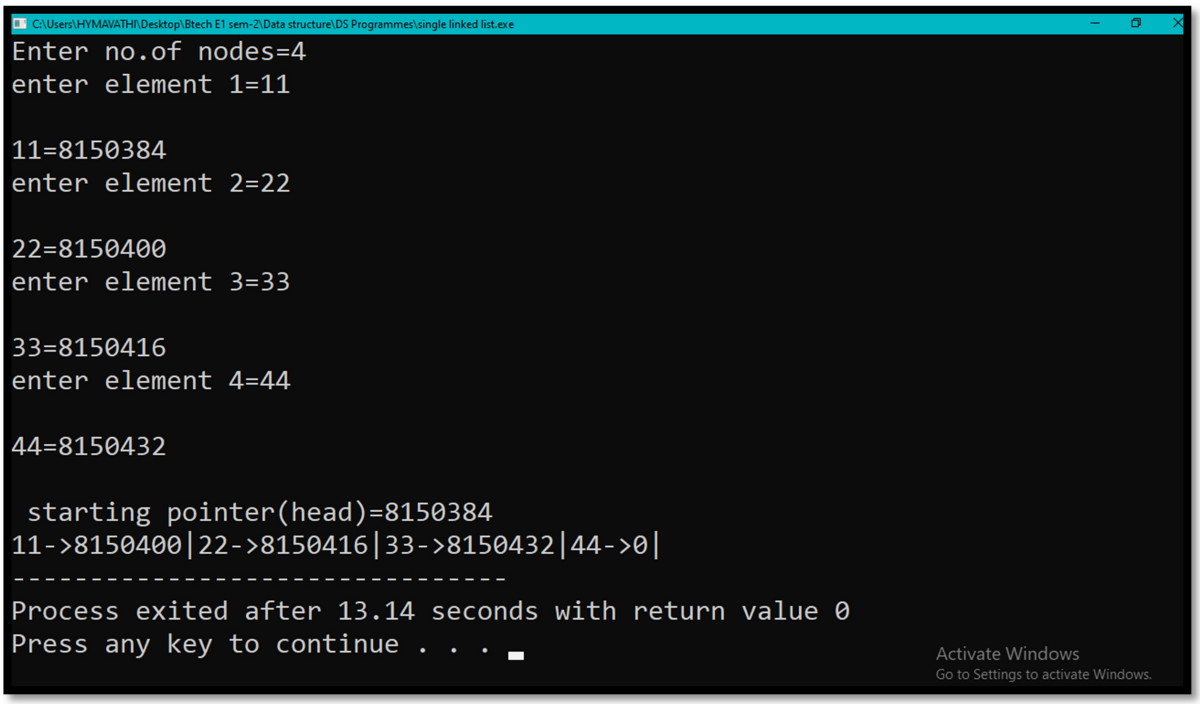
Operations in a Single linked list

Single linked list

```
#include<stdio.h>
#include<stdlib.h>
struct node{
        int data;
        struct node *link;
};
int main()
{
        int len,i;
        printf("Enter no.of nodes=");
        scanf("%d",&len);
        struct node *head,*pn,*temp;
        for(i=0;i<len;i++)
        {
                pn=(struct node*)malloc(sizeof(struct node));
                printf("enter element %d=",(i+1));
                scanf("%d",&pn->data);
                printf("\n%d=%u\n",pn->data,&pn->data);

                if(i==0)
                {
```

```
                head=temp=pn;
        }

        else
        {
                temp->link=pn;
                temp=pn;
        }

    }
    temp->link=NULL;
    temp=head;
    printf("\n starting pointer(head)=%d \n",head);
    while(temp!=NULL)
    {
        printf("%d->",temp->data);
        printf("%u|",temp->link);
       temp=temp->link;
    }
}
```

Output



Insertion{Single linked list}

Case-1 : Inserting a node at the starting of a single linked list

Algorithm

- **Step 1 -** Create a **newNode**(ptr) with some memory using dynamic memory allocation.
- **Step 2 -** Enter **ptr->data** by the user.
- **Step 3 -** Set **ptr->link=head**
- **Step 4 -** Then set **head=ptr**

Program

```
#include<stdio.h>

#include<stdlib.h>
```

```
struct node{
        int data;
        struct node *link;
};
int main()
{
        int len,i;
        printf("Enter no.of nodes=");
        scanf("%d",&len);
        struct node *head,*pn,*temp;
        for(i=0;i<len;i++)
        {
                pn=(struct node*)malloc(sizeof(struct node));
                printf("enter element %d=",(i+1));
                scanf("%d",&pn->data);
                printf("\n%d=%u\n",pn->data,&pn->data);

                if(i==0)
                {
                        head=temp=pn;
                }
```

```
        else
        {
                temp->link=pn;
                temp=pn;
        }

}
temp->link=NULL;
temp=head;
while(temp!=NULL)
{
        printf("%d->",temp->data);
        printf("%u|",temp->link);
        temp=temp->link;
}
```

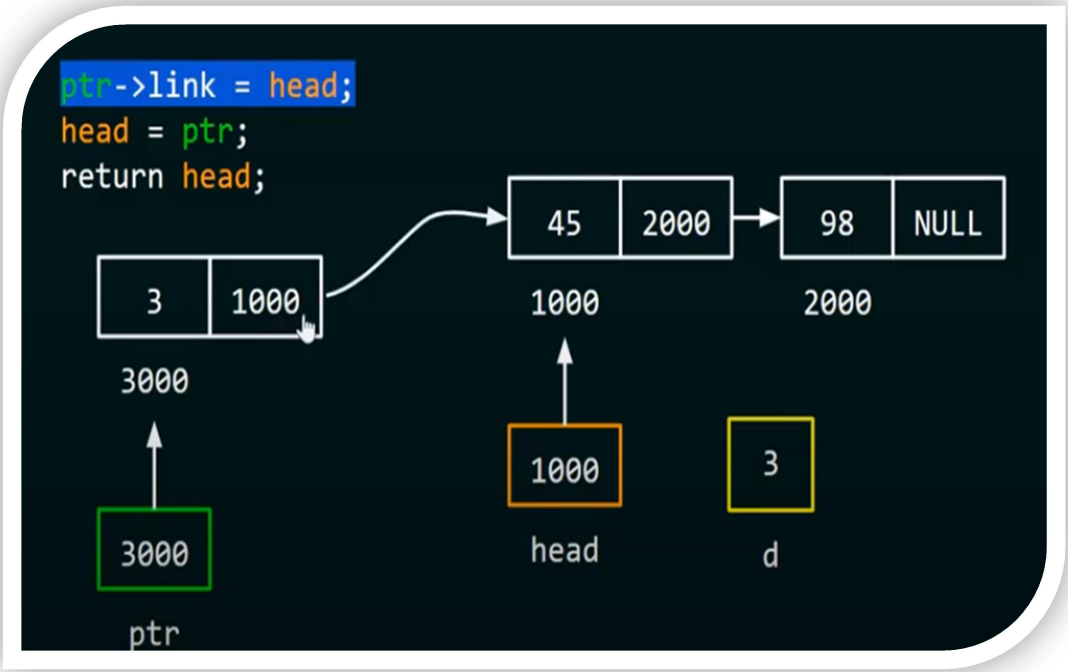
```
struct node *ptr;
ptr=(struct node*)malloc(sizeof(struct node));
printf("\nenter a element=");
scanf("%d",&ptr->data);
printf("\n%d=%u\n",ptr->data,&ptr->data);
ptr->link=head;
head=ptr;
```

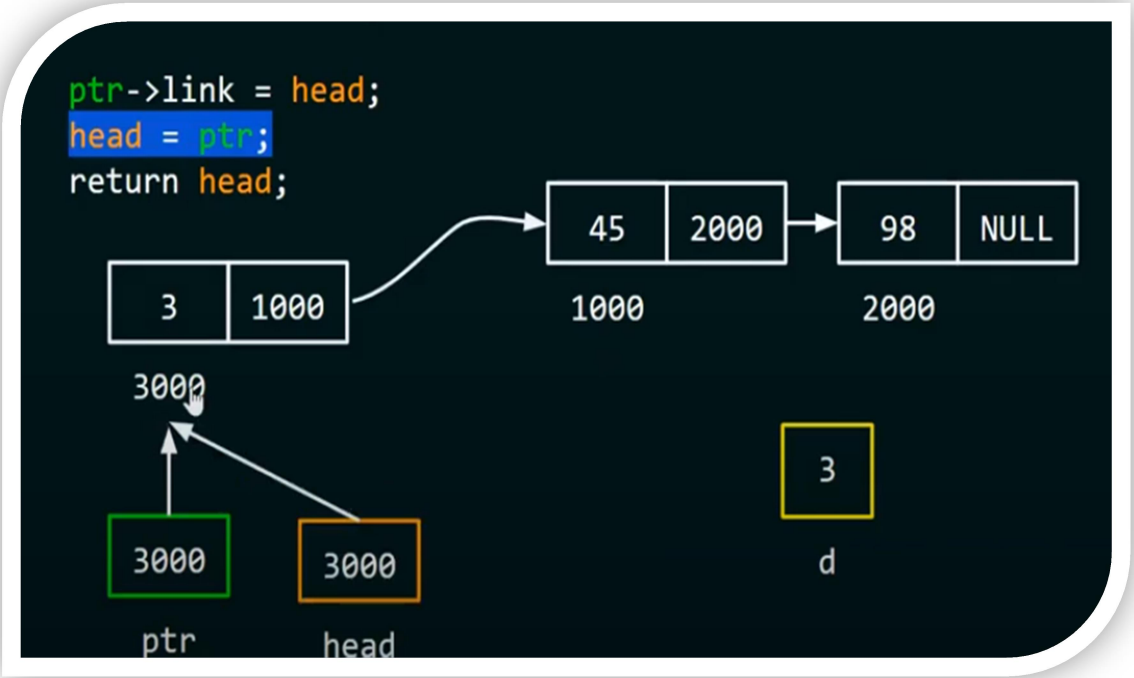
```
        temp=ptr;
        printf("starting point(head)=%d\n",temp);
        while(temp!=NULL)
        {
                printf("%d->",temp->data);
                printf("%u|",temp->link);
           temp=temp->link;
        }
}
```

