CS & IT ENGINEERING

Data structure & Programming

Linked List

Lec- 03

By- Pankaj Sharma sir

Linked List -3

Insertion

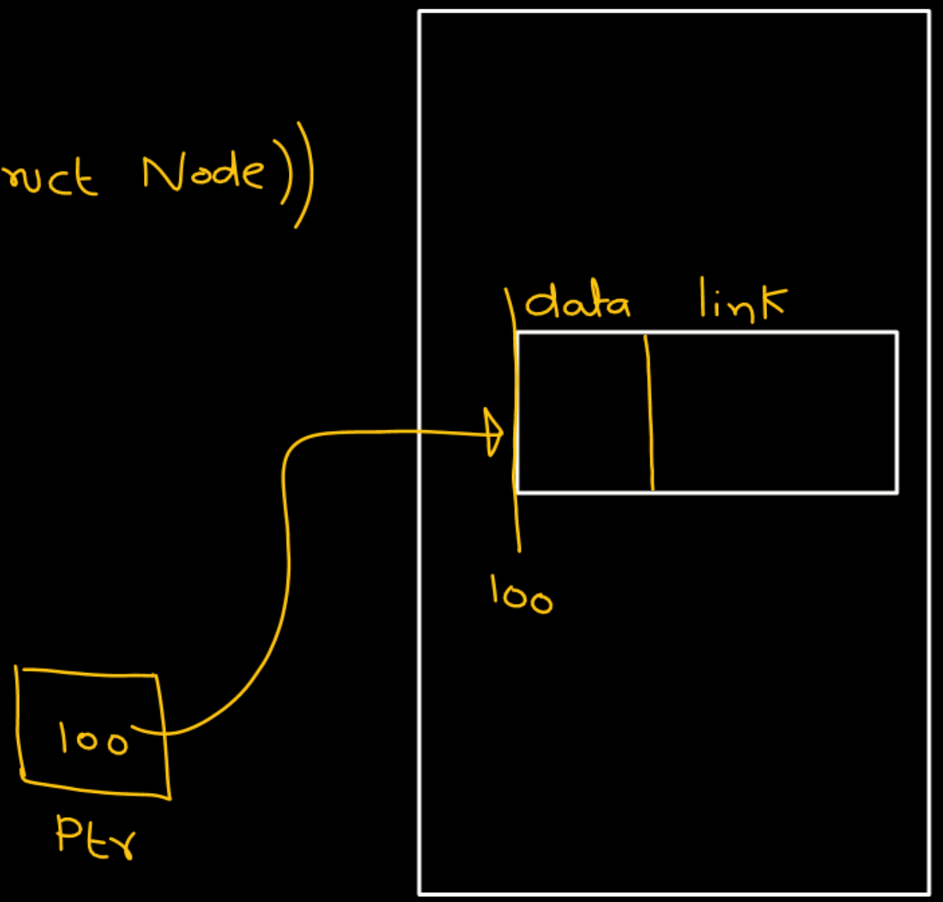
① Memory allocate

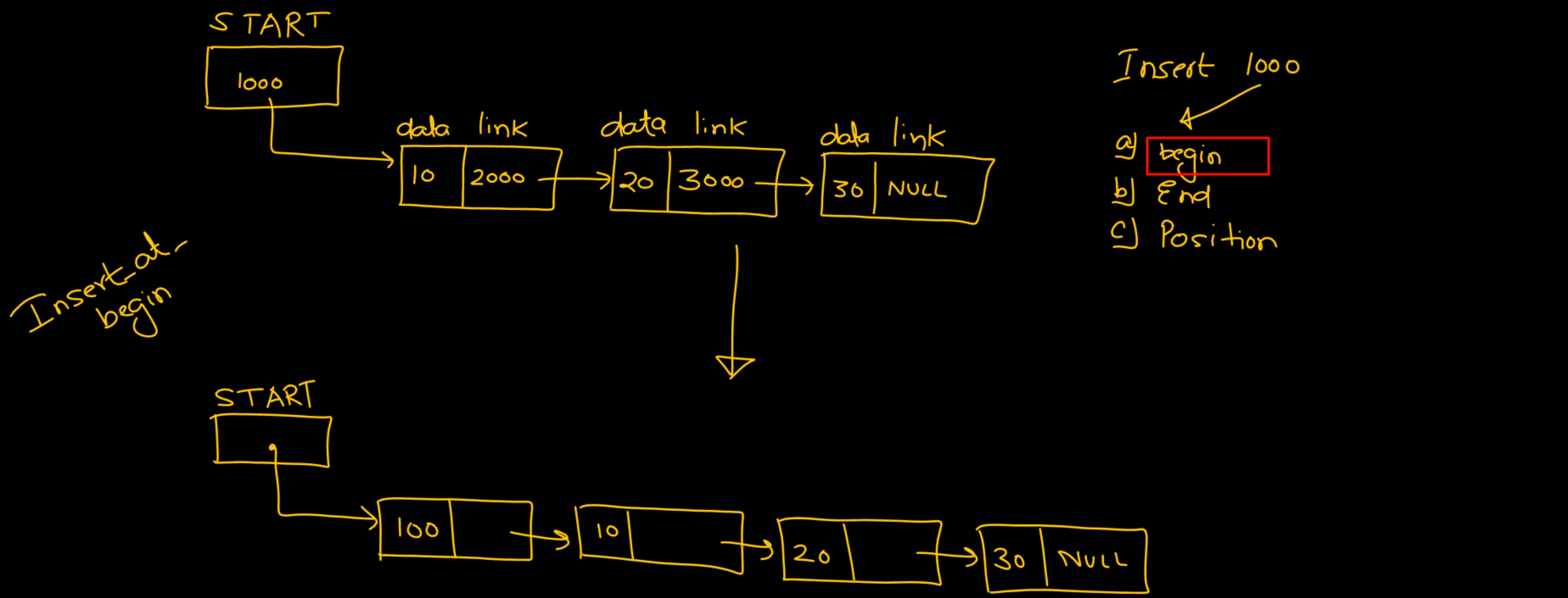
```
struct Node *Ptr;
Ptr = malloc(sizeof(struct Node))
```

