//////////////////////////////// DOUBLY LINK LIST //////////////////////////////////////////////

//DOUBLY LINEAR LINKLIST

//////////////////////////////////////////////////////////////////////

struct node

{ prev data next

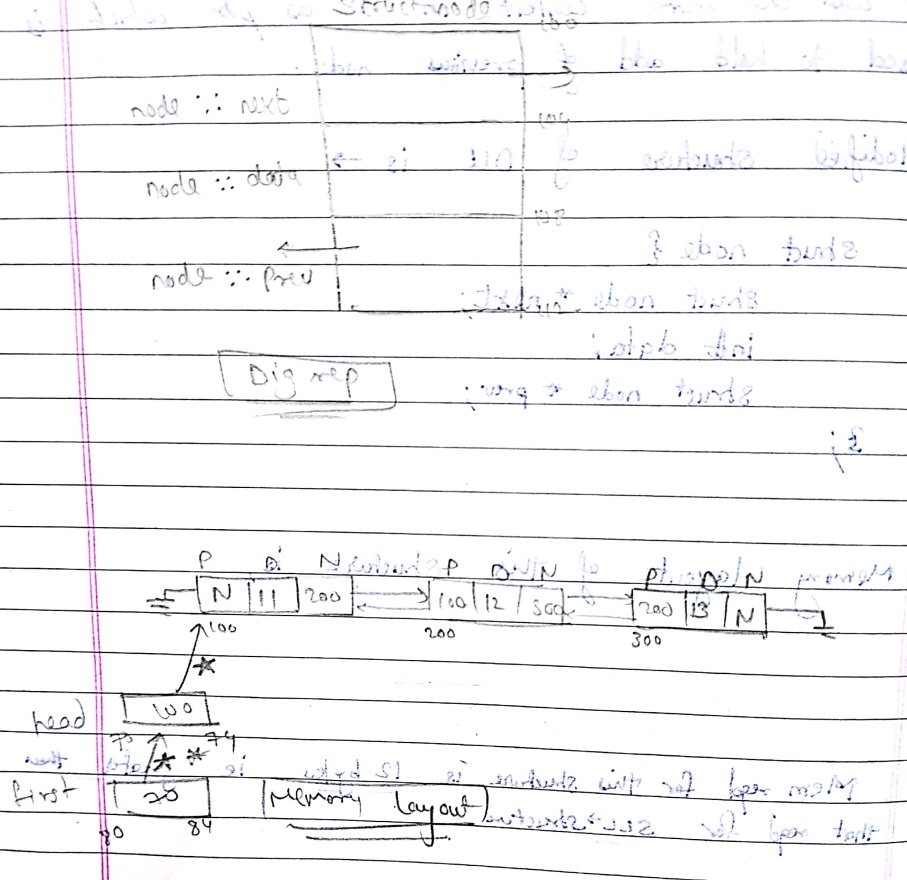
struct node \*Next,

int Data;

struct node \*Prev;

}; 100 104 108 112

////////////////////////////////////////////////////////////////////////////////////////////////////////



//Insert node at first position

int InsertFirst(struct node \*\*First, int value)

{

struct node \*Newn = NULL;

Newn = (struct node \*)malloc(sizeof(struct node ));

if (newn == NULL)

{

return FALSE;

}

Newn -> Data = value;

Newn -> Next = NULL;

Newn -> Prev = NULL;

if (\*first==NULL)

{

\*First = Newn;

}

else

{

Newn -> Next = \*First;

