**Single Linked List**

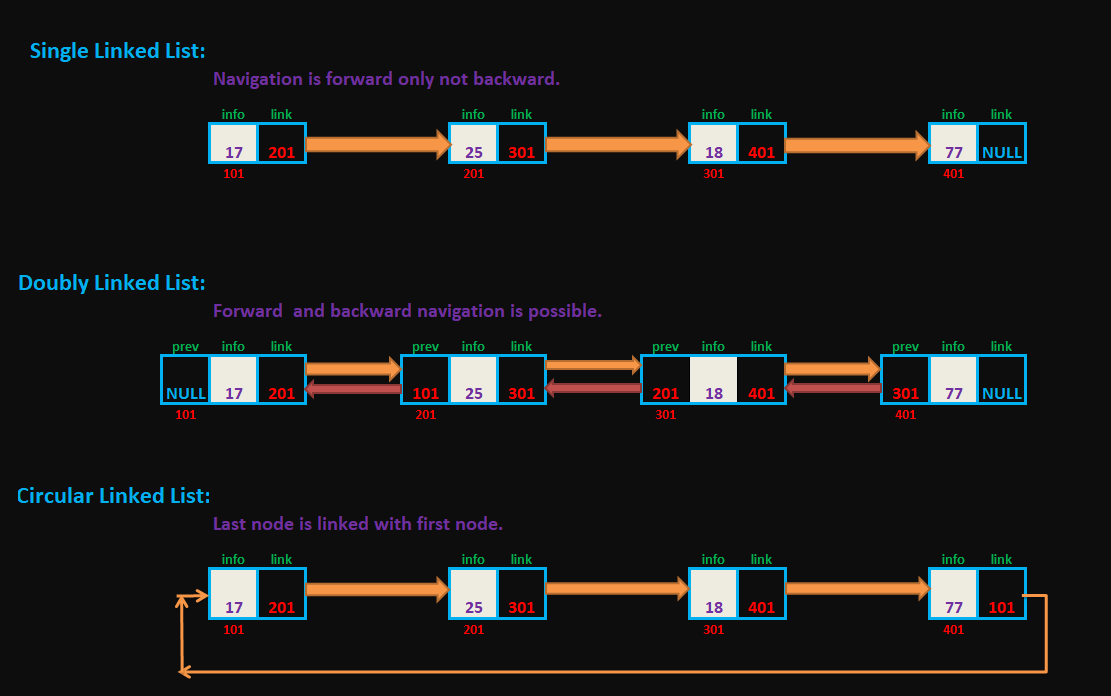
**Single Linked List:**

**A single linked list is a list made up of nodes that consists of two part-data and link.**

|  |  |
| --- | --- |
| 17 | 404 |
| data | **link**  **Contains the address of the next node of list** |

**Contains the actual data (elements)**

**In single linked list, nodes are scattered here and there in the memory but they are still connected with each other because of the link part of the node.**



**Note:**

* **Single linked list is known as *self-referential structure*.**

**Self-referential structure which contains a pointer-to-structure of same type**

struct node

{

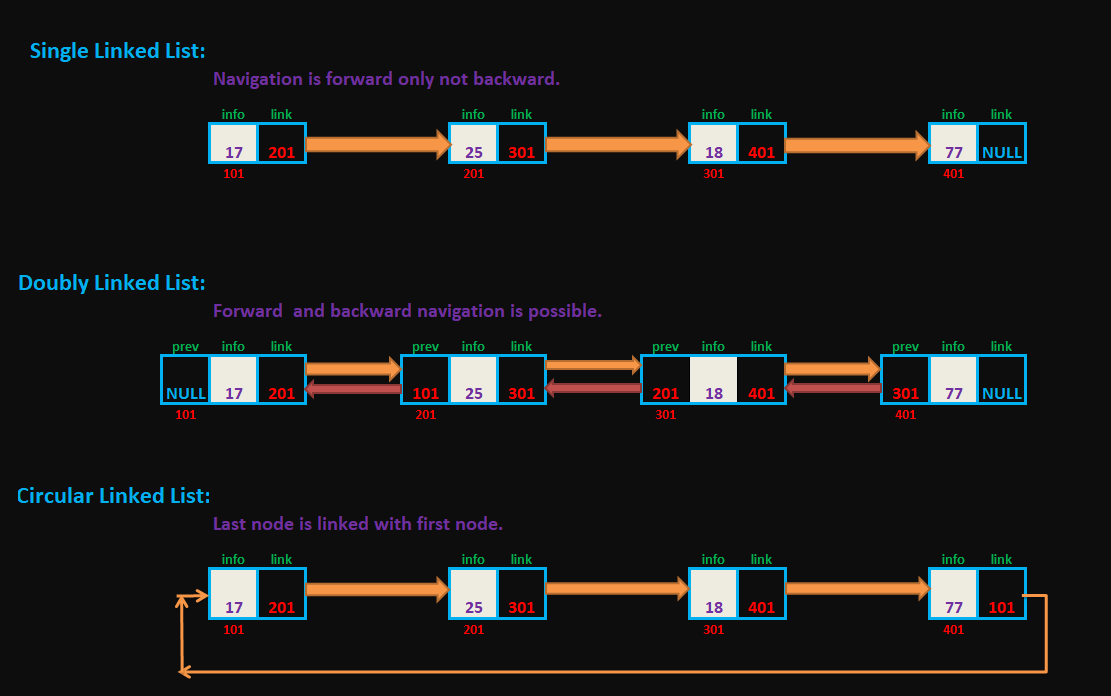
    int info;

