PROGRAMMING AND DATA STRUCTURES

Creating a node of a Single Linked List

Recall

Self Referential Structure is a structure which contains a pointer to a structure of the same type.

```
struct abc {
    int a;
    char b;
    struct abc *self;
};
```

Node representation in C

Node

Node representation in C

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

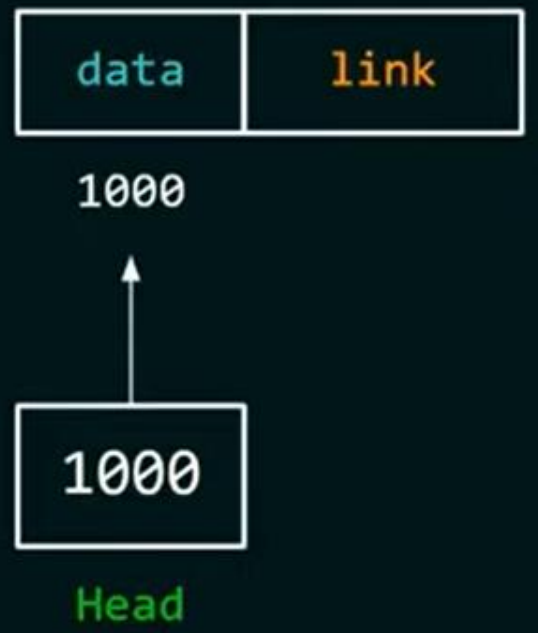
Node

Creating a node in C

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main() {
    struct node *head = NULL;



}
```

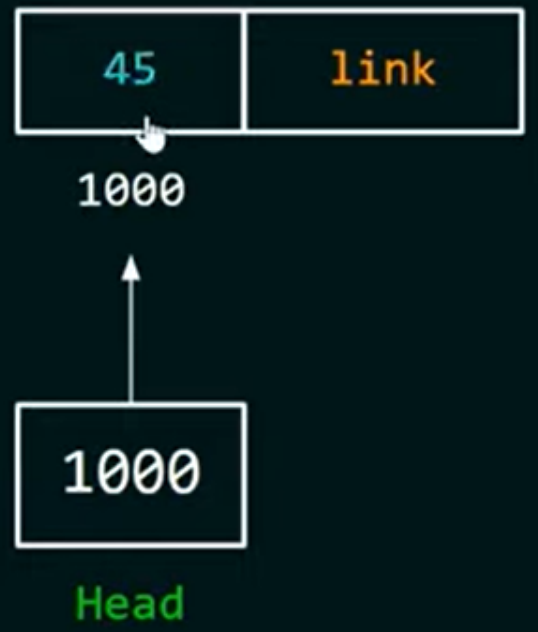
Creating a node in C

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main() {
    struct node *head = NULL;
    head = (struct node *)malloc(sizeof(struct node));



}
```



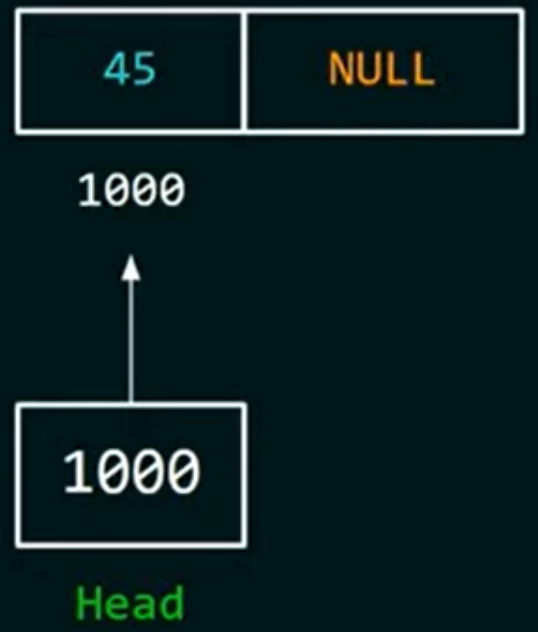
Creating a node in C

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main() {
    struct node *head = NULL;
    head = (struct node *)malloc(sizeof(struct node));

    head->data = 45;



}
```



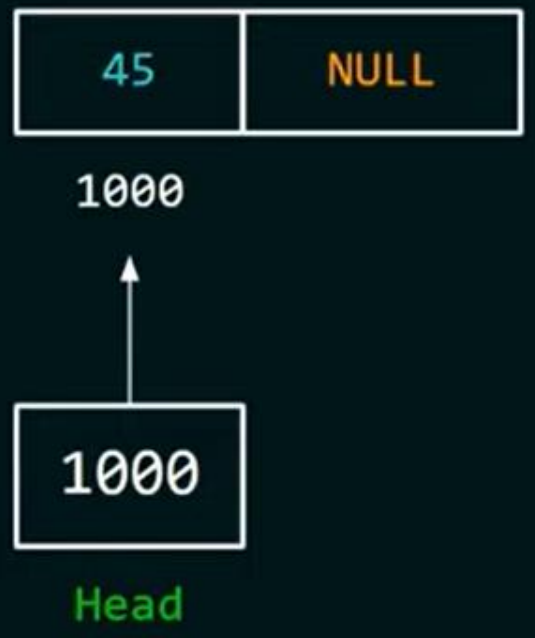
Creating a node in C

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main() {
    struct node *head = NULL;
    head = (struct node *)malloc(sizeof(struct node));

    head->data = 45;
    head->link = NULL;



}
```



Creating a node in C

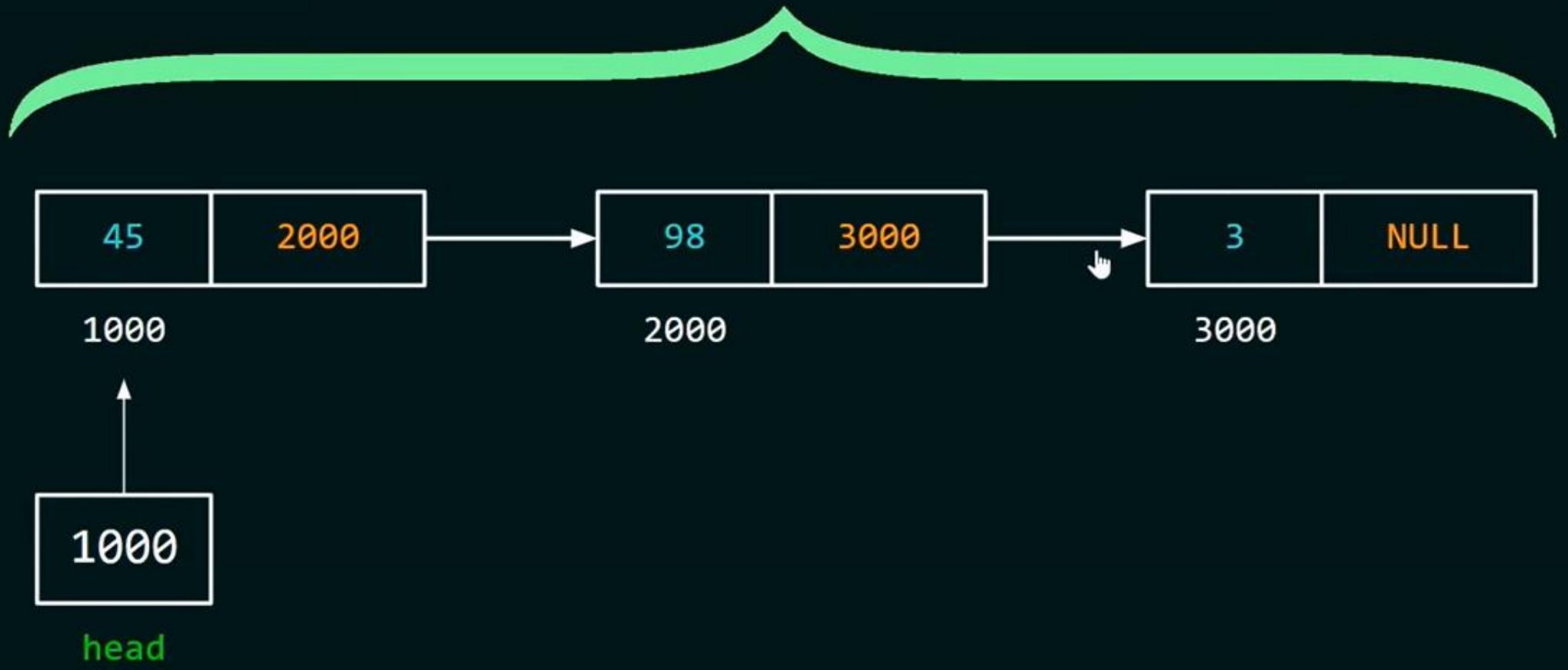
```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main() {
    struct node *head = NULL;
    head = (struct node *)malloc(sizeof(struct node));

    head->data = 45;
    head->link = NULL;

    printf("%d", head->data);
    return 0;
}
```



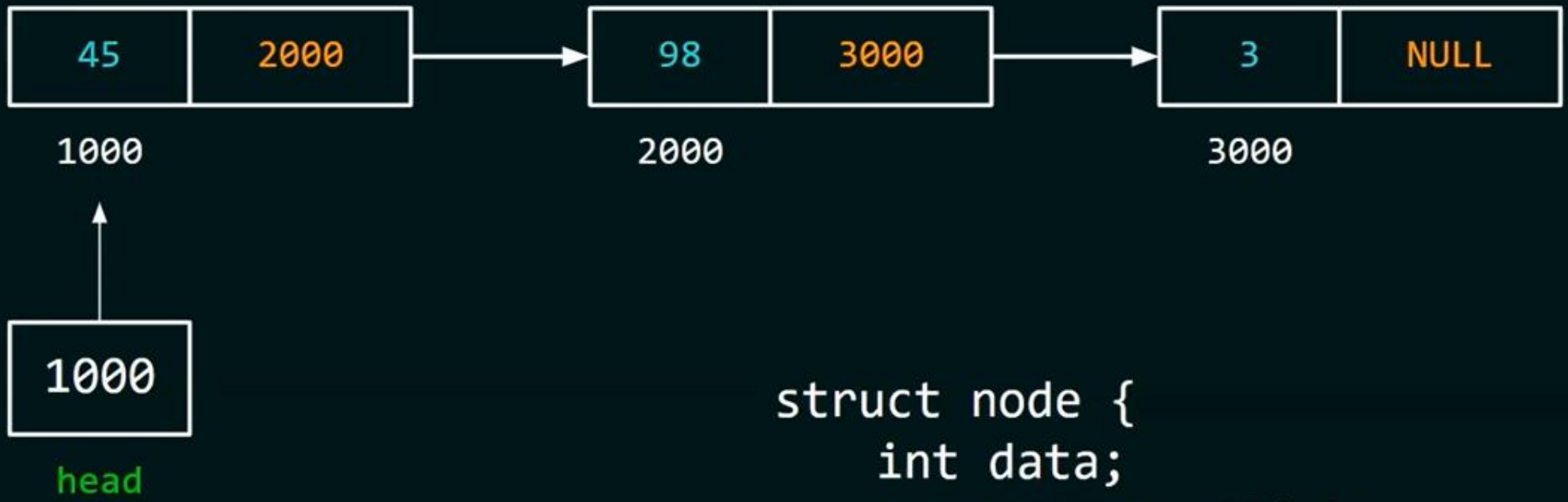
Output: 45

Programming and Data Structures

Creating a Single Linked List (Part 1)



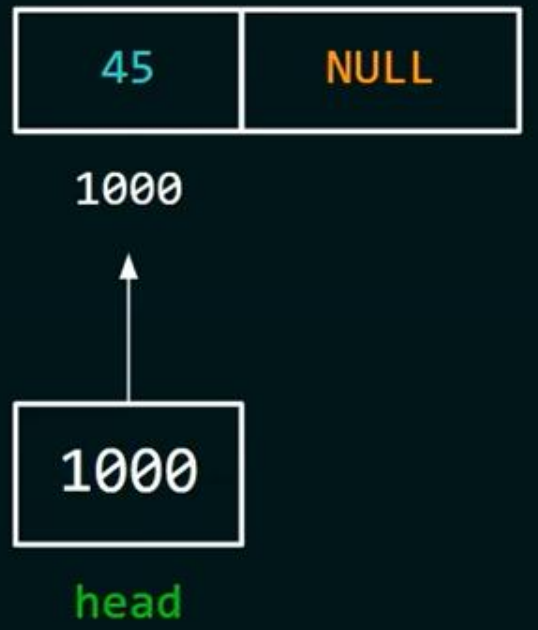
Question: How to create a node of the list?




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

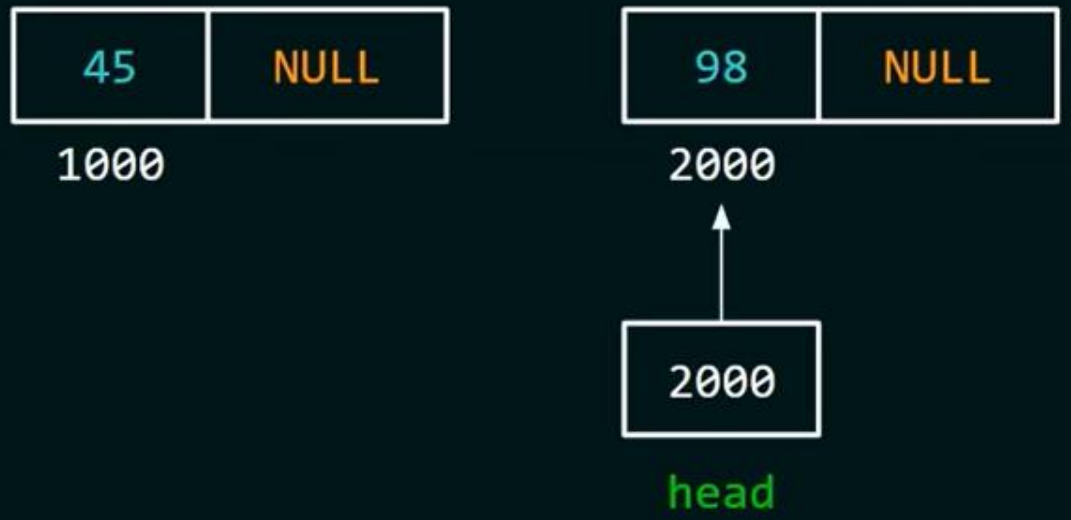
Program

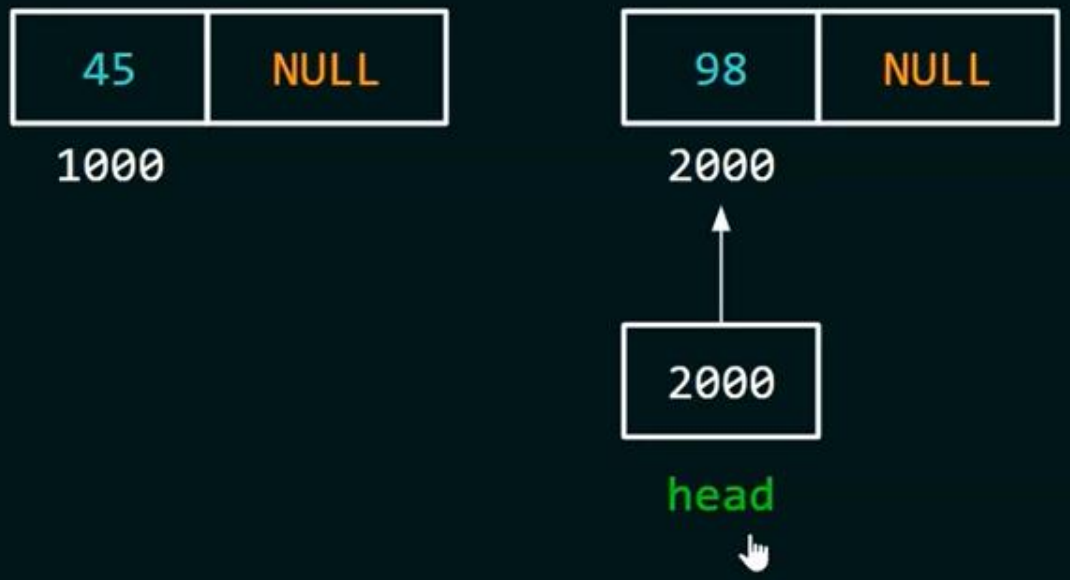
```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main() {
     struct node *head = malloc(sizeof(struct node));
     head->data = 45;
     head->link = NULL;
     return 0;
}
```



Program

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main() {
    struct node *head = malloc(sizeof(struct node));
    head->data = 45;
    head->link = NULL;

    head = malloc(sizeof(struct node));
    head->data = 98;
    head->link = NULL;

    return 0;
}
```

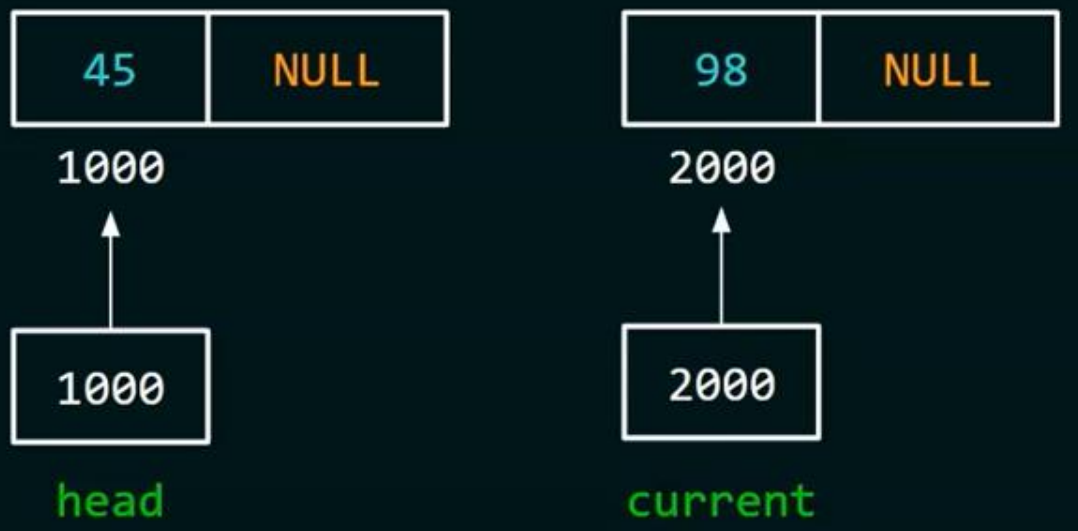




- head is now pointing to the second node.
- Now, there is no way to access the first node of the list.

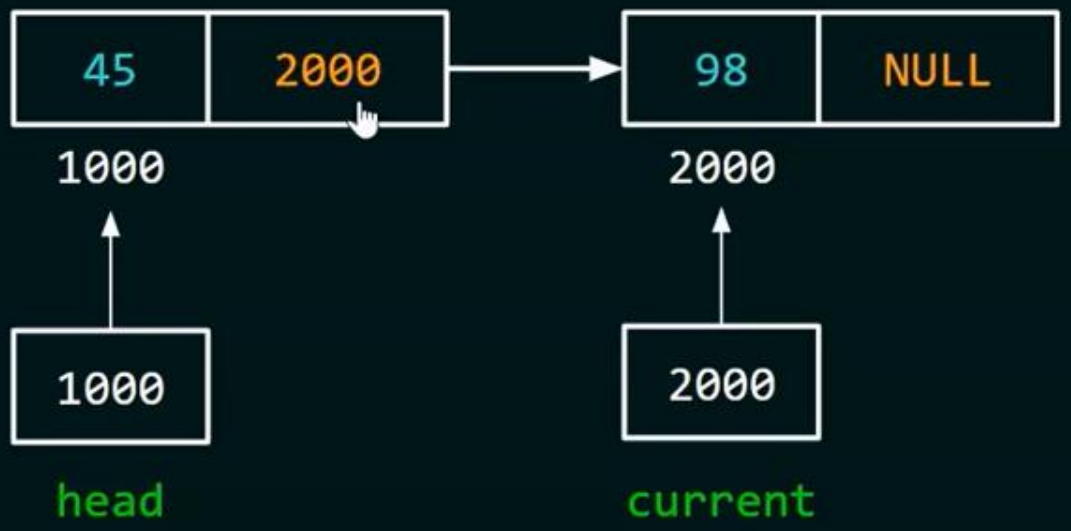
PROGRAM

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main() {
    struct node *head = malloc(sizeof(struct node));
    head->data = 45;
    head->link = NULL;

    struct node *current = malloc(sizeof(struct node));
    current->data = 98;
    current->link = NULL;

    return 0;
}
```



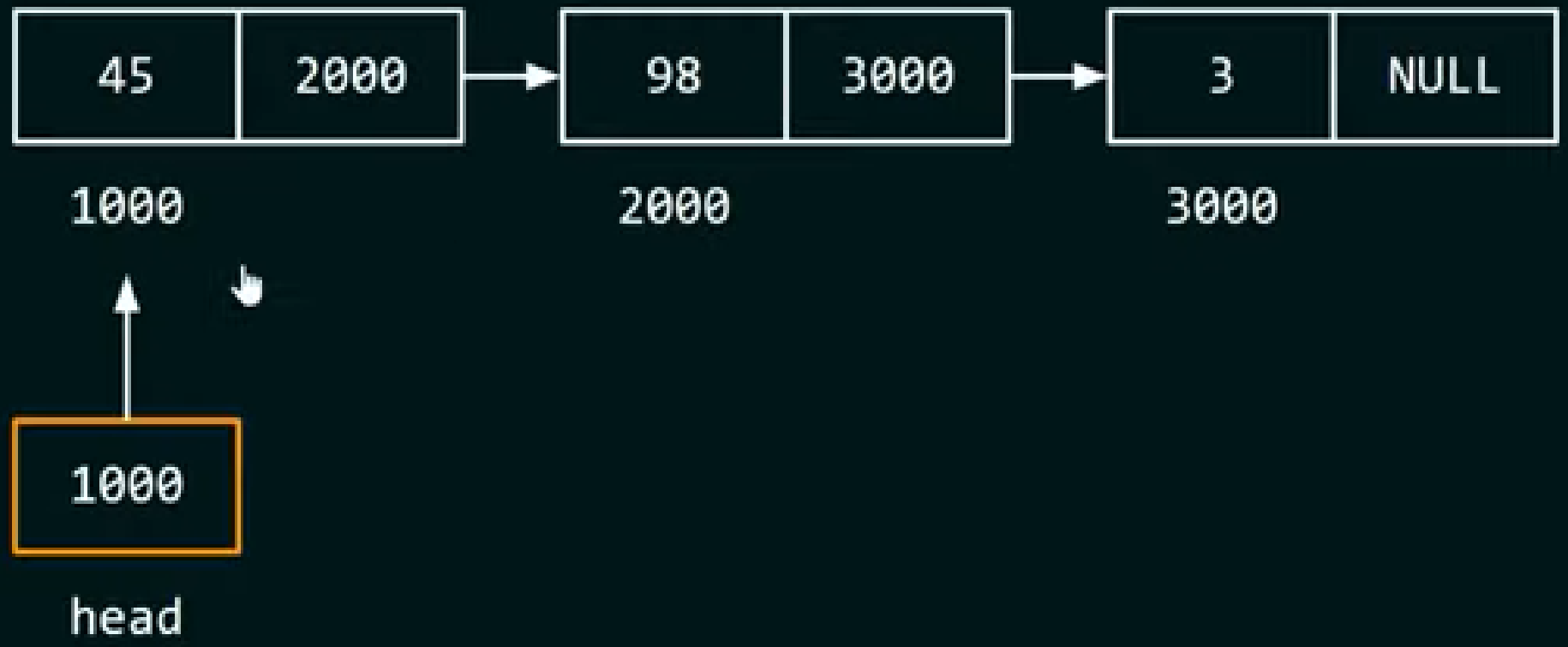
PROGRAM

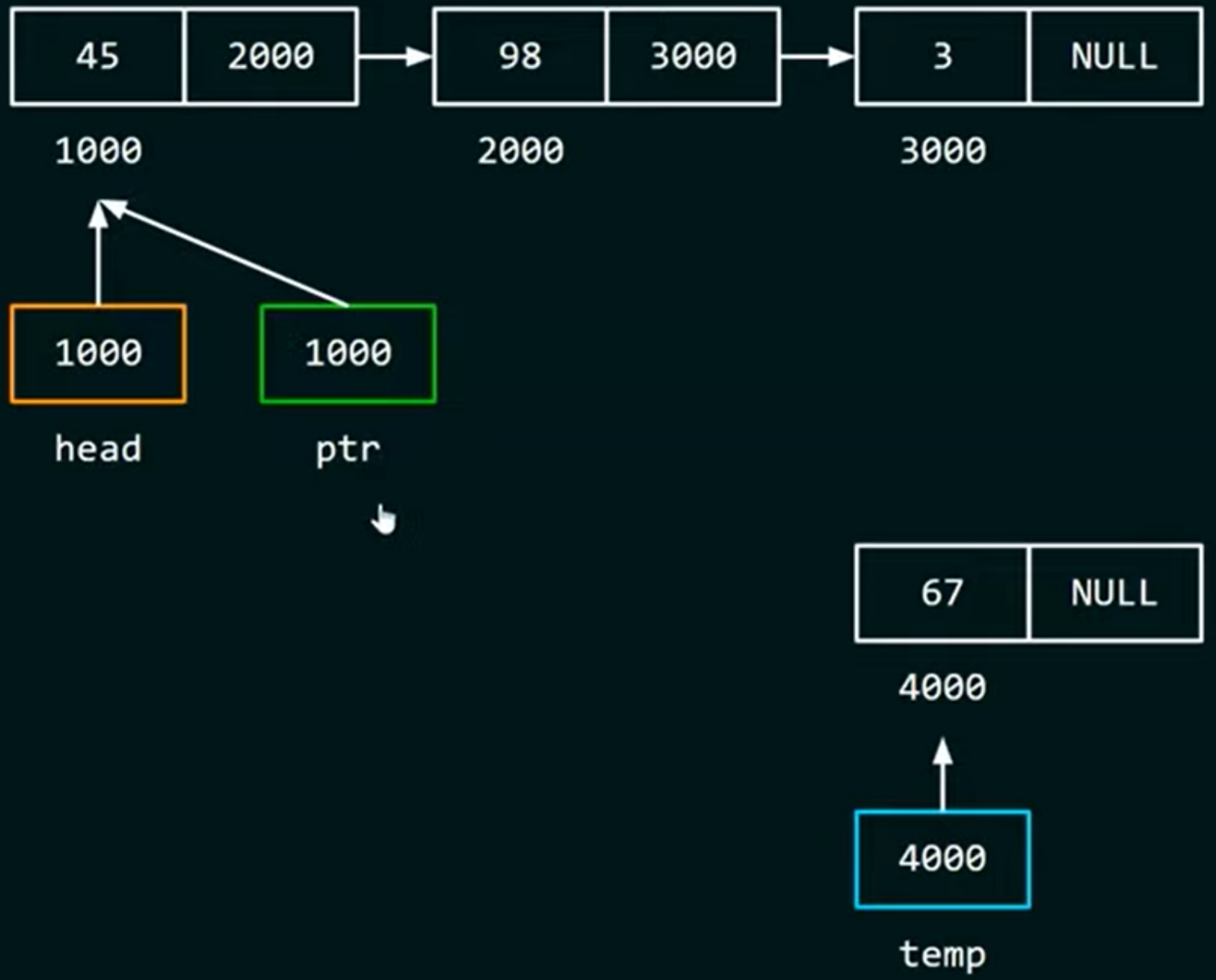
```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main() {
    struct node *head = malloc(sizeof(struct node));
    head->data = 45;
    head->link = NULL;

    struct node *current = malloc(sizeof(struct node));
    current->data = 98;
    current->link = NULL;
    head->link = current;
    return 0;
}
```

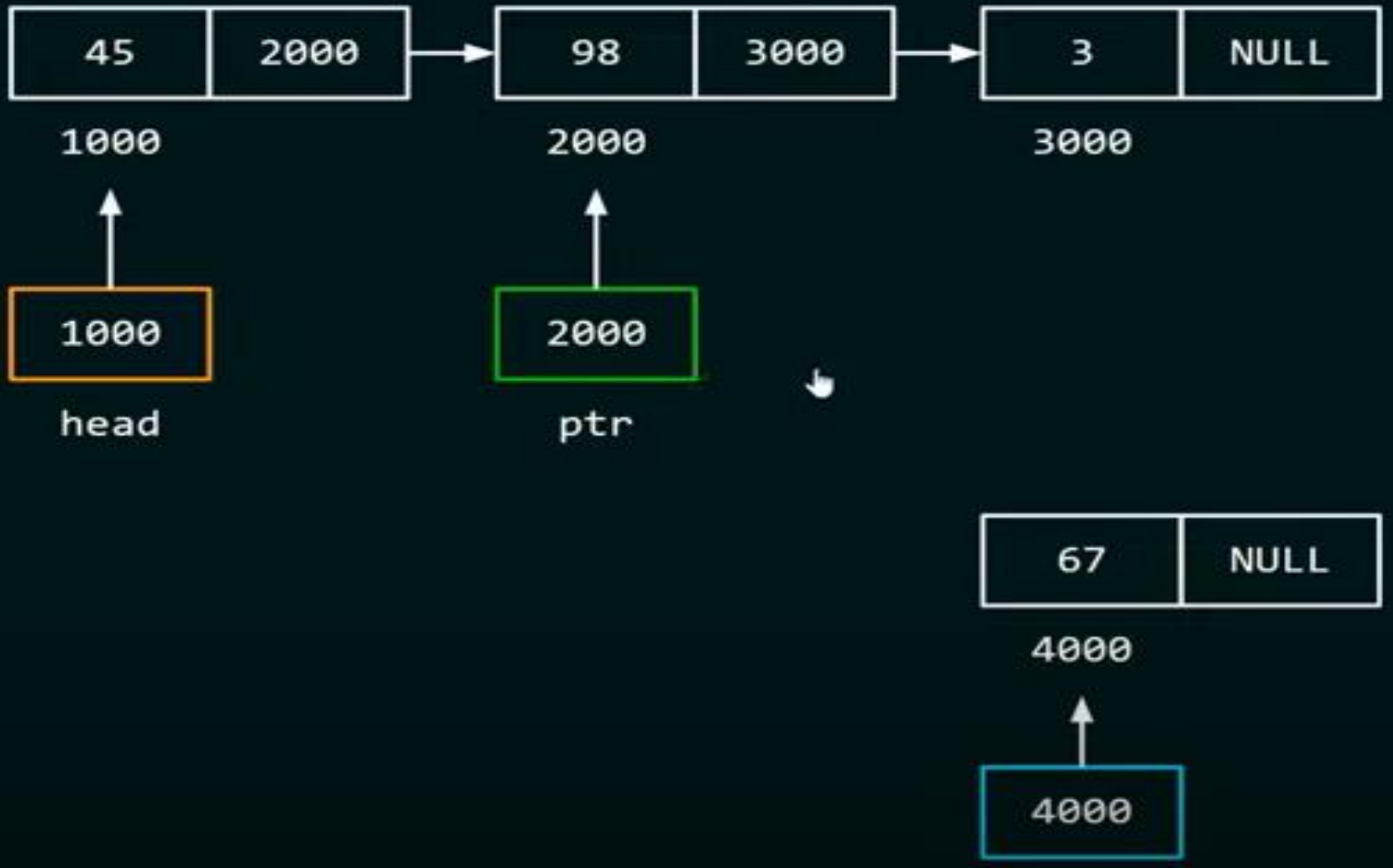


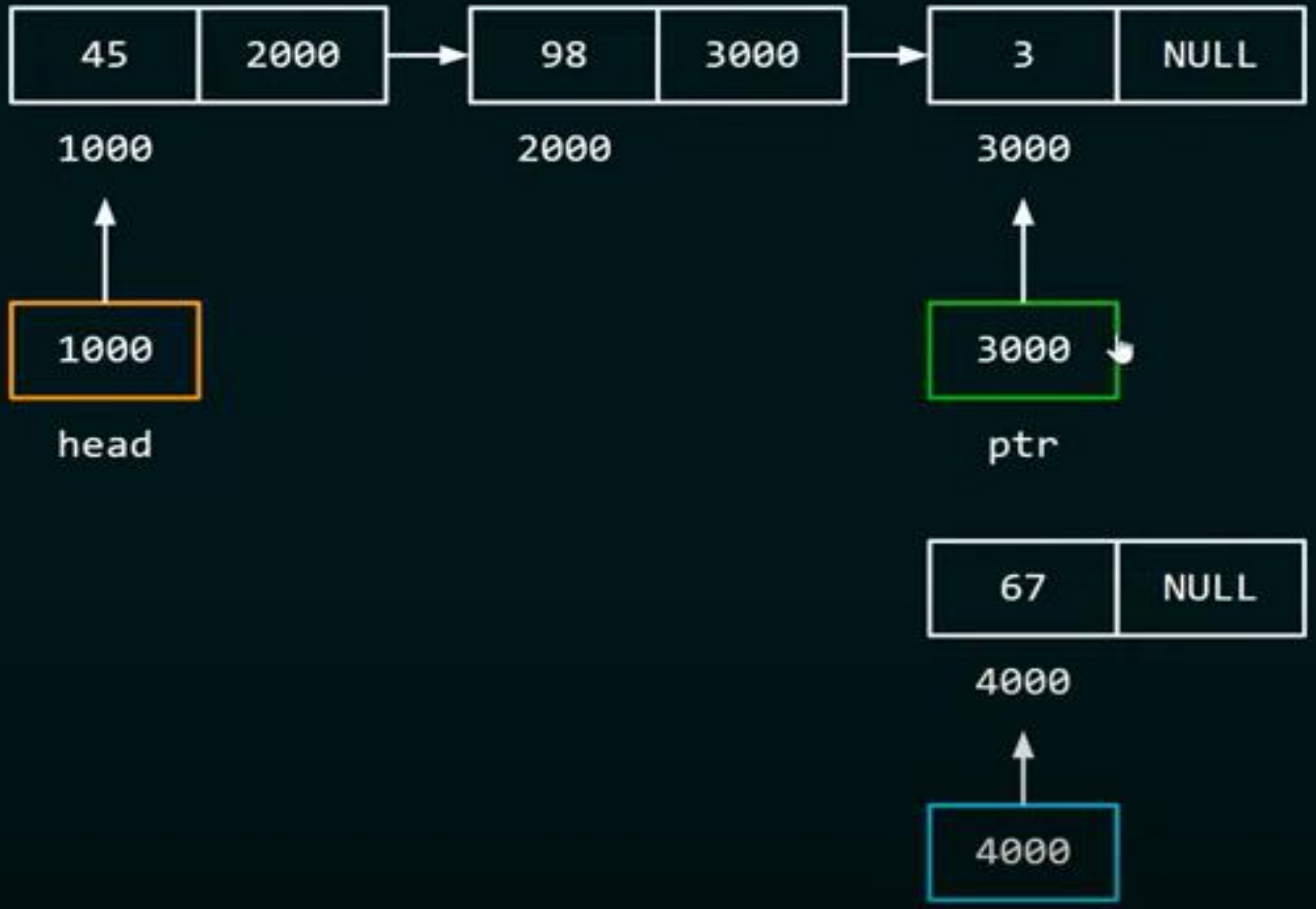
Single Linked List

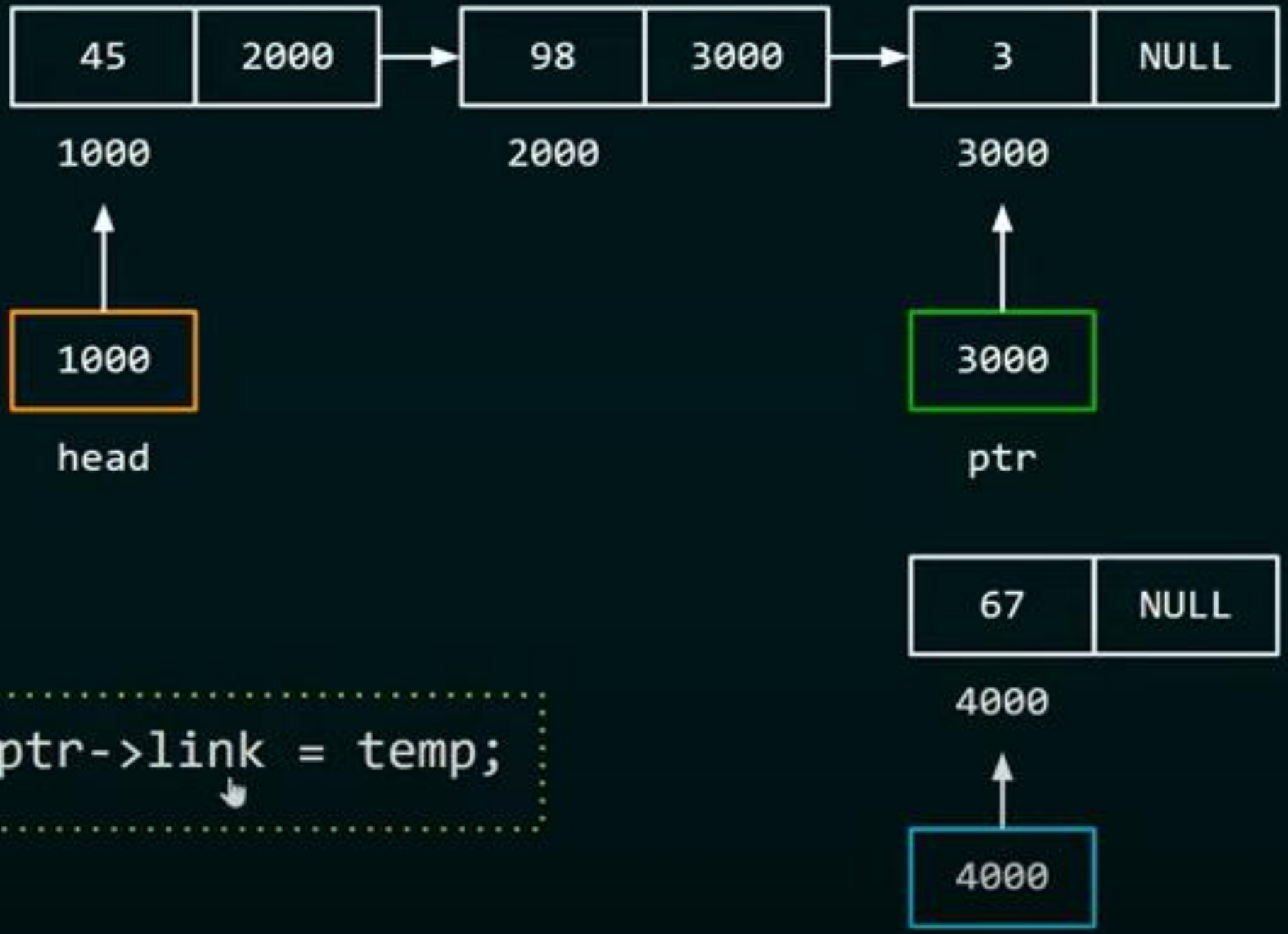
Inserting a Node at the End

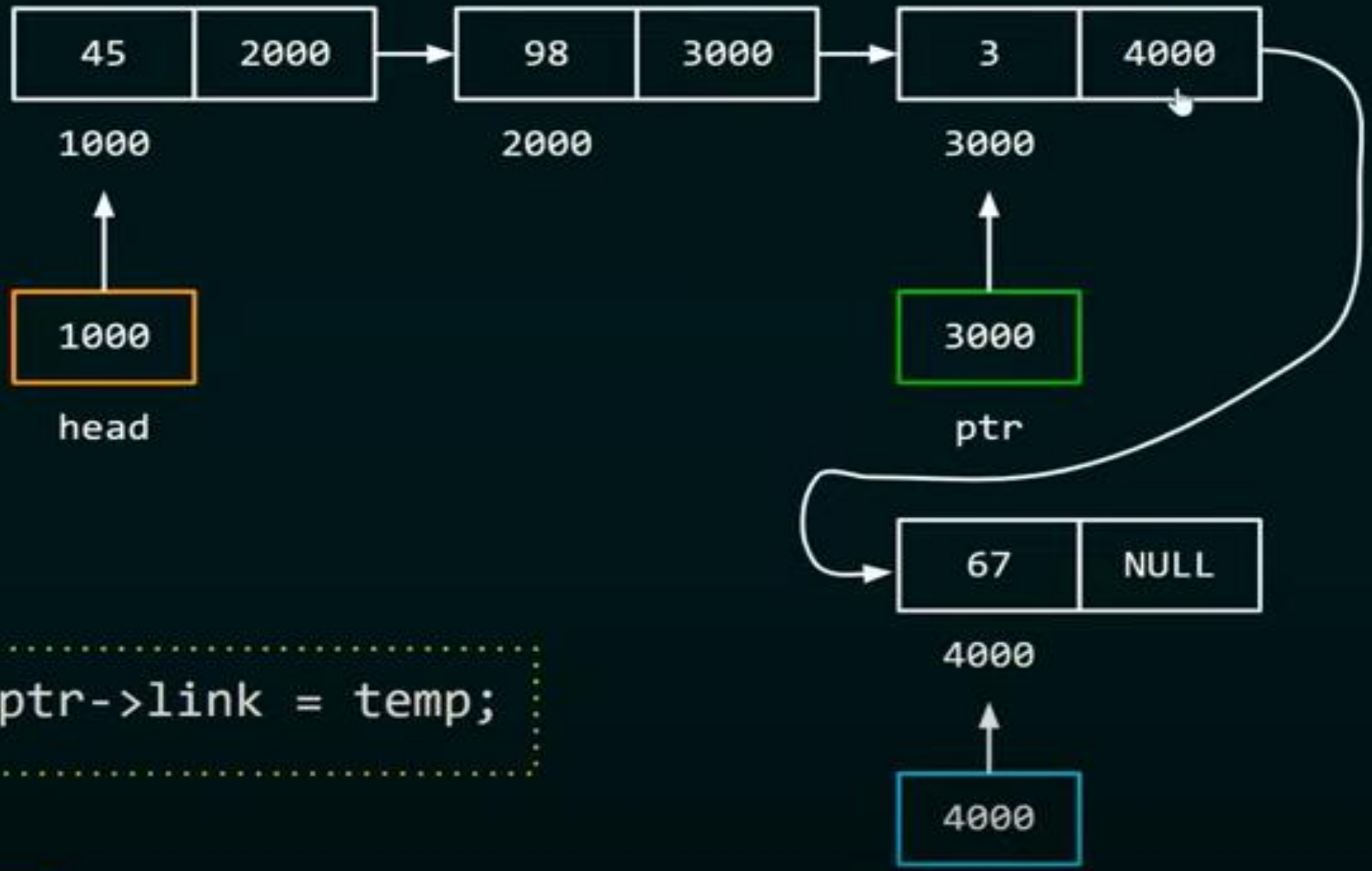

45
2000
98
3000
3
NULL
1000
2000
3000
1000
head

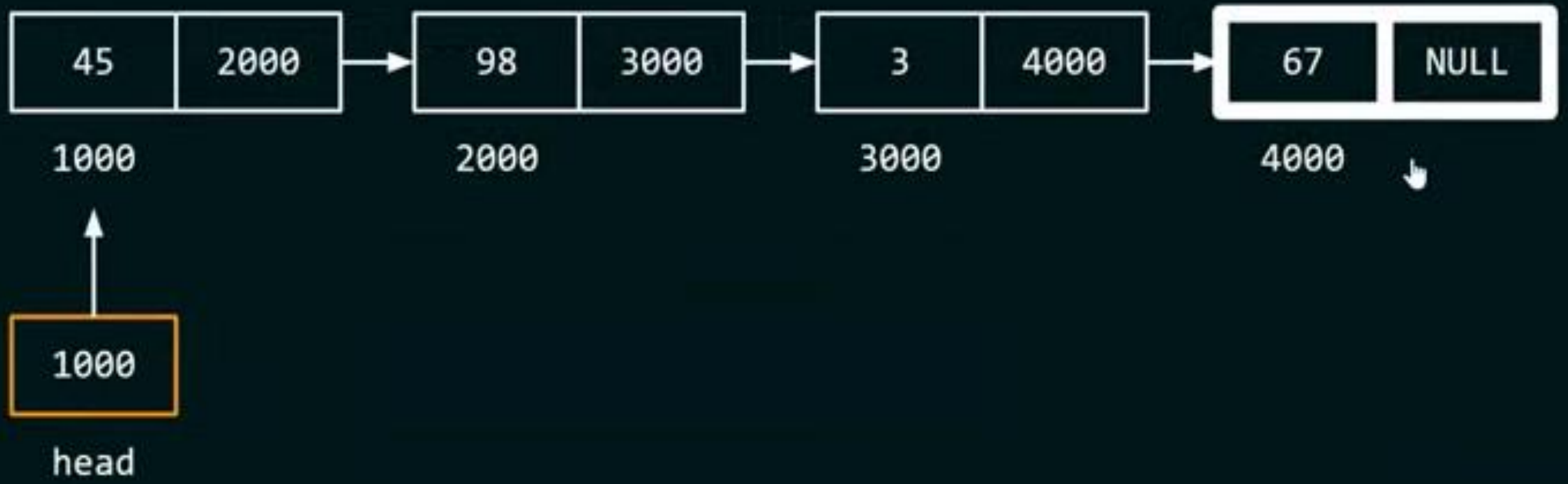



45
2000
98
3000
3
NULL
1000
2000
3000
1000
2000
head
ptr
67
NULL
4000
4000


45
2000
98
3000
3
NULL
1000
2000
3000
1000
head
3000
ptr
67
NULL
4000
4000

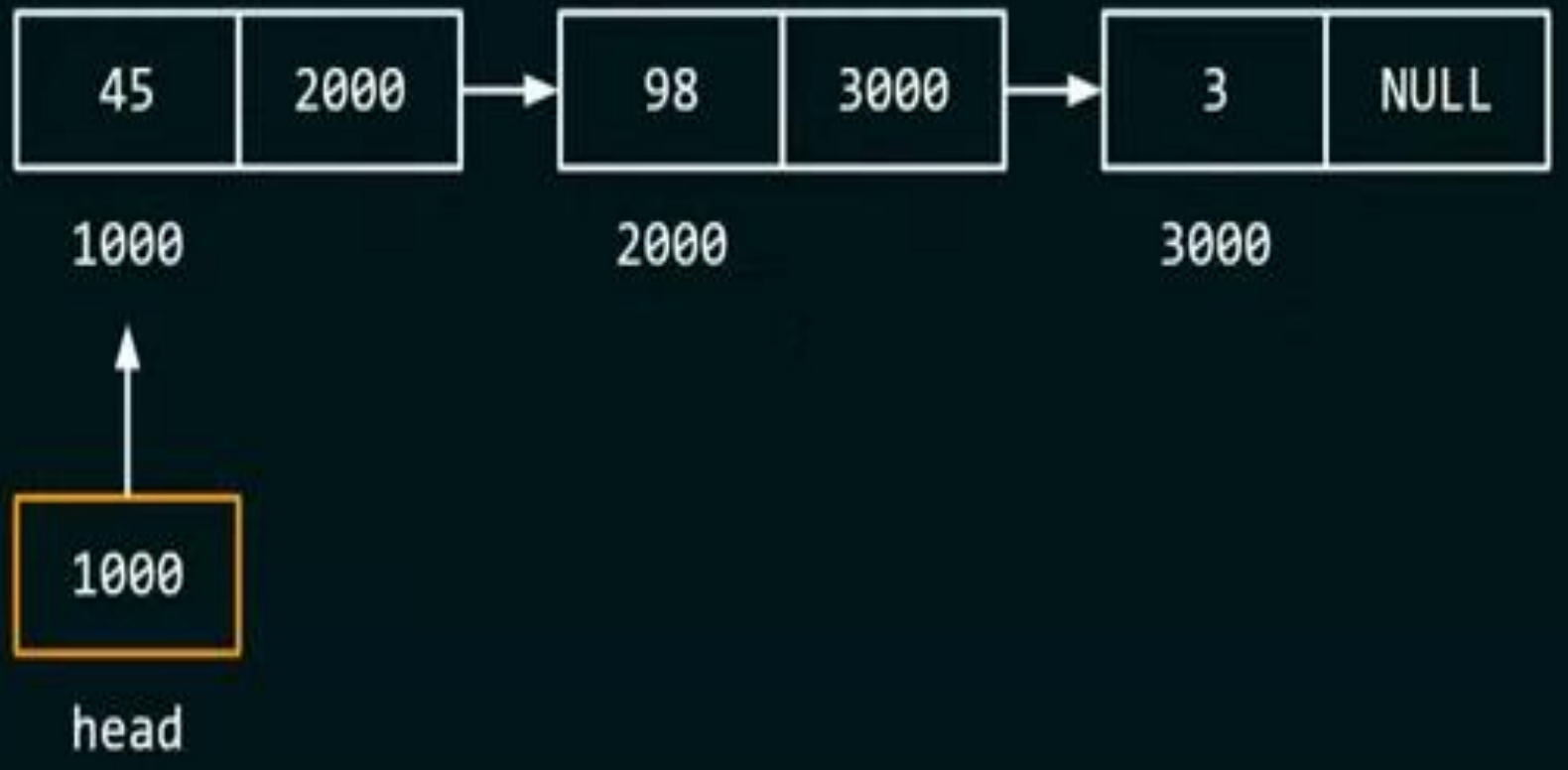

45
2000
98
3000
3
NULL
1000
2000
3000
1000
3000
head
ptr
67
NULL
4000
ptr->link = temp;
4000


45
2000
98
3000
3
4000
1000
2000
3000
1000
head
3000
ptr
67
NULL
4000
ptr->link = temp;
4000


