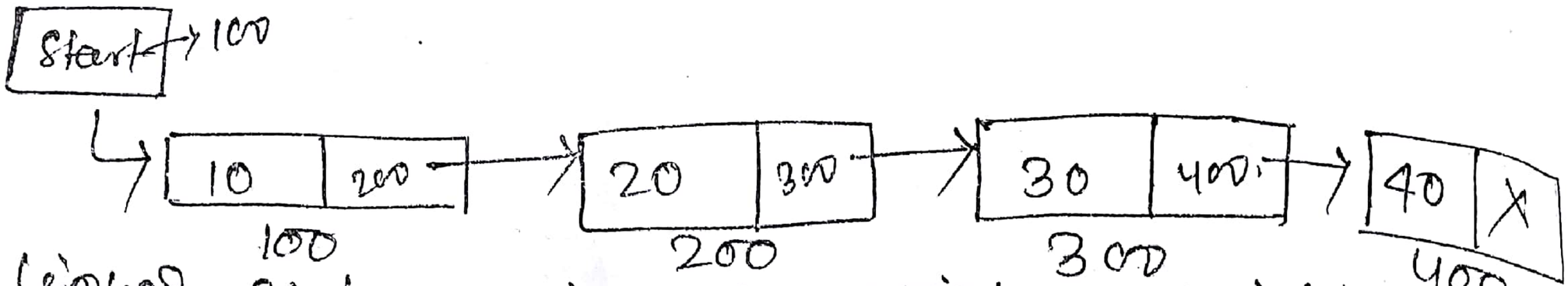
LINKED LIST

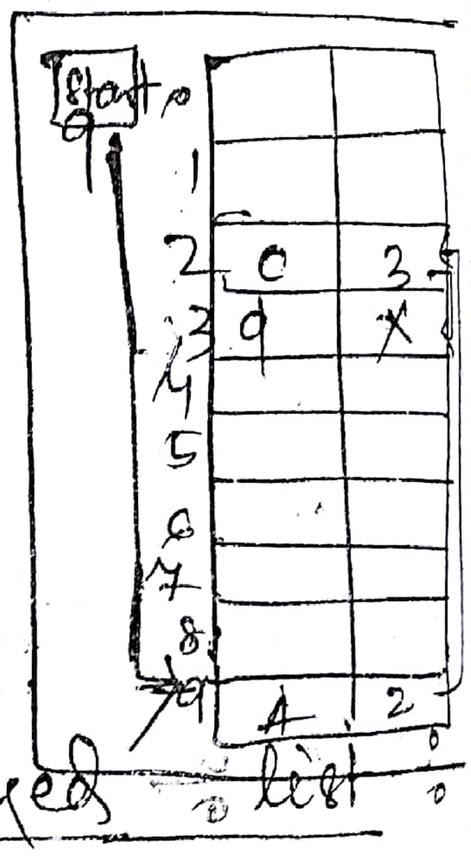
> Linked list is a linear collection of data elements
> These data elements are called nodes
> Linked list is a data structure which in turn can be used to implement other data structure like tree, stack, queue etc.
> A linked list can be perceived as a sequence of nodes in which each node contains one or more data fields and a pointer to the next node.



> Linked list contains a pointer variable start which is a external pointer stores the address of the first node in the list, after that all the nodes have 2 parts
(1) First part of the node contains the data that may include a simple data type or an array or a structure.
(2) Second part of the node contains a pointer to the next node or address of the next node in sequence.
> As last node has no next node connected to it, so it will store a special value called NULL or -1.
> if Start = NULL, then linked list is empty
> It used dynamic Memory allocation.