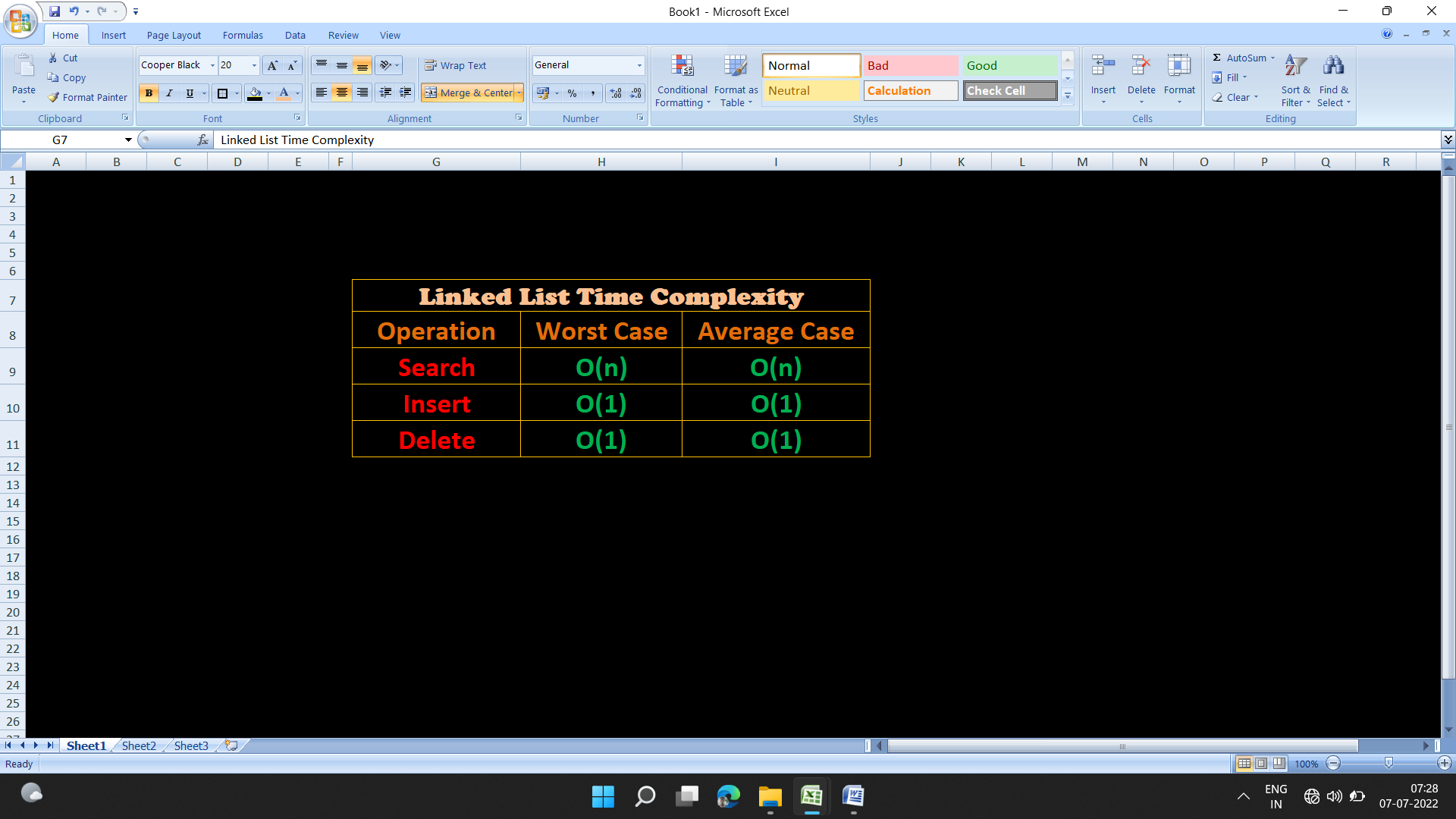
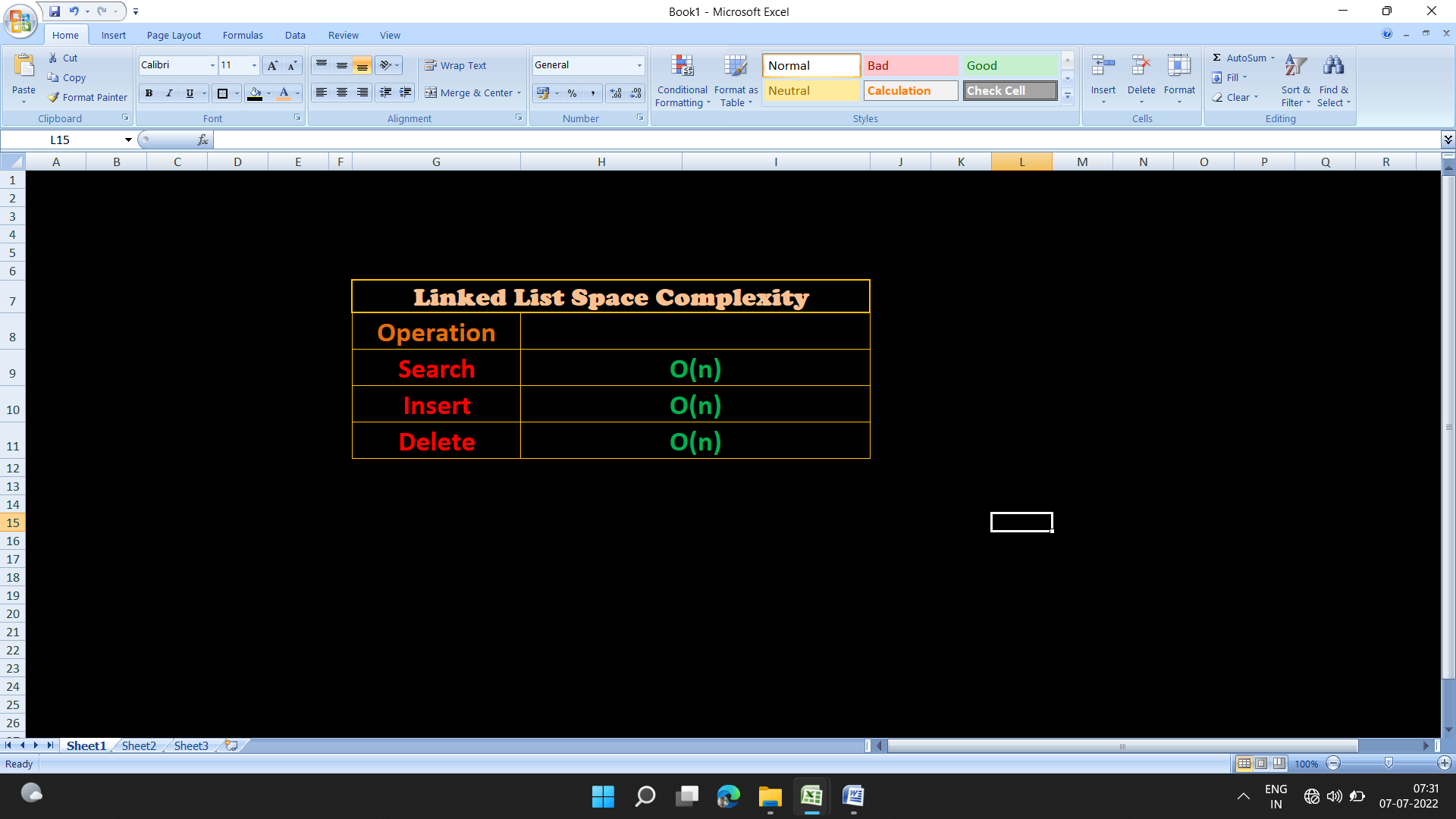
    struct node\* link;

};

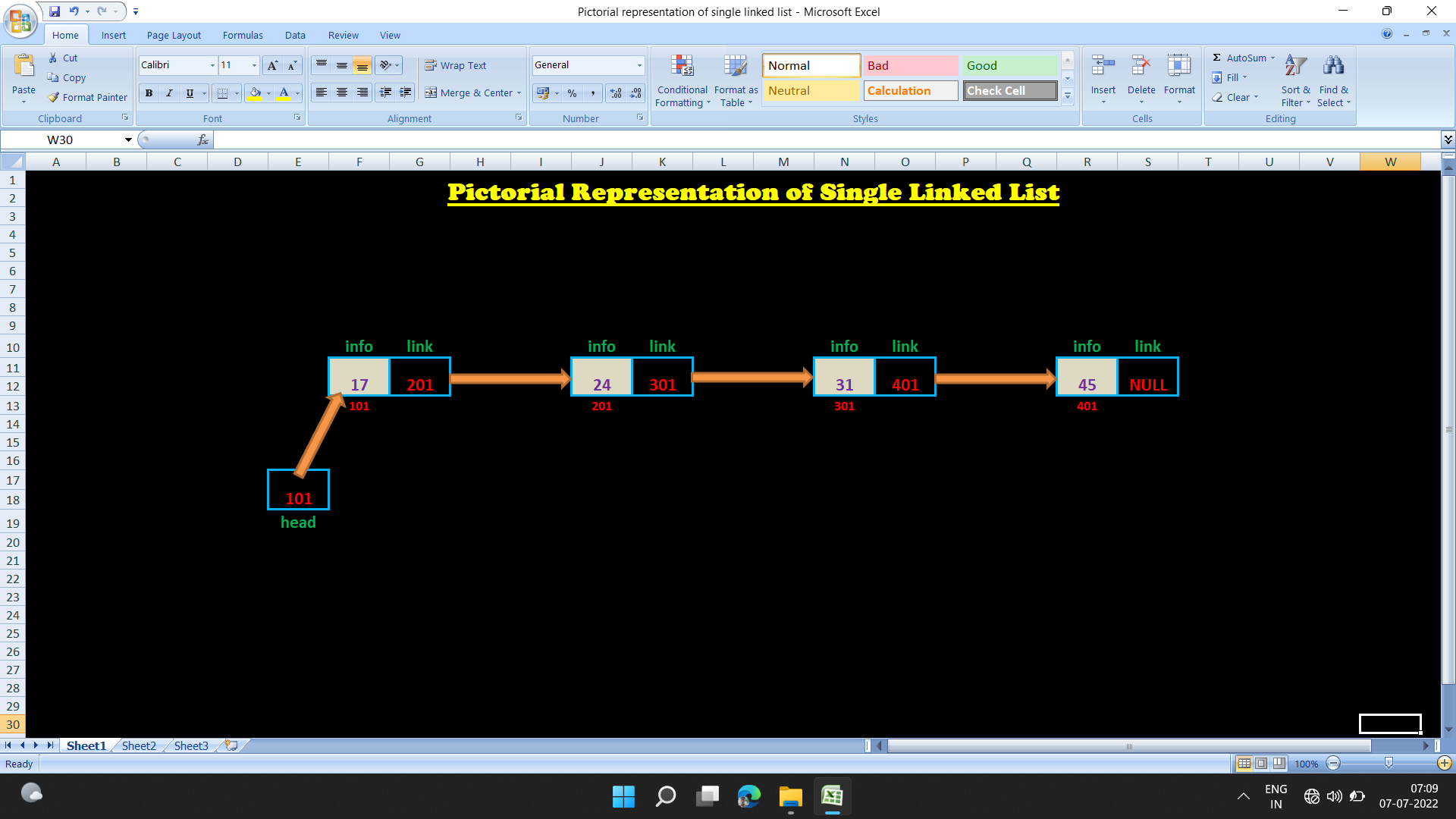
**In linked list, the link part of node is a pointer which point to other node (structure).So the node is nothing but self-referential structure.**

* **The linked list is specified using pointer.**
* **In case of last node, a null pointer is used for representing that there is no further node in the linked list.**

****

** **

**Create Single Linked List:**

****

**Write a program to create single linked list:**

#include<stdio.h>

#include<stdlib.h>

struct node\* create\_single\_linked\_list(struct node\*);