45
2000
98
3000
3
4000
67
NULL
1000
2000
3000
4000
1000
head

Program

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main() {
    add_at_end(head, 67);
}
```



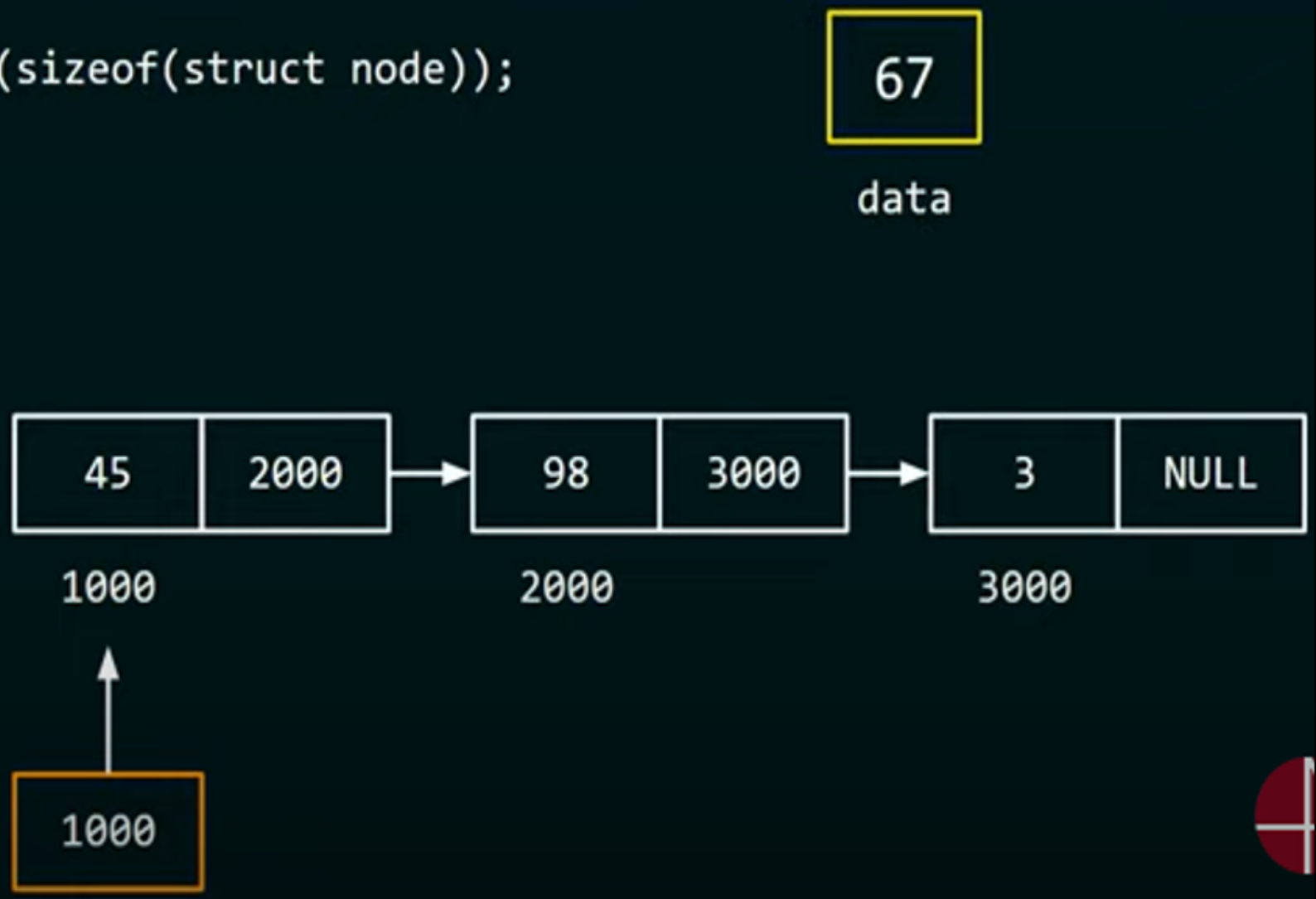
PROGRAM

```
tdio.h>
tdlib.h>

 {
a;
node *link;


{
end(head, 67);
```

```
void add_at_end(struct node *head, int data) {
    struct node *ptr, *temp;
    ptr = head;
    temp = (struct node*)malloc(sizeof(struct node));

    temp->data = data;
    temp->link = NULL;

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

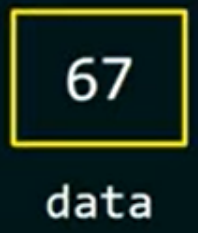
Program

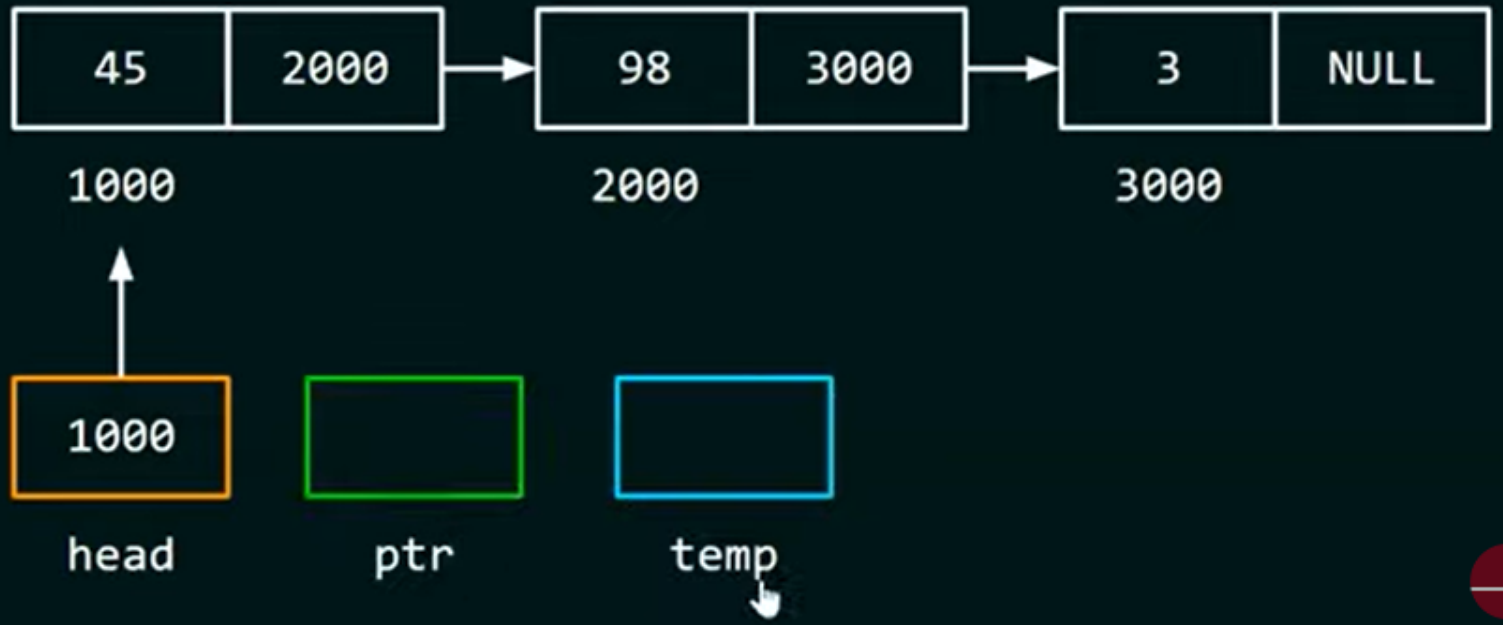
```
void add_at_end(struct node *head, int data) {
    struct node *ptr, *temp;
    ptr = head;
    temp = (struct node*)malloc(sizeof(struct node));

    temp->data = data;
    temp->link = NULL;

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



Program

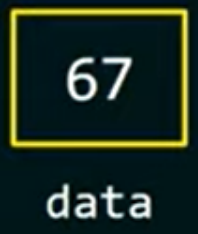
```
void add_at_end(struct node *head, int data) {
    struct node *ptr, *temp;
    ptr = head;
    temp = (struct node*)malloc(sizeof(struct node));

    temp->data = data;
    temp->link = NULL;

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

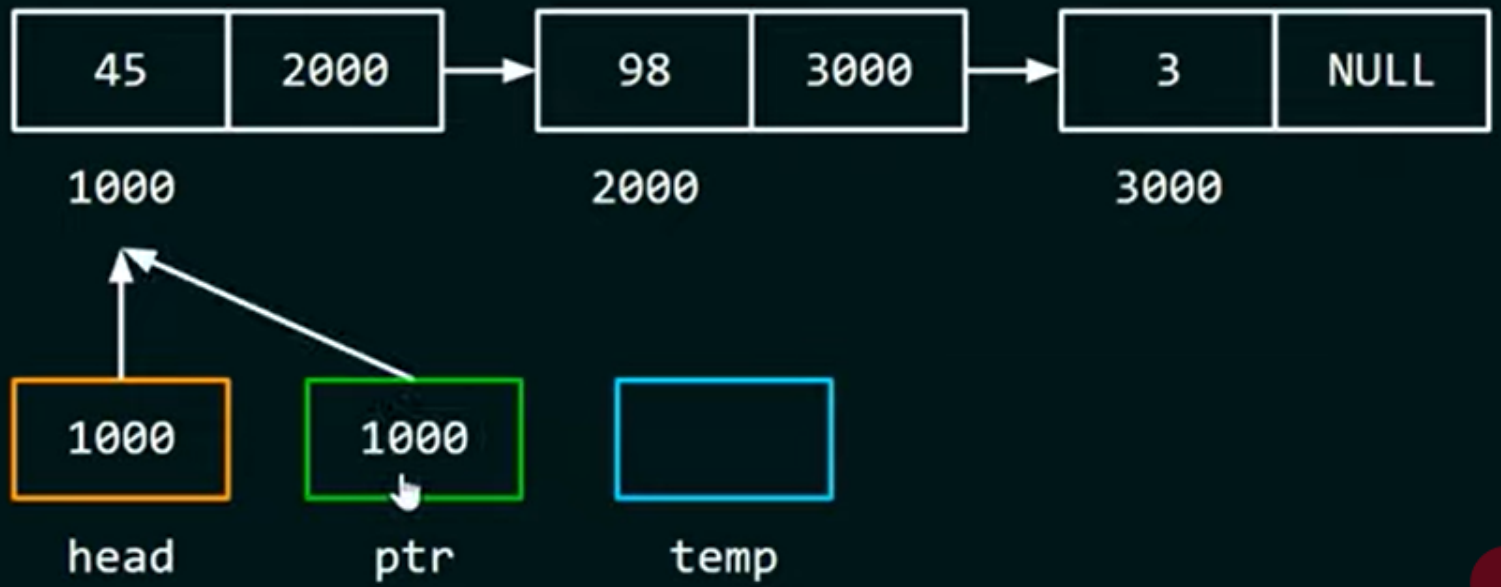




Program

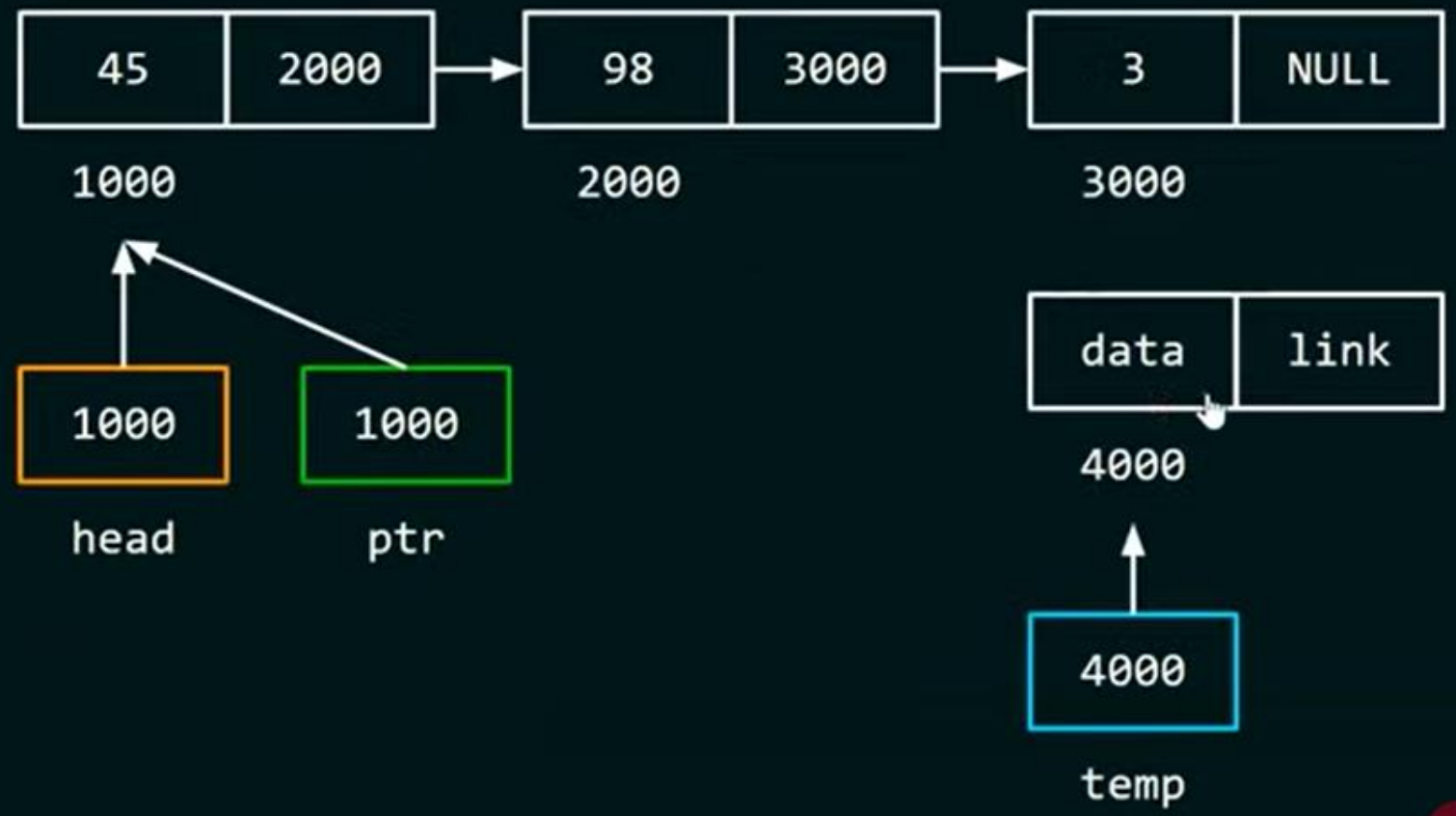
```
void add_at_end(struct node *head, int data) {
    struct node *ptr, *temp;
    ptr = head;
    temp = (struct node*)malloc(sizeof(struct node));

    temp->data = data;
    temp->link = NULL;

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





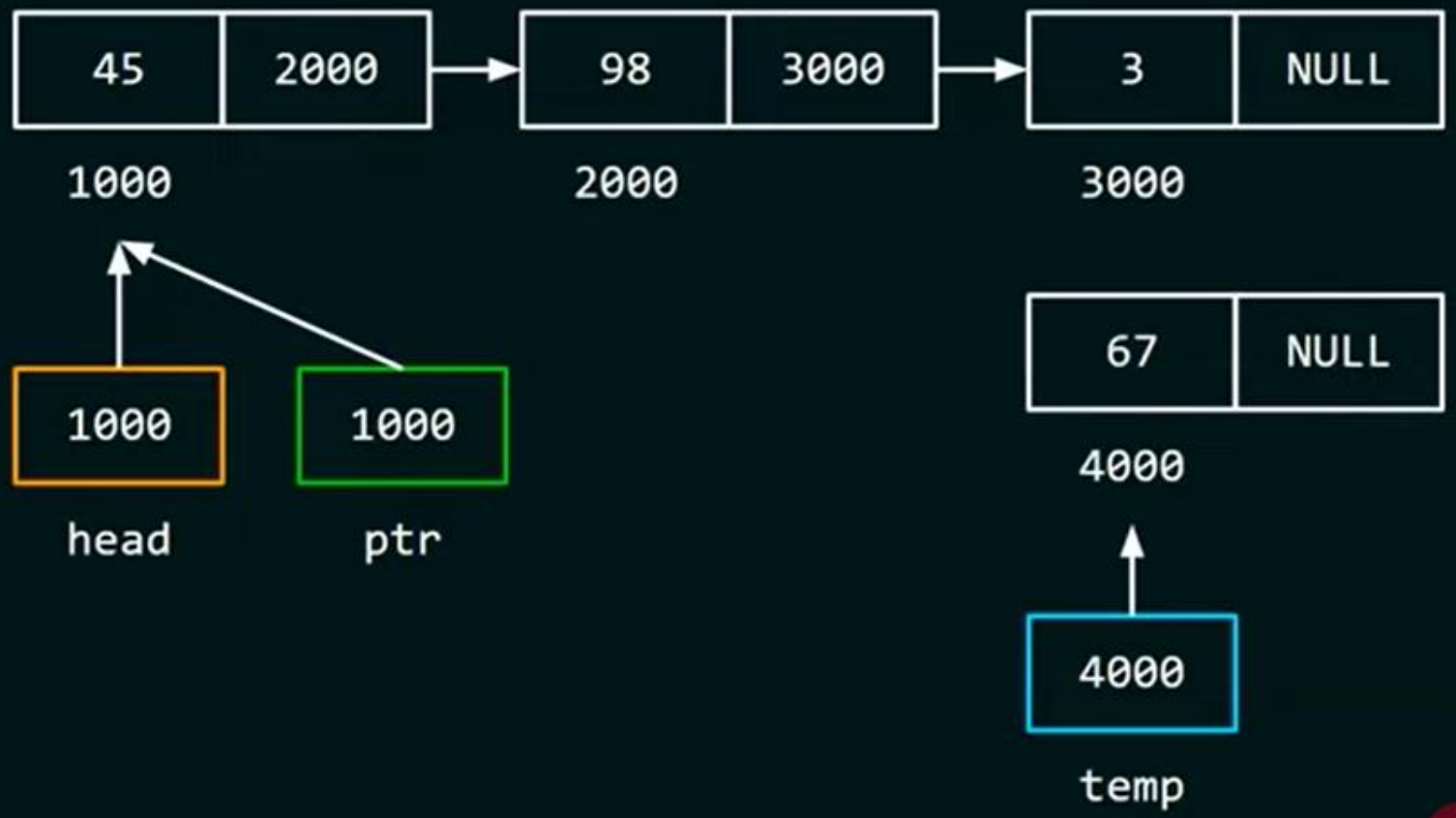
Program

```
void add_at_end(struct node *head, int data) {
    struct node *ptr, *temp;
    ptr = head;
    temp = (struct node*)malloc(sizeof(struct node));

    temp->data = data;
    temp->link = NULL;

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



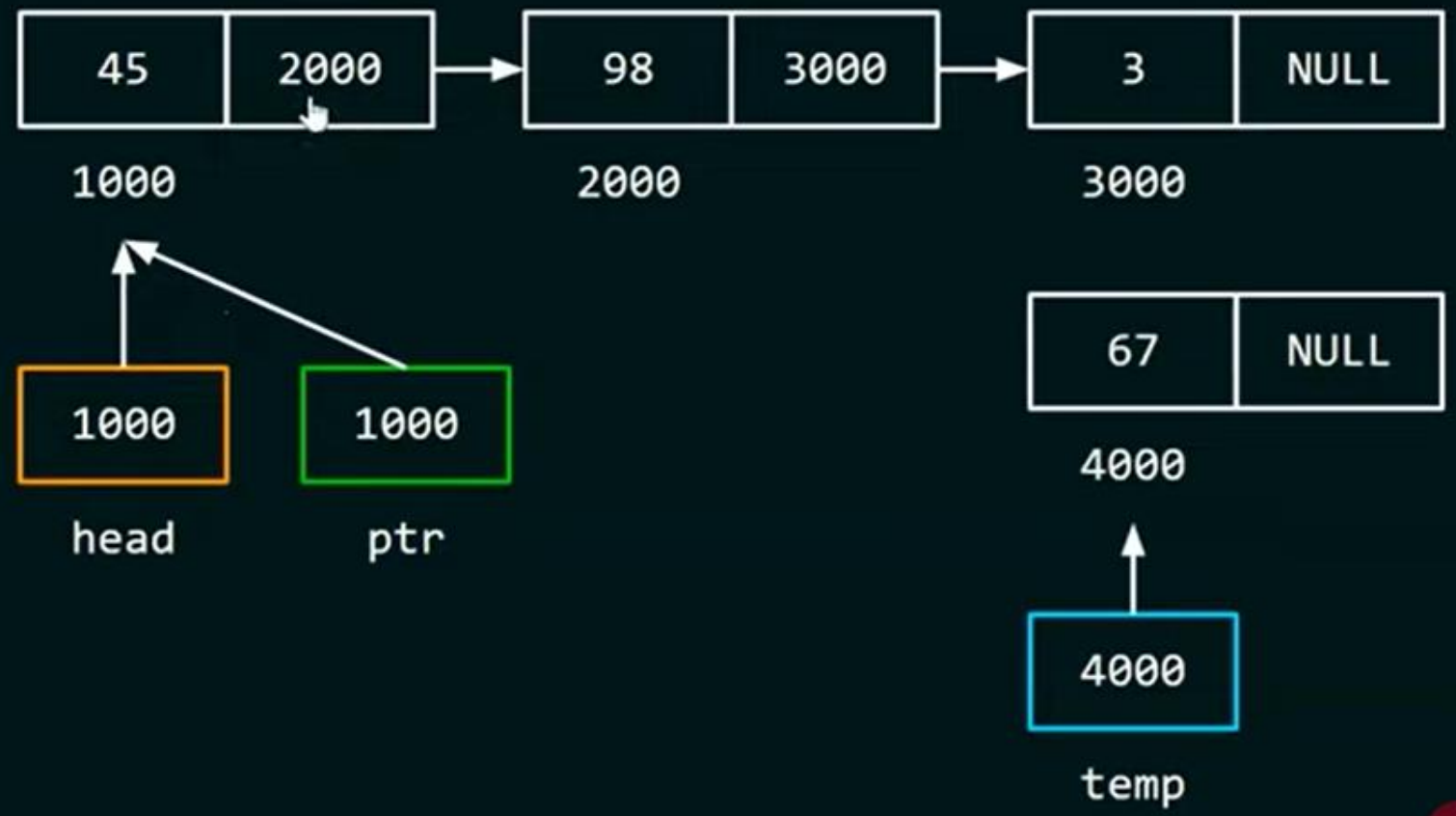
Program

```
void add_at_end(struct node *head, int data) {
    struct node *ptr, *temp;
    ptr = head;
    temp = (struct node*)malloc(sizeof(struct node));

    temp->data = data;
    temp->link = NULL;

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



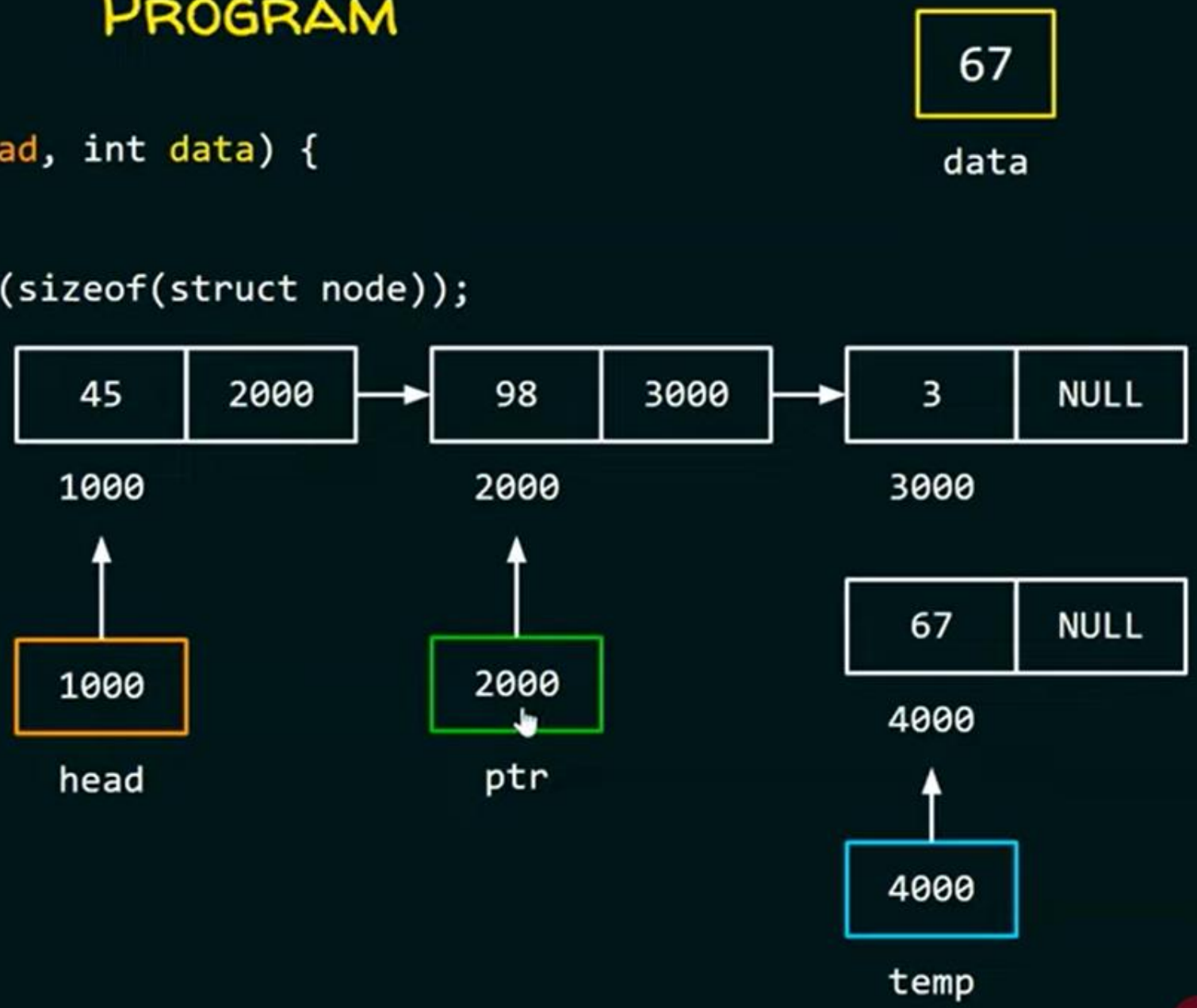
Program

```
void add_at_end(struct node *head, int data) {
    struct node *ptr, *temp;
    ptr = head;
    temp = (struct node*)malloc(sizeof(struct node));

    temp->data = data;
    temp->link = NULL;

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



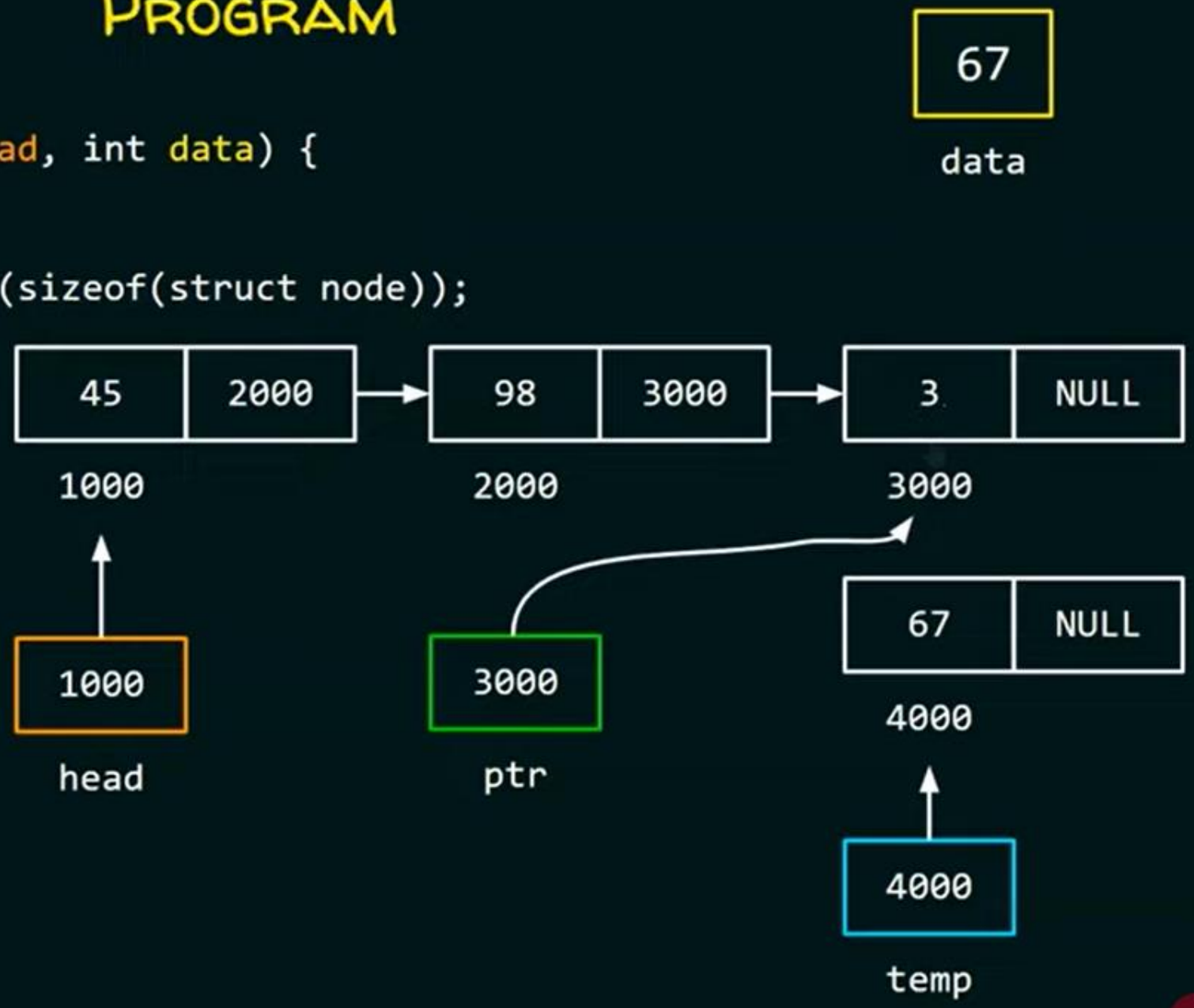
Program

```
void add_at_end(struct node *head, int data) {
    struct node *ptr, *temp;
    ptr = head;
    temp = (struct node*)malloc(sizeof(struct node));

    temp->data = data;
    temp->link = NULL;

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



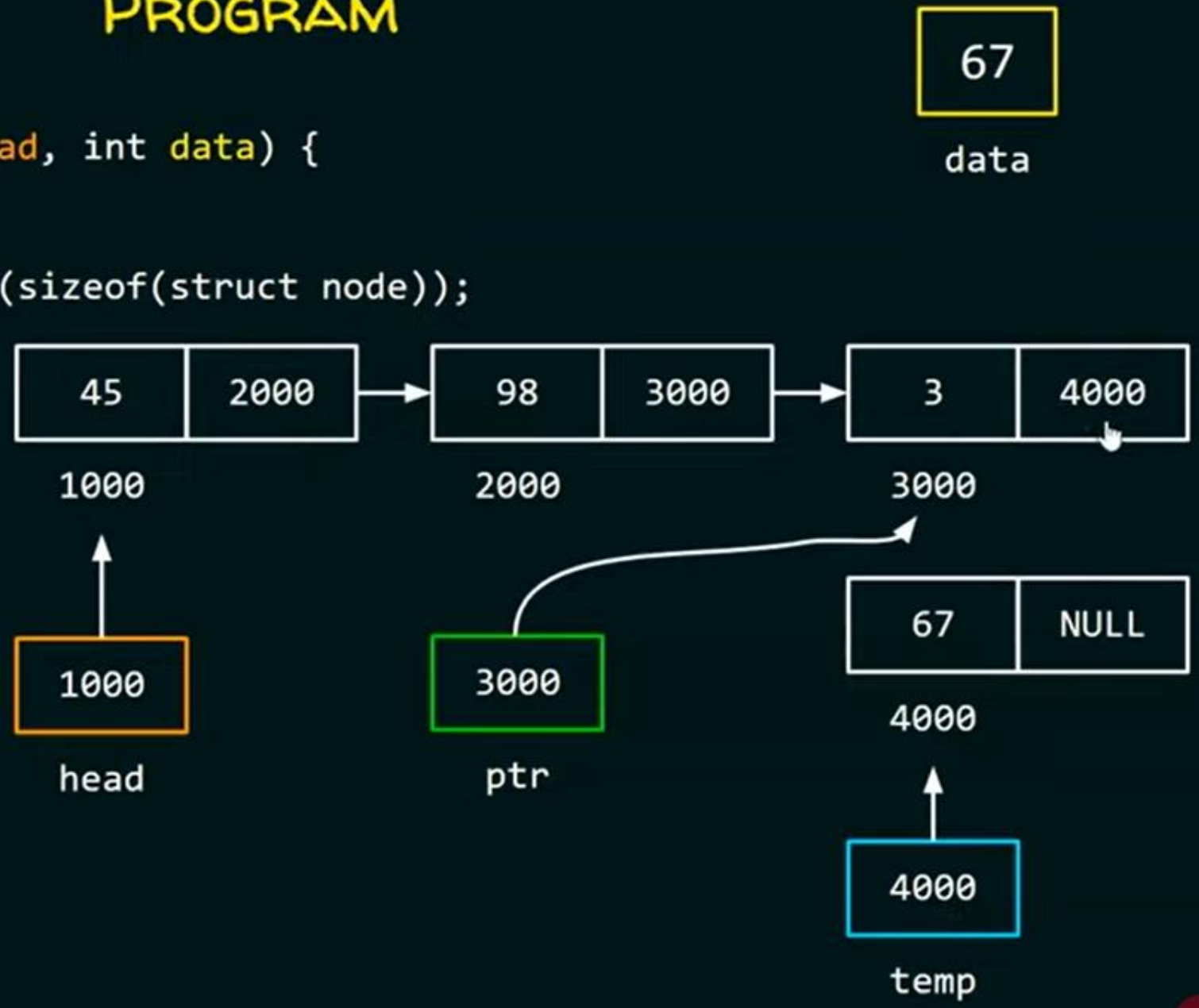
Program

```
void add_at_end(struct node *head, int data) {
    struct node *ptr, *temp;
    ptr = head;
    temp = (struct node*)malloc(sizeof(struct node));

    temp->data = data;
    temp->link = NULL;

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



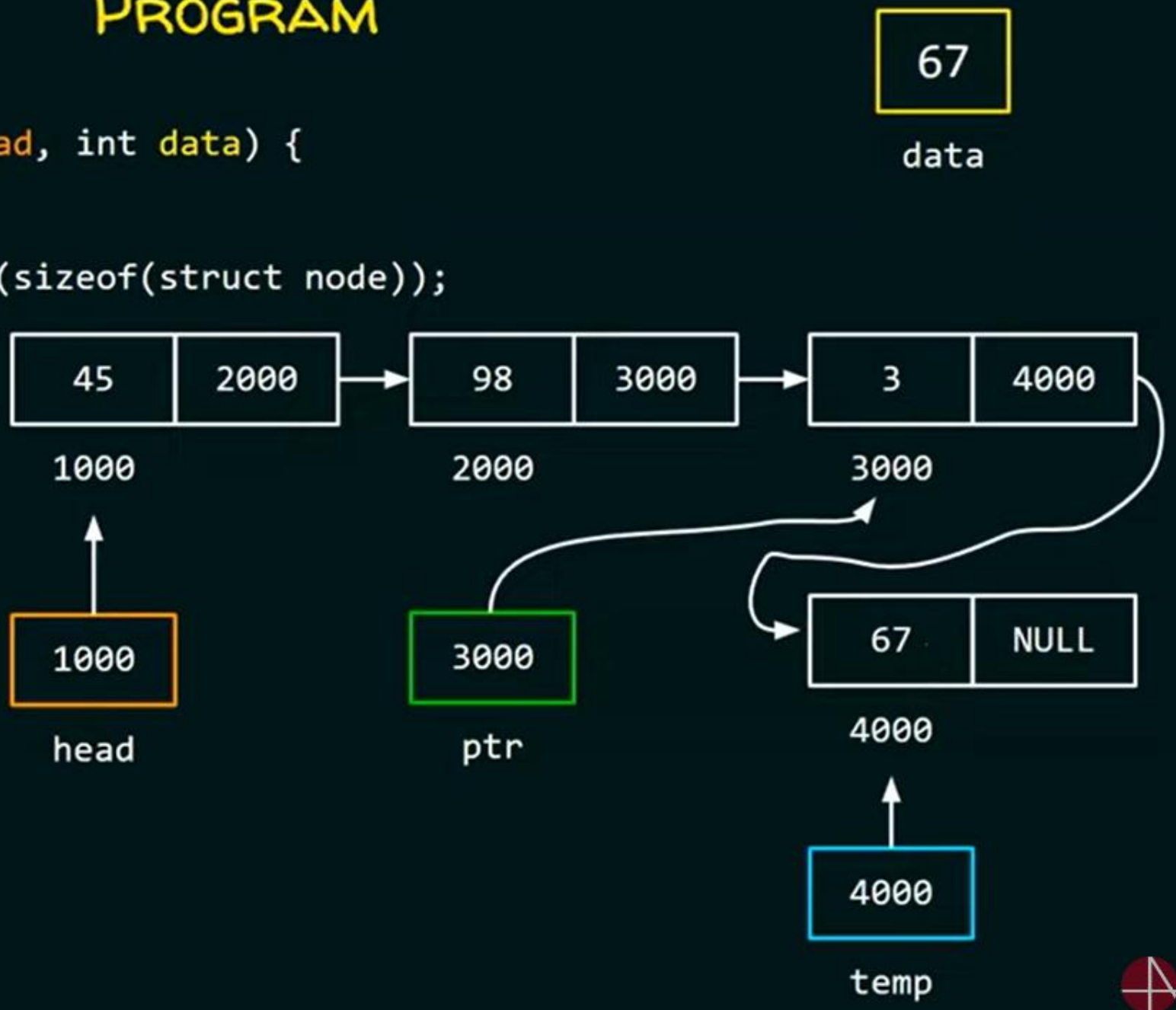
Program

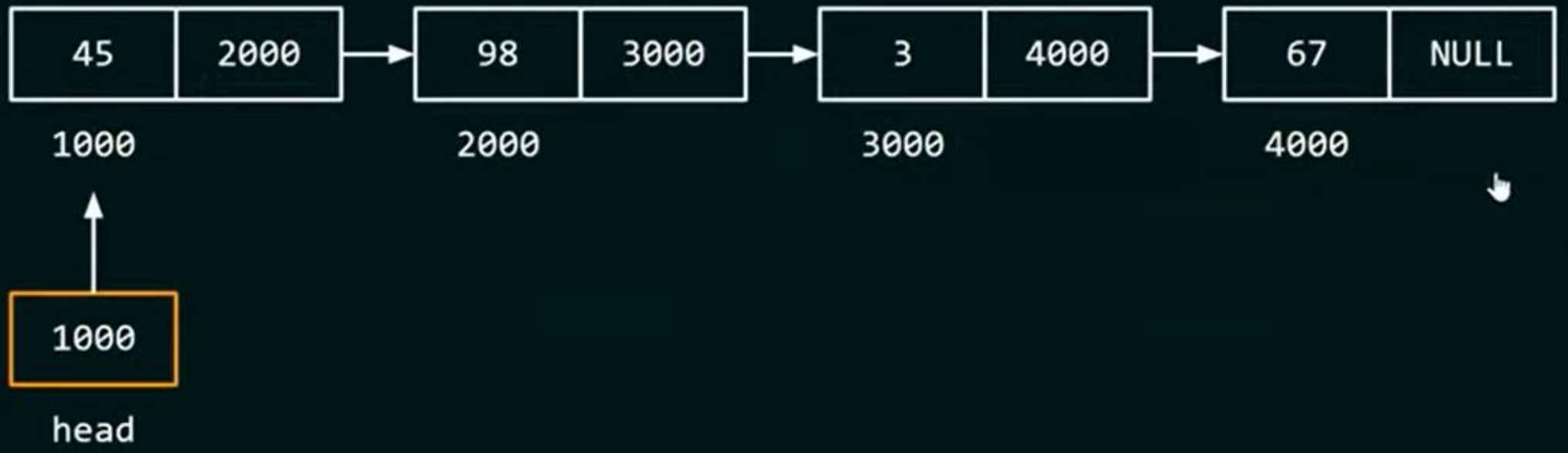
```
void add_at_end(struct node *head, int data) {
    struct node *ptr, *temp;
    ptr = head;
    temp = (struct node*)malloc(sizeof(struct node));

    temp->data = data;
    temp->link = NULL;

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



Program

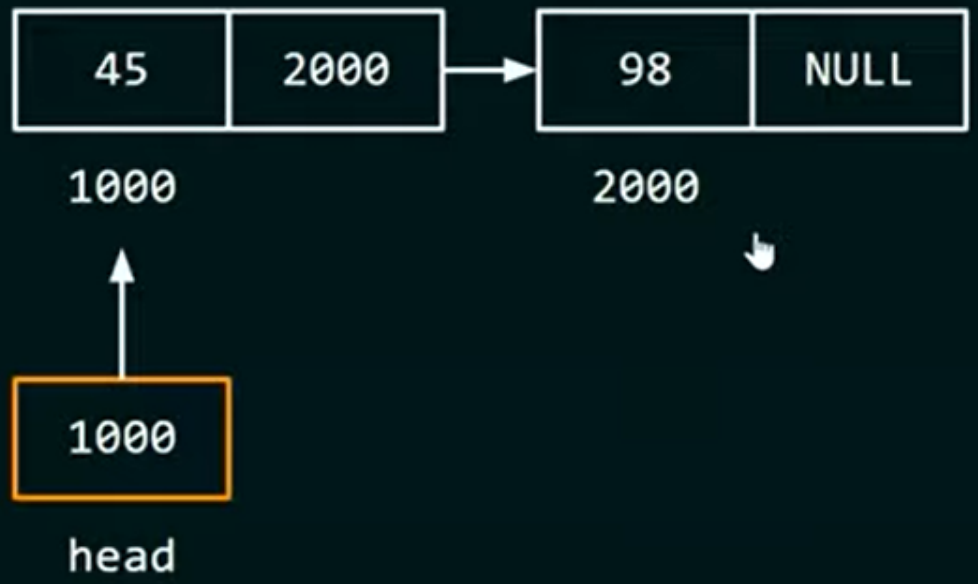
```
void add_at_end(struct node *head, int data) {
    struct node *ptr, *temp;
    ptr = head;
    temp = (struct node*)malloc(sizeof(struct node));

    temp->data = data;
    temp->link = NULL;

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

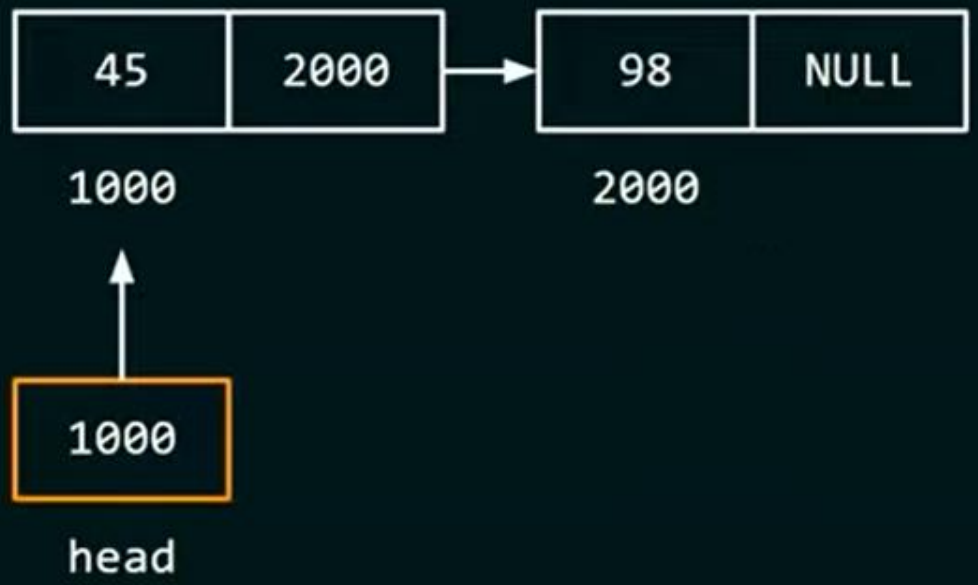


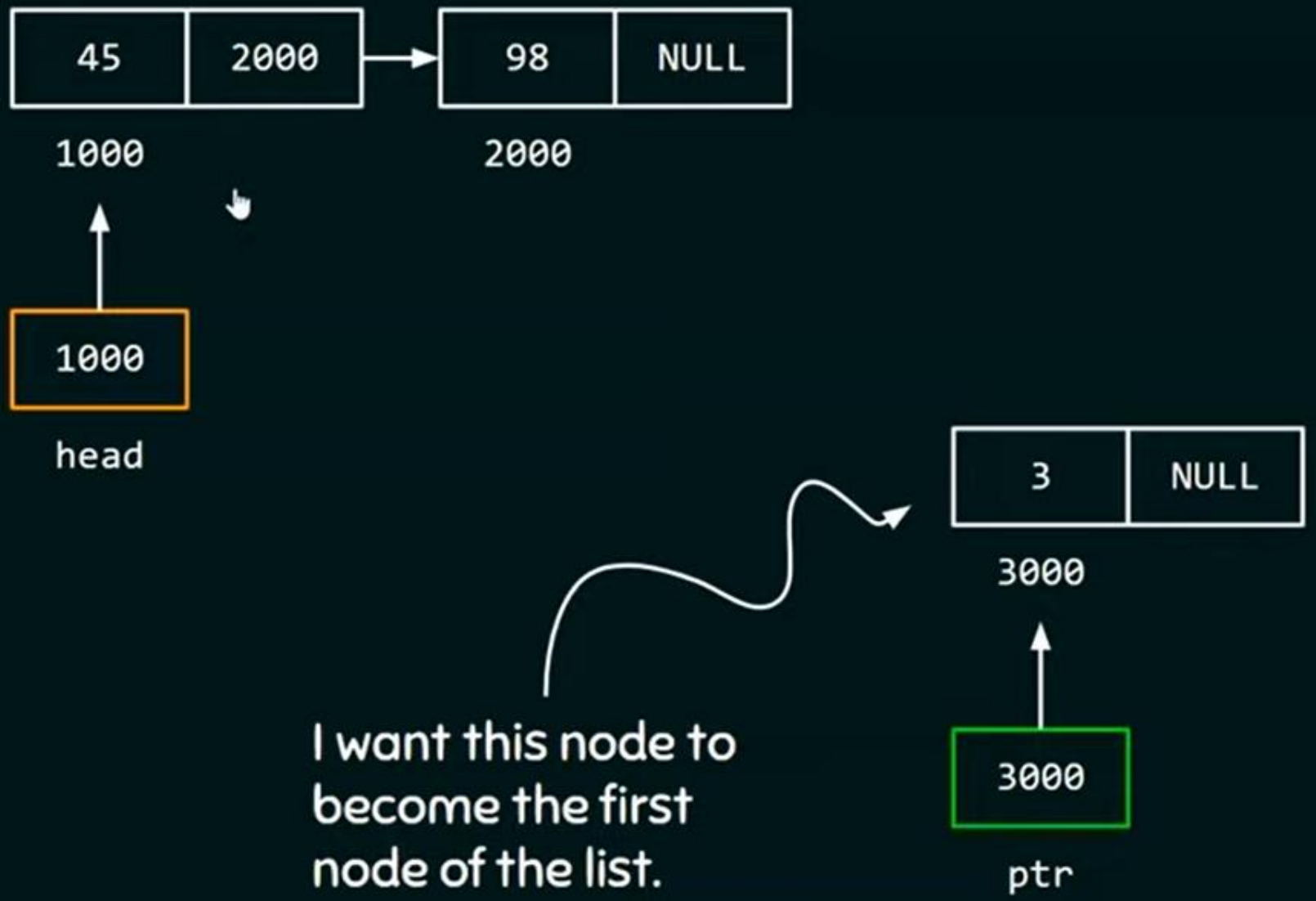


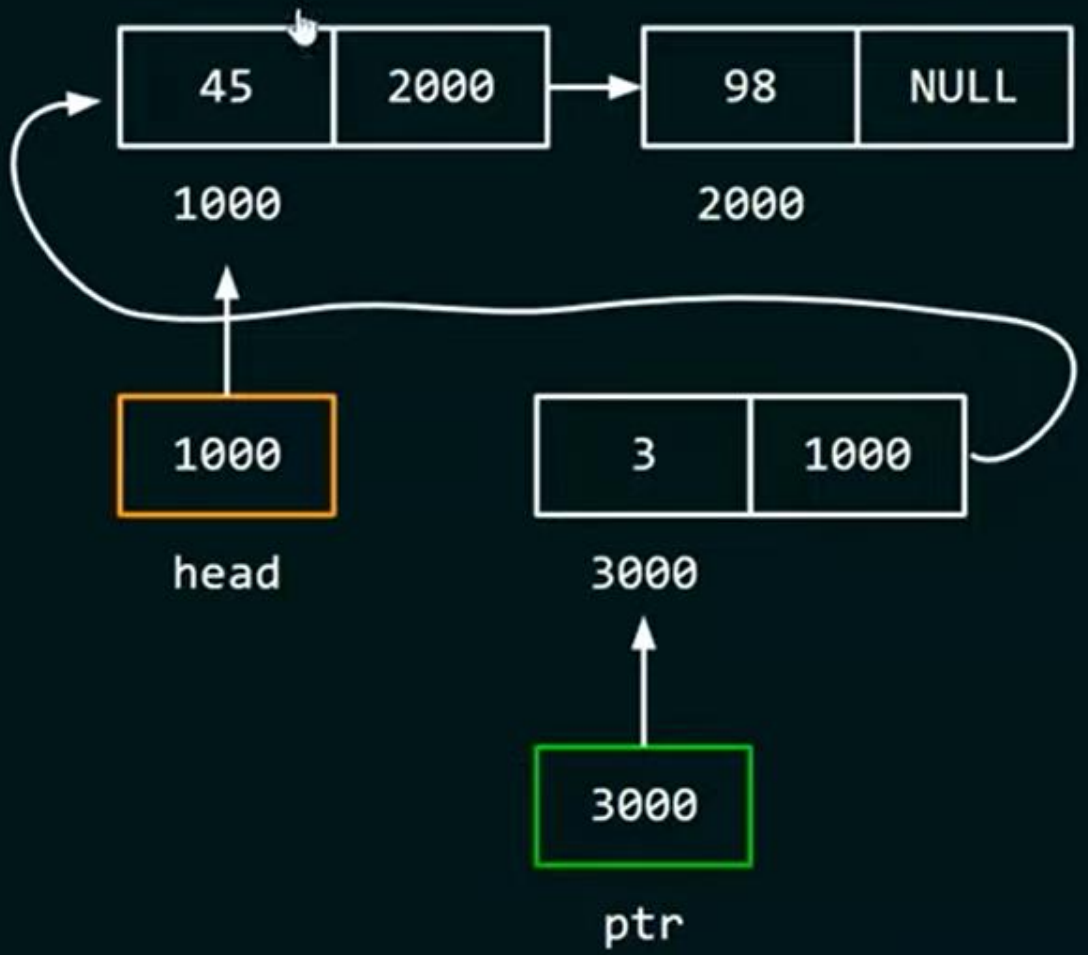
PROGRAMMING AND DATA STRUCTURES

Inserting Node at the Beginning of the List

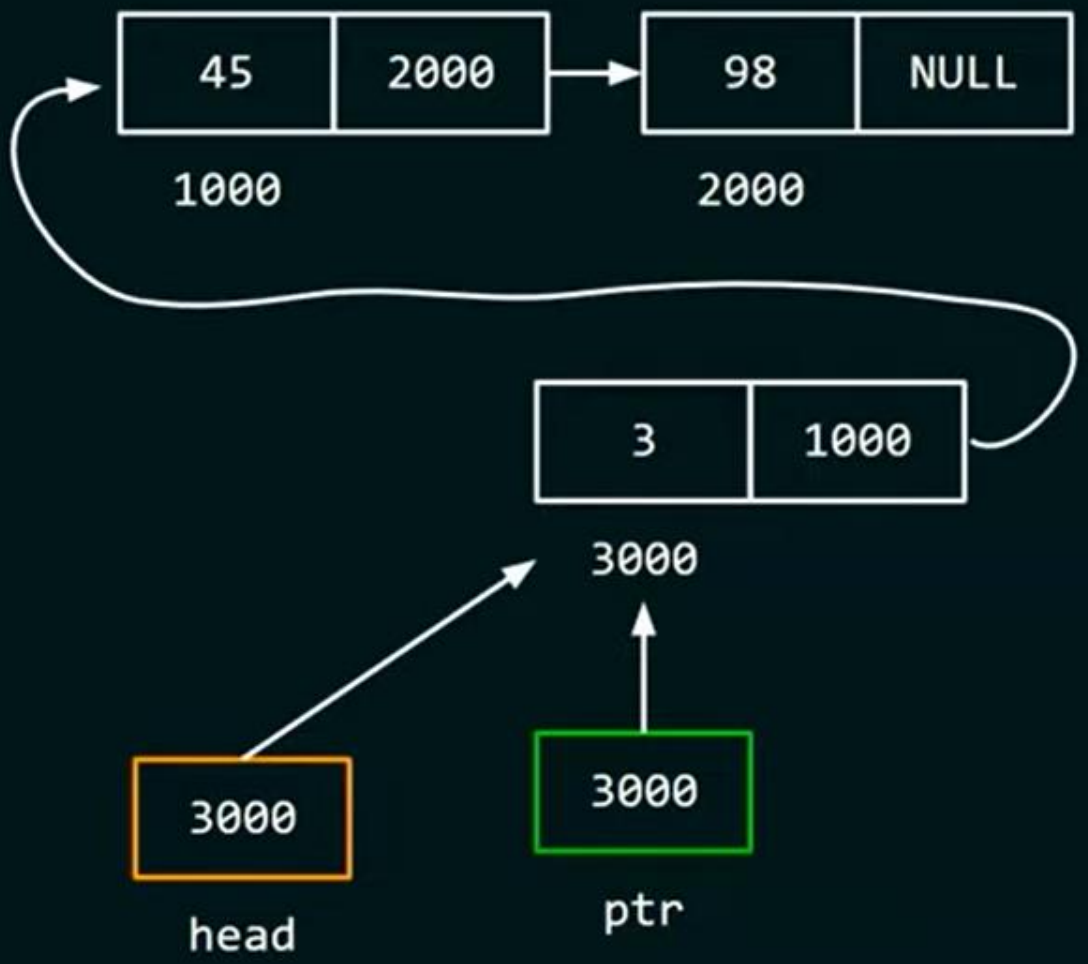









```
ptr -> link = head;
```




```
ptr -> link = head;

head = ptr;
```