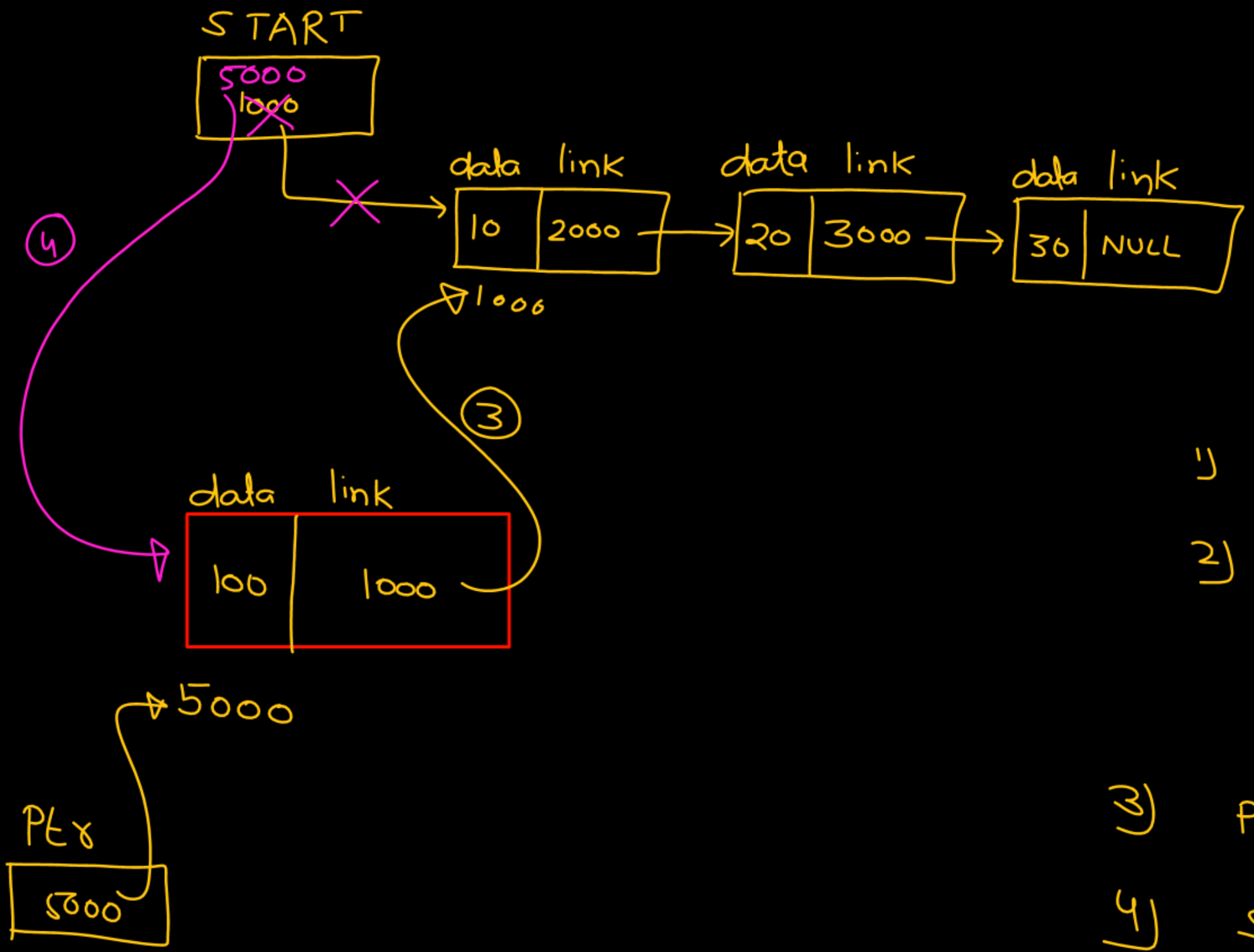
② Insert data/key in this new node

```
Ptr → data = 100;
Ptr → data = key;
```

③ Where to place this node




START
1000
data link
10 2000
data link
20 3000
data link
30 NULL
Insert 1000
a) begin
b) End
c) Position
Insert_at_begin
START
100
10
20
30 NULL



Insert 1000

a) begin
b) End
c) Position

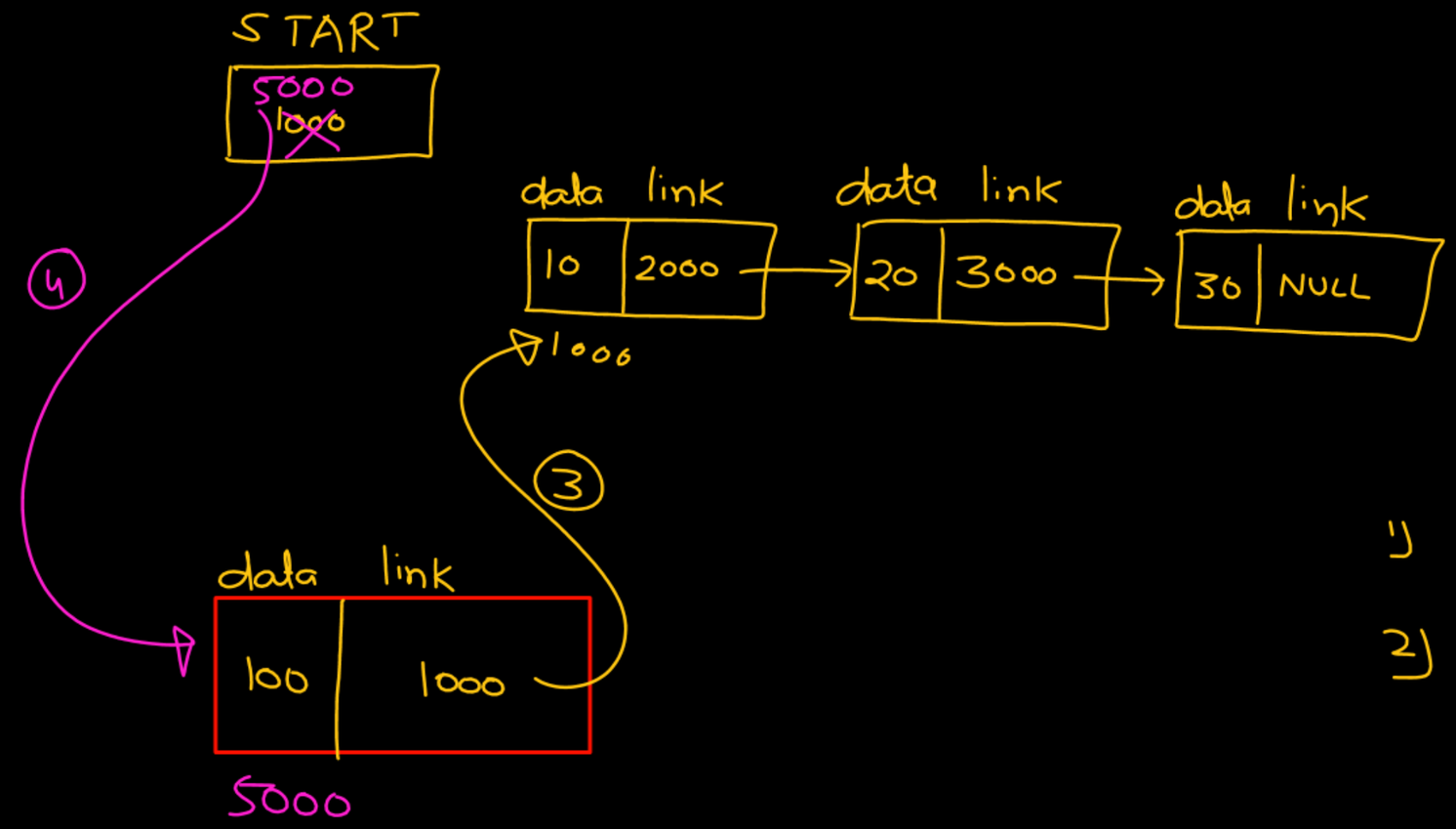
1) struct Node *Ptr ;

