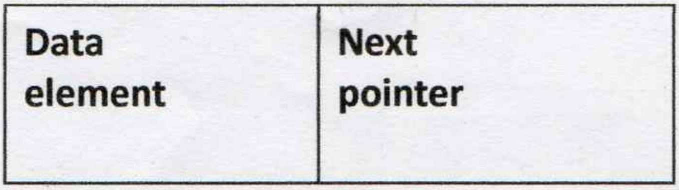
- # Declaration of node:

```
struct node
{
int data;
struct node *next;
} *new;
```



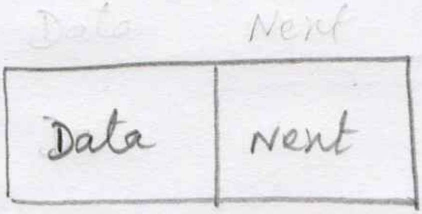
node

Memory Allocation

syntax:

```
New= (struct node *)malloc (size of (struct node));
```

This statement to allocate memory dynamically

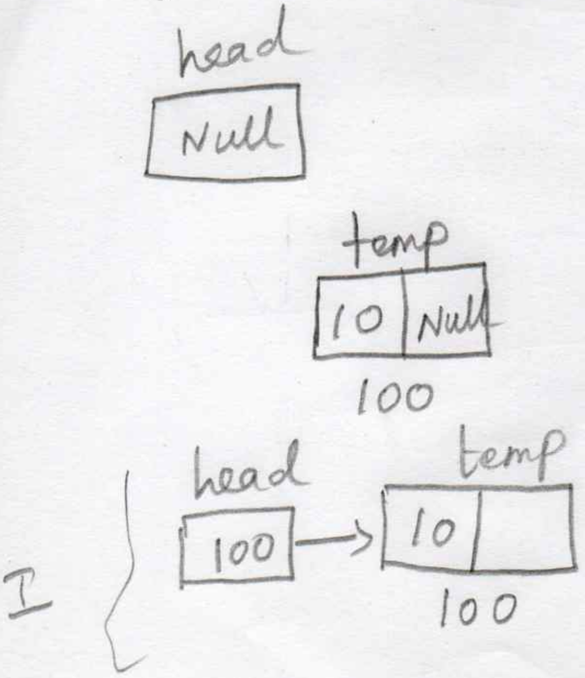


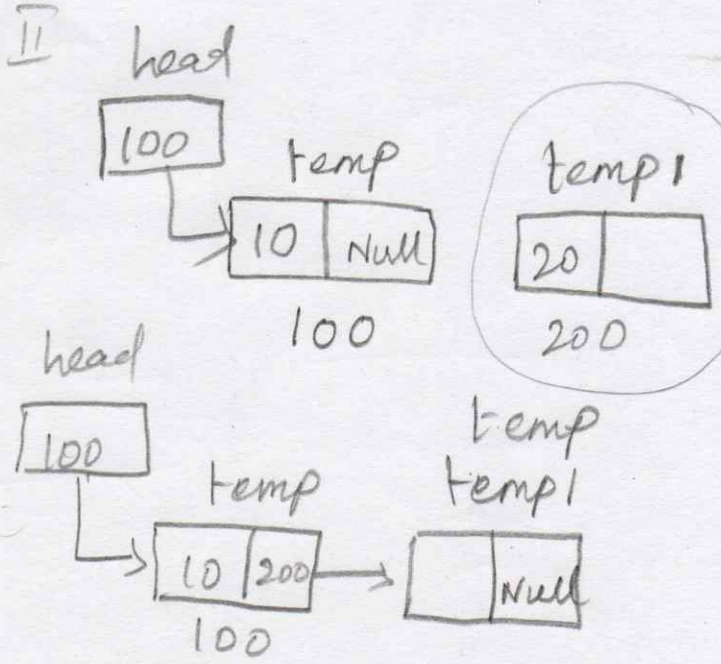
1000

↳ address of the node

Creation of Linked List

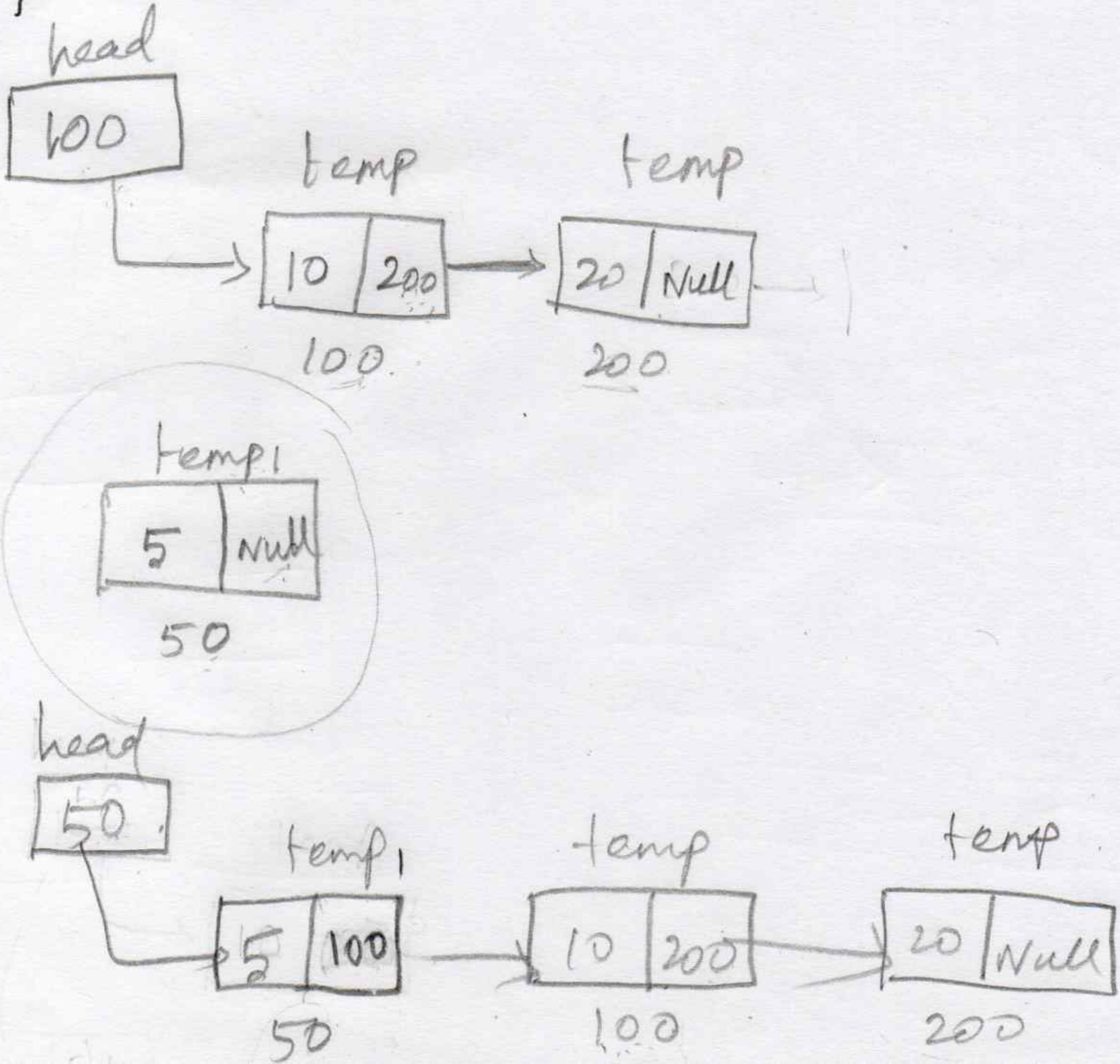
```
    void create(int)

    {

    if(head==null)

    {

    temp=(struct node *)malloc (size of (struct node));
    printf("enter the elements");
I   scanf("%d",&temp->data);
    temp->next=null // links the address field to NULL
    head->next=temp;
    }
    else
    {
    temp1=(struct node *)malloc (size of (struct node));
    printf("enter the elements");
    scanf("%d",&temp1->data);
    temp->next=temp1;
II  temp1->next=null;
    temp=temp1;
    }
    }
```