Program – Main Function

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main() {
    struct node *head = malloc(sizeof(struct node));
    head->data = 45;
    head->link = NULL;

    struct node *ptr = malloc(sizeof(struct node));
    ptr->data = 98;
    ptr->link = NULL;

    head->link = ptr;

    int data = 3;

    head = add_beg(head, data);
    ptr = head;
    while(ptr != NULL)
    {
        printf("%d ", ptr->data);
        ptr = ptr->link;
    }

      return 0;
}
```

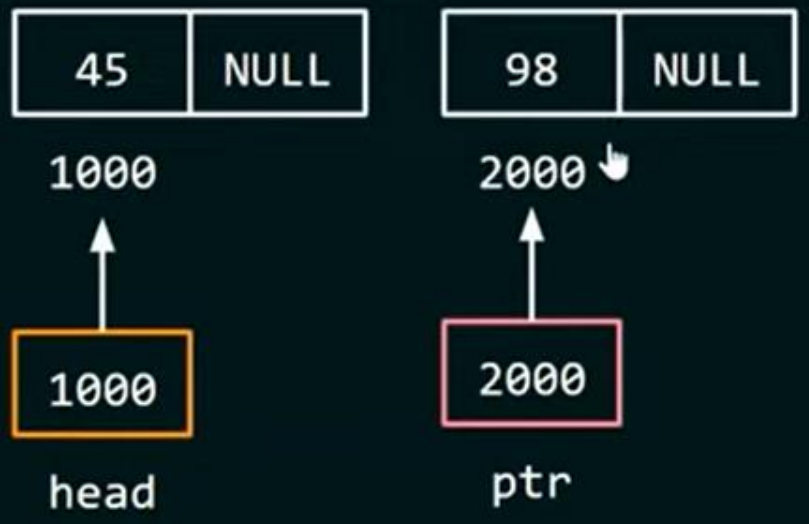
Program – Main Function

```c
#include <stdio.h>
#include <stdlib.h>

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


```c
int main() {
    struct node *head = malloc(sizeof(struct node));
    head->data = 45;
    head->link = NULL;

    struct node *ptr = malloc(sizeof(struct node));
    ptr->data = 98;
    ptr->link = NULL;

    head->link = ptr;

    int data = 3;

    head = add_beg(head, data);
    ptr = head;
    while(ptr != NULL)
    {
        printf("%d ", ptr->data);
        ptr = ptr->link;
    }

    return 0;
}
```

Program – Main Function

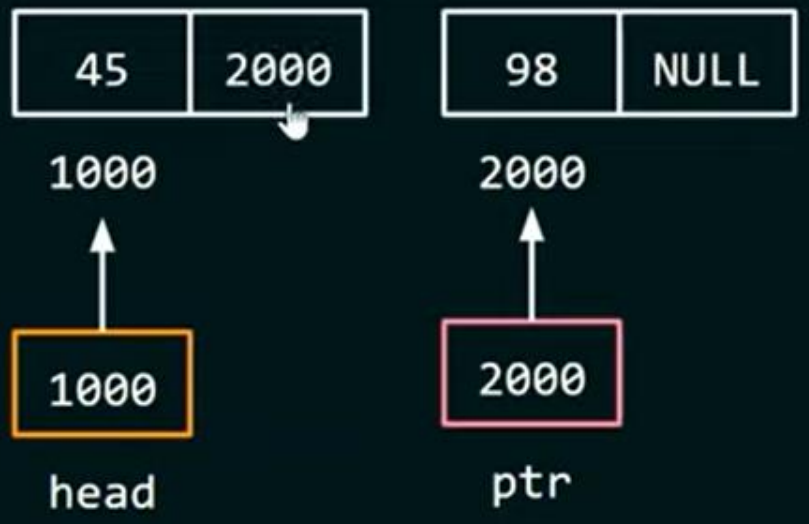
```c
#include <stdio.h>
#include <stdlib.h>

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




```c
int main() {
    struct node *head = malloc(sizeof(struct node));
    head->data = 45;
    head->link = NULL;

    struct node *ptr = malloc(sizeof(struct node));
    ptr->data = 98;
    ptr->link = NULL;

    head->link = ptr;

    int data = 3;

    head = add_beg(head, data);
    ptr = head;
    while(ptr != NULL)
    {
        printf("%d ", ptr->data);
        ptr = ptr->link;
    }

    return 0;
}
```

Program – Main Function

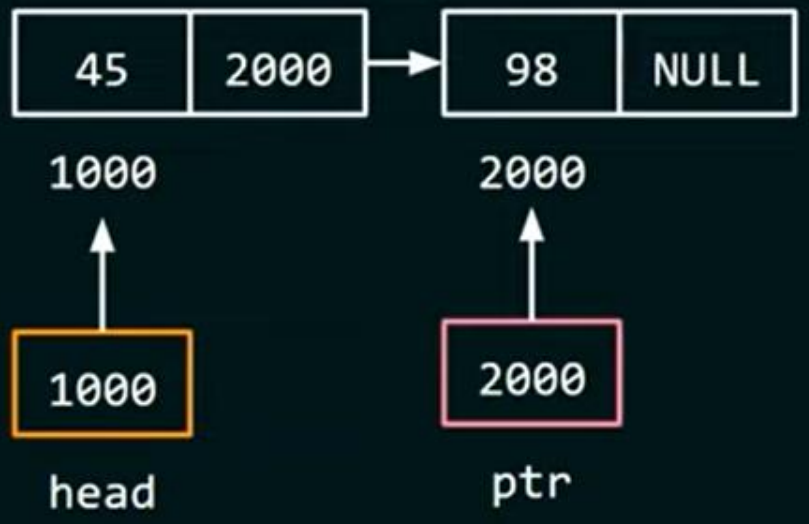
```c
#include <stdio.h>
#include <stdlib.h>

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




```c
int main() {
    struct node *head = malloc(sizeof(struct node));
    head->data = 45;
    head->link = NULL;

    struct node *ptr = malloc(sizeof(struct node));
    ptr->data = 98;
    ptr->link = NULL;

    head->link = ptr;

    int data = 3;

    head = add_beg(head, data);
    ptr = head;
    while(ptr != NULL)
    {
        printf("%d ", ptr->data);
        ptr = ptr->link;
    }

     return 0;
}
```

Program – Main Function

```c
#include <stdio.h>
#include <stdlib.h>

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

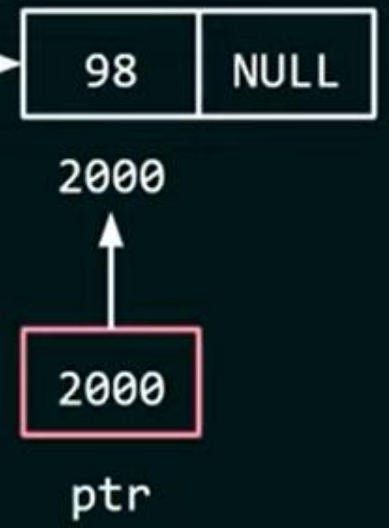



```c
int main() {
    struct node *head = malloc(sizeof(struct node));
    head->data = 45;
    head->link = NULL;

    struct node *ptr = malloc(sizeof(struct node));
    ptr->data = 98;
    ptr->link = NULL;

    head->link = ptr;

    int data = 3;

    head = add_beg(head, data);
    ptr = head;
    while(ptr != NULL)
    {
        printf("%d ", ptr->data);
        ptr = ptr->link;
    }

    return 0;
}
```

PROGRAM – MAIN FUNCTION

```
h>
.h>

*link;
```

```
int main() {
    struct node *head = malloc(sizeof(struct node));
    head->data = 45;
    head->link = NULL;

    struct node *ptr = malloc(sizeof(struct node));
    ptr->data = 98;
    ptr->link = NULL;

    head->link = ptr;

    int data = 3;

    head = add_beg(head, data);
    ptr = head;
    while(ptr != NULL)
    {
        printf("%d ", ptr->data);
        ptr = ptr->link;
    }

     return 0;
}
```

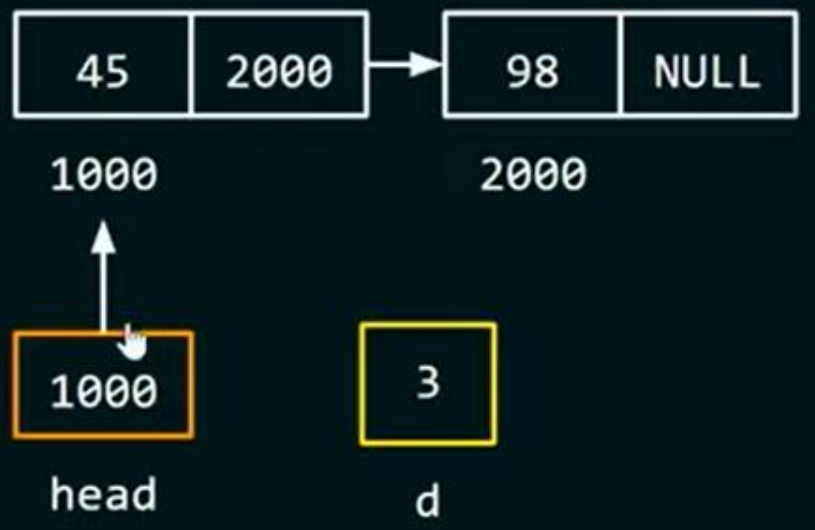


Adding the node at the beginning of the list

```
struct node* add_beg(struct node* head, int d)
{
    struct node *ptr = malloc(sizeof(struct node));
    ptr->data = d;
    ptr->link = NULL;

    ptr->link = head;
    head = ptr;
    return head;
}
```

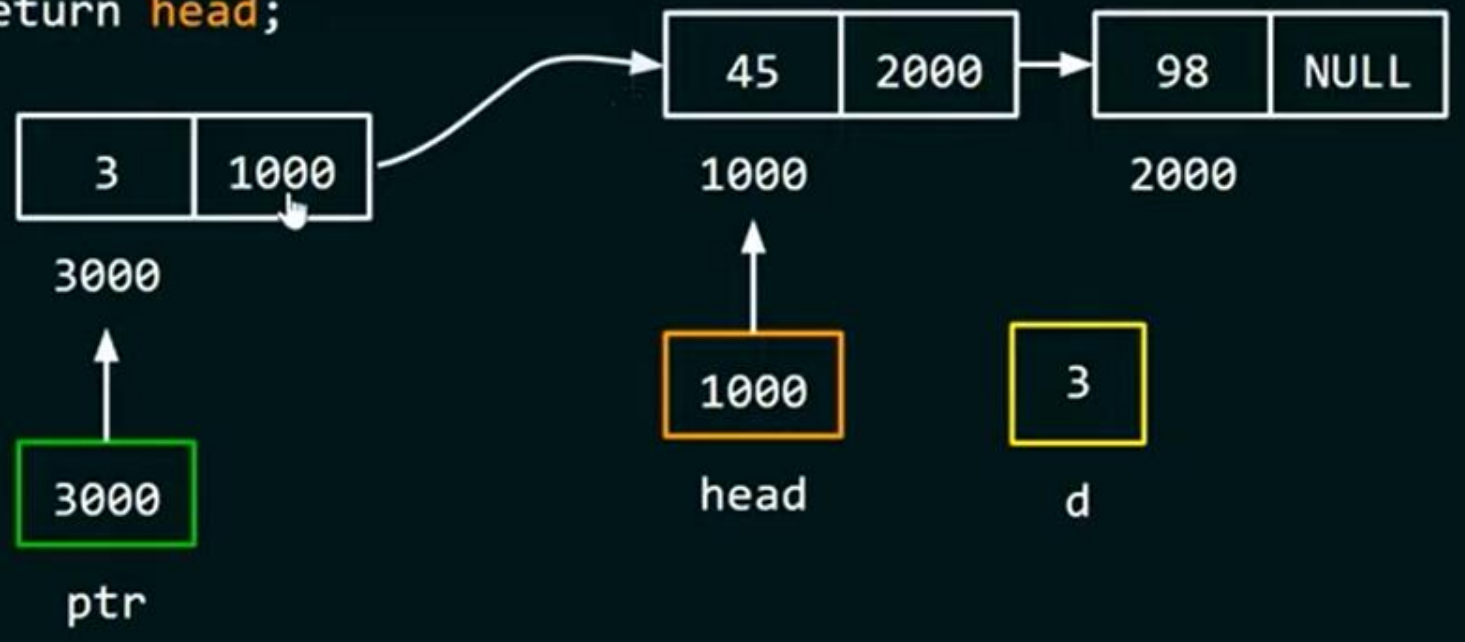
Adding the node at the beginning of the list

```
struct node* add_beg(struct node* head, int d)
{
    struct node *ptr = malloc(sizeof(struct node));
    ptr->data = d;
    ptr->link = NULL;

    ptr->link = head;
    head = ptr;
    return head;
}
```



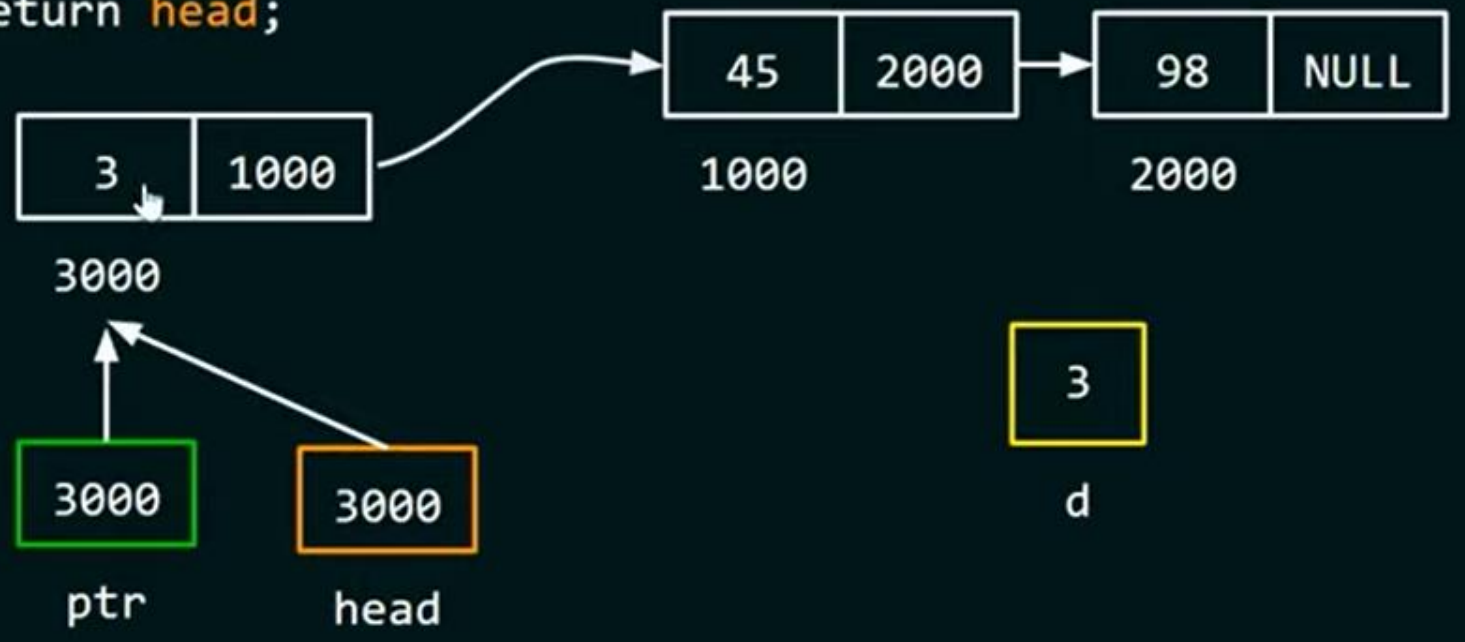
Adding the node at the beginning of the list

```
struct node* add_beg(struct node* head, int d)
{
    struct node *ptr = malloc(sizeof(struct node));
    ptr->data = d;
    ptr->link = NULL;

    ptr->link = head;
    head = ptr;
    return head;
}
```



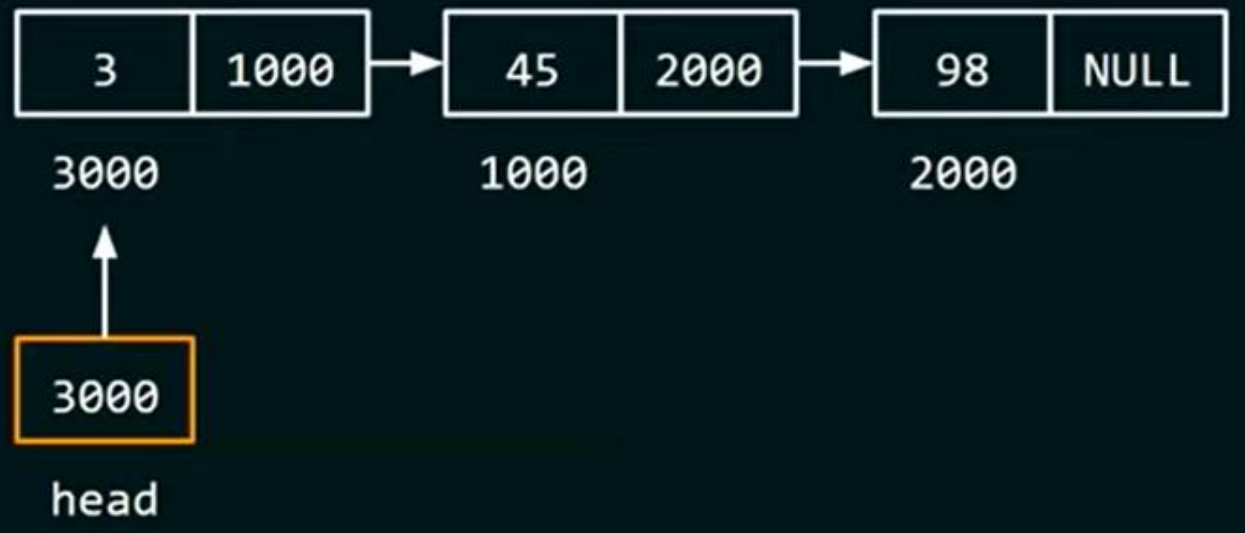
Adding the node at the beginning of the list

```
struct node* add_beg(struct node* head, int d)
{
    struct node *ptr = malloc(sizeof(struct node));
    ptr->data = d;
    ptr->link = NULL;

    ptr->link = head;
    head = ptr;
    return head;
}
```



Adding the node at the beginning of the list

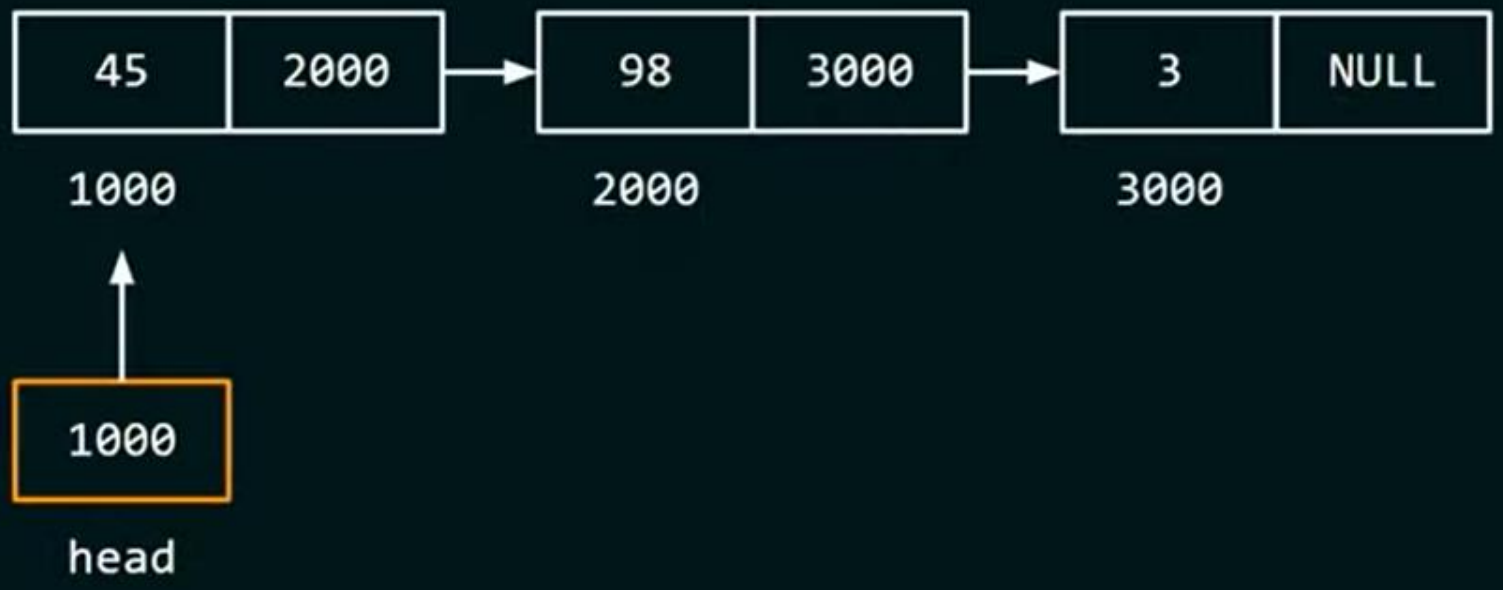
```
struct node* add_beg(struct node* head, int d)
{
    struct node *ptr = malloc(sizeof(struct node));
    ptr->data = d;
    ptr->link = NULL;

    ptr->link = head;
    head = ptr;
    return head;
}
```



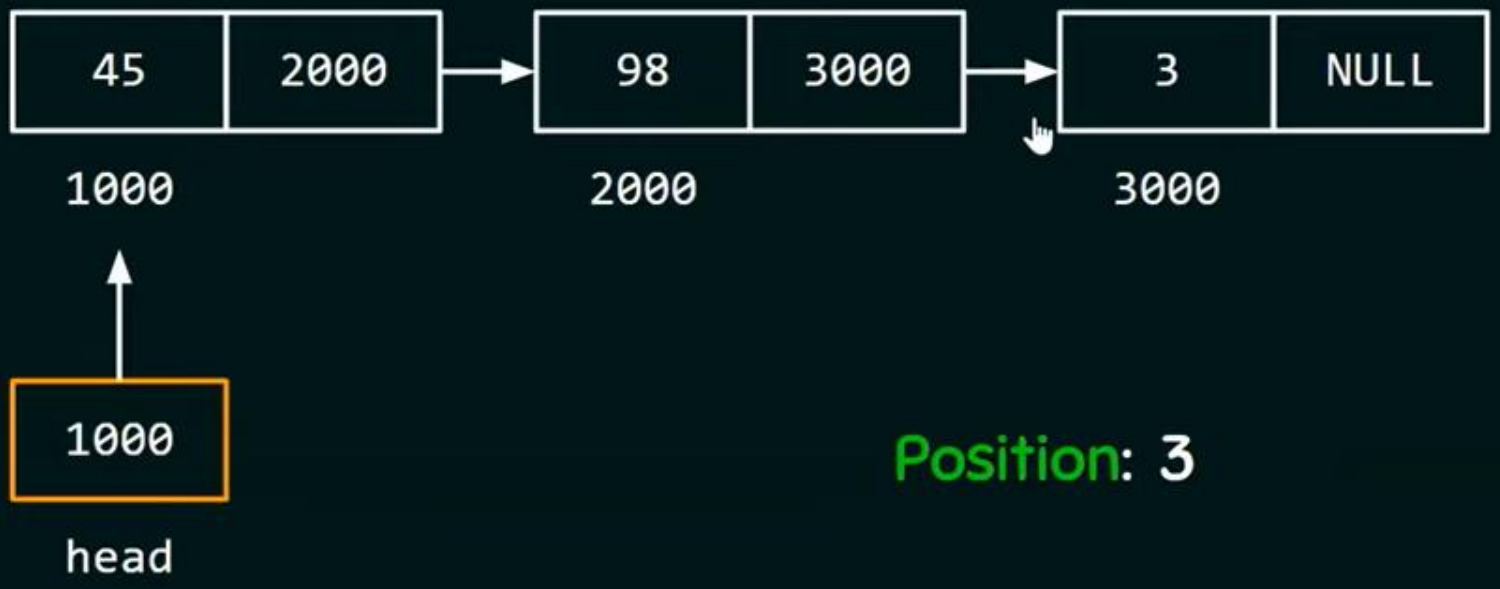
PROGRAMMING AND DATA STRUCTURES

Inserting a Node at Certain Position

Assumption: Position where the new node needs to be inserted is given.



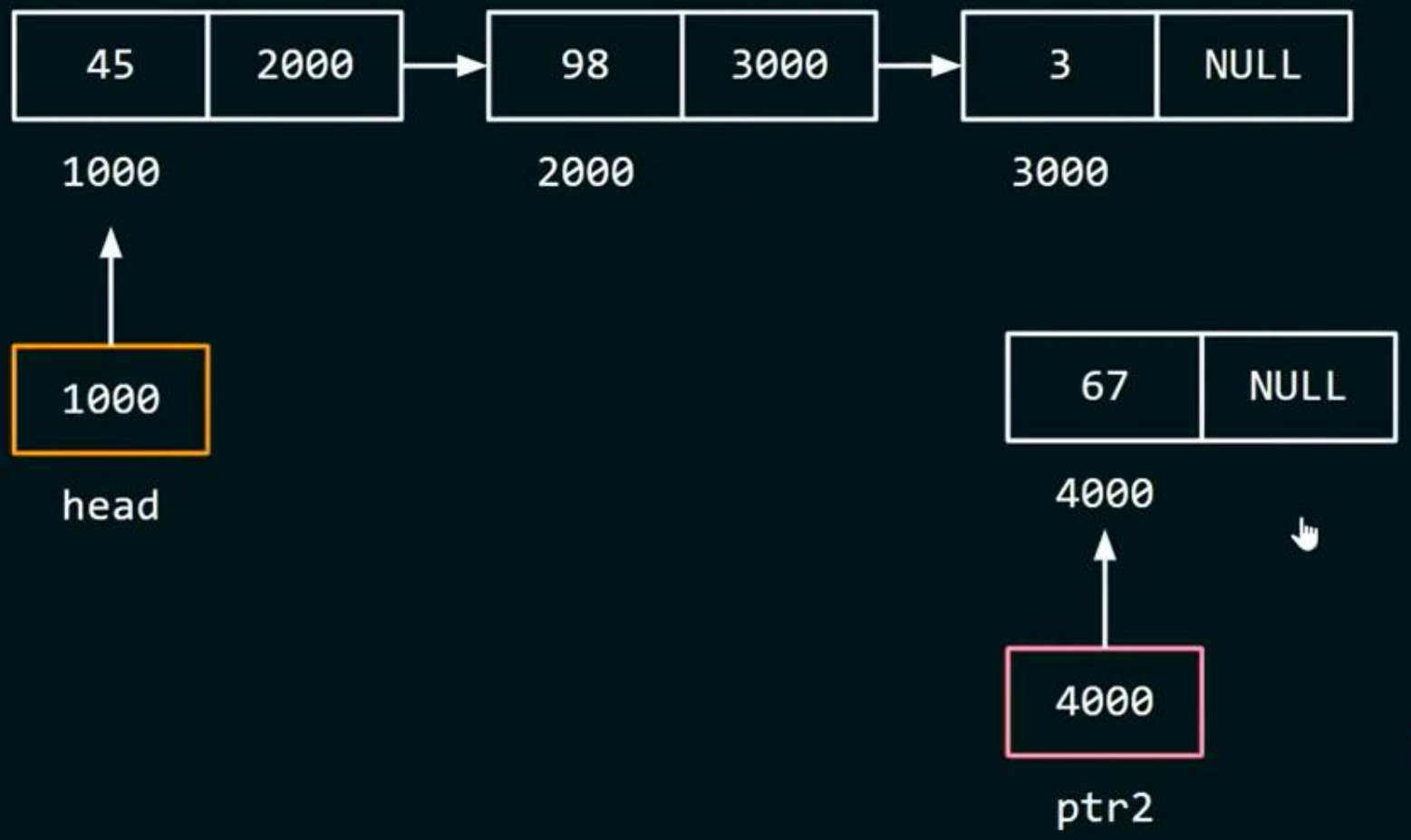
Assumption: Position where the new node needs to be inserted is given.



Position: 3

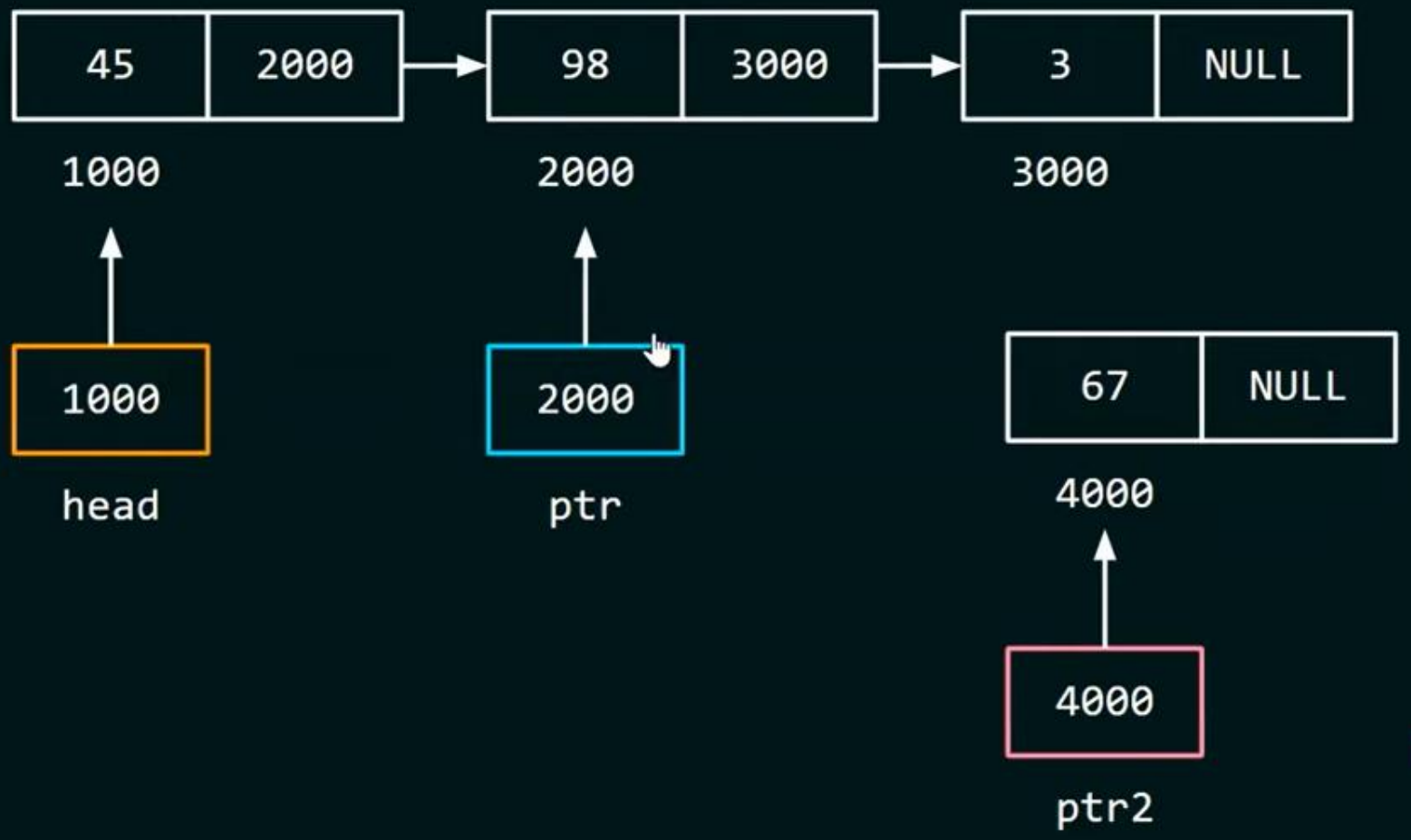
Step 1: Create a new node

Assumption: Position where the new node needs to be inserted is given.



Step 2: Traversal

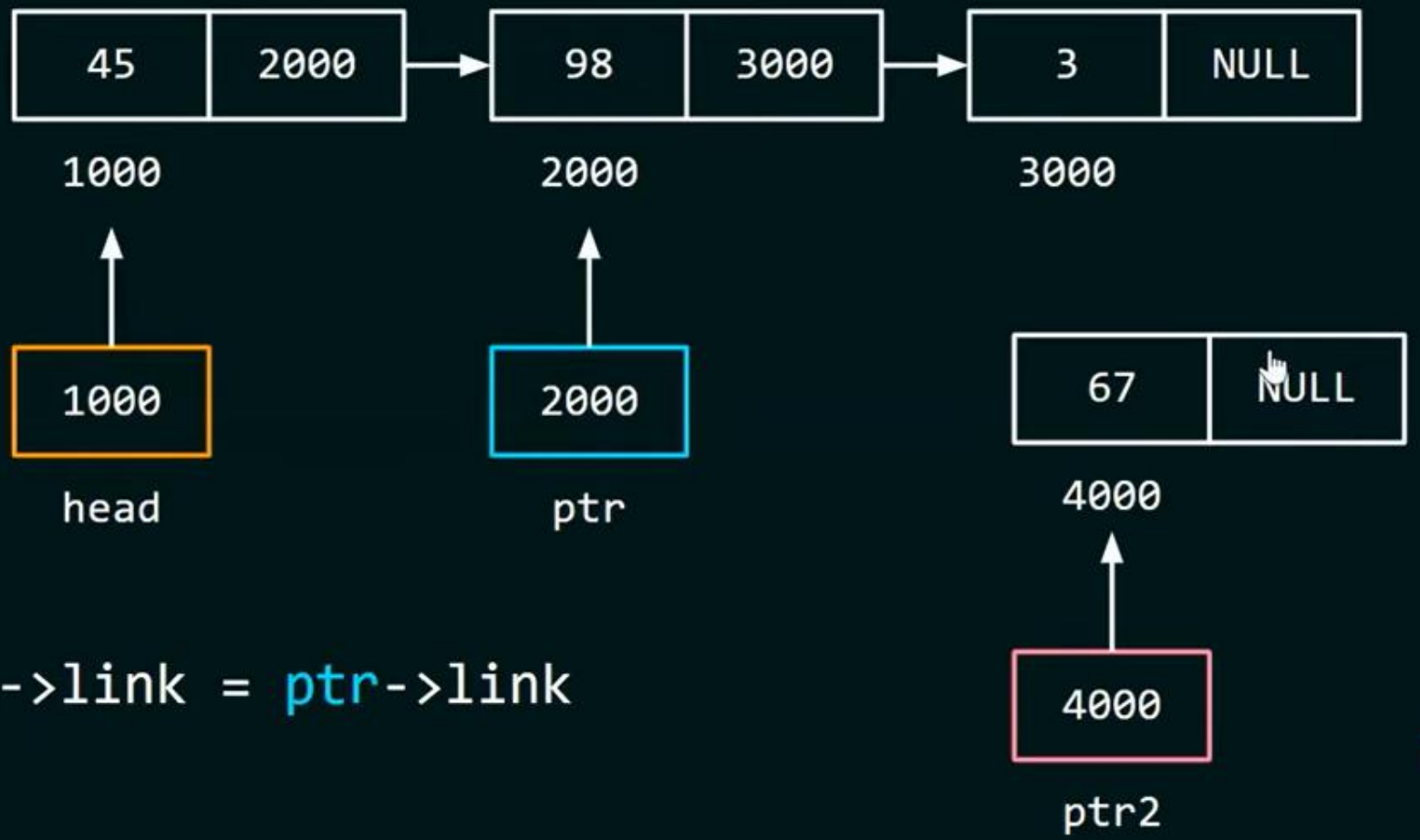
Assumption: Position where the new node needs to be inserted is given.



Position: 3

Step 3: Update the links

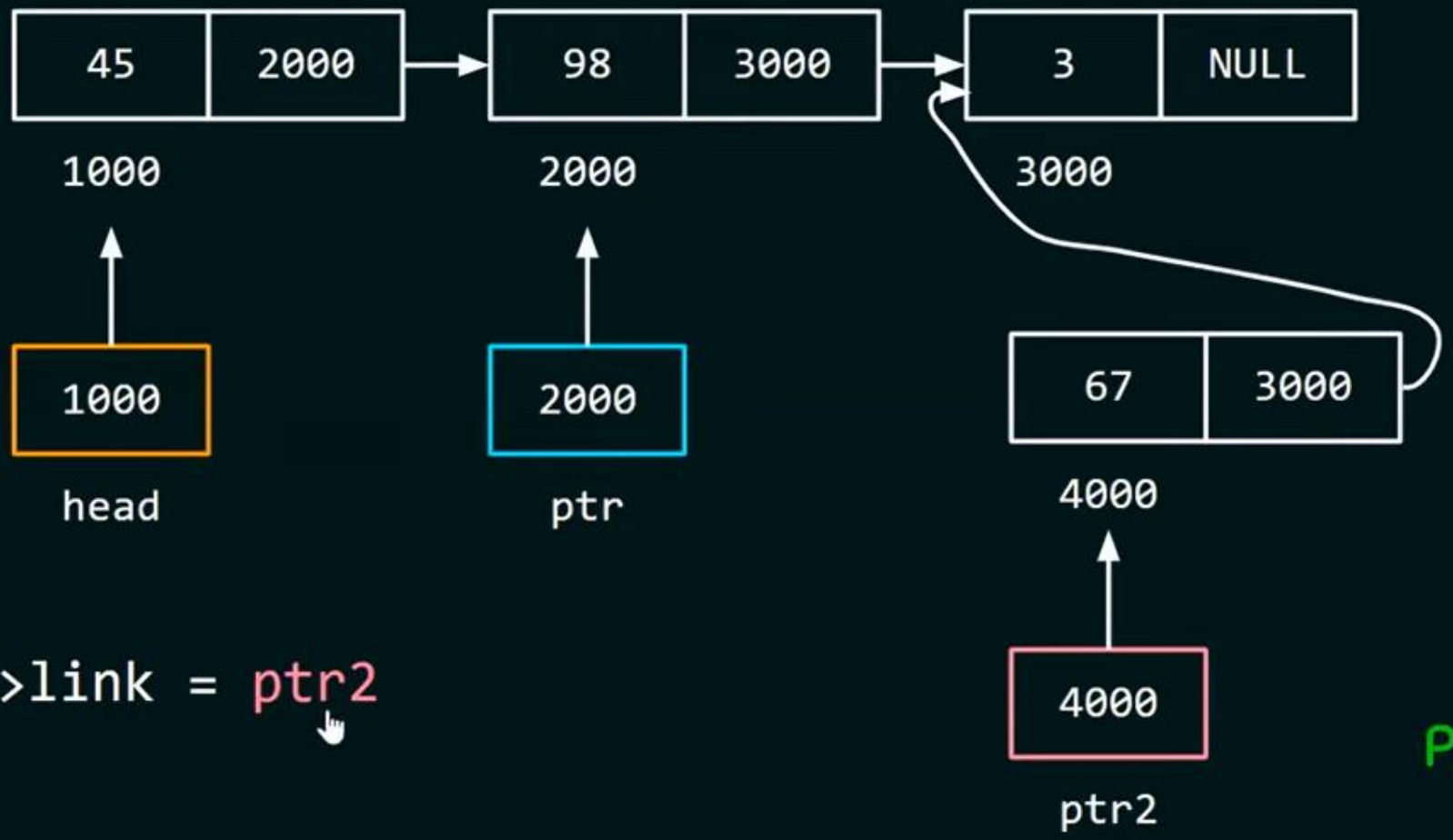
Assumption: Position where the new node needs to be inserted is given.




```
ptr2->link = ptr->link
```

Position: 3

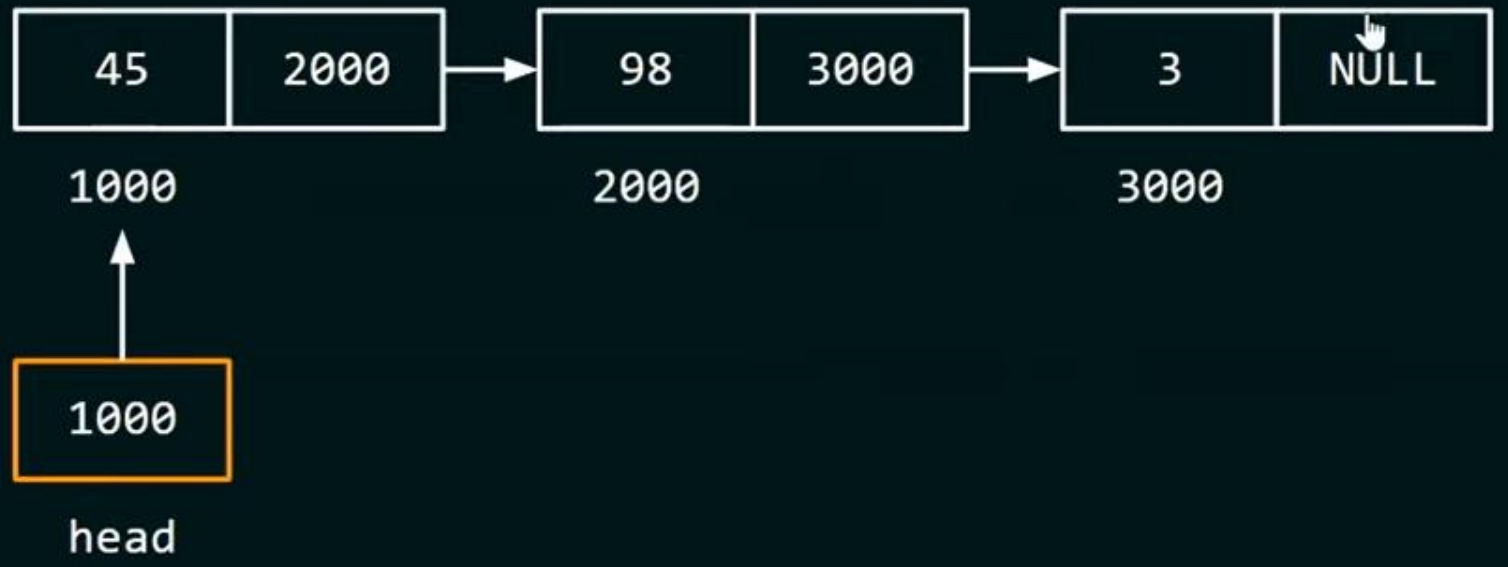
Assumption: Position where the new node needs to be inserted is given.

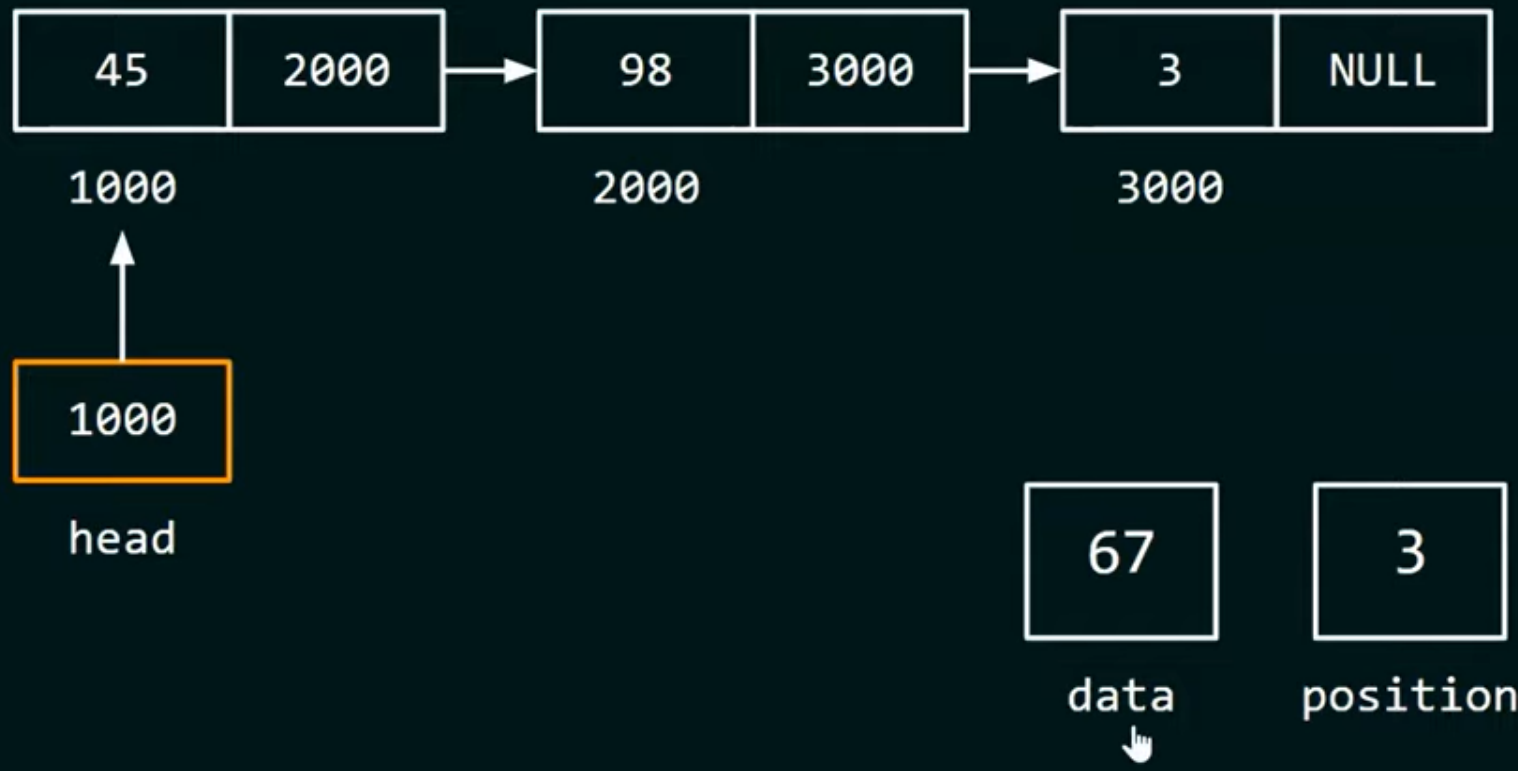



```
ptr->link = ptr2
```

Position: 3

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main()
{
    struct node *head = malloc(sizeof(struct node));
    head->data = 45;
    head->link = NULL;

    add_at_end(head, 98);
    add_at_end(head, 3);

    int data = 67, position = 3;

    add_at_pos(head, data, position);
    struct node *ptr = head;

    while(ptr != NULL)
    {
        printf("%d ", ptr->data);
        ptr = ptr->link;
    }
    return 0;
}
```

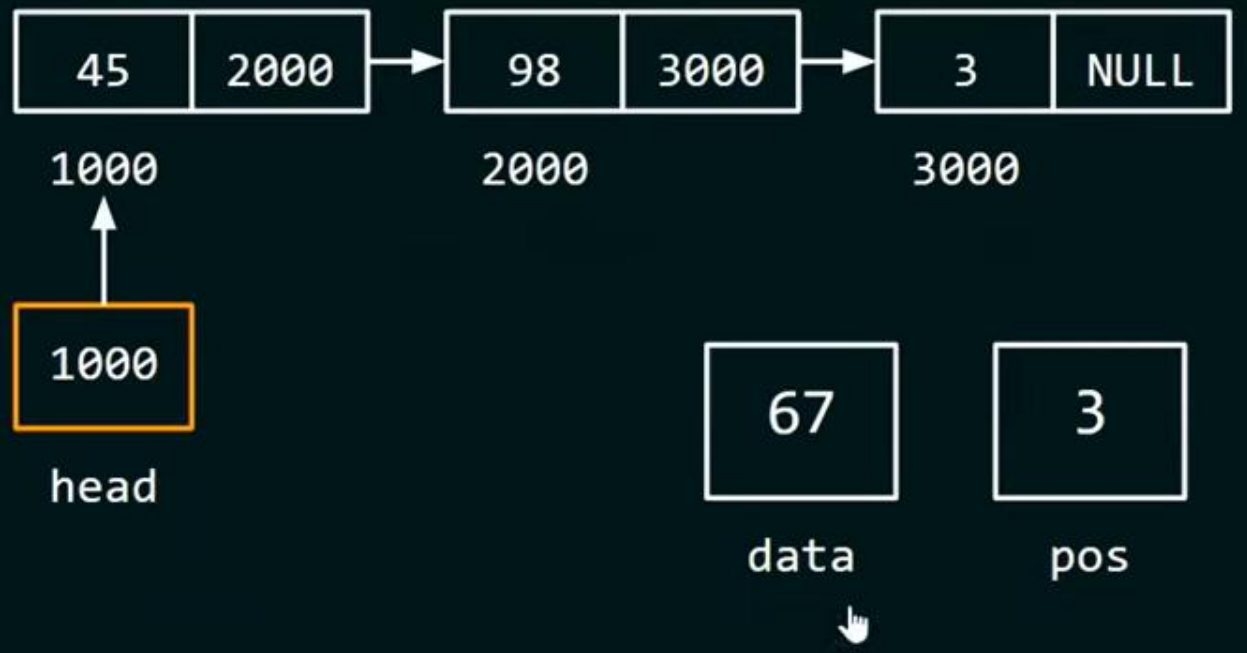




```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main()
{
    struct node *head = malloc(sizeof(struct node));
    head->data = 45;
    head->link = NULL;

    add_at_end(head, 98);
    add_at_end(head, 3);

    int data = 67, position = 3;

    add_at_pos(head, data, position);
    struct node *ptr = head;

    while(ptr != NULL)
    {
        printf("%d ", ptr->data);
        ptr = ptr->link;
    }
    return 0;
}
```

