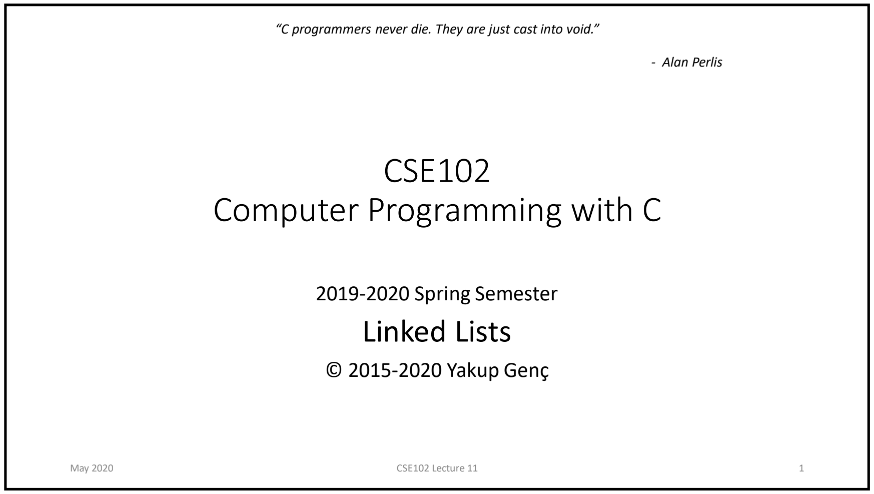


"C programmers never die. They are just cast into void."

