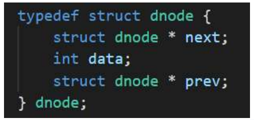
"Code never lies, comments sometimes do."

- Anonymous

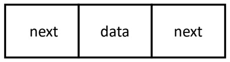
CSE102
Computer Programming with C

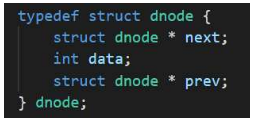
2019-2020 Spring Semester

Doubly Linked Lists

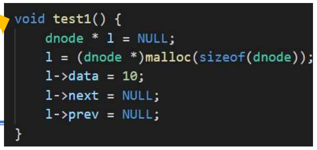


```c
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

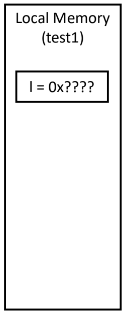


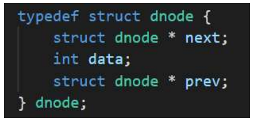


```c
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

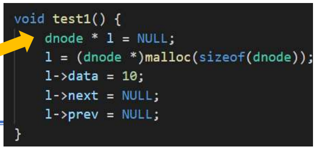


```c
void test1() {
    dnode * l = NULL;
    l = (dnode *)malloc(sizeof(dnode));
    l->data = 10;
    l->next = NULL;
    l->prev = NULL;
}
```

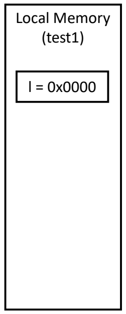


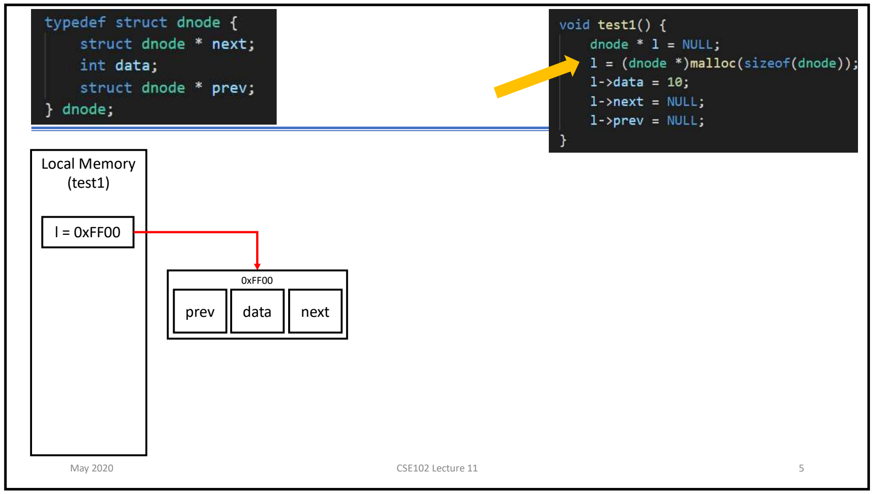


```c
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```



```c
void test1() {
    dnode * l = NULL;
    l = (dnode *)malloc(sizeof(dnode));
    l->data = 10;
    l->next = NULL;
    l->prev = NULL;
}
```



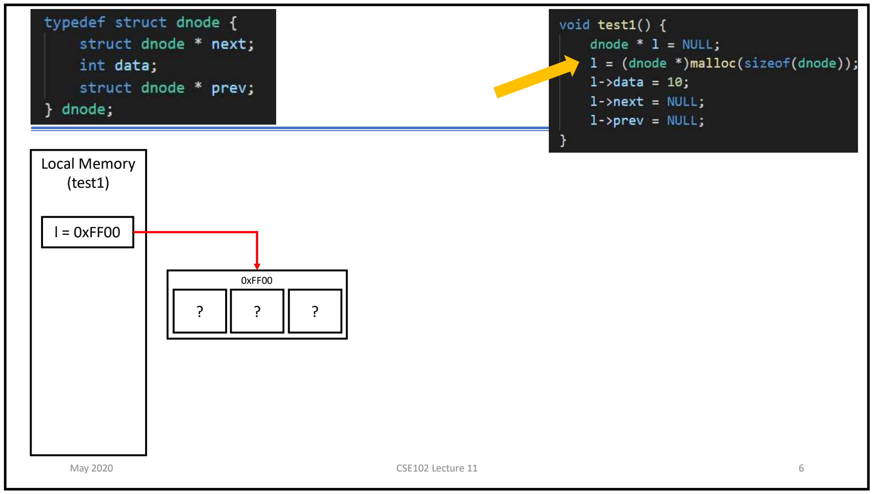


```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

```
void test1() {
    dnode * l = NULL;
    l = (dnode *)malloc(sizeof(dnode));
    l->data = 10;
    l->next = NULL;
    l->prev = NULL;
}
```

May 2020 CSE102 Lecture 11 5

5

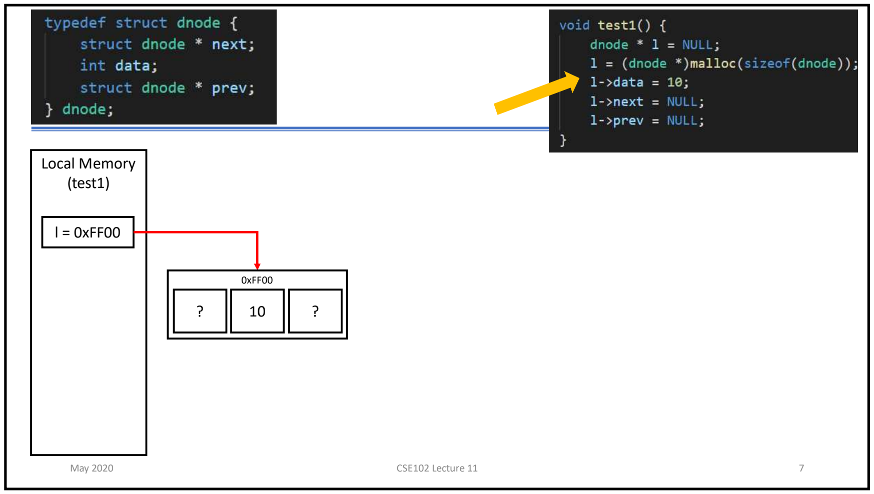


```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

```
void test1() {
    dnode * l = NULL;
    l = (dnode *)malloc(sizeof(dnode));
    l->data = 10;
    l->next = NULL;
    l->prev = NULL;
}
```

May 2020 CSE102 Lecture 11 6

6

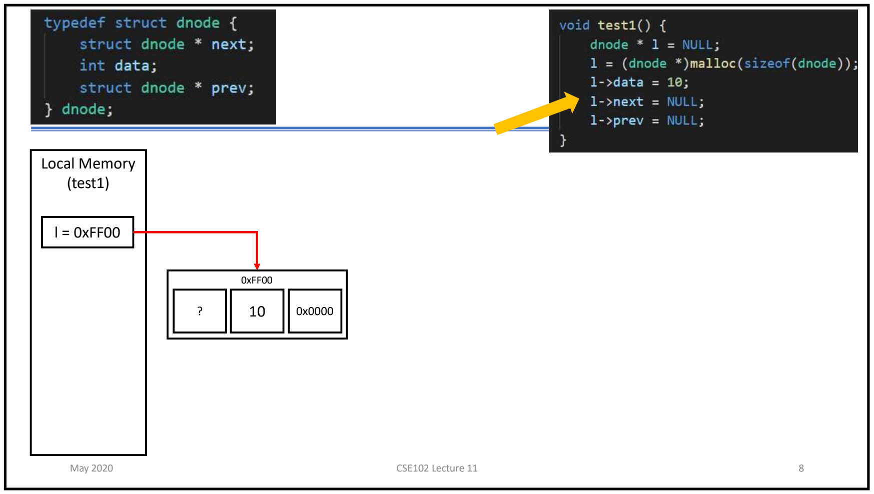


```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

```
void test1() {
    dnode * l = NULL;
    l = (dnode *)malloc(sizeof(dnode));
    l->data = 10;
    l->next = NULL;
    l->prev = NULL;
}
```

May 2020 CSE102 Lecture 11 7

7

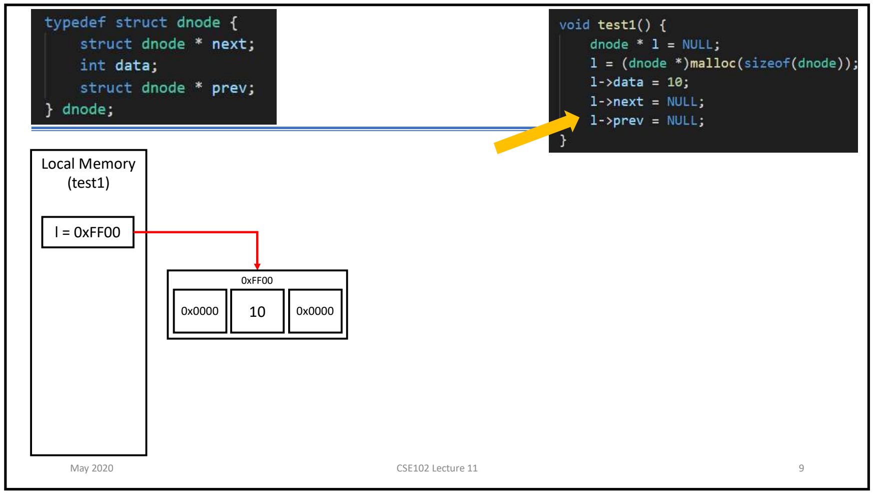


```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