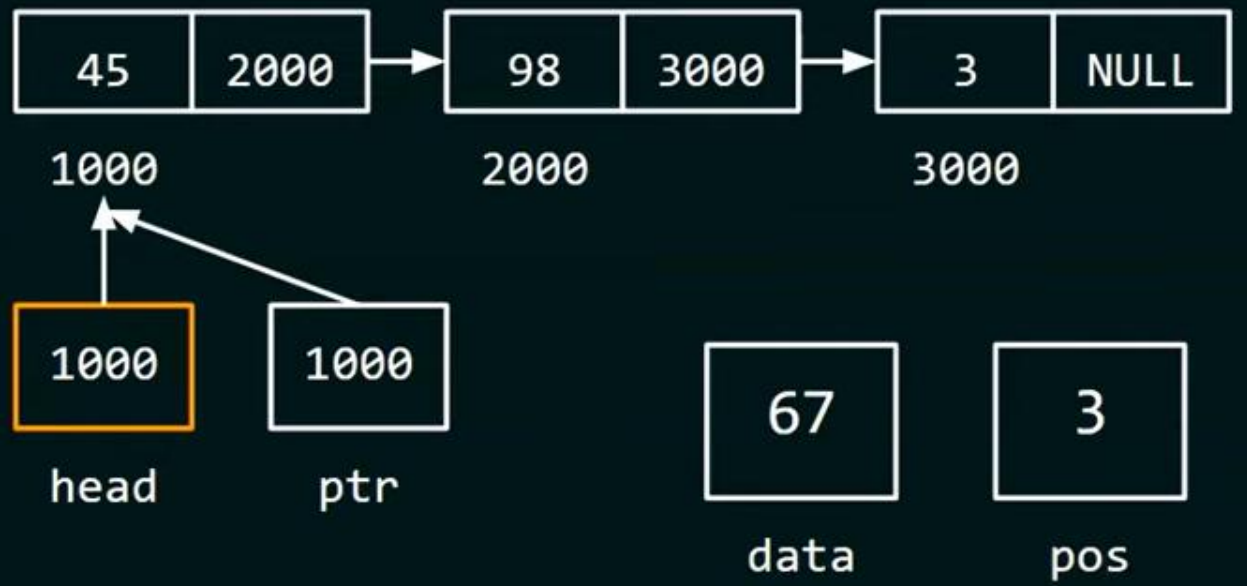

Add at position function

```
void add_at_pos(struct node* head, int data, int pos)
{
    struct node *ptr = head;
    struct node *ptr2 = malloc(sizeof(struct node));
    ptr2->data = data;
    ptr2->link = NULL;

    pos--;
    while(pos != 1)
    {
        ptr = ptr->link;
        pos--;

    }
    ptr2->link = ptr->link;
    ptr->link = ptr2;
}
```



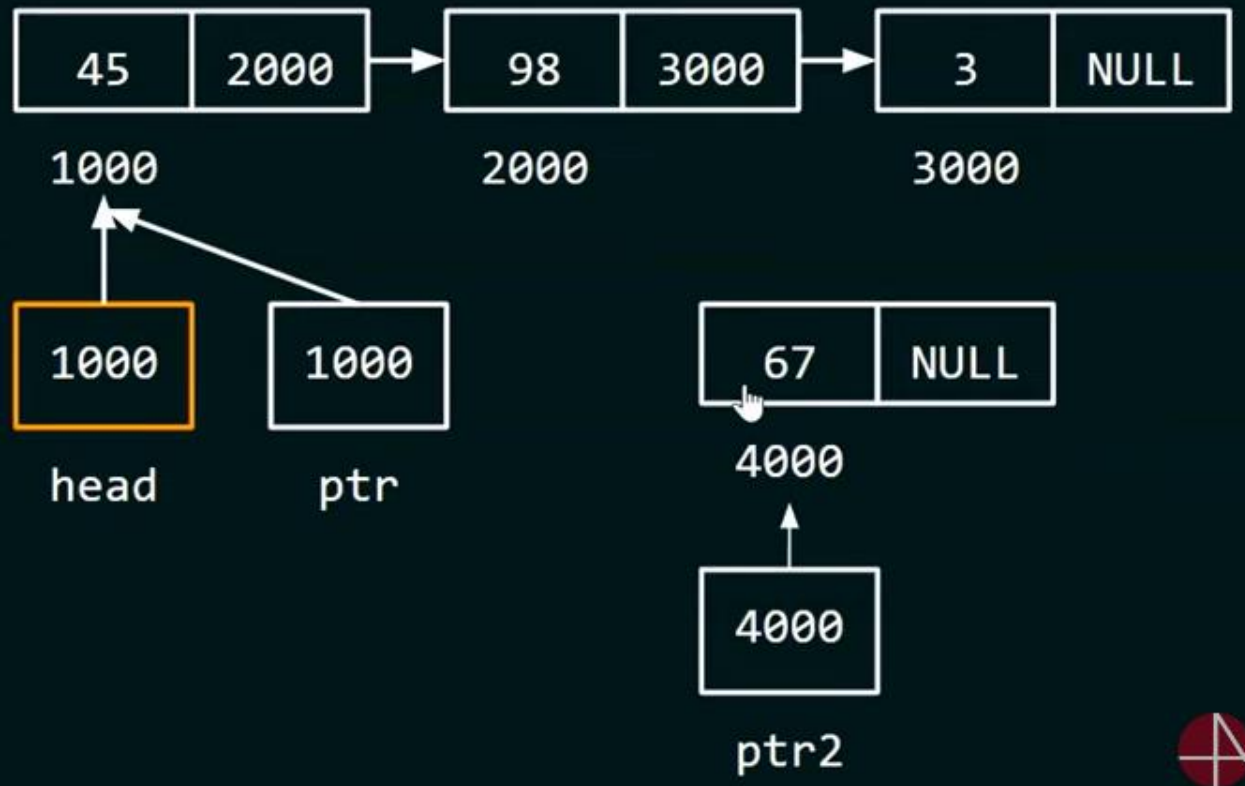
Add at position function

```
void add_at_pos(struct node* head, int data, int pos)
{
    struct node *ptr = head;
    struct node *ptr2 = malloc(sizeof(struct node));
    ptr2->data = data;
    ptr2->link = NULL;

    pos--;
    while(pos != 1)
    {
        ptr = ptr->link;
        pos--;

    }
    ptr2->link = ptr->link;
    ptr->link = ptr2;
}
```



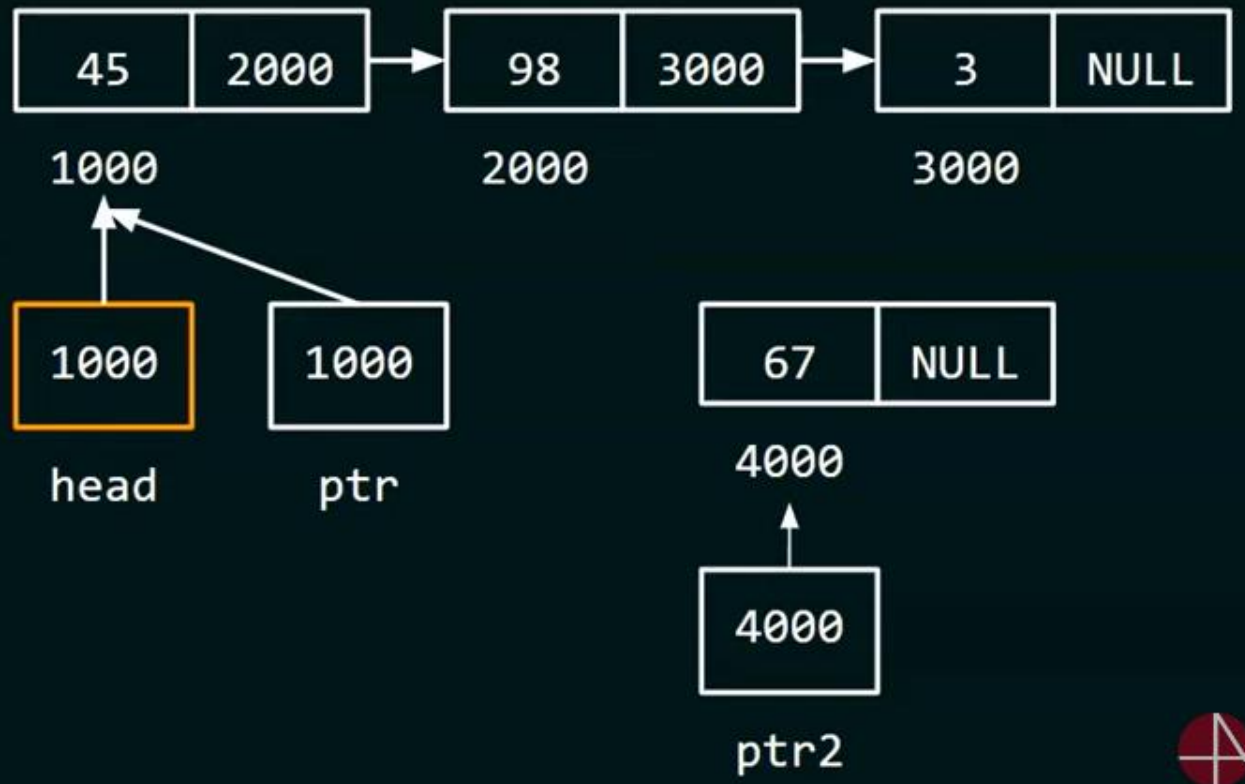
Add at position function

```
void add_at_pos(struct node* head, int data, int pos)
{
    struct node *ptr = head;
    struct node *ptr2 = malloc(sizeof(struct node));
    ptr2->data = data;
    ptr2->link = NULL;

    pos--;
    while(pos != 1)
    {
        ptr = ptr->link;
        pos--;

    }
    ptr2->link = ptr->link;
    ptr->link = ptr2;
}
```



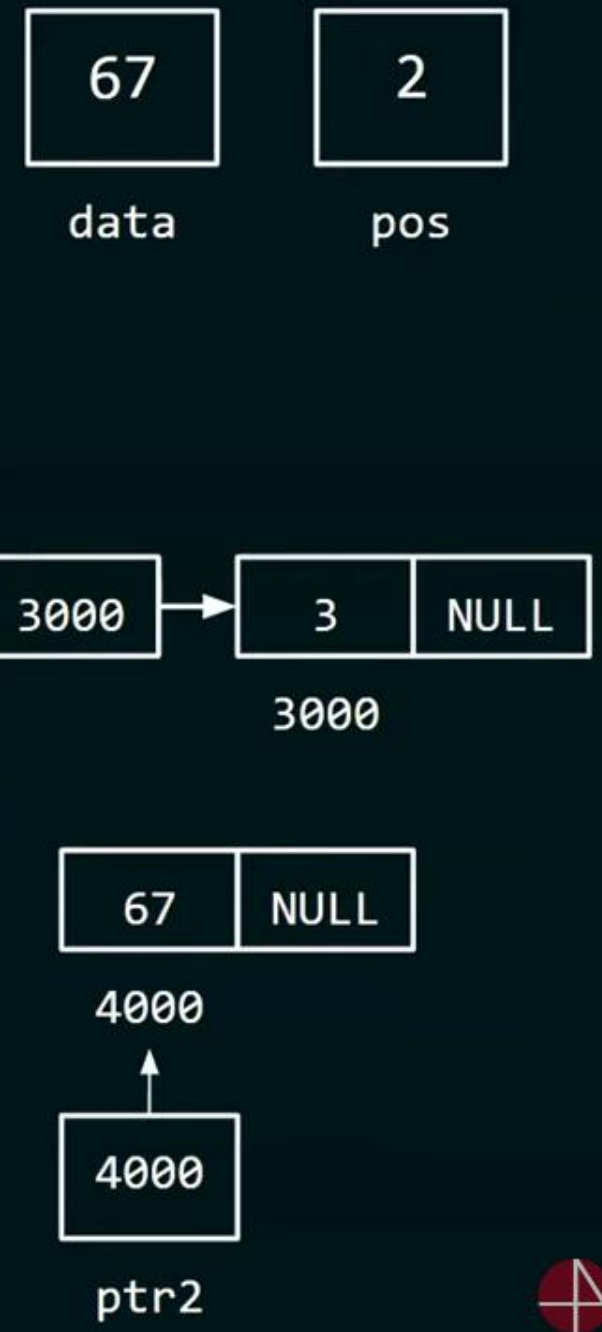
Add at position function

```
void add_at_pos(struct node* head, int data, int pos)
{
    struct node *ptr = head;
    struct node *ptr2 = malloc(sizeof(struct node));
    ptr2->data = data;
    ptr2->link = NULL;

    pos--;
    while(pos != 1)
    {
        ptr = ptr->link;
        pos--;

    }
    ptr2->link = ptr->link;
    ptr->link = ptr2;
}
```



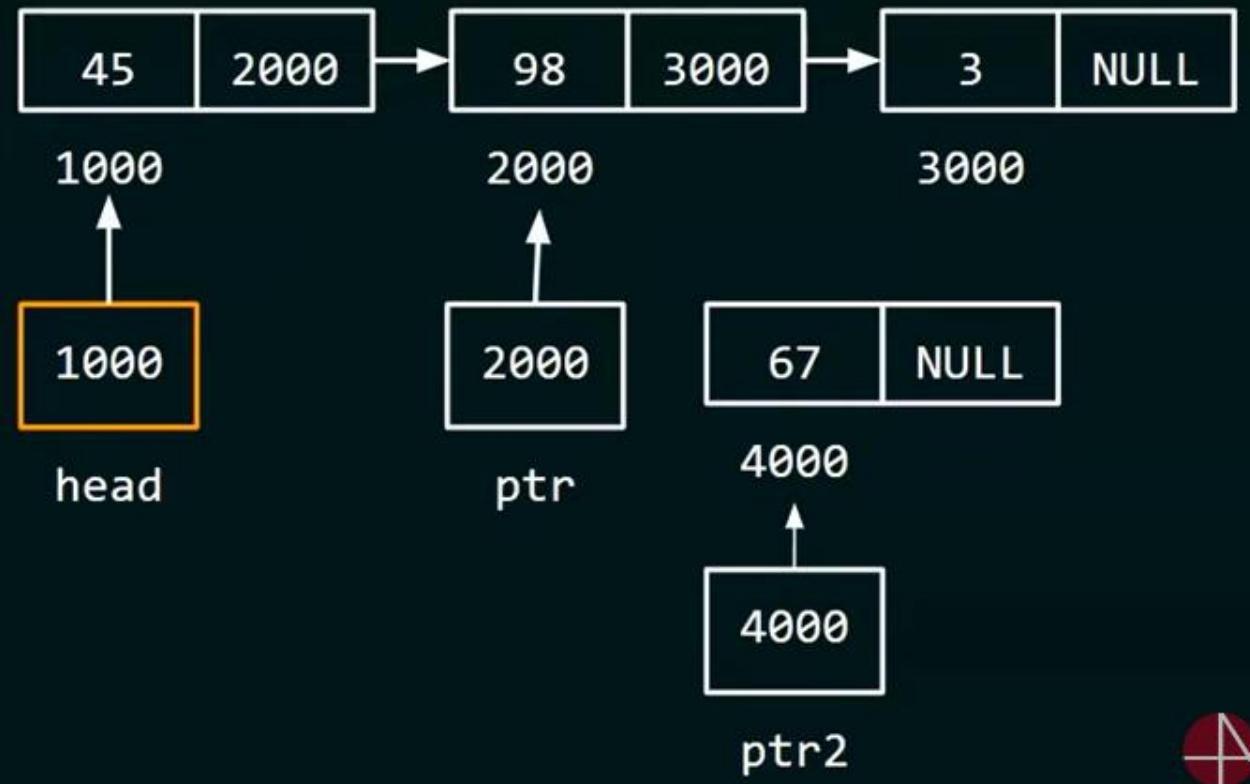
Add at position function

```
void add_at_pos(struct node* head, int data, int pos)
{
    struct node *ptr = head;
    struct node *ptr2 = malloc(sizeof(struct node));
    ptr2->data = data;
    ptr2->link = NULL;

    pos--;
    while(pos != 1)
    {
        ptr = ptr->link;
        pos--;

    }
    ptr2->link = ptr->link;
    ptr->link = ptr2;
}
```



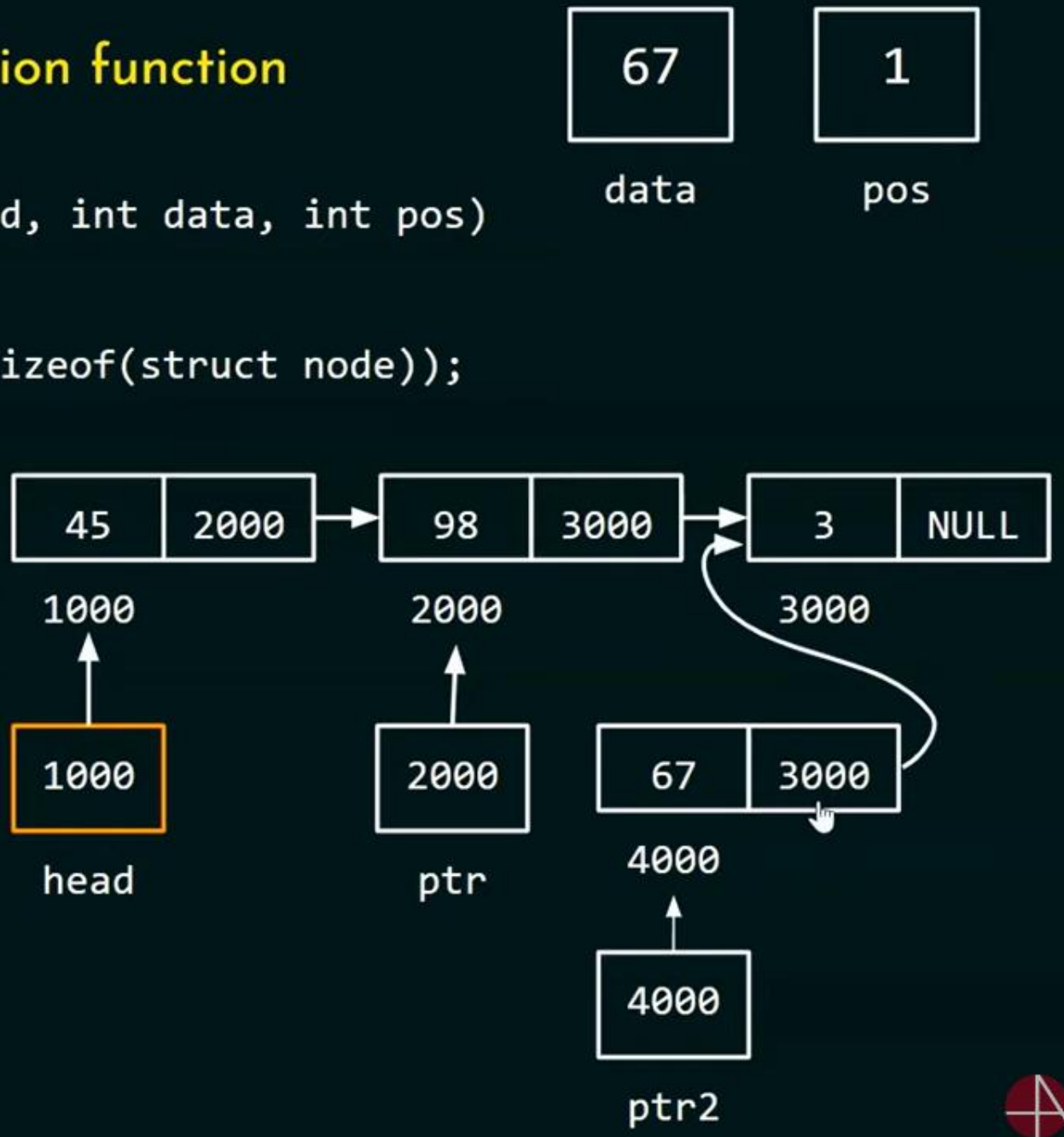
Add at position function

```
void add_at_pos(struct node* head, int data, int pos)
{
    struct node *ptr = head;
    struct node *ptr2 = malloc(sizeof(struct node));
    ptr2->data = data;
    ptr2->link = NULL;

    pos--;
    while(pos != 1)
    {
        ptr = ptr->link;
        pos--;

    }
    ptr2->link = ptr->link;
    ptr->link = ptr2;
}
```



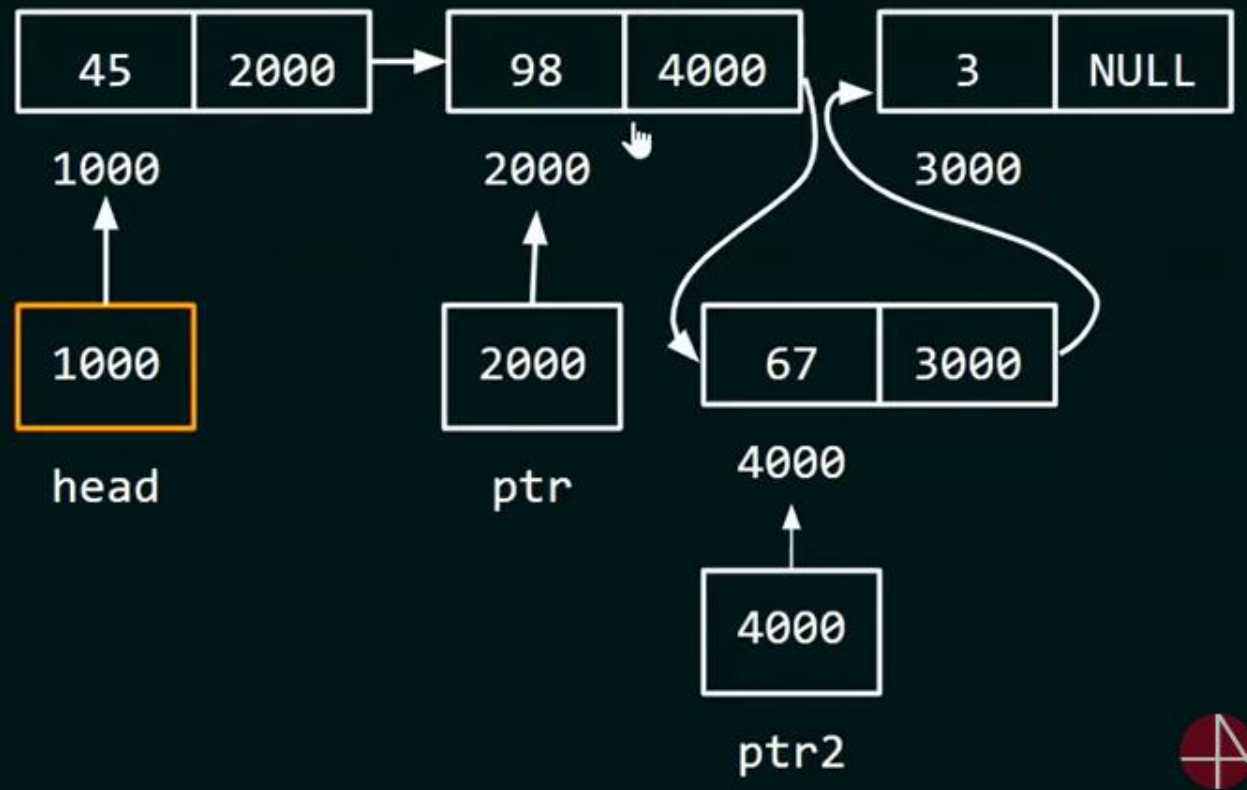
Add at position function

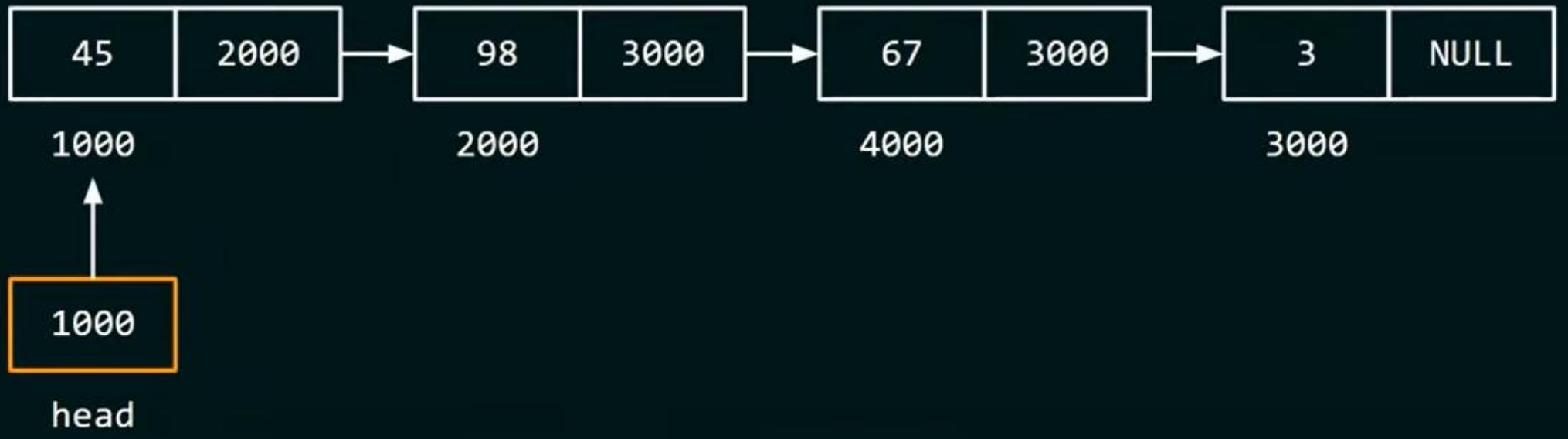
```
void add_at_pos(struct node* head, int data, int pos)
{
    struct node *ptr = head;
    struct node *ptr2 = malloc(sizeof(struct node));
    ptr2->data = data;
    ptr2->link = NULL;

    pos--;
    while(pos != 1)
    {
        ptr = ptr->link;
        pos--;

    }
    ptr2->link = ptr->link;
    ptr->link = ptr2;
}
```



Add at position function

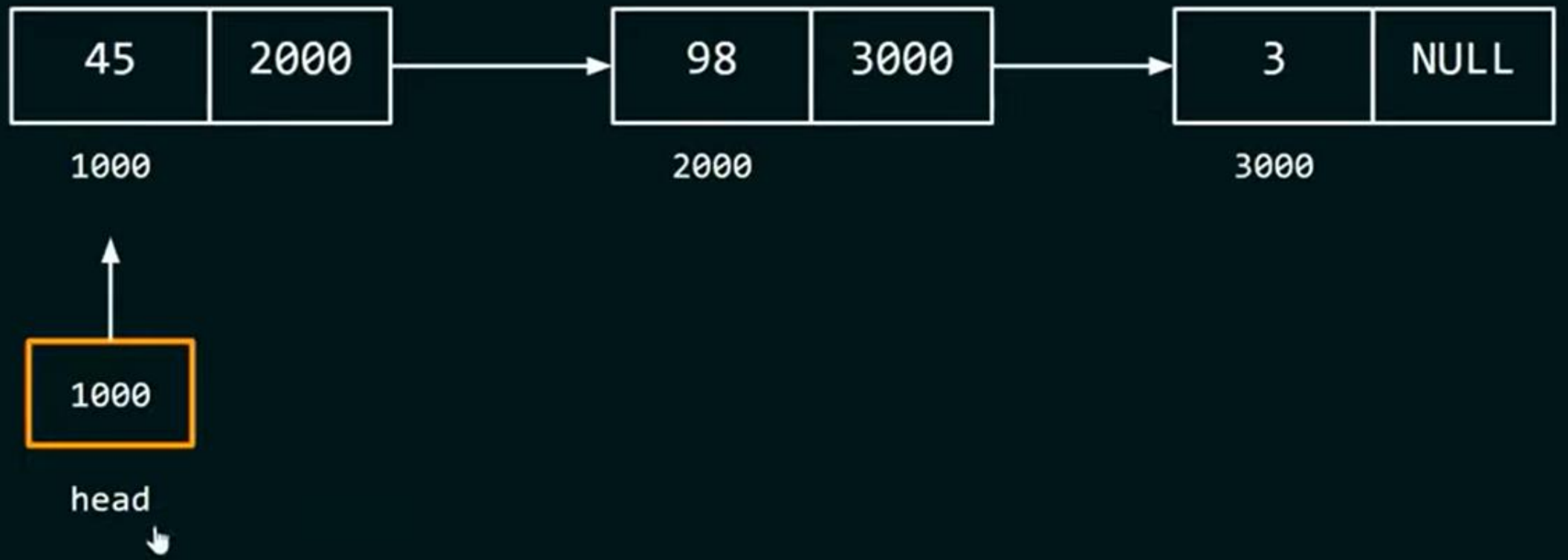
```
void add_at_pos(struct node* head, int data, int pos)
{
    struct node *ptr = head;
    struct node *ptr2 = malloc(sizeof(struct node));
    ptr2->data = data;
    ptr2->link = NULL;

    pos--;
    while(pos != 1)
    {
        ptr = ptr->link;
        pos--;

    }
    ptr2->link = ptr->link;
    ptr->link = ptr2;
}
```

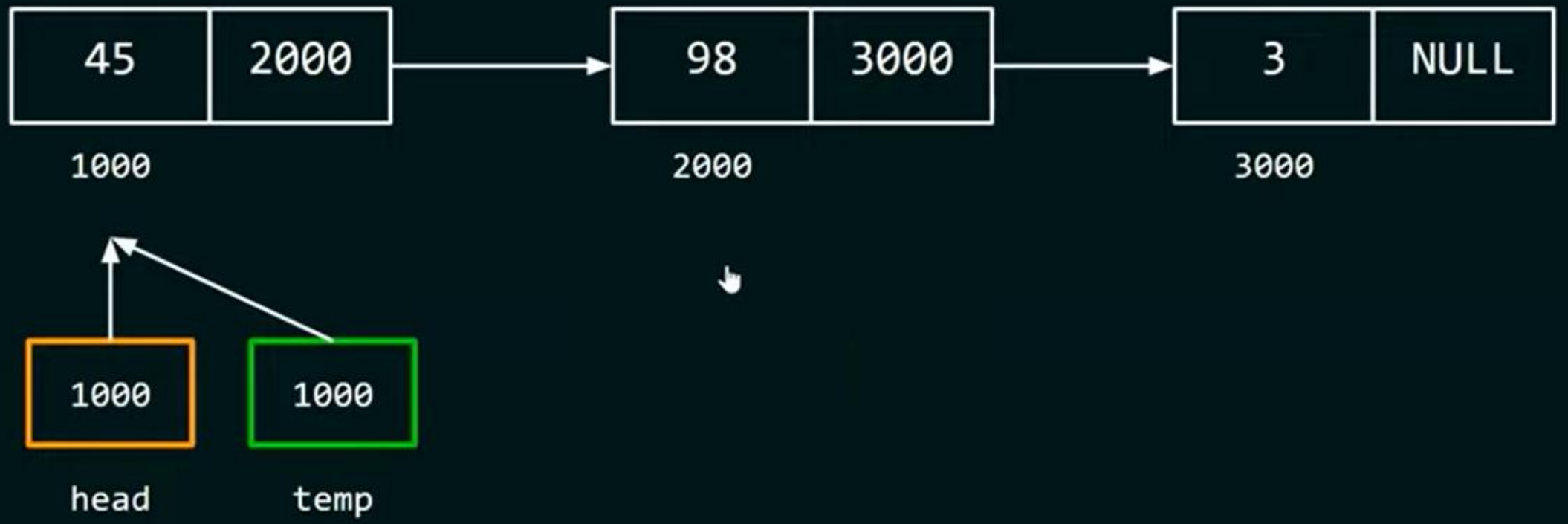


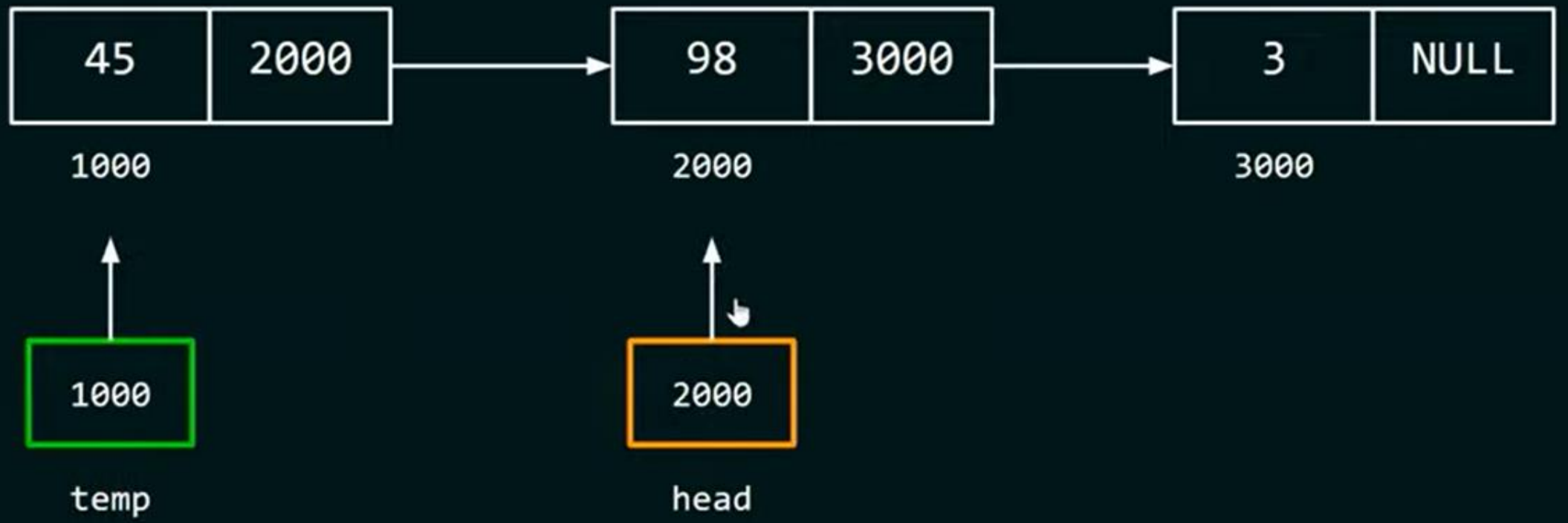


PROGRAMMING AND DATA STRUCTURES

Deleting the First Node of the Single Linked List





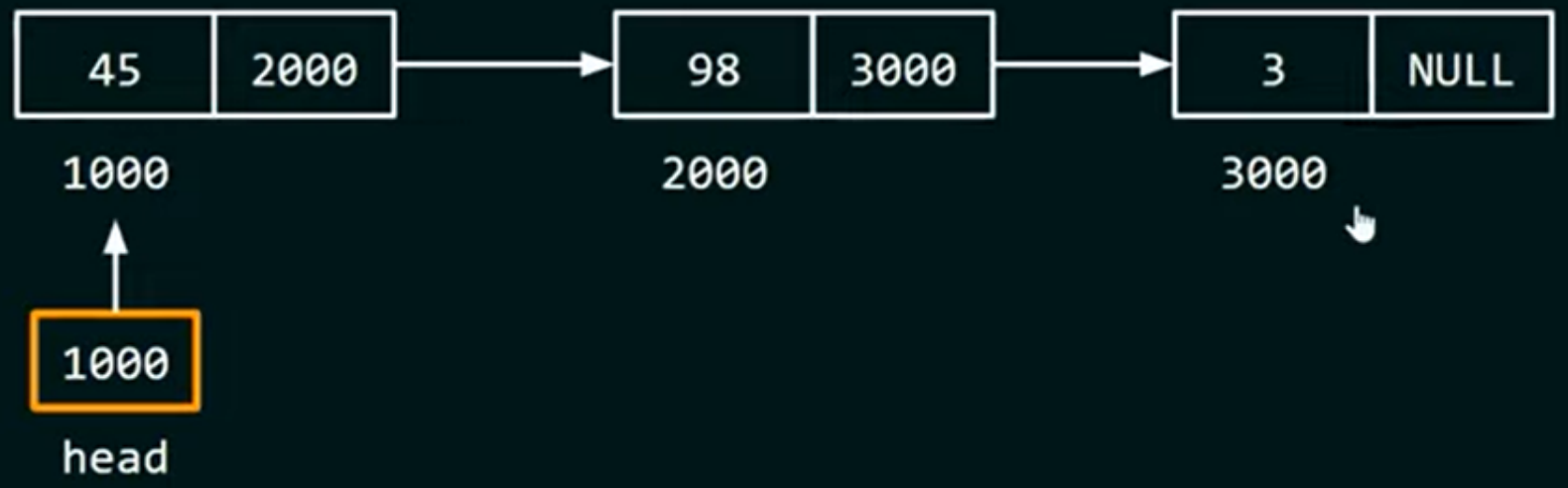


Program

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main() {
     head = del_first(head);
     ptr = head;
     while(ptr != NULL)
     {
         printf("%d ", ptr->data);
         ptr = ptr->link;
     }
     return 0;
}
```


Program

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main() {
     head = del_first(head);
     ptr = head;
     while(ptr != NULL)
     {
         printf("%d ", ptr->data);
         ptr = ptr->link;
     }
     return 0;
}
```



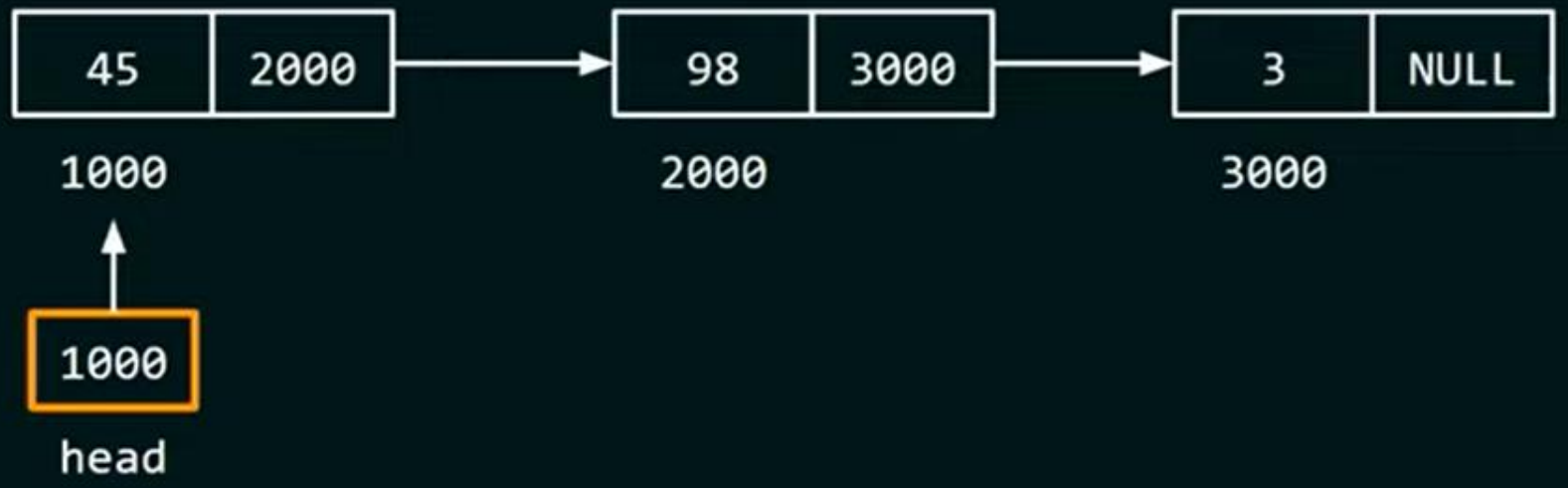
Program

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main() {
     head = del_first(head);
     ptr = head;
     while(ptr != NULL)
     {
         printf("%d ", ptr->data);
         ptr = ptr->link;
     }
     return 0;
}
```

```c
struct node* del_first(struct node *head)
{
    if(head == NULL)
        printf("List is already empty!");
    else
    {
    struct node *temp = head;
    head = head->link;
    free(temp);
    }
    return head;
}
```

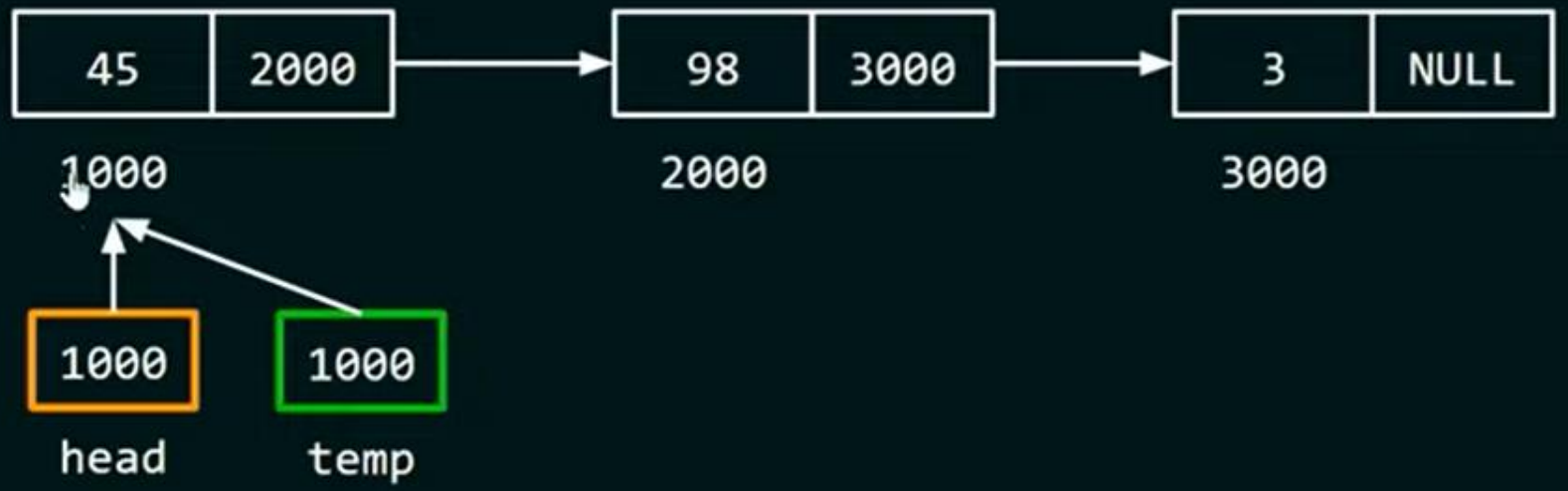

Program

```
struct node* del_first(struct node *head)
{
    if(head == NULL)
        printf("List is already empty!");
    else
    {
    struct node *temp = head;
    head = head->link;
    free(temp);
    temp = NULL;
    }
    return head;
}
```



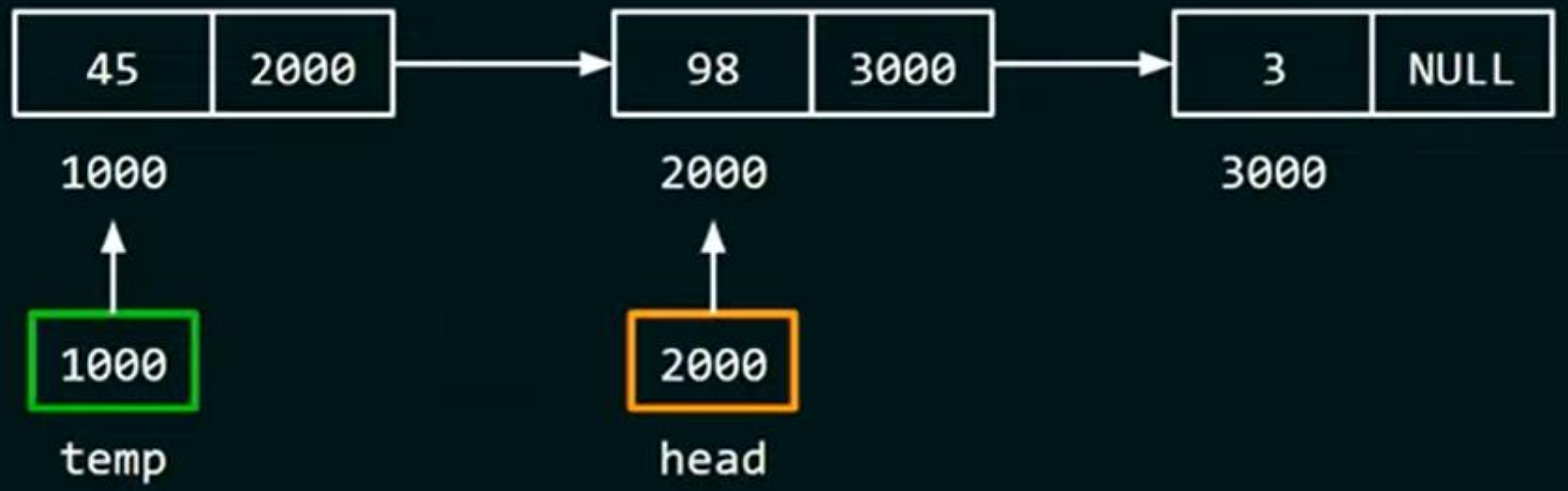
Program

```
struct node* del_first(struct node *head)
{
    if(head == NULL)
        printf("List is already empty!");
    else
    {
    struct node *temp = head;
    head = head->link;
    free(temp);
    temp = NULL;
    }
    return head;
}
```



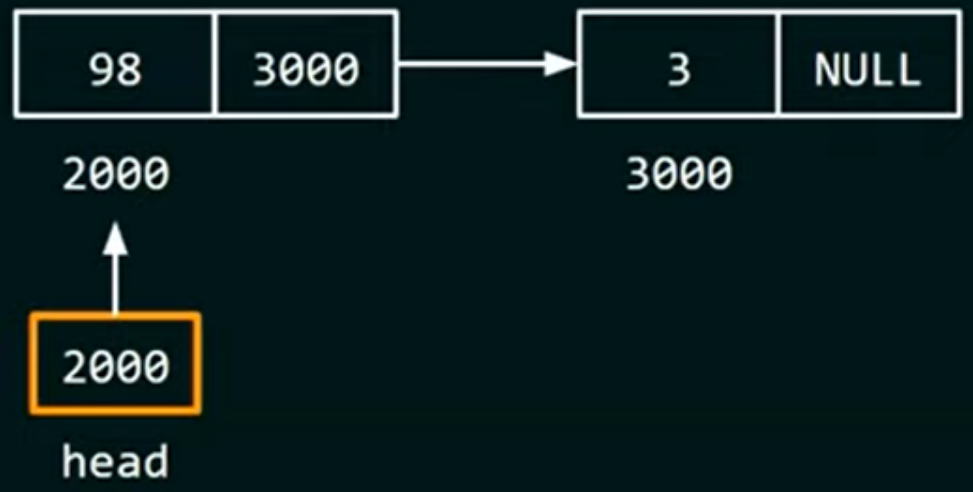
Program

```
struct node* del_first(struct node *head)
{
    if(head == NULL)
        printf("List is already empty!");
    else
    {
    struct node *temp = head;
    head = head->link;
    free(temp);
    temp = NULL;
    }
    return head;
}
```



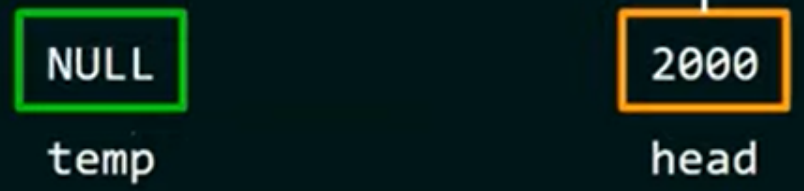
Program

```
struct node* del_first(struct node *head)
{
    if(head == NULL)
        printf("List is already empty!");
    else
    {
    struct node *temp = head;
    head = head->link;
    free(temp);
    temp = NULL;
    }
    return head;
}
```



Program

```
struct node* del_first(struct node *head)
{
    if(head == NULL)
        printf("List is already empty!");
    else
    {
    struct node *temp = head;
    head = head->link;
    free(temp);
    temp = NULL;
    }
    return head;
}
```

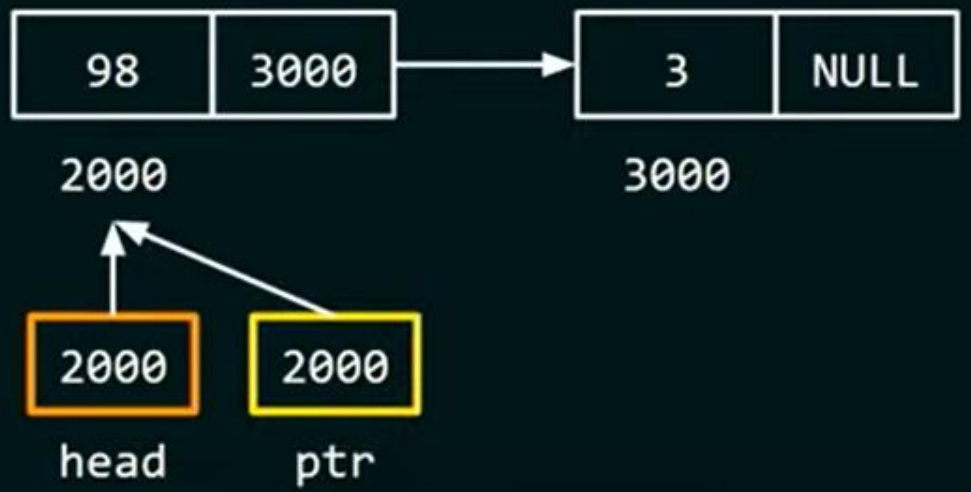


Program

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main() {
    head = del_first(head);
    ptr = head;
    while(ptr != NULL)
    {
        printf("%d ", ptr->data);
        ptr = ptr->link;
    }
    return 0;
}
```