2) Ptr = malloc(sizeof(struct Node))

Ptr → data = key;

3) Ptr → link = START

4) START = Ptr ;



Insert 1000

a) begin
b) End
c) Position

1) struct Node *Ptr ;

2) Ptr = malloc(sizeof(struct Node))

Ptr → data = key;

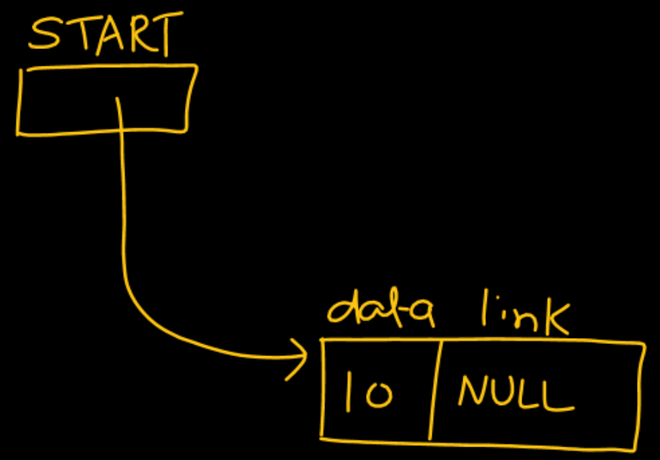
3) Ptr → link = START

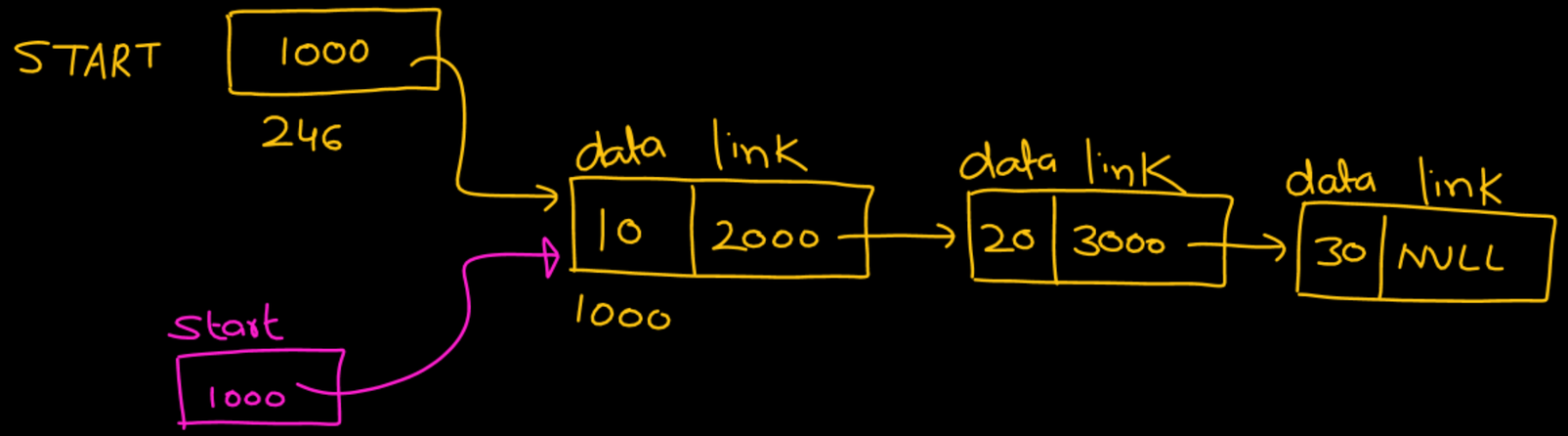
4) START = Ptr ;

```
void Insert_at_beg(int key)
    {
      struct Node *Ptr ;
      Ptr = malloc(sizeof(struct Node));
// Memory   if ( Ptr != NULL) {
  is
  available
          Ptr -> data = key ;
          Ptr -> link = START ;
          START = Ptr ;
                }

    }
```

```
Void main(){
    ≡
Insert_at_beg( key);
    ≡
    }
```

```
void Insert_at_beg(int key)
    {
      struct Node *Ptr;
      Ptr = malloc(sizeof(struct Node));
// Memory   if (Ptr != NULL) {
  is
  available
          Ptr -> data = key;
          Ptr -> link = START;
          START = Ptr;
                }

    }
```