```
void test1() {
    dnode * l = NULL;
    l = (dnode *)malloc(sizeof(dnode));
    l->data = 10;
    l->next = NULL;
    l->prev = NULL;
}
```

May 2020 CSE102 Lecture 11 8

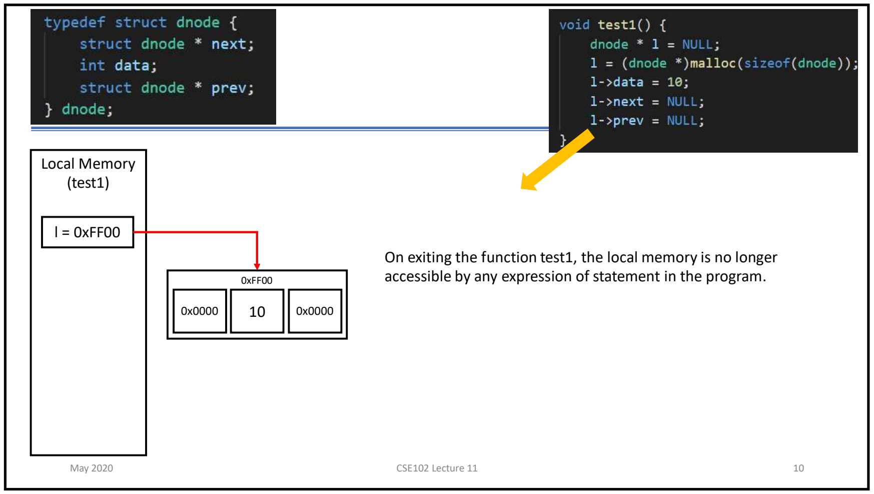
8



```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

```
void test1() {
    dnode * l = NULL;
    l = (dnode *)malloc(sizeof(dnode));
    l->data = 10;
    l->next = NULL;
    l->prev = NULL;
}
```

9

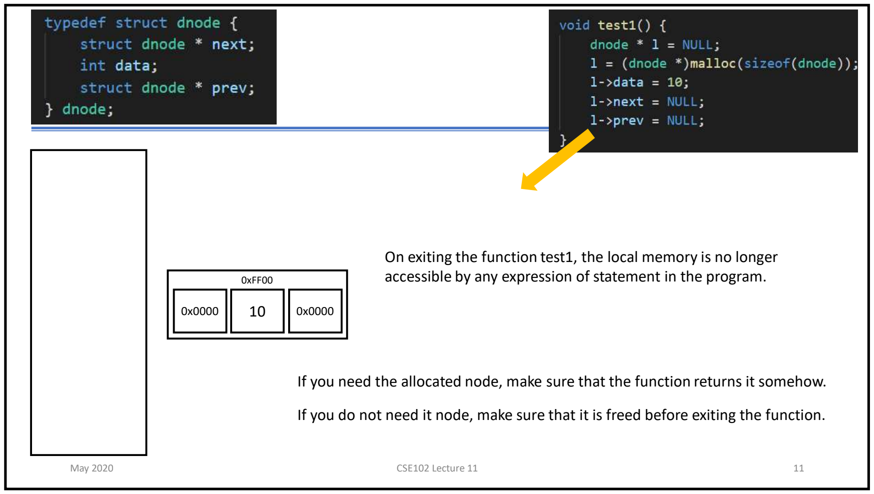


```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

```
void test1() {
    dnode * l = NULL;
    l = (dnode *)malloc(sizeof(dnode));
    l->data = 10;
    l->next = NULL;
    l->prev = NULL;
}
```

On exiting the function test1, the local memory is no longer accessible by any expression of statement in the program.

10



```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

```
void test1() {
    dnode * l = NULL;
    l = (dnode *)malloc(sizeof(dnode));
    l->data = 10;
    l->next = NULL;
    l->prev = NULL;
}
```

On exiting the function test1, the local memory is no longer accessible by any expression of statement in the program.

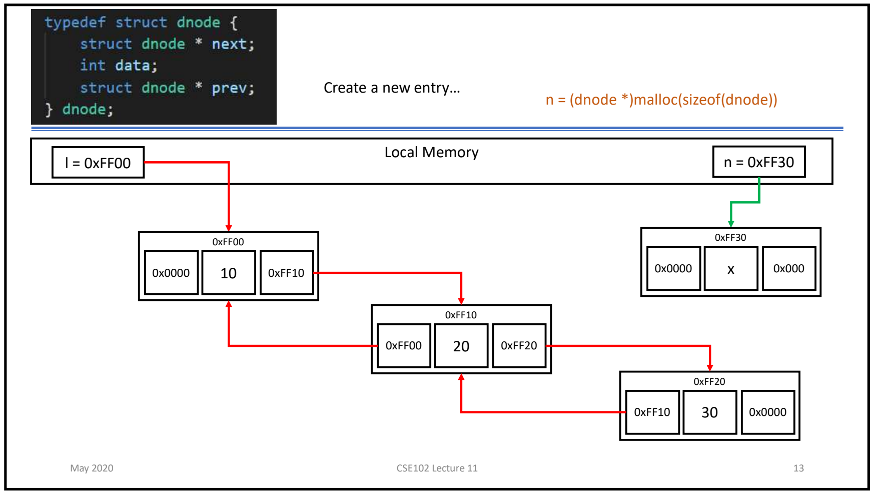
If you need the allocated node, make sure that the function returns it somehow.

If you do not need it node, make sure that it is freed before exiting the function.

11

Insert

12

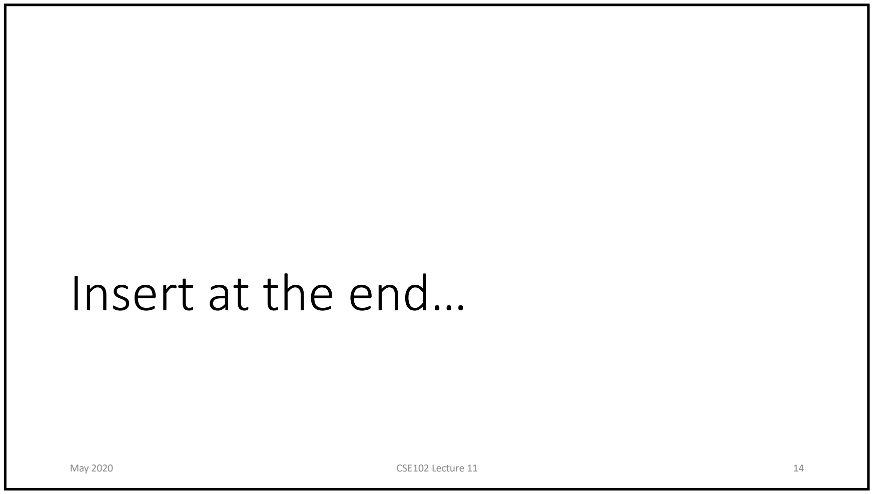


```
typedef struct dnode {
     struct dnode * next;
     int data;
     struct dnode * prev;
} dnode;
```

Create a new entry…

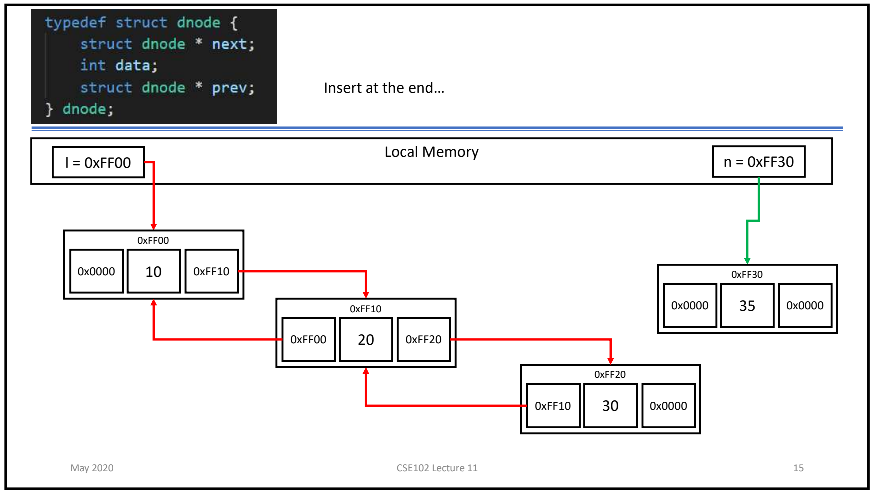
n = (dnode *)malloc(sizeof(dnode))

