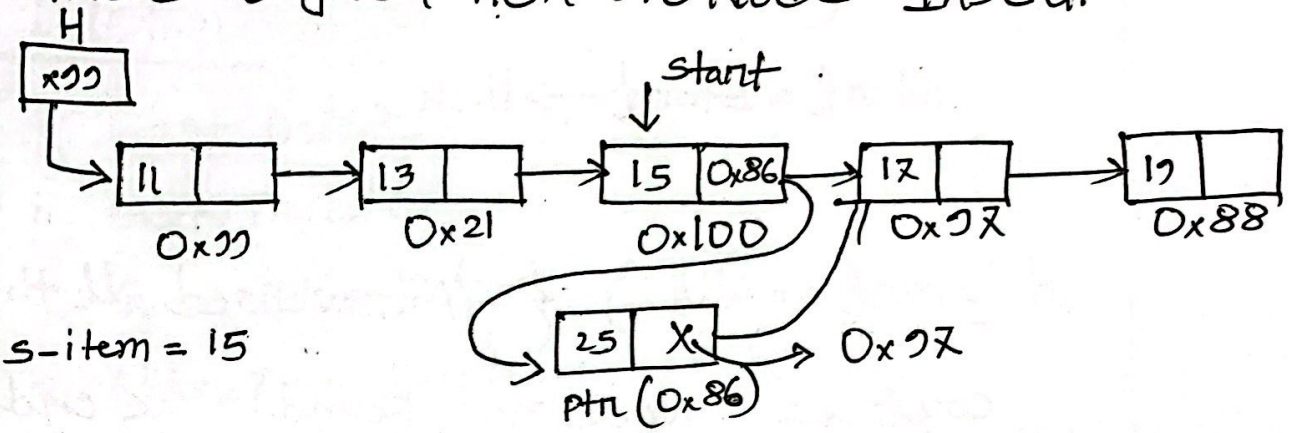

```
tmp->link = ptr;
ptr->link = start;

void beforeInsert() {
  // node creation
  int searchItem;
  node *start, *tmp;
  for (start = Head;;) {
    if (start->data == searchItem) {
      if (start==Head) {
        firstInsert();
      } else {
        CODE
      }
    } else {
      tmp = start;
      start = start->link;
    }
    if (start==NULL) {
      cout << "Item Not Found!" << endl;
    }
  }
```

```
else {
    // node creation
    ptr -> link = start;
    tmp -> link = ptr;
    break;
}
```

Lecture :- 04

Date :- 16-02-2020

iv. After a given item or Node Insertion :




```
for(start = Head; ;) {
    if (start -> data == s_item) {
        if (start -> link == NULL) {
            last_insert();
        }
        else {
            // node creation
            ptr -> data = item;
            ptr -> link = start -> link;
            start -> link = ptr;
            break;
        }
```

```
    else {
        start = start -> link;
    }
  if (start == NULL) { // traversed all the elements
    cout << " Node not Found!" << endl;
  }
}
```

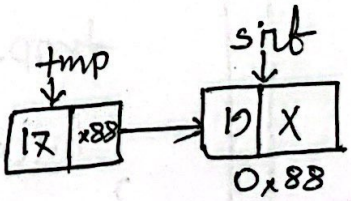
Deletion:

i. First Delete
ii. Last Delete
iii. Given Item Delete
iv. After Item Delete
v. Before Item Delete

i. First Delete:

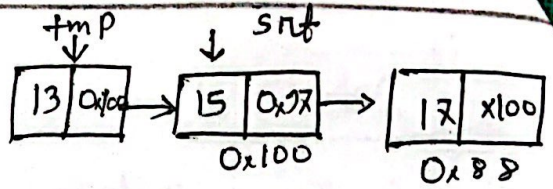
```
first_delete() {
    Head = Head -> link;
}
```

ii. Last Delete:




```
last_delete() {
  node *srt, *tmp;
  for(srt = Head; ;) {
     if (srt -> link == NULL) {
        tmp -> link == NULL;
     } break;
     else {
        tmp = srt;
        srt = srt -> link;
} }
```

iii. Given Item Deletion:



s-item = 15

```
given_item() {
for(srt = Head; ;) {
    if (srt->data == s-item) {
        if (srt == Head) {
            first_delete();
        }
        else if (srt->link == NULL) {
            last_delete();
        }
        else {
            tmp->link = srt->link;
            break;
        }
    else {
        tmp = srt;
        srt = srt->link;
    }
}
if (srt == NULL) {
cout << " Node not
    Found!" << endl;
}
```

Leeture-05

Date: 02-03-2020

iv.

```
afterDelete()
{
for(srt = Head; ;) {
    if (srt->data == searchItem) {
        tmp = srt->link;
        if(tmp->link == NULL) {
            srt->link = NULL;
            break;
        }
    else {
        srt->link = tmp->link;
        break;
    }
```

v.