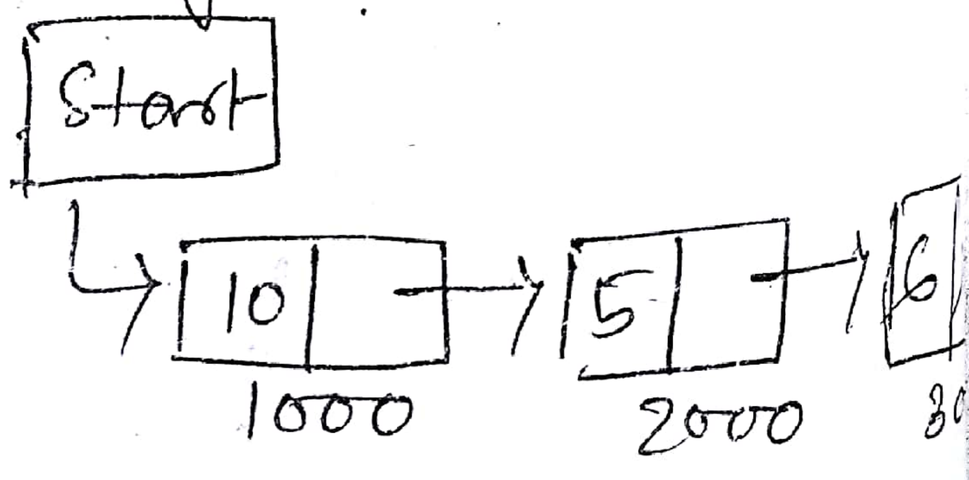
Types of linked list —
1) Single linked list 2) circular single linked list
3) double linked list 4) circular double linked list.

Single linked list —
> It is dynamic - linear data structure in which node of list contains 2 parts i.e data part & a pointer field which contains address of next node



1) Structure of a Node of Single linked list

```
typedef struct NODE
{
    int info;
    struct NODE *next;
}node;
```



typedef is a keyword is used to rename the old type with new name.

> By using typedef we can use "node" in place of struct NODE data-type.
> so in this case node is new name of old datatype ie struct NODE.

2) creation of Empty linked list —:

```
node *start = NULL;
```

3) operation in a linked list.
   i) creation ii) insertion iii) deletion iv) searching v) traversing etc.

i) creation of node —:

```
void create()
{ node *ptr;
  ptr = (node*) malloc(sizeof(node));
  printf("Enter the value");
  scanf("%d", &ptr->info);
  if(start == NULL)
  { ptr->next = NULL;
    start = ptr;
  }
  else { loc = start;
         while(loc->next != NULL)
           loc = loc->next;
         loc->next = ptr;
         ptr->next = NULL;
  }
```

ii) Insertion in single linked list —:
   i) insertion of node at begin
   ii) insertion of node at end
   iii) insertion of node at specific pos.

i) Insertion of node at begin —:

'C' code:

```
void sl_ins_beg()
{
node *ptr;
ptr = (node *) malloc(sizeof(node));
if(ptr == NULL)
  {printf("Memory Not allocated");
   getch();
   exit(0);
  }
```

```
else
{ printf("Enter element");
  scanf("%d", &ptr->info);
  ptr->next = start;
  start = ptr;
}
}
```

Algo :-

sl_ins_beg(start)

1) Allocate memory dynamically using malloc
   ptr = (node *) malloc (sizeof(node));

2) [check for memory full?]
   if ptr = NULL
   write Memory full

3) else
   Enter value or element

4) set ptr->next = start

5) set start = ptr
   [end of if structure]

6) EXIT.

ii) Insertion at end :-

'C' code:

```
void sl_ins_end(...)
{ node *ptr, *loc;
  ptr = (node *) malloc (sizeof(node));
  if(ptr==NULL)
  { printf("Memory full");
    getche();
    exit(0);
  }
  else
  { printf("Enter value of element");
    scanf("%d", &ptr->info)
    ptr->next = NULL;
```

```
if( start==NULL)
 {ptr->next=start;
  start=ptr;
 }
else
   { loc=start;
    while( loc->next!=NULL)
       { loc=loc->next;
       }
     loc->next=ptr;
   }
  }
 }
```

Algo :-

sl-ins-end(start)

1) Allocate memory dynamically by using malloc function.
   ptr=(node *) malloc(sizeof(node));
2) [check for memory full?]
   if ptr=NULL
   write memory full
3) else
   Enter value of element
4) set ptr->next=NULL
5) if start=NULL
   set ptr->next=start
   set start=ptr
6) else
   set loc=start
7) Repeat step 8 till loc->next!=NULL
8) set loc=loc->next
9) set loc->next=ptr.
   [end of if structure]
   [end of if structure]
10) EXIT.