- Alan Perlis

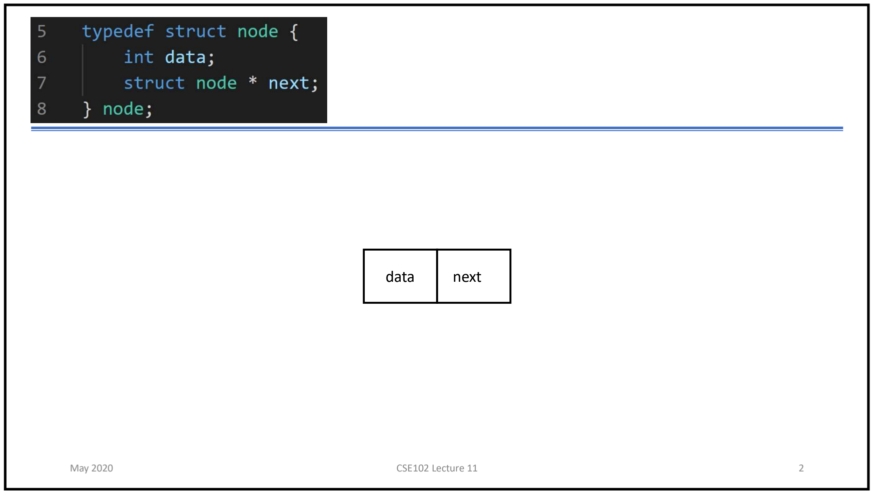
CSE102
Computer Programming with C

2019-2020 Spring Semester

Linked Lists

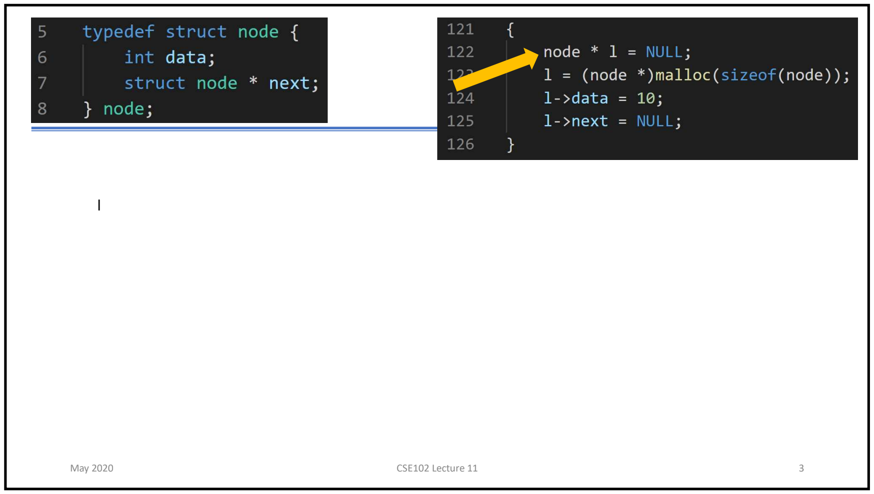
© 2015-2020 Yakup Genç

1



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

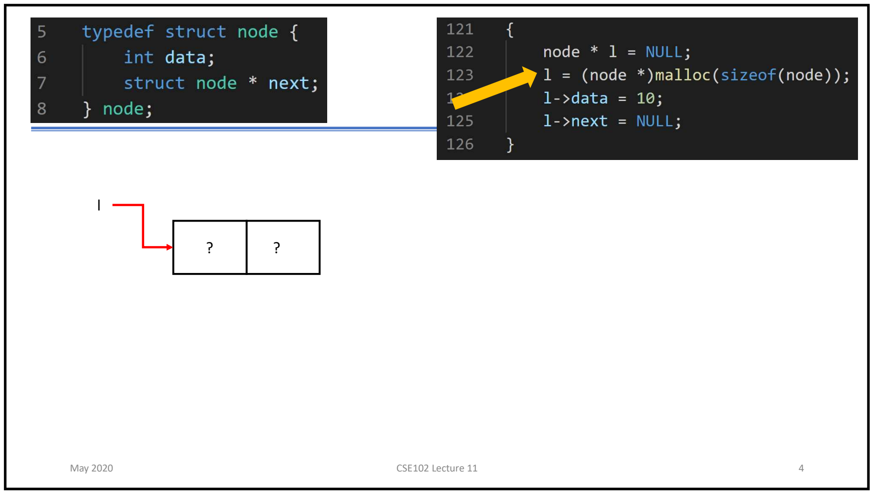
2



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

```c
{
    node * l = NULL;
    l = (node *)malloc(sizeof(node));
    l->data = 10;
    l->next = NULL;
}
```

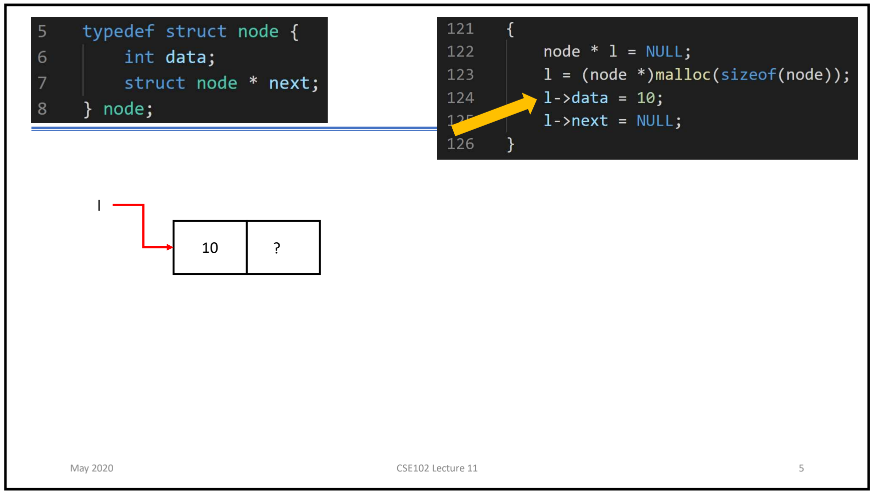
3



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

```c
{
    node * l = NULL;
    l = (node *)malloc(sizeof(node));
    l->data = 10;
    l->next = NULL;
}
```

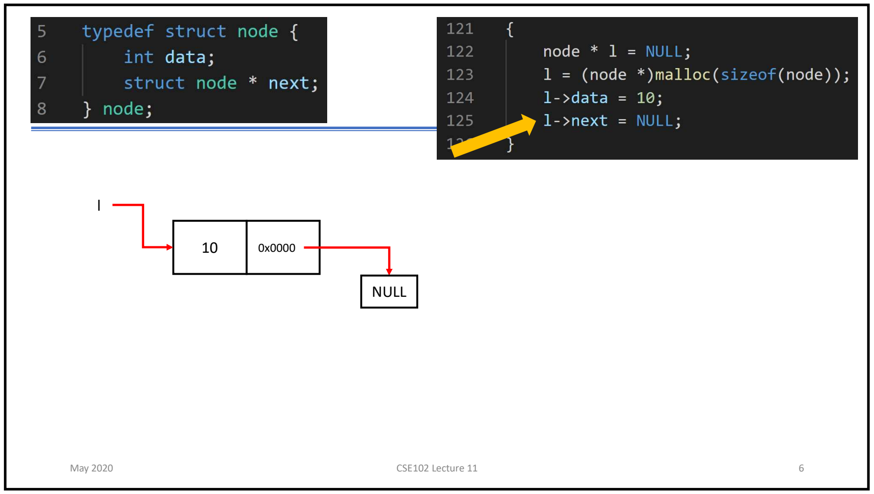
4



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

```
{
    node * l = NULL;
    l = (node *)malloc(sizeof(node));
    l->data = 10;
    l->next = NULL;
}
```