13



Insert at the end…

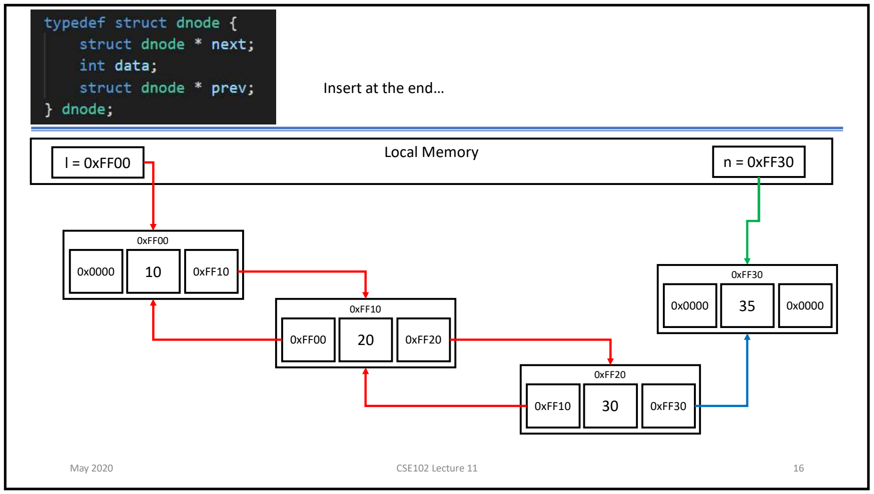
14



```
typedef struct dnode {
     struct dnode * next;
     int data;
     struct dnode * prev;
} dnode;
```

Insert at the end…

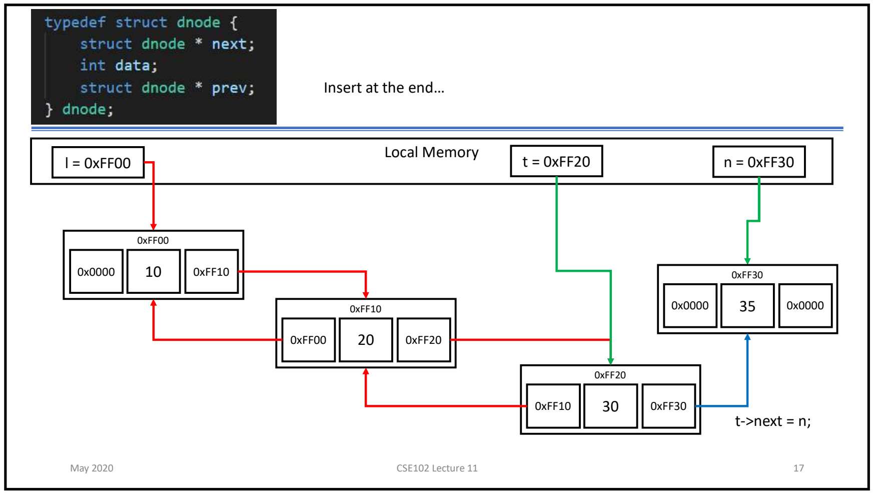
15



```
typedef struct dnode {
     struct dnode * next;
     int data;
     struct dnode * prev;
} dnode;
```

Insert at the end…

16

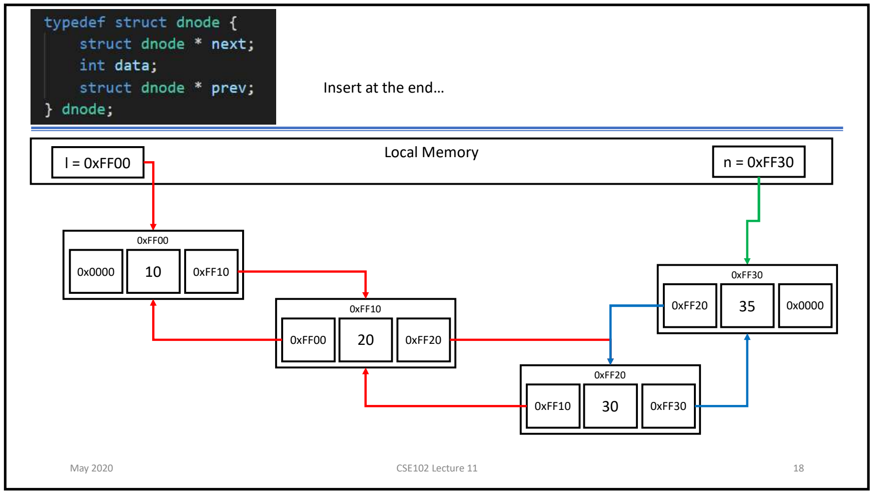


```c
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Insert at the end…

t->next = n;

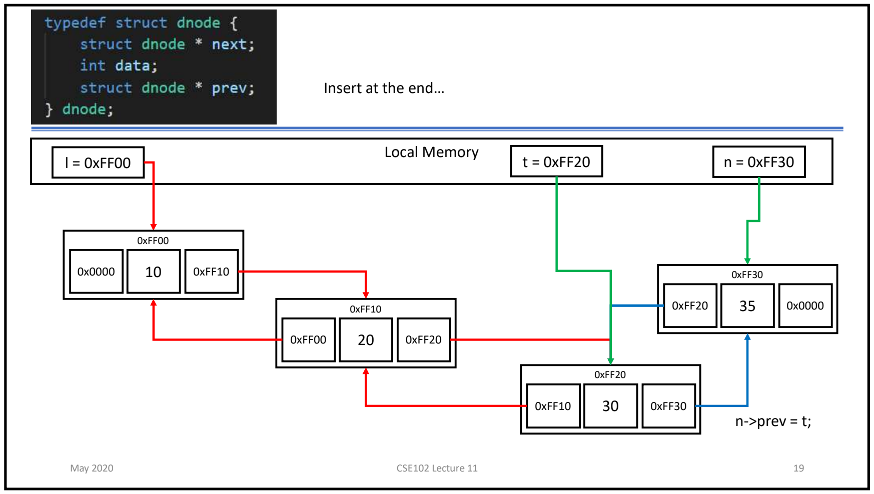
17



```c
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Insert at the end…

18

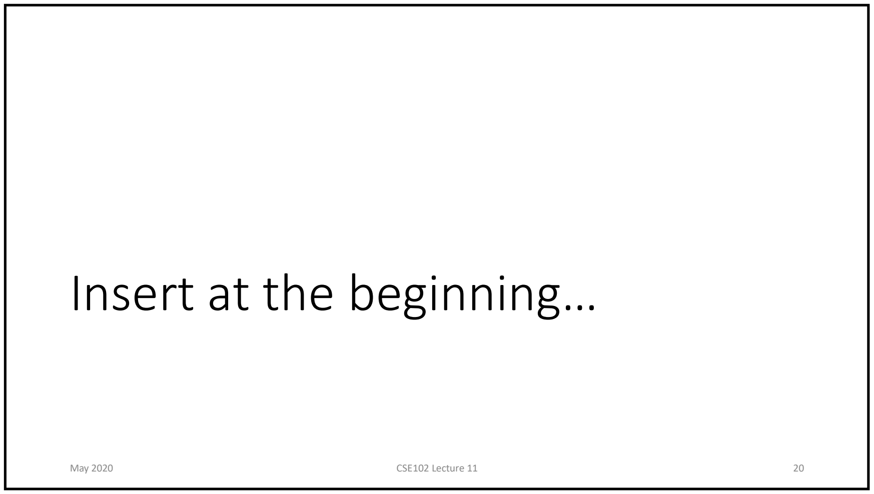


```c
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Insert at the end…

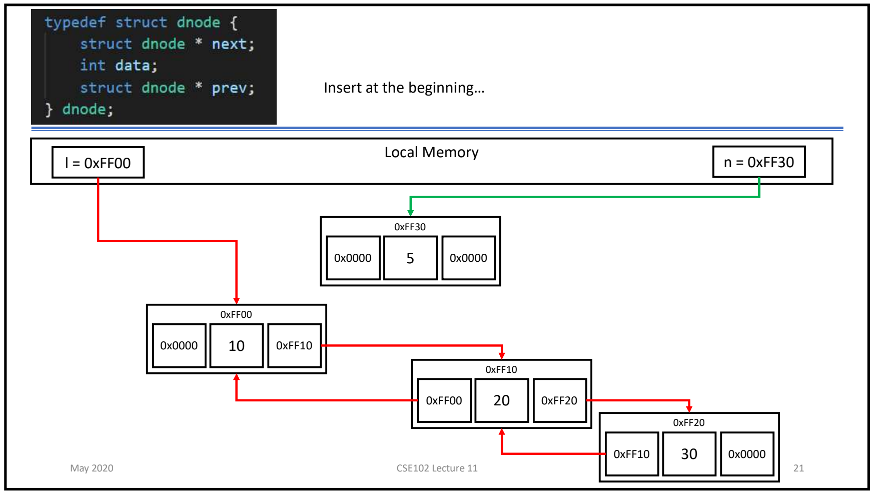
n->prev = t;

19



Insert at the beginning…

20