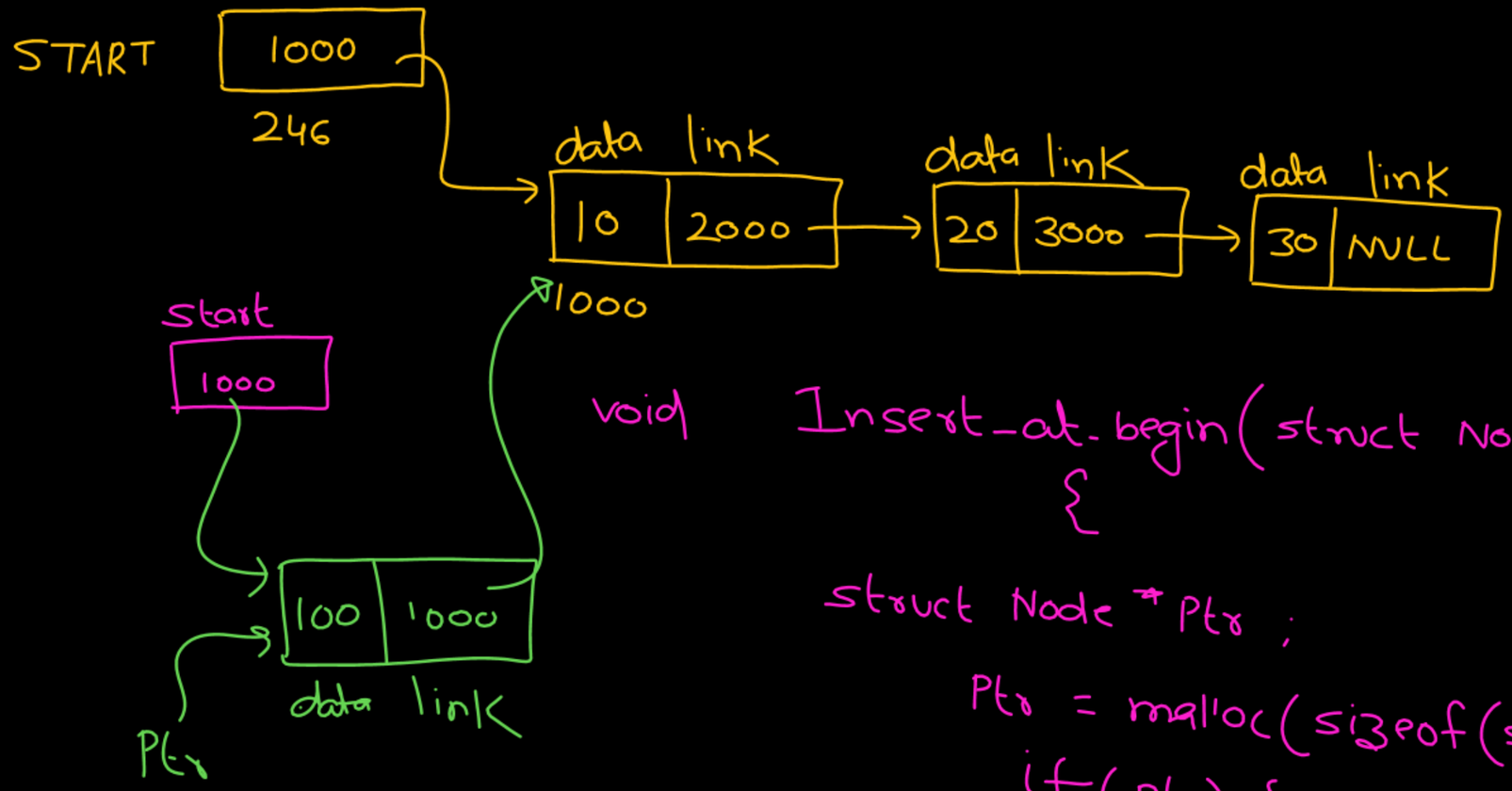





```
void Insert_at_begin(struct Node * start, int key)
    {
    struct Node *Ptr;
    Ptr = malloc(sizeof(struct Node));
    if(Ptr) {
        Ptr->data = key;
        Ptr->link = start;
        start = Ptr;
    }
    }
```

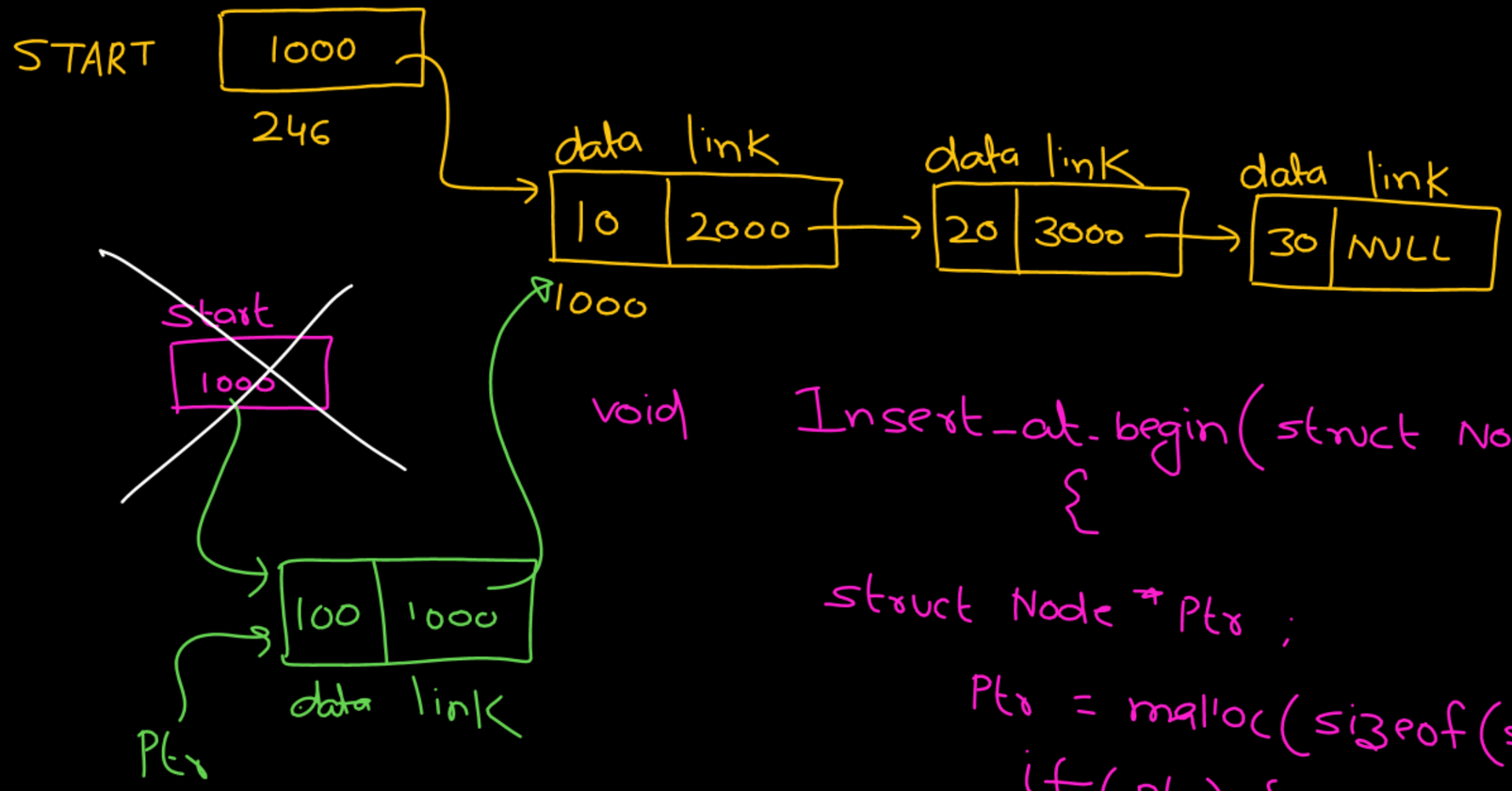
```
void main() {
    struct Node *START;
    ≡
    Insert_at_begin(START, 100);
    ≡
}
```