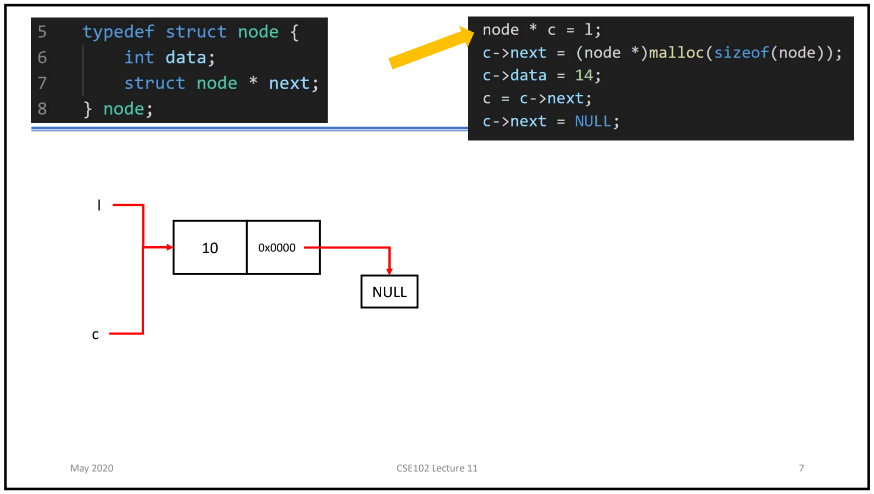
5



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

```
{
    node * l = NULL;
    l = (node *)malloc(sizeof(node));
    l->data = 10;
    l->next = NULL;
}
```

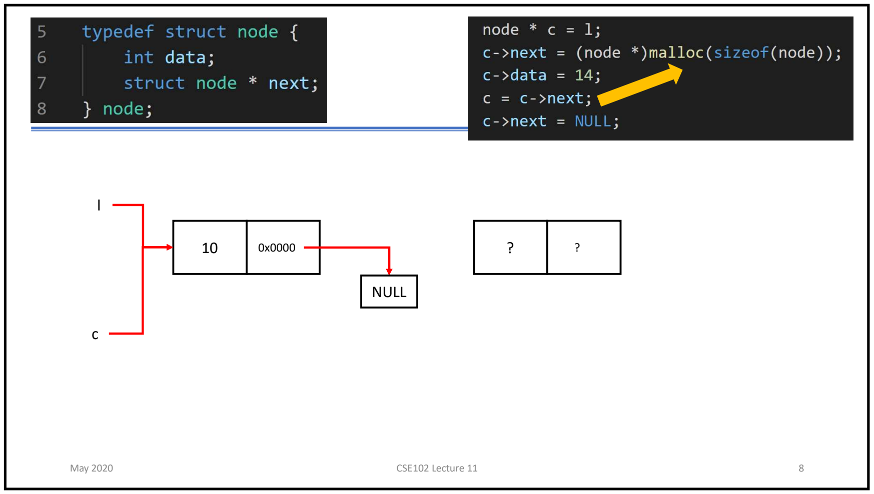
6



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

```
node * c = l;
c->next = (node *)malloc(sizeof(node));
c->data = 14;
c = c->next;
c->next = NULL;
```

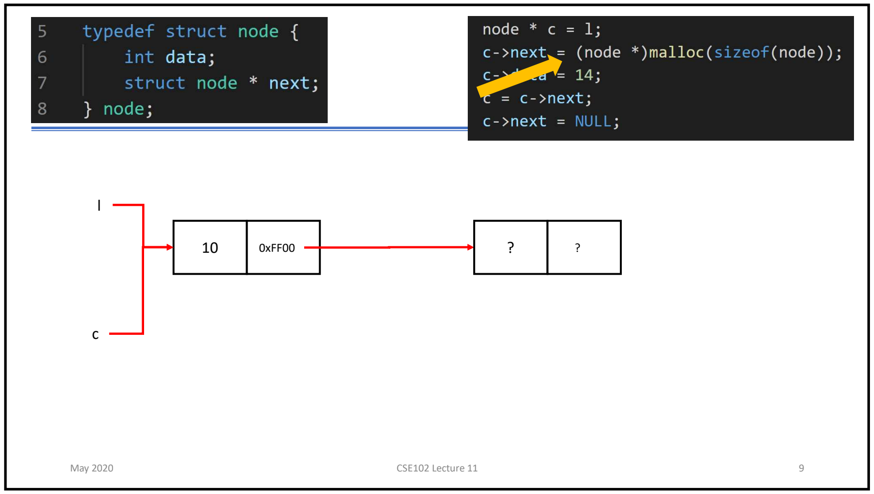
7



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

```
node * c = l;
c->next = (node *)malloc(sizeof(node));
c->data = 14;
c = c->next;
c->next = NULL;
```