```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Insert at the beginning…

Local Memory

l = 0xFF00

n = 0xFF30

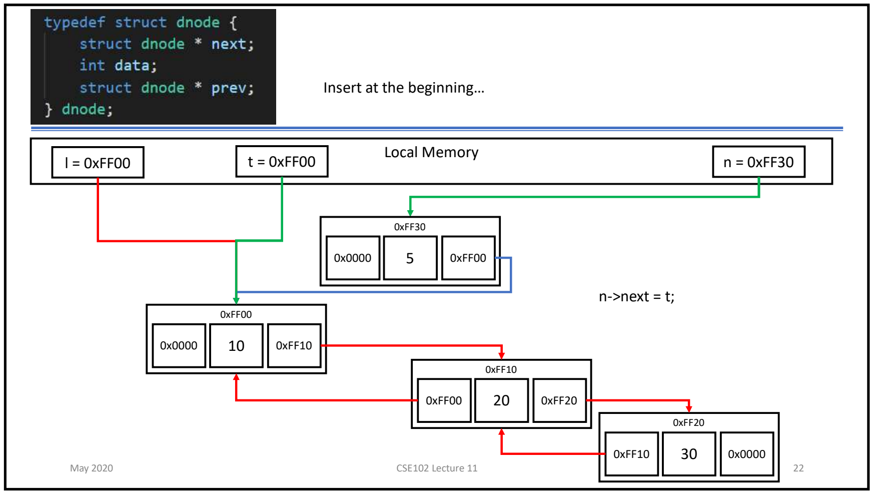
0xFF30: 0x0000 | 5 | 0x0000

0xFF00: 0x0000 | 10 | 0xFF10

0xFF10: 0xFF00 | 20 | 0xFF20

0xFF20: 0xFF10 | 30 | 0x0000

21



```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Insert at the beginning…

Local Memory

l = 0xFF00

t = 0xFF00

n = 0xFF30

0xFF30: 0x0000 | 5 | 0xFF00

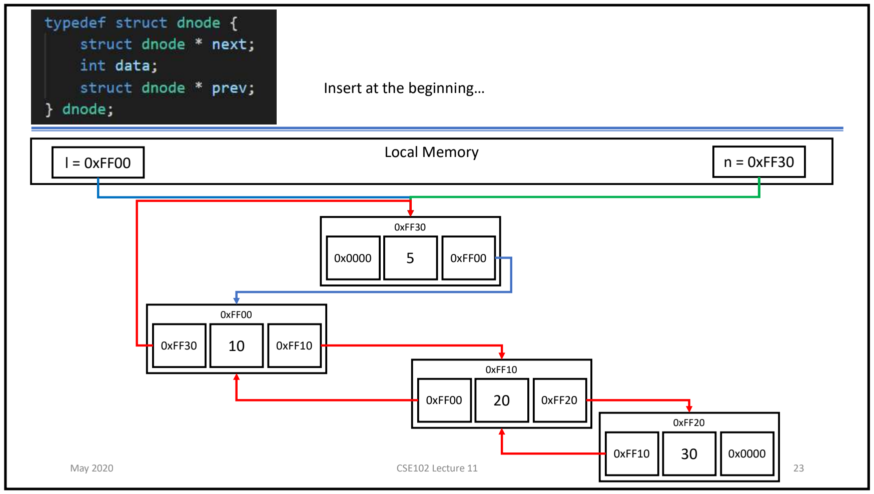
n->next = t;

0xFF00: 0x0000 | 10 | 0xFF10

0xFF10: 0xFF00 | 20 | 0xFF20

0xFF20: 0xFF10 | 30 | 0x0000

22



```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Insert at the beginning…

Local Memory

l = 0xFF00

n = 0xFF30

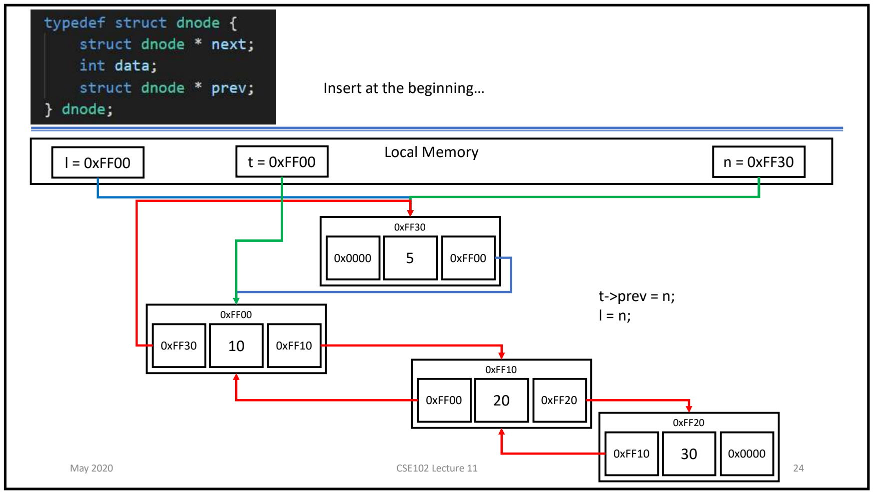
0xFF30: 0x0000 | 5 | 0xFF00

0xFF00: 0xFF30 | 10 | 0xFF10

0xFF10: 0xFF00 | 20 | 0xFF20

0xFF20: 0xFF10 | 30 | 0x0000

23



```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Insert at the beginning…

Local Memory

l = 0xFF00

t = 0xFF00

n = 0xFF30

0xFF30: 0x0000 | 5 | 0xFF00

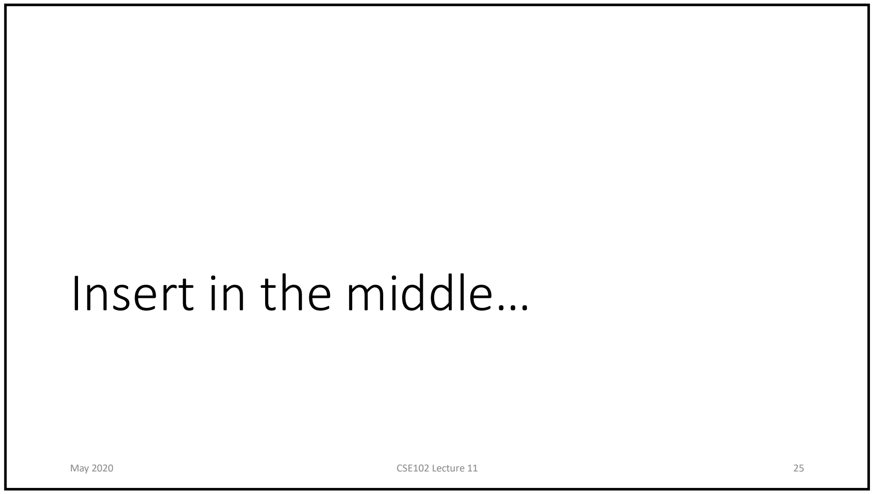
t->prev = n;
l = n;

0xFF00: 0xFF30 | 10 | 0xFF10

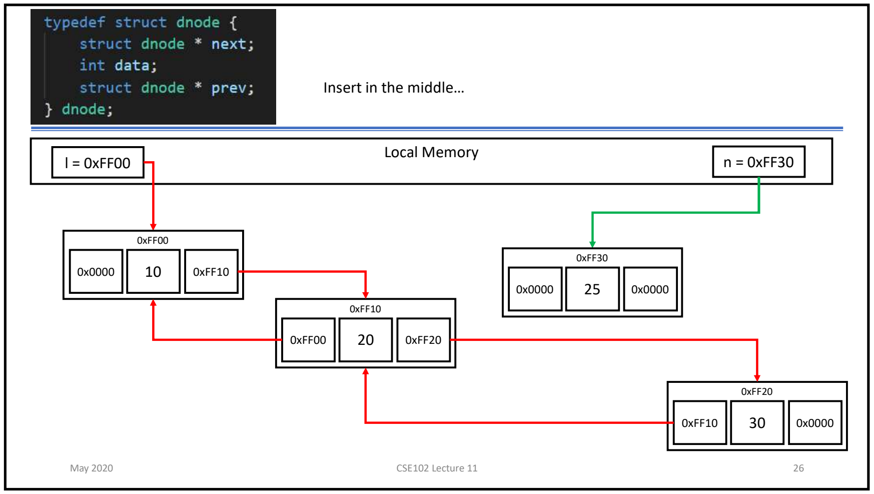
0xFF10: 0xFF00 | 20 | 0xFF20

0xFF20: 0xFF10 | 30 | 0x0000

24



Insert in the middle…



