



Linked List



ADT Single Linked List

Objects:

Finite number of nodes having the same type. Nodes are linked together as in a chain with the help of a field in them. A special variable points to the first element of the chain.



ADT Single Linked List ...

Operations:

init_l(cur) – initialise a list

empty_l(head) – boolean function to return true if list pointed to by head is empty

atend_l(cur) – boolean function to return true if cur points to the last node in the list

insert_front(target, head) – insert the node pointed to by target as the first node of the list pointed to by head

insert_after(target, prev) – insert the node pointed to by target after the node pointed to by prev

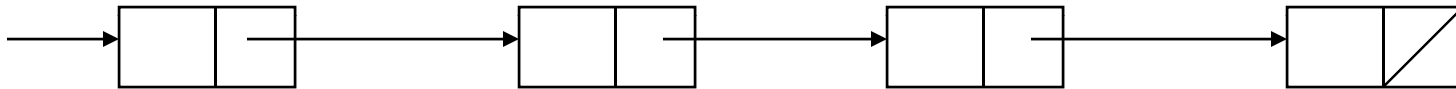
delete_front(head) – delete the first element of the list pointed to by head

delete_after(prev) – delete the node after the one pointed to by prev



ADT Single Linked List ...

head





Linked List Problems

1. Read integers from a file and arrange them in a linked list
(a) in the order they are read, (b) in reverse order.
2. Search for a key in (a) an unordered list, (b) an ordered list.
Return the node if key found and delete the element from original list.
3. Find the size of a list
4. Write a boolean function **equal_list**.
5. Print a list (a) in the same order, (b) in the reverse order.
6. Append a list at the end of another list.
7. Delete the n^{th} Node of a list.



Linked List Problems ...

8. Write a boolean function **ordered** to return true if the information in the list are ordered.
9. Merge two sorted lists
10. Insert a target node before a specified node.
11. Delete a list.
12. Reverse a list.
13. Sort a list.



C pointer implementation of Single Linked List

```
typedef struct nodetag {  
    T      info;  
    struct nodetag * next;  
} nodetype;  
  
nodetype *head, *cur, *prev, *next, *target;  
  
nodetype * createnode(T item) {  
    nodetype *head = NULL;  
    if ((head = (nodetype *)malloc(sizeof (nodetype))) == NULL)  
        perror("malloc error");  
    else {head->info = item; head->next = NULL;}  
    return head; }
```



Single Linked List Operations

```
nodetype * init_l() {return  NULL};

int empty_l(nodetype *head) { return (head == NULL)};

int atend_l(nodetype *cur) { if (cur == NULL)
                                return 0;
                                else return (cur->next == NULL);}

void insert_front(nodetype *target, nodetype **phead) {
    target ->next = *phead;
    *phead = target;
}
```




Linked List Operations ...

```
void insert_after (nodetype *target, nodetype *prev) {  
    nodetype *cur;  
    cur = prev -> next;  
    target -> next = cur;  
    prev -> next = target;}
```

```
void delete_front( nodetype **phead) {  
    nodetype *cur;  
    cur = *phead;  
    *phead = (*phead)->next;  
    free(cur);  
}
```



Linked List Operations ...

```
void delete_after (nodetype *prev) {  
    nodetype *cur;  
    if ( !(atend_l(prev))){  
        cur = prev -> next;  
        prev -> next = cur ->next;  
        free(cur);  
    }  
}
```

The concept of Linked List is applied in all linked structures.



Implementation of Single Linked List using Array

- Nodes consist of an info field and a cursor field.
- Cursors represent the array index of the next node.
- There is a node-pool of such nodes.
- All the linked lists are stored in the same node-pool.
- An “*available-nodes*” list maintains all the unused nodes.



Implementation of Single Linked List using Array

```
#define mne    <max no. of nodes>
```

```
typedef struct { T      info;  
                  int    cursor;  
                } nodetype;
```

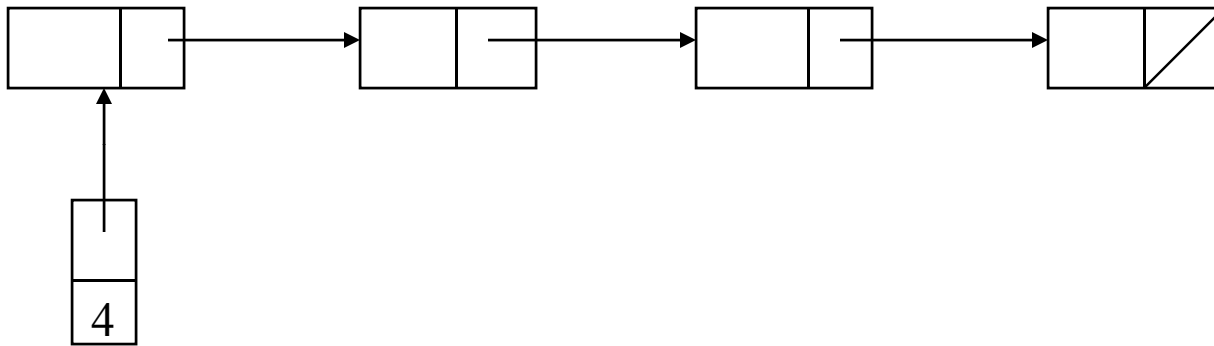
```
nodetype      node_pool [mne];
```

```
int  head1, head2, head3, cur, prev, avail;
```

