```
struct Point
{
   int x;
   int y;
};
```

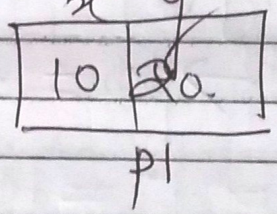
```
typedef struct
{
   int x;
   int y;
} Point;
```

datatype name

```
struct Point p1;   // p1 variable of type struct Point is created at compile time
```

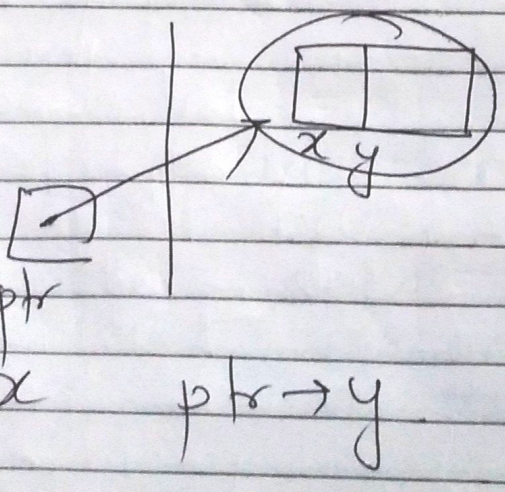
This variable is created inside of a method, & variable is local to that method.

```
p1.x = 10;
p1.y = 20;
```



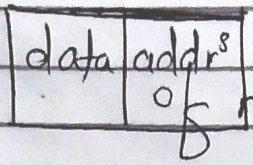
If I want to allocate memory for one point variable dynamically

```
Point *ptr = (Point *) malloc(sizeof(Point));
```

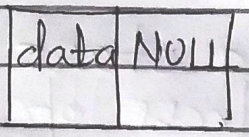


```
struct student
{
    int rollno;
    char name[20];
    float per;
};
student *ptr;
ptr = (student *) malloc
        (sizeof(student));
ptr->rollno=10
strcpy( ptr->name, "Yash Deshmukh");
ptr->per = 96.5;
```

→ Linked List collection of nodes
→ nodes are divided into 2 parts



→ Last node will have NULL in the second half



In java, null reference
In C, it NULL (pointer)

There is a designated pointer variable 'start' pointing to first node of every Linked List. This is required to traverse entire LL

Disadvantage :- for storing single data, we need ~~two~~ factors (wastage of memory)

Advantage: Allocation of memory at runtime.

Creation of Linked List:

LL can be a list of int/double/student If it is of type int then data part will be of type int. If it is of type student then data part will be of type student (you need nested structure)

```
typedef struct
{
    int data;
    .... *next;
} Node;
```

Initially the LL is empty (no nodes are created in memory)

Every LL will have a start pointer containing base address of first node.

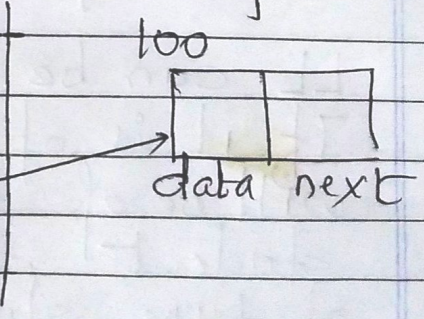
```
Node *start; // start is pointer of type Node
```

case 1: If LL is empty

```
start = NULL;
```

case 2: LL has one node

```
Node *p;
p = (Node *) malloc(sizeof(Node));
p -> data = 10;
p -> next = NULL;
start = p;
```



case 3: 2 nodes in LL

```
p = (Node *) malloc (sizeof (Node));
p->data = 20;
p->next = NULL;
```

// Need to link first node with second node

```
start -> next = p;
```

So, Lets define (declare) structure

```
typedef struct
{
  int data;
  Node *next;
} Node;
```

→ compilation error

```
typedef struct node
{
  int data;
  struct node *next;
} Node;
```

self referential structure

A structure has a member which is a pointer of type itself.