3

iii) Insertion of node at specific position

'C' code :

```
void SL_ins_spc()
{
 node *ptr, *loc;
 int i, pos;
 printf("Enter position of node");
 scanf("%d", &pos);
 ptr=(node *) malloc(sizeof(node));
 if(ptr==NULL)
  {printf("Memory full");
   getch();
   exit(0);
  }
 else
  {printf("Enter value of node");
   scanf("%d", &ptr->info);
   if(pos==0)
    { ptr->next = start;
      start=ptr;
    }
   else
    { loc=start;
      for(i=0; i<(pos-1); i++)
       {
        loc = loc->next;
        if(loc==NULL)
         {
          pf("Location Not found");
          getch();
          exit(0);
         }
       } ptr->next = loc->next;
       loc->next=ptr;
    }
  }
}
```

Algo :

sl_ins_spc(Start)

1) Enter position of node
2) Allocate memory dynamically using malloc function
   ptr=(node *) malloc (sizeof(node));
3) [check for memory full?]
   if ptr = NULL
   write memory full.
4) else
   .Enter the value of node from keyboard
5) if pos = 0
   set ptr→next = start
   set start = ptr
6) else
   .set loc = start & set i = 0
   Repeat step 7 for a till i < pos-1
7) set loc = loc→next
   if loc = NULL
8) write location Not found
9) set i = i+1
   [end of for loop]
10) set ptr→next = loc→next
11) set loc→next = ptr
    [end of if structure]
    [end of if structure]
12) Exit

5

'c' function

Q) WAP to insert a node after an element:-

```
void sl_ins_aft()
{
int aftele,
node *ptr, *loc;
printf("Enter aftere element");
scanf("%d", &aftele);
ptr = (node *) malloc( sizeof(node));
if(ptr==NULL)
{printf("Memory Not allocated");
    getche();
    exit(0);
}
else
{
printf("Enter info of node");
sf("%d", &ptr->info);
loc = start;
while((loc != NULL) && (loc->info != aftele))
  {  loc = loc->next;
  }
if(loc==NULL)
  printf("Location Not found");
else
  {ptr->next = loc->next;
   loc->next = ptr;
  }
}
}
```

'c' function

Q) WAP to insert a node before an element:-

```
void sl_ins_bef()
{
int befele;
 node *ptr,*loc, *bef;
  printf("Enter before ele");
   scanf ("%d", &befele);
```

```
ptr = (node *) malloc (sizeof(node));
if(ptr == NULL)
{ pf("Memory Not allocated");
  getch();
  exit(0);
}
else
{
  pf("Enter info node");
  sf("%d", &ptr->info);
  loc = start;
while((loc != NULL) && (loc->info != befele))
  { bef = loc;
    loc = loc->next;
  }
if(loc == NULL)
  printf("Location Not found");
else
{ if(start->info == befele)
  { ptr->next = start;
    start = ptr;
  }
else
  { ptr->next = bef->next;
    bef->next = ptr;
  }
}
}
```

WAP for traverse an element :-

```
void traverse()
{ node *ptr;
  ptr = start;
  while(ptr != NULL)
  {
    pf("%d", ptr->info);
    ptr = ptr->next;
  }
}
```

Deletion of node in linked list:-

i) Deletion of node at begin
ii) Deletion of node at end
iii) Deletion of node at specific pos?