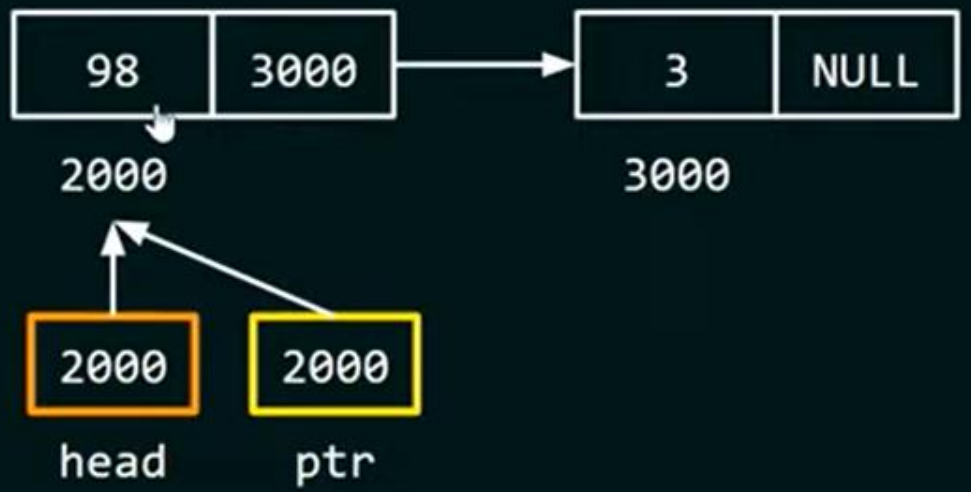
Program

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main() {
    head = del_first(head);
    ptr = head;
    while(ptr != NULL)
    {
        printf("%d ", ptr->data);
        ptr = ptr->link;
    }
    return 0;
}
```



Program

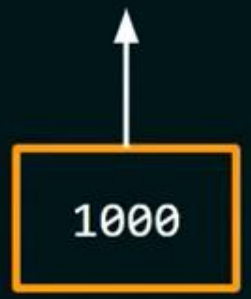
```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main() {
    head = del_first(head);
    ptr = head;
    while(ptr != NULL)
    {
        printf("%d ", ptr->data);
        ptr = ptr->link;
    }
    return 0;
}
```



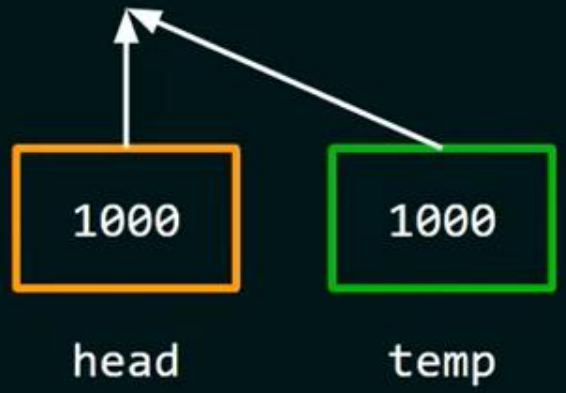
Output: 98 3

Programming and Data Structures

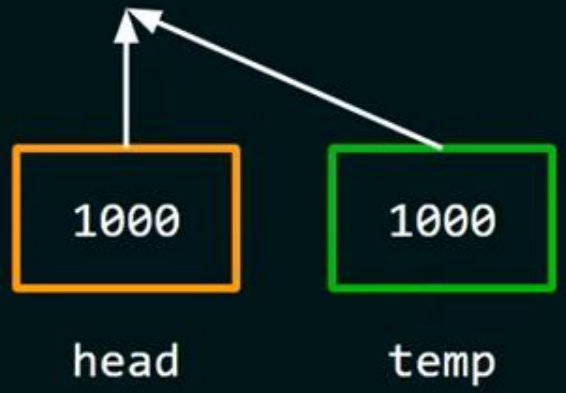
Deleting the Last Node of the Single Linked List



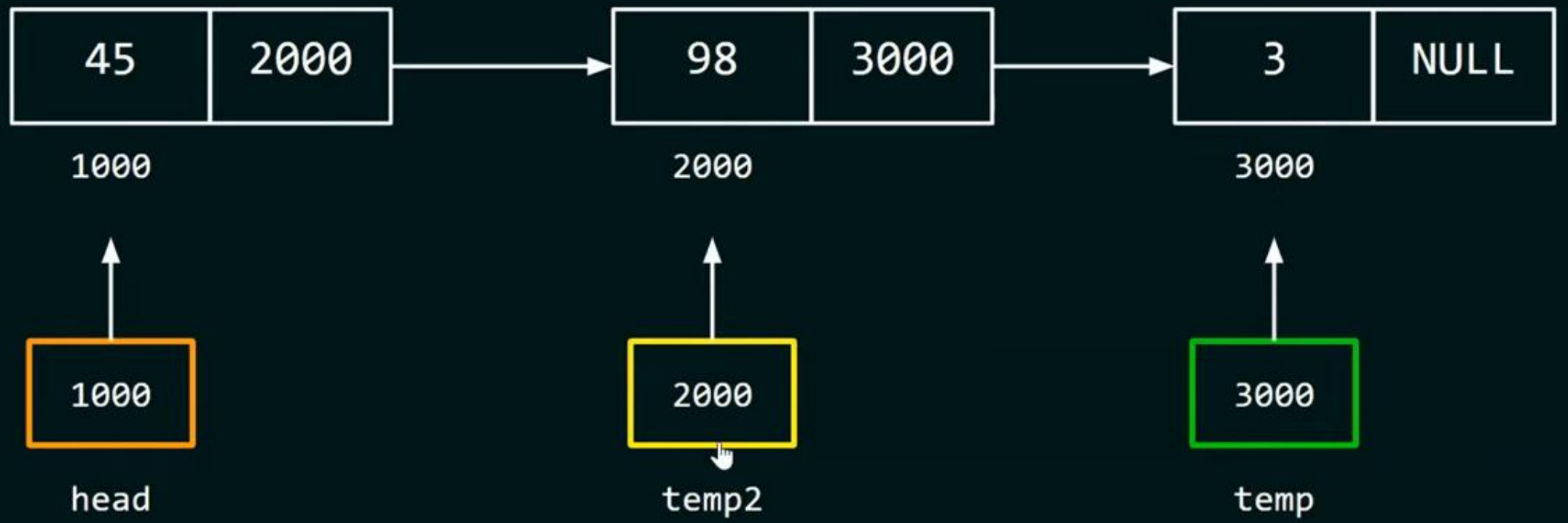
Note that we only have an head pointer and we can access the list only through head pointer.

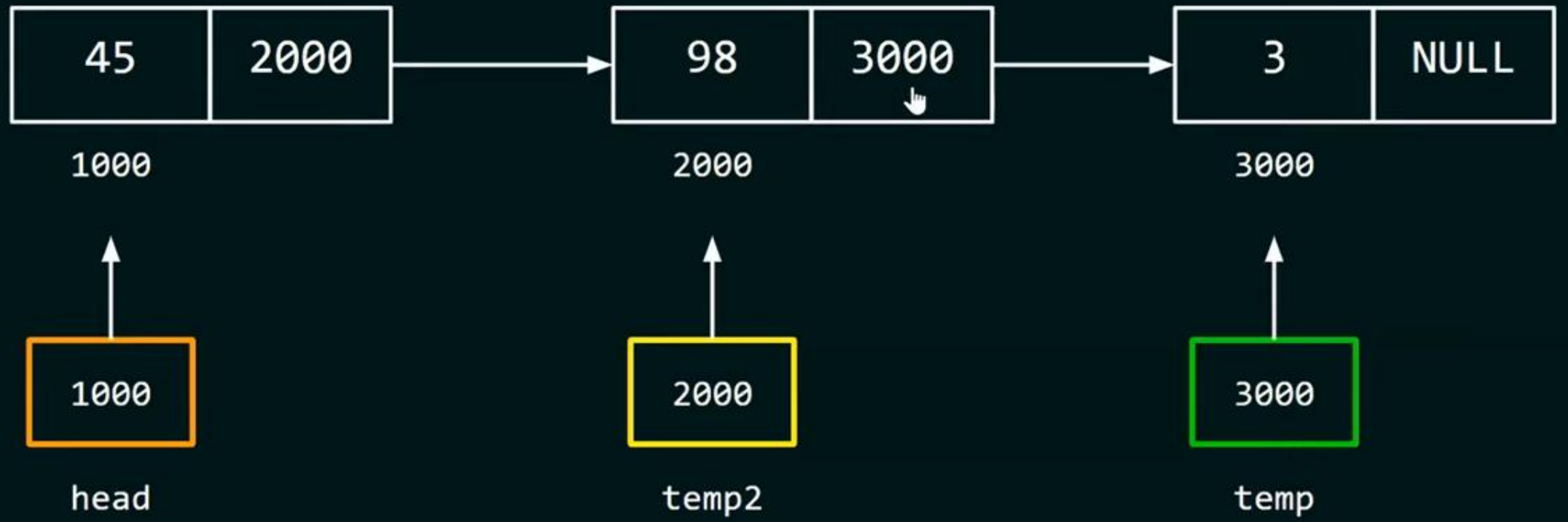


The next step is to traverse the list.

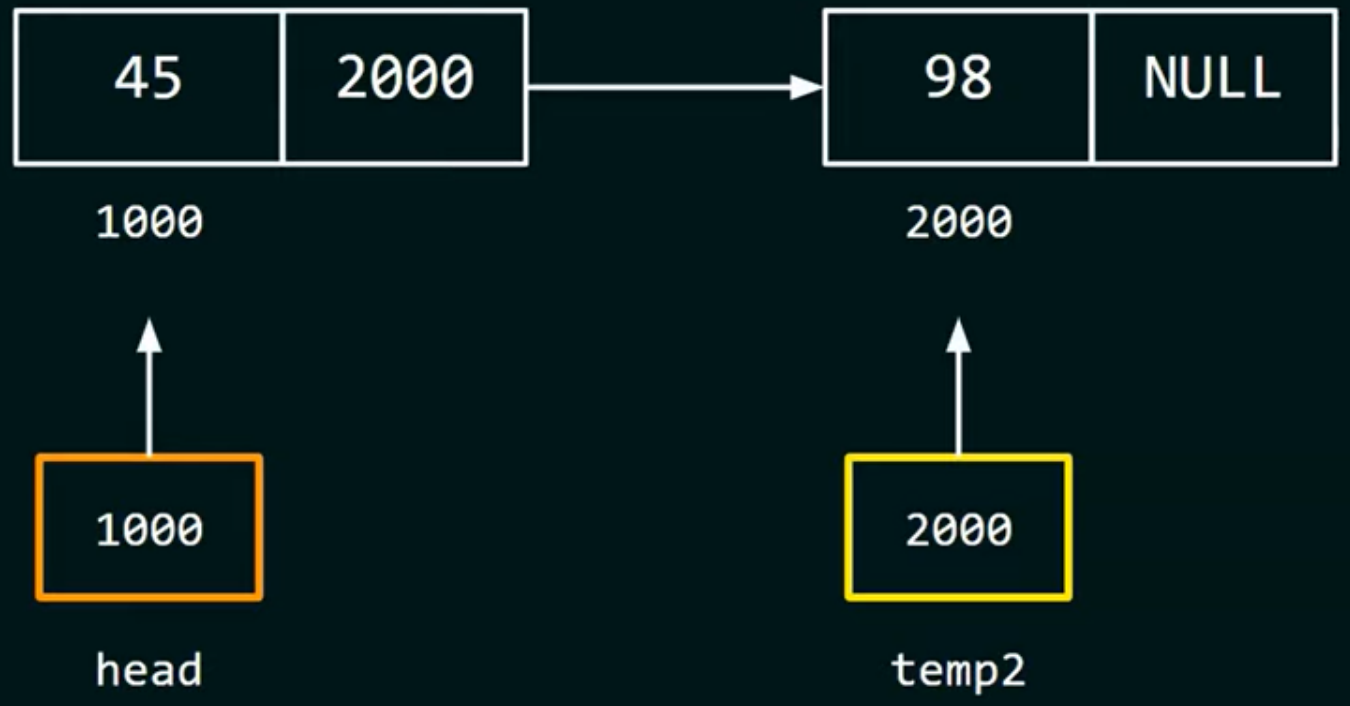


The next step is to traverse the list. We will keep two pointers. One pointer will stop at the last node of the list and the other pointer will stop at the second last node of the list.





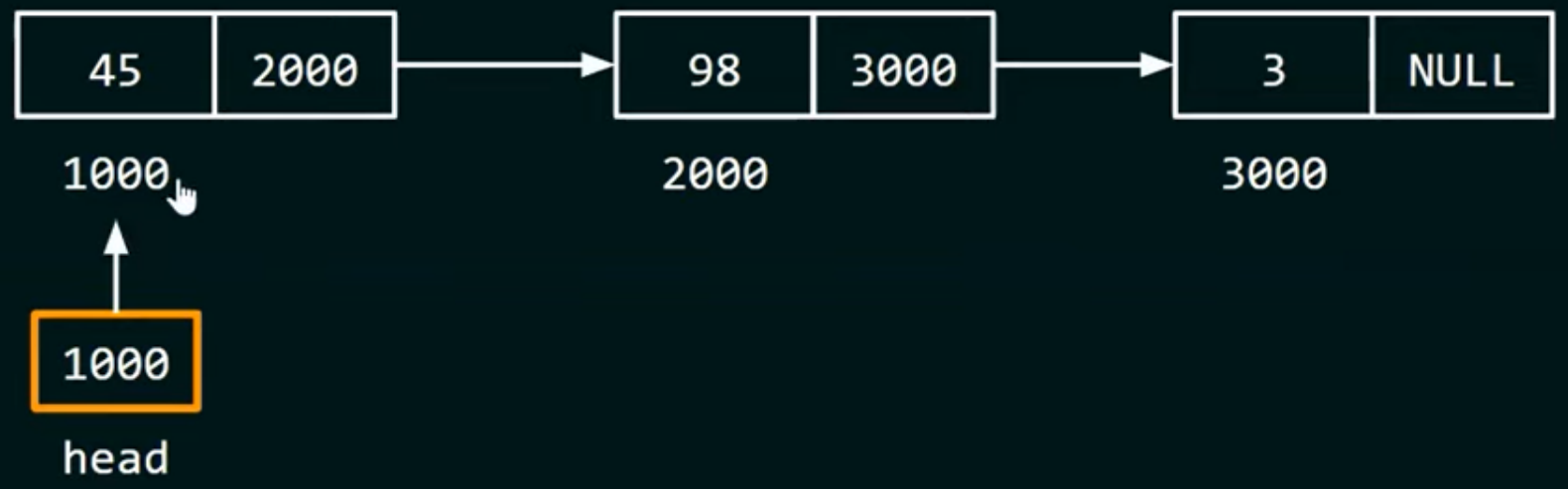

```
temp2->link = NULL;
```




```
temp2->link = NULL;
free(temp);
temp = NULL;
```

Program

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main() {
    head = del_last(head);
    ptr = head;
    while(ptr != NULL)
    {
        printf("%d ", ptr->data);
        ptr = ptr->link;
    }
     return 0;
}
```



Function

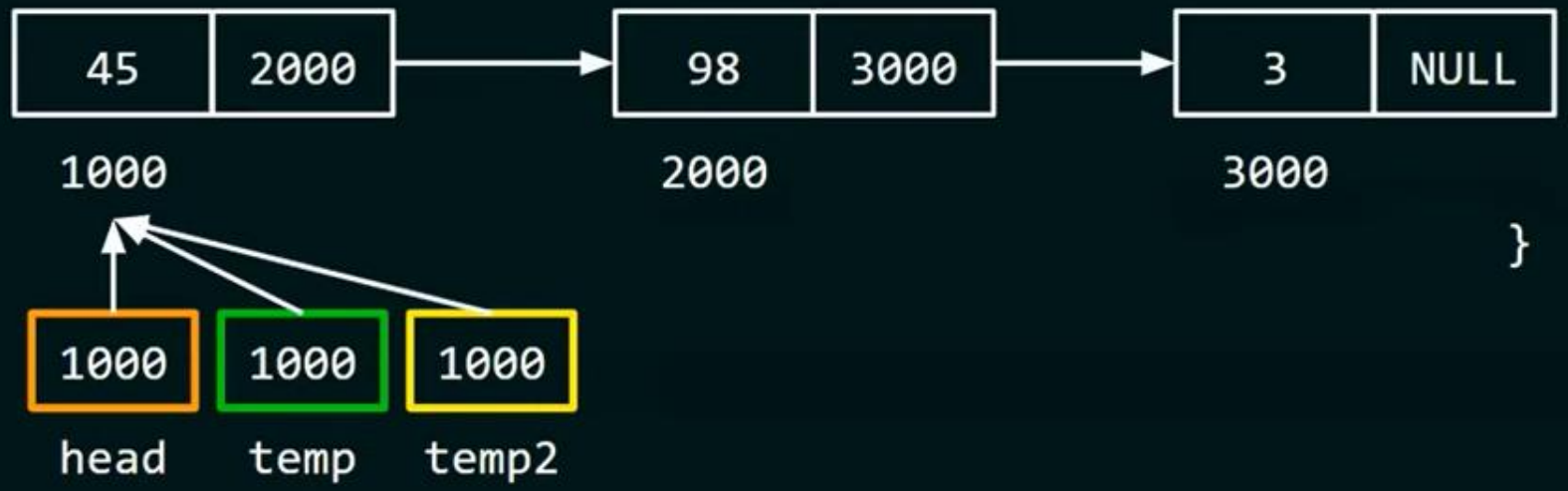
```
struct node* del_last(struct node *head)
{
    if(head == NULL)
        printf("List is already empty!");
    else if(head->link == NULL)
    {
        free(head);
        head = NULL;
    }
```

Check if there is no node in the linked list.

```
    else
    {
        struct node *temp = head;
        struct node *temp2 = head;
        while(temp->link != NULL)
        {
            temp2 = temp;
            temp = temp->link;
        }
        temp2->link = NULL;
        free(temp);
        temp = NULL;
    }
    return head;
}
```


Function

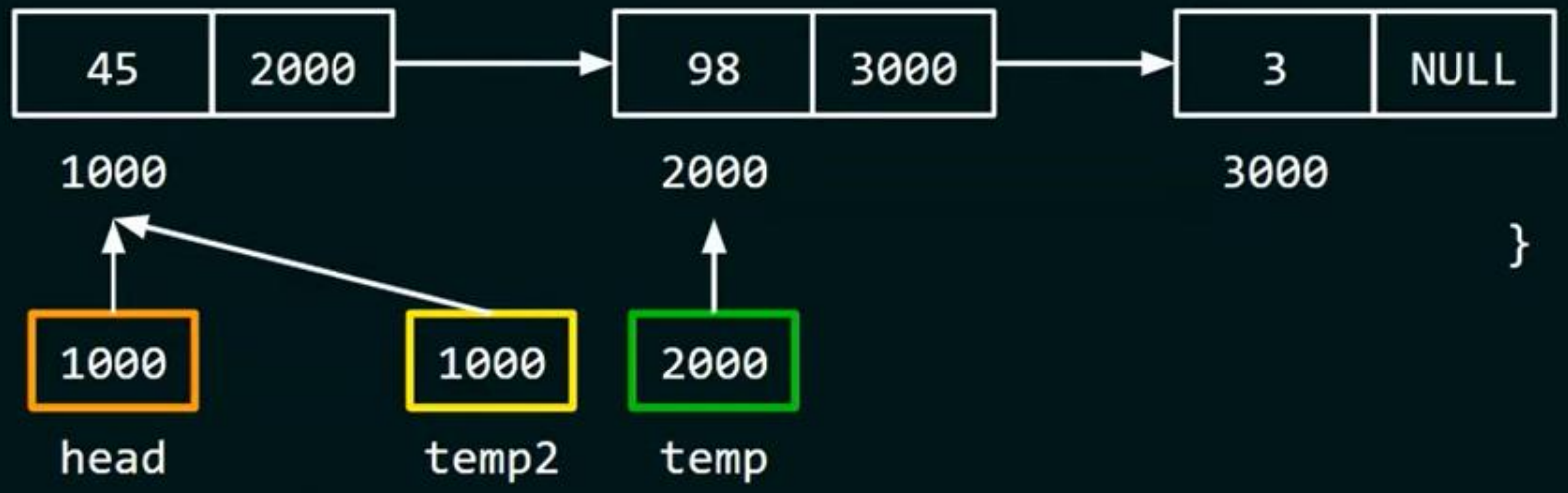
```
struct node* del_last(struct node *head)
{
    if(head == NULL)
        printf("List is already empty!");
    else if(head->link == NULL)
    {
        free(head);
        head = NULL;
    }
```




```
else
    {
    struct node *temp = head;
    struct node *temp2 = head;
    while(temp->link != NULL)
    {
        temp2 = temp;
        temp = temp->link;
    }
    temp2->link = NULL;
    free(temp);
    temp = NULL;
    }
    return head;
}
```

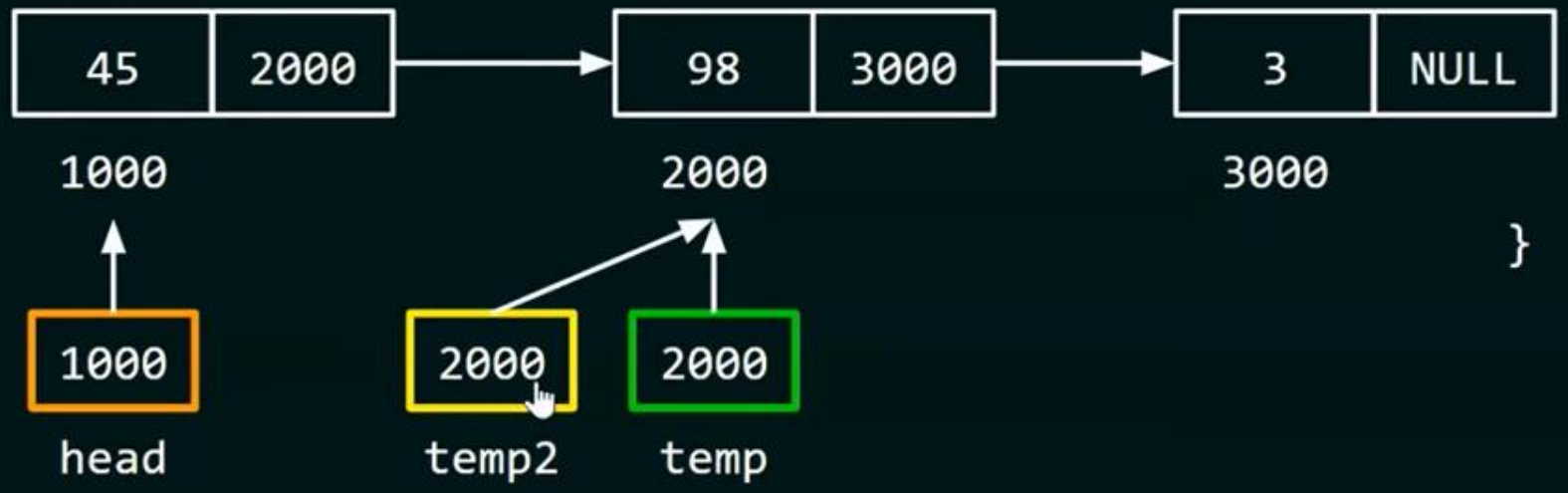
Function

```
struct node* del_last(struct node *head)
{
    if(head == NULL)
        printf("List is already empty!");
    else if(head->link == NULL)
    {
        free(head);
        head = NULL;
    }
else
    {
    struct node *temp = head;
    struct node *temp2 = head;
    while(temp->link != NULL)
    {
        temp2 = temp;
        temp = temp->link;
    }
    temp2->link = NULL;
    free(temp);
    temp = NULL;
    }
    return head;
}
```



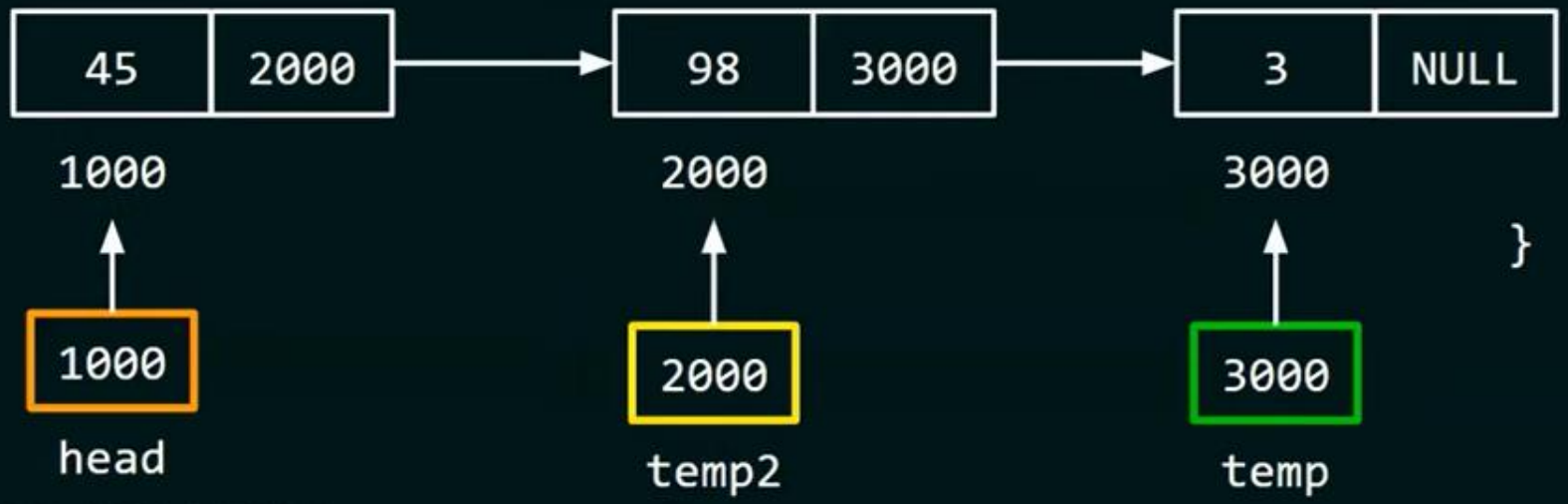
Function

```
struct node* del_last(struct node *head)
{
    if(head == NULL)
        printf("List is already empty!");
    else if(head->link == NULL)
    {
        free(head);
        head = NULL;
    }
else
    {
    struct node *temp = head;
    struct node *temp2 = head;
    while(temp->link != NULL)
    {
        temp2 = temp;
        temp = temp->link;
    }
    temp2->link = NULL;
    free(temp);
    temp = NULL;
    }
    return head;
}
```



Function

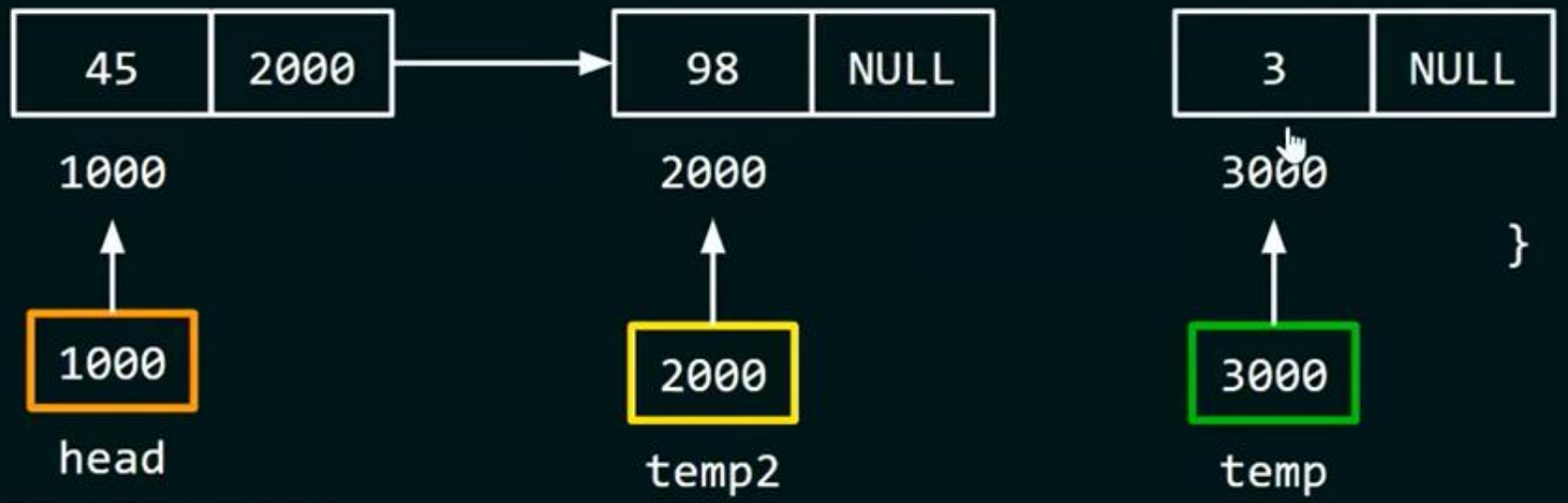
```
struct node* del_last(struct node *head)
{
    if(head == NULL)
        printf("List is already empty!");
    else if(head->link == NULL)
    {
        free(head);
        head = NULL;
    }
```




```
else
    {
    struct node *temp = head;
    struct node *temp2 = head;
    while(temp->link != NULL)
    {
        temp2 = temp;
        temp = temp->link;
    }
    temp2->link = NULL;
    free(temp);
    temp = NULL;
    }
    return head;
}
```

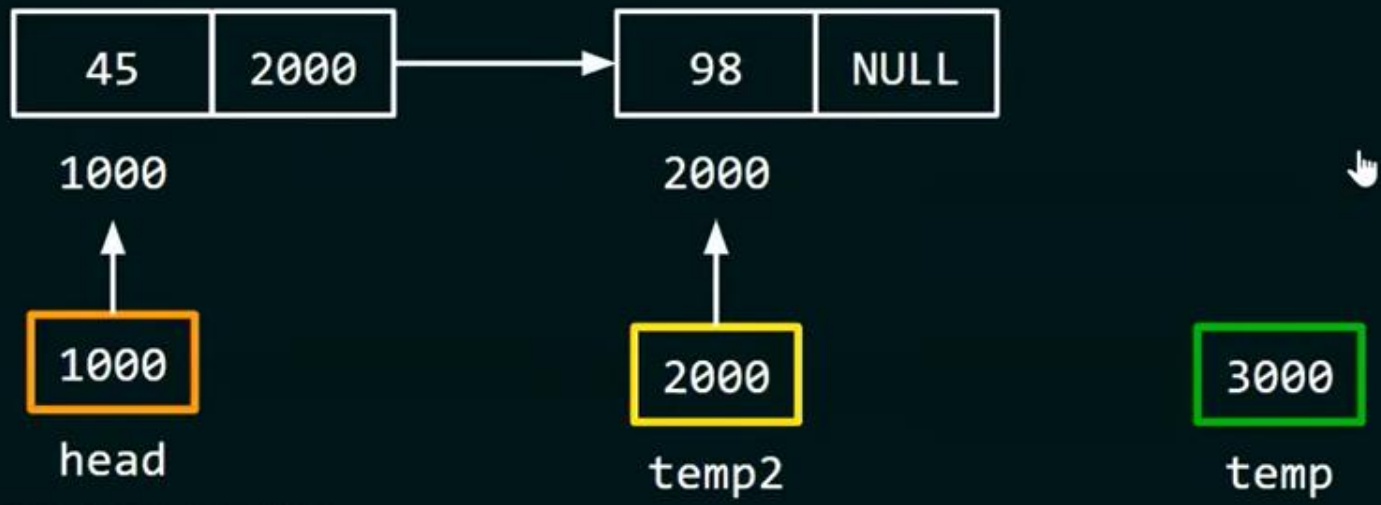
Function

```
struct node* del_last(struct node *head)
{
    if(head == NULL)
        printf("List is already empty!");
    else if(head->link == NULL)
    {
        free(head);
        head = NULL;
    }
else
    {
    struct node *temp = head;
    struct node *temp2 = head;
    while(temp->link != NULL)
    {
        temp2 = temp;
        temp = temp->link;
    }
    temp2->link = NULL;
    free(temp);
    temp = NULL;
    }
    return head;
}
```



Function

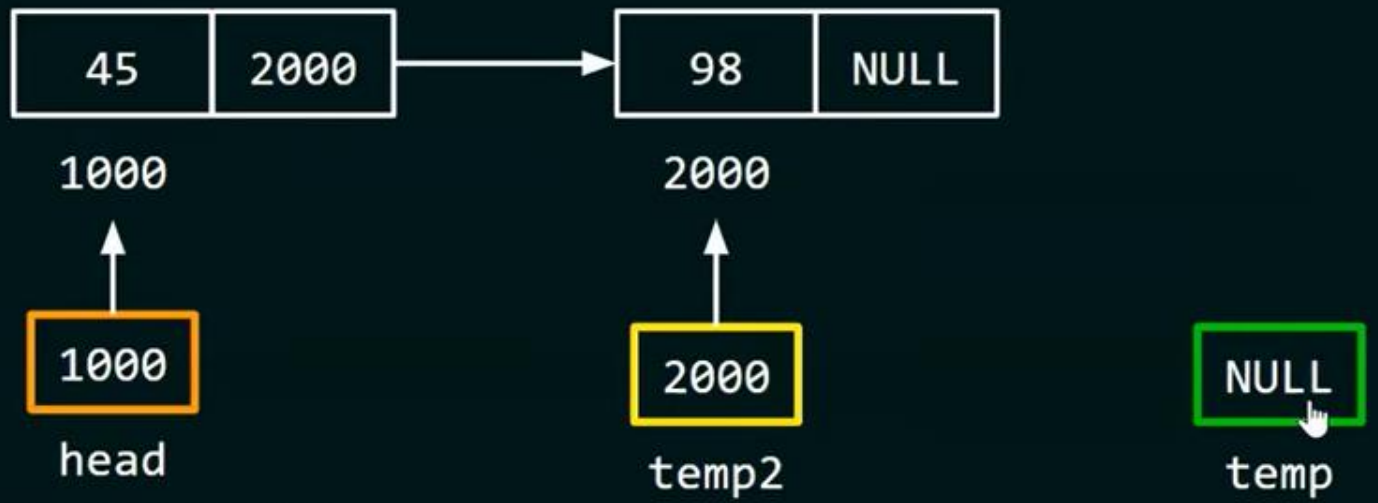
```
struct node* del_last(struct node *head)
{
    if(head == NULL)
        printf("List is already empty!");
    else if(head->link == NULL)
    {
        free(head);
        head = NULL;
    }
```




```
else
    {
    struct node *temp = head;
    struct node *temp2 = head;
    while(temp->link != NULL)
    {
        temp2 = temp;
        temp = temp->link;
    }
    temp2->link = NULL;
    free(temp);
    temp = NULL;
    }
    return head;
}
```

Function

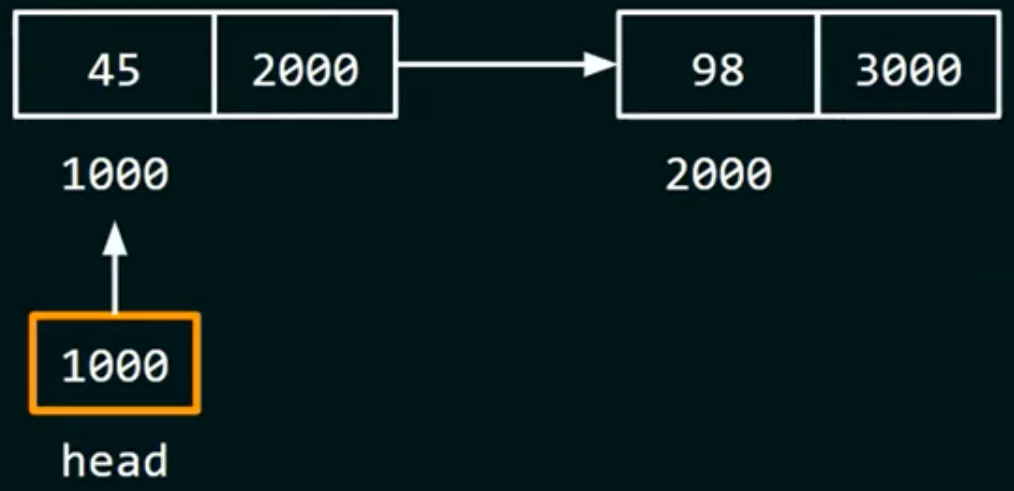
```
struct node* del_last(struct node *head)
{
    if(head == NULL)
        printf("List is already empty!");
    else if(head->link == NULL)
    {
        free(head);
        head = NULL;
    }
```




```
else
    {
    struct node *temp = head;
    struct node *temp2 = head;
    while(temp->link != NULL)
    {
        temp2 = temp;
        temp = temp->link;
    }
    temp2->link = NULL;
    free(temp);
    temp = NULL;
    }
    return head;
}
```

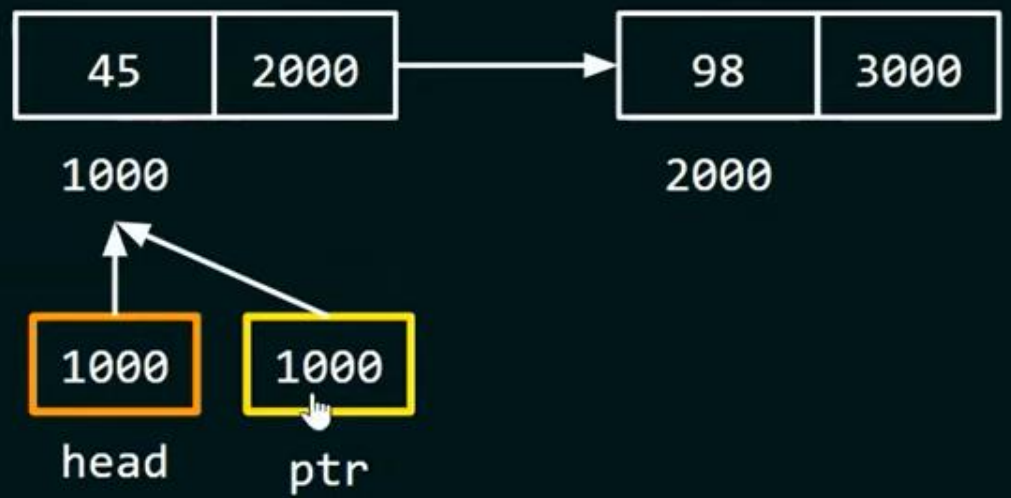
Program

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main() {
     head = del_first(head);
     ptr = head;
     while(ptr != NULL)
     {
         printf("%d ", ptr->data);
         ptr = ptr->link;
     }
     return 0;
}
```



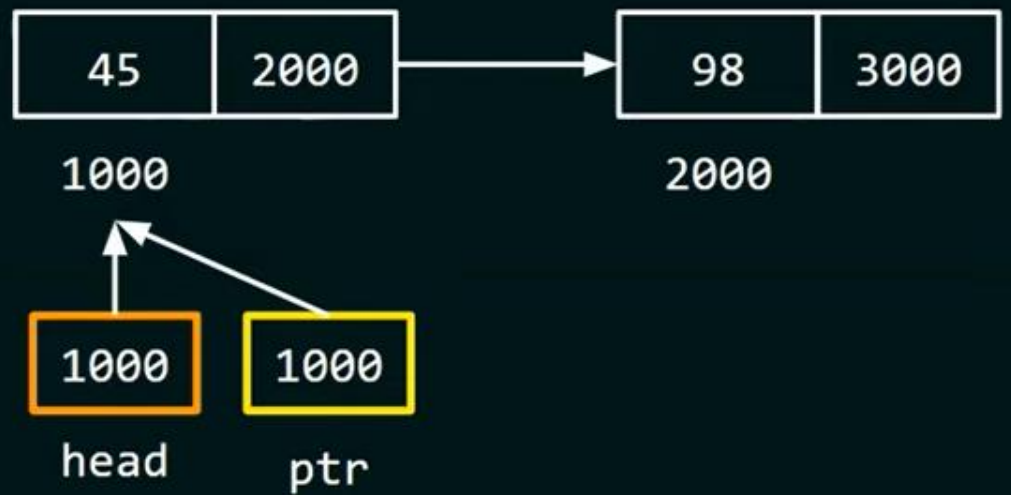
Program

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main() {
     head = del_first(head);
     ptr = head;
     while(ptr != NULL)
     {
         printf("%d ", ptr->data);
         ptr = ptr->link;
     }
     return 0;
}
```



Program

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main() {
     head = del_first(head);
     ptr = head;
     while(ptr != NULL)
     {
         printf("%d ", ptr->data);
         ptr = ptr->link;
     }
     return 0;
}
```



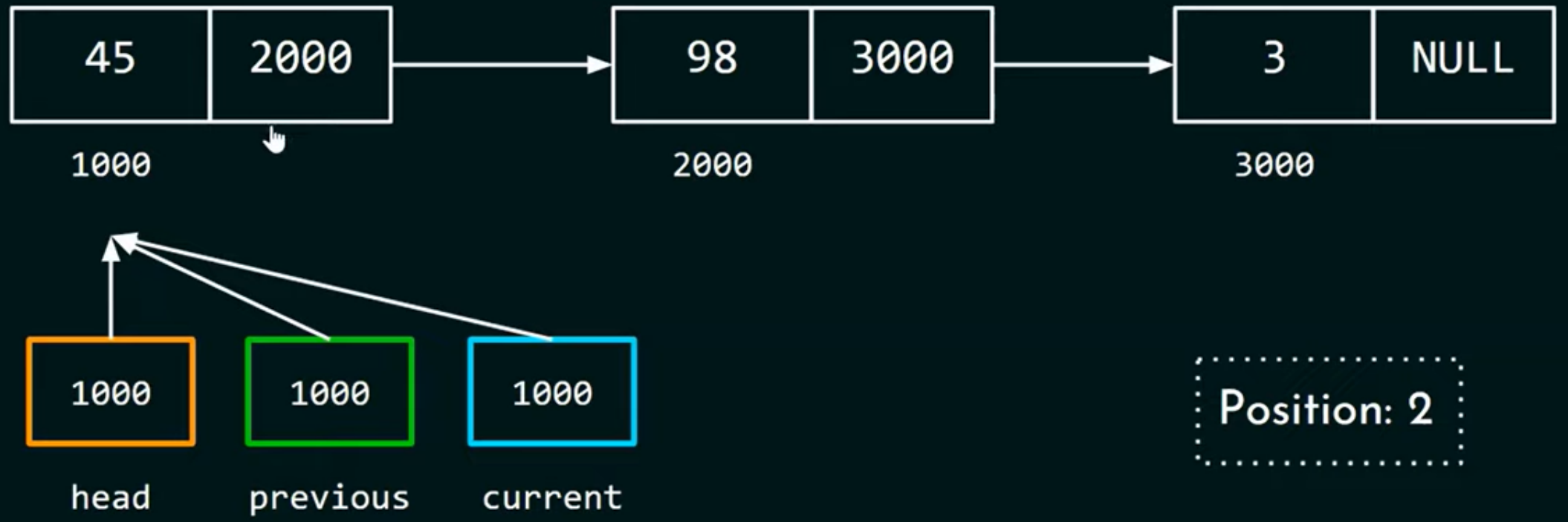
Output: 45 98

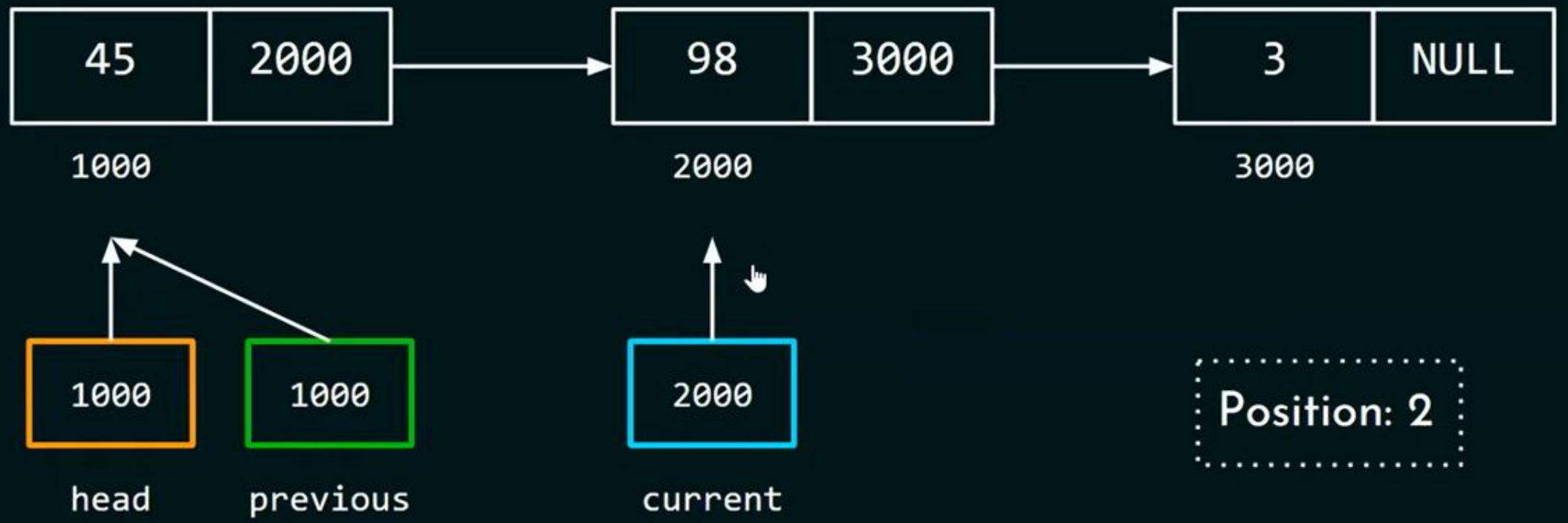
Note: There is no need to return the head pointer as we are not updating it in the function.

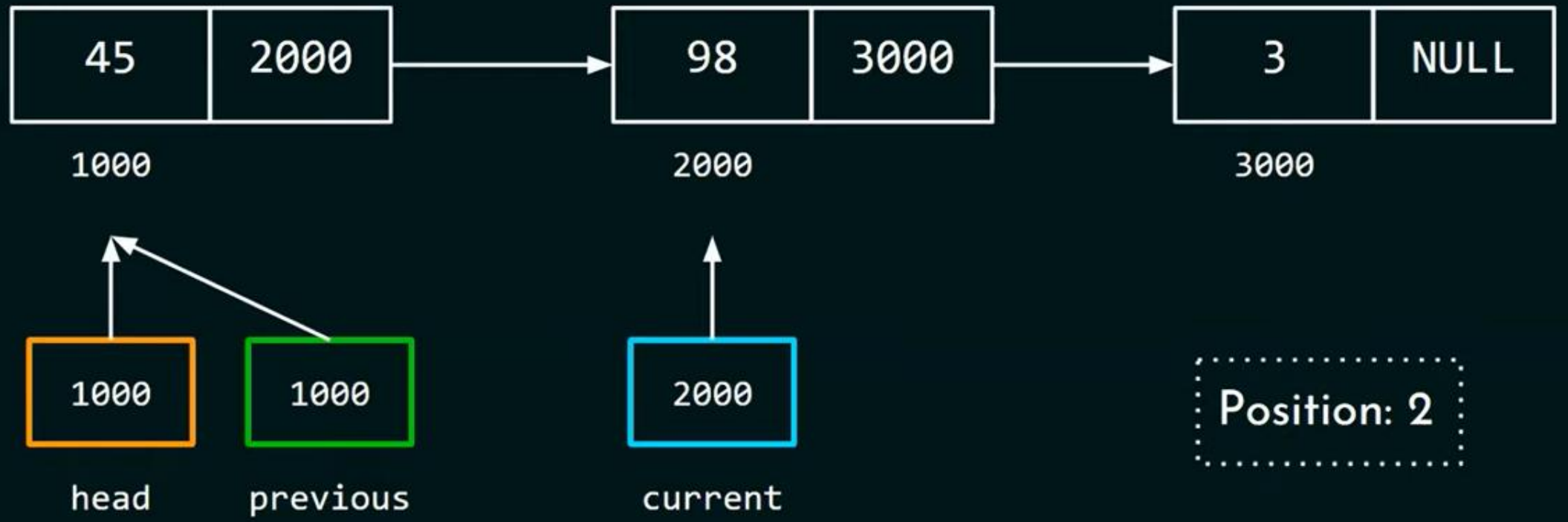
Programming and Data Structures

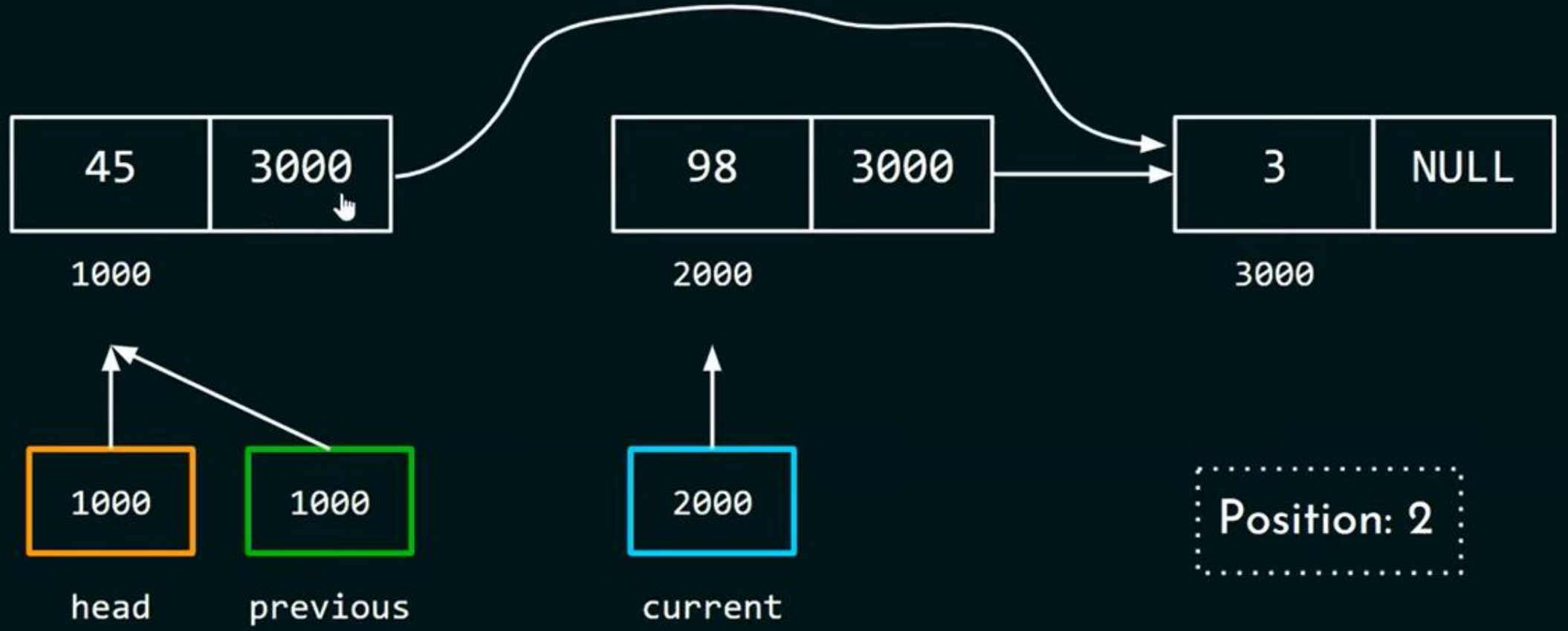
Deleting the Node at Particular Position

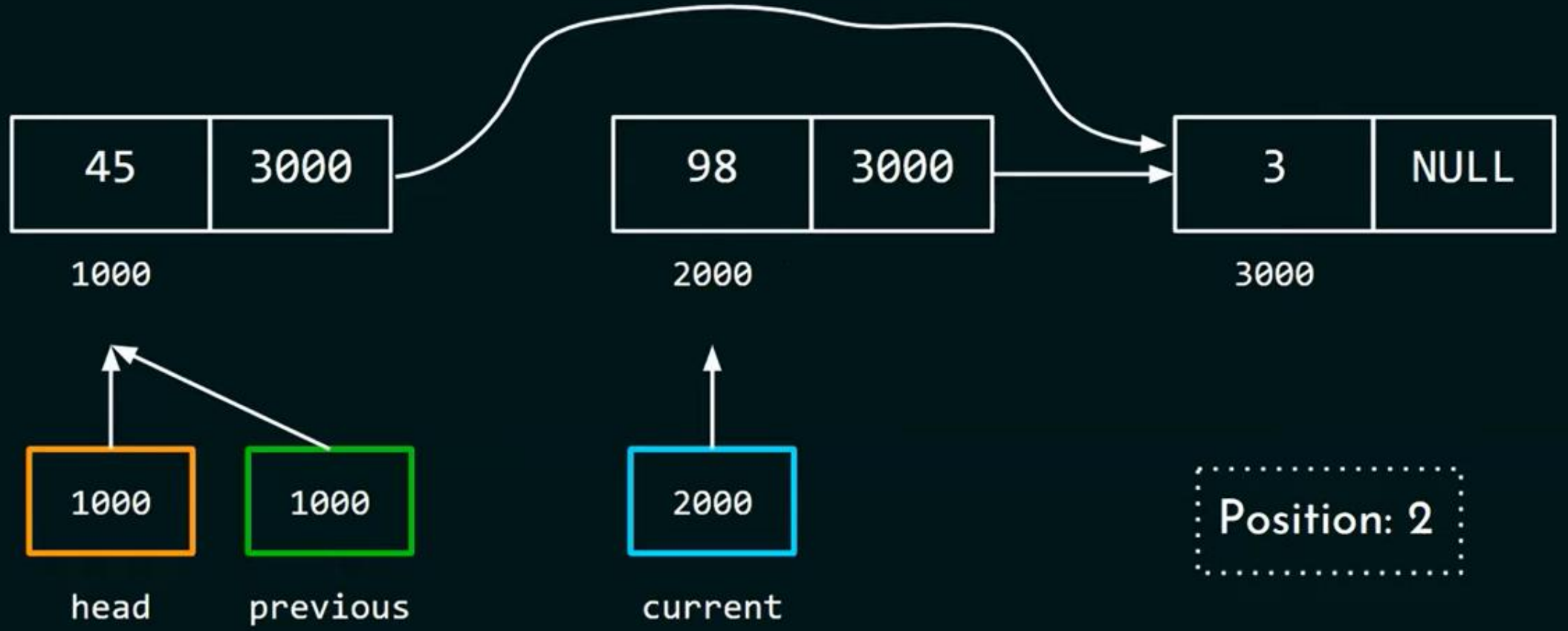
Position: 2





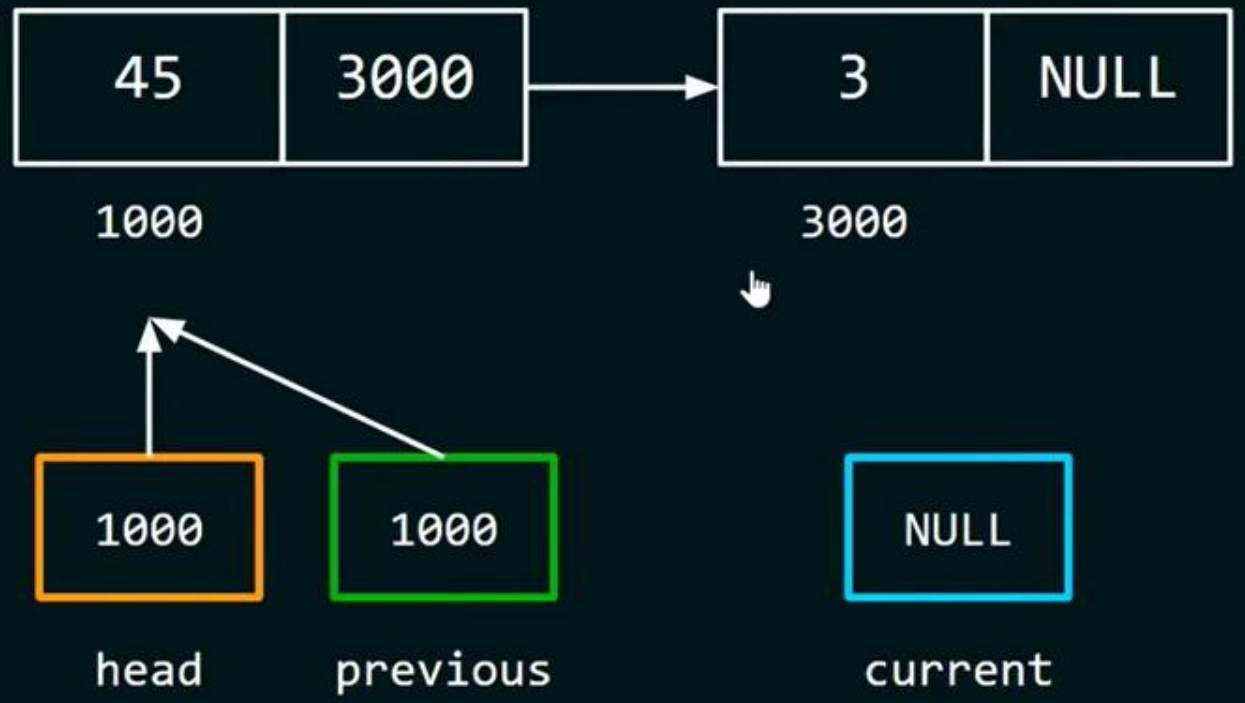






Position: 2