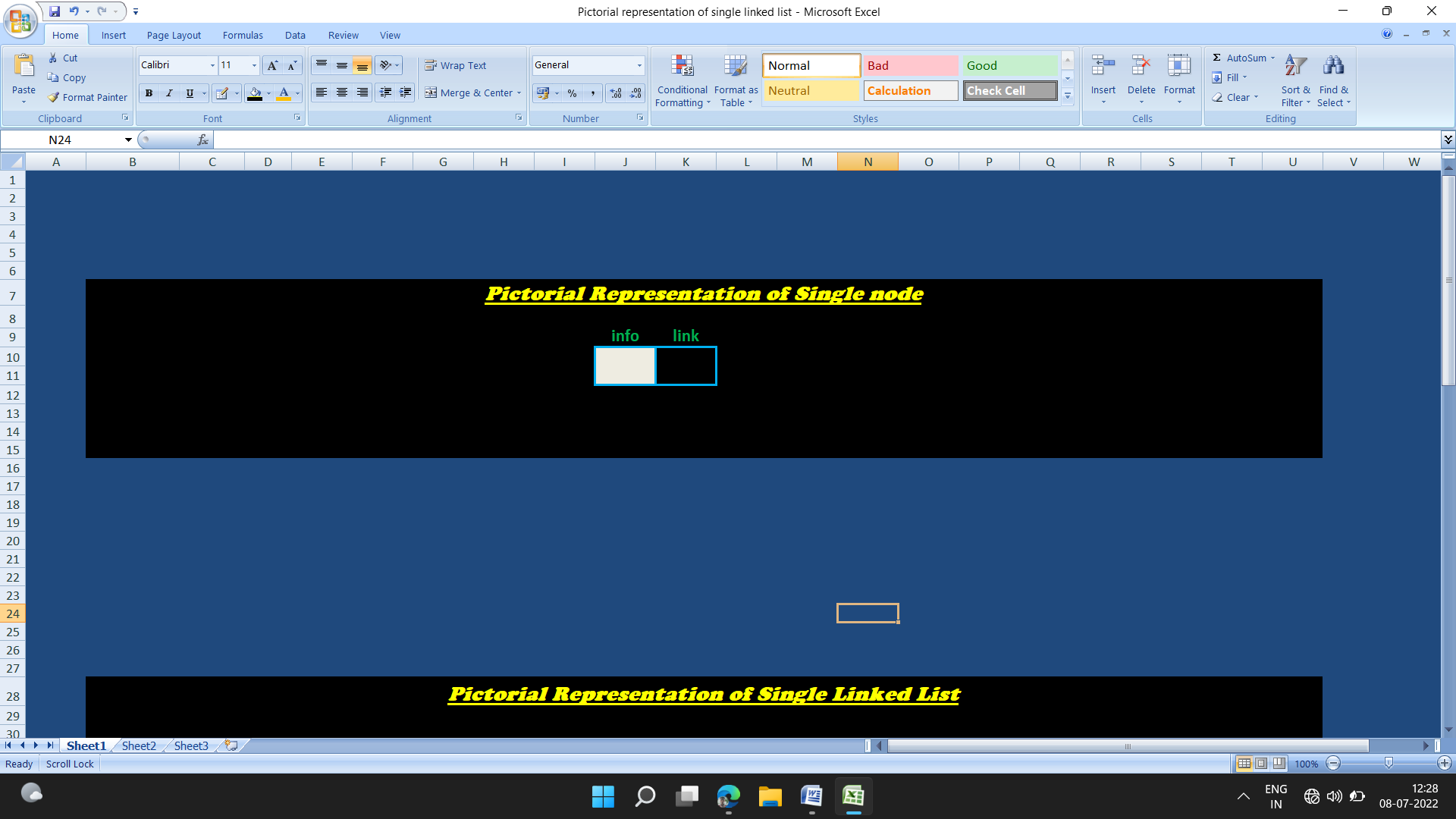
void print\_elements();

struct node  // Self-referentical structure

{

    int info;

    struct node\* link;



};

int main()

{ **101**

    struct node\* head;

    head=NULL;

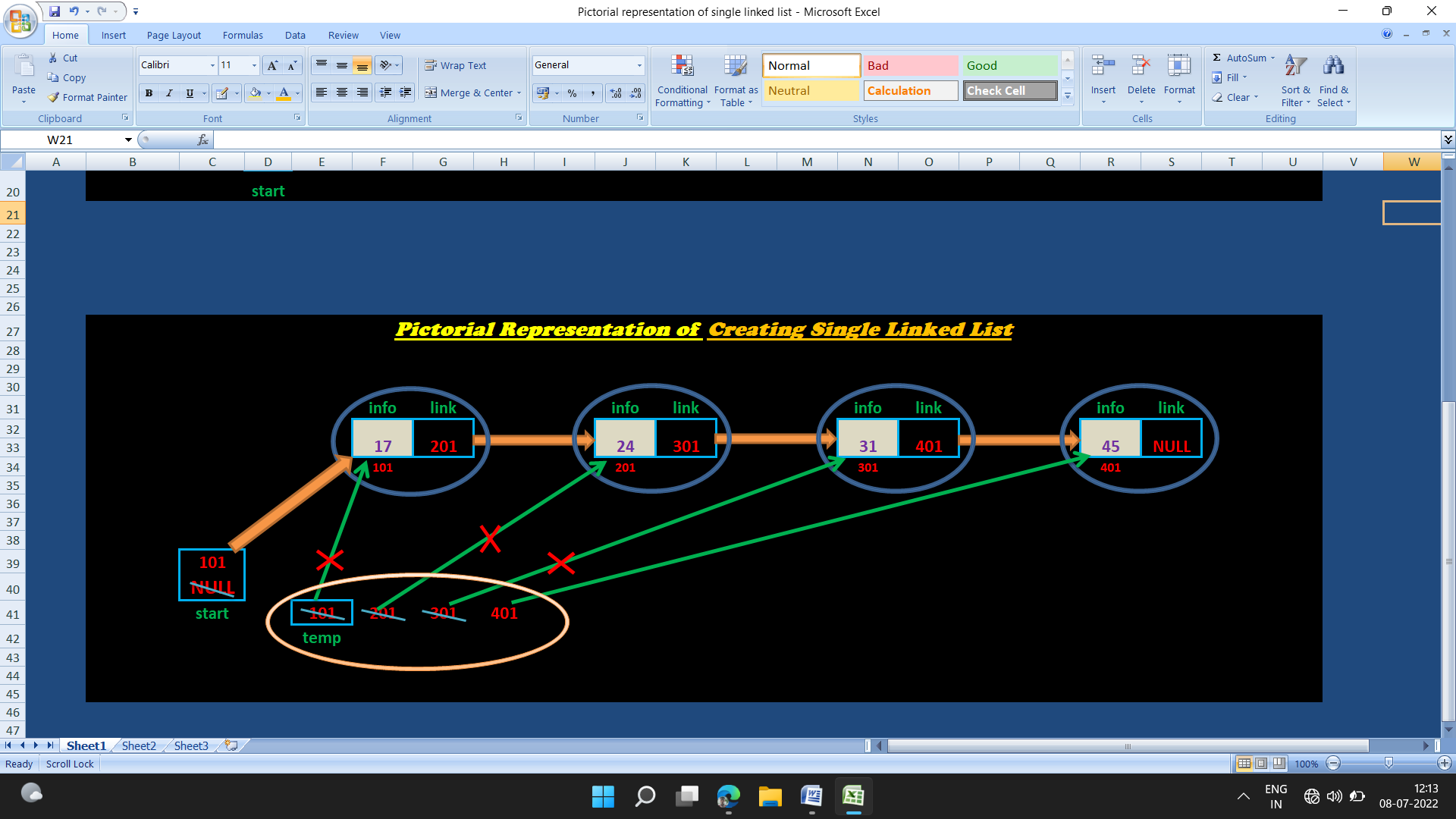
**head**

    head=create\_single\_linked\_list(head);

    print\_elements(head);

    return 0;

}

****

**NULL**

struct node\* create\_single\_linked\_list(struct node\* start)

{

    int n;

    char ans;

    struct node\* temp;

    temp=malloc(sizeof(struct node));

    start=temp;

    while (1)

    {

        printf("Enter the value to be inserted:");

        scanf("%d",&temp->info);

        fflush(stdin);

        printf("Continue(Y/N)?:");

        scanf("%c",&ans);

        if(ans=='Y' || ans=='y')

        {

          temp->link=malloc(sizeof(struct node));

          temp=temp->link;

        }

        else

        {

            temp->link=NULL;

            return start;

        }

**101**

    }

**start**

}

void print\_elements(struct node\* start)

{

    struct node\* ptr;

    ptr=start;

    printf("%d ",ptr->info);

    while (ptr!=NULL)

        print\_elements(ptr->link);