Call by value



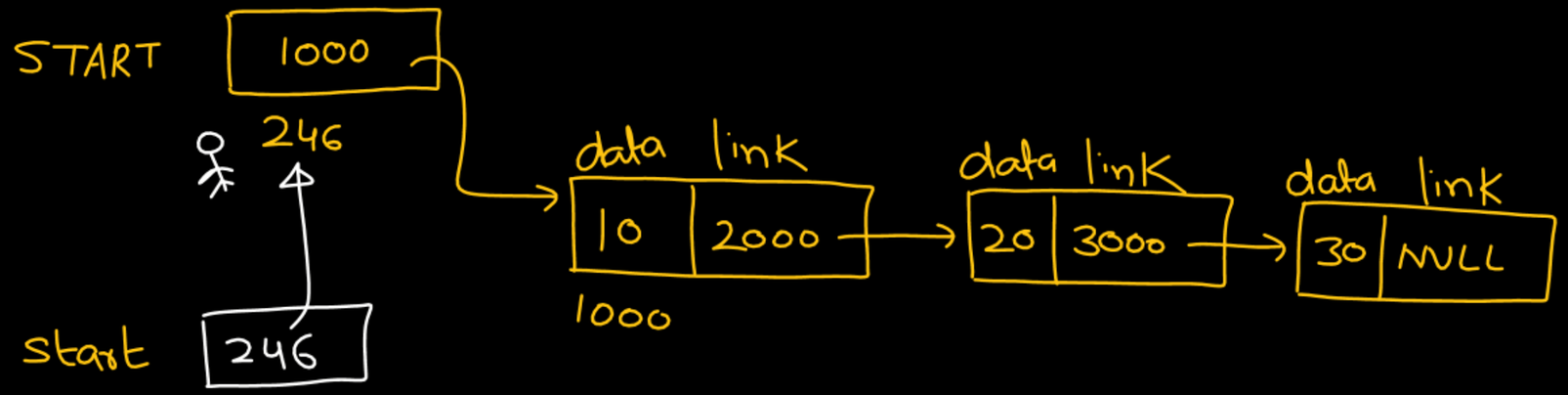

```
void Insert_at_begin(struct Node * start, int key)
{
struct Node *Ptr;
Ptr = malloc(sizeof(struct Node));
if(Ptr) {
Ptr->data = key;
Ptr->link = start;
start = Ptr;
}
}
```

code end

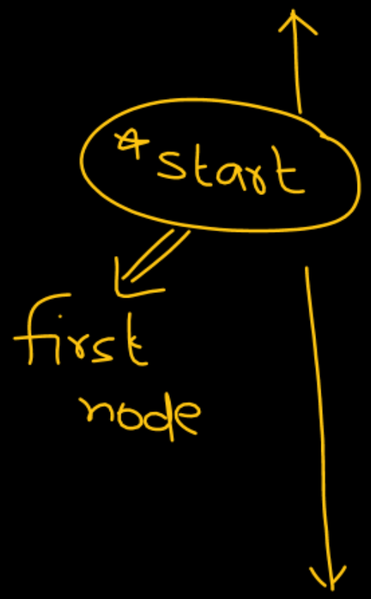



```
void Insert_at_begin(struct Node * start, int key)
{
struct Node *Ptr;
Ptr = malloc(sizeof(struct Node));
if(Ptr) {
Ptr->data = key;
Ptr->link = start;
start = Ptr;
}
}
```

code end




```
void Insert_at_begin(struct Node **start, int key)
{
struct Node *Ptr;
Ptr = malloc(sizeof(struct Node));
  if(Ptr){
    Ptr->data = key;
    Ptr->link = *start;
   *start = Ptr;
        }
}
```

