Linked List

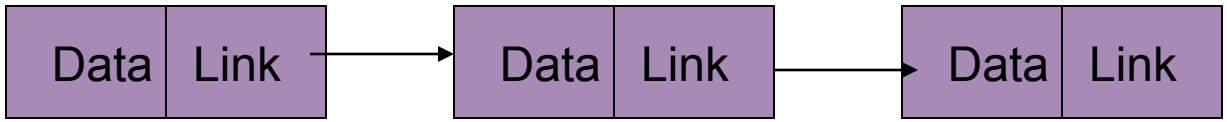
bitCode technologies

Drawbacks of Arrays

- Array does not grow dynamically (i.e length has to be known)
- Inefficient memory management.
- In ordered array insertion is slow.
- In both ordered and unordered array deletion is slow.
- Large number of data movements for insertion & deletion which is very expensive in case of array with large number of elements.
  Solution : Linked list.

Introduction to Linear Linked List

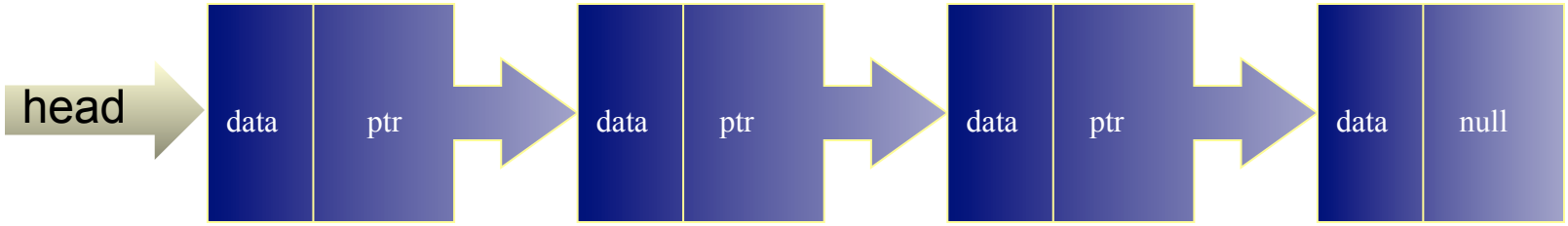
- We can overcome the drawbacks of the sequential storage.
- If the items are explicitly ordered, that is each item contained within itself the address of the next Item.
- Such an explicit ordering gives rise to a data structure called Linear Linked List.
- In Linked Lists elements are logically adjacent, need not be physically adjacent.



Linked List Structure

- Each Item in the list is called a node and contains two fields, an data field and next address field.
- The data field holds the actual element on the list.
- The link or the next address field contains the address of the next node in the list.
- Such an address which is used to access a particular node, is known as a pointer.
- The entire linked list is accessed from an external pointer Head, which points to the first node of the list.
- The next field of the last node in the list contains a special value, known as null, which is not a valid address.
- This null pointer is used to signal the end of the list.

Linked List



- Here the head is a pointer which is pointing to the first node of the list.
- The entire list can be accessed through head.
- So to maintain a list is nothing but maintaining a head pointer for the list.

Operations

- Insert ( )

1. Create a new node.
2. Set the fields of the new node.
3. If the linked list is empty insert the node as the first node.
4. If the node precedes all others in the list then insert the node at the front of the list.
5. Repeat through step 6 while information content of the node in the list is less than the information content of the new node.
6. Obtain the next node in the list.
7. Insert the new node in the list.

Operations

- Delete ( )

1. If the list is empty then write underflow and return.
2. Repeat through step 3 while the end of list has not been reached and the node has not been found.
3. Obtain the next node in the list and record its predecessor node.
4. If the end of the list is reached and node not found then write node not found and return.
5. If found delete the node from the list.
6. Free the node deleted.
7. Set the links of the nodes one which follows the deleted node and one which precedes the deleted node.

Operations

- Search ( )

  Start from the head node. Compare the key with the data item of each node. If match not found and end of list is reached then the element being searched is not present.

  If found return it's position.

- Display ()

  Start with the first node. Print the data of the current node. Get the address of the next node, and make it as current node.

  Continue the process un till the next node is NULL.

Implementation of Linked List - Node

```
class Node {

  int data;
  Node * next;

public:
  Node ( int data ) ;
  int getData ();
  void setData ( int data );
  Node * getNext ();
  void setNext ( Node * next );

};
```

Implementation of Linked List

```
class SinglyLinkedList {

  Node * head;

public:
        SinglyLinkedList();
        void insert ( int data );
        void insertByPos ( int data, int pos );
        void delByVal ( int val );
        void delByPos ( int pos );
        void display ();
        int search ( int val );
        void printReverse();

};
```