8

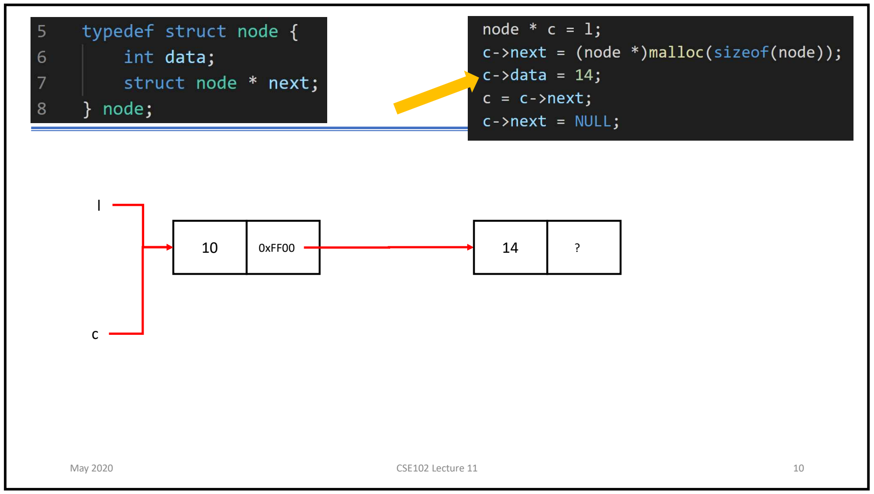


```
typedef struct node {
    int data;
    struct node * next;
} node;
```

```
node * c = l;
c->next = (node *)malloc(sizeof(node));
c->data = 14;
c = c->next;
c->next = NULL;
```

May 2020 CSE102 Lecture 11 9

9

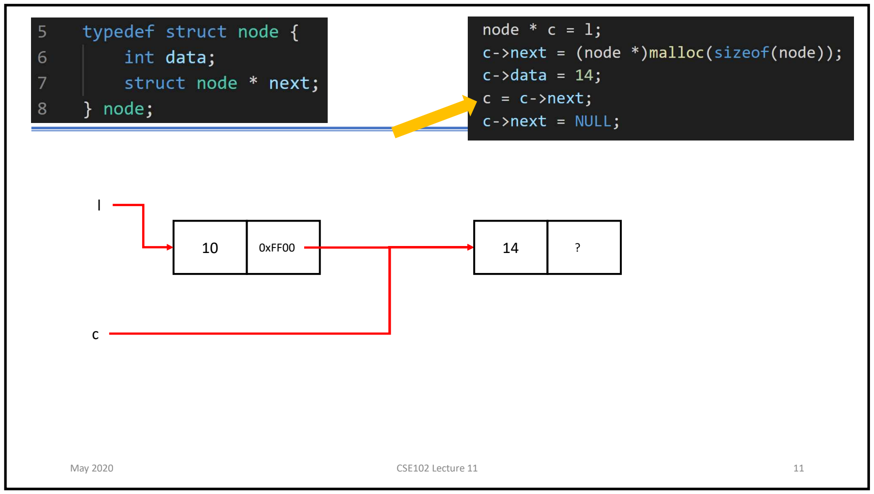


```
typedef struct node {
    int data;
    struct node * next;
} node;
```

```
node * c = l;
c->next = (node *)malloc(sizeof(node));
c->data = 14;
c = c->next;
c->next = NULL;
```

May 2020 CSE102 Lecture 11 10

10

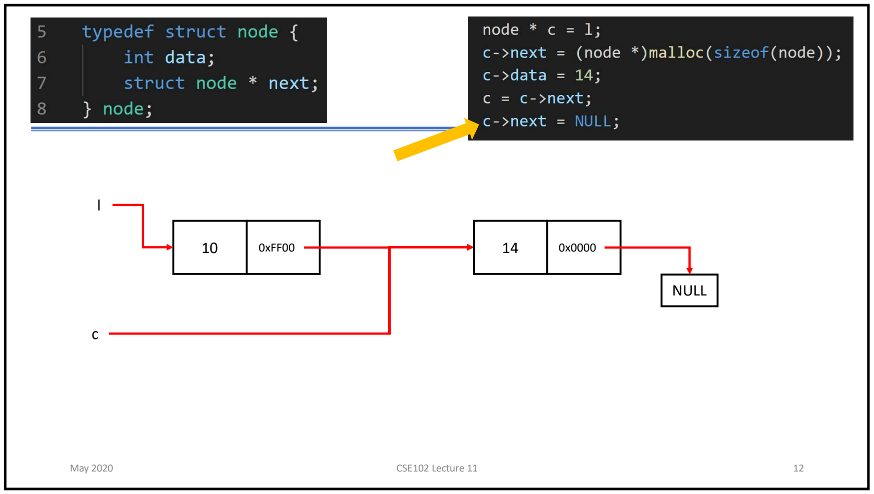


```
typedef struct node {
    int data;
    struct node * next;
} node;
```

```
node * c = l;
c->next = (node *)malloc(sizeof(node));
c->data = 14;
c = c->next;
c->next = NULL;
```