Write all the ADT operations of ADT Single Linked List for the implementation using array.



Variations of SLL



- List with a special head node containing the length of the list as info.
- One can also store the max, min and other statistics if required.



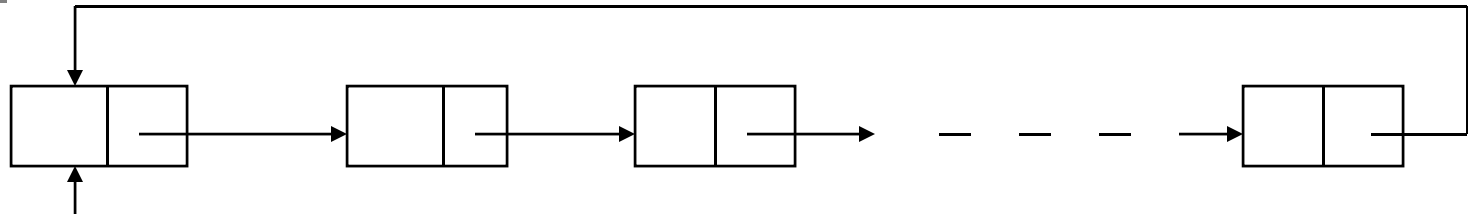
Variations of SLL ...



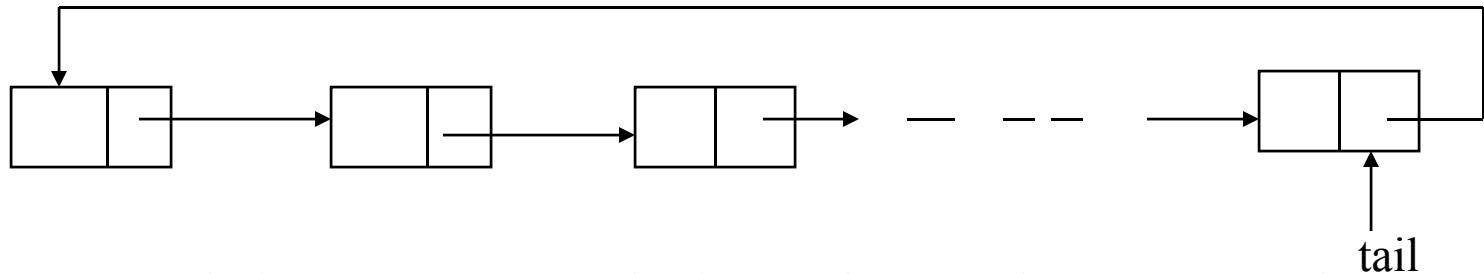
- List with a dummy first node.
- Uniform insertion and deletion operation –
insert_front and delete_front are not necessary.



Circular SLL



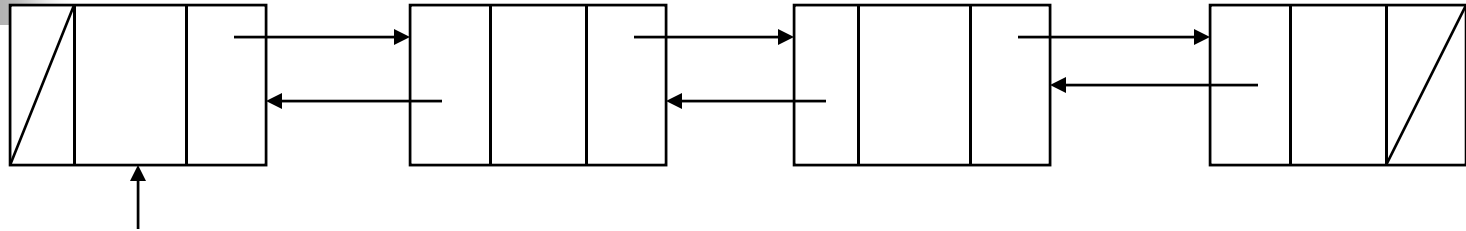
- Any node to any node traversal is possible
- But to enter a node after the last element, one has to traverse the whole list
- Using a tail pointer instead of a head pointer may be useful