```
previous->link = current->link
free(current)
current = NULL
```



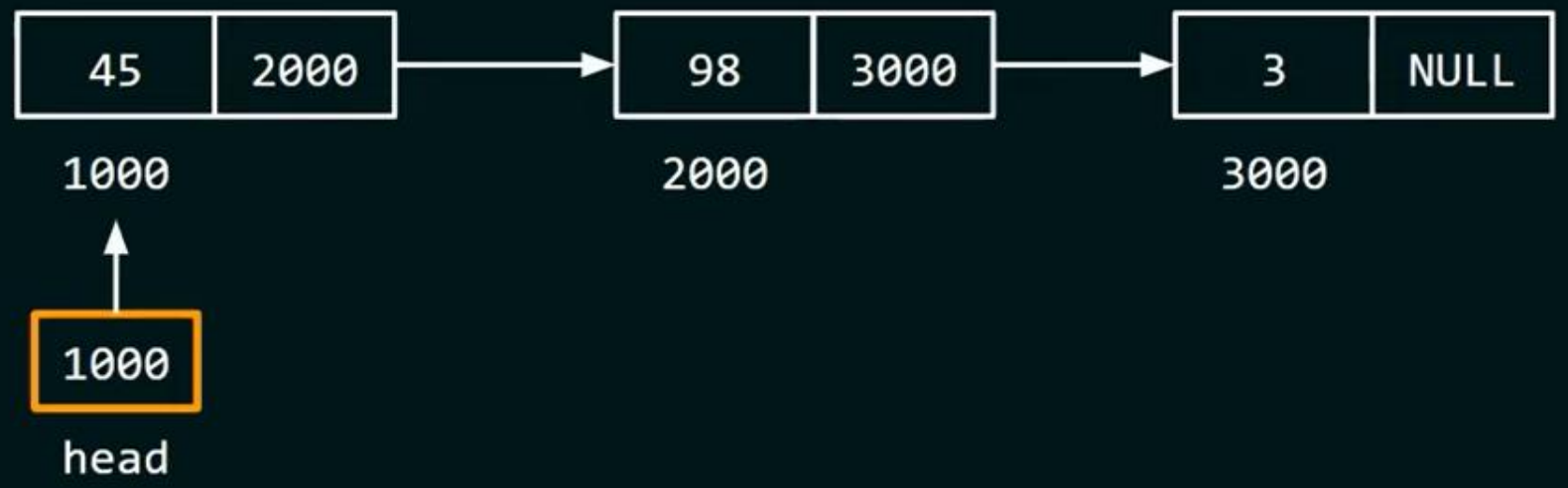
FINAL RESULT

Position: 2

```
previous->link = current->link
free(current)
current = NULL
```

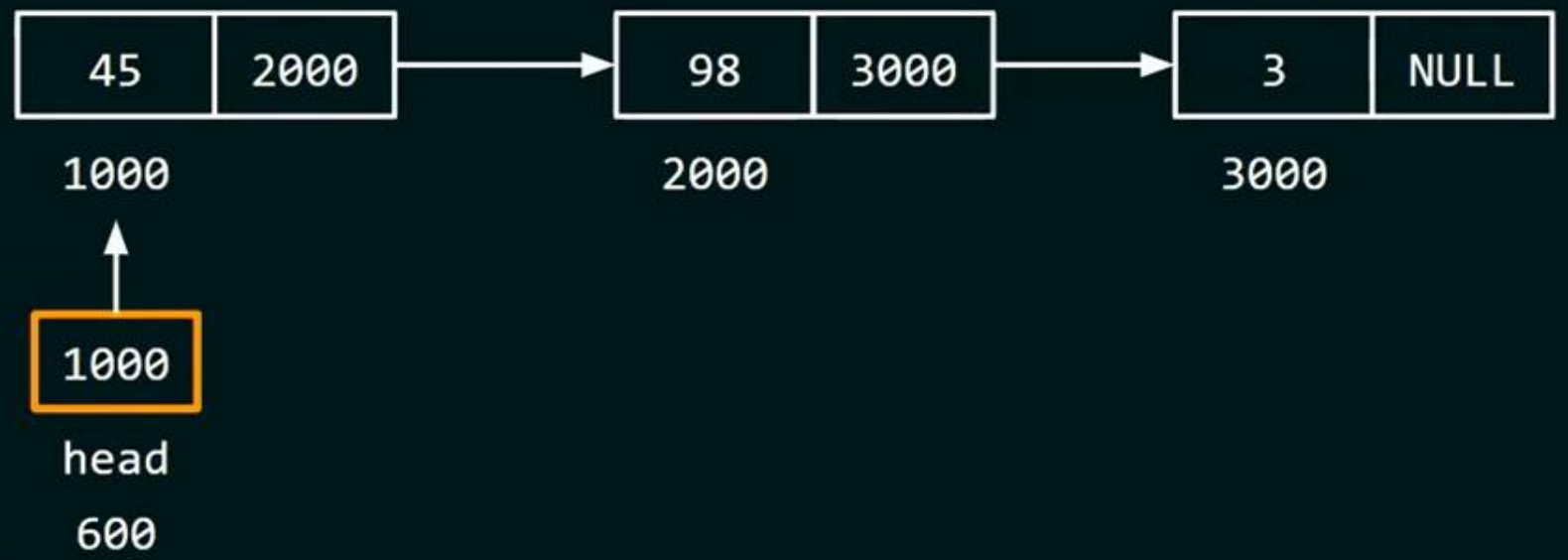
Program

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main() {
    int position = 2;
    del_pos(&head, position);
    ptr = head;
    while(ptr != NULL)
    {
        printf("%d ", ptr->data);
        ptr = ptr->link;
    }
     return 0;
}
```



Program

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main() {
    int position = 2;
    del_pos(&head, position);
    ptr = head;
    while(ptr != NULL)
    {
        printf("%d ", ptr->data);
        ptr = ptr->link;
    }
     return 0;
}
```



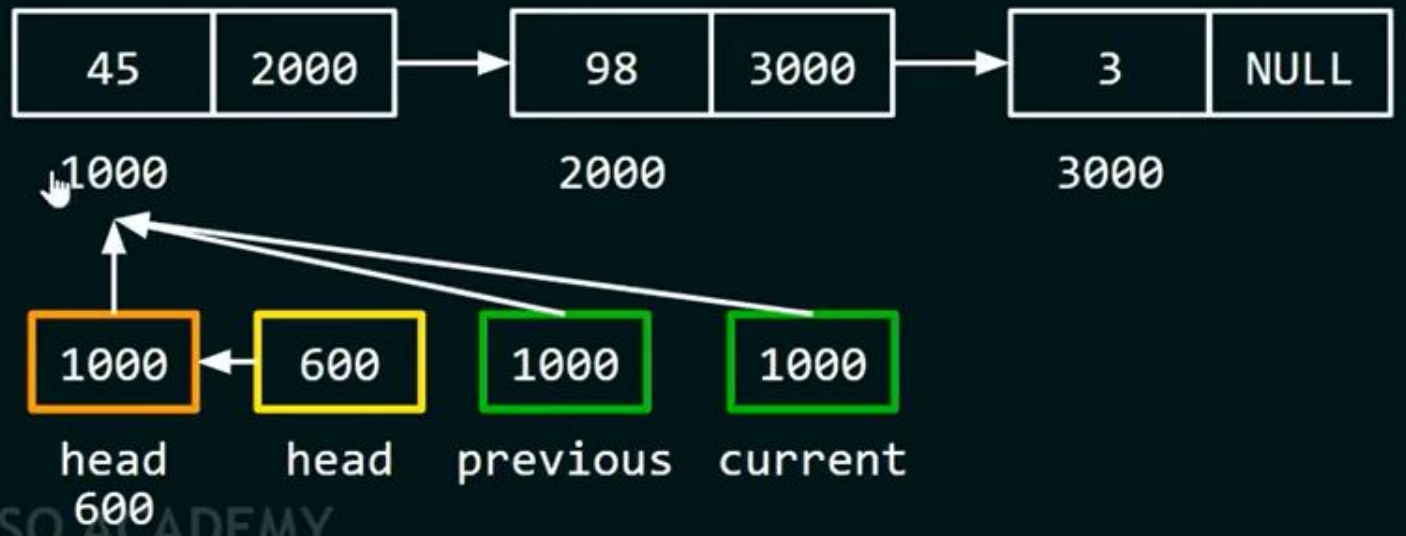
Function

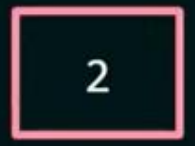
```
void del_pos(struct node **head, int position)
{
    struct node *current = *head;
    struct node *previous = *head;
    if(*head == NULL)
        printf("List is already empty!");
    else if(position == 1)
    {
        *head = current->link;
        free(current);
        current = NULL;
    }
    else
    {
        while(position != 1)
        {
            previous = current;
            current = current->link;
            position--;
        }
        previous->link = current->link;
        free(current);
        current = NULL;
    }
}
```


Function

```
void del_pos(struct node **head, int position)
{
    struct node *current = *head;
    struct node *previous = *head;
    if(*head == NULL)
        printf("List is already empty!");
    else if(position == 1)
    {
        *head = current->link;
        free(current);
        current = NULL;
    }
```

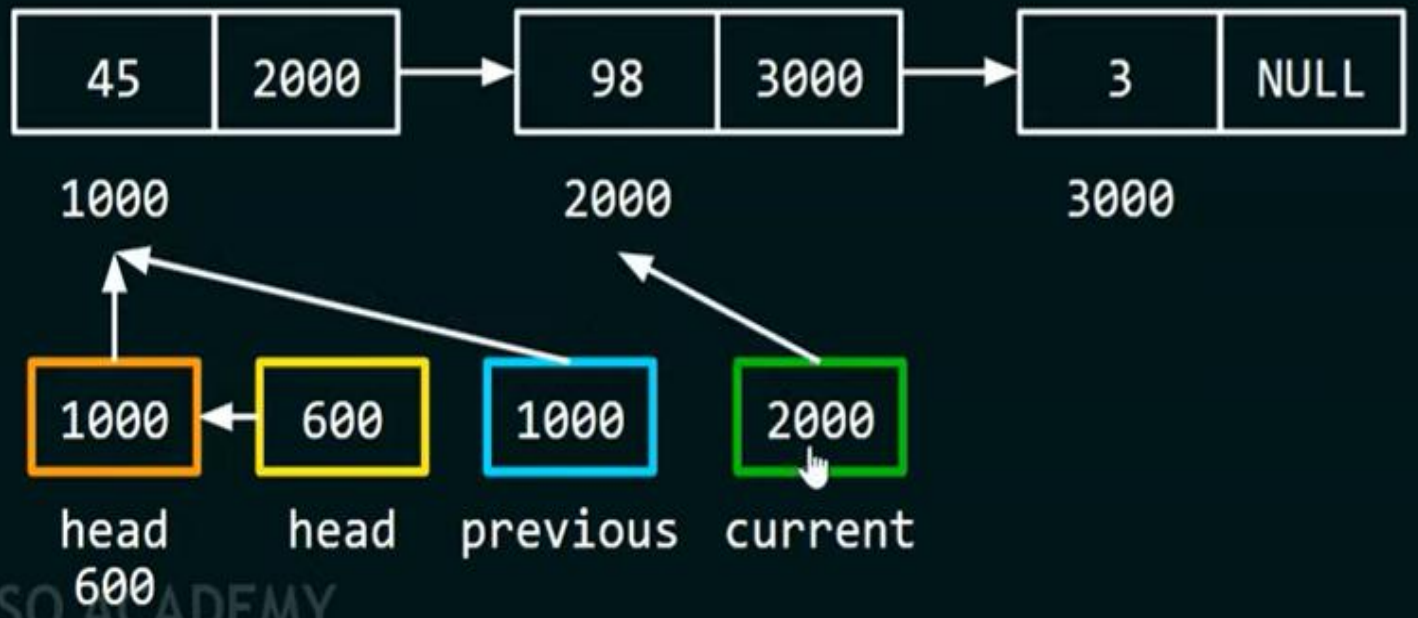
```
    else
        {
            while(position != 1)
            {
                previous = current;
                current = current->link;
                position--;
            }
            previous->link = current->link;
            free(current);
            current = NULL;
        }
}
```





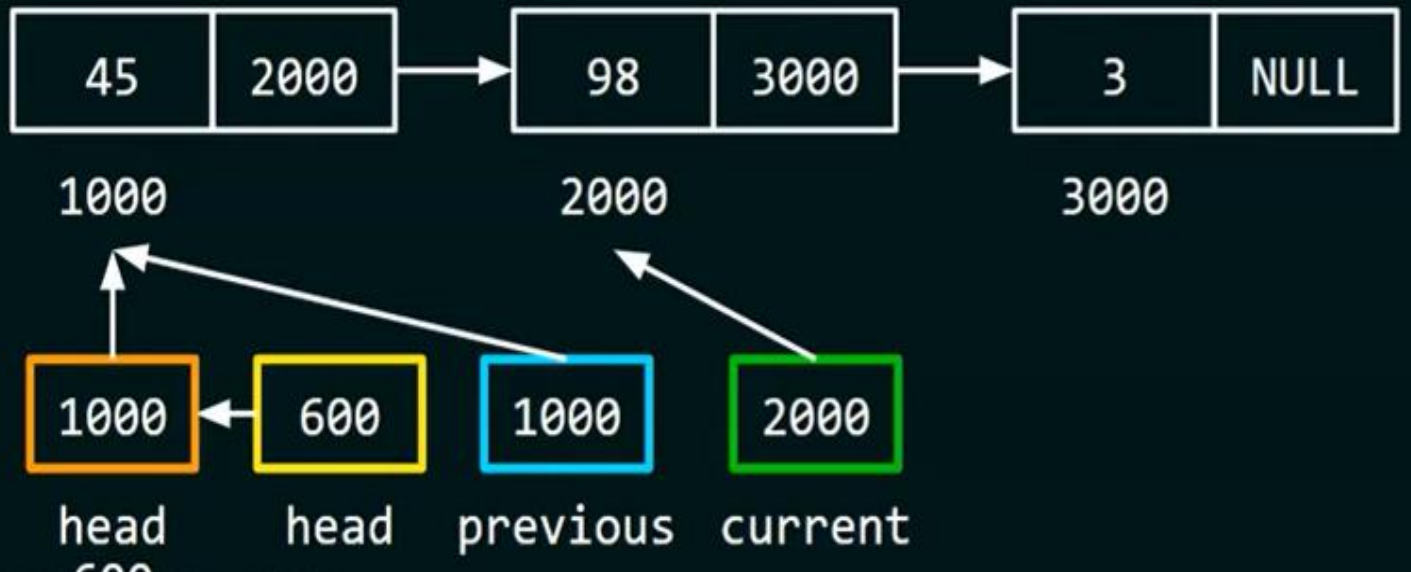
```
void del_pos(struct node **head, int position)
{
    struct node *current = *head;
    struct node *previous = *head;
    if(*head == NULL)
        printf("List is already empty!");
    else if(position == 1)
    {
        *head = current->link;
        free(current);
        current = NULL;
    }
```

```
    else
    {
        while(position != 1)
        {
            previous = current;
            current = current->link;
            position--;
        }
        previous->link = current->link;
        free(current);
        current = NULL;
    }
}
```



```
void del_pos(struct node **head, int position)
{
    struct node *current = *head;
    struct node *previous = *head;
    if(*head == NULL)
        printf("List is already empty!");
    else if(position == 1)
    {
        *head = current->link;
        free(current);
        current = NULL;
    }
```

```
    else
        {
            while(position != 1)
            {
                previous = current;
                current = current->link;
                position--;
            }
            previous->link = current->link;
            free(current);
            current = NULL;
        }
    }
```



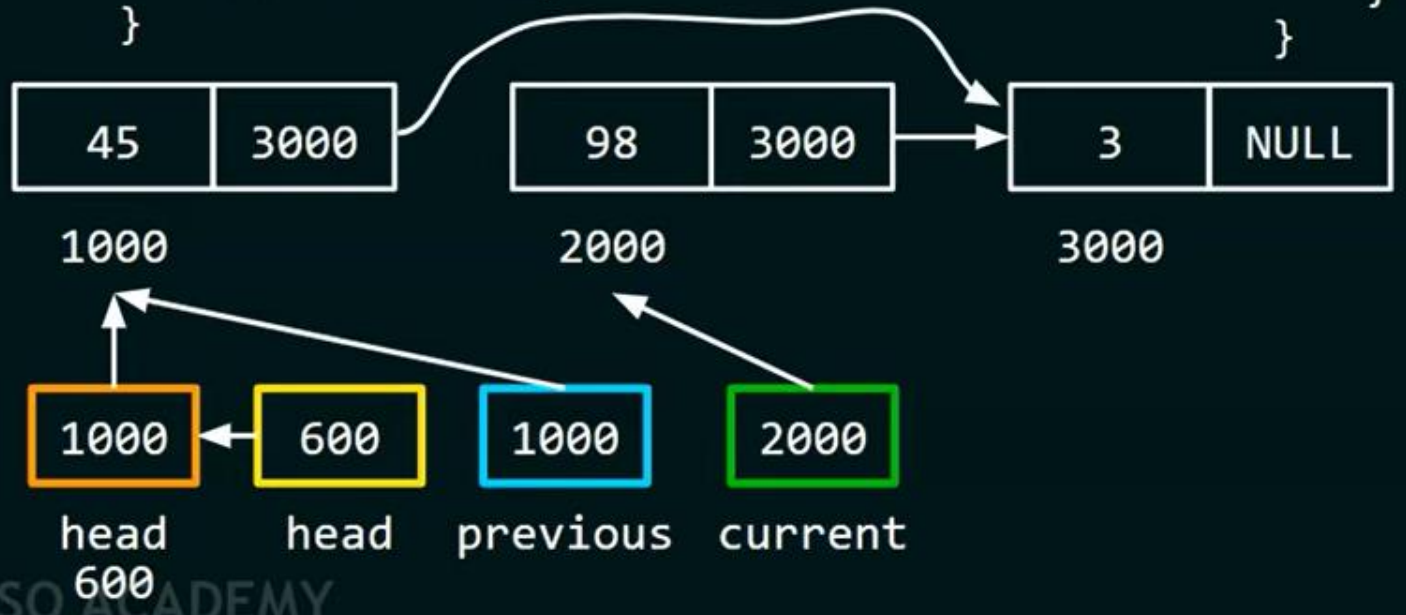
```
void del_pos(struct node **head, int position)
{
    struct node *current = *head;
    struct node *previous = *head;
    if(*head == NULL)
        printf("List is already empty!");
    else if(position == 1)
    {
        *head = current->link;
        free(current);
        current = NULL;
    }
```

```
    else
        {
            while(position != 1)
            {
                previous = current;
                current = current->link;
                position--;
            }
            previous->link = current->link;
            free(current);
            current = NULL;
        }
}
```

Function

```
void del_pos(struct node **head, int position)
{
    struct node *current = *head;
    struct node *previous = *head;
    if(*head == NULL)
        printf("List is already empty!");
    else if(position == 1)
    {
        *head = current->link;
        free(current);
        current = NULL;
    }
```

```
    else
        {
            while(position != 1)
            {
                previous = current;
                current = current->link;
                position--;
            }
            previous->link = current->link;
            free(current);
            current = NULL;
        }
}
```



Function

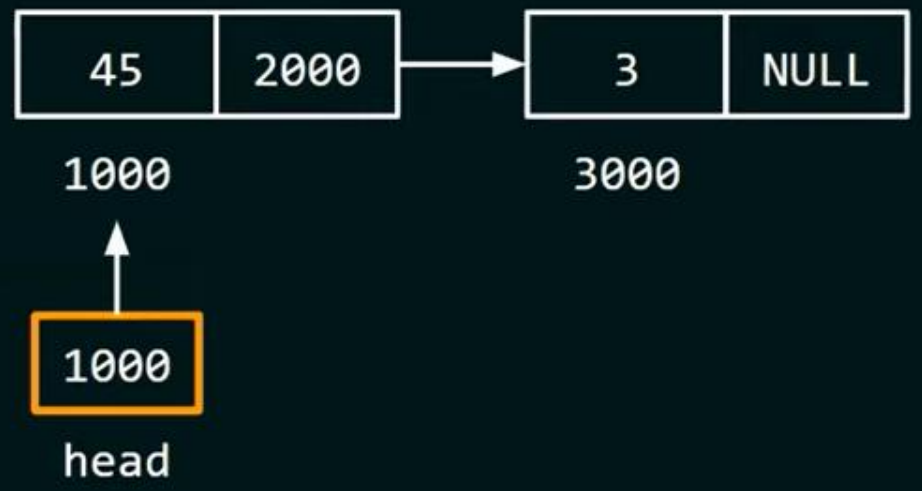
```
void del_pos(struct node **head, int position)
{
    struct node *current = *head;
    struct node *previous = *head;
    if(*head == NULL)
        printf("List is already empty!");
    else if(position == 1)
    {
        *head = current->link;
        free(current);
        current = NULL;
    }
    else
        {
```

```
while(position != 1)
{
    previous = current;
    current = current->link;
    position--;
}
previous->link = current->link;
free(current);
current = NULL;
```

```
        }
}
```

Program

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *link;
};

int main() {
    int position = 2;
    del_pos(&head, position);
    ptr = head;
    while(ptr != NULL)
    {
        printf("%d ", ptr->data);
        ptr = ptr->link;
    }
     return 0;
}
```

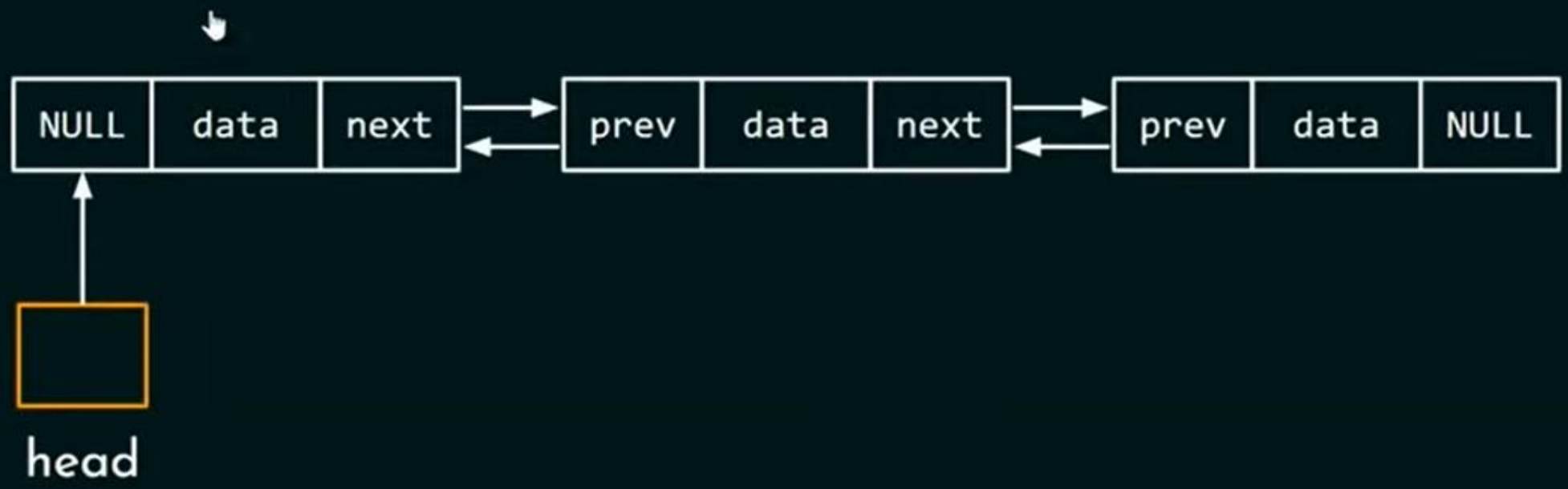


PROGRAMMING AND DATA STRUCTURES

Introduction to Doubly Linked List

Doubly Linked List

A doubly linked list is different from a singly linked list in a way that each node has an extra pointer that points to the previous node, together with the next pointer and data similar to singly linked list.



Doubly Linked List

Add just one more field to the self-referential structure.

Singly Linked List

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

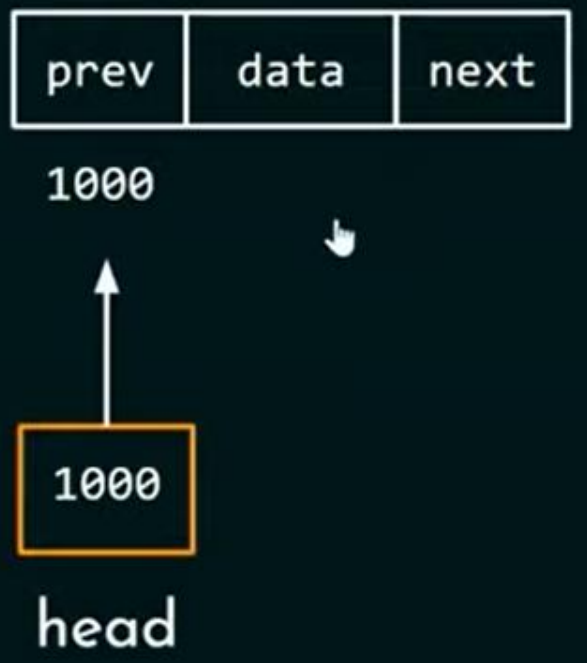
Doubly linked list

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

CREATING A NODE OF THE DOUBLY LINKED LIST

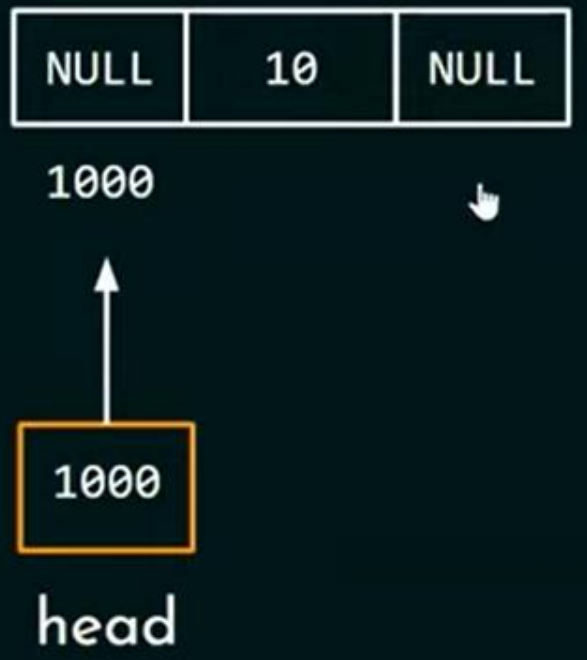
Creating a node

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

int main()
{
     struct node *head = malloc(sizeof(struct node));
}
```



Creating a node

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

int main()
{
     struct node *head = malloc(sizeof(struct node));
     head->prev = NULL;
     head->data = 10;
     head->next = NULL;
}
```



PROGRAMMING AND DATA STRUCTURES

Doubly Linked List: Insertion in an empty list

Insertion in an empty list

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    struct node* prev;
    int data;
    struct node* next;
};

int main() {
    struct node* head = NULL;
     head = addToEmpty(head, 45);
     printf("%d", head->data);
     return 0;
}
```

```c
struct node* addToEmpty(struct node* head, int data)
{
    struct node* temp = malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;
    head = temp;
    return head;
}
```


Insertion in an empty list

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    struct node* prev;
    int data;
    struct node* next;
};

int main() {
    struct node* head = NULL;
    head = addToEmpty(head, 45);
    printf("%d", head->data);
    return 0;
}
```

```c
struct node* addToEmpty(struct node* head, int data)
{
    struct node* temp = malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;
    head = temp;
    return head;
}
```

Insertion in an empty list

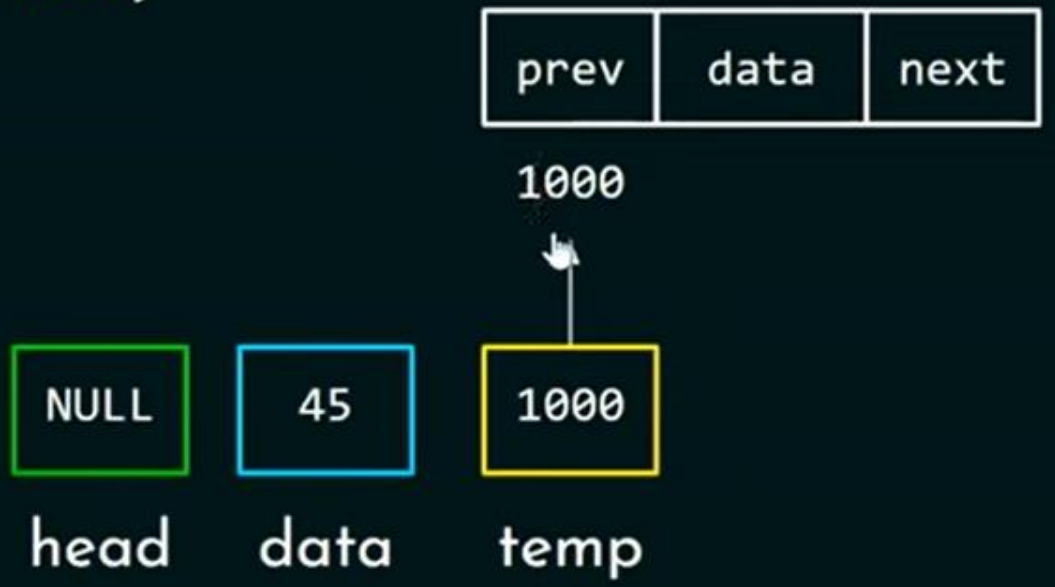
```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    struct node* prev;
    int data;
    struct node* next;
};

int main() {
    struct node* head = NULL;
    head = addToEmpty(head, 45);
    printf("%d", head->data);
    return 0;
}
```

```c
struct node* addToEmpty(struct node* head, int data)
{
    struct node* temp = malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;
    head = temp;
    return head;
}
```

Insertion in an empty list

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    struct node* prev;
    int data;
    struct node* next;
};

int main() {
    struct node* head = NULL;
     head = addToEmpty(head, 45);
     printf("%d", head->data);
     return 0;
}
```

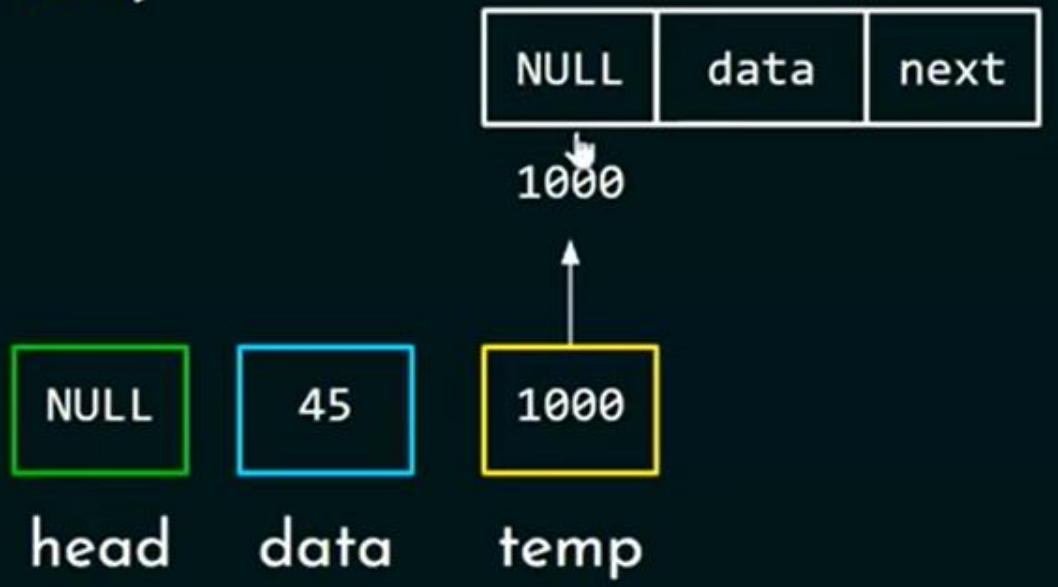

```c
struct node* addToEmpty(struct node* head, int data)
{
    struct node* temp = malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;
    head = temp;
    return head;
}
```



Insertion in an empty list

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    struct node* prev;
    int data;
    struct node* next;
};

int main() {
    struct node* head = NULL;
     head = addToEmpty(head, 45);
     printf("%d", head->data);
     return 0;
}
```

```c
struct node* addToEmpty(struct node* head, int data)
{
    struct node* temp = malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;
    head = temp;
    return head;
}
```



Insertion in an empty list

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    struct node* prev;
    int data;
    struct node* next;
};

int main() {
    struct node* head = NULL;
     head = addToEmpty(head, 45);
     printf("%d", head->data);
     return 0;
}
```


```c
struct node* addToEmpty(struct node* head, int data)
{
    struct node* temp = malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;
    head = temp;
    return head;
}
```

Insertion in an empty list

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    struct node* prev;
    int data;
    struct node* next;
};

int main() {
    struct node* head = NULL;
     head = addToEmpty(head, 45);
     printf("%d", head->data);
     return 0;
}
```

head

```c
struct node* addToEmpty(struct node* head, int data)
{
    struct node* temp = malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;
    head = temp;
    return head;
}
```

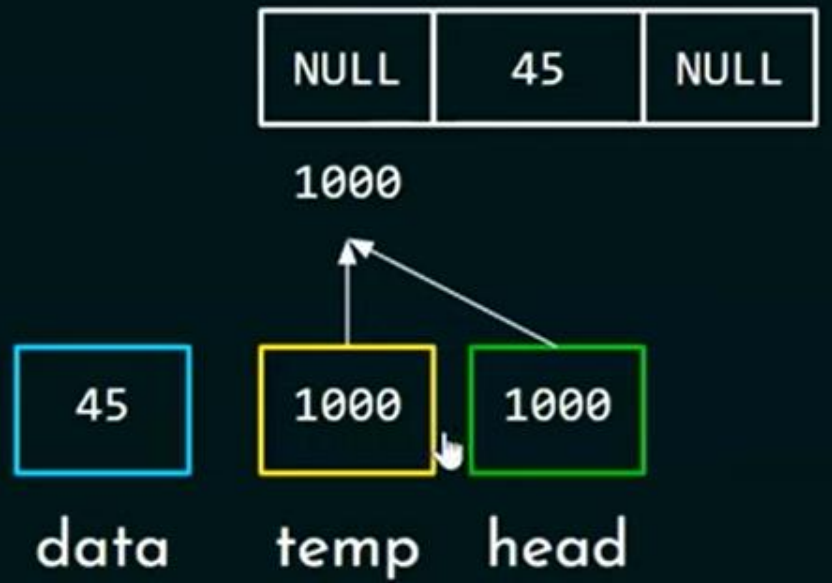
1000

head data temp

Insertion in an empty list

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    struct node* prev;
    int data;
    struct node* next;
};

int main() {
    struct node* head = NULL;
     head = addToEmpty(head, 45);
     printf("%d", head->data);
     return 0;
}
```


```c
struct node* addToEmpty(struct node* head, int data)
{
    struct node* temp = malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;
    head = temp;
    return head;
}
```



Insertion in an empty list

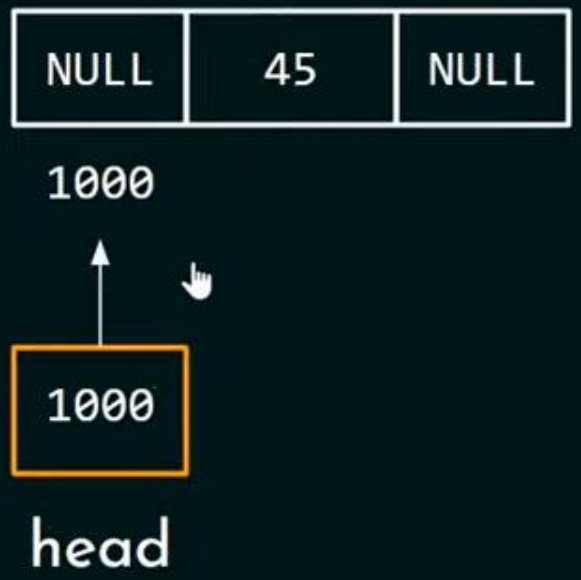
```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    struct node* prev;
    int data;
    struct node* next;
};

int main() {
    struct node* head = NULL;
     head = addToEmpty(head, 45);
     printf("%d", head->data);
     return 0;
}
```

```c
struct node* addToEmpty(struct node* head, int data)
{
    struct node* temp = malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;
    head = temp;
    return head;
}
```

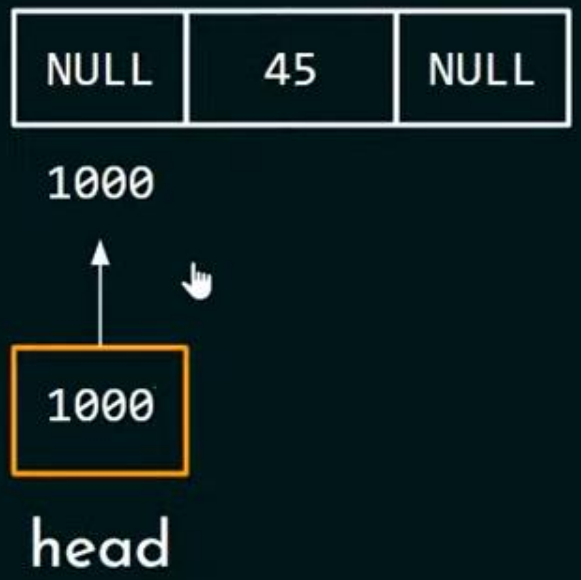
PROGRAMMING AND DATA STRUCTURES

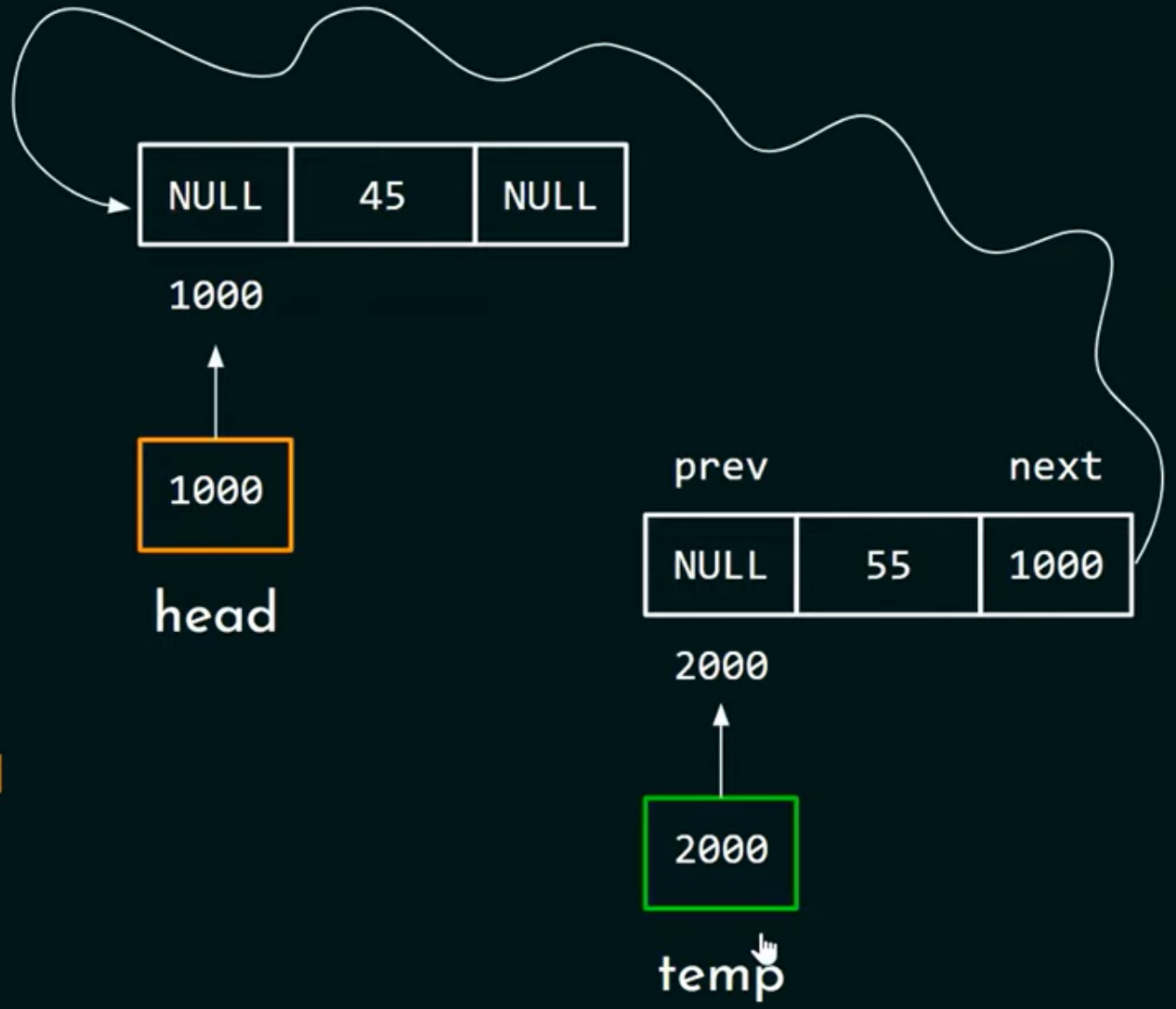
Doubly Linked List:
Insertion at the beginning of the list

Initial State

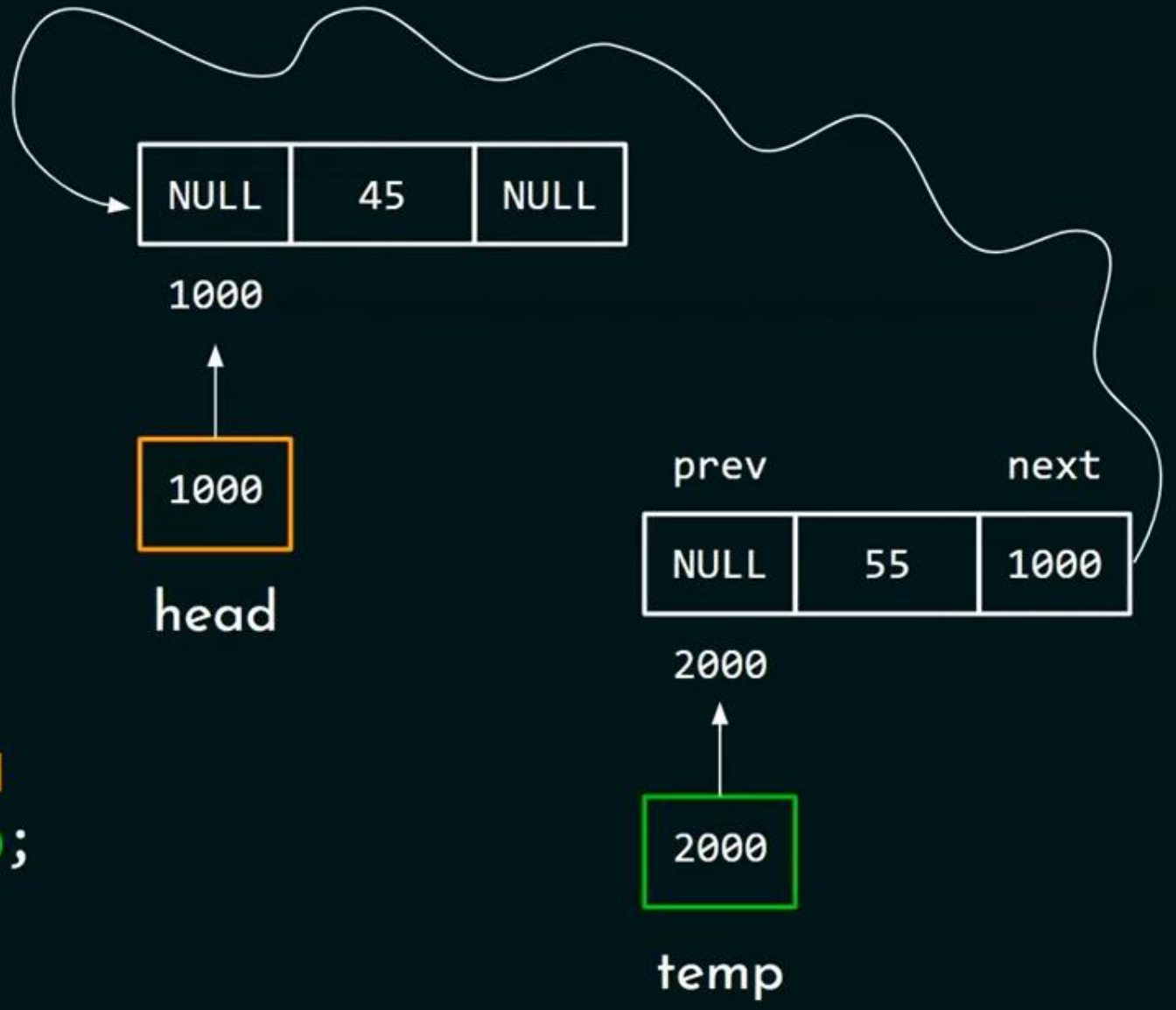


Initial State

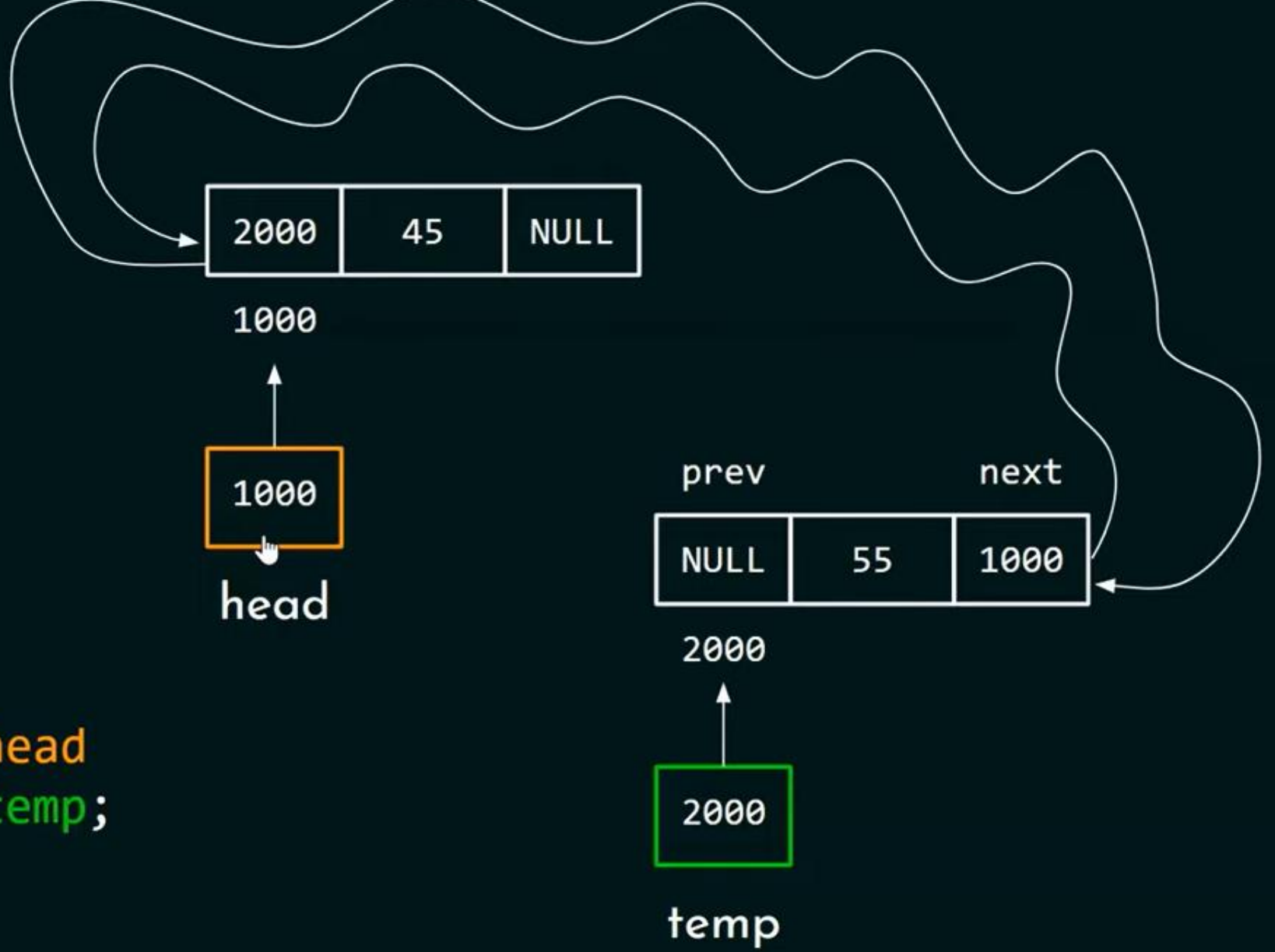





