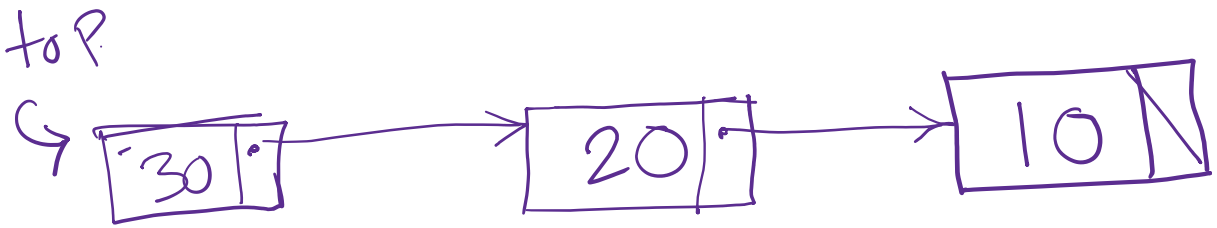
Implementation of stack using linked linear list:

Stack has LIFO property

e.g push 10 , push 20 , push 30



```
struct node {
        int info;
        struct node *link;
        };

struct node *top, *ptr, *temp;
int main() {
   top=null;
    :

}
void push() {
        int x;
```

```
printf("Enter Information Part: ");
scanf("%d",&x);
ptr=(struct node *) malloc(sizeof(struct node));
ptr->info=x;
```

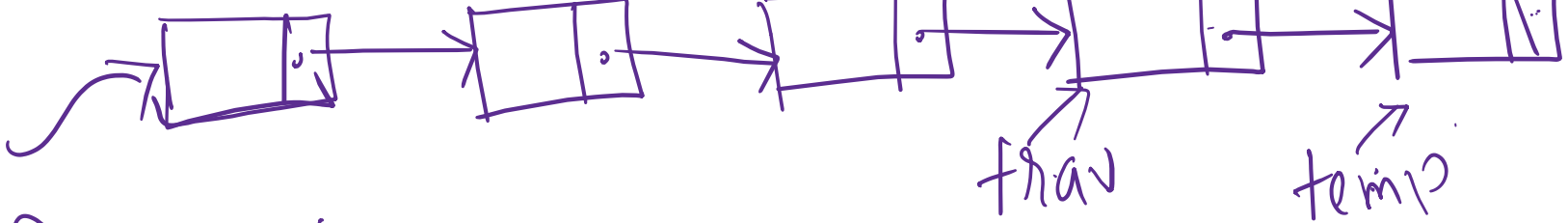
O(1)

top=null

```
if (top==null)
    { top= ptr; top->link= null; }
else { ptr->link= top; top= ptr; }
}
```

```
void pop() {
    if (top == null)
    {
        printf("Stack is empty");
        return;
    }
    temp = top;
    top = top -> link;
    temp -> link = null;
    free(temp);
```

O(1)



N

(N-1)

top

```
trav = top;
while (trav -> link != temp)
    trav = trav -> link;
```

O(N)

O(N)

O(N)

Implementation of queue using linked list:

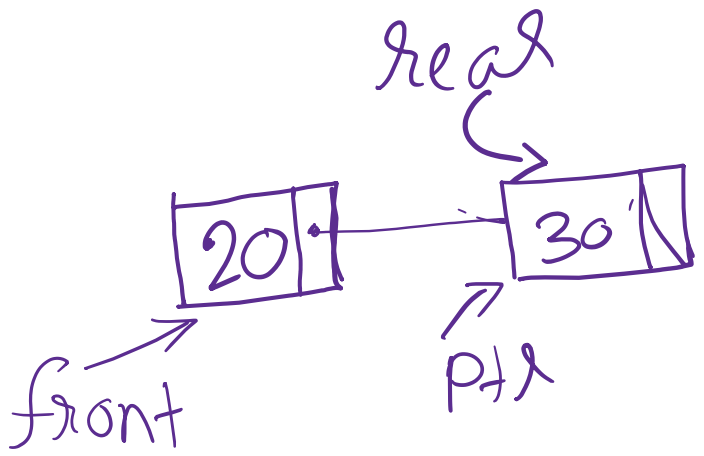
Two pointer variables: front, rear

20
30

```
struct node *front, *rear;
```

temp
=null

```
front = null;
rear = null;
```

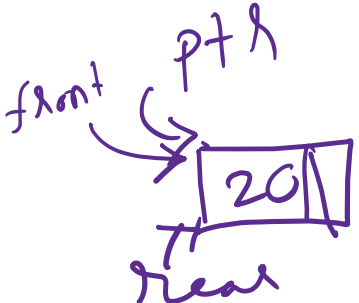


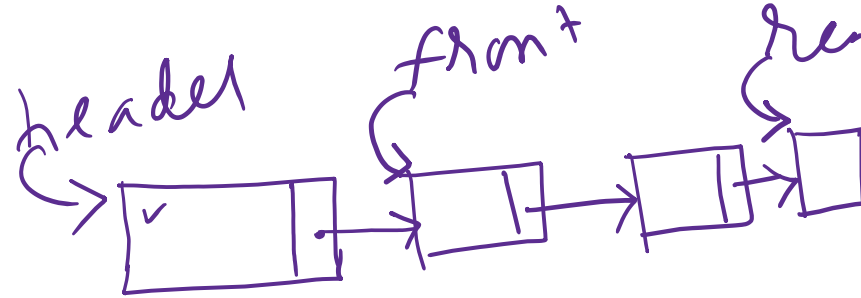
Insert 10
Insert 20
Insert 30

```
rear->link = ptr;
rear = ptr;
```

```
struct node {
        int info;
        struct node *link;
    };

struct node *front, *rear, *temp, *ptr;
int queue_size;
int main() {
    front = null;
    rear = null;
    enqueue();
    enqueue();
                    dequeue();
```

```
void enqueue() {
    int x;
    printf("Enter element to insert: ");
    scanf("%d", &x);
    ptr = (struct node *) malloc(sizeof(struct node));
    ptr->info = x;
    ptr->link = null;
    if (rear == null) {
        front = ptr;
        rear = ptr;
    }
```





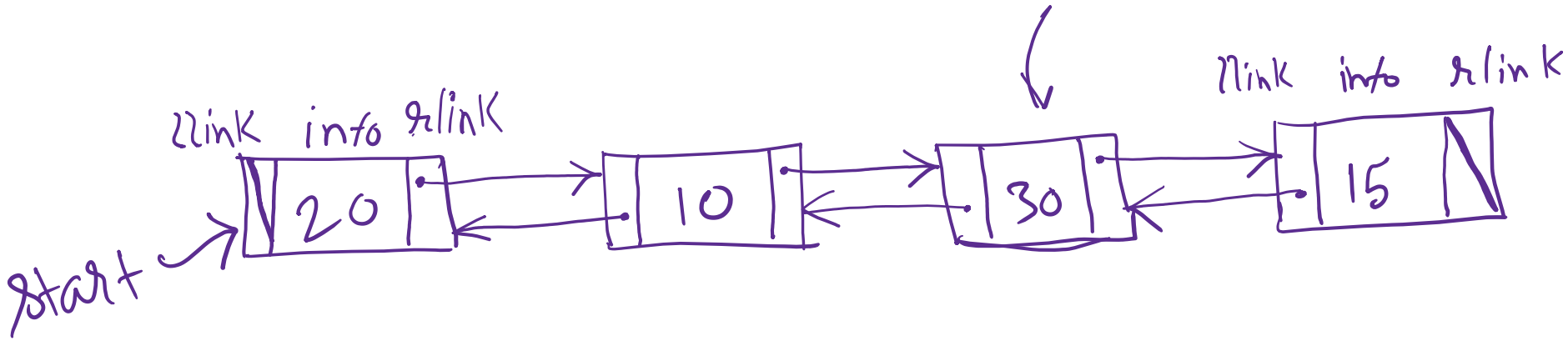
```
else {
    rear -> link = ptr;
    rear = ptr;
   }

void dequeue() {
    if (front==null) {printf("Queue underflow"); return;} else {
     temp = front;
     front = front -> link;
      temp -> link = null;
      free(temp);}
   }
```

```
void display() {
    temp = front;
    while ( temp != null ) {
    printf("%d\n", temp->info);
    temp = temp->link;
    }
```