```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Insert in the middle…

Local Memory

l = 0xFF00

n = 0xFF30

0xFF00

0x0000 10 0xFF10

0xFF10

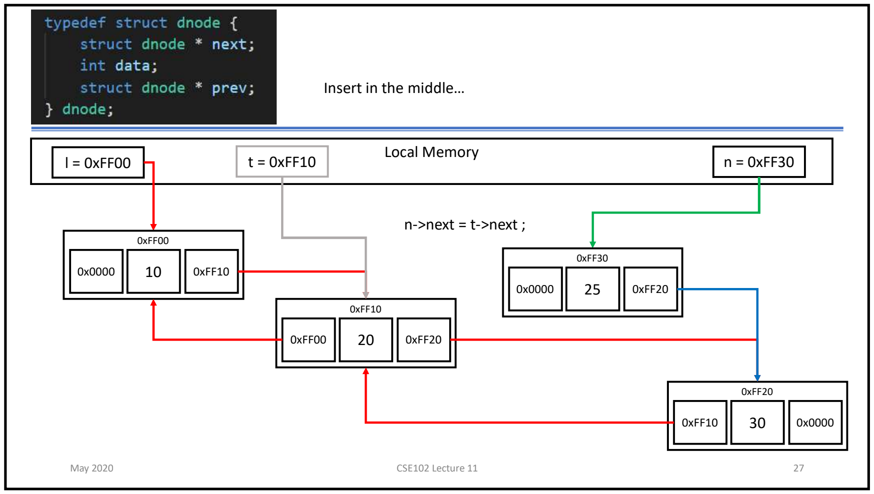
0xFF00 20 0xFF20

0xFF30

0x0000 25 0x0000

0xFF20

0xFF10 30 0x0000



```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Insert in the middle…

Local Memory

l = 0xFF00

t = 0xFF10

n = 0xFF30

n->next = t->next ;

0xFF00

0x0000 10 0xFF10

0xFF10

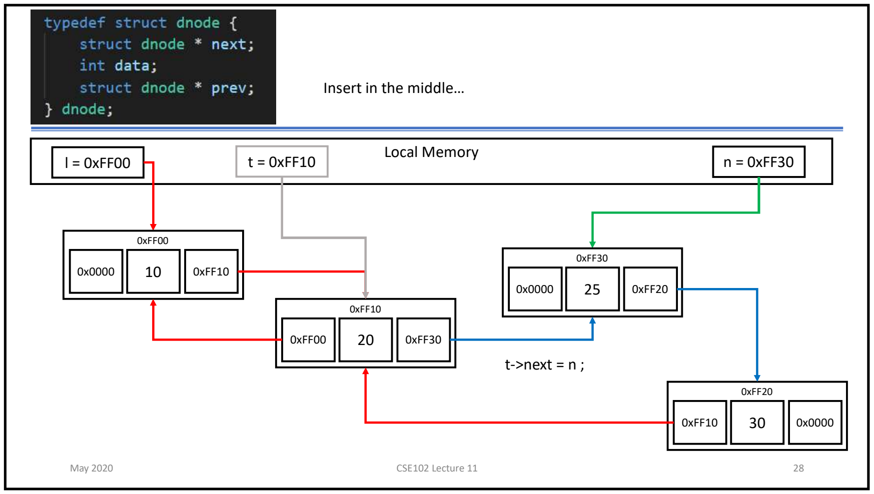
0xFF00 20 0xFF20

0xFF30

0x0000 25 0xFF20

0xFF20

0xFF10 30 0x0000



```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Insert in the middle…

Local Memory

l = 0xFF00

t = 0xFF10

n = 0xFF30

0xFF00

0x0000 10 0xFF10

0xFF10

0xFF00 20 0xFF30

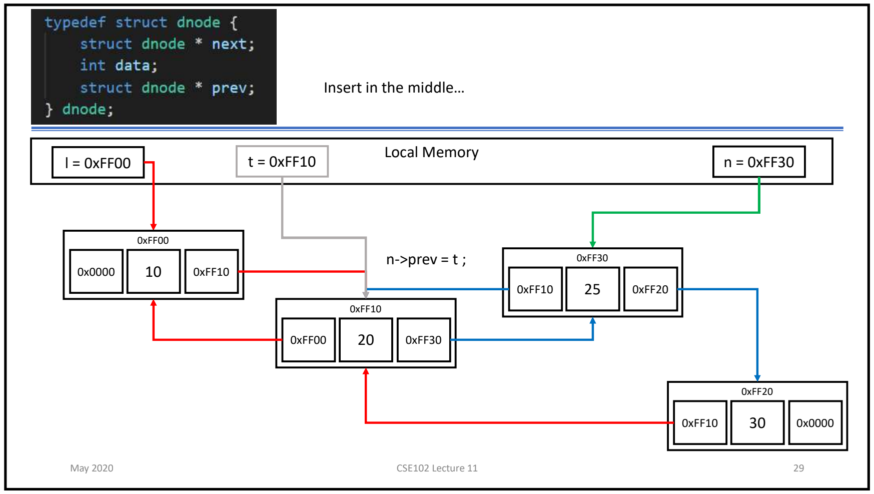
0xFF30

0x0000 25 0xFF20

t->next = n ;

0xFF20

0xFF10 30 0x0000

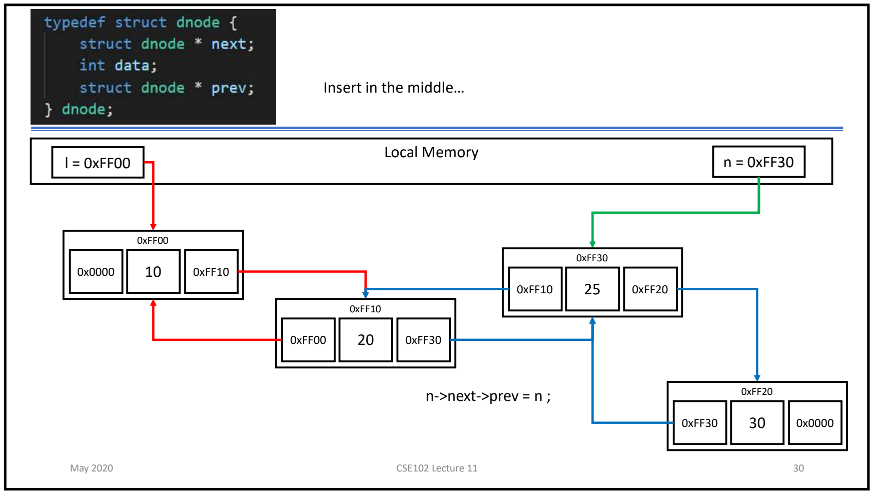


```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Insert in the middle…

n->prev = t ;

29

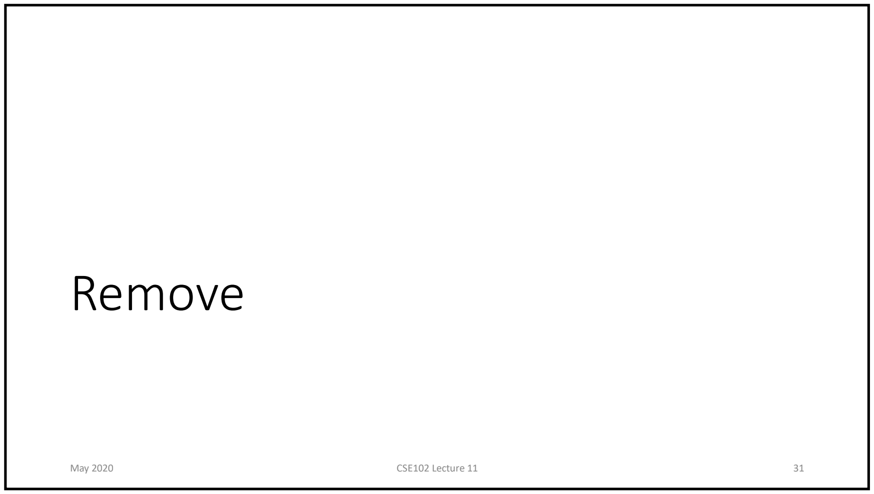


```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Insert in the middle…

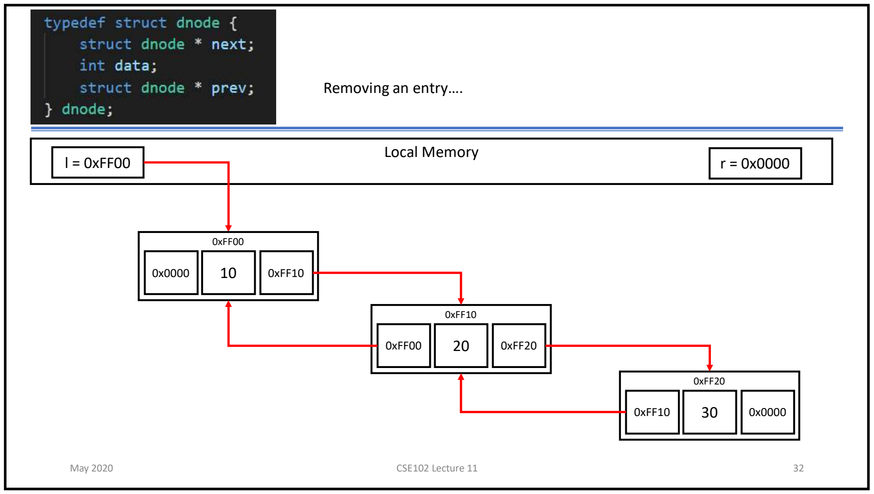
n->next->prev = n ;

30



Remove

31



```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

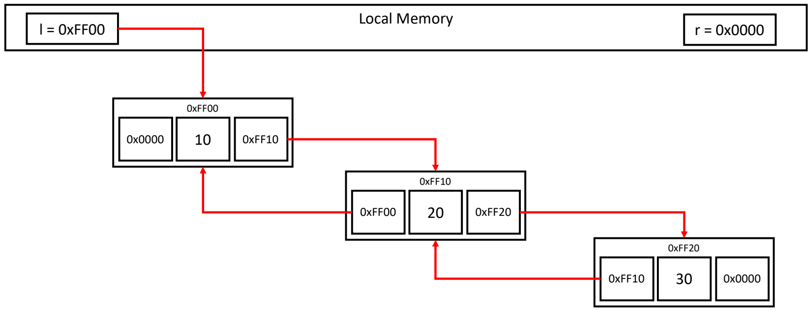
Removing an entry….

32

Remove Last

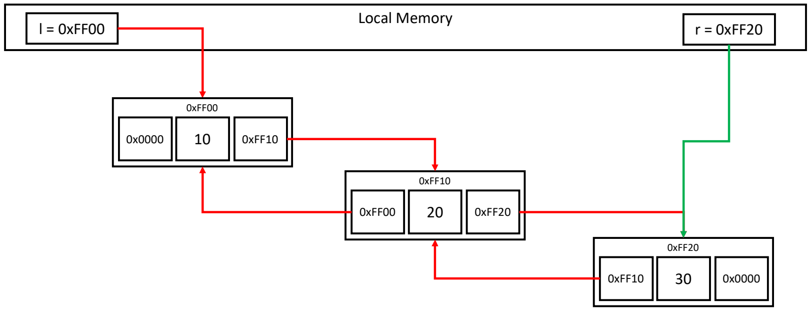
```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Removing the last entry....