(i) Deletion of node at begin:-

'c' code:-

```
void sl-del-beg( , )
{
node *ptr;
 ptr = start;
  if (start == NULL)
    {printf("List is empty");
     getche();
     exit(0);
    }
  else
    {start = ptr->next;

   free(ptr);
    }
}
```

Algo

sl-del-beg(start)

1) Set ptr = start
2) [check for empty list?]
   if start = NULL
   write list is empty
3) else
   set start = ptr->next
4) deallocate memory by using free(ptr)
   [end of if structure]
5) EXIT.

Deletion from end :-

(c) code :

```
void sl_del_end()
{
  node *ptr, *loc;
  if(start == NULL)
  {
    printf("list is empty");
    getch();
    exit(0);
  }
  else
  {
    ptr = start;
    if(start->next == NULL)
    {
      start = NULL;
    }
    else
    {
      while(ptr->next != NULL)
      {
        loc = ptr;
        ptr = ptr->next;
      }
      loc->next = NULL;
    }
    free(ptr);
  }
}
```

Algo :

sl_del_end(start)

1) [Check for empty list]
   if start = NULL
   write list is empty

2) else
   set ptr = start

3) if start->next = NULL
   set start = NULL

4) else
   Repeat step 5 & 6 till ptr->next != NULL

9

5) set loc = ptr

6) set ptr = ptr->next
end of while

7) set loc->next = NULL
[end of if structure]

8) deallocate memory by using free(ptr)
[end of if structure]

9) EXIT.

iii) deletion at specific - location :-

```
void sl_del_spc()
{
node *ptr, *loc;
int pos, i;
pf("Enter position of element");
sf("%d", &pos);
ptr = start;
if(pos == 0)
{
start = ptr->next;
}
else
{
for(i=0; i<pos ; i++)
{ loc = ptr;
ptr = ptr->next;
if(ptr == NULL)
{printf("Location Not found");
getch();
exit(0);
}
}
loc->next = ptr->next;
free(ptr);
```

Algorithm:

sl_del_spc(start)

1) Enter position of element from keyboard
2) set ptr = start
3) if pos = 0
   set start = ptr → next
4) else
   set i = 0
   repeat step 5 to 8 till i < pos
5) set loc = ptr
6) set ptr = ptr → next
7) if ptr = NULL
   write location Not found
8) set i = i + 1
   [end of for loop]
9) set loc → next = ptr → next
   [end of if structure]
10) deallocate memory by using free(ptr)
11) EXIT.

Write a 'c' function to delete an element in a linked list after an element.

```
void sl_del_aft()
{
node *ptr
int after;
ptr = start;
pf("Enter after element where to delete");
sf("%d", &after);
if(start == NULL)
  {
  pf("underflow");
    getch();
    exit(0);
```

```
else {
    while((ptr != NULL) && (ptr->info != after)
    {
    ptr = ptr->next;
    }
    if(ptr == NULL)
    {
    printf("location not found");
    getch();
    exit(0);
    }
    else {
    loc = ptr;
    ptr = ptr->next;
    loc->next = ptr->next;
    free(ptr);
    }
}
}
```

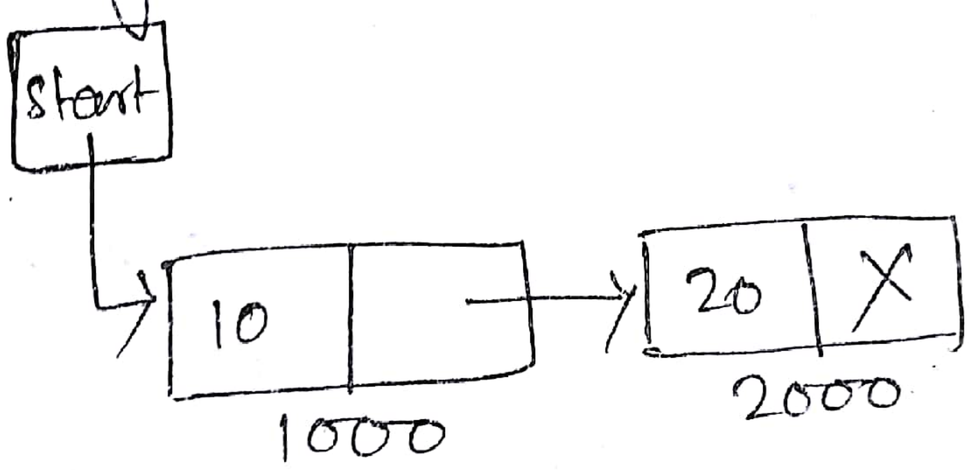
Q. WAP to write 'c' func to search element in a linked list