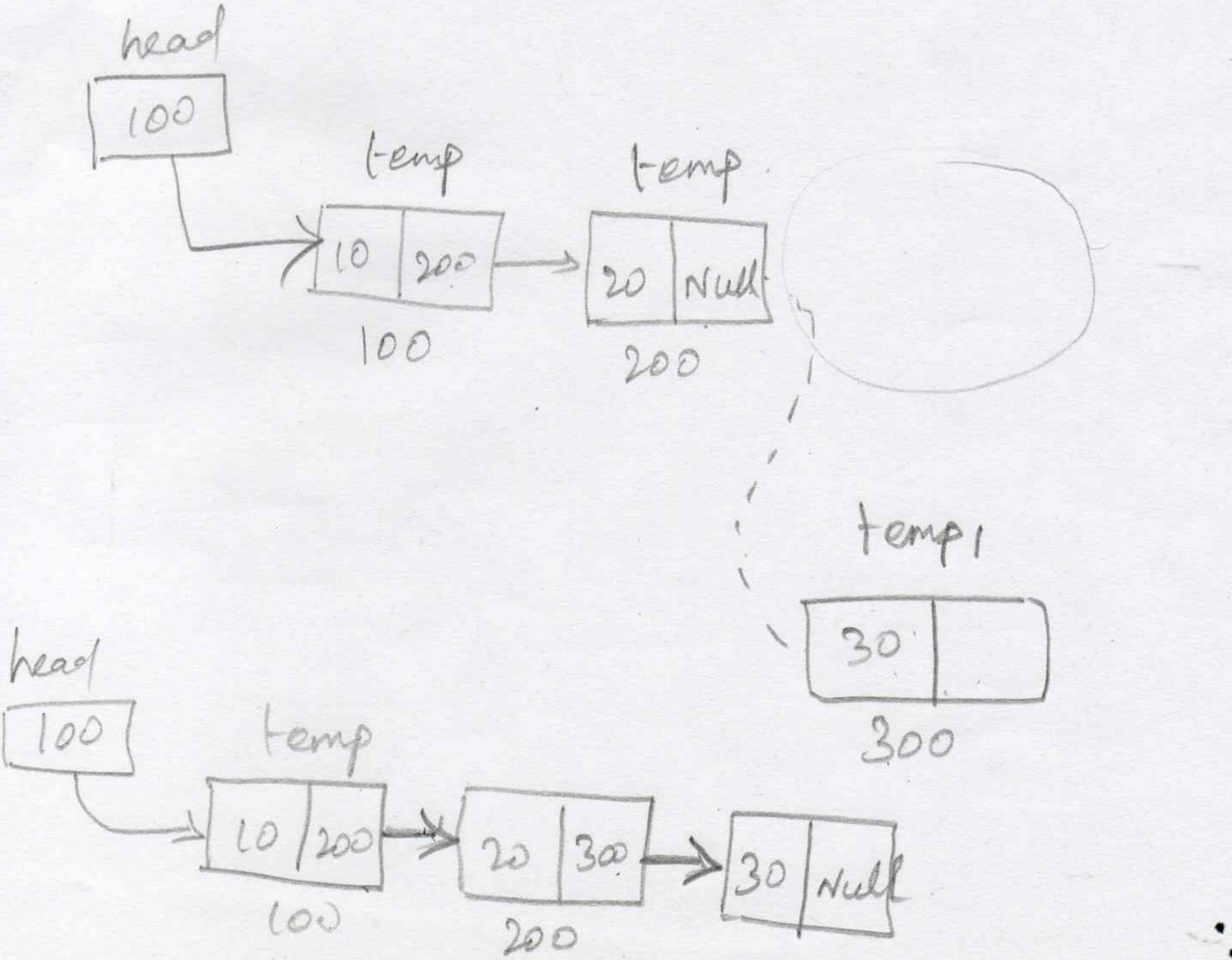
Insertion at beginning

```
void insertfirst(int)

{
head=temp;
struct node *temp1;
temp1=(struct node *)malloc (size of (struct node));
printf("enter the elements");
scanf("%d",&temp1->data);

while (temp->next! == NULL)
    {
       temp=temp->next;
}
   ~~temp->next=temp1~~;
   temp1->next=head;
   head=temp1
   }
```