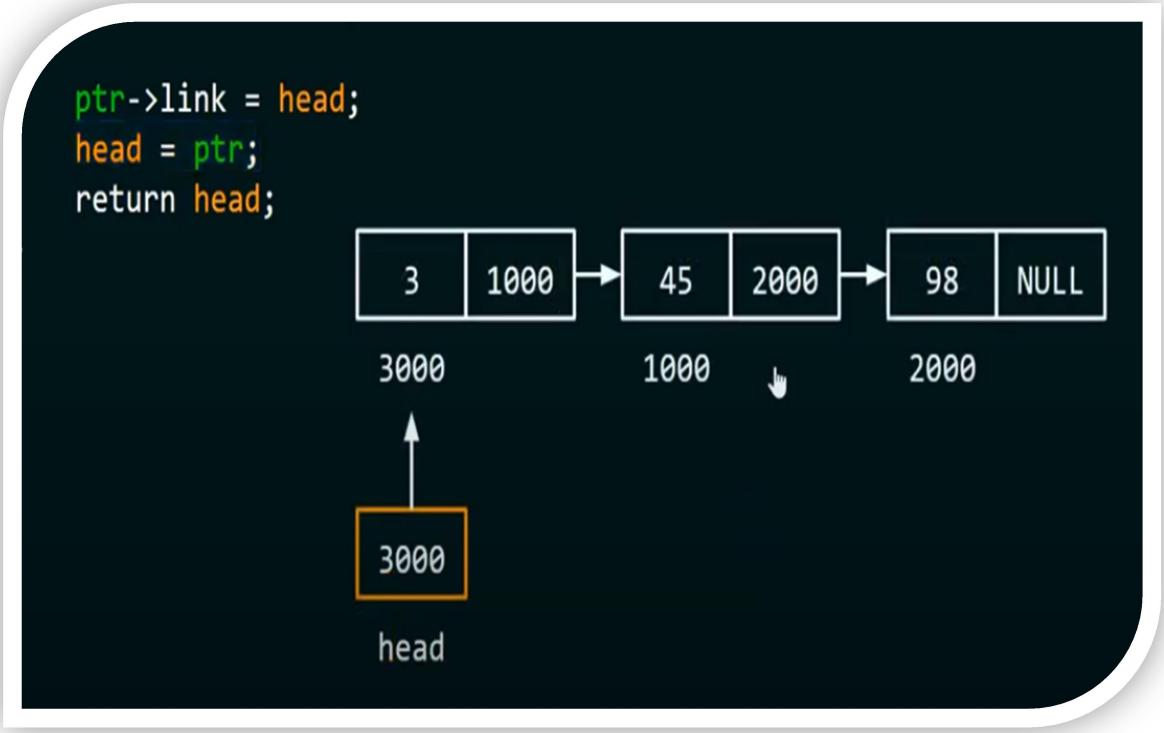
Output

```
C:\Users\HYMAVATHI\Desktop\Btech E1 sem-2\Data structure\DS Programmes\Untitled4.exe
Enter no.of nodes=3
enter element 1=10

10=11755088
enter element 2=20

20=11755104
enter element 3=30

30=11755120
10->11755104|20->11755120|30->0|
enter a element=40

40=11755136
starting point(head)=11755136
40->11755088|10->11755104|20->11755120|30->0|
--------------------------------
Process exited after 248.3 seconds with return value 0
Press any key to continue . . .
```

Explanation







Case-2 : Inserting a node at the ending of a single linked list

Algorithm:

- **Step 1 -** Create a **newNode**(new) with given value.
- **Step 2 -** Enter **ptr->data** by the user and set **ptr->link=NULL**
- **Step 3 –** Create a pointer (temp) pointing to head**(temp=head)**
- **Step 4 -** Check **temp->link!=head.**
- **Step 5 -** If it is **True** then, set **temp= temp->link**
- **Step 6 -** If it is **False** then, set **temp->link=new**

Program

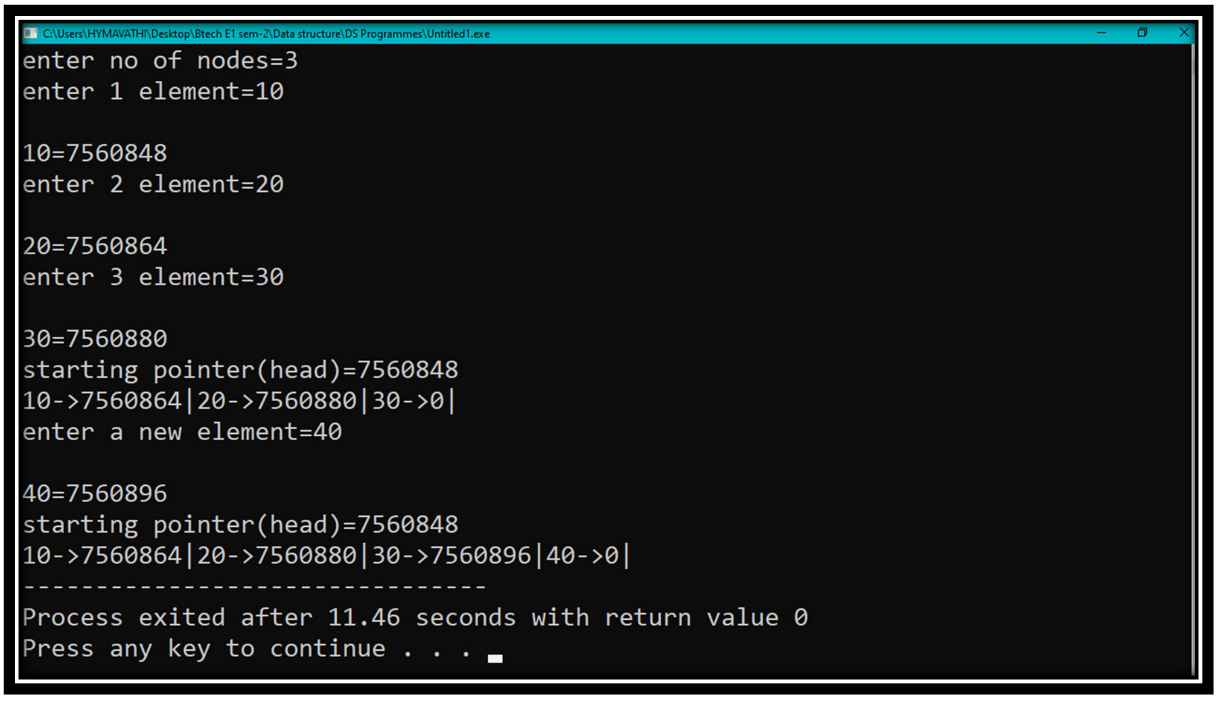
```
#include<stdio.h>
```

```
#include<stdlib.h>
struct node{
        int data;
        struct node *link;
};
int main()
{
        int len,i;
        printf("enter no of nodes=");
        scanf("%d",&len);

        struct node *head,*ptr,*temp;
        for(i=0;i<len;i++)
        {
                ptr=(struct node*)malloc(sizeof(struct node));
                printf("enter %d element=",(i+1));
                scanf("%d",&ptr->data);
                printf("\n%d=%u\n",ptr->data,&ptr->data);
                if(i==0)
                {
                   head=temp=ptr;
            }
            else
            {
```

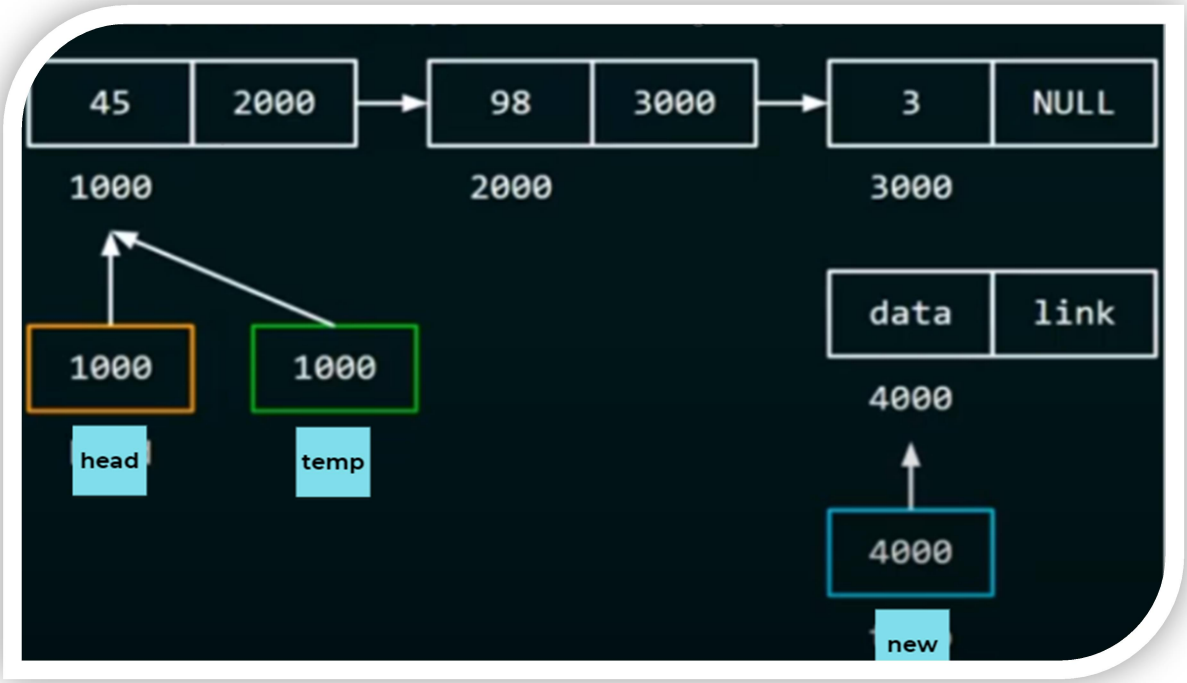
```
            temp->link=ptr;
            temp=ptr;
            }

    }
    temp->link=NULL;
    temp=head;
    printf("starting pointer(head)=%d\n",temp);
    while(temp!=NULL)
    {
            printf("%d->",temp->data);
            printf("%u|",temp->link);
            temp=temp->link;
    }


    struct node *new;
    new=(struct node*)malloc(sizeof(struct node));
    printf("\nenter a new element=");
    scanf("%d",&new->data);
    printf("\n%d=%u\n",new->data,&new->data);
    new->link=NULL;
    temp=head;
    while(temp->link!=NULL)
```