10
20

```
for(temp = front; temp != null; temp = temp
    printf("%d\n", temp->info);
```

Doubly Linked Linear List



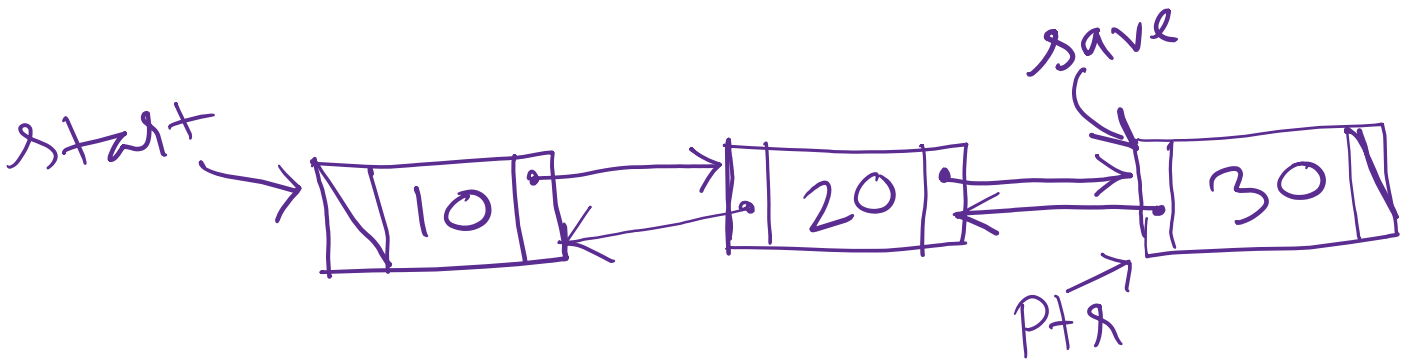

```
struct node {
        int info;
        struct node *rlink;
        struct node *llink;
        }
```

Creation of a doubly linked list

10 20 30 15

Initially, start = null

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



ptr → info = 10;

ptr → llink = null;

ptr → rlink = null;

start = ptr;

save = ptr;

ptr → info = 20;

save → rlink = ptr;

ptr → llink = save;

save = ptr;

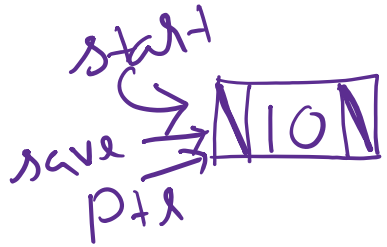
ptr → info = 30;

Implementation:

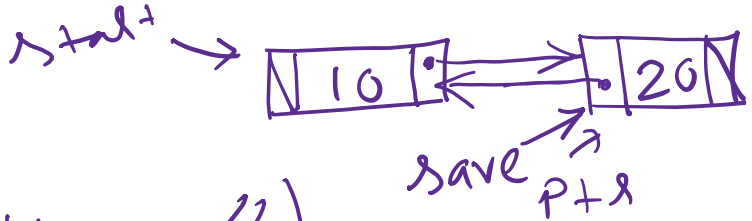
```
struct node {
        int info;
        struct node *rlink;
        struct node *llink;
    };
struct node *start;
```

```
int main() {
    .
    .
    .
  create();
    .
    .
    .
  return 0;
}

Void create() {
    int element, choice = 1;
    struct node *ptr, *save;
```