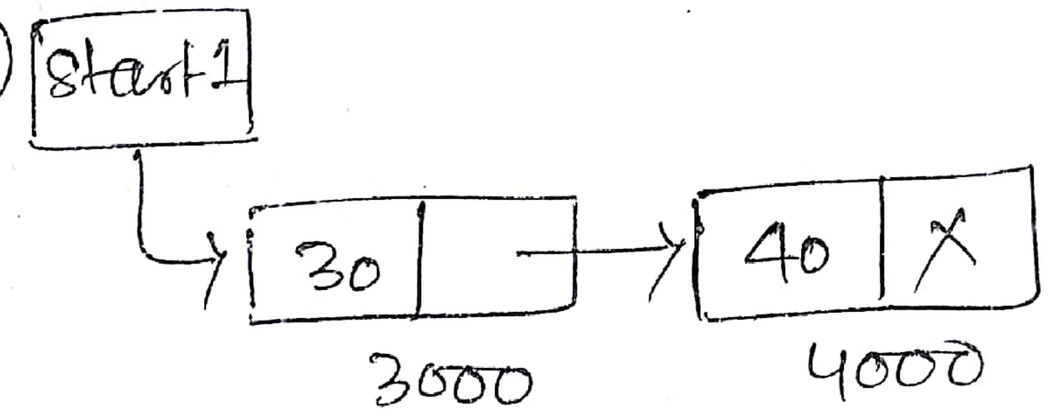
```
void search()
{
int ele, pos=0;
node *loc;
loc = start;
printf("Enter element");
if("x:d", &ele);
```

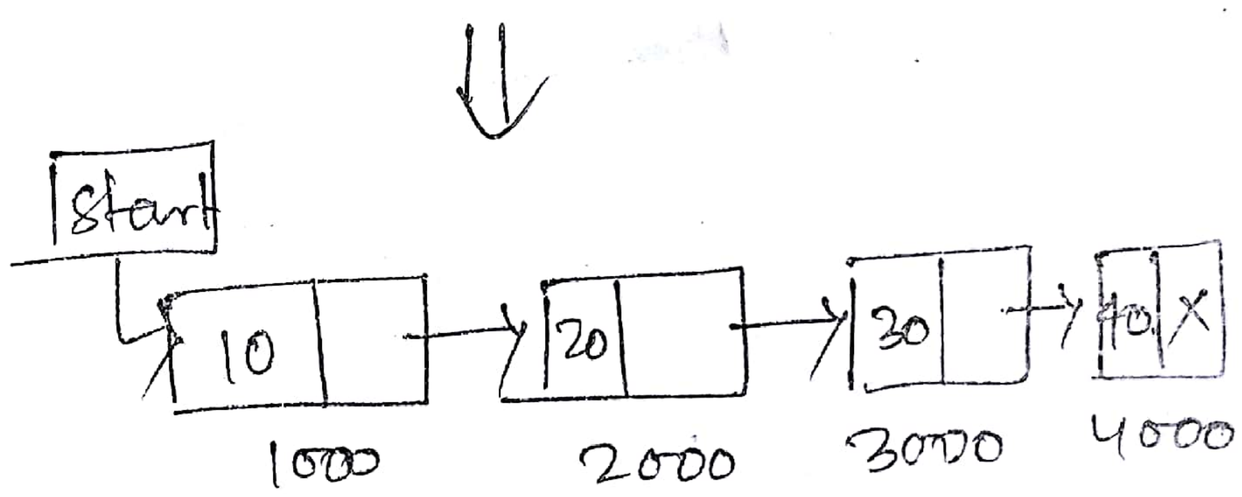
```
while((loc != NULL) && (loc->info != sele))
    {
      loc = loc->next;
            pos++;
    }
if(loc == NULL)
  {
   printf("element Not found");
  }
else
  {printf("element is at %d", pos);
  }
}
```