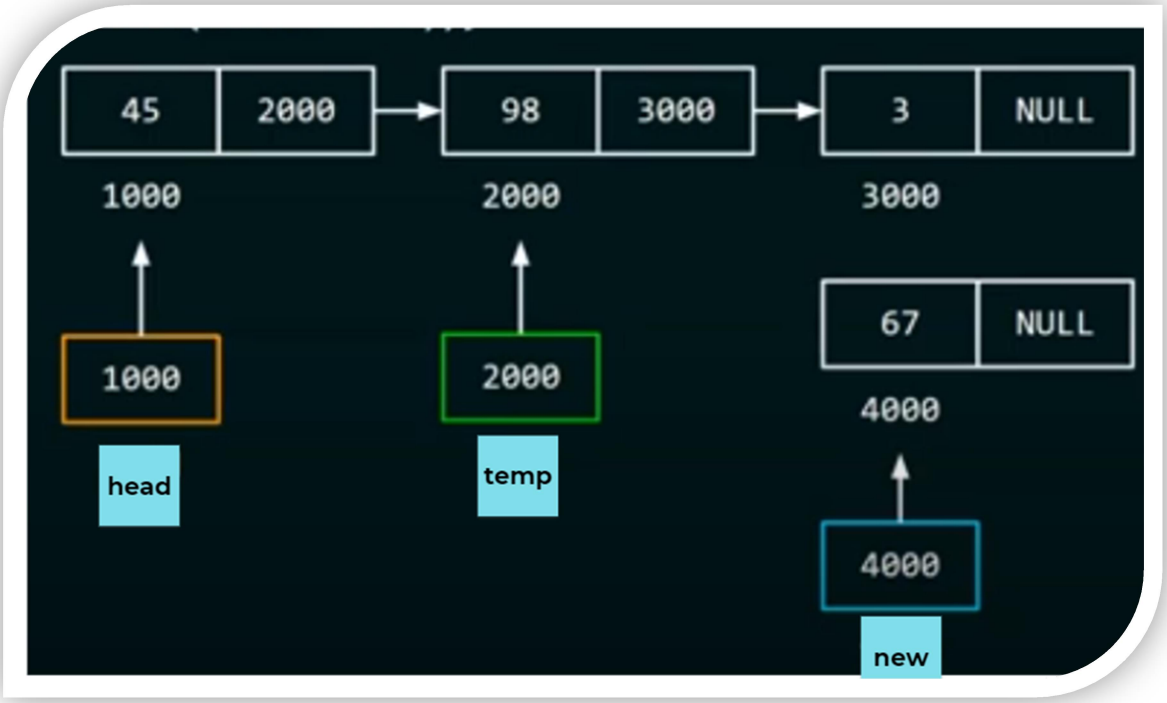
```
        {
            temp=temp->link;
        }
        temp->link=new;
        temp=head;
        printf("starting pointer(head)=%d\n",temp);
        while(temp!=NULL)
        {
                printf("%d->",temp->data);
                printf("%u|",temp->link);
                temp=temp->link;
        }
}
```

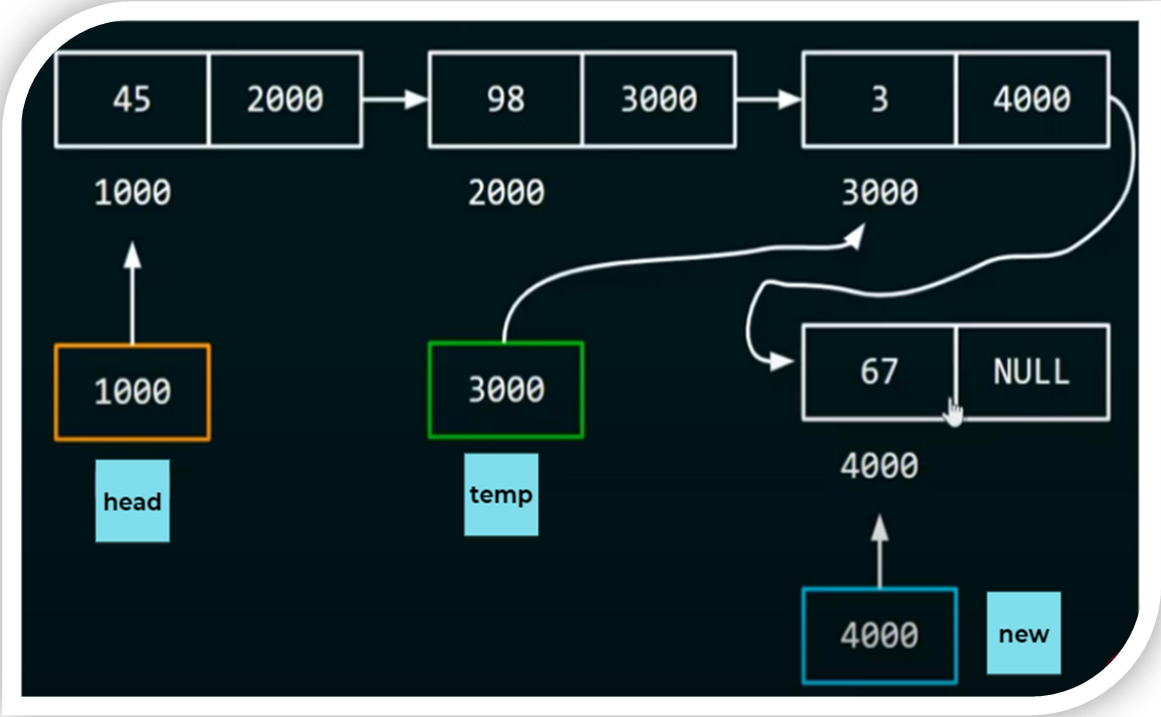
Output



Explanation







Case-3 : Inserting a node at the middle of a single linked list

Algorithm

- **Step 1 -** Create a **newNode**(ptr2) with given value.
- **Step 2 –** Create a pointer (ptr) pointing to head**(ptr=head)**
- **Step 3 –** Ask the user which position(p) they want to add the node.
- **Step 4 -** Check **p!=2.**
- **Step 5 -** If it is **True** then, set **ptr= ptr->link** and **p—**
- **Step 6 -** If it is **False** then, set **ptr2->link=ptr->link** and **ptr->link=ptr2**

Program

```
#include<stdio.h>
#include<stdlib.h>
struct node{
        int data;
        struct node *link;
};
int main()
{
        int len,i;
        printf("Enter no.of nodes=");
        scanf("%d",&len);
        struct node *head,*pn,*temp;
        for(i=0;i<len;i++)
        {
                pn=(struct node*)malloc(sizeof(struct node));
                printf("enter element %d=",(i+1));
                scanf("%d",&pn->data);
                printf("\n%d=%u\n",pn->data,&pn->data);

                if(i==0)
                {
                        head=temp=pn;
                }
```

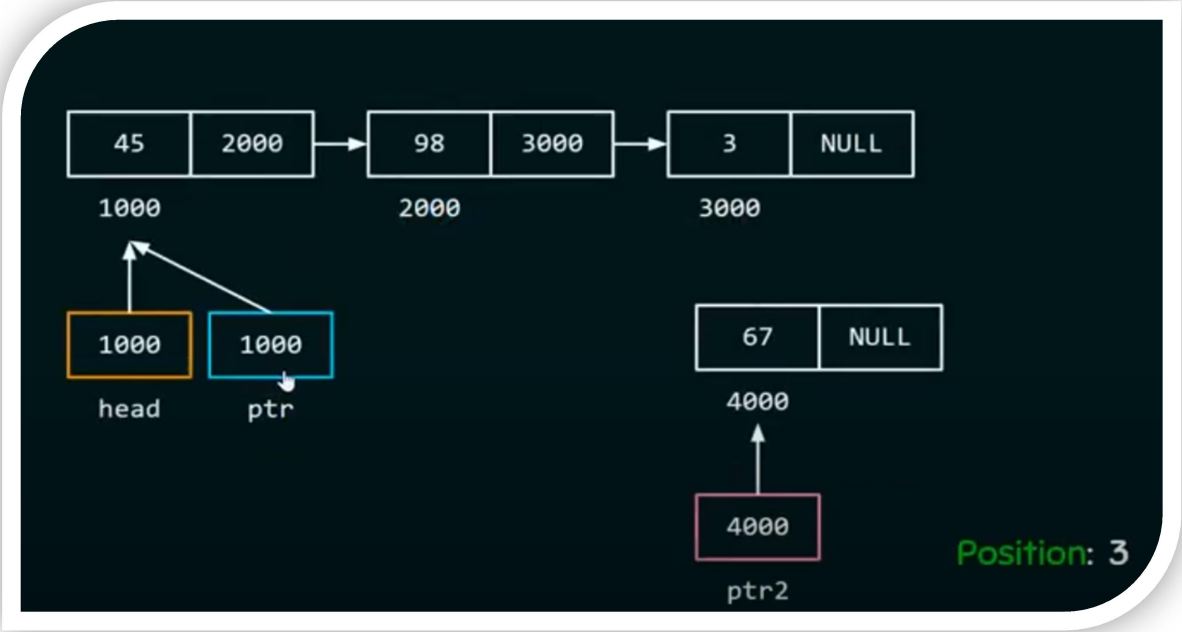
```
        else
        {
                temp->link=pn;
                temp=pn;
        }

}
temp->link=NULL;
temp=head;
while(temp!=NULL)
{
        printf("%d->",temp->data);
        printf("%u|",temp->link);
   temp=temp->link;
}
struct node *ptr,*ptr2;
ptr2=(struct node*)malloc(sizeof(struct node));
printf("\nenter a element=");
scanf("%d",&ptr2->data);
printf("\n%d=%u\n",ptr2->data,&ptr2->data);
ptr2->link=NULL;
int p;
printf("enter which position we want add new node=");
```

