Linked List of integers

data next

-25 1024

18 956

956

13 785

data next

1024

head = 600

6 NULL (0)

785

Create LL data next

-25 ~~NULL~~ 1024

~~newnode = 956~~

data next ~~temp = 956~~ data next

13 ~~NULL~~ 785

18 ~~NULL~~ 956

~~newnode = 600~~ ~~newnode = 1024~~

head = 600 ~~temp = 1024~~

~~temp = 600~~

6 NULL

newnode = 785

temp = 785

6 NULL (0)

-25 1024

13 785

18 956

data next

LL =

head = 600 956 1024 785

temp temp temp temp temp

Traverse the above linked list

struct node \*temp;

temp = head; // 600

while (temp != NULL) // 600 != 0, T 956 != 0, T 1024 != 0, T 785 != NULL, T NULL != NULL, F

{

printf(“%d “,temp->data); // 18, -25, 13, 6

temp = temp->next; // 956, 1024, 785,

}

sumit != sumit F

sumit != pooja T

sumit != rajesh T

RAM (Random Access Memory)

Keyboard Monitor

t 🡨 O/p

name (variable)

t

actually we typed sumit bcoz as we type next letters/chars, previous characters get erased

buffer (physical memory in RAM)

var data of newnode ask

Keyboard RAM

18

18 (ENTER) will flush the buffer

Enter

while (ask == ‘y’ || ask == ‘Y) // ENTER is != ‘Y’ So loop is false & terminates

{

……

}

6 NULL (0)

13 785

-25 1024

18 956

To Add a Node at the beginning of LL

head = 600 956 1024 785

data next

89 600

newnode = 4589

head = 4589

newnode->next = head;

head = newnode;

To Add a Node at the end of LL

13 785

-25 1024

18 956

6 ~~NULL (0)~~ 805

LL 🡺

head = 600 956 1024 785

temp temp temp temp

data next

90 NULL (0)

newnode = 805

newnode->next = NULL;

// newnode is not a part of above LL

// traverse the LL from head to current last node

for (temp=head;temp->next != NULL;temp = temp->next); // 956!=0, T 1024!=0, T

// 785!=0, T 0!=0, F

temp->next = newnode;

6 NULL (0)

18 956

-25 1024

13 785

To delete a node at the beginning of LL

~~head = 600~~ head = 956

temp

temp = head;

head = head->next;

free(temp); // free() is opposite of malloc(). malloc() allocates memory whereas

// free() deallocates/releases the memory (frees the memory)

6 NULL (0)

// Only 1 node remains in LL, the code to execute is same. No special case reqd

~~head = 785~~ head = NULL (0)

temp

To delete a node at the end of LL

1. Here if there are multiple nodes in LL, then the address of head node will not change.
2. If the LL contains a single node, then it will change the address of head node.

13 ~~785~~ 0

6 NULL (0)

-25 1024

18 956

head = 600 956 1024 785

~~temp~~ ~~temp~~ temp

Traverse to 2nd last node & delete it

temp = head; // 600

while (temp->next->next != NULL) // 1024 != 0, T 785 != 0, T 0 != 0, F

temp = temp->next; // 956, 1024

// temp is @ 1024 i.e. 2nd last node

free(temp->next);

temp->next = NULL;

return head; // unchanged

18 956

6 NULL (0)

To Delete a Node at given position in LL

Positions 1 2 3 4

-25 ~~1024~~

785

13 785

head = 600 956 1024 785

~~temp1~~ ~~temp2~~, temp1 temp2

Note that these positions in LL are not inherent or built in like indexes of an array. So we need to move a temp var from one node to another till the node to be deleted is not reached.

Sorting of Data from nodes of LL of integers

18 956

-25 1024

13 785

6 NULL (0)

head = 600 956 1024 785

temp1 temp2

temp1 temp2

temp1 temp2

In LL, the nodes do not have inherent (built in) like indexes (positions) of elements from an array. So to compare the node data, we will require 2 vars to store addresses of 2 nodes. However we need to compare the data of nodes again and again & this can be achieved using nested loops.