May 2020 CSE102 Lecture 11 11

11

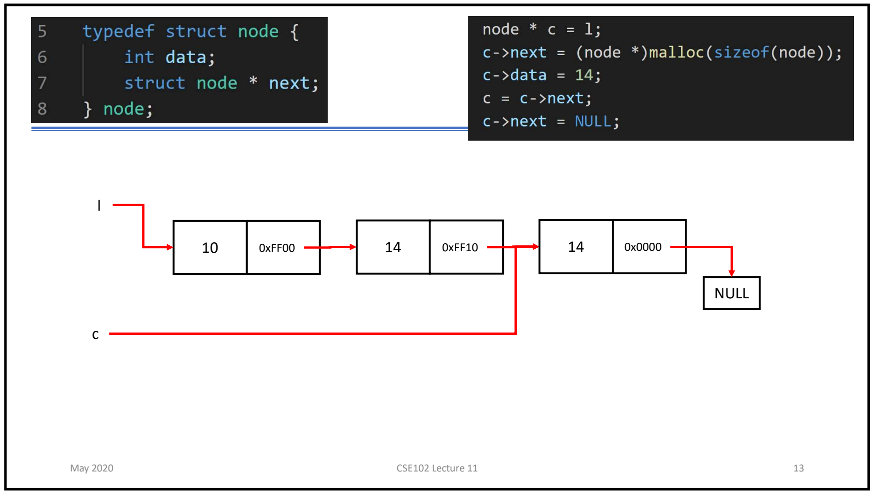


```
typedef struct node {
    int data;
    struct node * next;
} node;
```

```
node * c = l;
c->next = (node *)malloc(sizeof(node));
c->data = 14;
c = c->next;
c->next = NULL;
```

May 2020 CSE102 Lecture 11 12

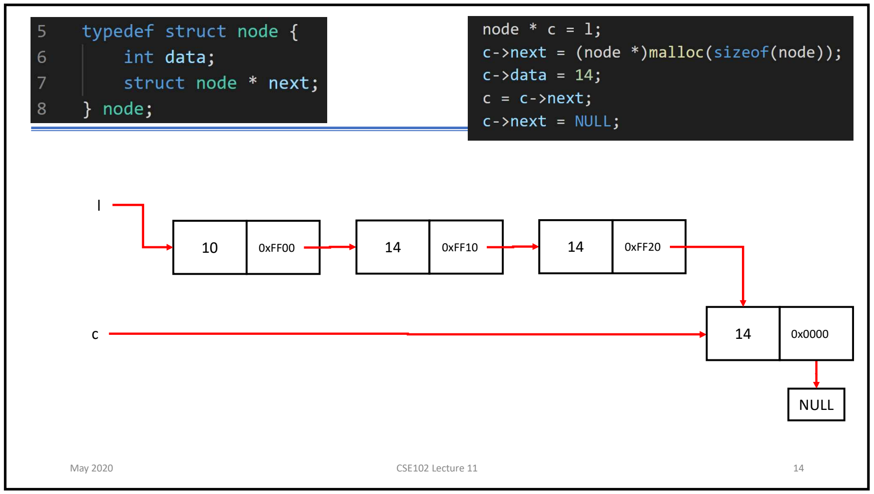
12



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

```
node * c = l;
c->next = (node *)malloc(sizeof(node));
c->data = 14;
c = c->next;
c->next = NULL;
```

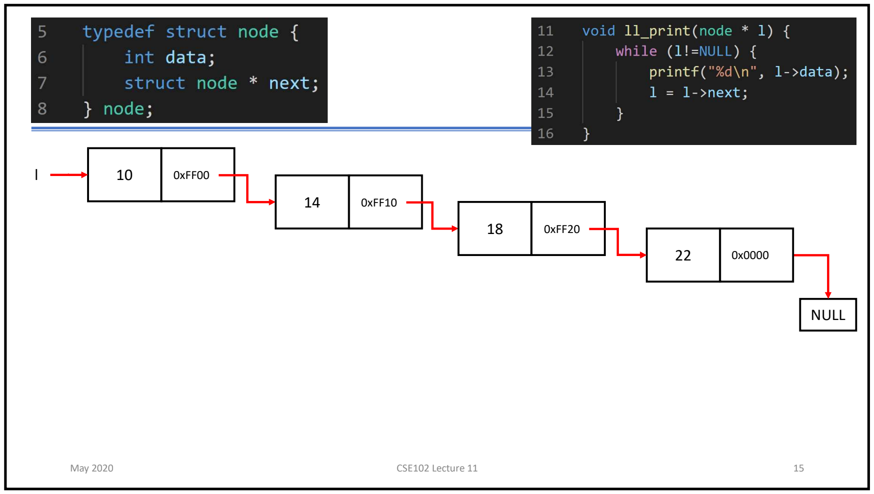
13



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

```
node * c = l;
c->next = (node *)malloc(sizeof(node));
c->data = 14;
c = c->next;
c->next = NULL;
```