}

**OUTPUT:**

Enter a number to be inserted:21

Continue(Y/N)?:y

Enter a number to be inserted:54

Continue(Y/N)?:y

Enter a number to be inserted:67

Continue(Y/N)?:n

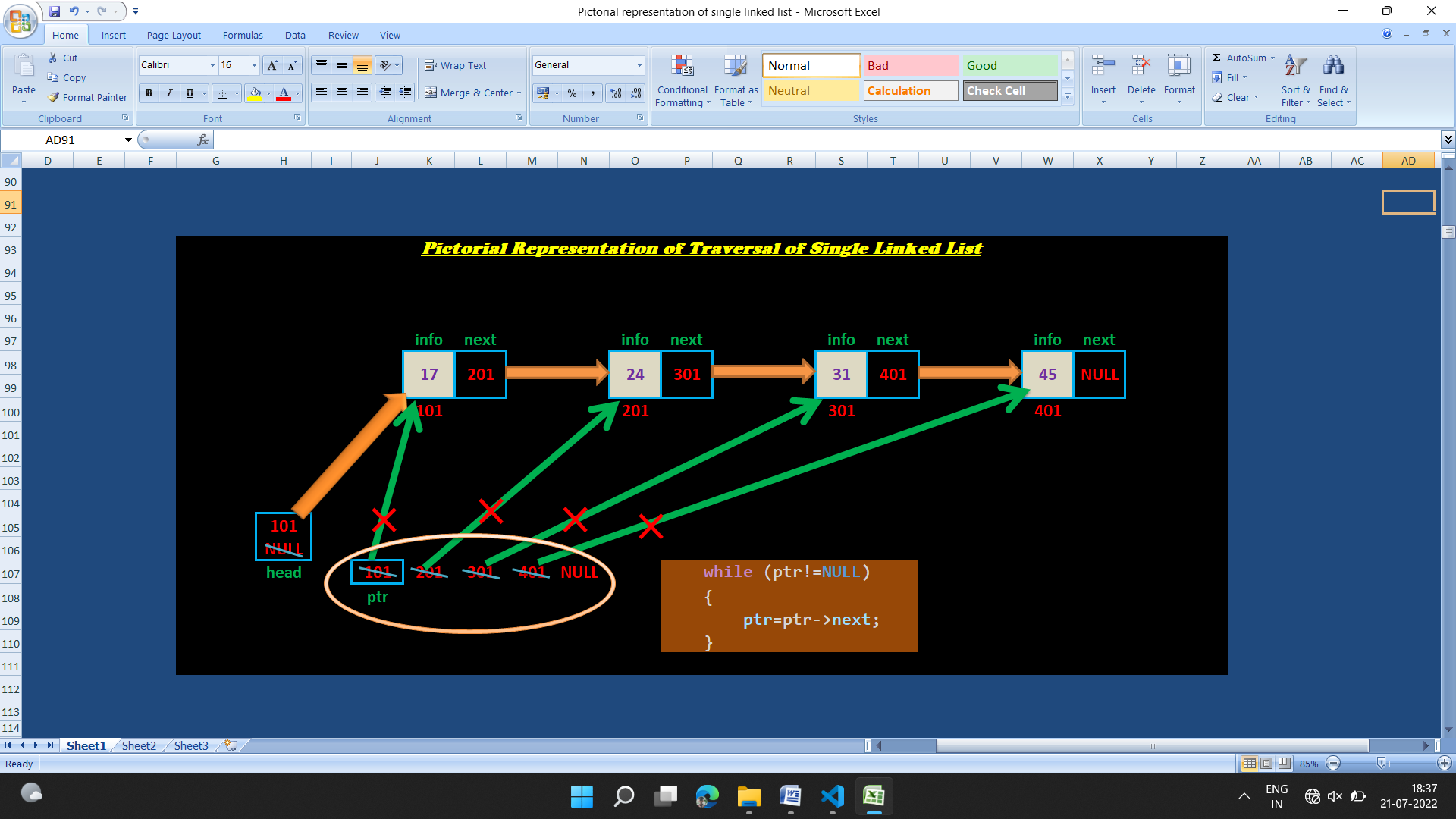
|21|13765856| |54|13765872| |67|0|

**Traversal of Single Linked List:**

**Traversing linked list means visiting each nodes of the single linked list until the end node is reached.**

**We can calculate the total number of nodes (also known as size of the linked list) in the single linked list by traversing the single linked list.**

**Total number of nodes can be called as size of the linked list.**



**Write a program to count & print the item of the nodes by traversing the single linked list:**

**#include "create\_single\_linked\_list.c"**

**#include "print\_elements\_of\_single\_linked\_list.c"**

**int traversal\_of\_single\_linked\_list(struct node\*);**

**int main()**

**{**

**struct node\* head;**

**int count;**

**head=create\_single\_linked\_list(head);**

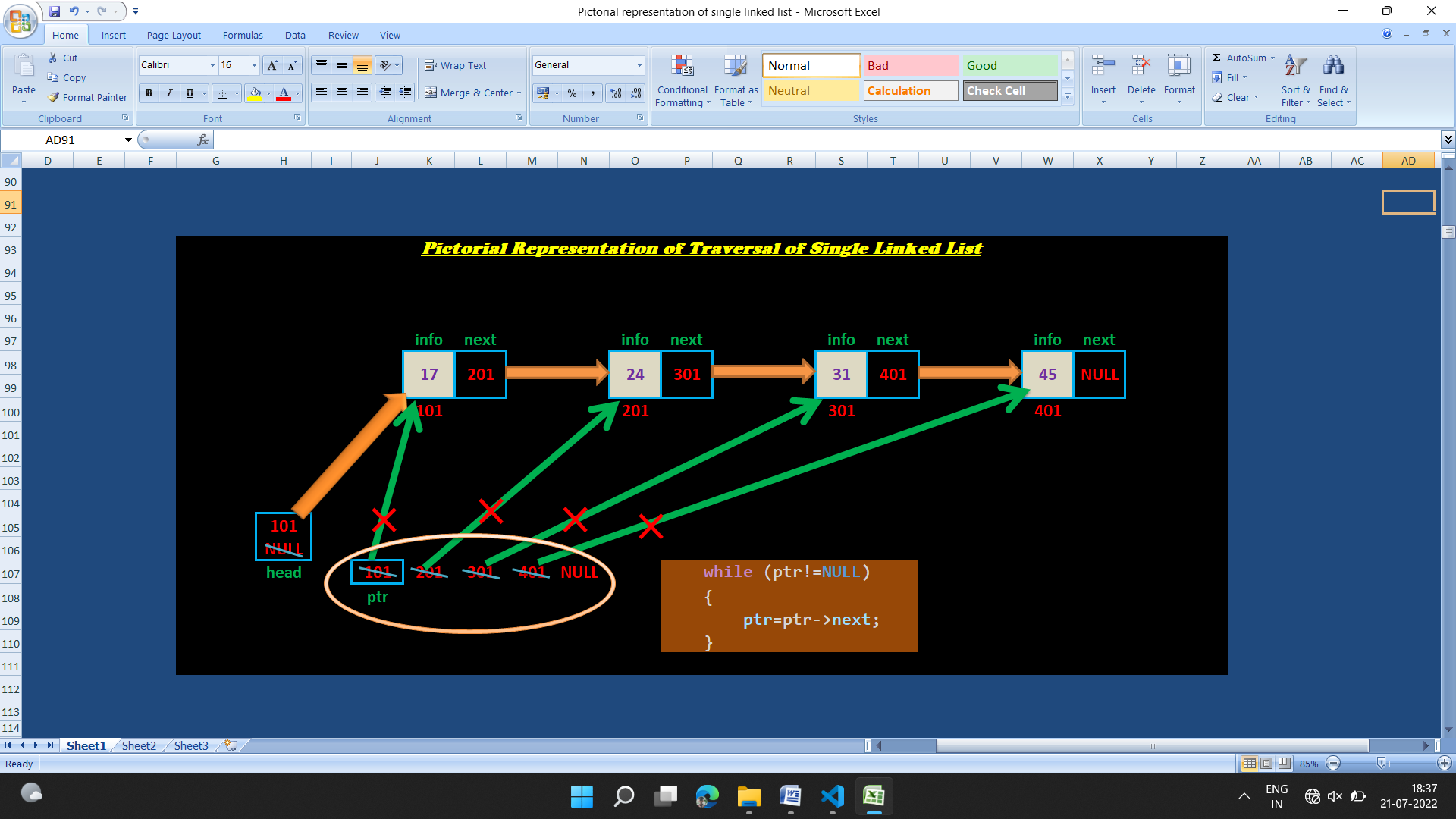
**count=traversal\_of\_single\_linked\_list(head);**