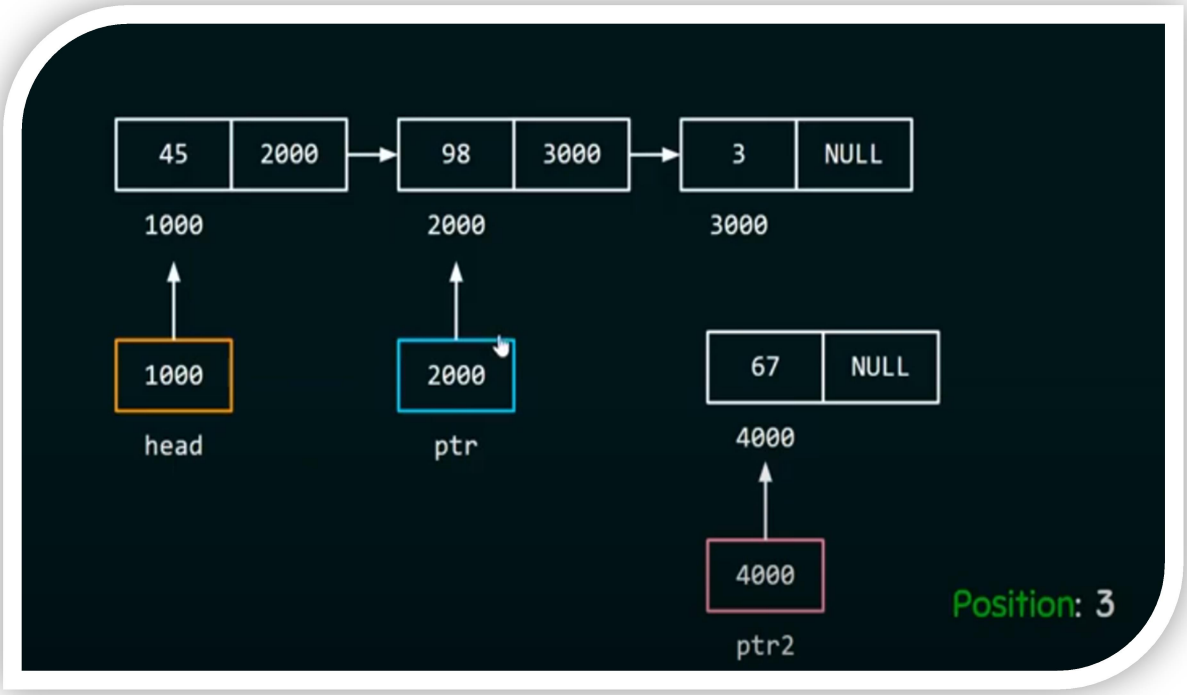
```
        scanf("%d",&p);
        ptr=head;
        while(p!=2)
        {
                ptr=ptr->link;
                p--;
        }
        ptr2->link=ptr->link;
        ptr->link=ptr2;
        ptr=head;
        printf("starting point(head)=%d\n",ptr);
        while(ptr!=NULL)
        {
                printf("%d->",ptr->data);
                printf("%u|",ptr->link);
                ptr=ptr->link;
        }
}
```

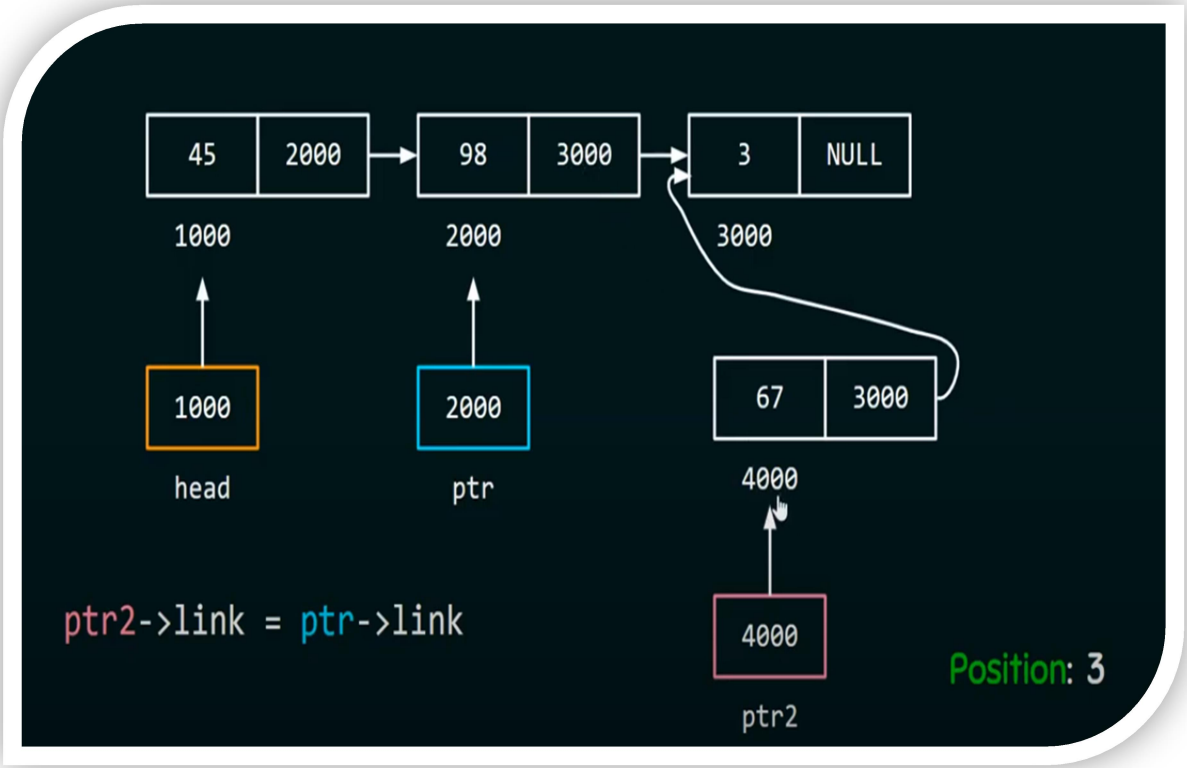
Output

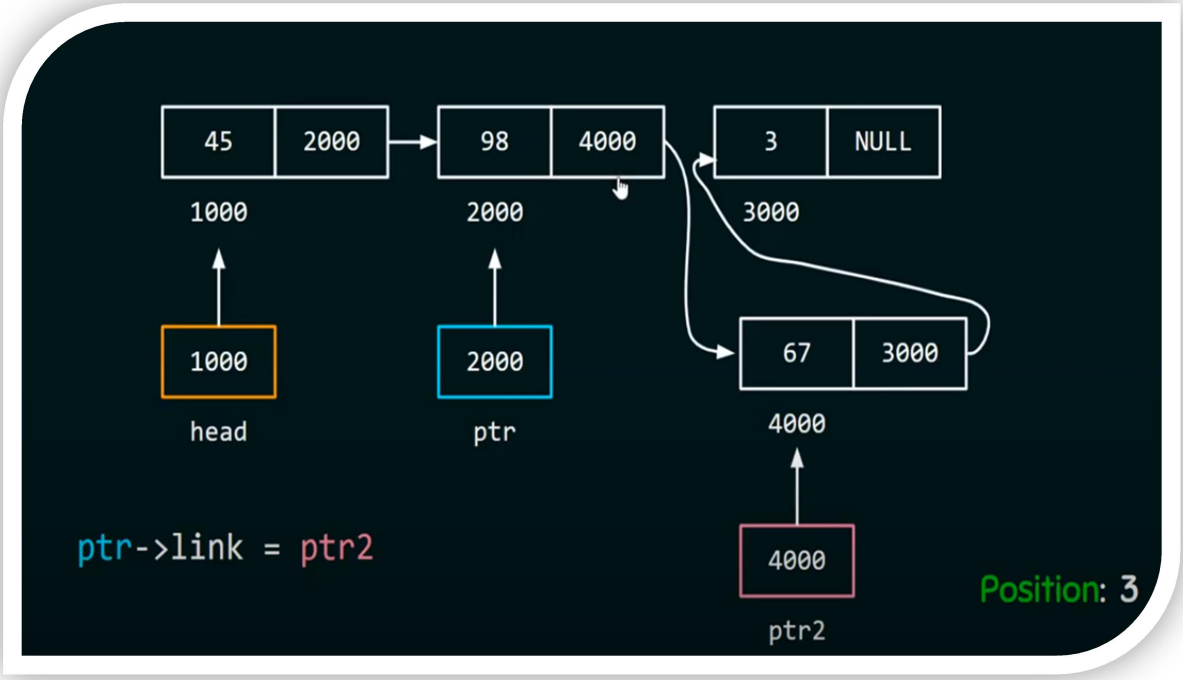
```
C:\Users\HYMAVATHI\Desktop\Btech E1 sem-2\Data structure\DS Programmes\Insert at middle.exe
Enter no.of nodes=3
enter element 1=10

10=6971056
enter element 2=20

20=6971072
enter element 3=30

30=6971088
10->6971072|20->6971088|30->0|
enter a element=40

40=6971104
enter which position we want add new node=3
starting point(head)=6971056
10->6971072|20->6971104|40->6971088|30->0|
--------------------------------
Process exited after 18.57 seconds with return value 0
Press any key to continue . . . _
```