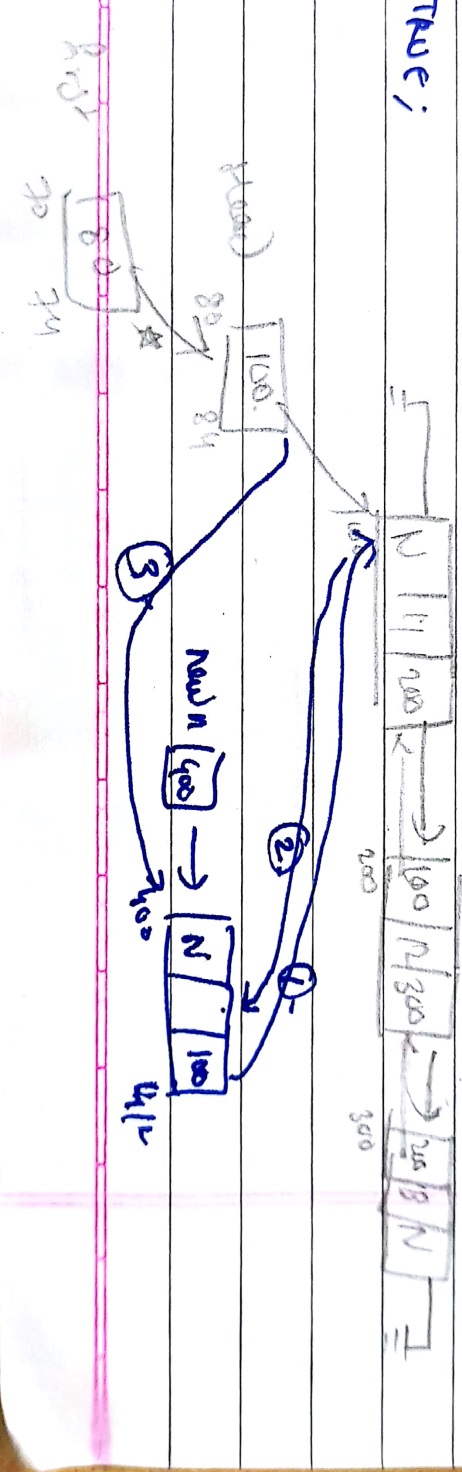
(\*First) -> Prev = Newn;

\*First = Newn;

}

return TRUE;

}



/////////////////////////////////////////////////////////////////////////////

//Display LL in fwd direction

void Display(struct node \*First)

{

while(First != NULL)

{

printf("|%d|-> ",First -> Data);

First = First -> Next;

}

}

/////////////////////////////////////////////////////////////////////////////

//Display in backward direction(reverse)

void DisplayBackward(struct node \*First )

{

while(First->Next != NULL)

{

First = First ->Next;

}

while(first != NULL)

{

printf("|%d|<=>",first -> Data);

first = first -> Prev;

}

}

///////////////////////////////////////////////

//insert node at last position

BOOL InsertLast (struct node \*\*First,int val)

{

struct node \*Newn = NULL;

struct node \*Temp = \*first;

Newn = (struct node \*)malloc(sizeof(struct node ));

if (NULL == Newn)

{

return FALSE;

}

Newn -> Data = value;

Newn -> Next = NULL;

Newn ->Prev = NULL;

if (\*First == NULL)

{

\*First = Newn;

}

else

{

while(Temp ->Next != NULL)

{

Temp = Temp -> Next;

}

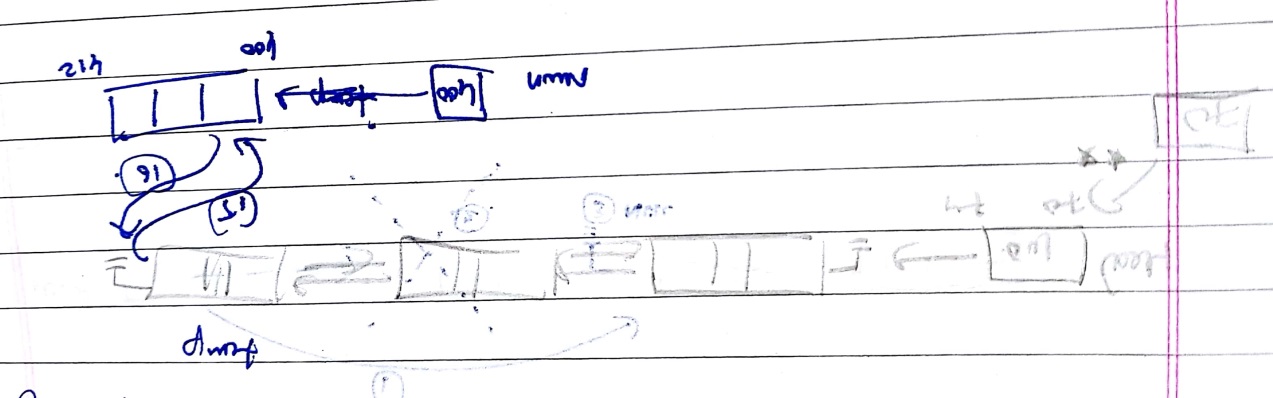
Temp -> Next = Newn;