**printf("\nNo of elements of linked list are: %d\n",count);**

**printf("\nSize of linked list= %d\n", count);**

**return 0;**

**}**

****

**int traversal\_of\_single\_linked\_list (struct node\* head)**

**{**

**int count; 0 1 2 3**

**count=0;**

**Count**

**while (head!=NULL)**

**{**

**printf("%d ",head->info);**

**count++;**

**head=head->next;**

**}**

**return count;**

**}**

**OUTPUT:**

Enter a number to be inserted:12

Continue(Y/N)?:y

Enter a number to be inserted:54

Continue(Y/N)?:y

Enter a number to be inserted:76

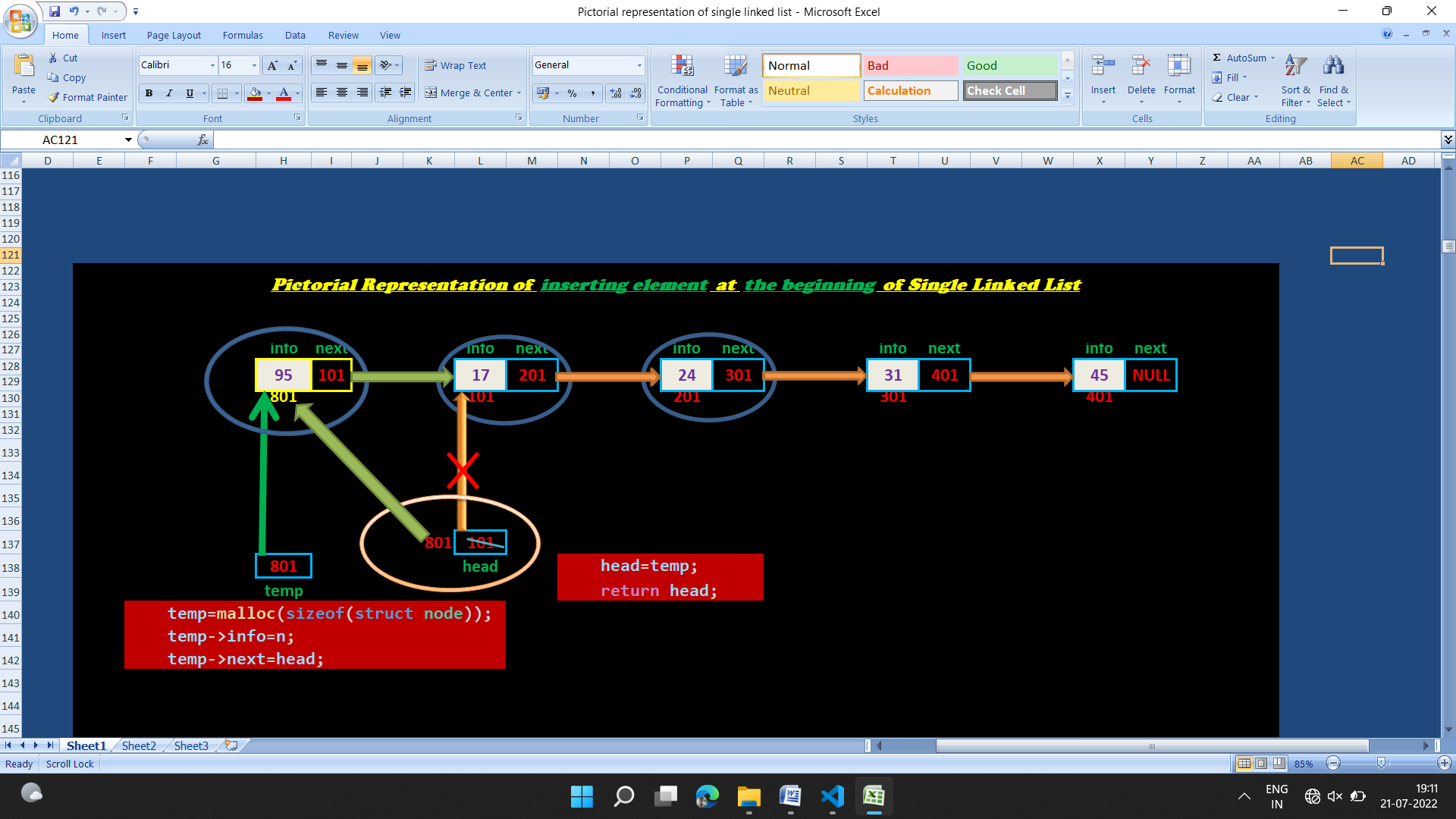
Continue(Y/N)?:n

12 54 76

No of elements of linked list are: 3

Size of linked list= 3

**Insert element at the beginning of the Single Linked List:**



**Write a program to insert element at the beginning of the single linked list:**

#include "create\_single\_linked\_list.c"

#include "print\_nodes\_of\_single\_linked\_list.c"

struct node\* insert\_element\_at\_begnning\_of\_linked\_list(struct node\*);

int main()

{

    struct node\* head;

    head=NULL;

    head=create\_single\_linked\_list(head);

    printf("Single linked list before insert at beginning:");

    print\_nodes\_of\_single\_linked\_list(head);

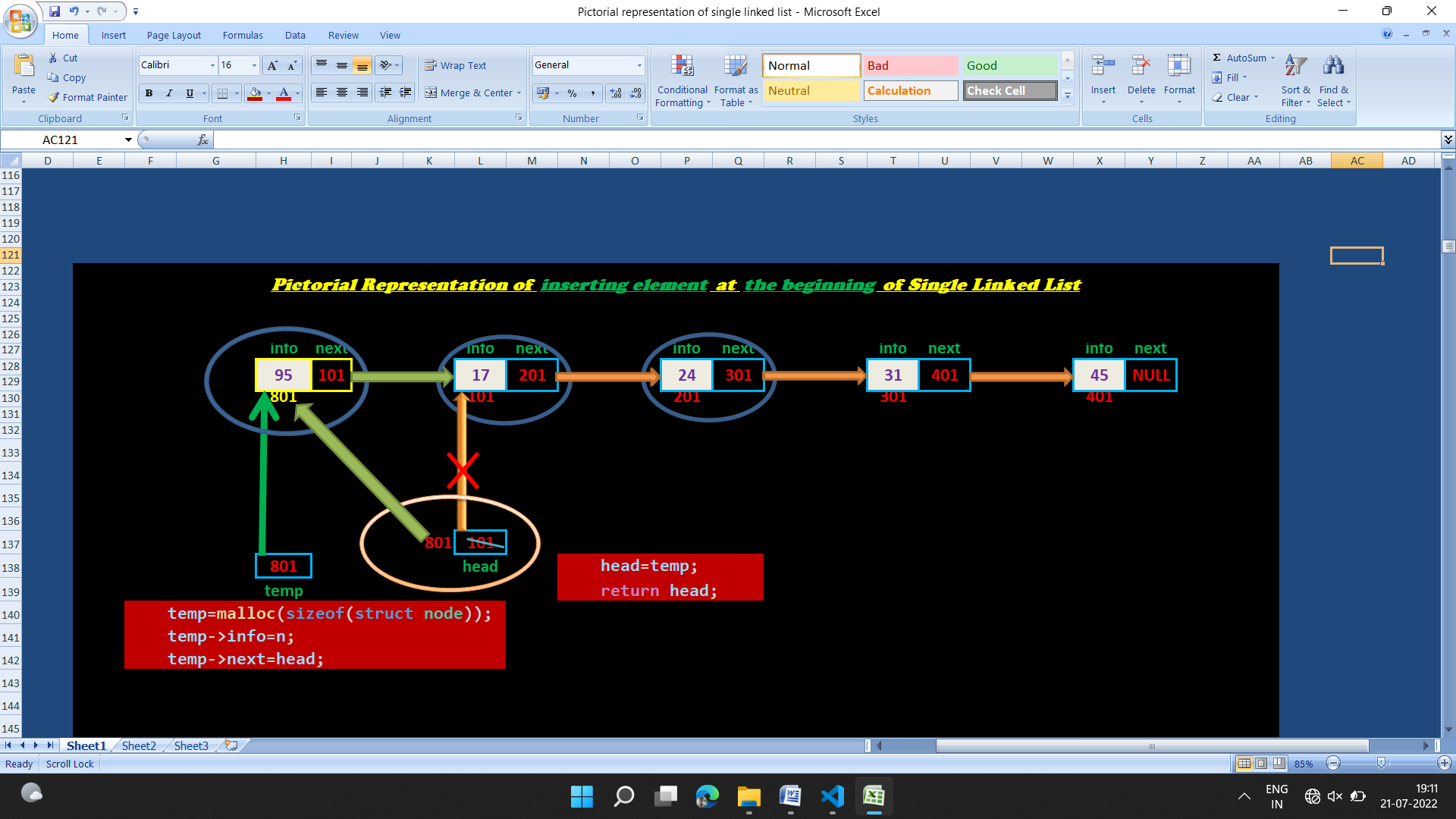
    head=insert\_element\_at\_begnning\_of\_linked\_list(head);

    printf("Single linked list after insert at beginning:");

    print\_nodes\_of\_single\_linked\_list(head);

    return 0;

}



struct node\* insert\_element\_at\_begnning\_of\_linked\_list(struct node\* head)

{

    int n;

    struct node\* temp;

    printf("Enter the number to be inserted at the beginning of the single linked list:");

    scanf("%d",&n);

    temp=malloc(sizeof(struct node));

    temp->info=n;

    temp->next=head;

    head=temp;

    return head;