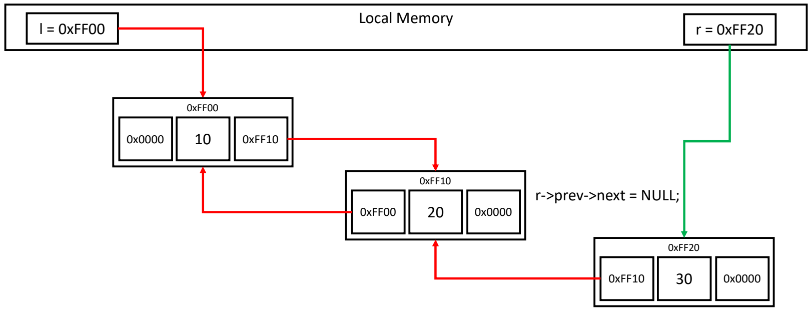
```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Removing the last entry....

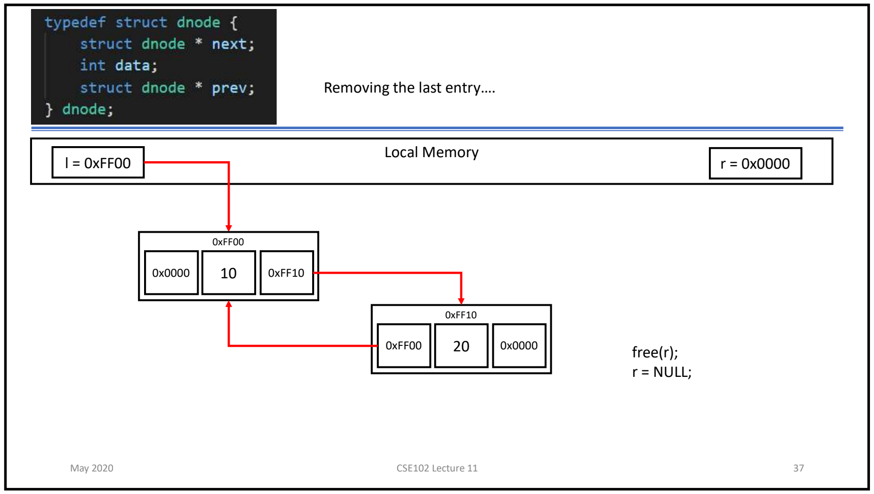


```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Removing the last entry....



r->prev->next = NULL;



```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Removing the last entry….

Local Memory

l = 0xFF00

r = 0x0000

0xFF00: 0x0000 | 10 | 0xFF10

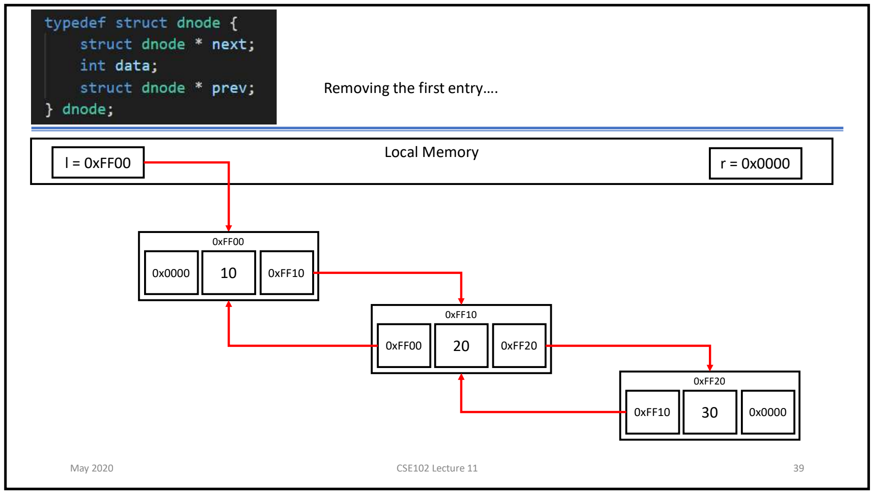
0xFF10: 0xFF00 | 20 | 0x0000

free(r);
r = NULL;

37

Remove First

38



```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Removing the first entry….

Local Memory

l = 0xFF00

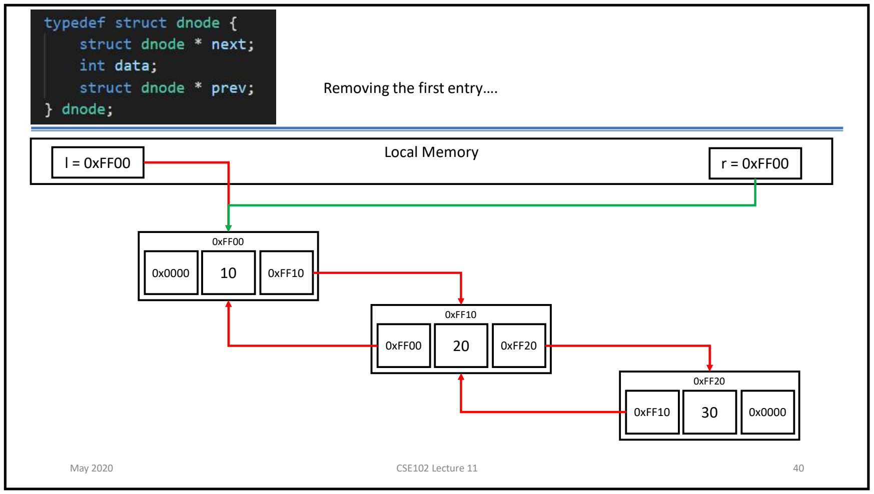
r = 0x0000

0xFF00: 0x0000 | 10 | 0xFF10

0xFF10: 0xFF00 | 20 | 0xFF20

0xFF20: 0xFF10 | 30 | 0x0000

39



```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Removing the first entry….

Local Memory

l = 0xFF00

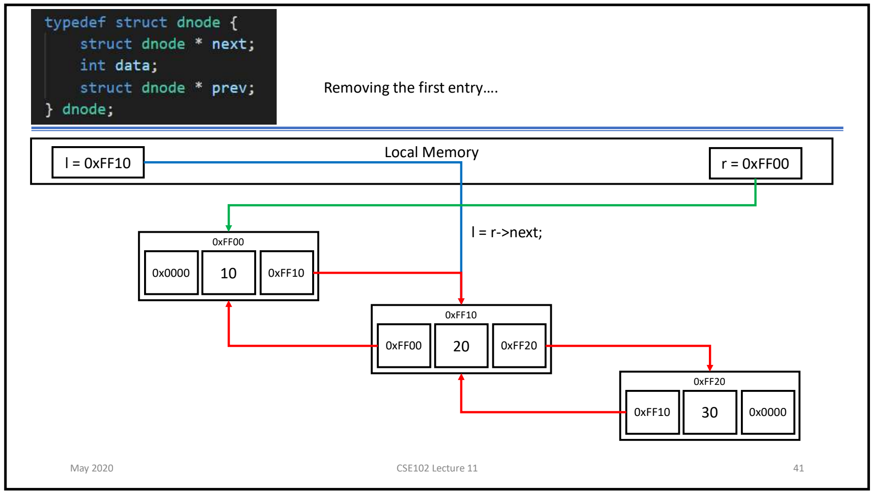
r = 0xFF00

0xFF00: 0x0000 | 10 | 0xFF10

0xFF10: 0xFF00 | 20 | 0xFF20

0xFF20: 0xFF10 | 30 | 0x0000

40



```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Removing the first entry….

Local Memory

l = 0xFF10

r = 0xFF00

l = r->next;