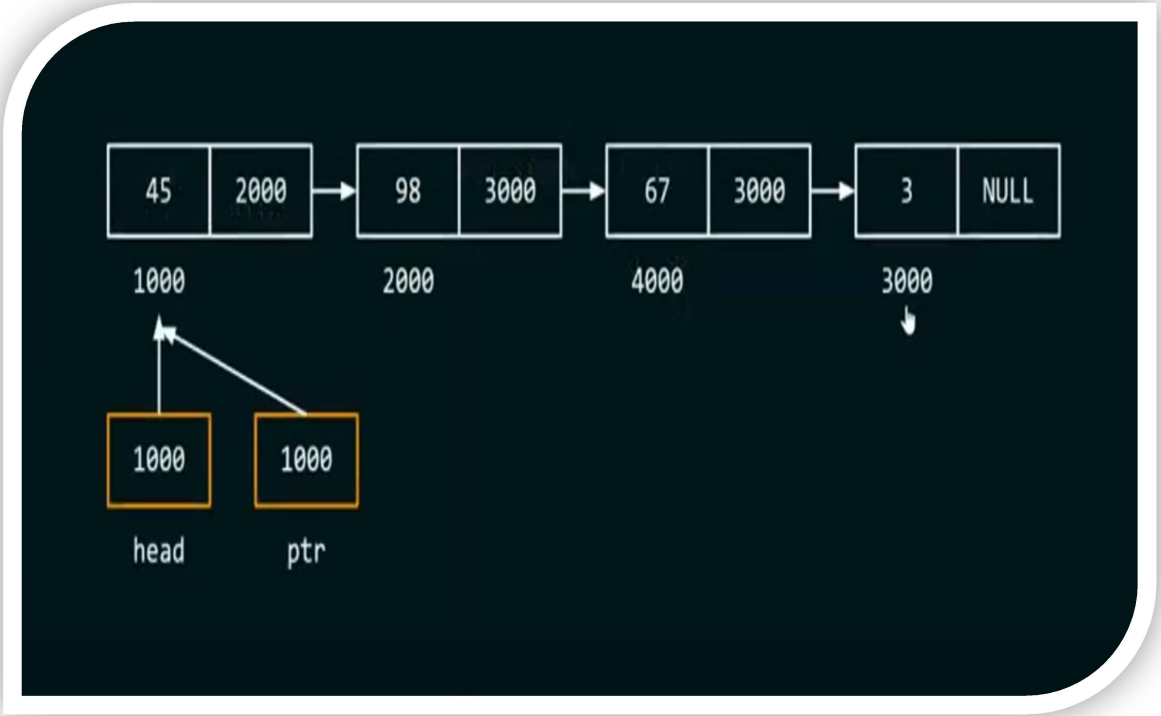
Explanation




45
2000
98
3000
3
NULL
1000
2000
3000
1000
head
2000
ptr
67
NULL
4000
4000
ptr2
Position: 3


45
2000
98
3000
3
NULL
1000
2000
3000
1000
head
2000
ptr
67
3000
4000
ptr2->link = ptr->link
4000
ptr2
Position: 3


45
2000
98
4000
3
NULL
1000
2000
3000
1000
2000
67
3000
head
ptr
4000
ptr->link = ptr2
4000
ptr2
Position: 3


45
2000
98
3000
67
3000
3
NULL
1000
2000
4000
3000
1000
1000
head
ptr

Deletion{Single linked list}

Case-1 : Deleting the first node of a single linked list.

Algorithm

- **Step 1 -** Create a pointer **(temp)** pointing to the head**(temp=temp)**
- **Step 2 -** Set **head=head->link**
- **Step 3 -** Then free **temp**

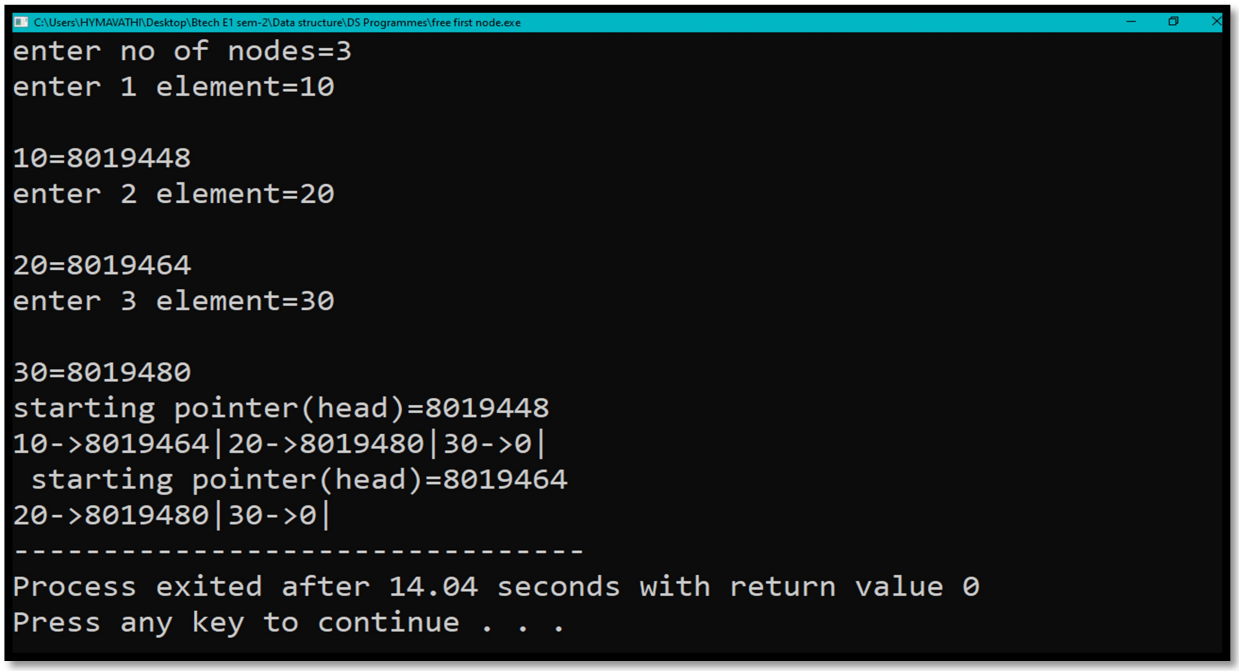
Program

```
#include<stdio.h>
#include<stdlib.h>
struct node{
        int data;
        struct node *link;
};
int main()
{
        int len,i;
        printf("enter no of nodes=");
        scanf("%d",&len);

        struct node *head,*ptr,*temp;
        for(i=0;i<len;i++)
        {
```

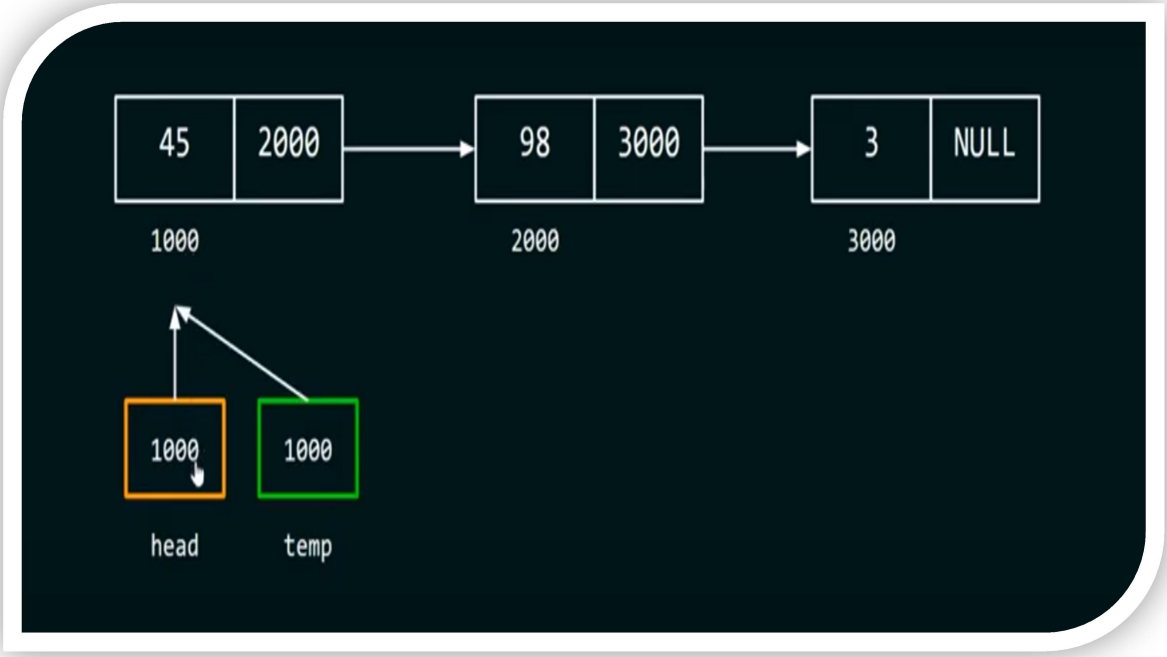
```
            ptr=(struct node*)malloc(sizeof(struct node));
            printf("enter %d element=",(i+1));
            scanf("%d",&ptr->data);
            printf("\n%d=%u\n",ptr->data,&ptr->data);
            if(i==0)
            {
               head=temp=ptr;
        }
        else
        {
            temp->link=ptr;
            temp=ptr;
            }

    }
    temp->link=NULL;
    temp=head;
    printf("starting pointer(head)=%d\n",temp);
    while(temp!=NULL)
    {
            printf("%d->",temp->data);
            printf("%u|",temp->link);
            temp=temp->link;
    }
    temp=head;
```