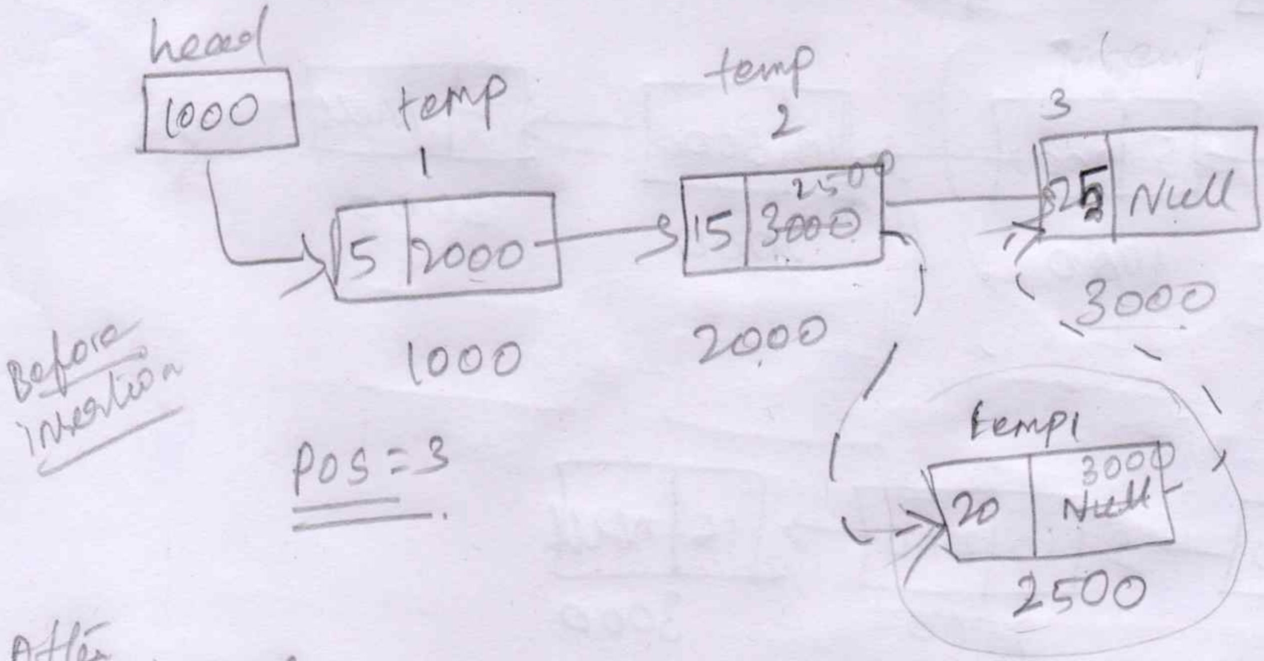
insertion at end

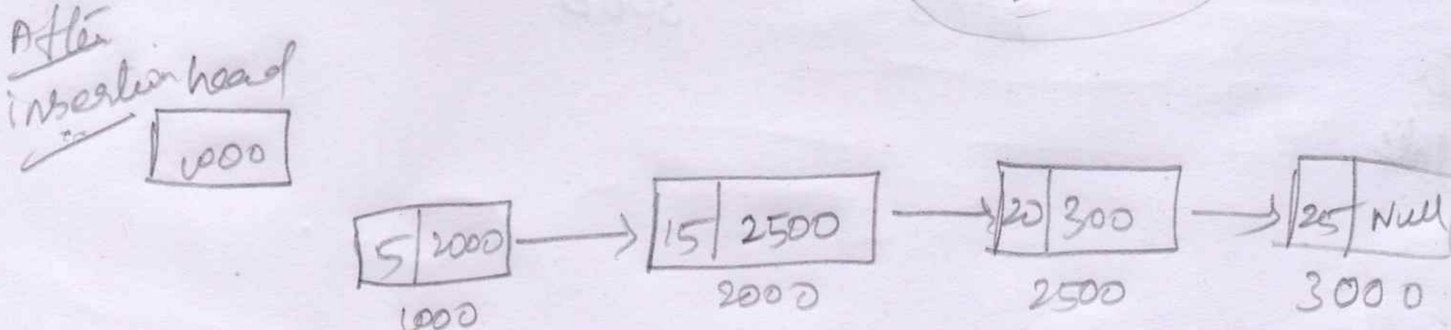
```
void insertend(int)

{
head=temp;
struct node *temp1;
temp1=(struct node *)malloc (size of (struct node));
printf("enter the elements");
scanf("%d",&temp1->data);

while (temp->next! == NULL)
   {
      temp=temp->next;
}
                           300
   temp->next=temp1;
   temp1->next=null;

   }
```



Insertion at middle

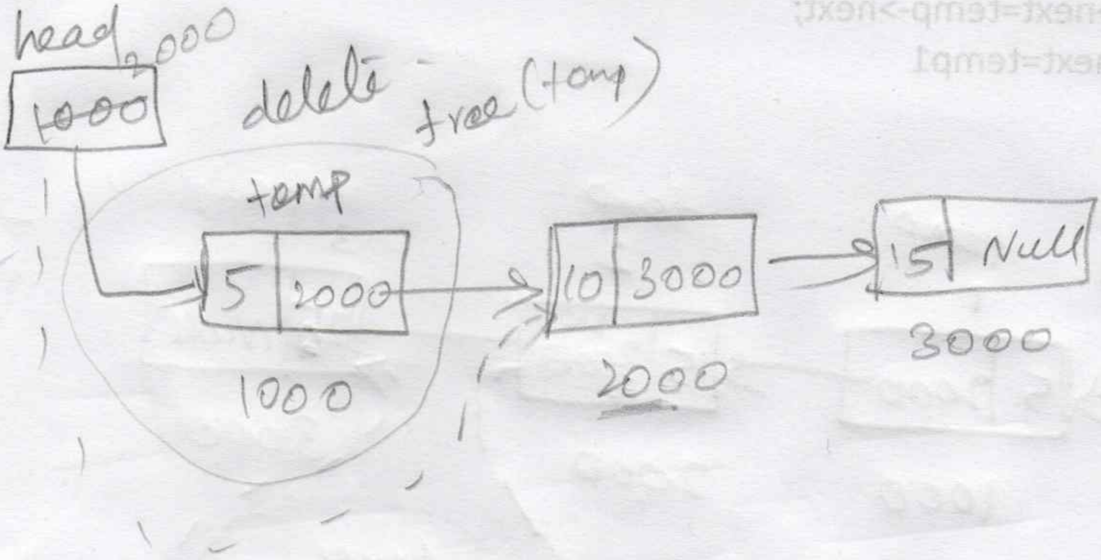
```
void insertend(int)

{
temp=head->next;
struct node *temp1;
temp1=(struct node *)malloc (size of (struct node));
printf(“enter the elements”);
scanf(“%d”,&temp1->data);
printf(“enter the pos”)
scanf(“%d”,&pos);
for(i=1;i<pos;i++)
{
temp=temp->next;
}
temp1->next=temp->next;
temp->next=temp1
}
```



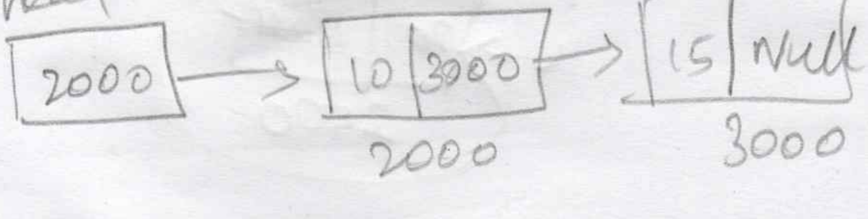


Deletion at first

```
void deletefirst()
  {
    temp=head->next;
    if(head == NULL)
    {
      printf("\nList is empty");
    }
    else
    {
      head->next = temp->next;
      free(temp);
      printf("\n Node deleted from the begining ...");
    }
  }
```