Newn ->Prev = Temp;

}

return TRUE;

}



//DeleteFirst node

int DeleteFirst( struct node \*\*First)

{

if(\*First == NULL)

{

return FALSE;

}

Elseif((\*first->next)==NULL)

{

Free(\*first);

(\*first)=NULL

}

else

{

\*First = \*first -> Next;

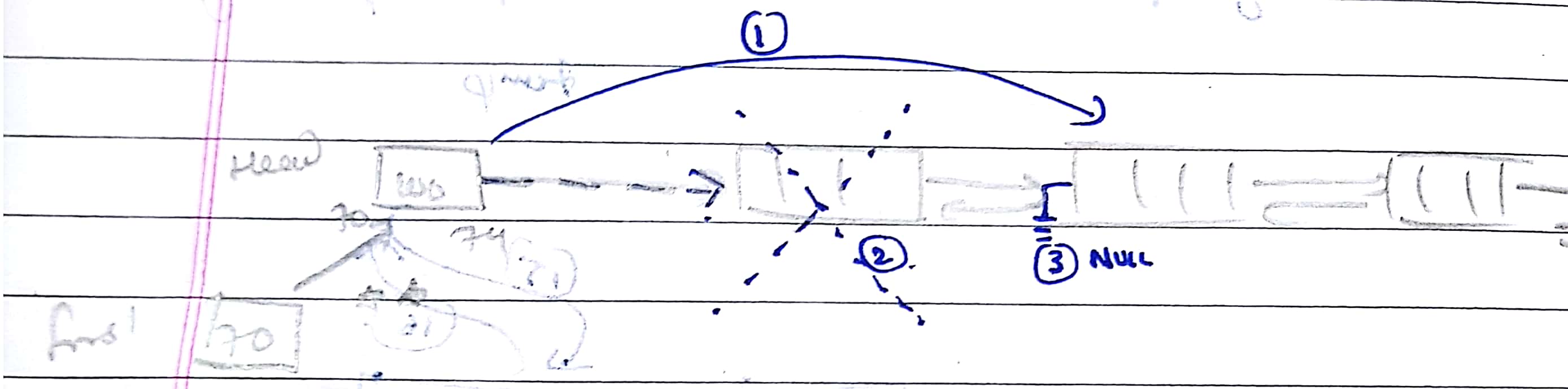
free(\*First) ->Prev;

(\*First )->prev=NULL;

}

return TRUE;

}



//Delete last node

int DeleteLast(struct node \*\*First)

{

struct node \*temp=\*First;

if(\*First == NULL)

{

return FALSE;

}

Elseif(\*first->next==NULL)

{

Free(\*first);

\*first=NULL;

}

else

{

while(temp -> Next->next != NULL)

{

temp = temp -> Next;

}

Free(temp -> Next);

temp -> Next = NULL;

}

return TRUE;

}

///////////////////////////////////////////////////////////////////////////////////////

//Delete at any specified position

int DeleteAtPosition(STruct node \*\*First,int pos)

{

struct node \*temp = \*First;

if(pos < 0)

{

Return FALSE;

}

elseif(pos > (CountNode(\*First)))

{

return FALSE;

}

else if(pos == 1)

{

DeleteFirst(First);

}

else if(pos == (CountNode(\*First)))

{

DeleteLast(First);

}

else

{

While(pos!=2)

{

temp = temp -> Next;

pos--;

}

temp -> Next= temp -> Next ->next;

free(temp -> Next->prev);

temp -> Next->prev=temp;

}

return TRUE;

}