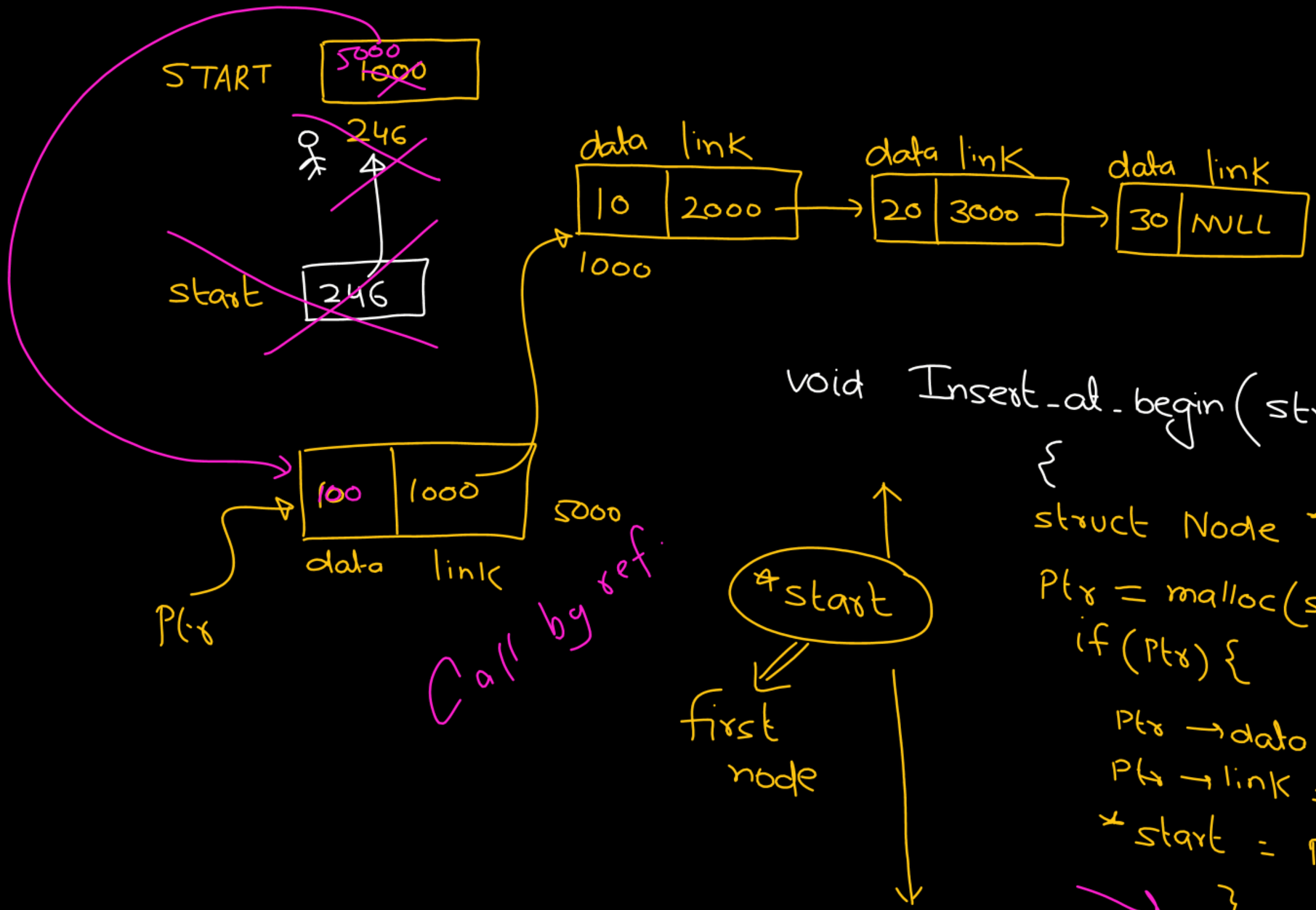



```
void main(){
  struct Node *START;
```

≡

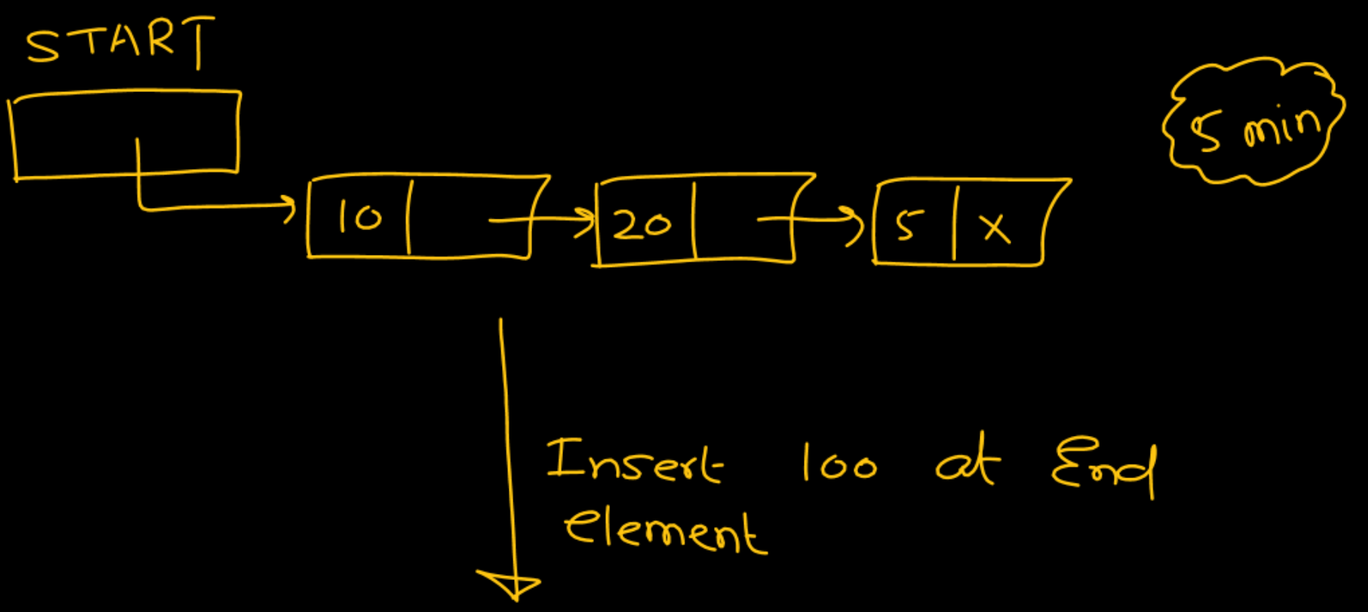
```
Insert_at_begin(&START, 100);
}
```

Pointer of address




```
void Insert_at_begin(struct Node **start, int key)
{
struct Node *Ptr;
Ptr = malloc(sizeof(struct Node));
 if(Ptr){
   Ptr->data = key;
   Ptr->link = *start;
  *start = Ptr;
 }
}
```