```
temp->next = head
```

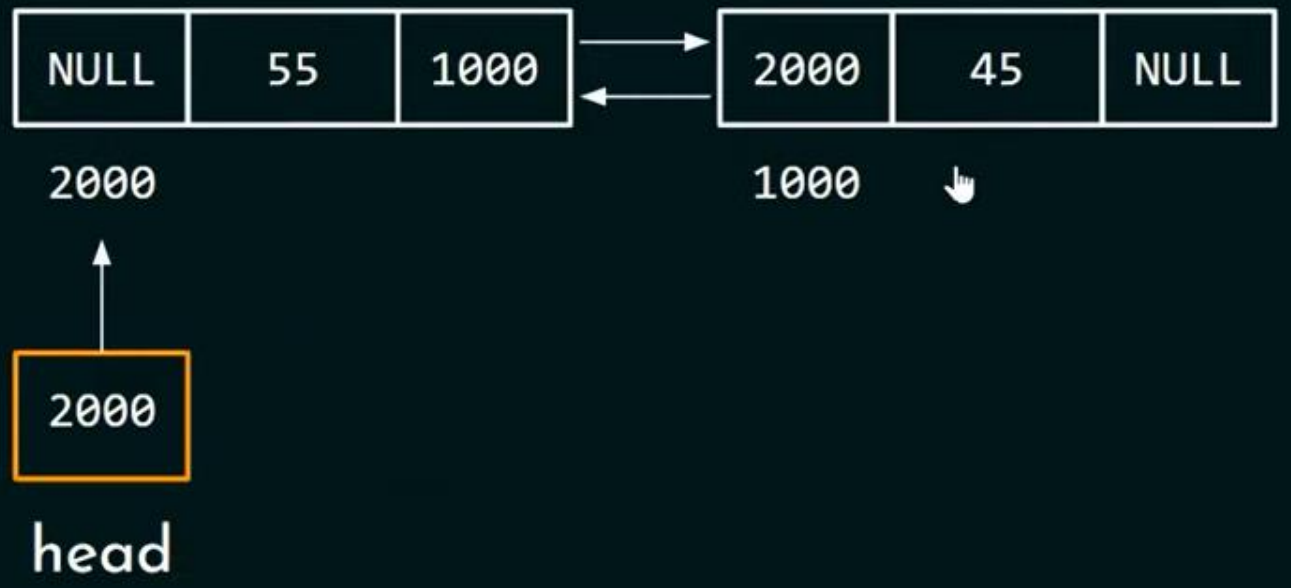



```
temp->next = head
head->prev = temp;
```




```
temp->next = head
head->prev = temp;
head = temp;
```

Resultant List



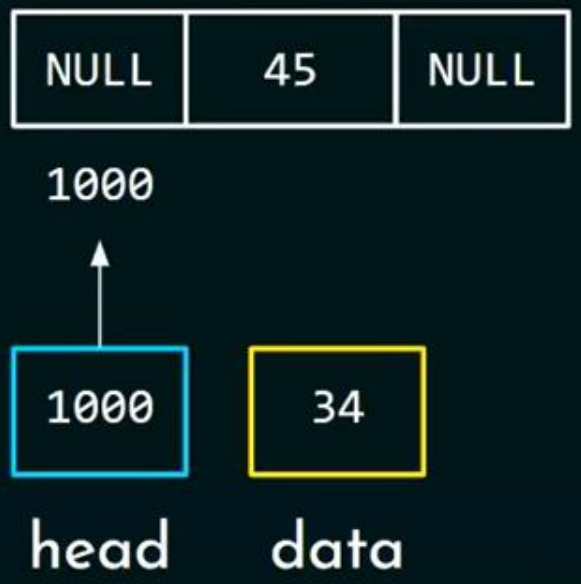
Program

```c
#include <stdio.h>
#include <stdlib.h>

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

```c
int main() {
    struct node* head = NULL;
    struct node* ptr;
    head = addToEmpty(head, 45);
    head = addAtBeg(head, 34);

    ptr = head;
    while(ptr != NULL)
    {
        printf("%d ", ptr->data);
        ptr = ptr->next;
    }
    return 0;
}
```

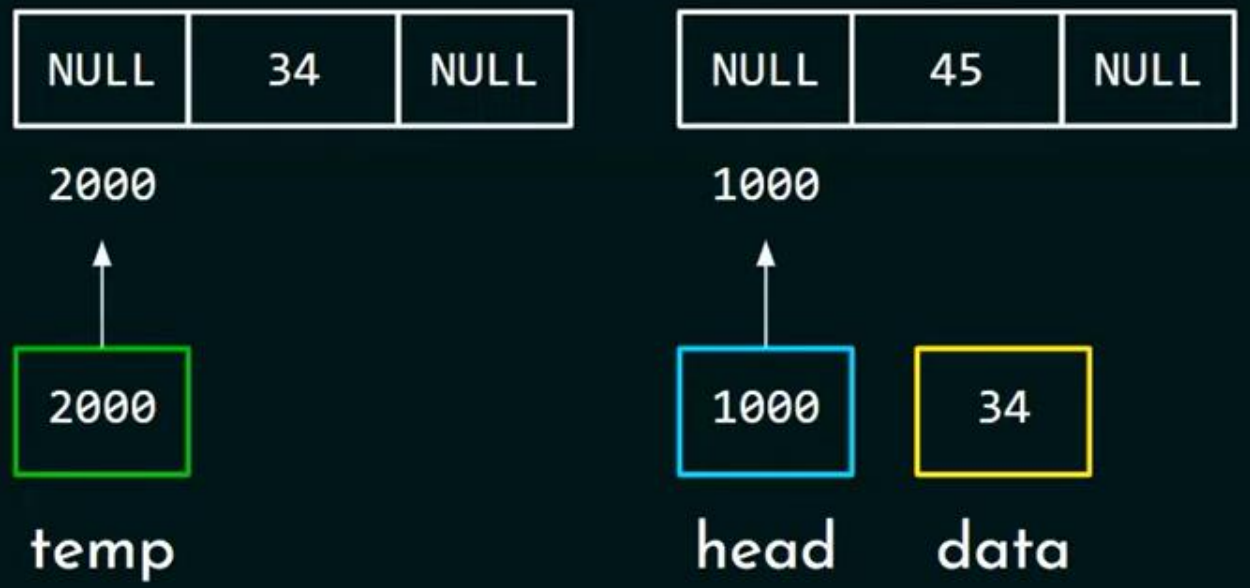

Program

```
struct node* addAtBeg(struct node* head, int data)
{
    struct node* temp = malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;
    temp->next = head;
    head->prev = temp;
    head = temp;
    return head;
}
```



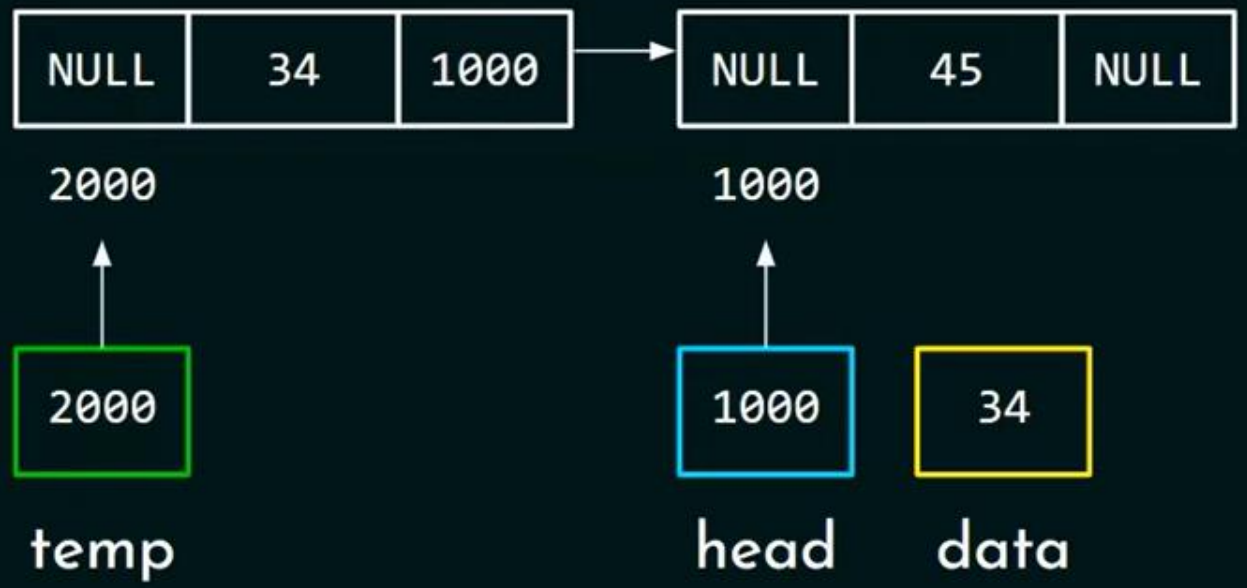
Program

```
struct node* addAtBeg(struct node* head, int data)
{
    struct node* temp = malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;
    temp->next = head;
    head->prev = temp;
    head = temp;
    return head;
}
```



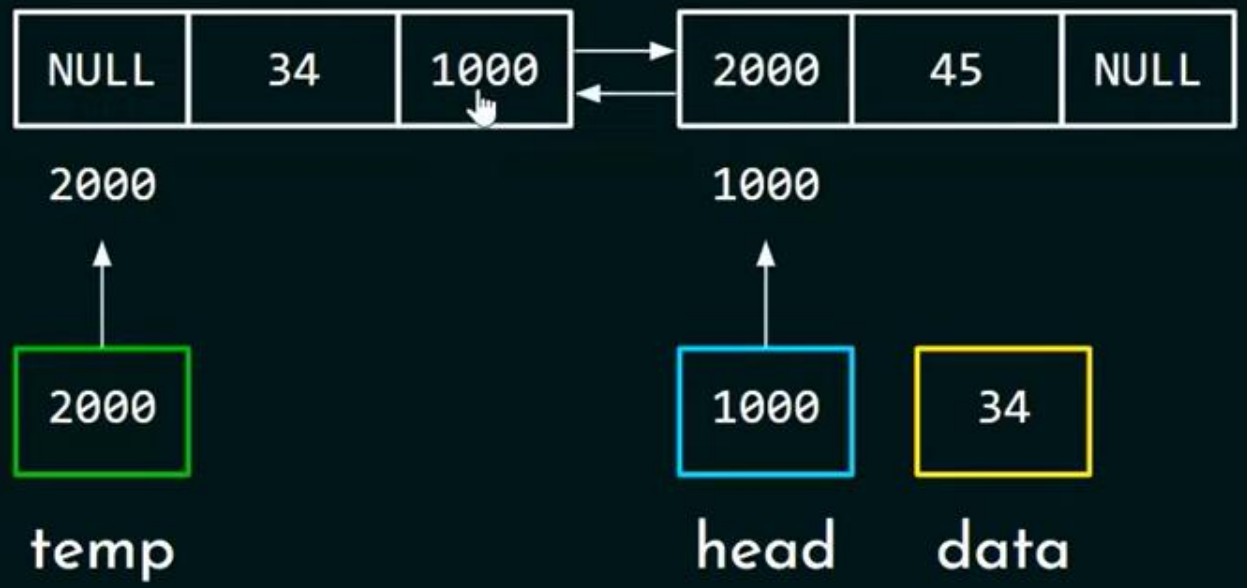
Program

```
struct node* addAtBeg(struct node* head, int data)
{
    struct node* temp = malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;
    temp->next = head;
    head->prev = temp;
    head = temp;
    return head;
}
```



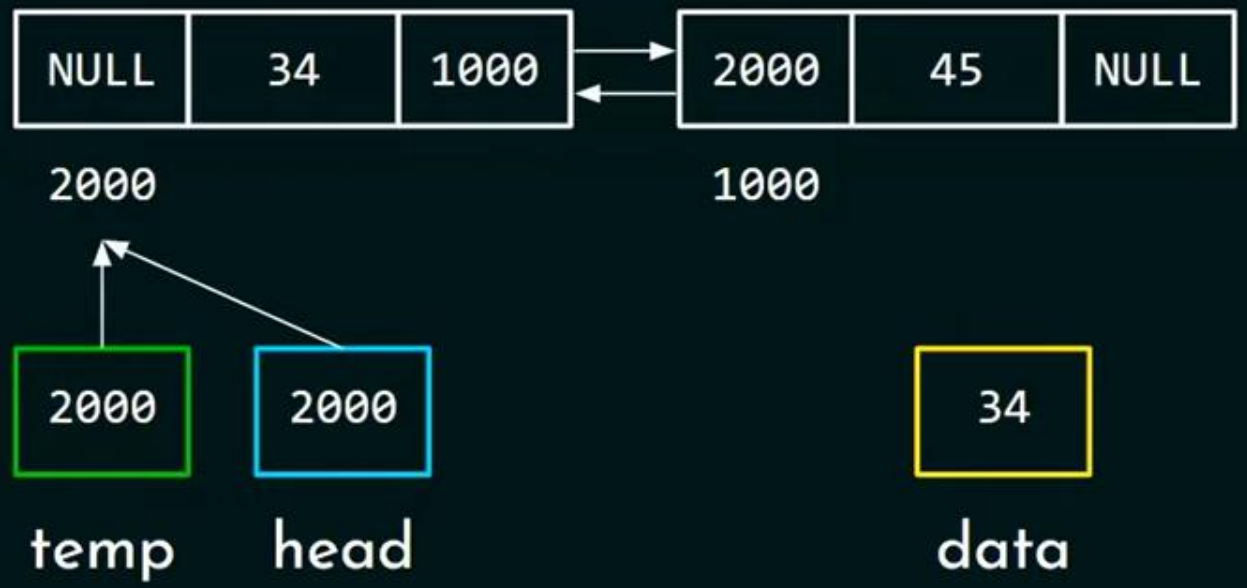
Program

```
struct node* addAtBeg(struct node* head, int data)
{
    struct node* temp = malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;
    temp->next = head;
    head->prev = temp;
    head = temp;
    return head;
}
```



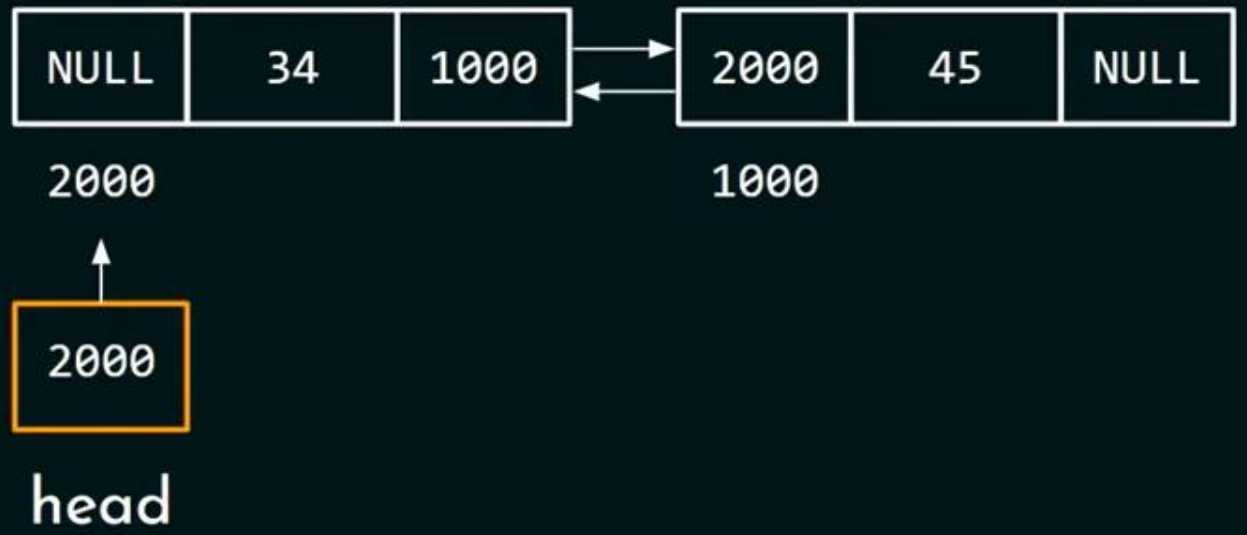
Program

```
struct node* addAtBeg(struct node* head, int data)
{
    struct node* temp = malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;
    temp->next = head;
    head->prev = temp;
    head = temp;
    return head;
}
```



Program

```
#include <stdio.h>
#include <stdlib.h>

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



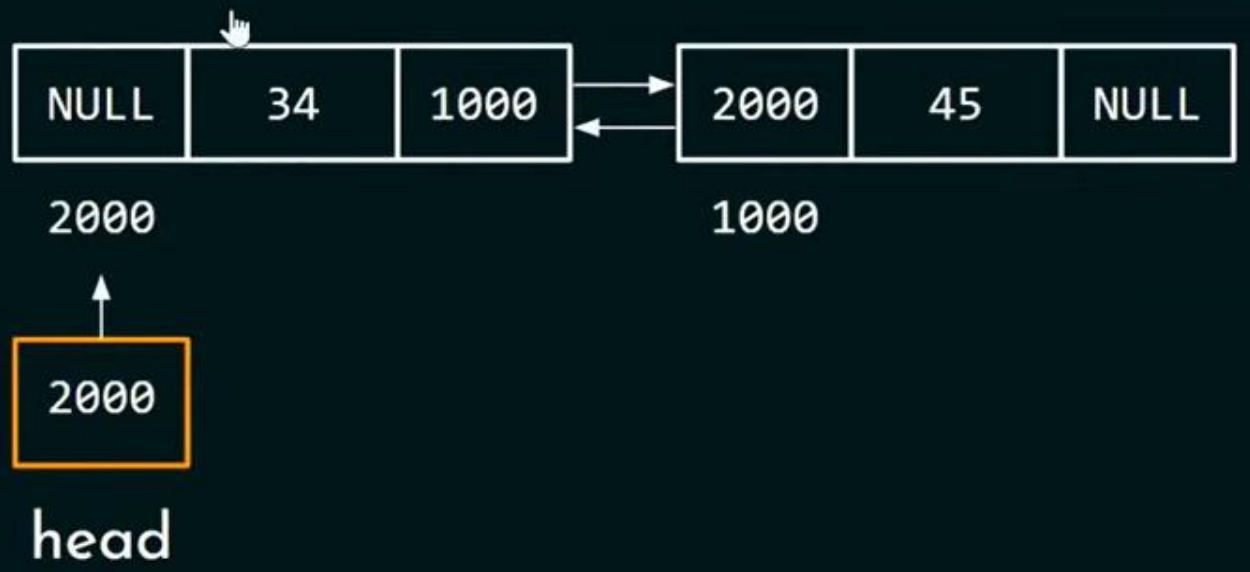

```
int main() {
    struct node* head = NULL;
    struct node* ptr;
    head = addToEmpty(head, 45);
    head = addAtBeg(head, 34);

    ptr = head;
    while(ptr != NULL)
    {
        printf("%d ", ptr->data);
        ptr = ptr->next;
    }
    return 0;
}
```

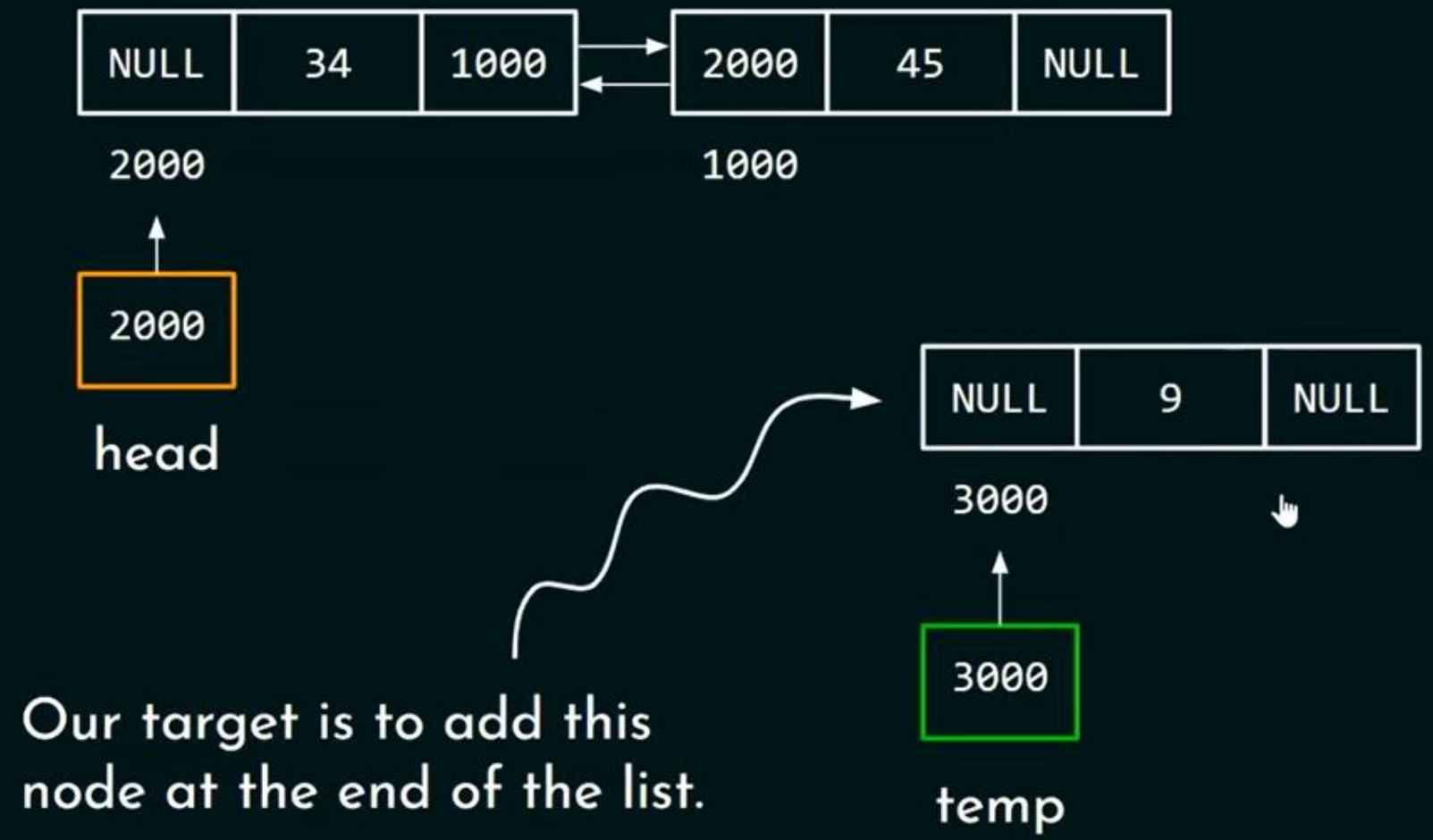
Output: 34 45

Programming and Data Structures

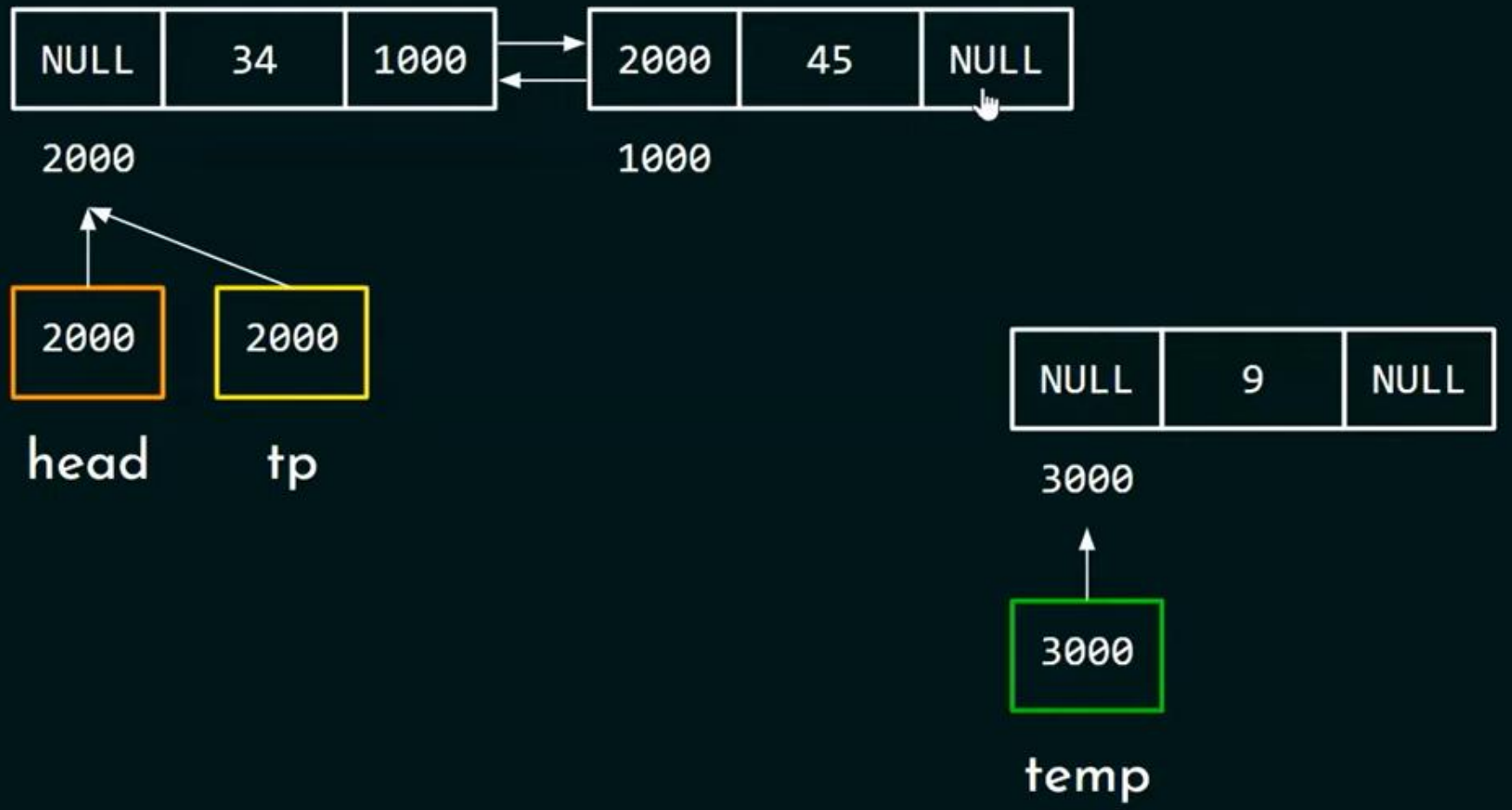
Doubly Linked List:
Insertion at the end of the list

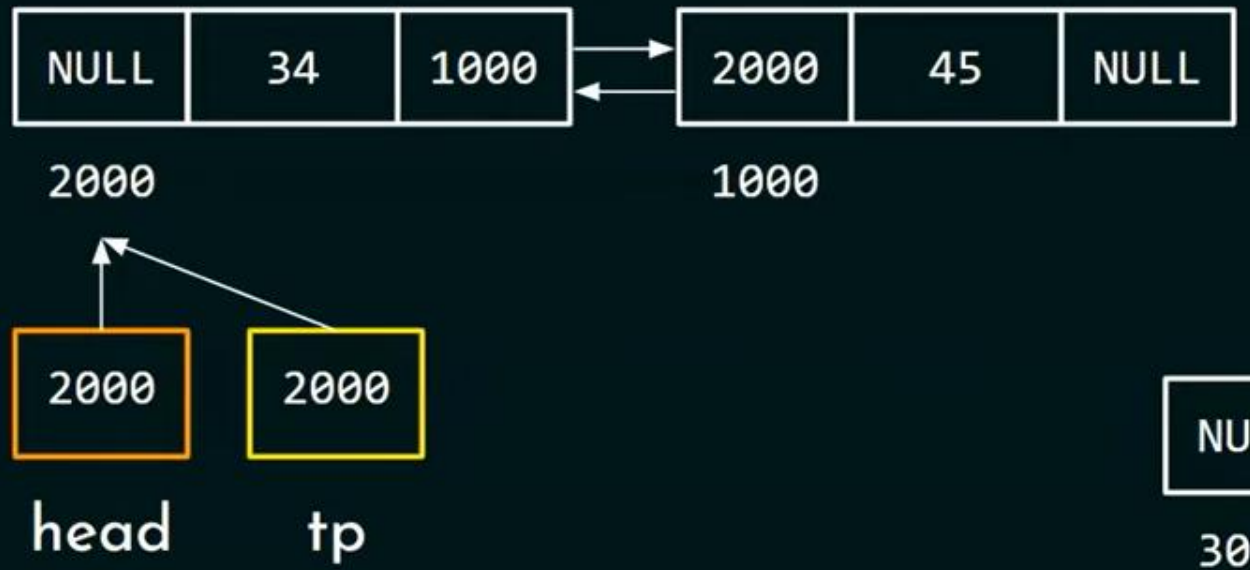
Initial State

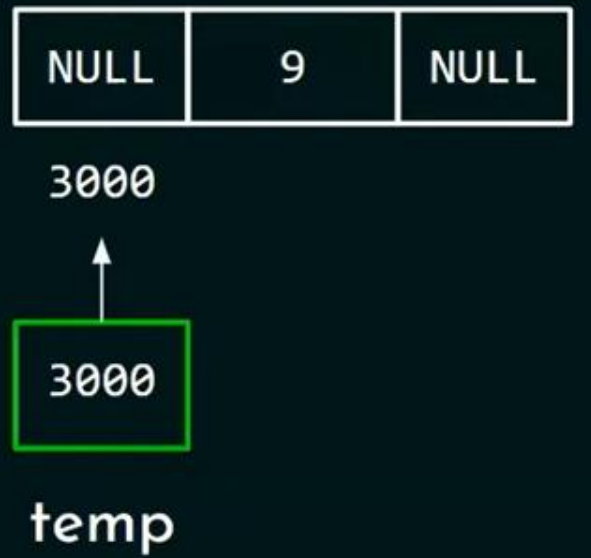




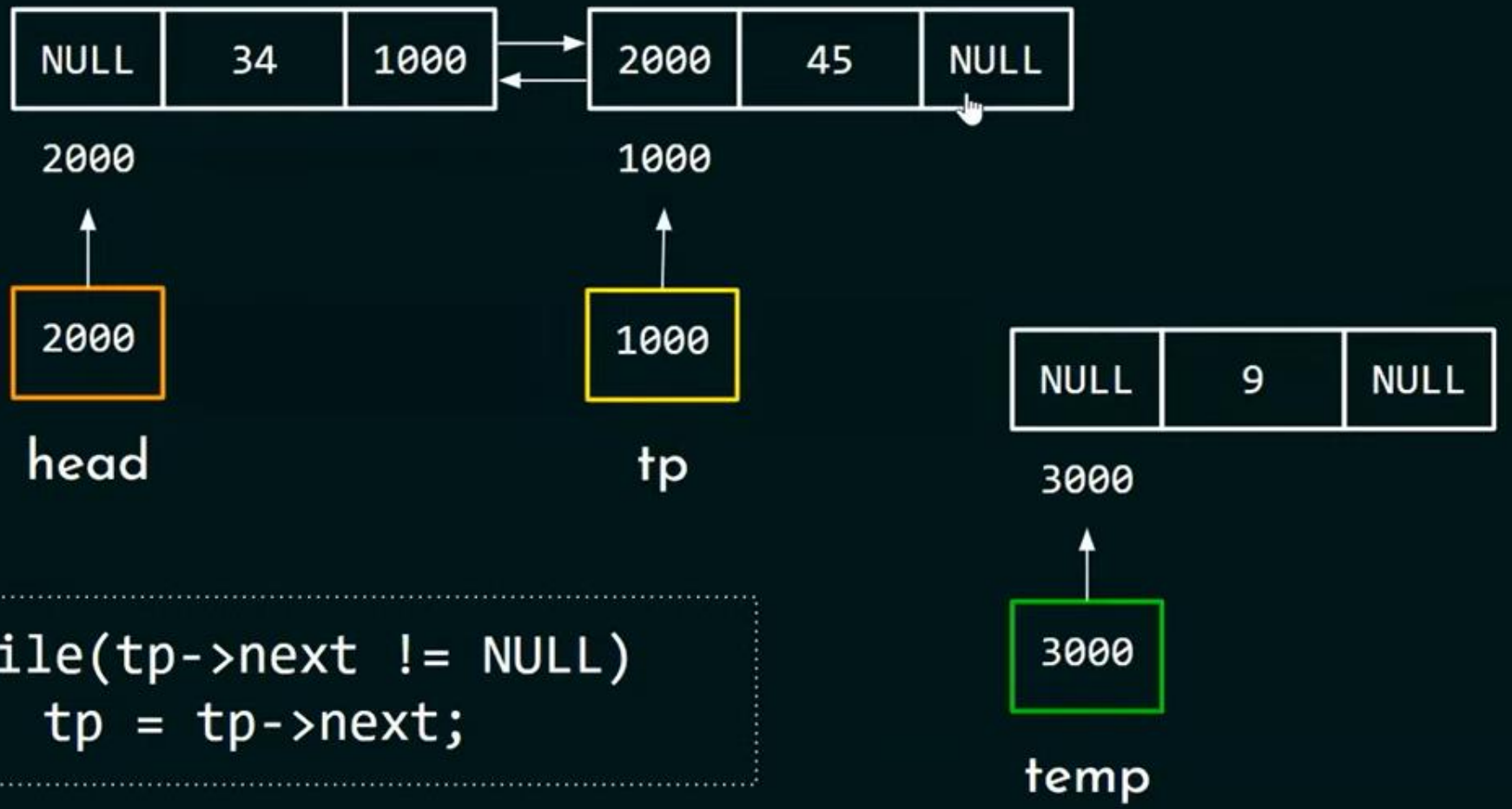
Step 1: Traversal



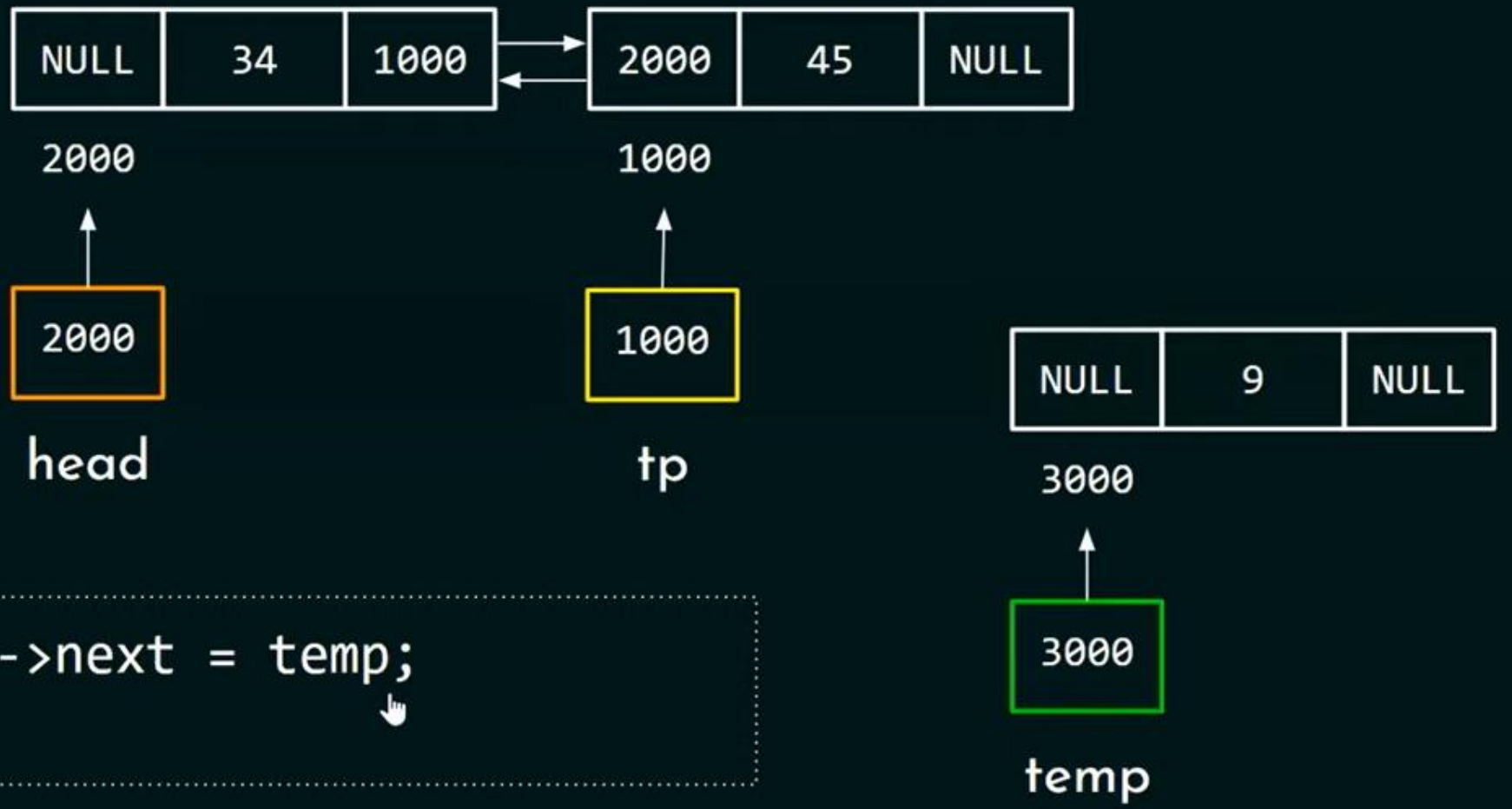


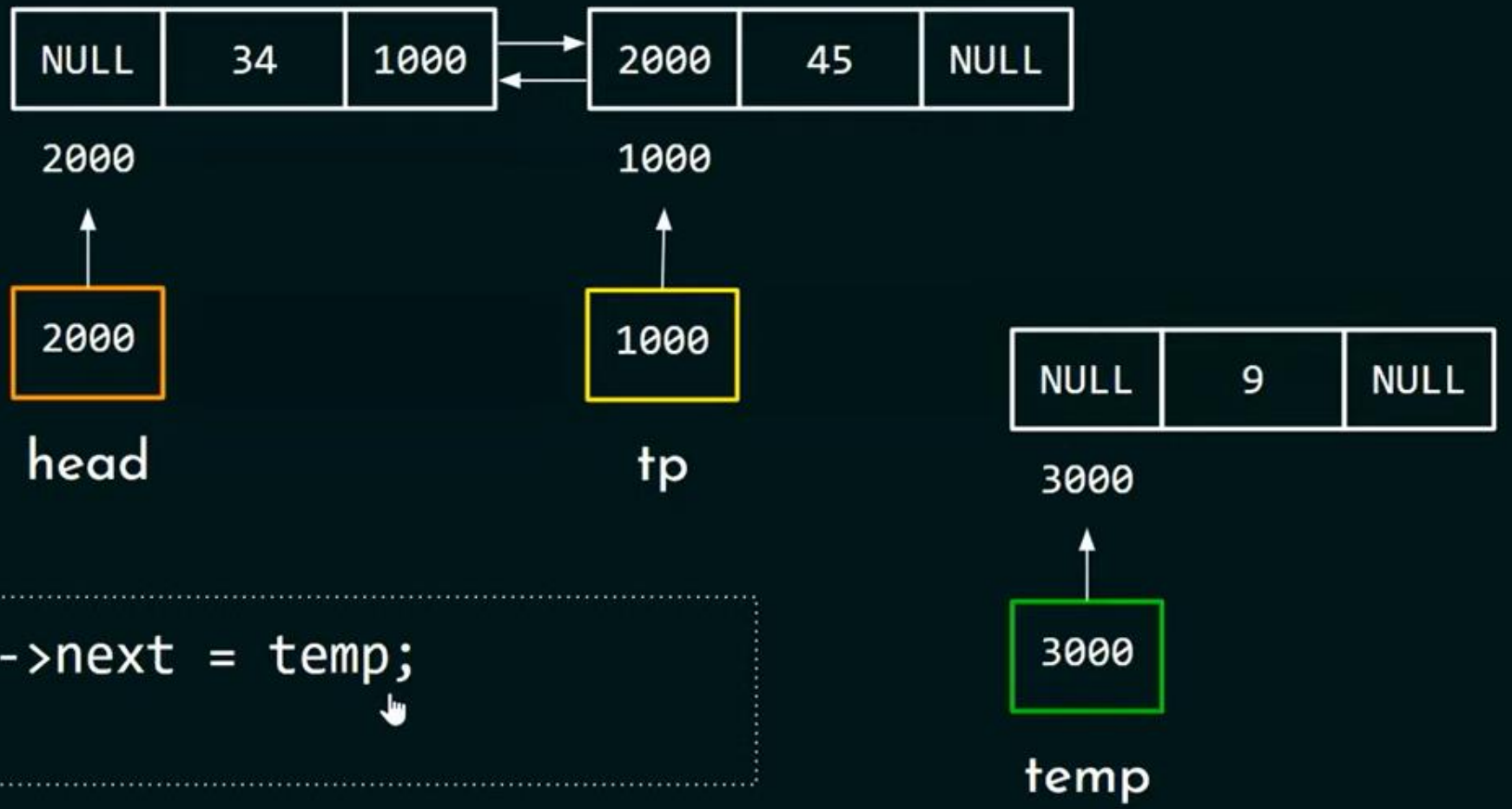


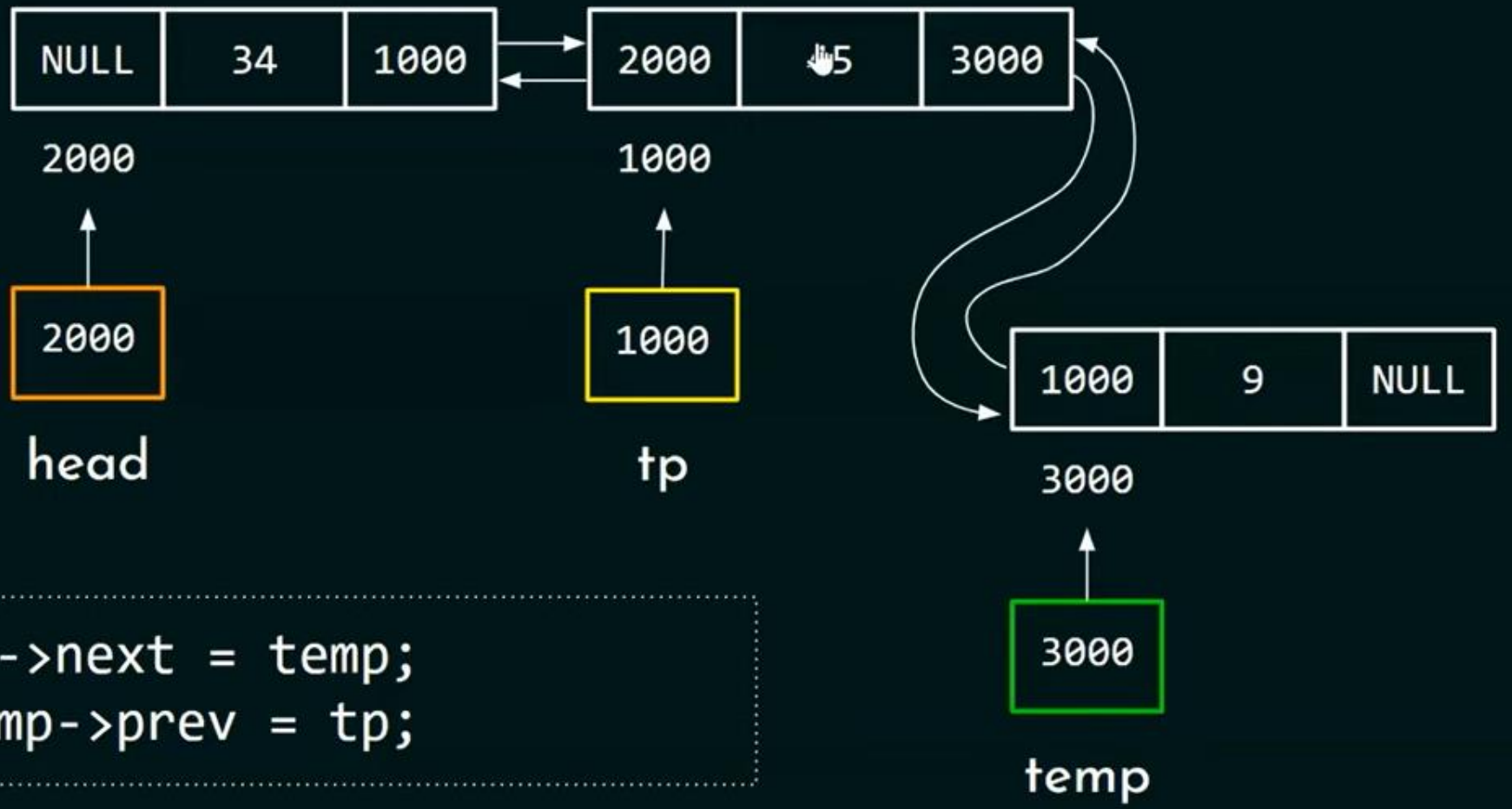

```
while(tp->next != NULL)
    tp = tp->next;
```



Step 2: Attach new node to the end node of the list

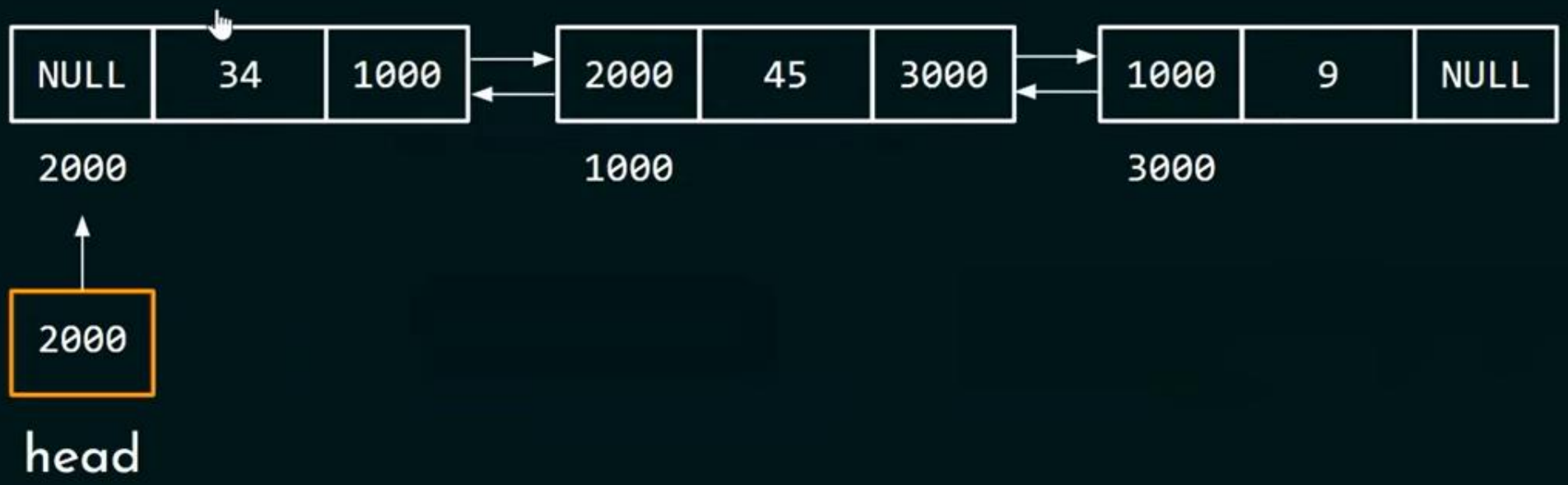






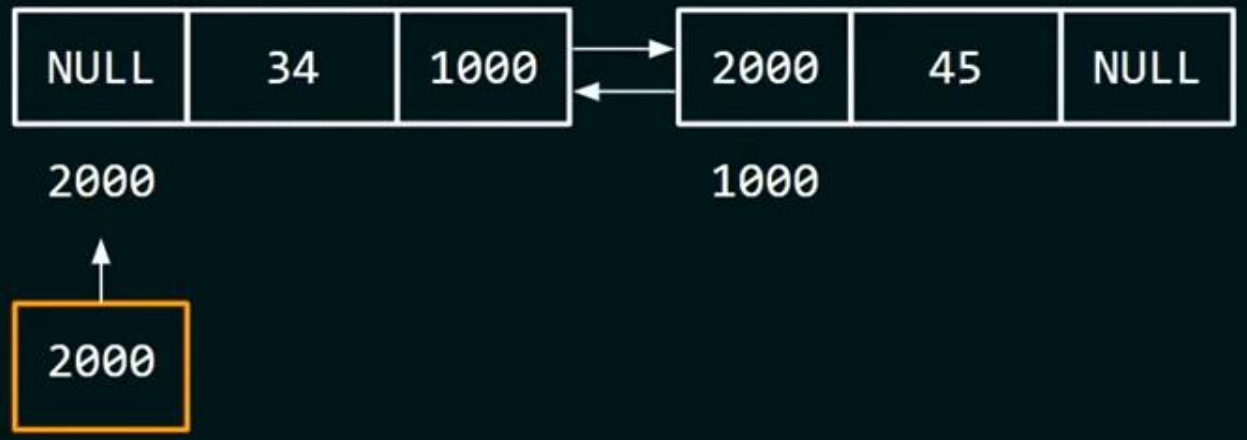