0xFF00

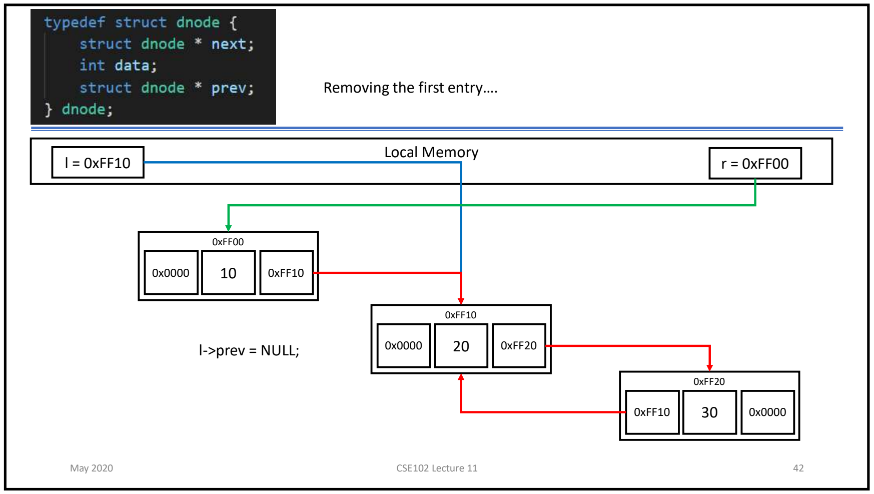
0x0000	10	0xFF10

0xFF10

0xFF00	20	0xFF20

0xFF20

0xFF10	30	0x0000



```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Removing the first entry….

Local Memory

l = 0xFF10

r = 0xFF00

0xFF00

0x0000	10	0xFF10

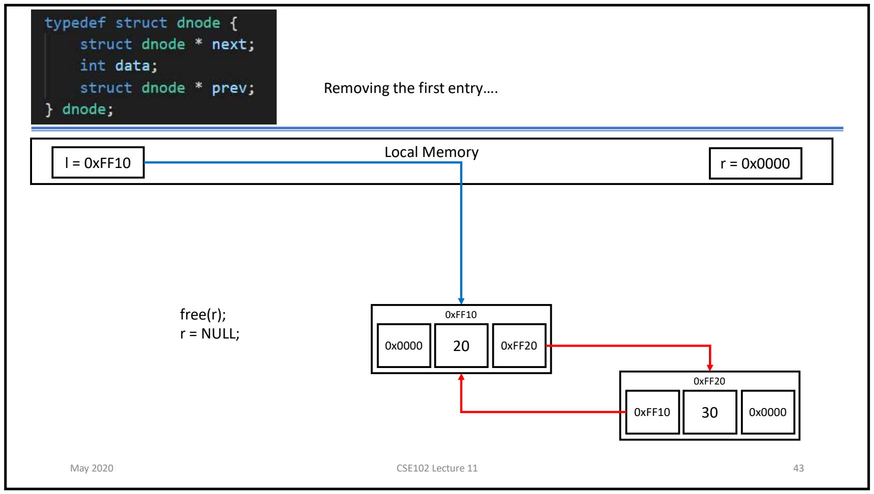
l->prev = NULL;

0xFF10

0x0000	20	0xFF20

0xFF20

0xFF10	30	0x0000



```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Removing the first entry….

Local Memory

l = 0xFF10

r = 0x0000

free(r);
r = NULL;

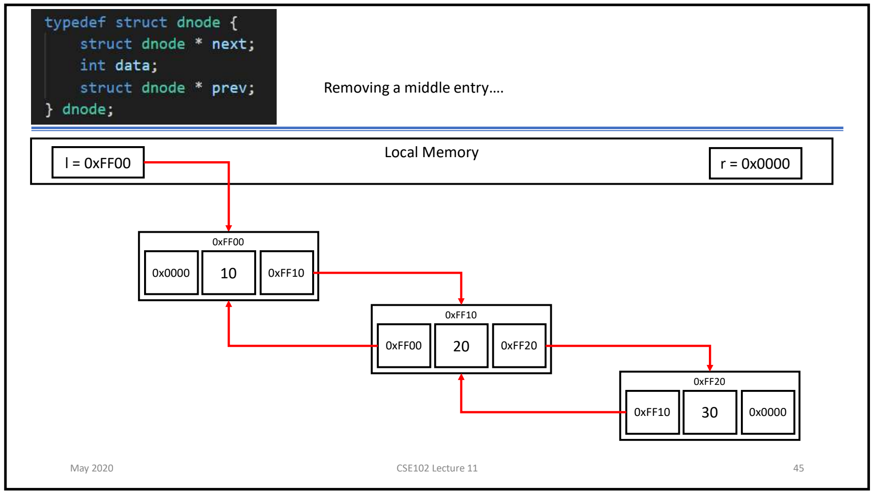
0xFF10

0x0000	20	0xFF20

0xFF20

0xFF10	30	0x0000

Remove Middle



```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Removing a middle entry….

Local Memory

l = 0xFF00

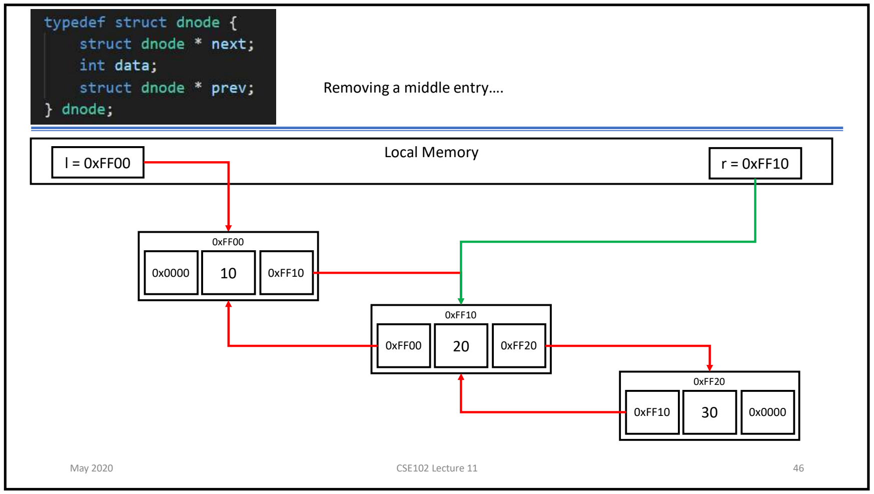
r = 0x0000

0xFF00: 0x0000 | 10 | 0xFF10

0xFF10: 0xFF00 | 20 | 0xFF20

0xFF20: 0xFF10 | 30 | 0x0000

45



```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Removing a middle entry….

Local Memory

l = 0xFF00

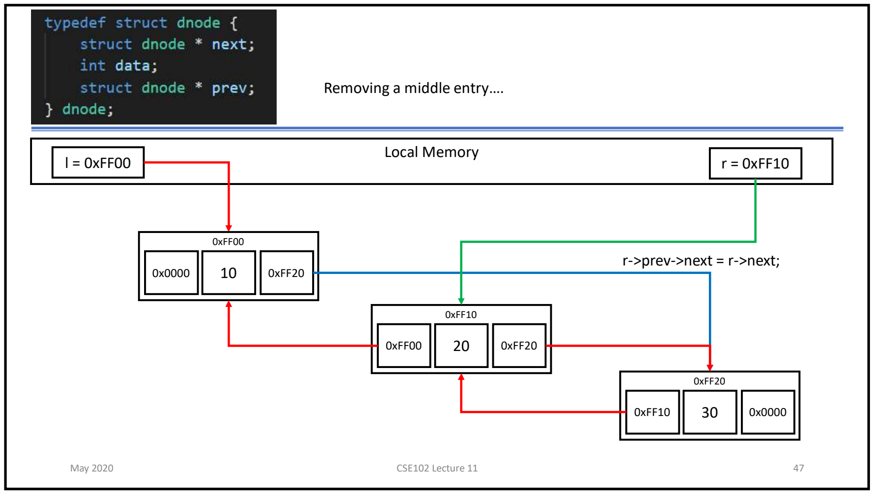
r = 0xFF10

0xFF00: 0x0000 | 10 | 0xFF10

0xFF10: 0xFF00 | 20 | 0xFF20

0xFF20: 0xFF10 | 30 | 0x0000

46



```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Removing a middle entry….

Local Memory

l = 0xFF00

r = 0xFF10

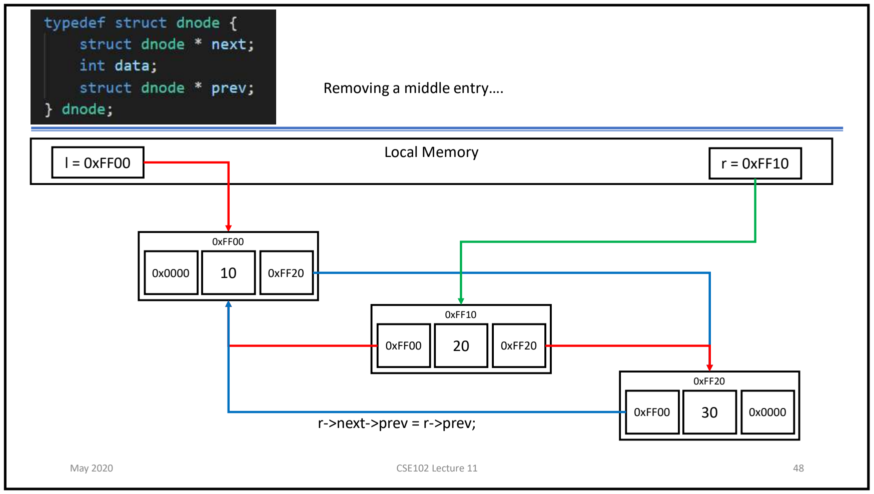
0xFF00: 0x0000 | 10 | 0xFF20

r->prev->next = r->next;

0xFF10: 0xFF00 | 20 | 0xFF20

0xFF20: 0xFF10 | 30 | 0x0000

47



```
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Removing a middle entry….