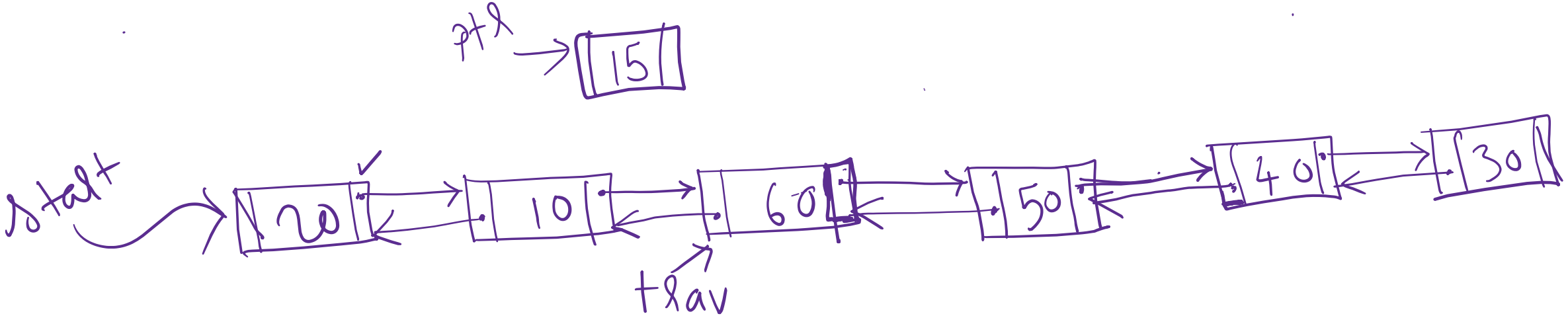
```
printf("Enter information: ");
scanf("%d", &element);
ptr=(struct node *) malloc(sizeof(struct node));
ptr → info = element;
ptr → rlink = null;
ptr → llink = null;
start = ptr;
save = ptr;
printf("Do you want to continue: 1-Yes: 0-No");
scanf("%d", &choice);
```



```
while (choice) {
    printf("Enter information: ");
    scanf("%d", &element);
    ptr = (struct node *) malloc (sizeof (struct node));
    ptr -> info = element;
    ptr -> rlink = null;
    ptr -> llink = null;
    save -> rlink = ptr;
    ptr -> llink = save;
    save = ptr;
```



```
printf("Do you want to continue?.");
scanf("%d",&choice);

}
```



Insert 50 after 60

```
void insert_after( ) {
    int x, y;
    struct node *trav;
    printf("After which? ");
    scanf("%d", &x);
```

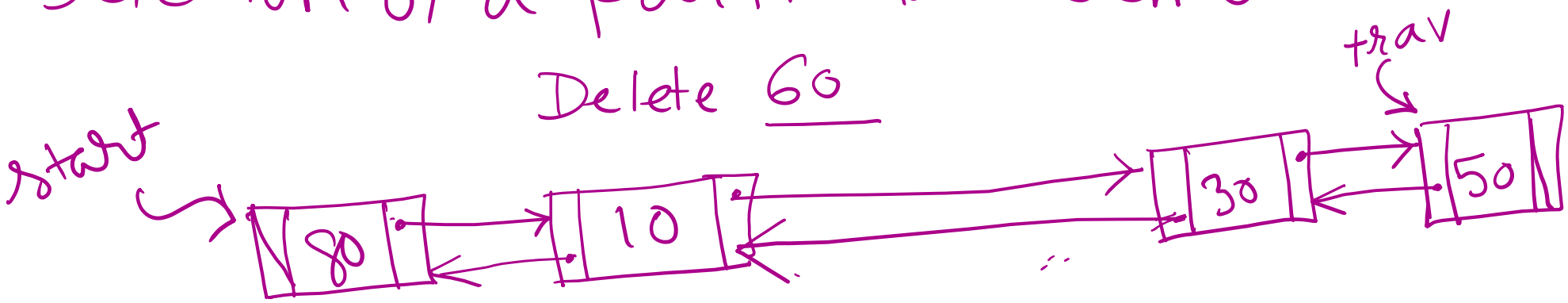
x = 60

y = 50

```
printf("New element?");
scanf("%d", &y);
trav = start;
while (trav->info != x)
        trav = trav->rlink;

ptr = (struct node *) malloc (sizeof (struct node));
ptr->rlink = trav->rlink;
ptr->llink = trav;
trav->rlink->llink = ptr; trav->rlink = ptr;
```

Doubly Linked Linear List:

- Creation
- Insertion of element after a given element
- Deletion of a particular element.