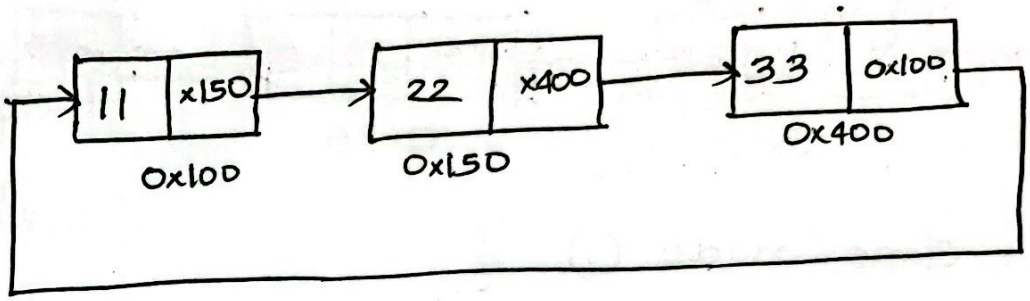
```
beforeDelete () {
    node *srt, *tmp, *prev = NULL;
    for (srt = Head; ;) {
        if (srt -> info == searchItem) {
            if (tmp == Head) { firstDelete(); } else {
            prev -> link = tmp -> link;
        } } break;
        else {
            prev = tmp;
            tmp = ~~Head~~; srt;
            srt = srt -> link;
        }
    }
```

① Overflow
② Underflow
③ Garbage Collection

Lecture - 05

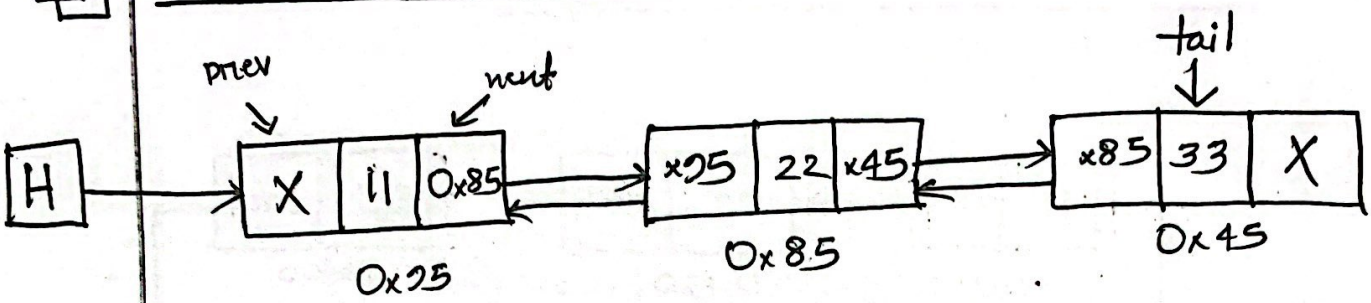
Date : 05-03-2020

Circular Linked List :



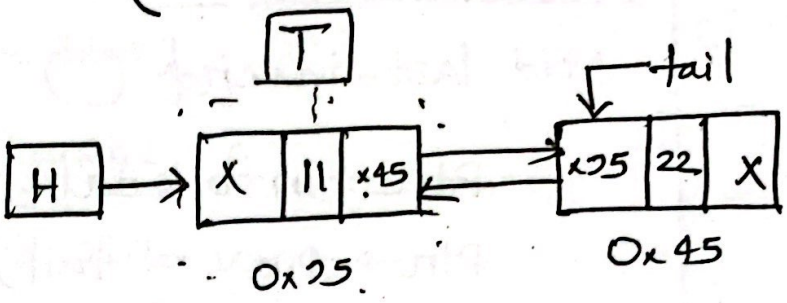

```
creation (data) {
    // ptr
    if (Head == NULL) {

    }
    else {

    }
    tmp->link = head;
}
```

Doubly Linked List:




```
class node() {
    int data;
    node *prev;
    node *next;
};
node *head = *tail = NULL;
Creation (data) {
    node *ptr;
    ptr = new node();
    ptr -> info = data;
    ptr -> prev = NULL;
    ptr -> next = NULL;
```

```
if (head == NULL) {
    head = ptr;
    tail = ptr;
}
else {
    tail -> next = ptr;
    ptr -> prev = tail;
    tail = ptr;
}
}
```



Basic Operations:

① First Insert:

```
void first_insert() {
    ptr -> info = data;
    ptr -> prev = NULL;
    ptr -> next = Head;
    Head -> prev = ptr;
    Head = ptr;
}
```

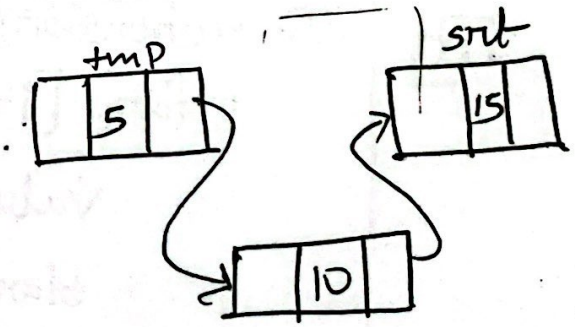
② Last Insert:

```
void last_insert () {
    ptr->info = data;
    ptr->prev = tail;
    tail->next = ptr;
    ptr->next = NULL;
    tail = ptr;
}
```

③ After Insert:

```
void after_insert() {
    for (srt = head; ) {
        if (srt->info == s_item) {
            tmp = srt->next;
            if (tmp == NULL) {
                last_insert();
            }
            else {
                // new node - ptr.
```

```
        ptr -> prev = srt;
        ptr -> next = tmp;
        srt -> next = ptr;
        tmp -> prev = ptr;
        break;
      }
    else {
      srt = srt -> next;
      tmp = srt -> next;
    }
}
```



④ Before Insert :

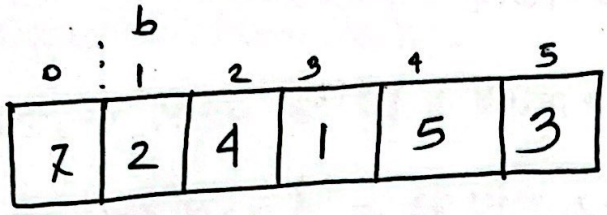
```
void before_insert() {
    for(srt = head; ;) {
        if (srt -> info == s_item) {
            ptr -> next = srt -> prev;
            srt -> prev = ptr;
            ptr -> prev = tmp -> next;
            tmp -> next = ptr;
        }
```

Lecture-06

Date: 12-03-2020

Sorting Algorithms

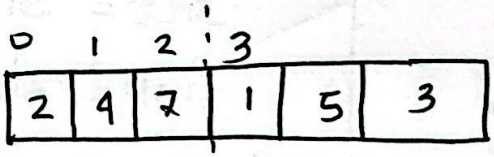
Insertion Sort:



Value = A[i]

blank = i

while (b>0 && A[b-1] > Value)



val = 1
blank = 3

Algorithm:

```
InsertionSort (A, n) {
    for [i = 1 to (n-1)] {
        value ← A[i];
        blank ← i;                          } C1 (n-1 times executed)
        while (blank>0 && A[blank-1] > value){
            A[blank] = A[blank-1];          } C2
            blank--;
        }
        A[blank] = value;                   } C3 (n-1 times executed)
    }
}
```

Complexity :

① Best Case: 1, 2, 3, 4, 5

$$T(n) = c_1(n-1) + c_2 \cdot 0 + c_3(n-1)$$

$$= (c_1 + c_3)(n-1)$$

$$= ~~an+b~~ \quad an^{1}+b$$

Complexity : $O(n)$

② Worst Case : 5, 4, 3, 2, 1

$$T(n) = c_1(n-1) + c_3(n-1) + c_2(1+2+3+\cdots+n-1)$$

$$= (c_1+c_3)(n-1) + c_2\left[\frac{n(n-1)}{2}\right]$$

$$= an^{2}+bn+c$$

Complexity : $O(n^2)$

Selection Sort:

0	1	2	3	4	5
2	7	4	1	5	3

Algorithm:

```
selectionSort (A, n) {
    for (i = 0 to n-2) {
        iMin = i } (C1) (n-1)
        for (j = 0 to n-1) {
            if A[j] < A[iMin] { } (C2)
                iMin = j;
            }
        swap (A[i], A[iMin]); (C3) (n-1)
    }
}
```

Complexity :

① Best Case : 1, 2, 3, 4, 5

$$T(n) = (c_1+c_3)(n-1) + c_2[(n-1)+(n-2)+\cdots+2+1]$$

$$= (c_1+c_3)(n-1) + c_2\left[\frac{n(n-1)}{2}\right]$$

$$= an^2+bn+c$$

Complexity : $O(n^2)$

② Worst Case : 5, 4, 3, 2, 1

Complexity : $O(n^2)$

same complexity for both Best and Worst case.