- In SLL it is not easy to find out the predecessor node
- Solution is Doubly Linked List



Doubly Linked List



head
typedef struct nodetag {

T info;

struct nodetag *prev, *next;

} nodetype;

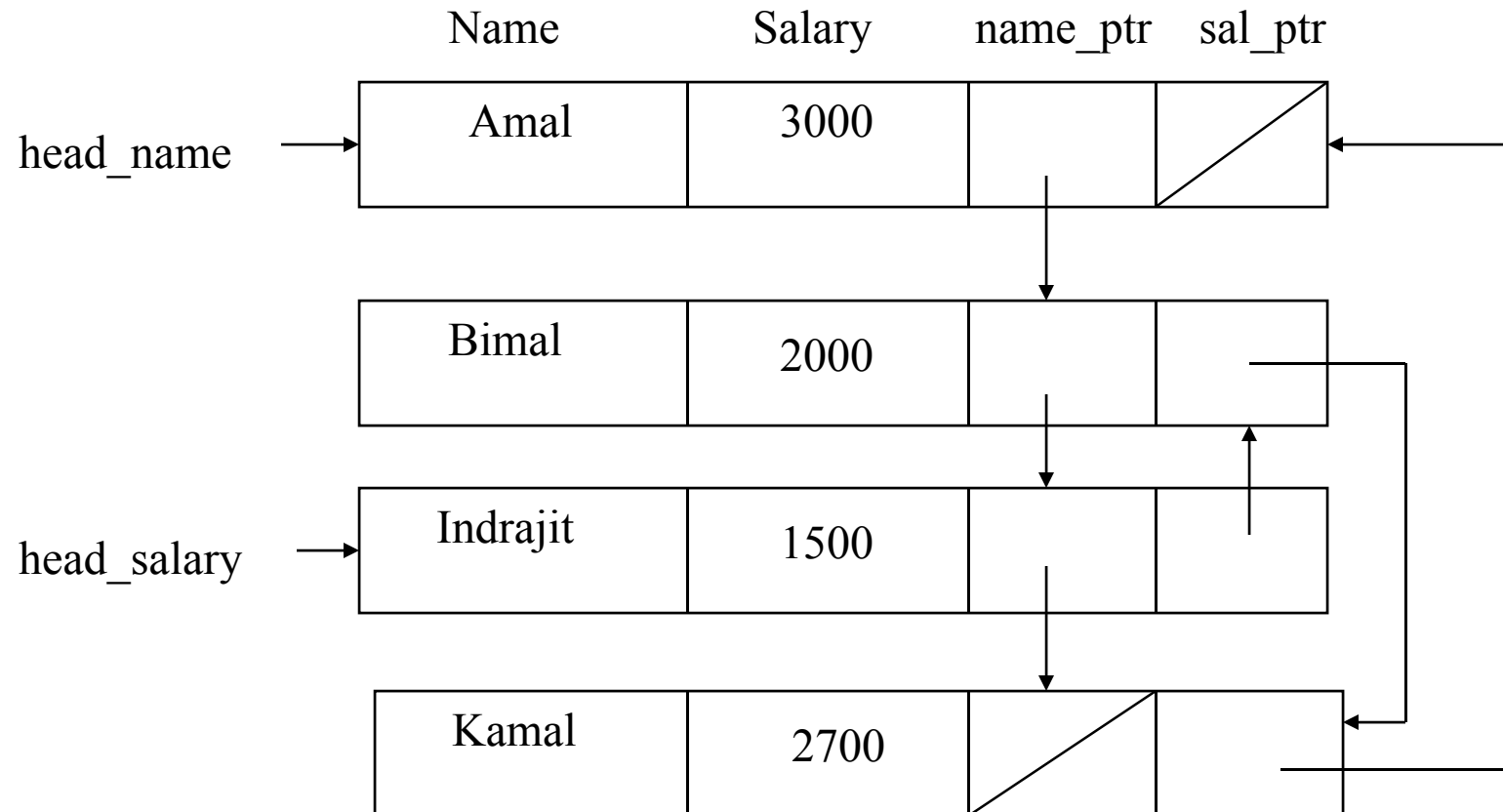
- Two-way movement becomes easy
- Current node can be deleted.
- DLL with special head node is possible.
- DLL with two dummy nodes at the two ends is also a possible configuration.
- Circular DLLs are also useful
- Find the operations possible for ADT DLL and implement them



Multi-way Linked List

When different links can be used to traverse different nodes in a list, we call it a multi-way linked list or simply, multi-list.

Example:





Sparse Matrix using Multi-list

0	0	0	55	0
34	0	0	28	0
0	0	0	0	0
0	0	83	0	0