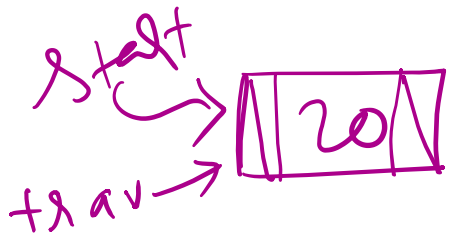


trav->llink->rlink = trav->rl

trav->rlink->llink = trav->ll

trav->rlink = null;
trav->llink = null;
free(trav); trav = null;

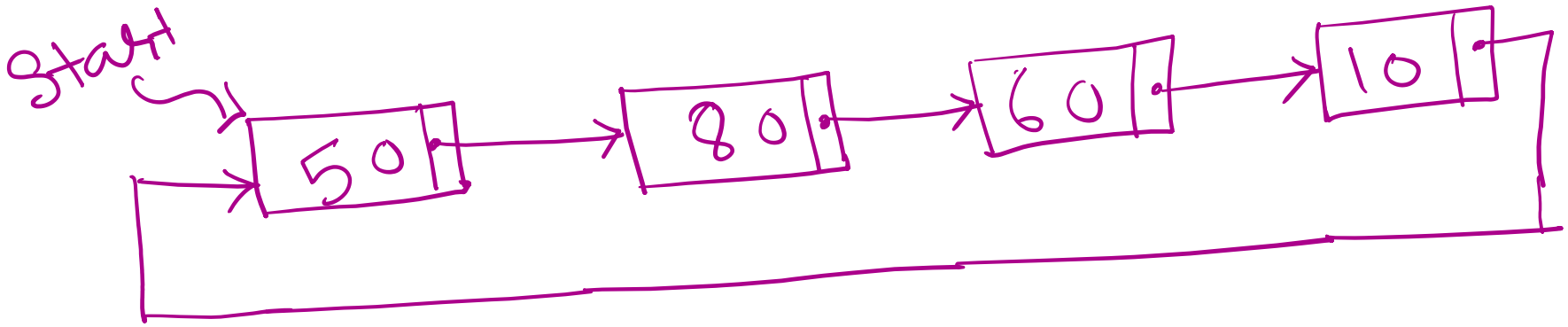
```
if (start == null) {

}

else
{ if ( ) { }
} else { }
```




```
trav = start;
while (trav -> info != 2
    trav = trav -> rlink
```

Singly Linked Circular List :

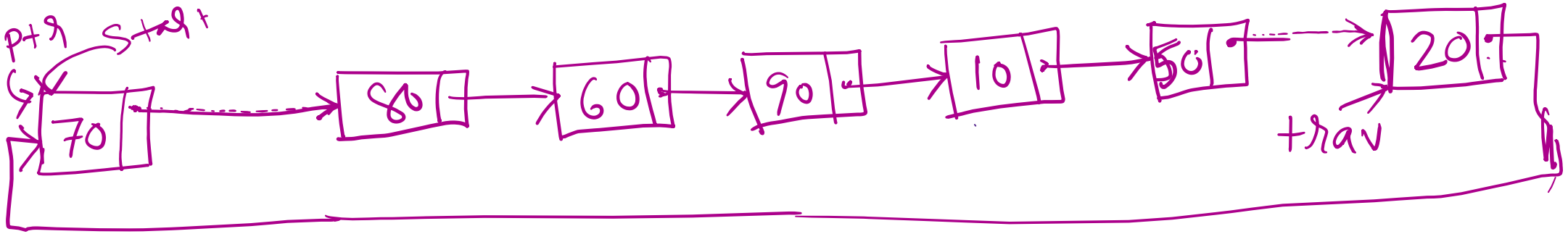
(Circular Singly Linked List)




```
trav = start;
printf("%d\n", trav->info);
trav = trav->link;
while (trav != start) {
    printf("%d\n", trav->info);
    trav = trav->link;
}
```

```
50
80
60
10
```