```
tp->next = temp;
temp->prev = tp;
```

Resultant list



Program

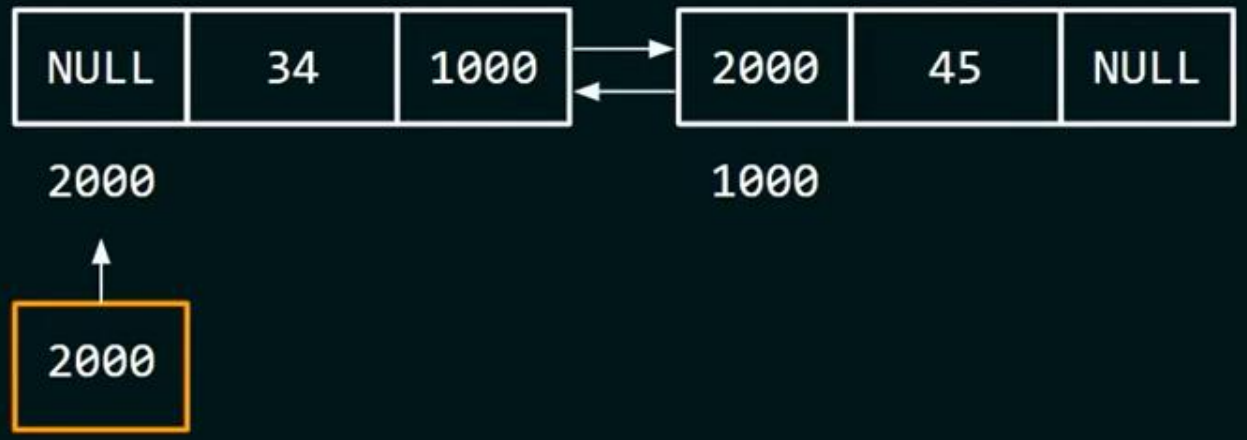
```c
#include <stdio.h>
#include <stdlib.h>

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




```c
int main() {
    struct node* head = NULL;
     struct node* ptr;
     head = addToEmpty(head, 45);
     head = addAtBeg(head, 34);
     head = addAtEnd(head, 9);

     ptr = head;
     while(ptr != NULL)
     {
         printf("%d ", ptr->data);
         ptr = ptr->next;
     }
     return 0;
}
```

Program

```c
#include <stdio.h>
#include <stdlib.h>

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




```c
int main() {
    struct node* head = NULL;
     struct node* ptr;
     head = addToEmpty(head, 45);
     head = addAtBeg(head, 34);
     head = addAtEnd(head, 9);

     ptr = head;
     while(ptr != NULL)
     {
         printf("%d ", ptr->data);
         ptr = ptr->next;
     }
     return 0;
}
```

Program

```
struct node* addAtEnd(struct node* head, int data)
{
    struct node* temp, *tp;
    temp = malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;
    tp = head;
    while(tp->next != NULL)
        tp = tp->next;
    tp->next = temp;
    temp->prev = tp;
    return head;
}
```

head

data

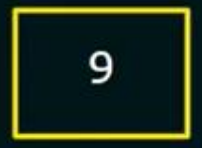
Program

```
struct node* addAtEnd(struct node* head, int data)
{
    struct node* temp, *tp;
    temp = malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;
    tp = head;
    while(tp->next != NULL)
        tp = tp->next;
    tp->next = temp;
    temp->prev = tp;
    return head;
}
```

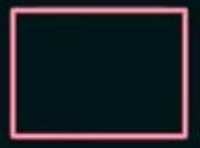
2000 1000

head

3000



data

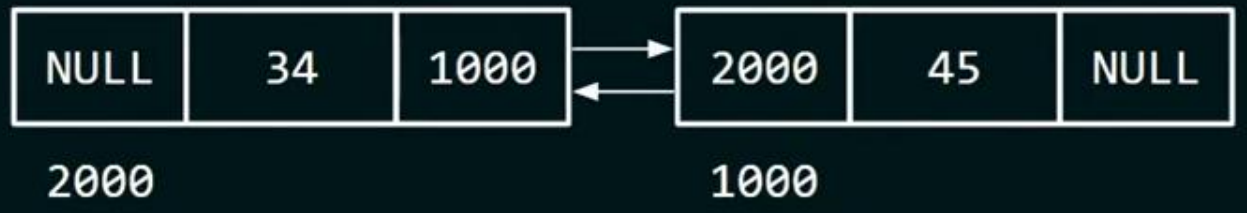


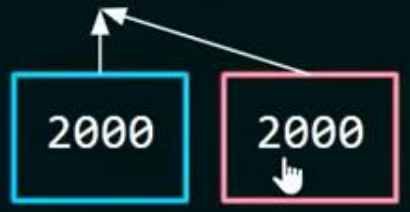
tp

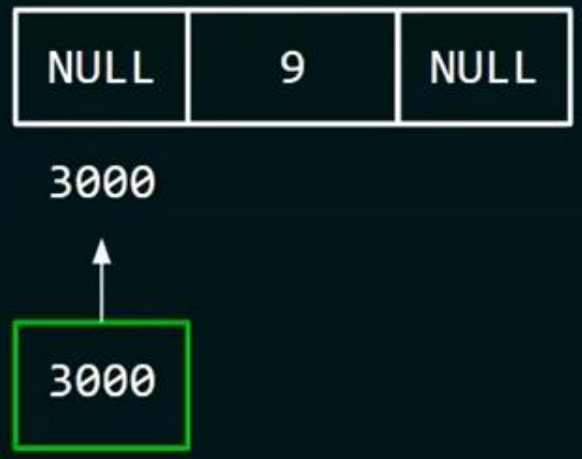
temp

Program

```
struct node* addAtEnd(struct node* head, int data)
{
    struct node* temp, *tp;
    temp = malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;
    tp = head;
    while(tp->next != NULL)
        tp = tp->next;
    tp->next = temp;
    temp->prev = tp;
    return head;
}
```

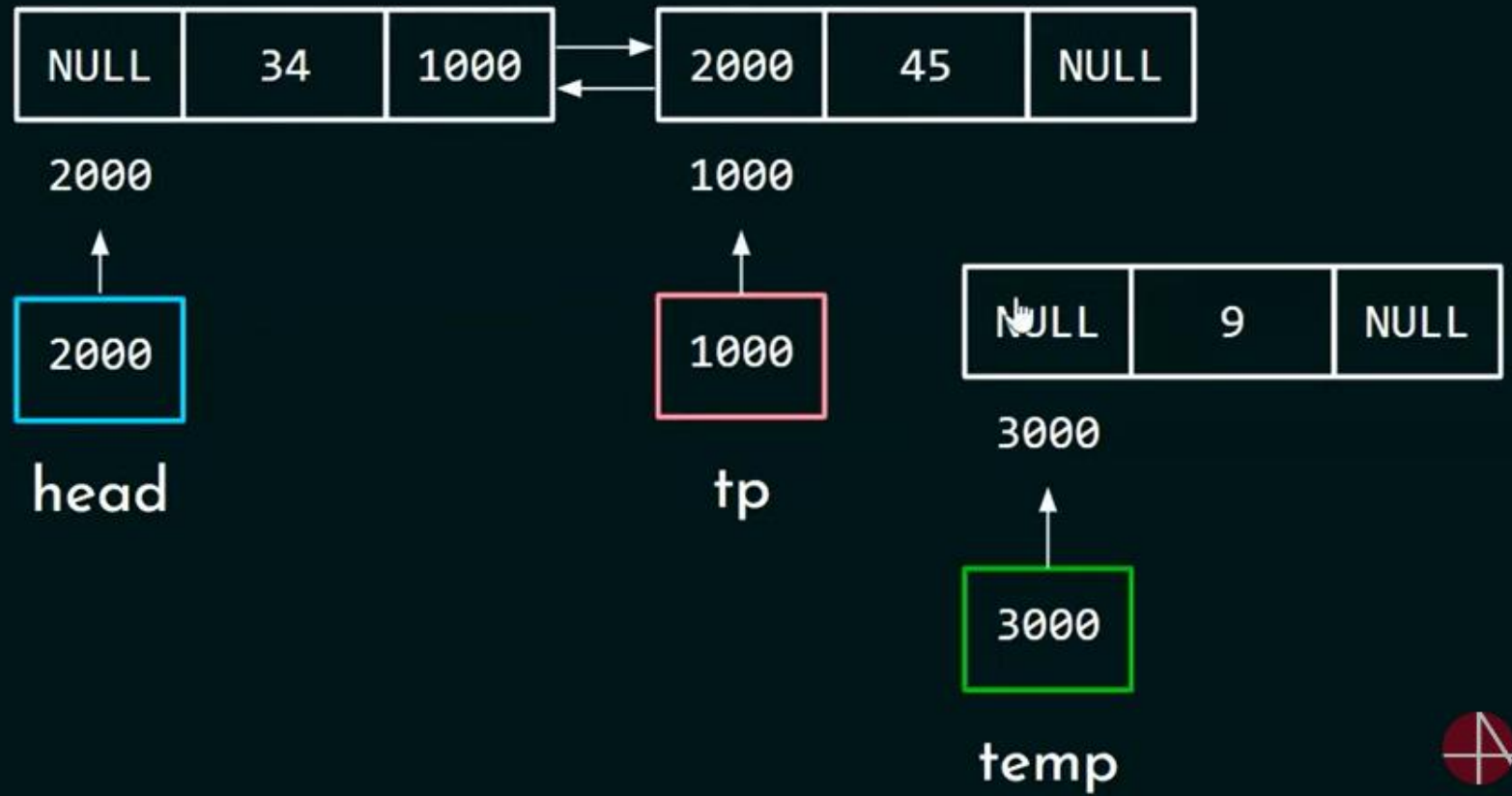






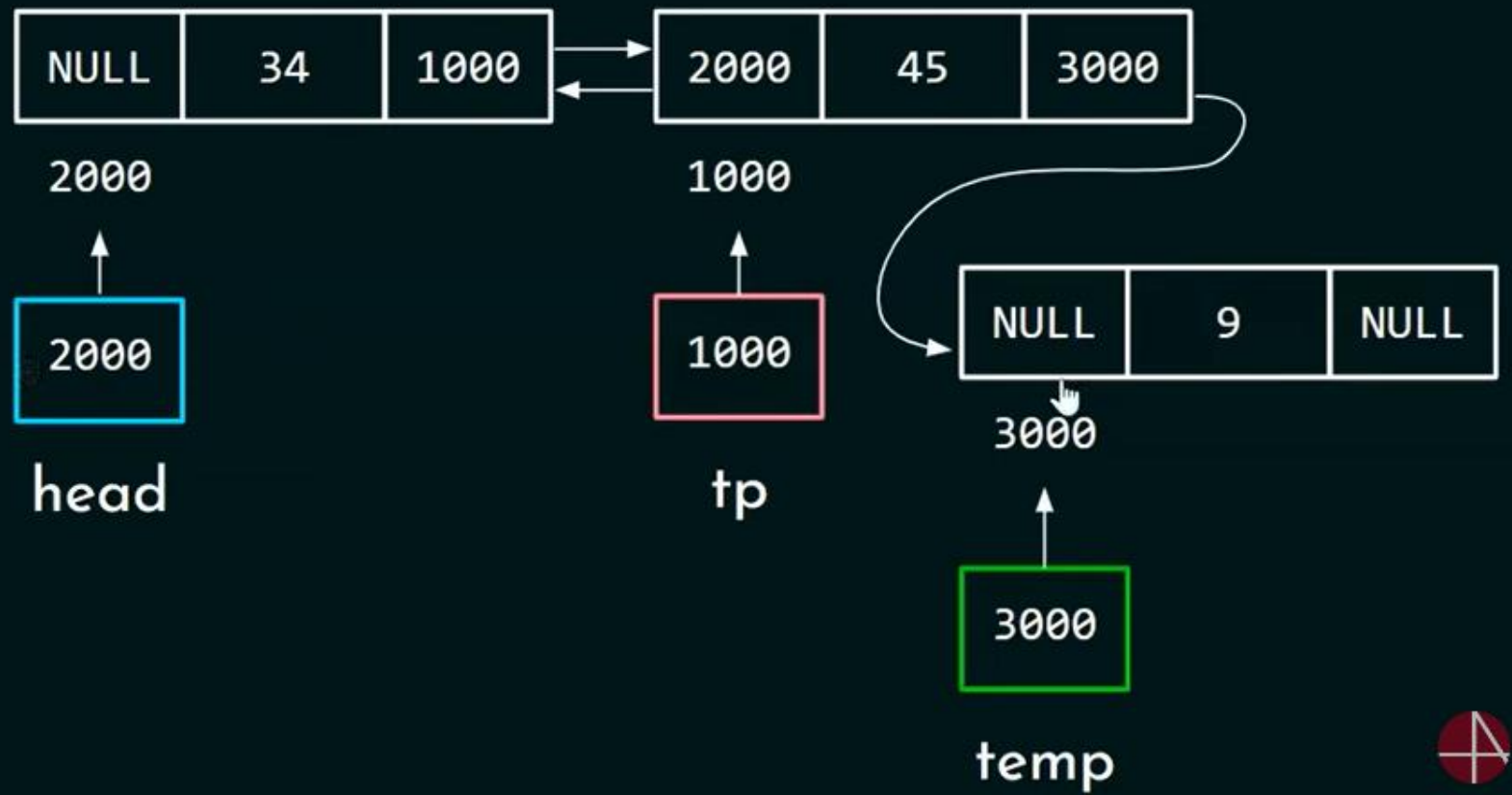
Program

```
struct node* addAtEnd(struct node* head, int data)
{
    struct node* temp, *tp;
    temp = malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;
    tp = head;
    while(tp->next != NULL)
        tp = tp->next;
    tp->next = temp;
    temp->prev = tp;
    return head;
}
```



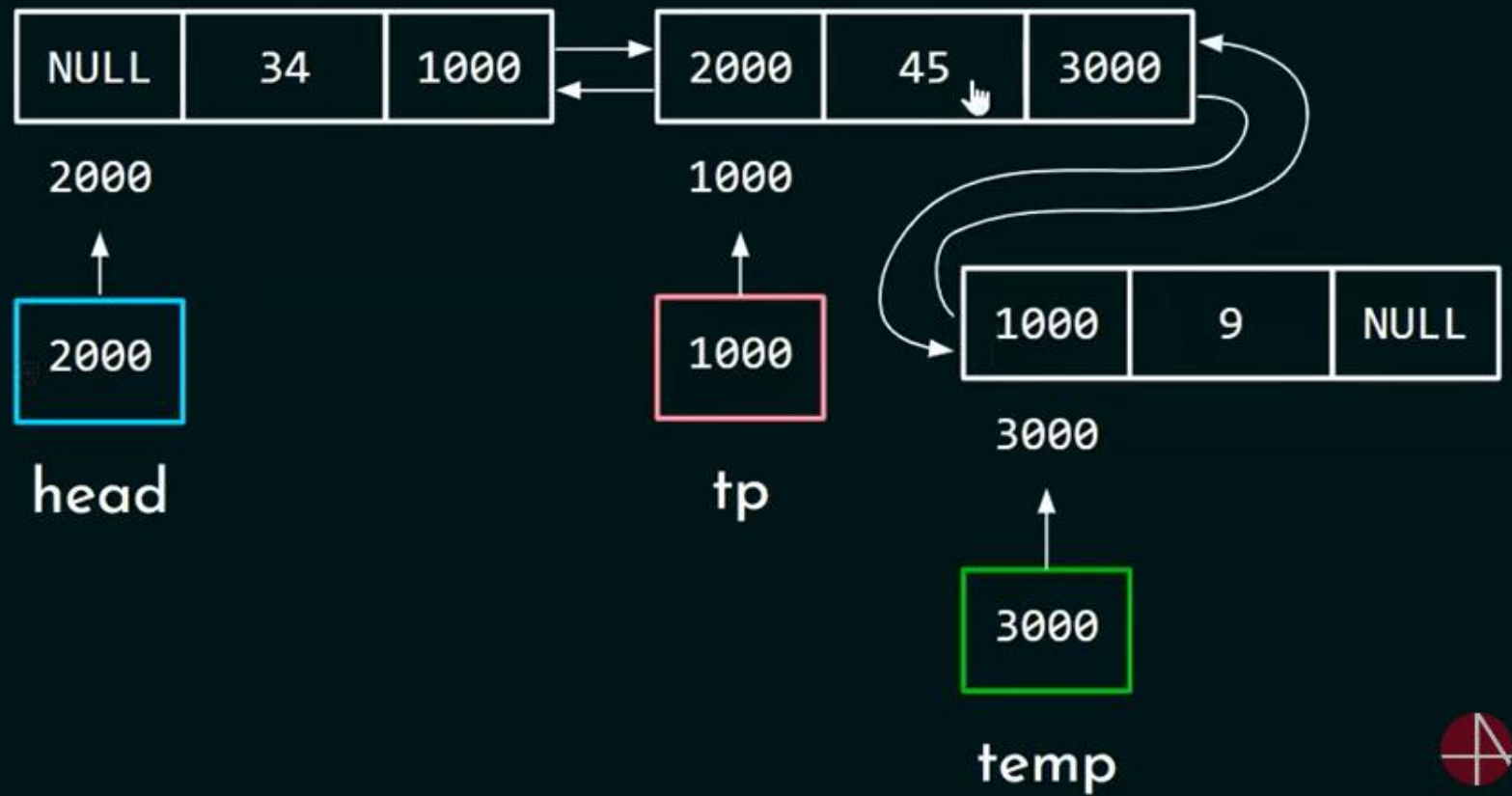
Program

```
struct node* addAtEnd(struct node* head, int data)
{
    struct node* temp, *tp;
    temp = malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;
    tp = head;
    while(tp->next != NULL)
        tp = tp->next;
    tp->next = temp;
    temp->prev = tp;
    return head;
}
```



Program

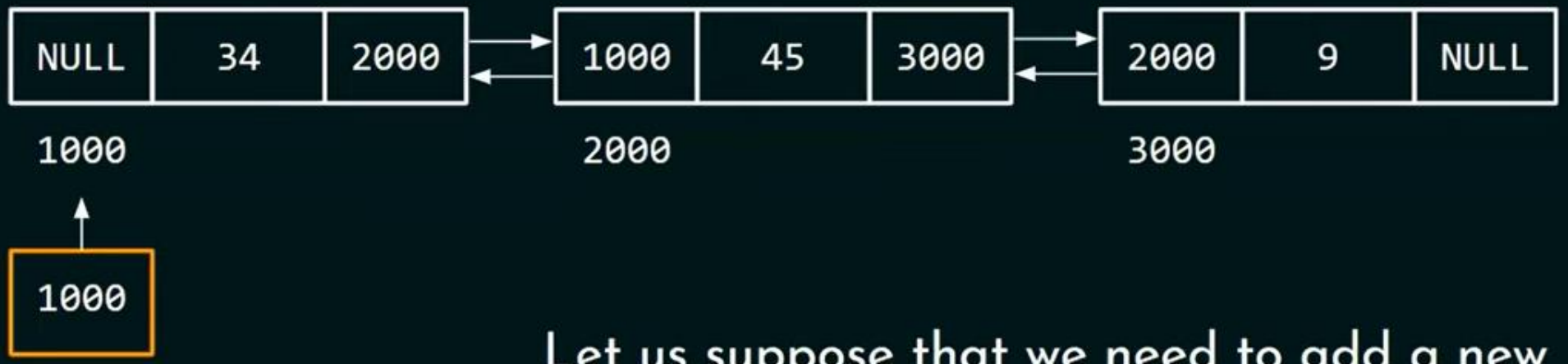
```
struct node* addAtEnd(struct node* head, int data)
{
    struct node* temp, *tp;
    temp = malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;
    tp = head;
    while(tp->next != NULL)
        tp = tp->next;
    tp->next = temp;
    temp->prev = tp;
    return head;
}
```



Programming and Data Structures

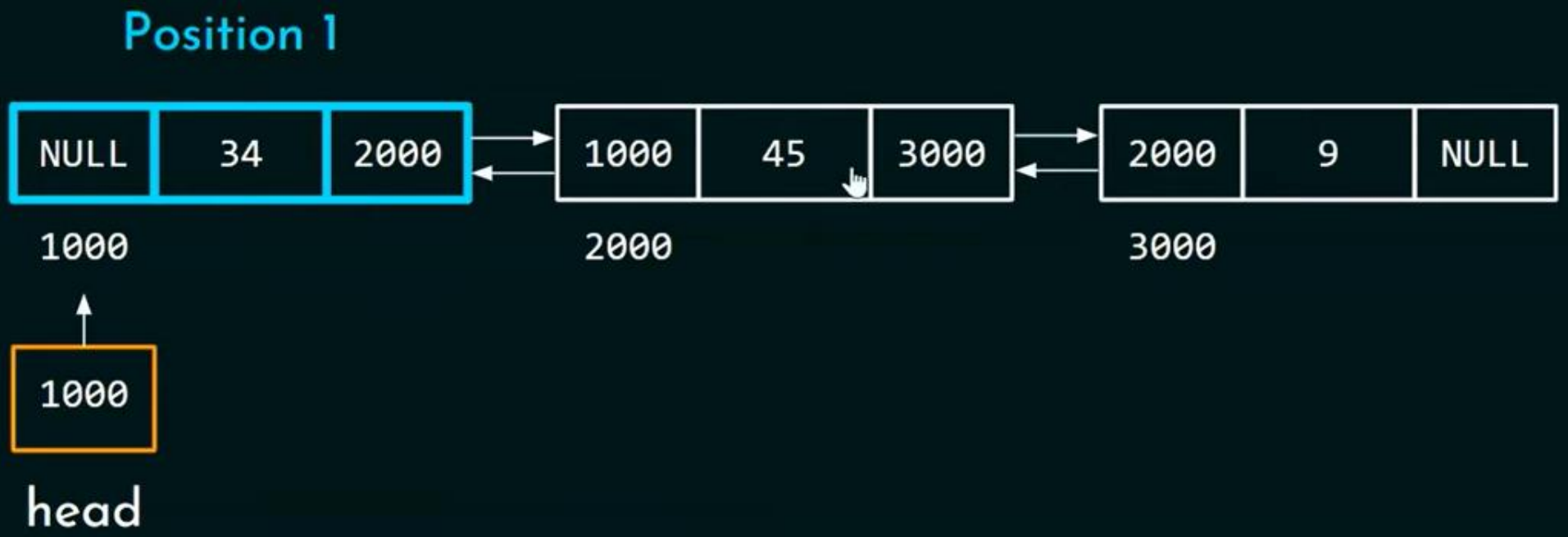
Doubly Linked List: Insertion in between the nodes (Part 1)

Insertion after the given position

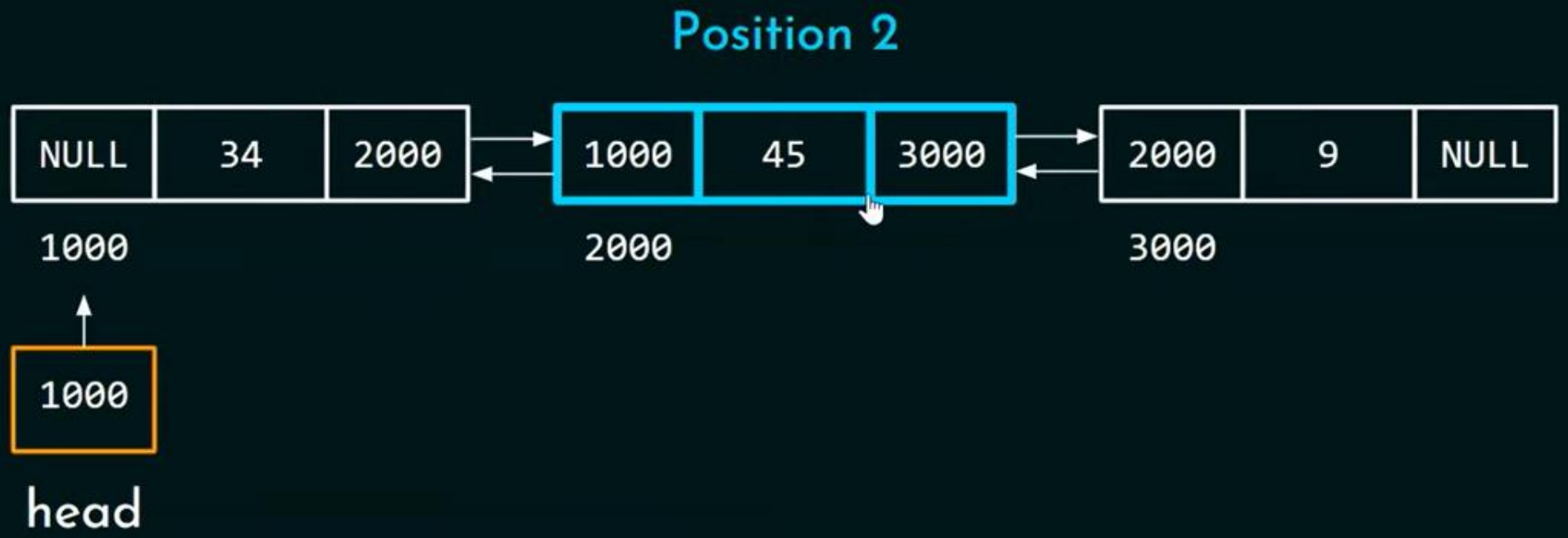


Let us suppose that we need to add a new node after position 2.

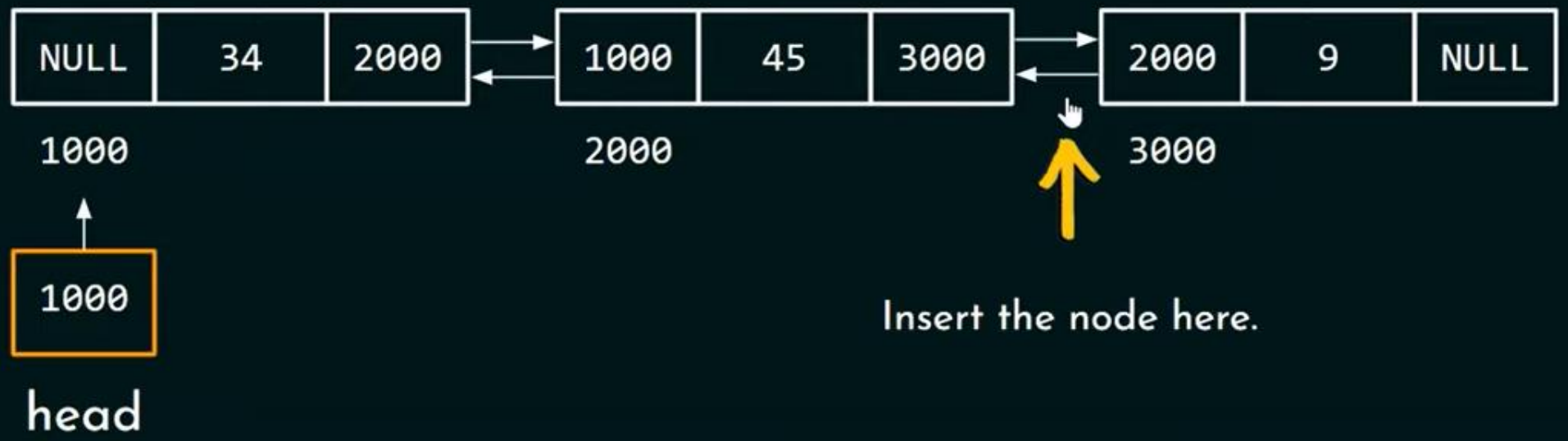
Insertion after the given position



Insertion after the given position

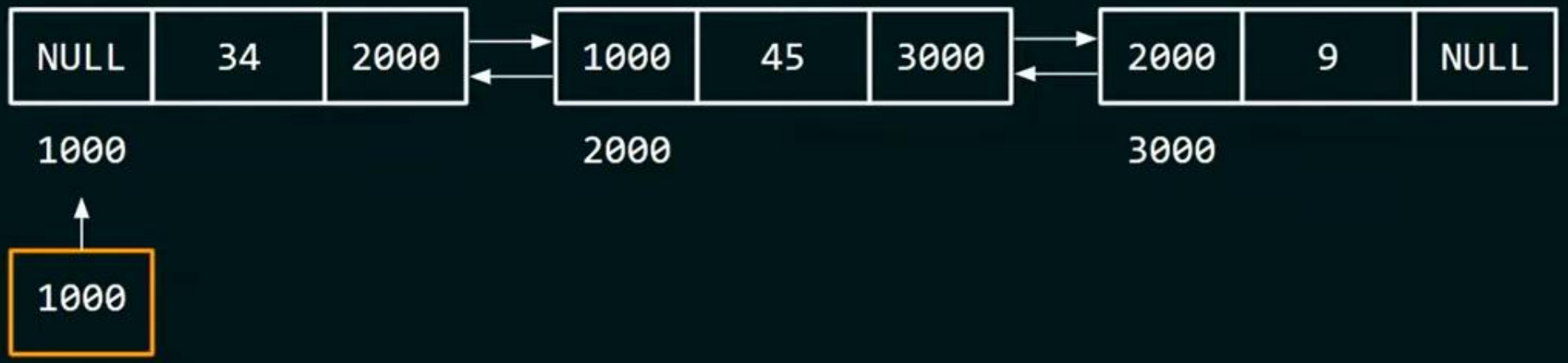


Insertion after the given position

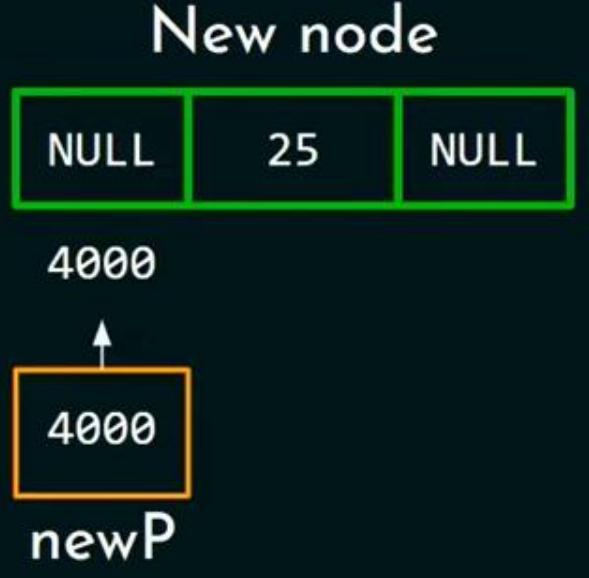


Insertion after the given position

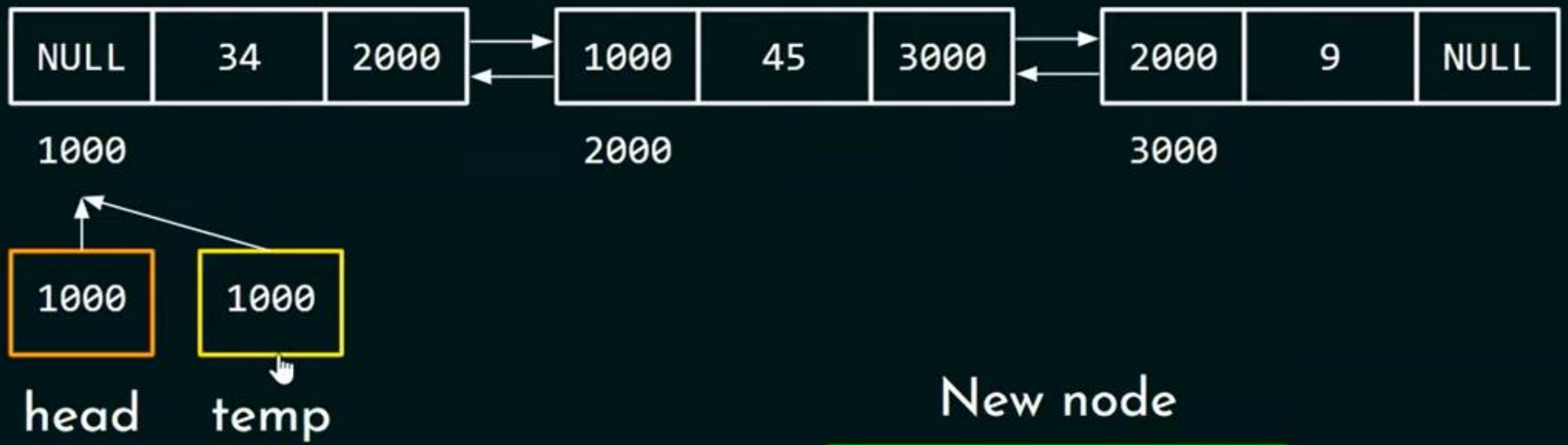
Insertion after the given position



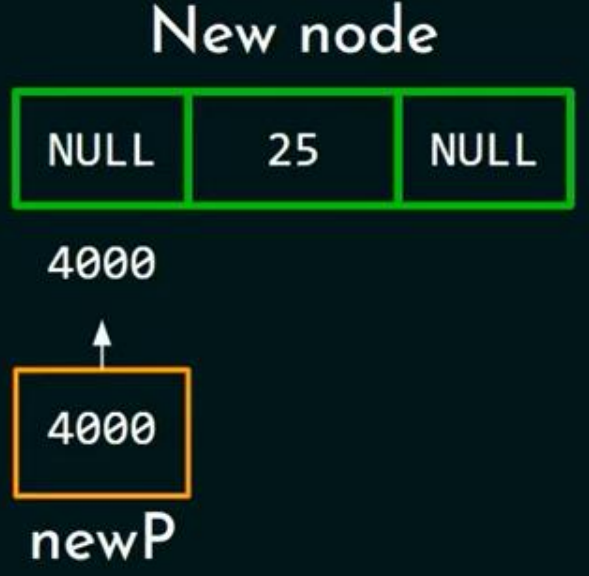
We need a pointer pointing to the second node of the list.



Insertion after the given position

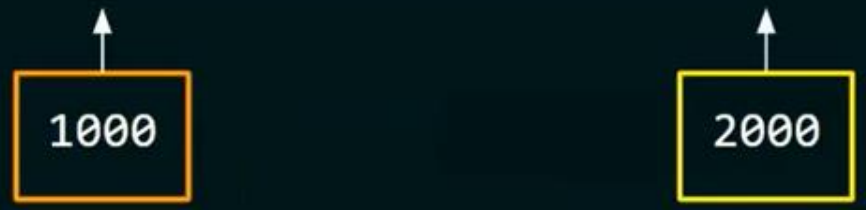


We need a pointer pointing to the second node of the list.



Insertion after the given position

1000 2000 3000



head temp

```
while(position != 1)
{
     temp = temp->next;
     position--;
}
```

New node

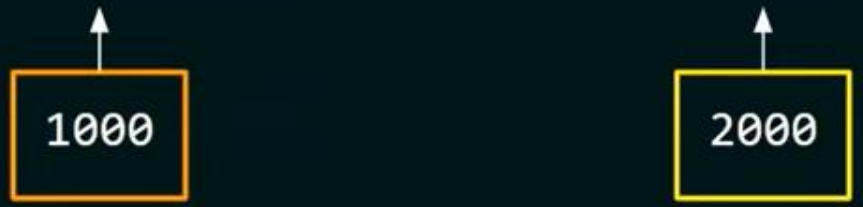
4000

newP

position

Insertion after the given position

1000 2000 3000



head temp

position

```
while(position != 1)
{
     temp = temp->next;
     position--;
}
temp2 = temp->next;
```

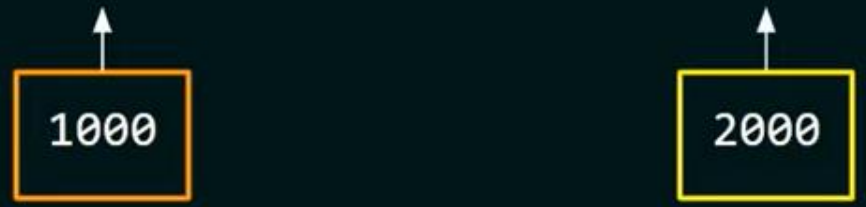
New node

4000

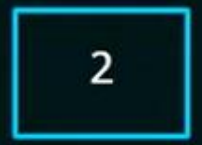
newP

Insertion after the given position

1000 2000 3000



head temp temp2



position

```
while(position != 1)
{
     temp = temp->next;
     position--;
}
temp2 = temp->next;
```

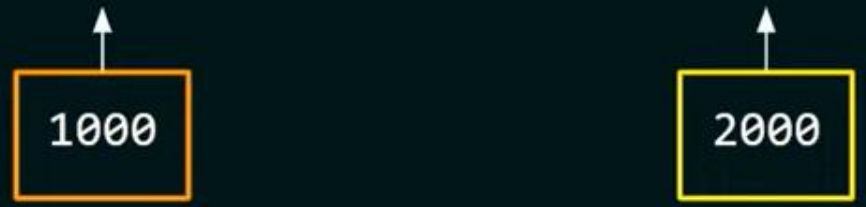
New node

4000

newP

Insertion after the given position

1000 2000 3000



head temp temp2

position

```
temp->next = newP;
```

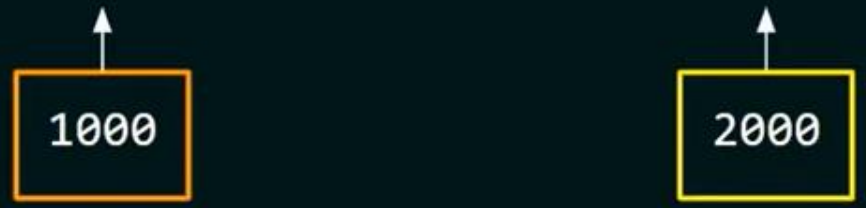
New node

4000

newP

Insertion after the given position

1000 2000 3000



head temp temp2

2

position

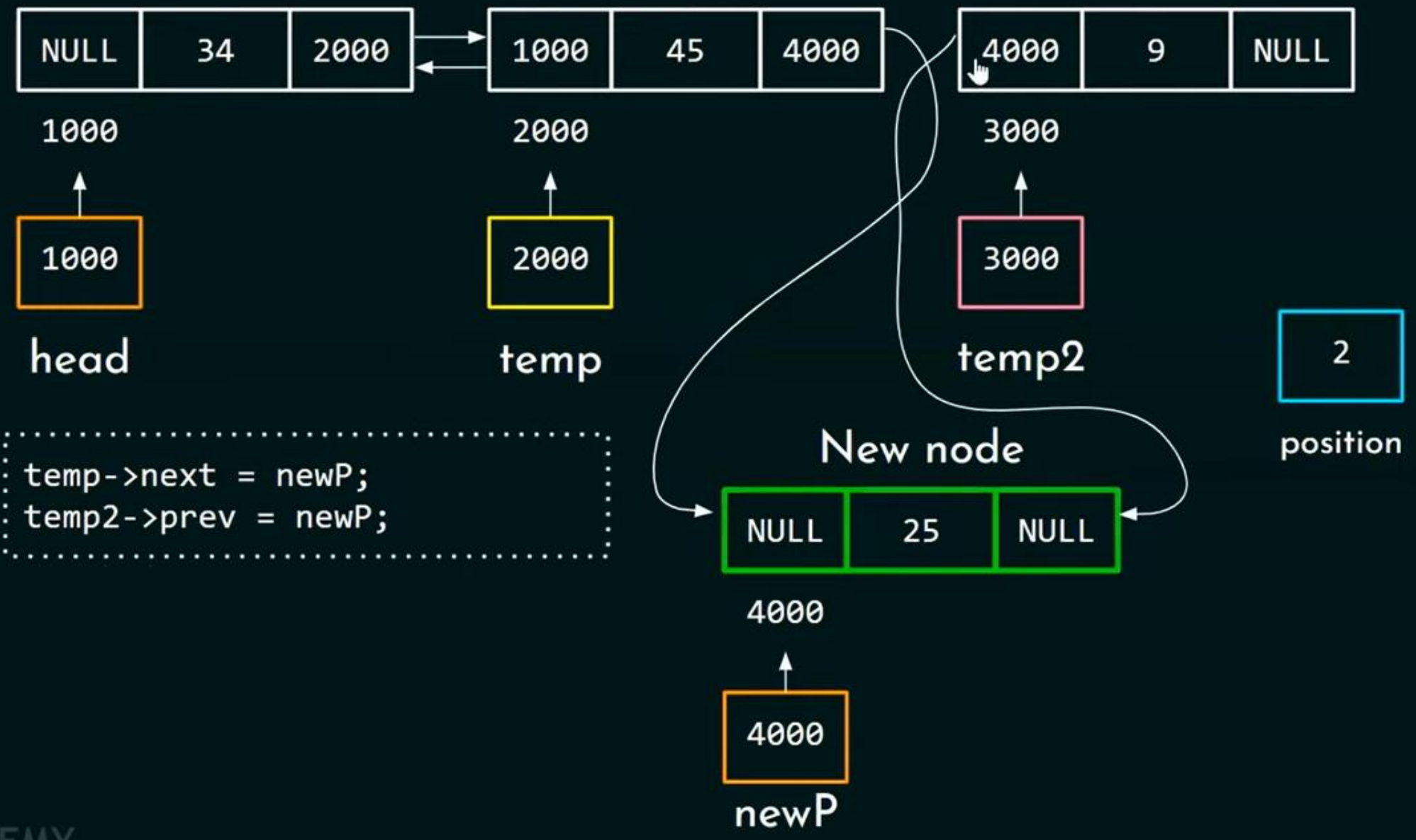
```
temp->next = newP;
```

New node

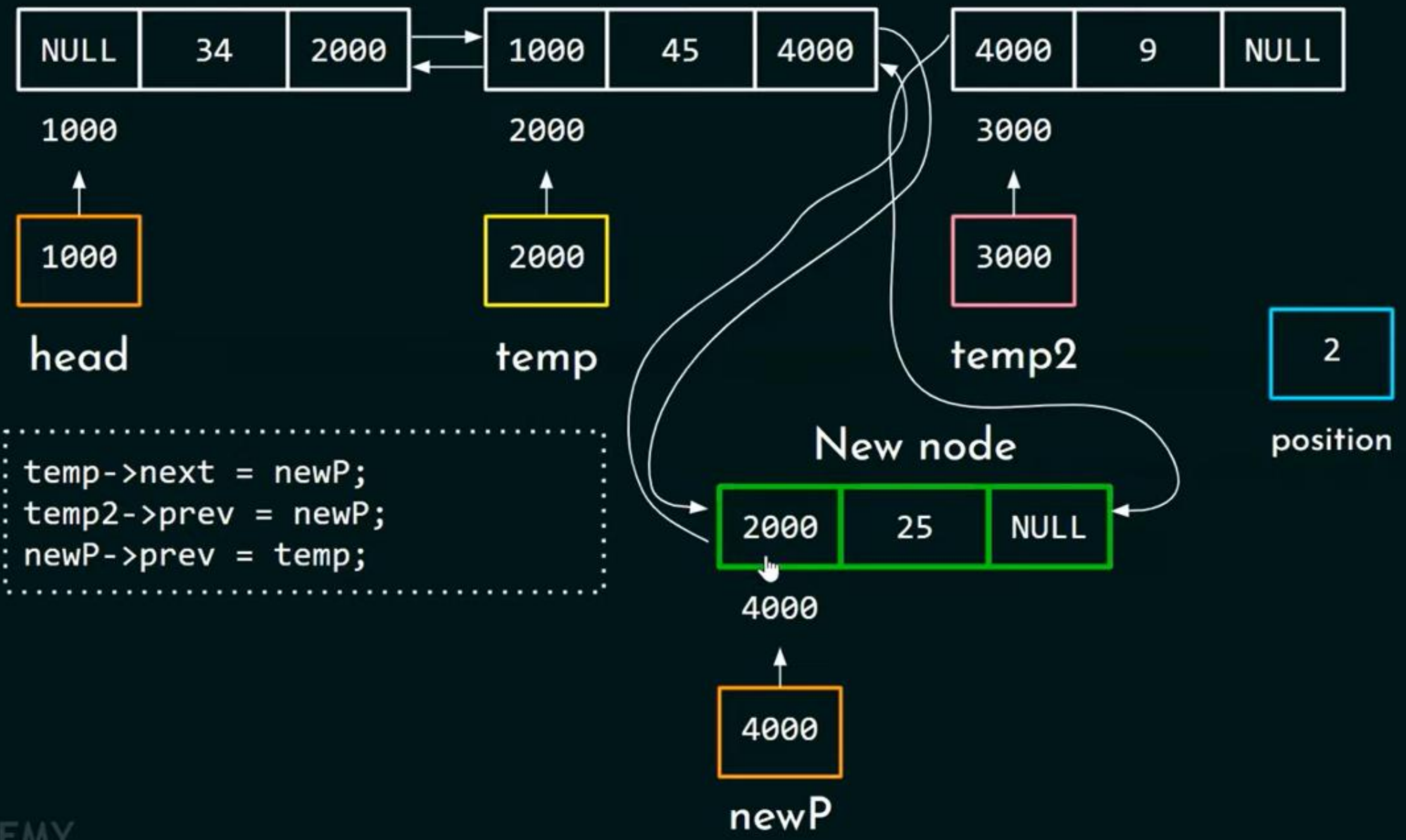
4000

newP

Insertion after the given position

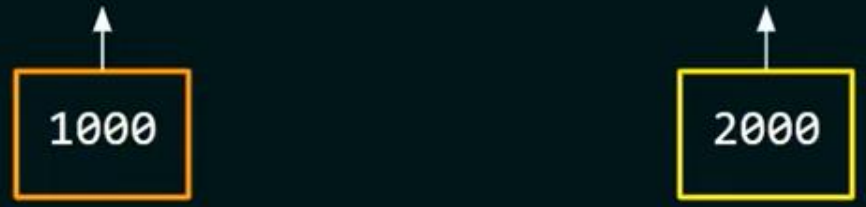


Insertion after the given position



Insertion after the given position

1000 2000 3000



head temp temp2

2

position

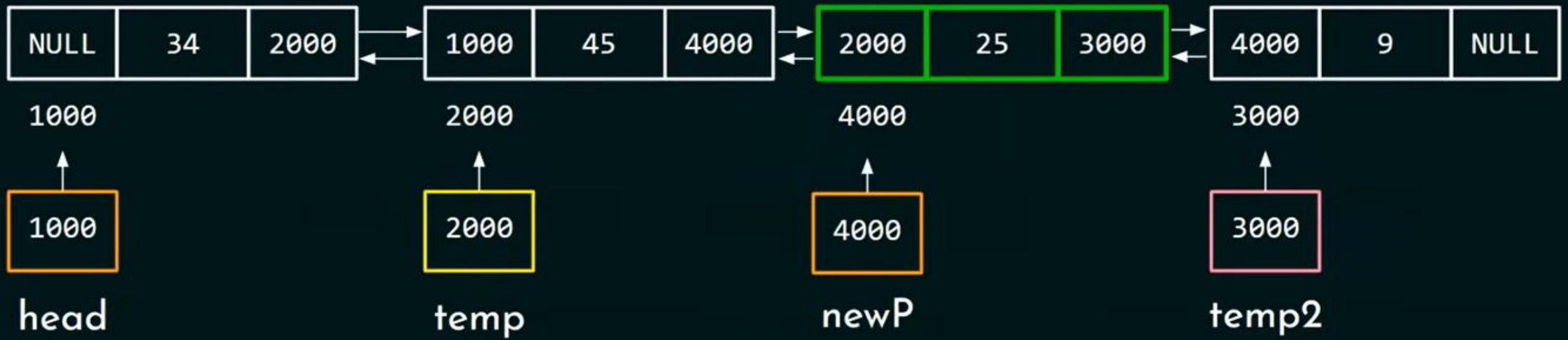
```
temp->next = newP;
temp2->prev = newP;
newP->prev = temp;
newP->next = temp2;
```

New node

4000

newP

Insertion after the given position



addAfterPos Function

```
struct node* addAfterPos(struct node* head, int data, int position)
{
    struct node* newP = NULL;
    struct node* temp = head;
    struct node* temp2 = NULL;
    newP = addToEmpty(newP, data);

    while(position != 1)
    {
        temp = temp->next;
        position--;
    }
    temp2 = temp->next;
    temp->next = newP;
    temp2->prev = newP;
    newP->next = temp2;
    newP->prev = temp;
    return head;
};
```


We also have to consider the case where we need to add a node at the end of the list.

addAfterPos Function

```
struct node* addAfterPos(struct node* head, int data, int position)
{
    struct node* newP = NULL;
    struct node* temp = head;
    struct node* temp2 = NULL;
    newP = addToEmpty(newP, data);

    while(position != 1)
    {
        temp = temp->next;
        position--;
    }
    temp2 = temp->next;
    temp->next = newP;
    temp2->prev = newP;
    newP->next = temp2;
    newP->prev = temp;
    return head;
};
```

After this while loop, we will check the condition

addAfterPos Function

```
struct node* addAfterPos(struct node* head, int data, int position)
{
    struct node* newP = NULL;
    struct node* temp = head;
    struct node* temp2 = NULL;
    newP = addToEmpty(newP, data);

    while(position != 1)
    {
        temp = temp->next;
        position--;
    }
    temp2 = temp->next;
    temp->next = newP;
    temp2->prev = newP;
    newP->next = temp2;
    newP->prev = temp;
    return head;
};
```

```
if (temp->next == NULL)
{
    temp->next = newP;
    newP->prev = temp;
}
```

addAfterPos Function

```
struct node* addAfterPos(struct node* head, int data, int position)
{
    struct node* newP = NULL;
    struct node* temp = head;
    struct node* temp2 = NULL;
    newP = addToEmpty(newP, data);

    while(position != 1)
    {
        temp = temp->next;
        position--;
    }
    temp2 = temp->next;
    temp->next = newP;
    temp2->prev = newP;
    newP->next = temp2;
    newP->prev = temp;
    return head;
};
```

```
if (temp->next == NULL)
{
    temp->next = newP;
    newP->prev = temp;
}
else
{
    This peace of code will
    execute.
}
```

Start here | doubly-addAfterPos.c

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    struct node* prev;
    int data;
    struct node* next;
};

struct node* addToEmpty(struct node* head, int data)
{
    struct node* temp = malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;
    head = temp;
    return head;
}

struct node* addAtBeg(struct node* head, int data)
{
    struct node* temp = malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;
    temp->next = head;
    head->prev = temp;
    head = temp;
    return head;
}

struct node* addAtEnd(struct node* head, int data)
{
```

Start here | doubly-addAfterPos.c

```
28      head = temp;
29      return head;
30  }
31
32  struct node* addAtEnd(struct node* head, int data)
33  {
34      struct node* temp, *tp;
35      temp = malloc(sizeof(struct node));
36      temp->prev = NULL;
37      temp->data = data;
38      temp->next = NULL;
39      tp = head;
40      while(tp->next != NULL)
41          tp = tp->next;
42      tp->next = temp;
43      temp->prev = tp;
44      return head;
45  }
46
47  struct node* addAfterPos(struct node* head, int data, int position)
48  {
49      struct node* newP = NULL;
50      struct node* temp = head;
51      struct node* temp2 = NULL;
52      newP = addToEmpty(newP, data);
53
54      while(position != 1)
55      {
56          temp = temp->next;
57          position--;
58      }
59
60      if(temp->next == NULL) //last node
```

Start here | doubly-addAfterPos.c

```
61      {
62          temp->next = newP;
63          newP->prev = temp;
64      }
65      else {
66          temp2 = temp->next;
67          temp->next = newP;
68          temp2->prev = newP;
69          newP->next = temp2;
70          newP->prev = temp;
71      }
72      return head;
73  };
74
75  int main() {
76      struct node* head = NULL;
77      struct node* ptr;
78      int position = 2;
79      head = addToEmpty(head, 45);
80      head = addAtBeg(head, 34);
81      head = addAtEnd(head, 9);
82      head = addAfterPos(head, 25, position);
83
84      ptr = head;
85      while(ptr != NULL)
86      {
87          printf("%d ", ptr->data);
88          ptr = ptr->next;
89      }
90      return 0;
91  }
92
93
```