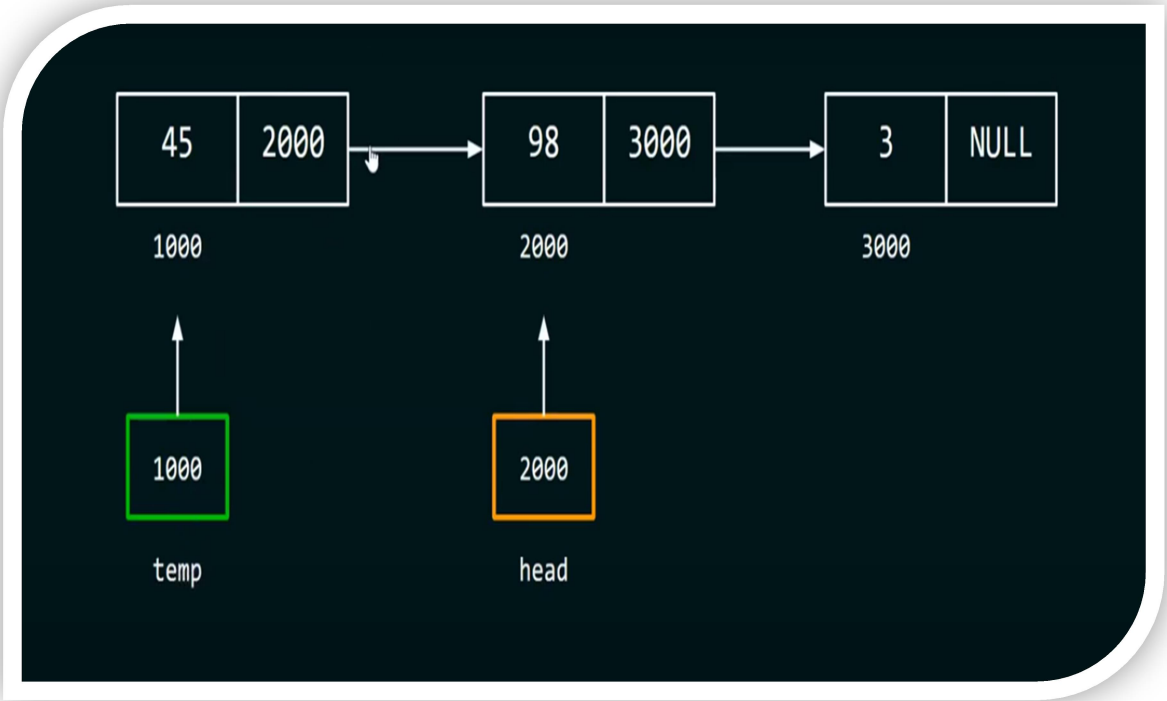
```
        head=head->link;
        free(temp);
        printf("\n starting pointer(head)=%d\n",head);
        while(head!=NULL)
        {
                printf("%d->",head->data);
                printf("%u|",head->link);
                head=head->link;
        }
}
```

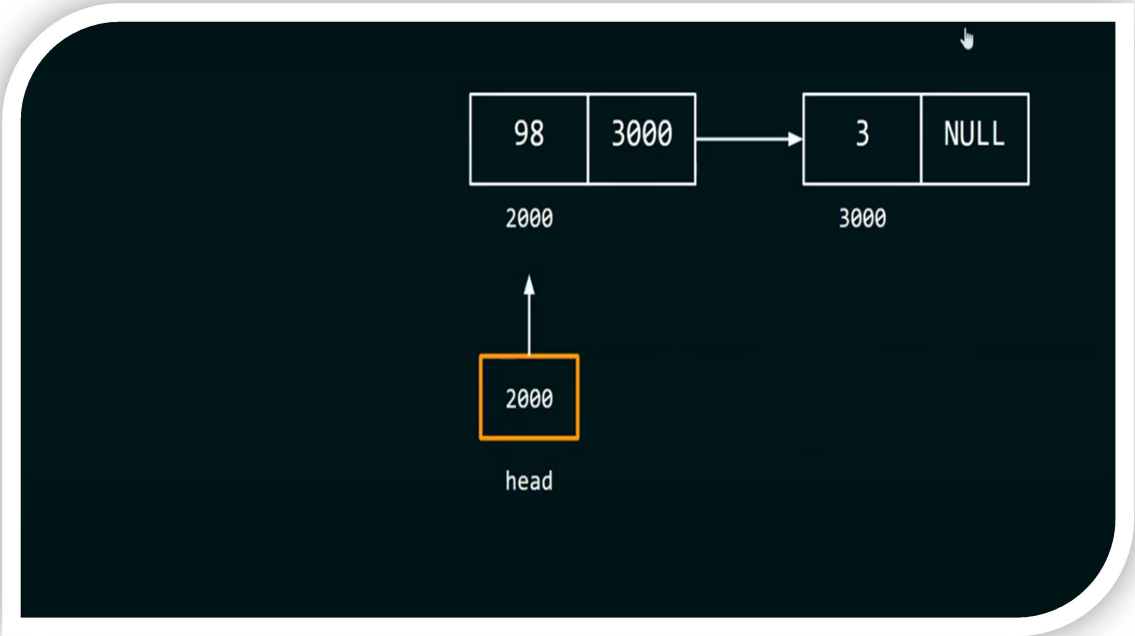
Output



Explanation


45
2000
98
3000
3
NULL
1000
2000
3000
1000
1000
head
temp


45
2000
98
3000
3
NULL
1000
2000
3000
1000
2000
temp
head



Case-2 : Deleting the last node of a single linked list.

Algorithm

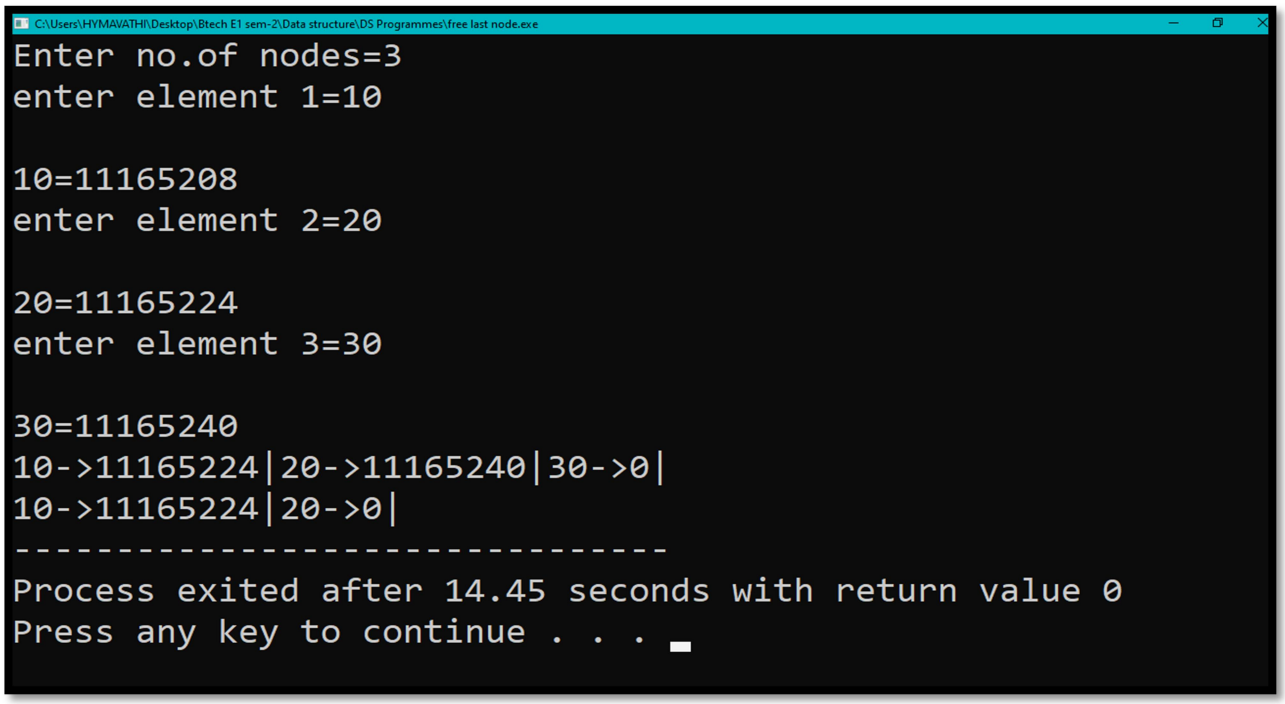
- **Step 1 -** Create a pointer **(temp)** pointing to the head**(temp=temp)**
- **Step 2 -** Check **temp->link->link!=0**
- **Step 3 -** If it is **True** then, set **temp= temp->link**
- **Step 4 -** If it is **False** then, free **temp->link**
- **Step 5 -** Then **temp->link=NULL**

Program

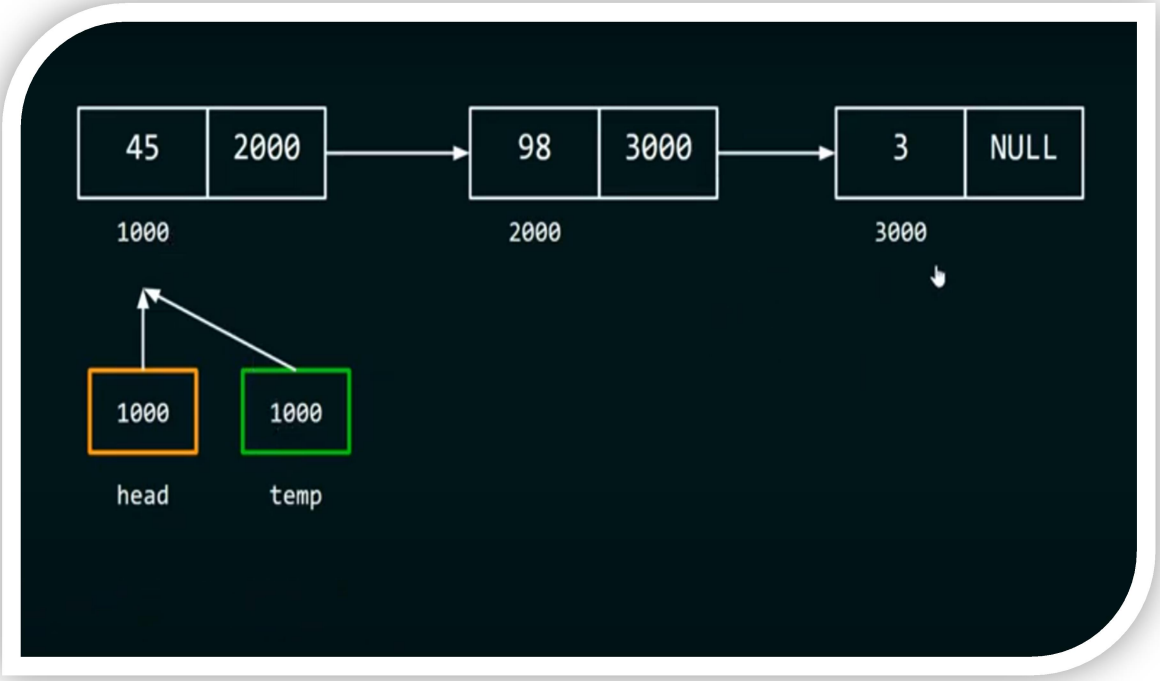
```
#include<stdio.h>
#include<stdlib.h>
struct node{
        int data;
        struct node *link;
};
int main()
{
        int len,i;
        printf("Enter no.of nodes=");
        scanf("%d",&len);
        struct node *head,*ptr,*temp;
        for(i=0;i<len;i++)
        {
                ptr=(struct node*)malloc(sizeof(struct node));
                printf("enter element %d=",(i+1));
                scanf("%d",&ptr->data);
                printf("\n%d=%u\n",ptr->data,&ptr->data);

                if(i==0)
                {
                        head=temp=ptr;
                }

                else
                {
                        temp->link=ptr;
                        temp=ptr;
                }

        }
        temp->link=NULL;
        temp=head;
        while(temp!=NULL)
        {
                printf("%d->",temp->data);
                printf("%u|",temp->link);
```