Write 'C' function to merge a linked list.

```
void merge(node *start,
           node *start1)
  {
   node *ptr;
   ptr = start;
   while(ptr->next != NULL)
     {
       ptr = ptr->next;
     }
   ptr->next = start1;
  }
```



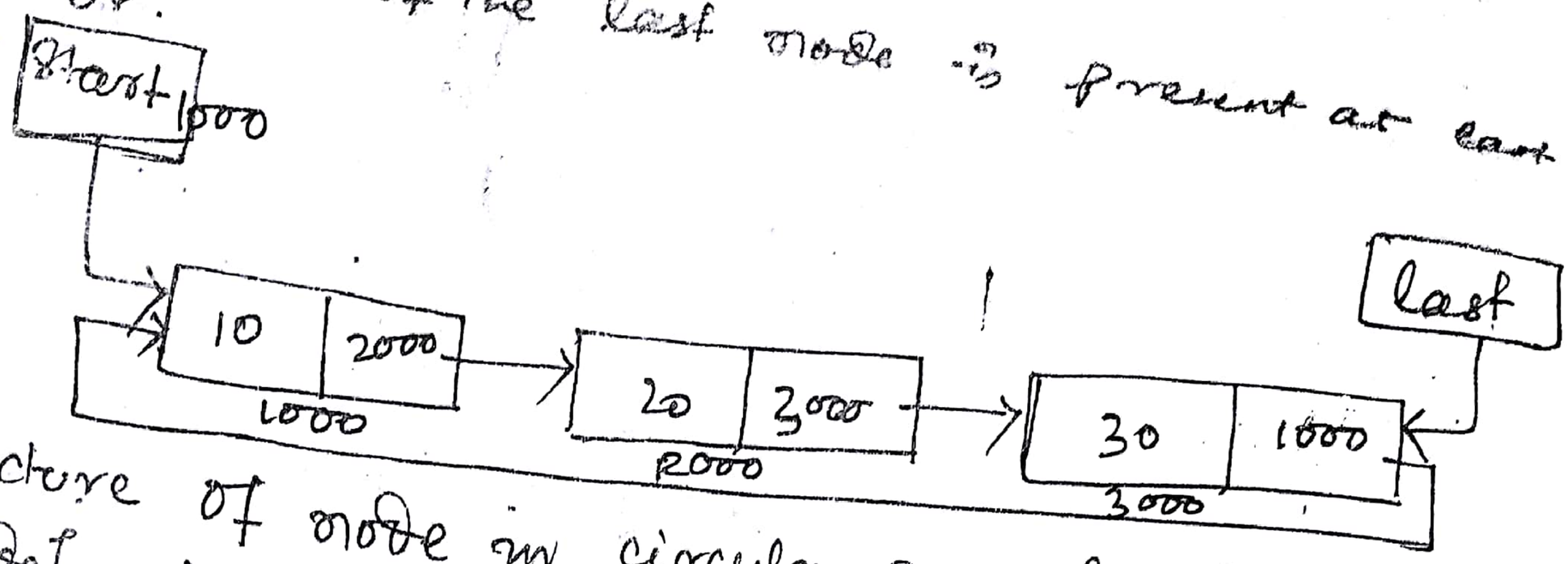




Q) Write 'C' function to split a linked list after an element.

Q) Write 'C' function to find minimum and maximum element in linked list & swap the values.

... the last node contains the address of the last node ... present at last extra pointer.