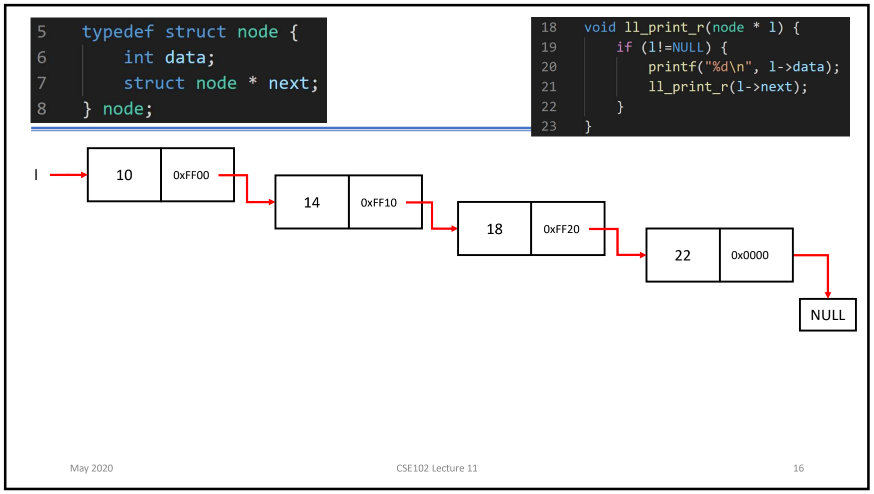
14



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

```
void ll_print(node * l) {
    while (l!=NULL) {
        printf("%d\n", l->data);
        l = l->next;
    }
}
```

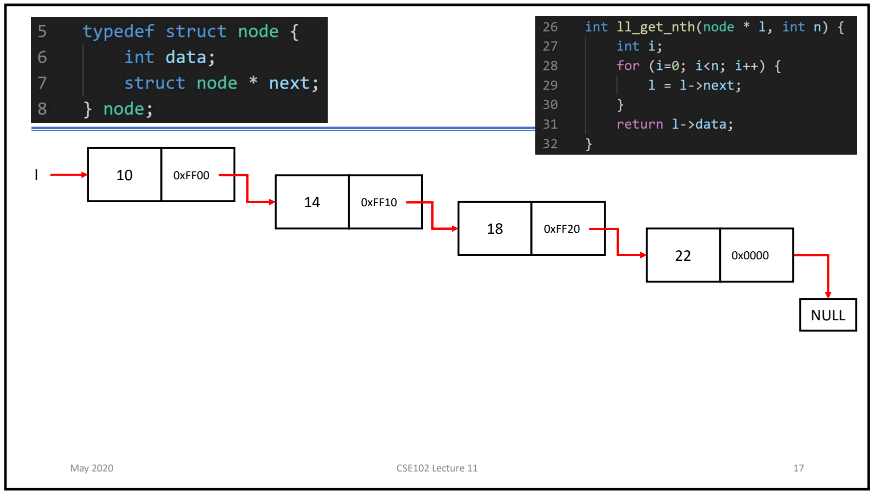
15



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

```
void ll_print_r(node * l) {
    if (l!=NULL) {
        printf("%d\n", l->data);
        ll_print_r(l->next);
    }
}
```

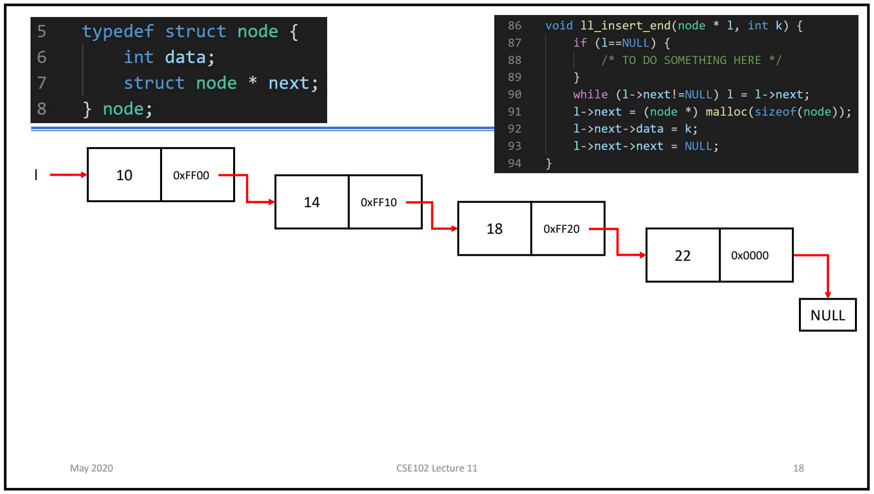
16



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

```c
int ll_get_nth(node * l, int n) {
    int i;
    for (i=0; i<n; i++) {
        l = l->next;
    }
    return l->data;
}
```