}

**OUTPUT:**

Enter a number to be inserted:17

Continue(Y/N)?:y

Enter a number to be inserted:24

Continue(Y/N)?:y

Enter a number to be inserted:31

Continue(Y/N)?:y

Enter a number to be inserted:45

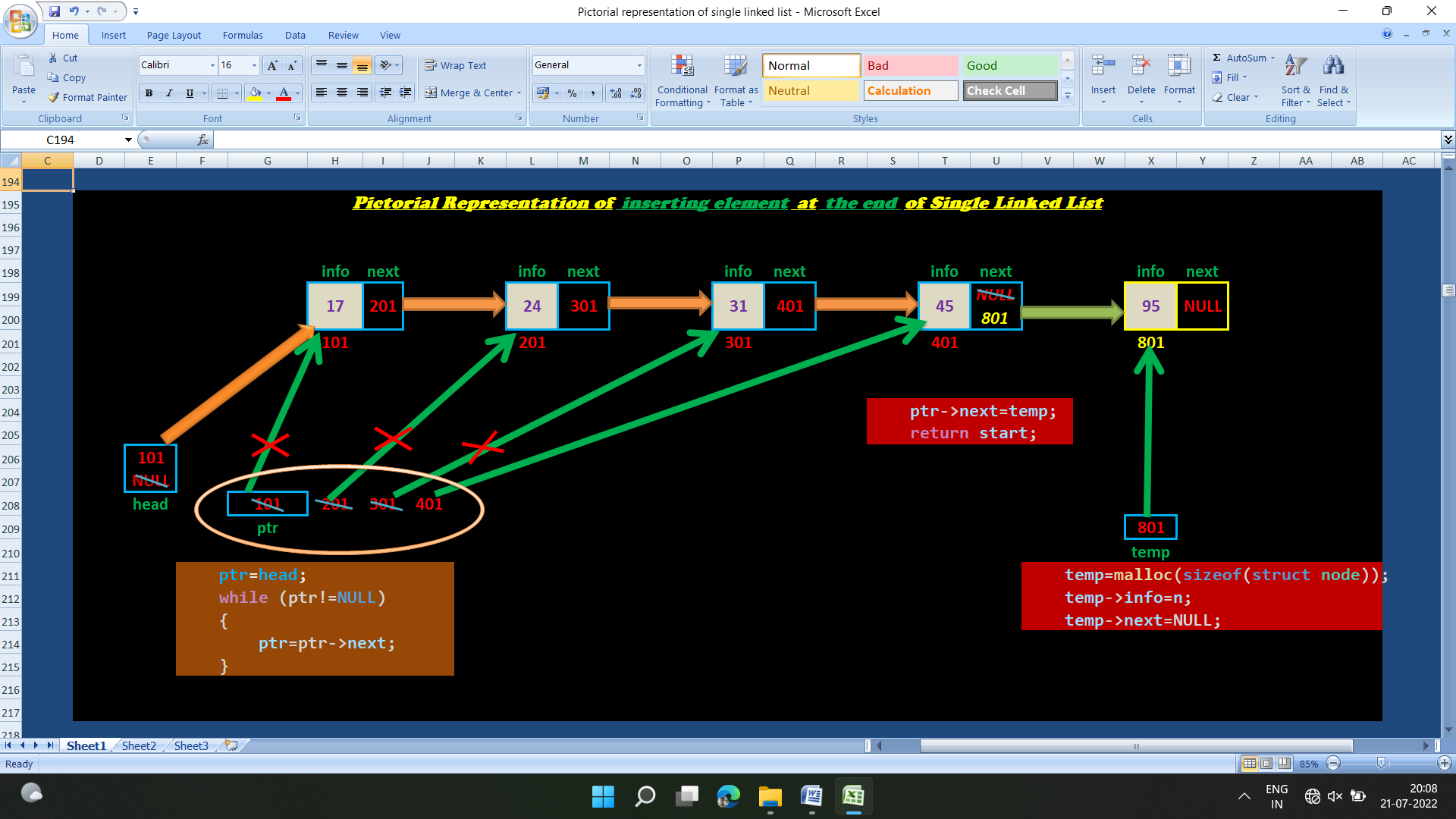
Continue(Y/N)?:n

Single linked list before insert at beginning: |17|13896928| |24|13896944| |31|13896960| |45|0|

Enter the number to be inserted at the beginning of the single linked list:95

Single linked list after insert at beginning:|95|13905872| |17|13896928| |24|13896944| |31|13896960| |45|0|

**Insert element at the end of the Single Linked List:**

****

**Write a program to insert element at the end of the single linked list:**

**#include "create\_single\_linked\_list.c"**

**#include "print\_nodes\_of\_single\_linked\_list.c"**

**struct node\* insert\_element\_at\_end\_of\_single\_linked\_list(struct node\*,int);**

**int main()**

**{**

**struct node\* head;**

**int data;**

**head=create\_single\_linked\_list(head);**

**printf("Single linked list before insert elements at the end:\n");**

**print\_nodes\_of\_single\_linked\_list(head);**

**printf("\n");**

**printf("Enter a number to be inserted at the end of the linked list:");**

**scanf("%d",&data);**

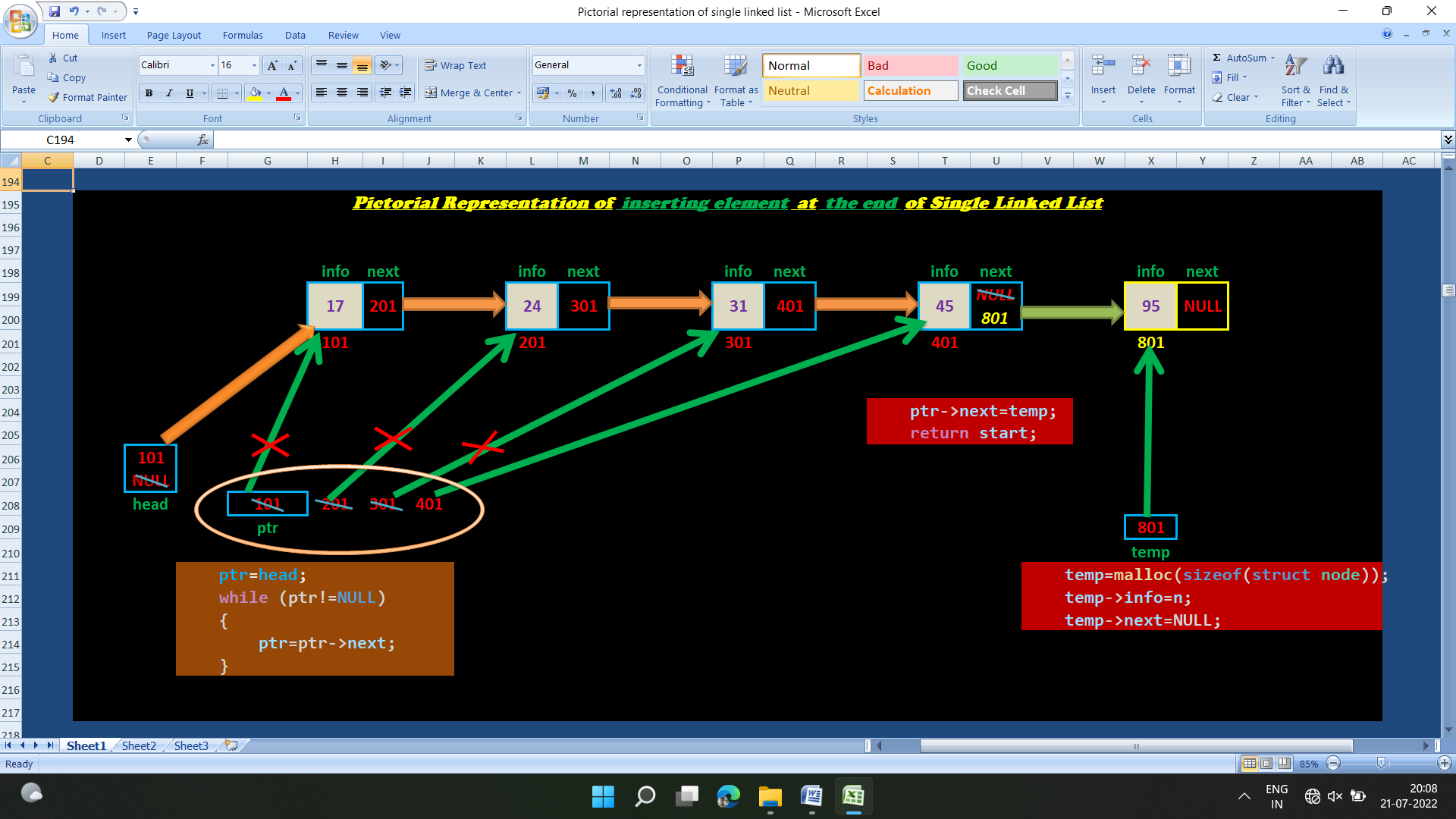
**//insert element at end of single linked list**

**head=insert\_element\_at\_end\_of\_single\_linked\_list(head,data);**

**printf("Single linked list after insert elements at the end:\n");**

**print\_nodes\_of\_single\_linked\_list(head);**

**}**

****

**struct node\* insert\_element\_at\_end\_of\_single\_linked\_list(struct node \*head, int data)**

**{**

**struct node \*temp,\*ptr;**

**temp=malloc(sizeof(struct node));**

**temp->info=data;**

**temp->next=NULL;**

**ptr=head;**

**while (ptr->next!=NULL)**

**ptr=ptr->next;**

**ptr->next=temp;**

**return head;**

**}**

**OUTPUT:**

Enter a number to be inserted: 17

Continue(Y/N)?:y

Enter a number to be inserted: 24

Continue(Y/N)?:y

Enter a number to be inserted: 31

Continue(Y/N)?:y

Enter a number to be inserted: 45

Continue(Y/N)?:n

Single linked list before insert elements at the end:

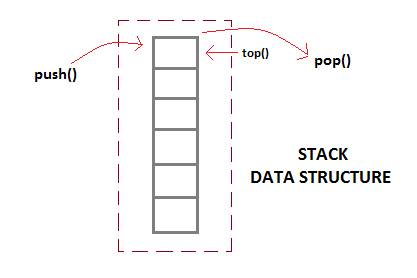
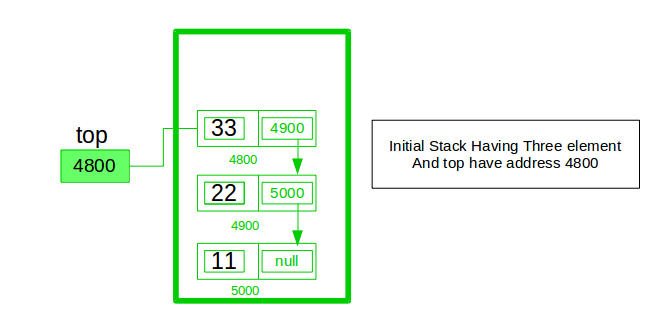
|17|12913888| |24|12913904| |31|12913920| |45|0|

Enter a number to be inserted at the end of the linked list:95

Single linked list after insert elements at the end:

|17|12913888| |24|12913904| |31|12913920| |45|12913936| |95|0|

**Implement Stack using Single Linked List:**

** **

**Instead of using array, we can also use linked list to implement stack. Linked list allocates the memory dynamically.**