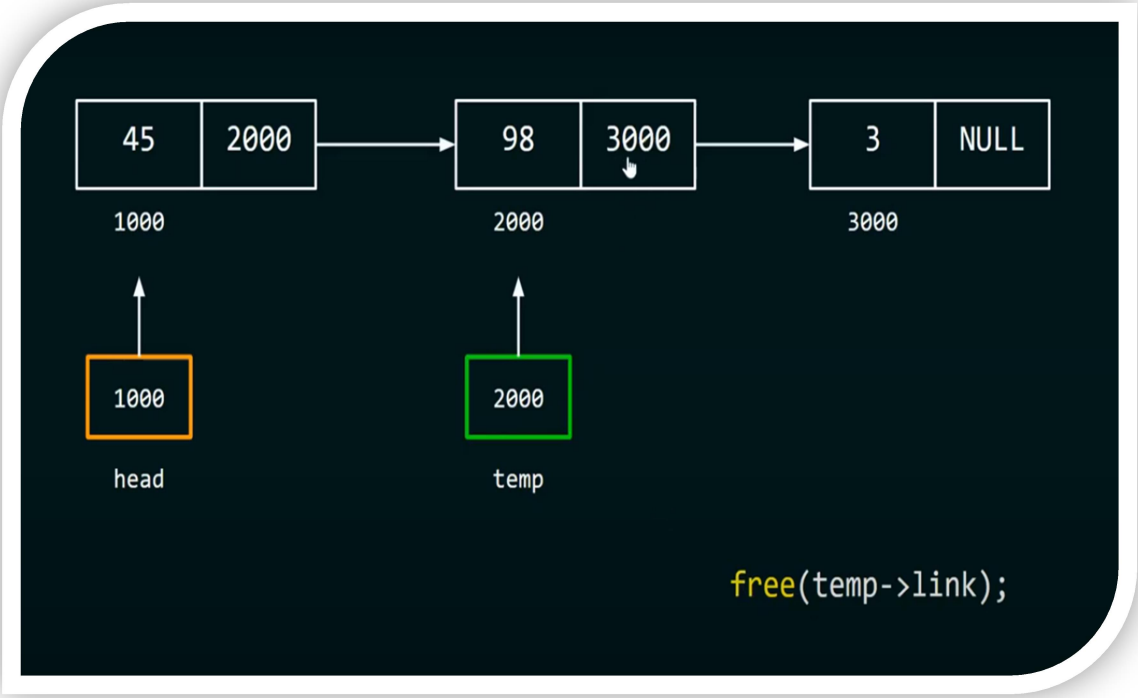
```
            temp=temp->link;
        }
        temp=head;
        while(temp->link->link!=0)
        {
                temp=temp->link;
        }
        free(temp->link);
        temp->link=NULL;
        temp=head;
        printf("\n");
        while(temp!=NULL)
        {
                printf("%d->",temp->data);
                printf("%u|",temp->link);
                temp=temp->link;
        }
}
```

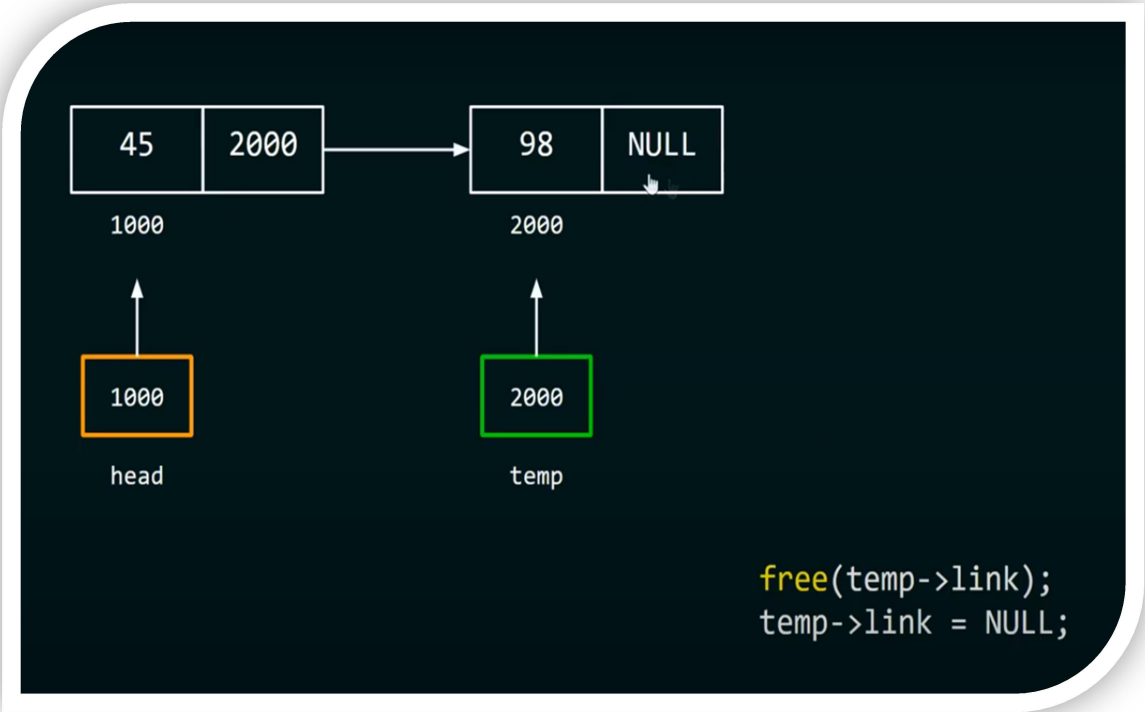
Output :



Explanation







Case-3 : Deleting middle node of a single linked list

Algorithm

- **Step 1 -** Create two pointer **previous** and **current** pointing to **head**. **current=previous=head**
- **Step 2-** Ask the user which position(p) they want to add the node.
- **Step 3-** Check **p!=1.**
- **Step 4-** If it is **True** then, set **previous=current** and **current=current->line**
- **Step 5-**Then decrement p--;
- **Step6-** If it is **False** then, set **previous->link=current->link**
- **Step7-** Then **free(current)**

Program

```
#include<stdio.h>
#include<stdlib.h>
struct node{
        int data;
        struct node *link;
};
int main()
{
        int len,i;
        printf("Enter no.of nodes=");
        scanf("%d",&len);
        struct node *head,*pn,*temp;
        for(i=0;i<len;i++)
        {
                pn=(struct node*)malloc(sizeof(struct node));
                printf("enter element %d=",(i+1));
                scanf("%d",&pn->data);
                printf("\n%d=%u\n",pn->data,&pn->data);

                if(i==0)
                {
                        head=temp=pn;
                }
```

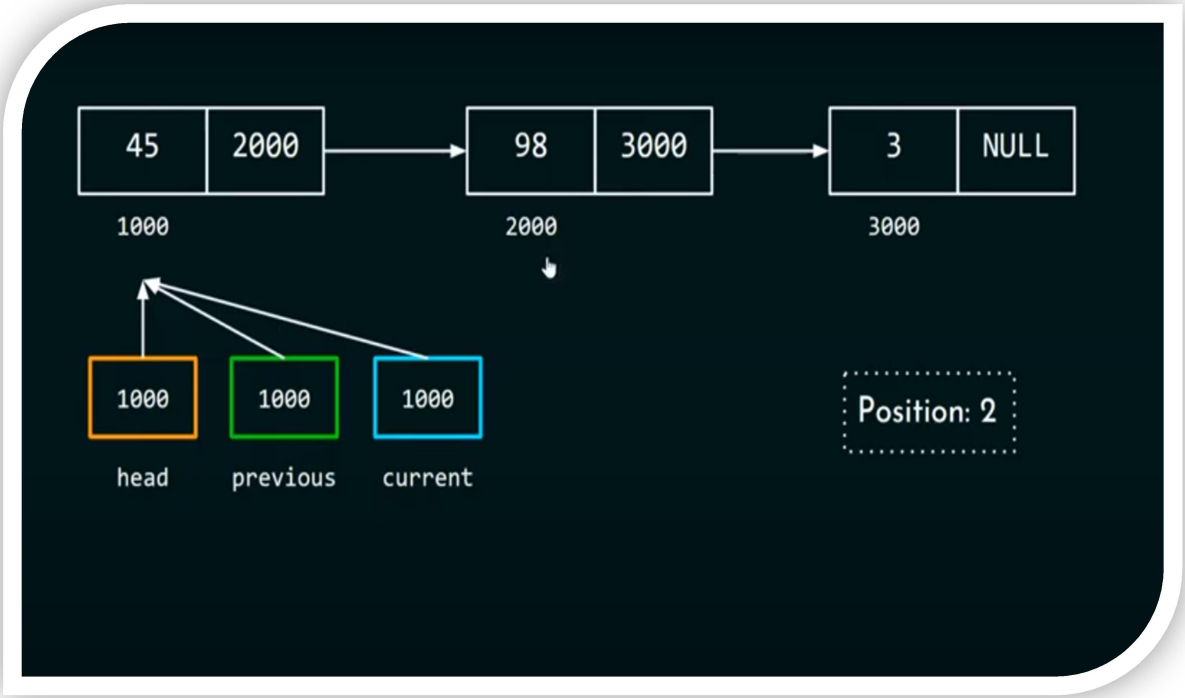
```
        else
        {
                temp->link=pn;
                temp=pn;
        }

}
temp->link=NULL;
temp=head;
while(temp!=NULL)
{
        printf("%d->",temp->data);
        printf("%u|",temp->link);
   temp=temp->link;
}
struct node *current,*previous;
int p;
printf("\nwhich position node you want to del=");
scanf("%d",&p);
current=previous=head;
while(p!=1)
{
        previous=current;
        current=current->link;
```