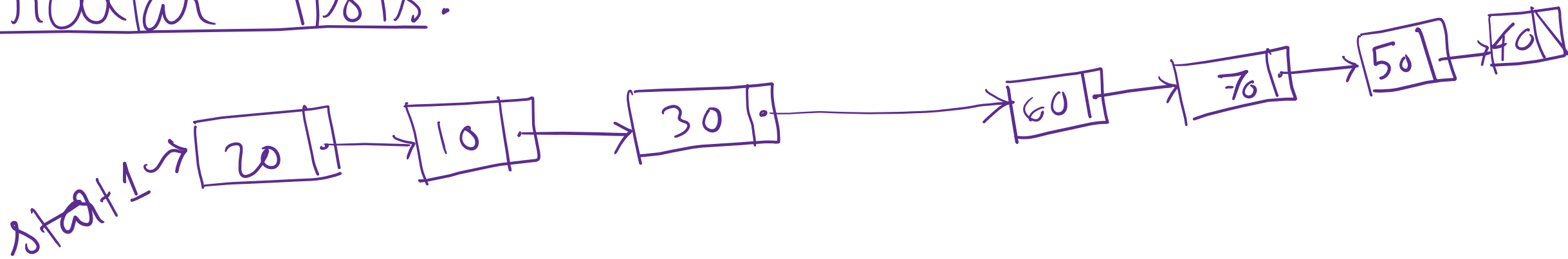
Insertion at the end of the list:



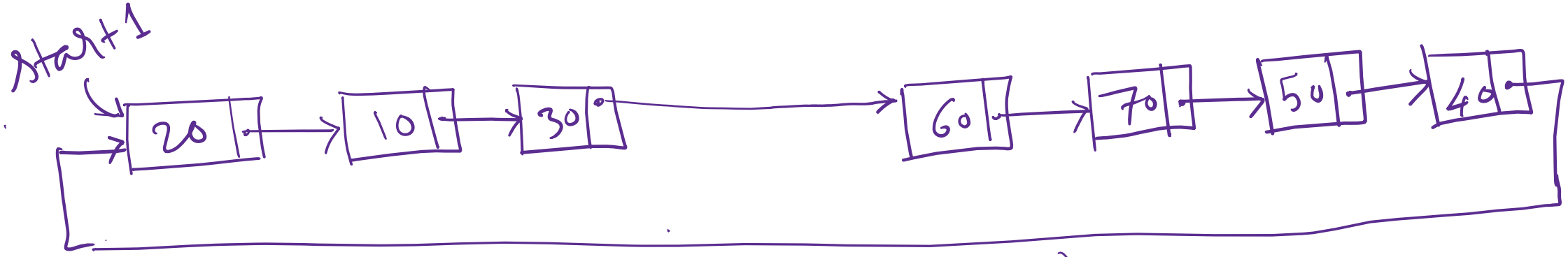
Insertion at the beginning of list:

* Create a singly linked circular list
  i) always by inserting at end
  ii) always by inserting at the beginning

Concatenation of two singly linked non circular lists:




```
trav = start1;
while (trav -> link != null)
    trav = trav -> link;

trav -> link = start2;
trav = start2 = null;
```