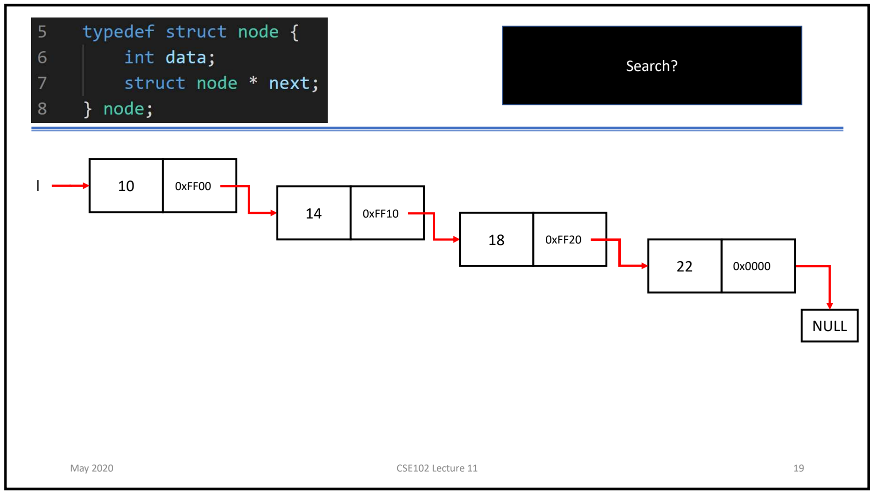
17



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

```c
void ll_insert_end(node * l, int k) {
    if (l==NULL) {
        /* TO DO SOMETHING HERE */
    }
    while (l->next!=NULL) l = l->next;
    l->next = (node *) malloc(sizeof(node));
    l->next->data = k;
    l->next->next = NULL;
}
```

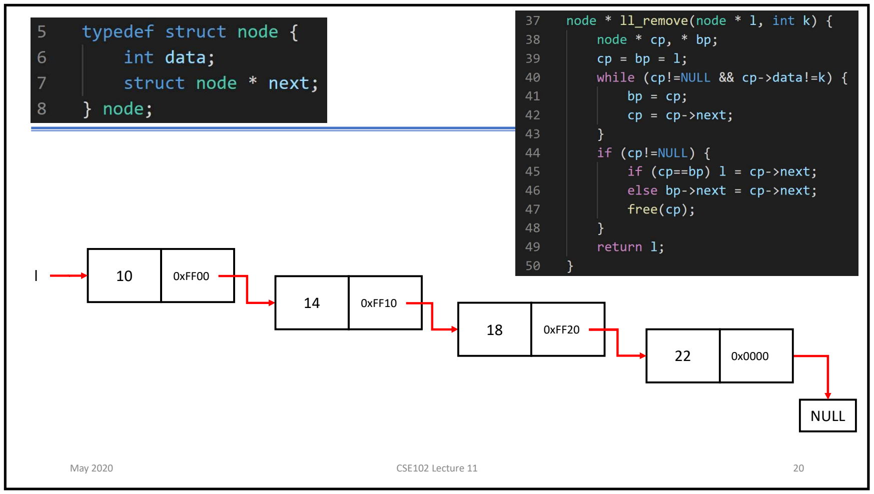
18



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

Search?