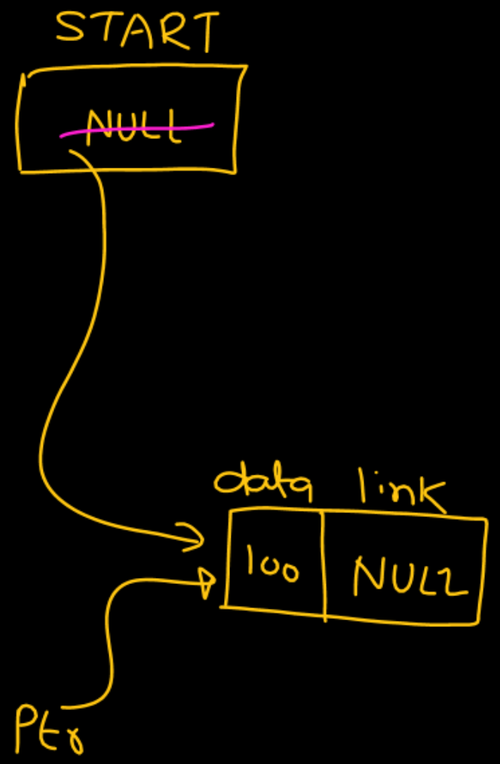
Insert_at_last
Insert_at_End



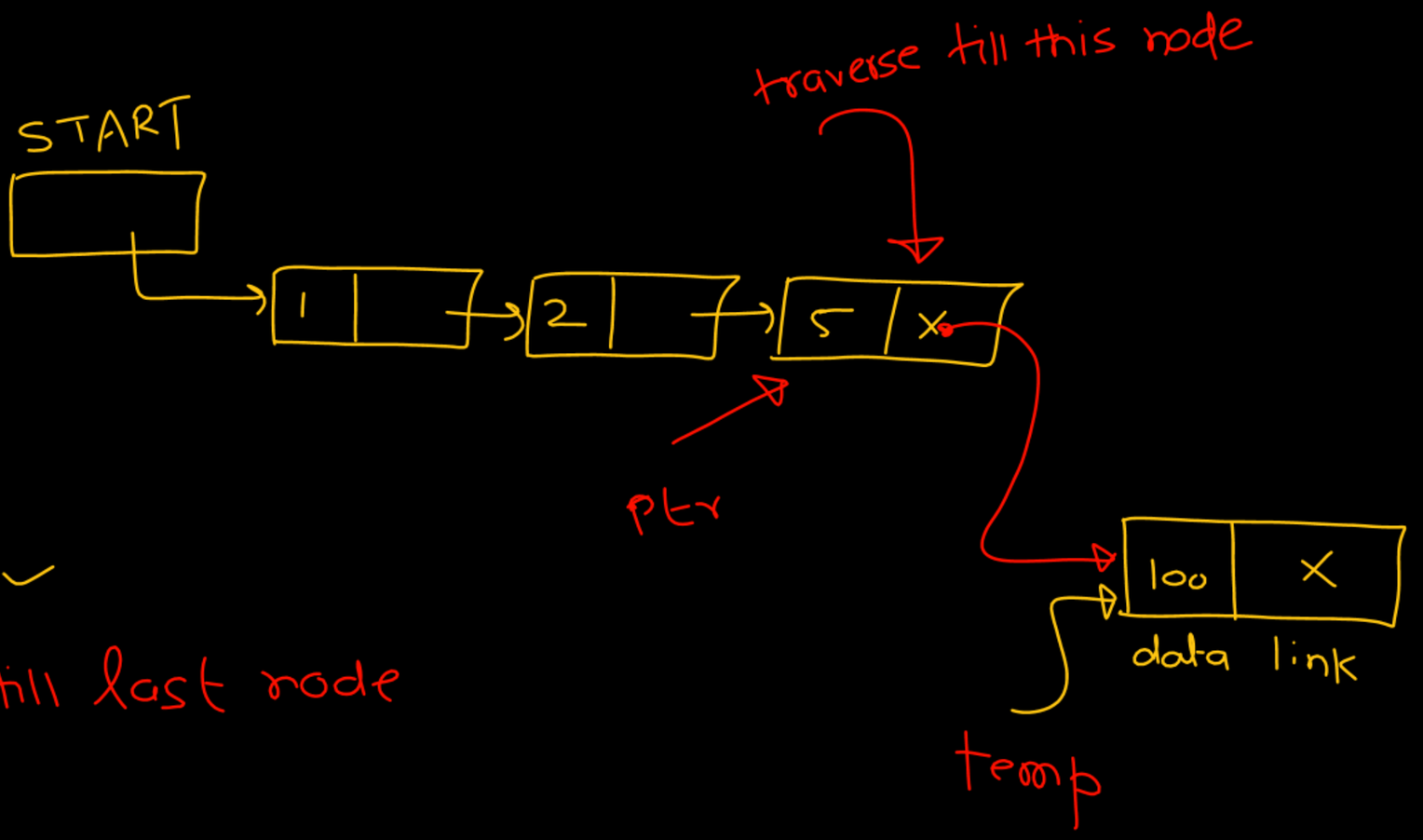
① Node ⇒ create ⇒ Ptr → data = key ;
Ptr → link = NULL ; (last node)

Case 1 : L.L is Empty

```
if ( START == NULL)
    START = Ptr ;
```



L.L is not Empty



(i) Node create ✓

(ii) Traverse till last node

Ensurity ⇓ Atleast 1 node

```
Ptr = START
While (Ptr -> link != NULL)
    {
        Ptr = Ptr -> link
    }
```