$O(n)$

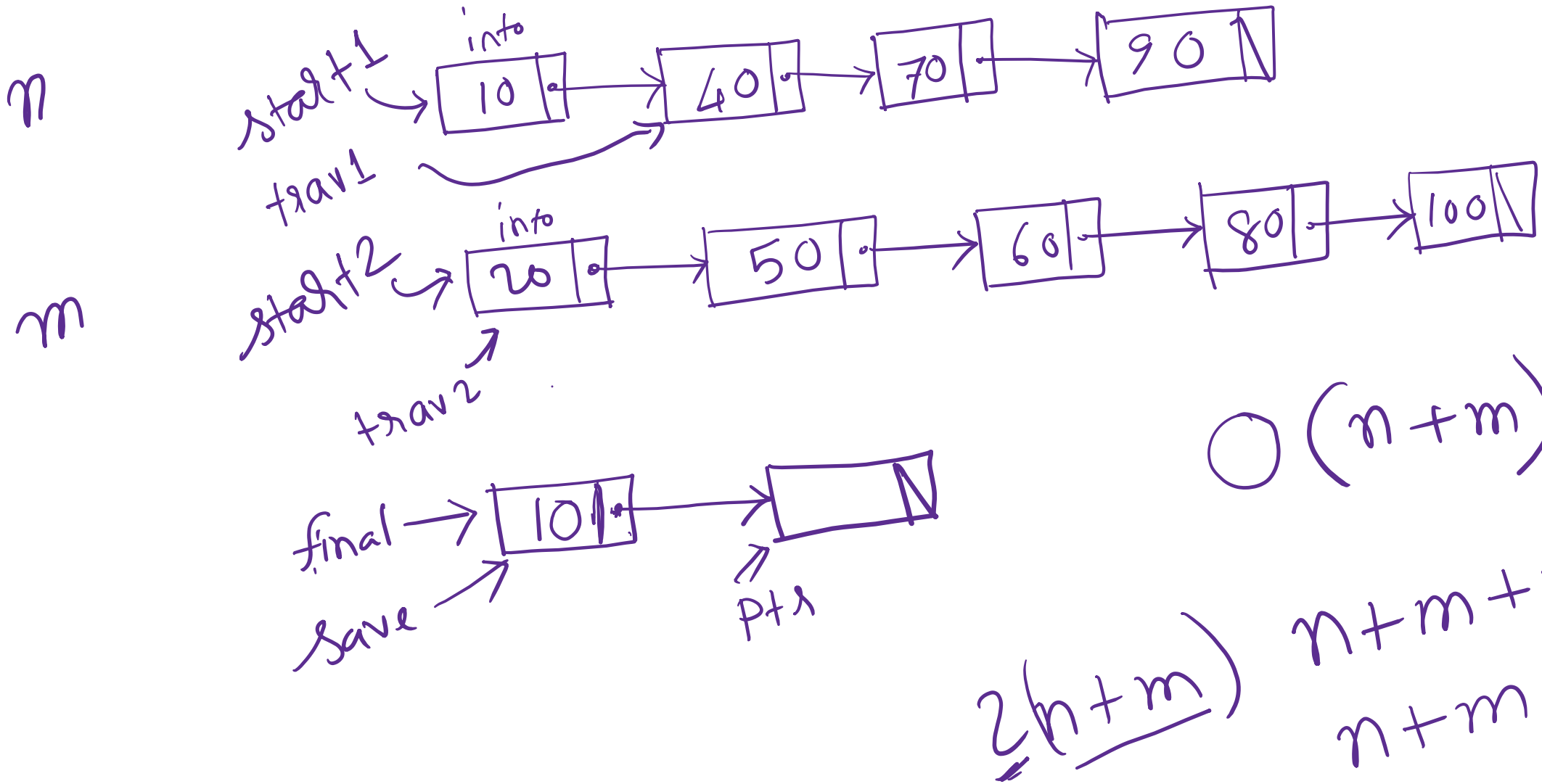
```
trav1 = start1;
while (1) {
    if (trav1 -> link == start1)
        break;
    else
        trav1 = trav1 -> link;
}
```

$O(m)$

```
trav2 = start2;
while (1) {
    if (trav2 -> link == s
        break;
    else
        trav2 = trav2 ->
}
```

```
trav1 -> link = start2; trav2 -> link = start1; trav1 = trav2 =
start1 = null;
```

Merging of two sorted linked lists:



$O(n+m)$

$2(n+m)$

$n+m+n+m$

$n+m$

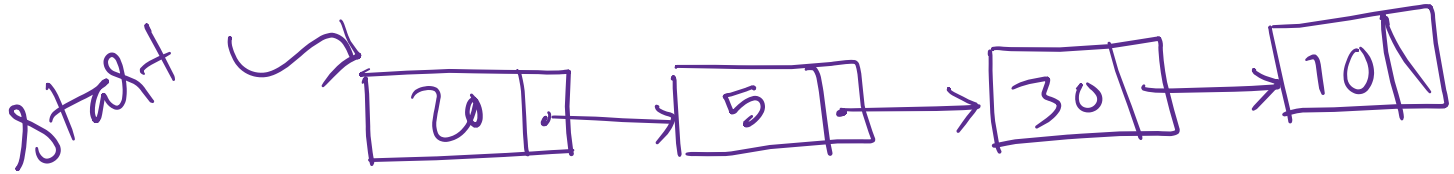
```
trav1 = start1;
trav2 = start2;
ptr = (struct node *) malloc(sizeof(struct node));
ptr -> link = null;
if (final == null) final = ptr;
if (trav1 -> info <= trav2 -> info) {
        ptr -> info = trav1 -> info;
        trav1 = trav1 -> link;
    }
else {
        ptr -> info = trav2 -> info;
        trav2 = trav2 -> link;
    }
```

```
save = ptr;
while (trav1 != null && trav2 != null) {
    ptr = (struct node *) malloc (sizeof (struct node));
    ptr -> link = null;
    if (trav1 -> info <= trav2 -> info) {
        ptr -> info = trav1 -> info;
        trav1 = trav1 -> link;
        save -> link = ptr; save = ptr; }
    else {
        ptr -> info = trav2 -> info;
        trav2 = trav2 -> link;
        save -> link = ptr; save = ptr; }
}
```

```
while (trav1 != null) {
    ptr = (struct node *) malloc (sizeof (struct node));
    ptr -> link = null;
    ptr -> info = trav1 -> info;
    trav1 = trav1 -> link;
    save -> link = ptr;
    save = ptr;
}
```

```
while (trav2 != null) {
    ptr = (struct node *) malloc(sizeof(struct node));
    ptr->link = null;
    ptr->info = trav2->info;
    trav2 = trav2->link;
    save->link = ptr;
    save = ptr;
}
```