Local Memory

l = 0xFF00

r = 0xFF10

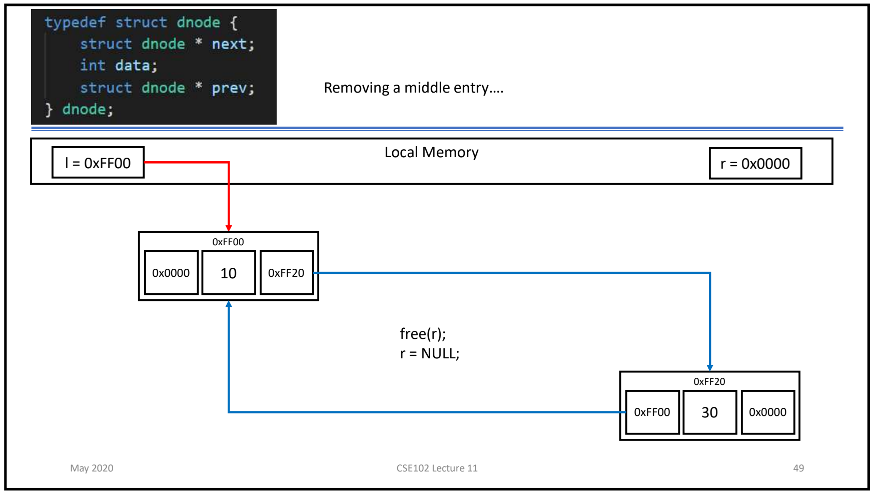
0xFF00: 0x0000 | 10 | 0xFF20

0xFF10: 0xFF00 | 20 | 0xFF20

0xFF20: 0xFF00 | 30 | 0x0000

r->next->prev = r->prev;

48

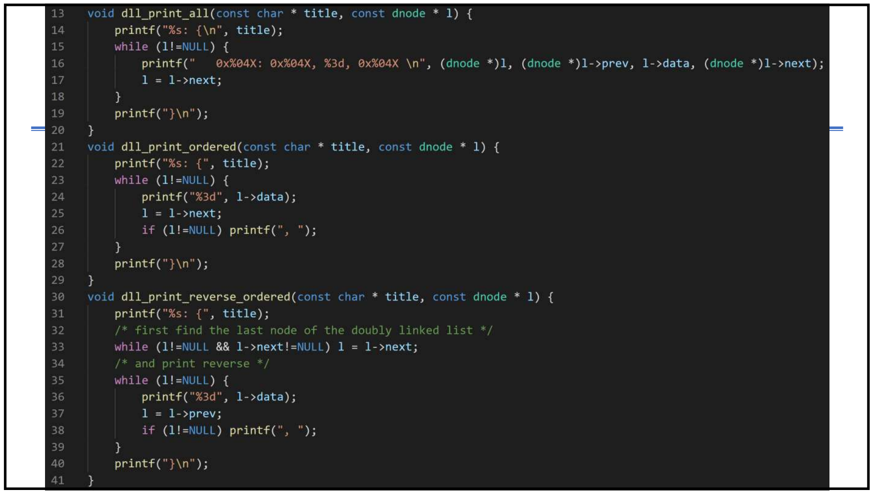


```c
typedef struct dnode {
    struct dnode * next;
    int data;
    struct dnode * prev;
} dnode;
```

Removing a middle entry….

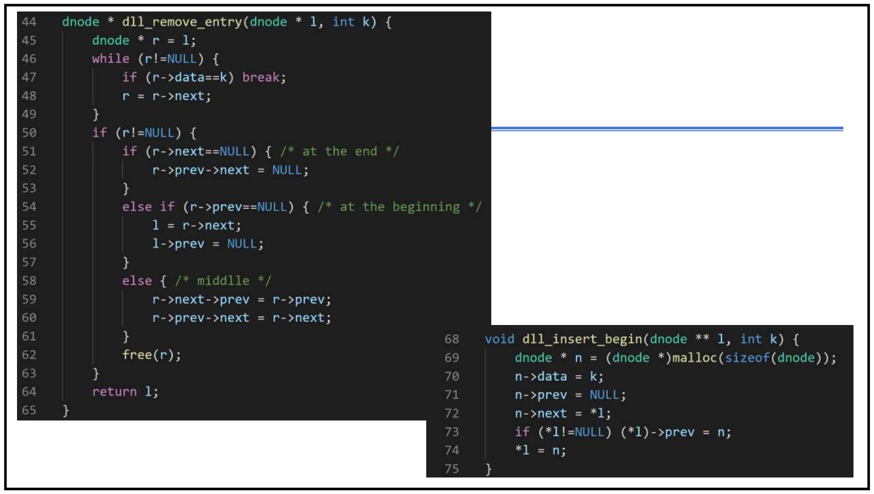
free(r);
r = NULL;

49



```c
void dll_print_all(const char * title, const dnode * l) {
    printf("%s: {\n", title);
    while (l!=NULL) {
        printf("   0x%04X: 0x%04X, %3d, 0x%04X \n", (dnode *)l, (dnode *)l->prev, l->data, (dnode *)l->next);
        l = l->next;
    }
    printf("}\n");
}
void dll_print_ordered(const char * title, const dnode * l) {
    printf("%s: {", title);
    while (l!=NULL) {
        printf("%3d", l->data);
        l = l->next;
        if (l!=NULL) printf(", ");
    }
    printf("}\n");
}
void dll_print_reverse_ordered(const char * title, const dnode * l) {
    printf("%s: {", title);
    /* first find the last node of the doubly linked list */
    while (l!=NULL && l->next!=NULL) l = l->next;
    /* and print reverse */
    while (l!=NULL) {
        printf("%3d", l->data);
        l = l->prev;
        if (l!=NULL) printf(", ");
    }
    printf("}\n");
}
```

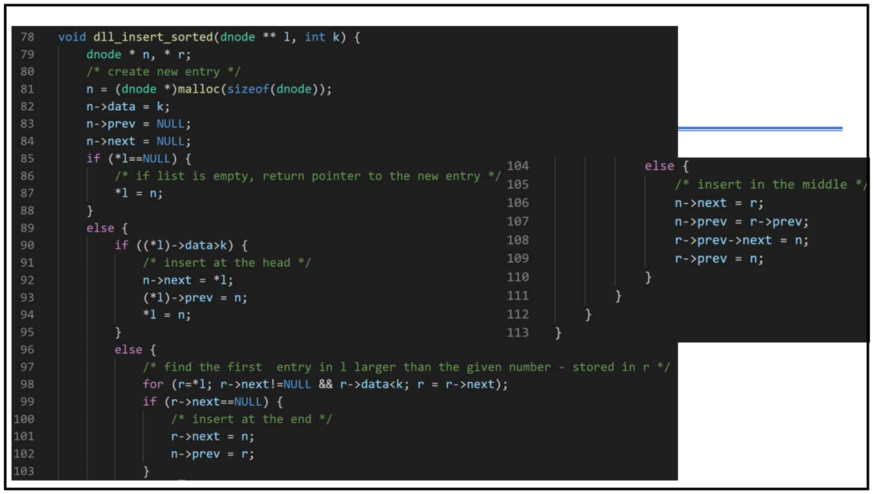
50



```c
dnode * dll_remove_entry(dnode * l, int k) {
    dnode * r = l;
    while (r!=NULL) {
        if (r->data==k) break;
        r = r->next;
    }
    if (r!=NULL) {
        if (r->next==NULL) { /* at the end */
            r->prev->next = NULL;
        }
        else if (r->prev==NULL) { /* at the beginning */
            l = r->next;
            l->prev = NULL;
        }
        else { /* middlle */
            r->next->prev = r->prev;
            r->prev->next = r->next;
        }
        free(r);
    }
    return l;
}
```

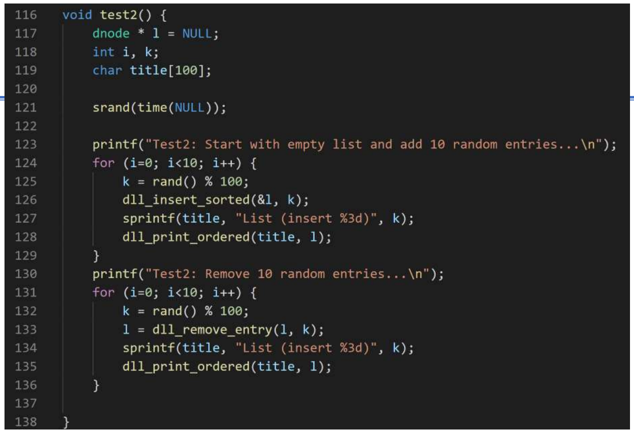
```c
void dll_insert_begin(dnode ** l, int k) {
    dnode * n = (dnode *)malloc(sizeof(dnode));
    n->data = k;
    n->prev = NULL;
    n->next = *l;
    if (*l!=NULL) (*l)->prev = n;
    *l = n;
}
```

51



```c
void dll_insert_sorted(dnode ** l, int k) {
    dnode * n, * r;
    /* create new entry */
    n = (dnode *)malloc(sizeof(dnode));
    n->data = k;
    n->prev = NULL;
    n->next = NULL;
    if (*l==NULL) {
        /* if list is empty, return pointer to the new entry */
        *l = n;
    }
    else {
        if ((*l)->data>k) {
            /* insert at the head */
            n->next = *l;
            (*l)->prev = n;
            *l = n;
        }
        else {
            /* find the first  entry in l larger than the given number - stored in r */
            for (r=*l; r->next!=NULL && r->data<k; r = r->next);
            if (r->next==NULL) {
                /* insert at the end */
                r->next = n;
                n->prev = r;
            }
            else {
                /* insert in the middle */
                n->next = r;
                n->prev = r->prev;
                r->prev->next = n;
                r->prev = n;
            }
        }
    }
}
```

52



```c
void test2() {
    dnode * l = NULL;
    int i, k;
    char title[100];

    srand(time(NULL));

    printf("Test2: Start with empty list and add 10 random entries...\n");
    for (i=0; i<10; i++) {
        k = rand() % 100;
        dll_insert_sorted(&l, k);
        sprintf(title, "List (insert %3d)", k);
        dll_print_ordered(title, l);
    }
    printf("Test2: Remove 10 random entries...\n");
    for (i=0; i<10; i++) {
        k = rand() % 100;
        l = dll_remove_entry(l, k);
        sprintf(title, "List (insert %3d)", k);
        dll_print_ordered(title, l);
    }

}
```