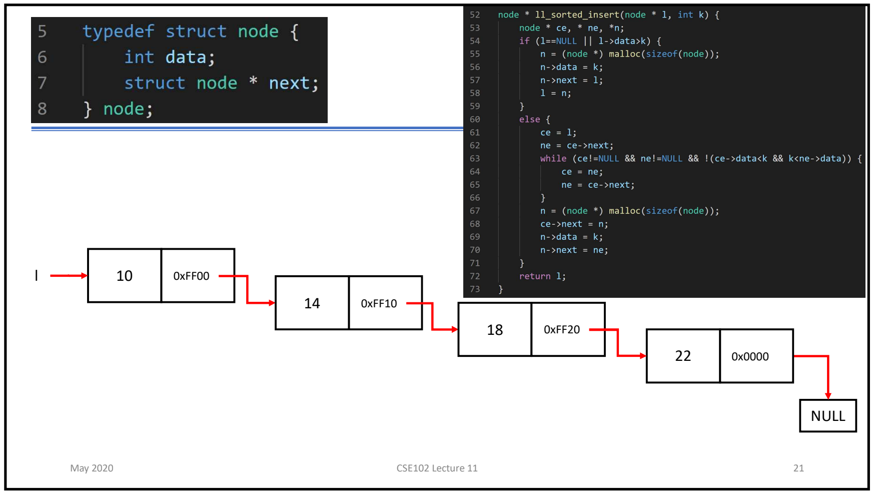
19



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

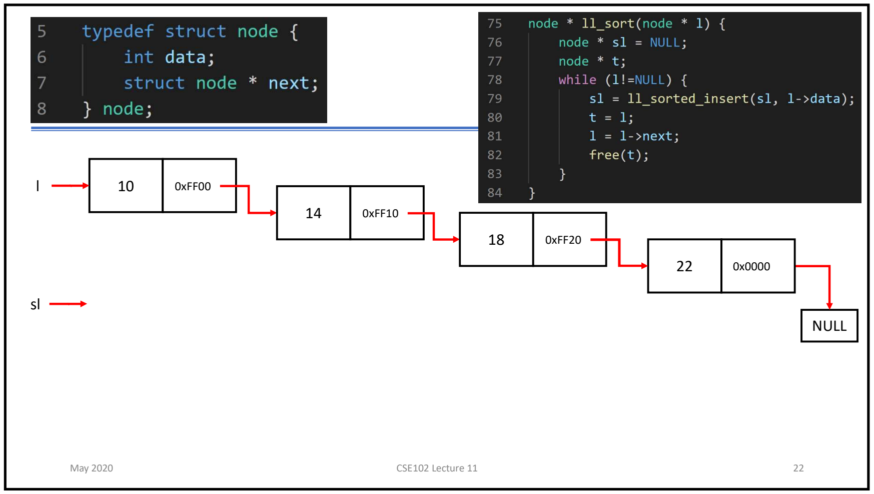
```c
node * ll_remove(node * l, int k) {
    node * cp, * bp;
    cp = bp = l;
    while (cp!=NULL && cp->data!=k) {
        bp = cp;
        cp = cp->next;
    }
    if (cp!=NULL) {
        if (cp==bp) l = cp->next;
        else bp->next = cp->next;
        free(cp);
    }
    return l;
}
```

20



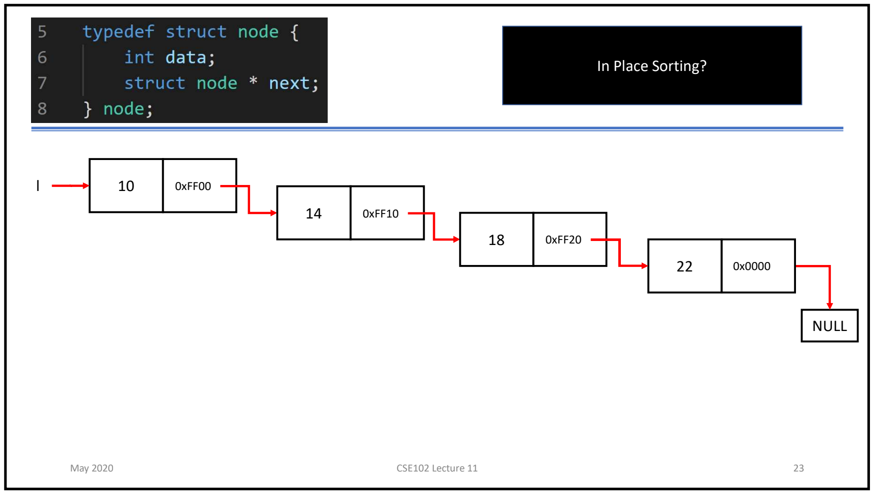
```
typedef struct node {
    int data;
    struct node * next;
} node;
```

```
node * ll_sorted_insert(node * l, int k) {
    node * ce, * ne, *n;
    if (l==NULL || l->data>k) {
        n = (node *) malloc(sizeof(node));
        n->data = k;
        n->next = l;
        l = n;
    }
    else {
        ce = l;
        ne = ce->next;
        while (ce!=NULL && ne!=NULL && !(ce->data<k && k<ne->data)) {
            ce = ne;
            ne = ce->next;
        }
        n = (node *) malloc(sizeof(node));
        ce->next = n;
        n->data = k;
        n->next = ne;
    }
    return l;
}
```



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

```
node * ll_sort(node * l) {
    node * sl = NULL;
    node * t;
    while (l!=NULL) {
        sl = ll_sorted_insert(sl, l->data);
        t = l;
        l = l->next;
        free(t);
    }
}
```



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

In Place Sorting?