**Benefit of implementing stack using linked list in C over arrays?**

**In using array will put a restriction to the maximum capacity of the array which can lead to stack overflow.**

**Benefit of implementing stack using linked list in C over arrays is that, it allows to grow the stack as per the requirements,i.e, memory can be allocated dynamically. So we can use linked list to represent stack when the size of the stack not known in advance.**

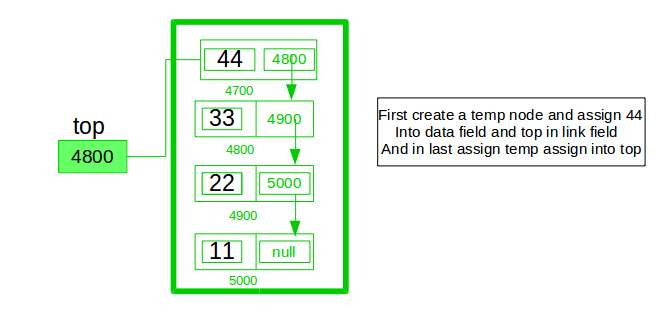
**In linked list implementation of stack, the nodes are maintained non-contiguously in the memory. Stack is said to be overflown if the space left in the memory heap is not enough to create a node. In case of implementing stack using linked list, stack overflow is very rear.**

**Linked list gives us the advantage to increase the size of stack as much as required.**

**To implement a stack using singly linked list concept, all the singly linked list operations(like pup,pop,) are performed based on Stack operations LIFO(last in first out) and with the help of that knowledge we are going to implement a stack using single linked list.**

**In stack Implementation, a stack contains a top pointer, which is top of the stack where pushing and popping items happens at the head of the list. First node have null in link field and second node link have first node address in link field and so on and last node address in “top” pointer.**

**Adding a node to the stack (Push operation):**

****

**Adding a node to the top of stack is referred to as** **push** **operation.** **For push operation, a node will be inserted at the beginning of the linked list. So the code of push() function is similar to the code of inserting node at the beginning of the singly linked list.**

**In order to push an element onto the stack, the following steps are involved.**

1. **Create a node first and allocate memory to it.**

temp=malloc(sizeof(STACK))

1. **If the stack/list is empty then the item is to be pushed as the start node of the list. This includes assigning value to the data part of the node and assign null to the address part of the node.**

temp->info=n;

temp->next=NULL;

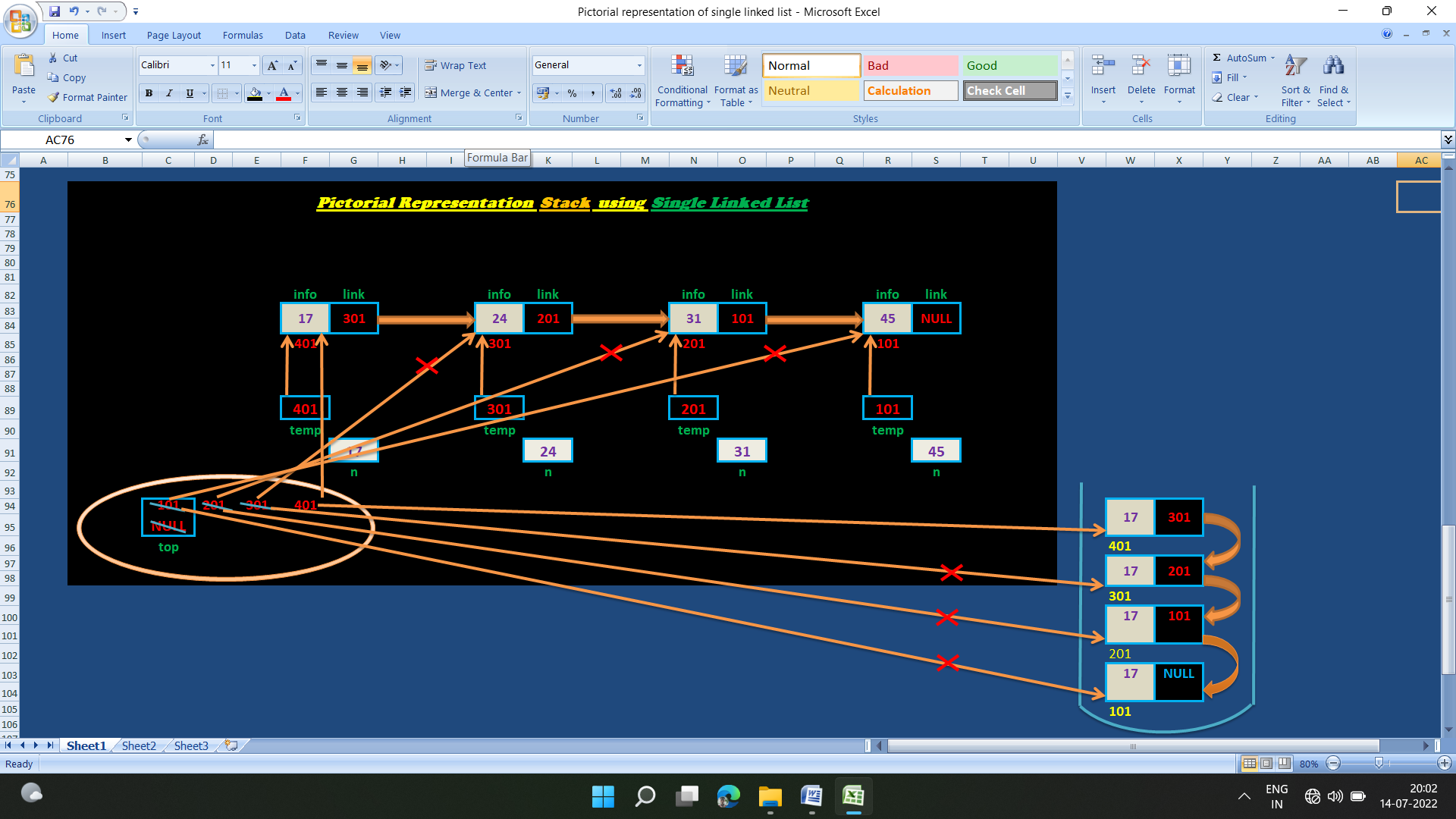
1. **If there are some nodes in the stack/list already, then we have to add the new element in the beginning of the list (to not violate the property of the stack). For this purpose, assign the address of the starting node to the address field of the new node and make the new node, the starting node of the list.**

temp->info=n;

temp->next=\*top;

1. **top pointer always points to the first node of the linked list /top element of the stack.**

\*top=temp;

****

**Stack**

**Code of push() function:**

void push(STACK\*\* top,int n)

{

    STACK\* temp;

    temp=malloc(sizeof(STACK));

    if (temp==NULL) //Stack overflow occurs

    {

        printf("Stack Overflow!\n");

        printf("We need to pop elements from linked list:\n");

        return;

    }

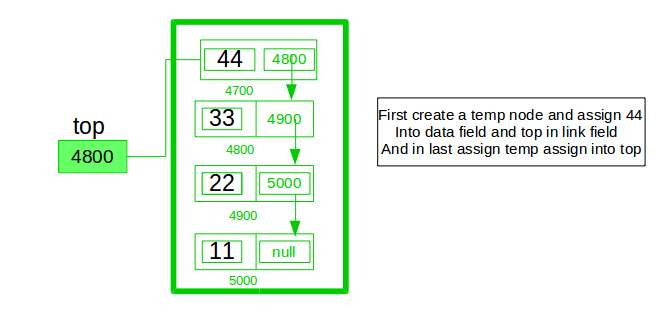
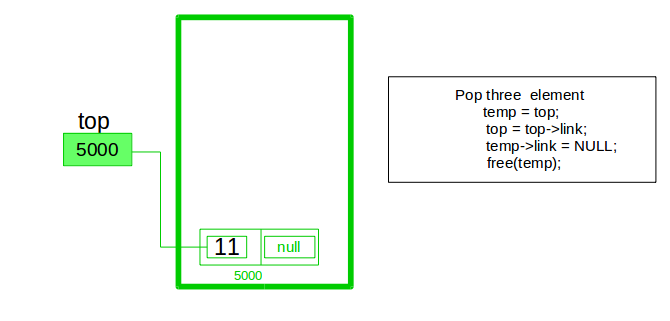
    temp->info=n;

    temp->next=\*ps;

    \*top=temp;

}

**Deleting a node from the stack (POP operation):**

** **

**Deleting a node from the top of stack is referred to as** **pop operation. For pop operation, a node will be deleted from the beginning of the linked list. So the code of pop () function is similar to the code of deleting node from the beginning of the singly linked list.**

**\*Deleting a node from the linked list implementation of stack is different from that in the array implementation.**

**In order to pop an element from the stack, we need to follow the following steps:**

* 1. **Check for the underflow condition: The underflow condition occurs when we try to pop from an already empty stack. The stack will be empty if the top pointer of the list points to null.**