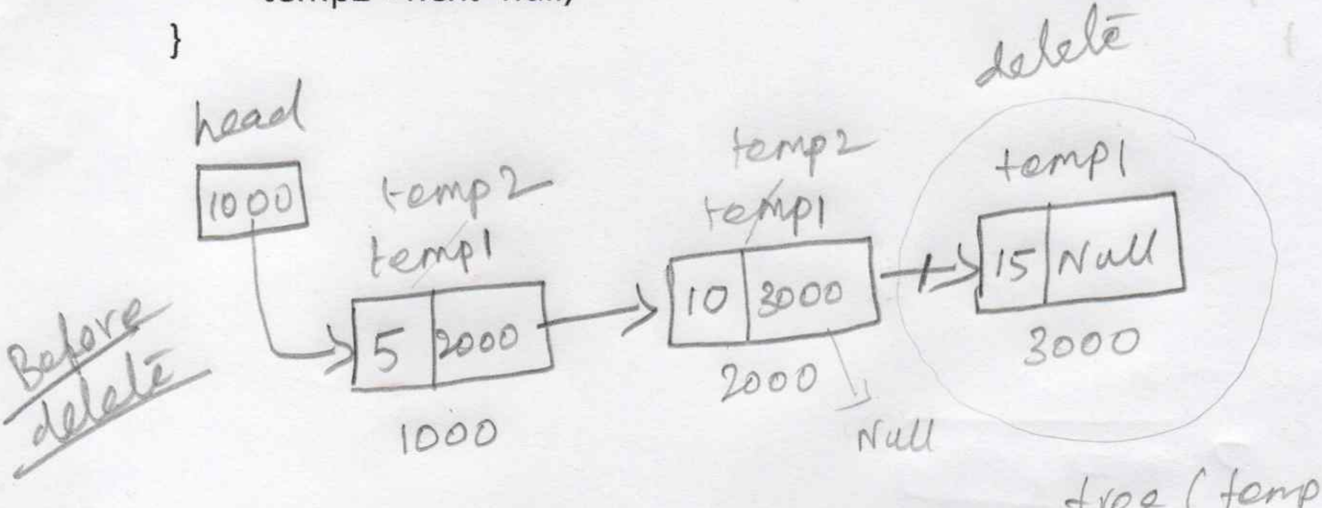
Before delete



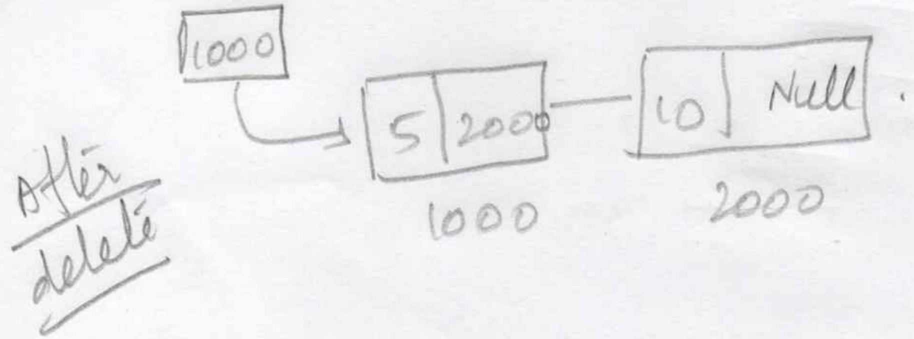
After delete

Deletion at last

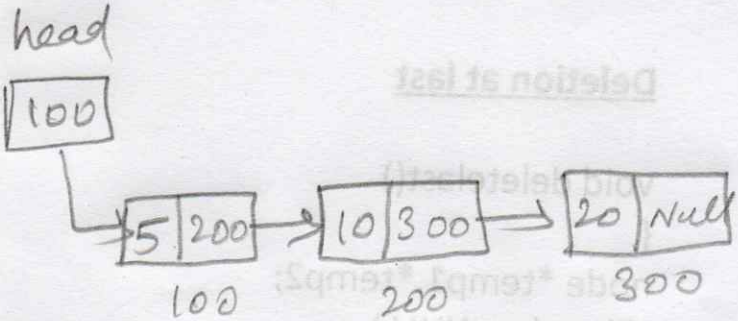
```
void deletelast()
{
node *temp1,*temp2;
if(head == NULL)
      {
         printf("\nList is empty");
      }
      else
      {
       temp1=head->next;
        while(temp1->next!=null)
        temp1=temp1->next;
        temp2=temp1
        }
        free(temp1) — delete
        temp2->next=null;
}
```



free (temp1)
↳ delete the link.



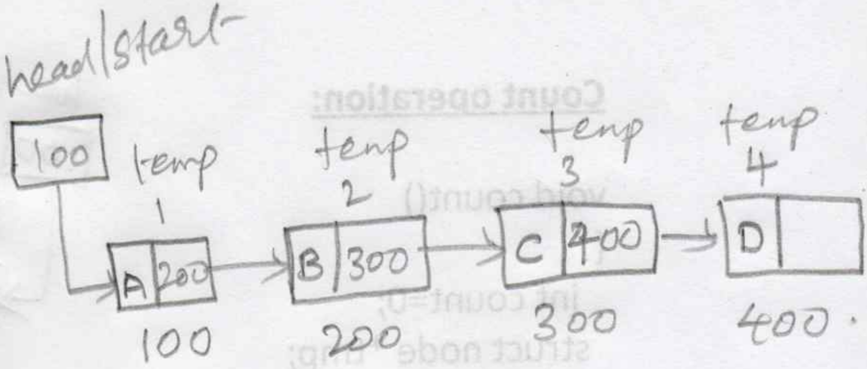
Display operation:



Ans: 5 10 20

```
    void displayList()
    {
      struct node *tmp;
     temp=head->next;
      if(head == NULL)
      {
        printf(" List is empty.");
      }
     else
      {
       while(temp->next != NULL)
        {