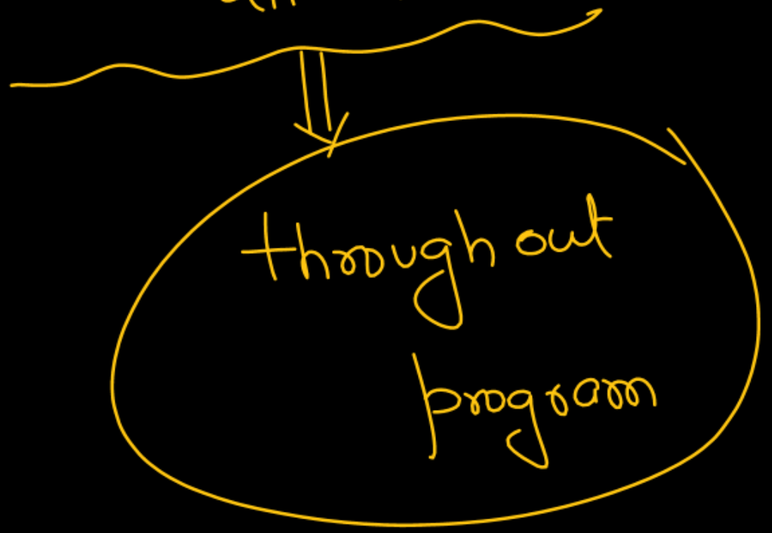
```
Ptr -> link = temp;
```

```
struct Node *temp, *Ptr;
temp = malloc(sizeof(struct Node));
if (temp) {
        temp -> data = key;
        temp -> link = NULL;
        if( START == NULL)
        {
                START = temp;
                return;
        }
```

Atleast 1 node ≠ ∅ ⟶ Ptr = START

```
while(Ptr -> link != NULL)
        Ptr = Ptr -> link;
Ptr -> link = temp;
}
```

Dynamic memory allocation




```
int fun(){
    int a,b;
    int x;
    ≡
    x = a+b*3;
    ≡
    ≡
    return x;
}
```

L.L. ✓ 90%

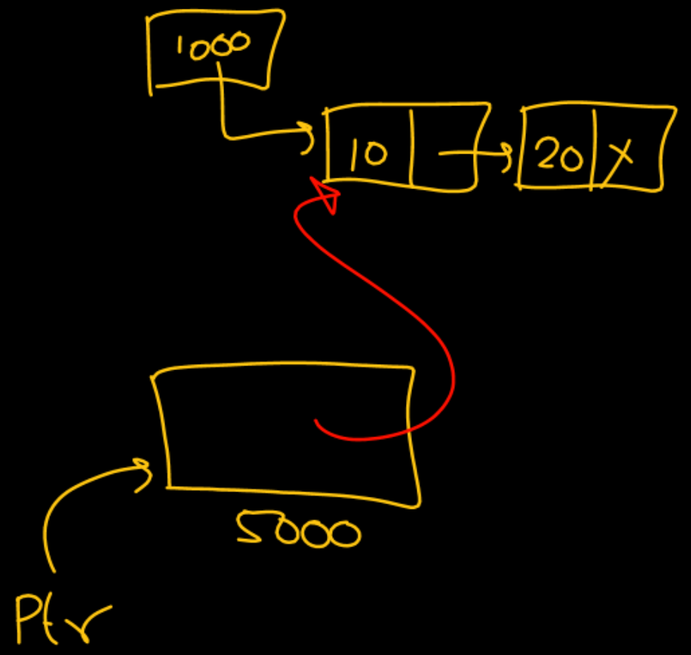
struct Node *

Insert_at_begin(struct Node *start, int Key)

{

struct Node *Ptr ;

Ptr = malloc(____) ;



=

=

=

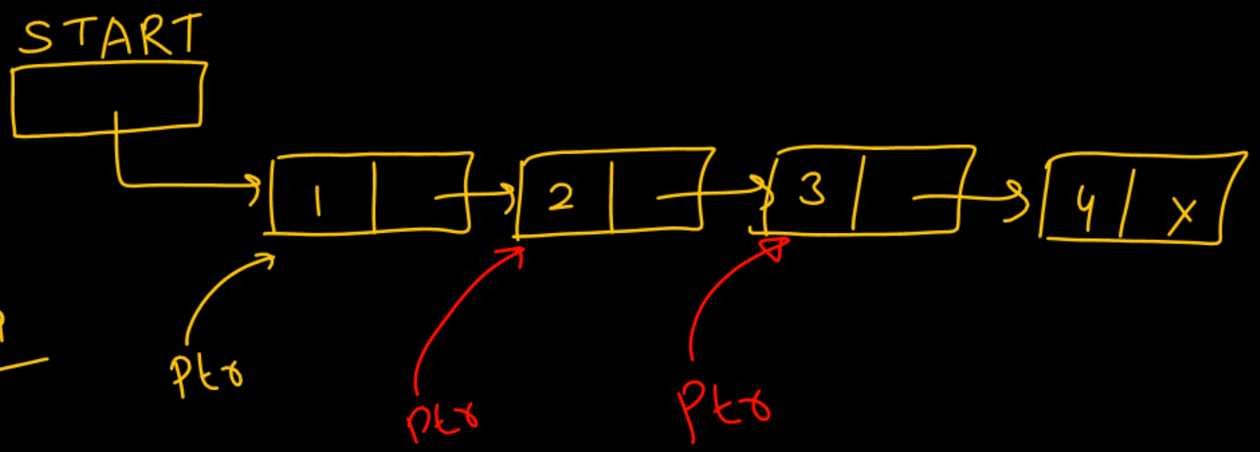
return (Ptr) ; valid

}

main() {

str ___

START = Insert()



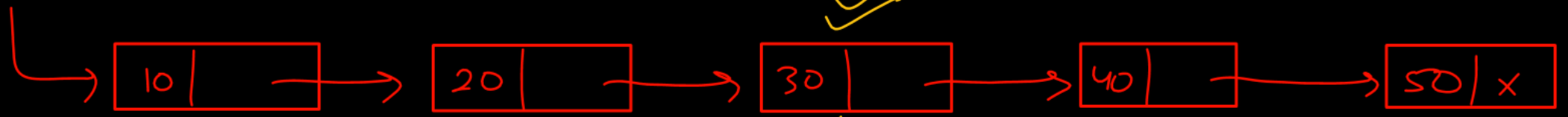
Count = 4

count = count / 2 ;

Count = 2 → 2 times

Ptr = Ptr → link

START



count = 5

count = 2

Ptr

2 times

Ptr = Ptr → link

if count = 0

No middle node

-1

- Traversal
- Search
- Insertion
- Middle Node
- Count Nodes
- Last node
- 2nd last Node

(i) Reverse

(ii) loop

(iii) Intersection point

(iv) Merge 2 sorted l.l.

(v) Deletion

vi) Insertion in sorted l.l.


THANK YOU GW
SOLDIERS !