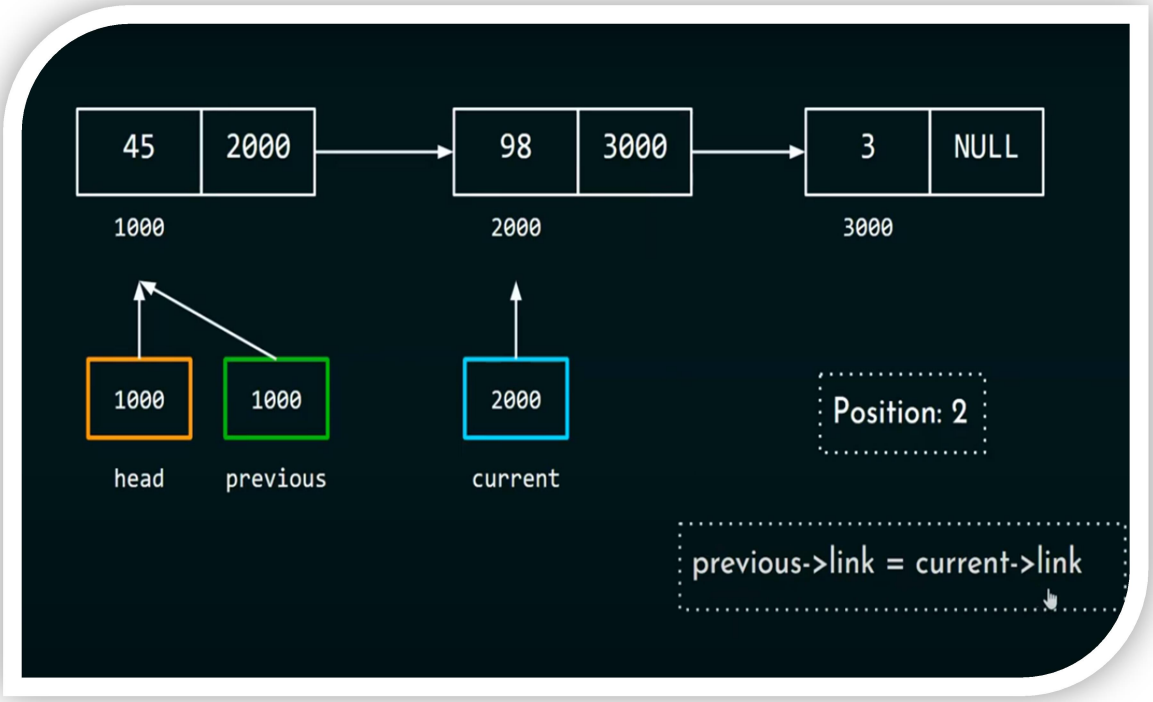
```
            p--;
        }
        previous->link=current->link;
        free(current);
        temp=head;
        printf("starting point(head)=%d\n",temp);
        while(temp!=NULL)
        {
            printf("%d->",temp->data);
            printf("%u|",temp->link);
            temp=temp->link;
        }
}
```

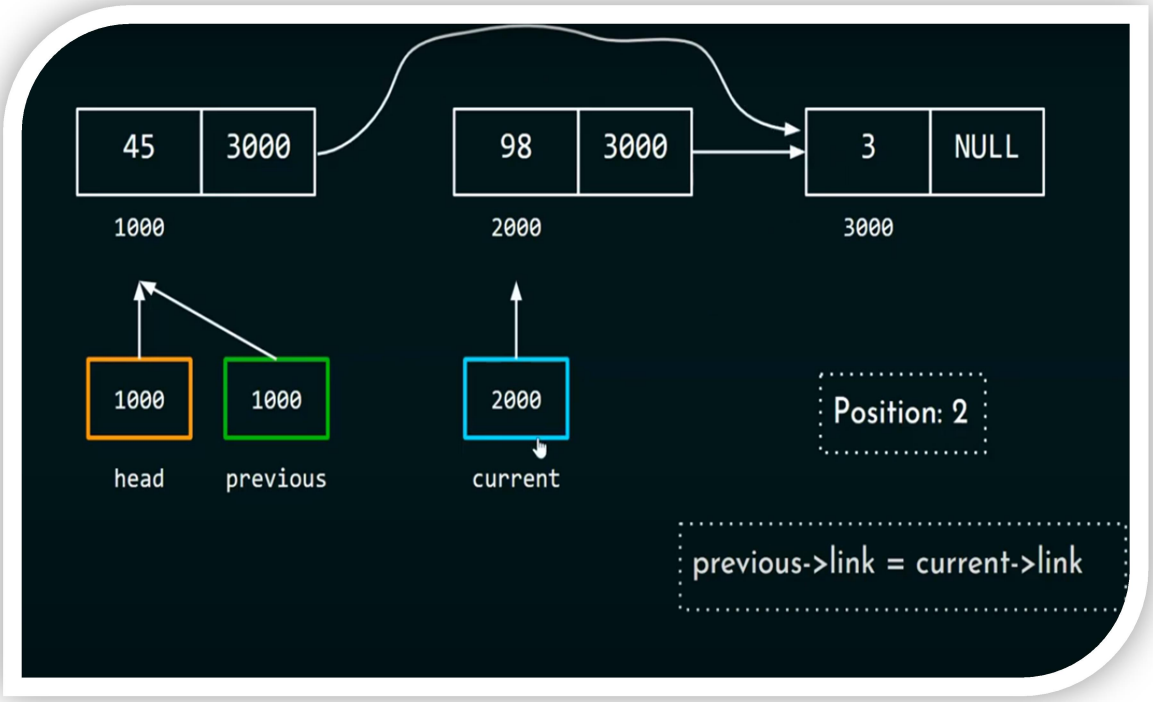
Output :

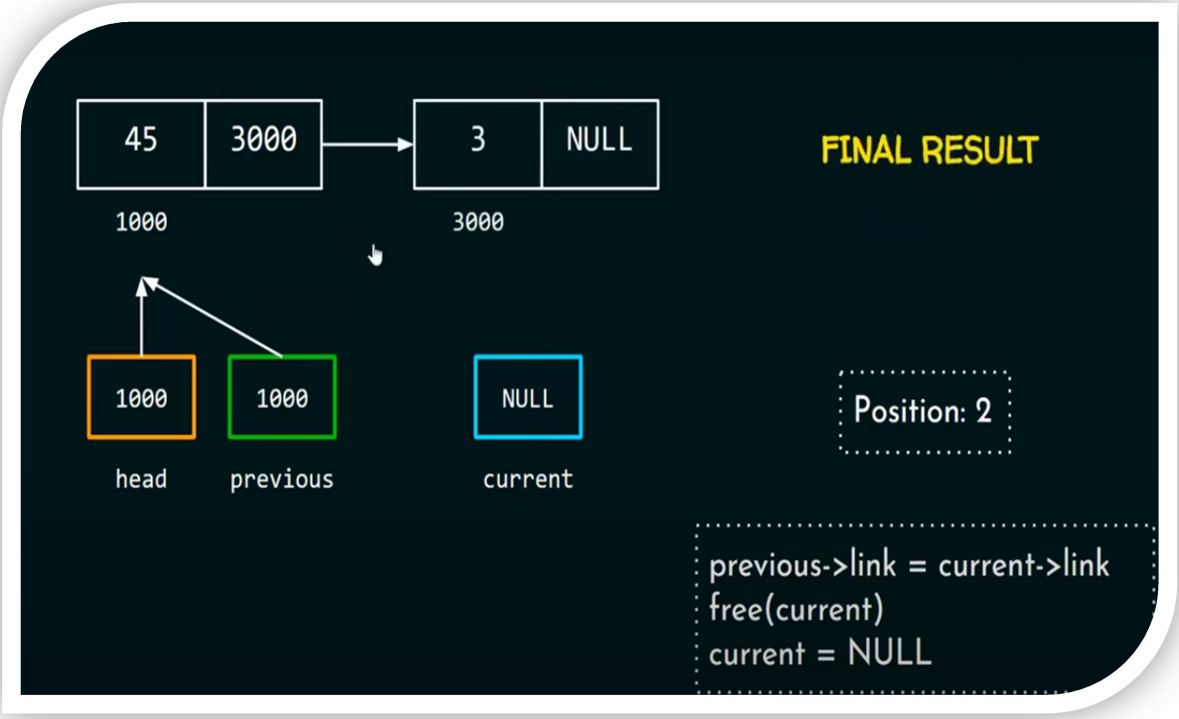
```
C:\Users\HYMAVATHI\Desktop\Btech E1 sem-2\Data structure\DS Programmes\free middle node.exe
Enter no.of nodes=5
enter element 1=11

11=12738144
enter element 2=22

22=12738160
enter element 3=33

33=12738176
enter element 4=44

44=12738192
enter element 5=55

55=12738208
11->12738160|22->12738176|33->12738192|44->12738208|55->0|
which position node you want to del=3
starting point(head)=12738144
11->12738160|22->12738192|44->12738208|55->0|
--------------------------------
Process exited after 17.57 seconds with return value 0
```


Explanation




45
2000
98
3000
3
NULL
1000
2000
3000
1000
1000
2000
head
previous
current
Position: 2
previous->link = current->link


45
3000
98
3000
3
NULL
1000
2000
3000
1000
1000
2000
head
previous
current
Position: 2
previous->link = current->link


45
3000
3
NULL
FINAL RESULT
1000
3000
1000
1000
NULL
Position: 2
head
previous
current
previous->link = current->link
free(current)
current = NULL