          printf("temp->data”);     // prints the data of current node
          temp = temp->next;      // advances the position of current node
        }
      }
    }
```

Search operation:

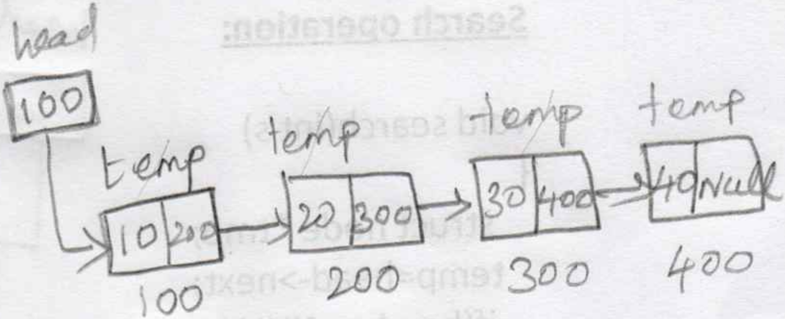


Searchkey = C

C=C – element present

```
void search(int s)
{
  struct node *tmp;
 temp=head->next;
  if(head == NULL)
  {
    printf(" List is empty.");
  }
  else
 {
 while(temp->next!=null&&temp->data=searchkey)
 {
 temp=temp->next;
 }
 printf("element is present");
 else
 {
 printf("not present");
 }
```

Count operation:



Count = 4

```
 void count()
 {
   int count=0;
  struct node *tmp;
   temp=head->next;
    if(head == NULL)
    {
      printf(" List is empty.");
    }
     else
  {
  while(temp->next!=NULL)
  {
  count++;
  temp=temp->next;
  }
   printf(“display count”);
}
}
```