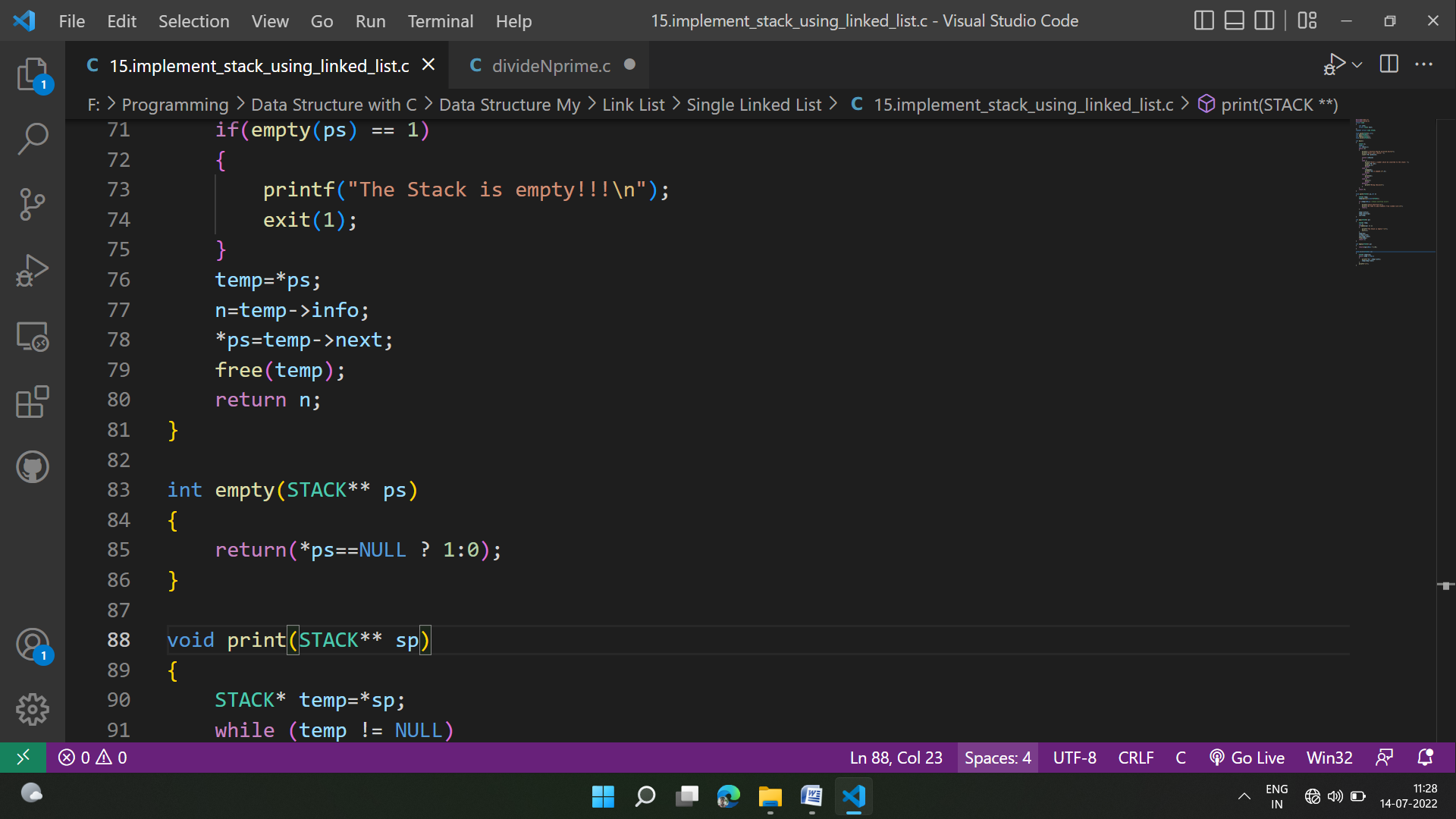
  if(empty(top) == 1)

    {

        printf("The Stack is empty!!!\n");

        exit(1);

    }



* 1. **Create a temporary pointer ‘temp’ for the purpose of deletion.**

STACK\* temp;

* 1. **Update the temporary pointer ‘temp’ so that it can point to the first node of the linked list.**

temp=\*top;

* 1. **Store the value (info) of the first node in integer variable n.**

**int** n=temp->info;

* 1. **Update the top pointer so that it can point to the next node of the linked list.**

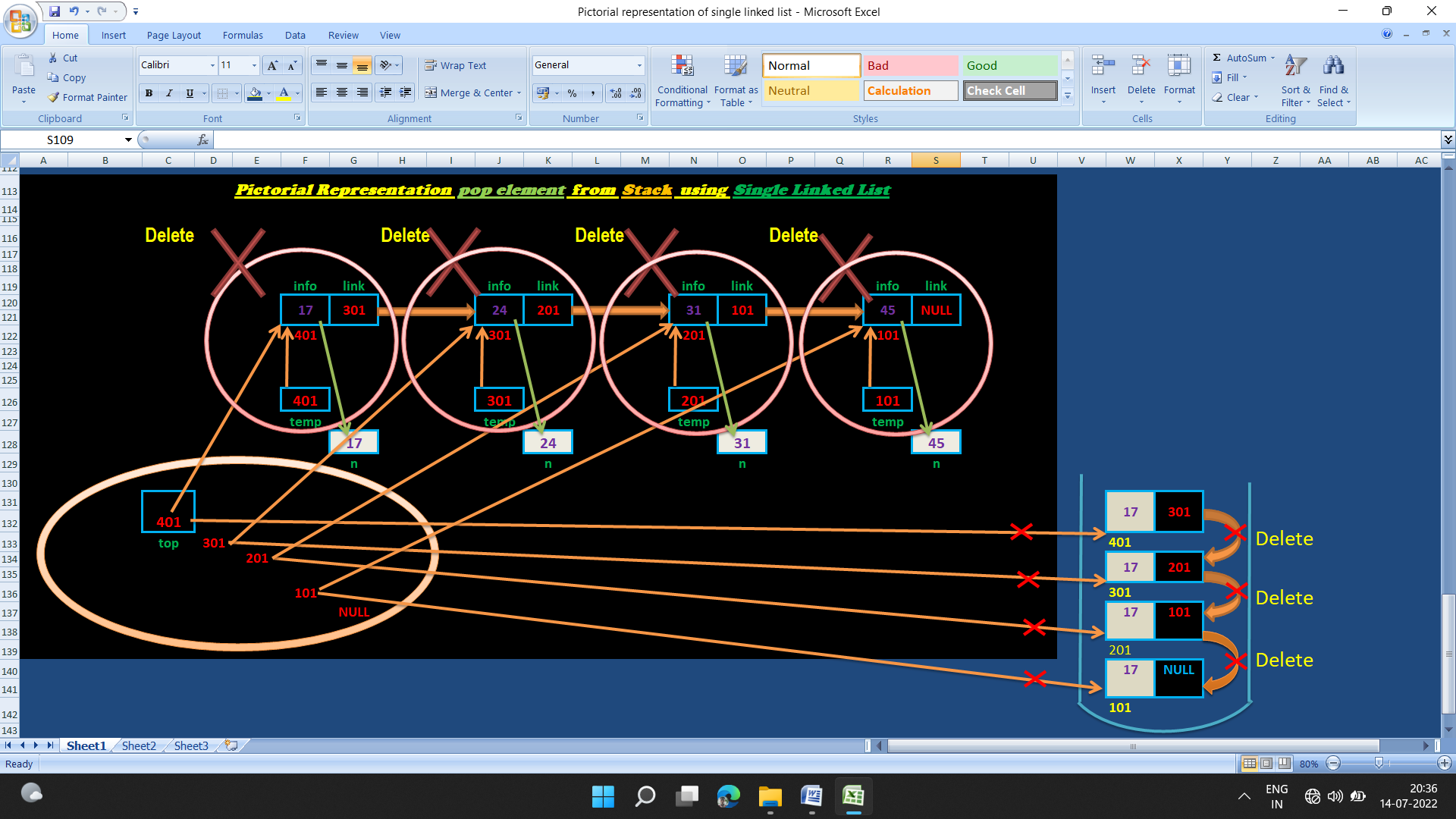
\*top=temp->next;

* 1. **Delete the node pointed by the temporary pointer ‘temp’.**

free(temp);

* 1. **Return the value of the first node which is now stored in variable n.**

return n;

****

**Code of pop() function:**

int pop(STACK\*\* top)

{

    STACK\* temp;

    int n;

    if(empty(top) == 1)

    {

        printf("The Stack is empty!!!\n");

        exit(1);

    }

    temp=\*top;

    n=temp->info;

    \*top=temp->next;

    free(temp);

    return n;

}

int empty(STACK\*\* top)

{

    return(\*top==NULL ? 1:0);

}

**Stack Operations:**

1. [**push()**](https://www.geeksforgeeks.org/stack-push-and-pop-in-c-stl/)**:** Insert a new element into stack i.e just inserting a new element at the beginning of the linked list.
2. [**pop()**](https://www.geeksforgeeks.org/stack-push-and-pop-in-c-stl/)**:** Return top element of the Stack i.e. simply deleting the first element from the linked list.
3. [**peek()**](https://www.geeksforgeeks.org/stack-peek-method-in-java/)**:** Return the top element.
4. **display():** Print all elements in Stack.

**The time complexity for all push(), pop(), and peek() operations is O(1) as we are not performing any kind of traversal over the list. We perform all the operations through the current pointer only.**

**Space complexity: O(n) where n is size of the stack**