How to write an algorithm for linked list:



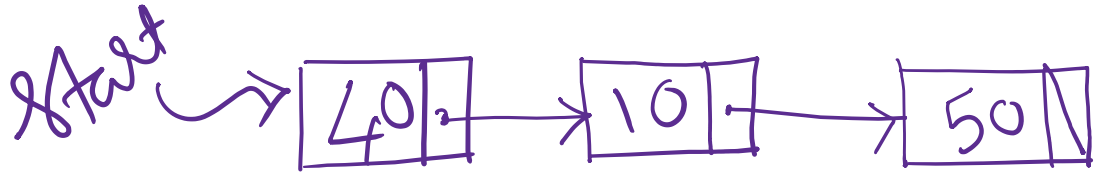
A linked list is already created in memory. The variable 'start' points to the first node of the list. Write an algorithm to display the list.

Algorithm Display (start)

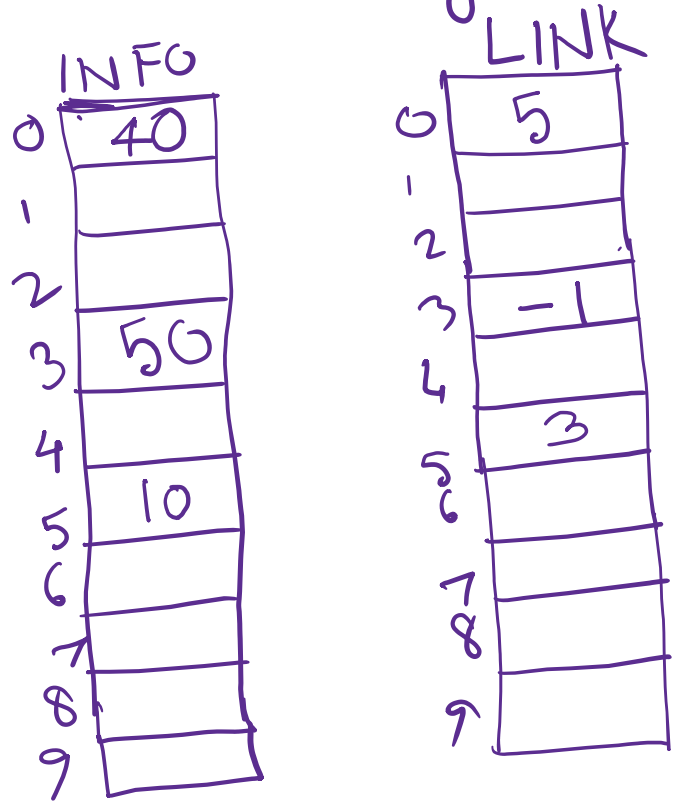
Begin

1. If (start = null) Then
2. Write (`List is empty')
3. Return
4. End If
5. temp ← start
6. Repeat step 7 through 8 while temp <> null
7. Write('The content is:', INFO(temp))
8. temp ← LINK(temp)
9. Return

Simulation of a linked list using array



we take two arrays: INFO and LINK



start = 0

INFO(temp)	INFO(temp)
RLINK(temp)	NEXT(temp)
LLINK(temp)	PREVIOUS(temp)

Implementation of stack using doubly linked list

" " queue " " " "

Non linear data structures:

Graph:

- townships
- towns are connected by roads

Routers