Structure of node in circular ~~queue~~ linked list :-

```
typedef struct NODE
{
  int info;
  struct NODE *next;
}node;
node *start, *last;
start = NULL; (empty list creation)
last = NULL;
```

Insertion in circular linked list :-

1) Insertion at begin :-

```
void cins_beg()
{
  node *ptr;
  ptr = (node *) malloc(sizeof(node));
  if(ptr == NULL)
  {
    pf("Memory Not allocated\n");
    getch();
    exit(0);
  }
else
{
  Pf("Enter element in the node");
  sf("%d", &ptr->info);
  if(start == NULL)
    start = ptr;
```

Algo :-

cins_beg(start)

1) Allocate memory dynamically for node by using malloc & assign starting address to ptr.

Ptr = (node *) malloc(sizeof(node))

2) if ptr = NULL
write Memory full

3) else
Enter value of node info field from keyboard.

```
last = ptr;
ptr -> next = ptr;
}
else
{ ptr -> next = start;
  start = ptr;
  last -> next = ptr;
}}}
```

2) Insertion at end :-

(c) code

```
void cins_end()
{ node *ptr;
 ptr = (node *) malloc (sizeof(node));
 if (ptr == NULL)
 {
 printf("Memory Not allocated");
  getch();
  exit(0);
 }
else
{
 printf("Enter element");
 scanf("%d", &ptr->info);
 if (start == NULL)
 { start = ptr;
   last = ptr;
   ptr -> next = ptr;
 }
else
 { last -> next = ptr;
  loc = start;
   ptr -> next = start;
   last = ptr;
 }
}}}
```

Algo

```
cins-end (start)
1) Allocate Memory dynamically
   by using malloc
   ptr = (node *) malloc (sizeof(node));
2) if ptr = NULL
   write Memory full
   exit

3) else
   Enter ptr -> info from
   keyboards.
4) if start = NULL
   set start = ptr
   set last = ptr
5) set last = ptr
6) set ptr -> next = ptr
7) else
   set last -> next = ptr
8) set ptr -> next = start
9) set last = ptr
   [end of if structure]
[end of if structure]
10) EXIT.
```

Deletion in circular linked list :-

1) Deletion at begin :-

```
void cdel_beg()
{
 node *ptr;
 if (start == NULL)
 { printf("No element");
```

Algo :-

```
cdel_beg (&start, last)
1) [check for node
   present or not]
   if start = NULL
      write No element.
```

```
        getch();
        exit(0);
    }
    else
    {
        ptr = start;
        pf("Element deleted is %d", ptr->info);
        if( last == start && start->next == start)
        {
            start = NULL;
            last = NULL;
        }
        else
        {
            start = ptr->next
            last->next = start;
        }
    free(ptr);
    }
}
```

2) deletion from end :-

```
void edel_end()
{
    node *ptr, *loc;
    if (start == NULL)
    {
        pf("List is empty");
        getch();
        exit(0);
    }
    else
    {   ptr = start;
        if (start->next == ptr)
        {
            pf("Element deleted %d", ptr->info);
            start = NULL;
            last = NULL;
        }
        else {
            while(ptr->next != start)
            {   loc = ptr;
                ptr = ptr->next;
```

2) else
   set ptr = start
   write ptr->info
3) if last = start
   set start = NULL
   set last = NULL
4) set last = NULL
5) else
   set start = ptr->next
   set last->next = start
6) [end of if structure]
7) deallocate memory by using
   free(ptr)
8) [end of if structure]
8) Exit

```
edel_end(start, last)
```

1) [check for is list empty ?]
   if start = NULL
   write list is empty
2) else
   set ptr = start
3) if start->next = ptr
   write ptr->info
4) set start = NULL
5) last = NULL
6) else

Repeat step 7 & 8 till ptr->next != start

7) set loc = ptr
8) set ptr = ptr->next
   ends of while-loop.
9) write element deleted is
   ptr->info

```
pf("Element deleted is %d", ptr->info);
 loc->next = start;
  last = loc;
 }
 free(ptr);

}
}
```

10) set loc->next = start
11) set last = loc
[end of if structure]
12) deallocate memory by using free(ptr)
[end of if structure]
13) Exit.

Q) Write a program insert