1. The List ADT

1.2 Linked-list

Linked-list Implementation

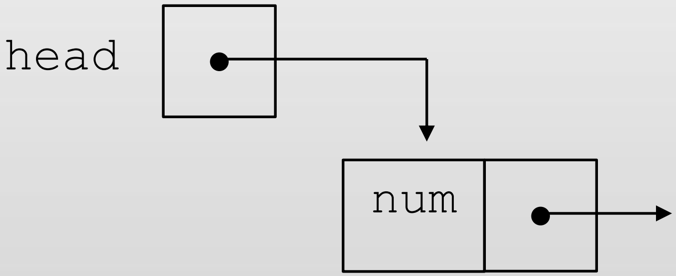
- Elements may not be contiguously stored in the main memory
- Size can grow or shrink at run time
- Slower sequential access to an element

Linked-list Implementation

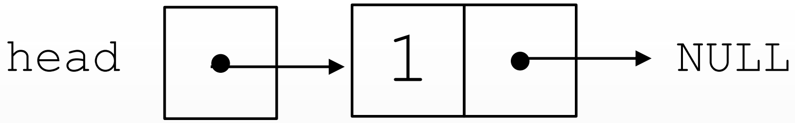
- Implementation issues
  - circular or non-circular
  - singly or doubly
  - use of *dummy* or *sentinel* or *header* node/cell to avoid special cases in the list operations

Recall: Self-referential Structure

```
typedef struct node{
    int num;
    struct node *next;
}list;

list *head;
```

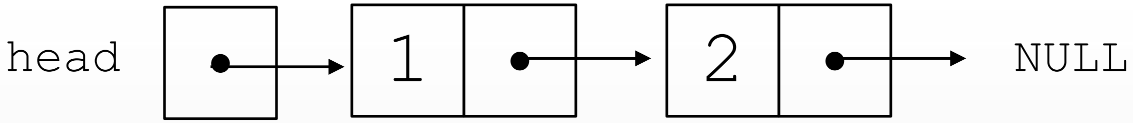


Recall: Build list



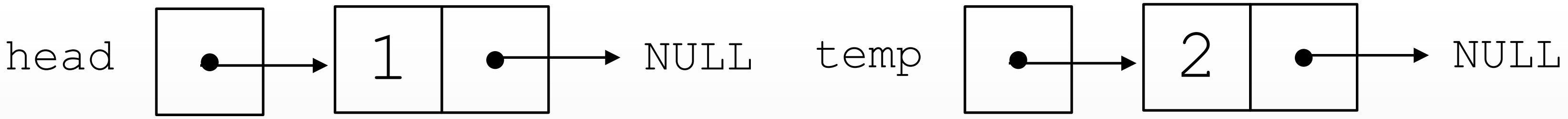
```
head = (list *)malloc(sizeof(list));
head->num = 1;
head->next = NULL;
```

Recall: Build list



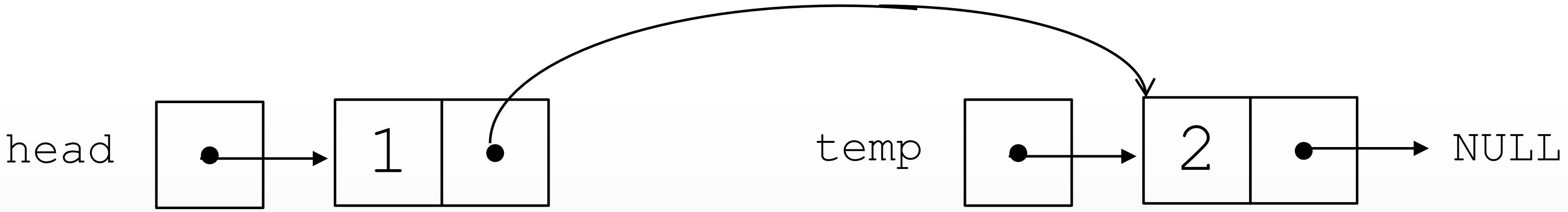
```
    head = (list *)malloc(sizeof(list));
    head->num = 1;
    head->next = NULL;

    temp = (list *)malloc(sizeof(list));
    temp->num = 2;
    temp->next = NULL;
    head->next = temp;
```

Recall: Build list



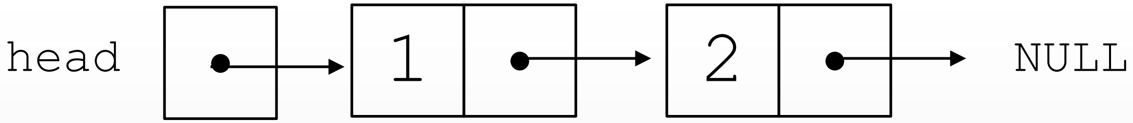
```
head = (list *)malloc(sizeof(list));
head->num = 1;
head->next = NULL;

temp = (list *)malloc(sizeof(list));
temp->num = 2;
temp->next = NULL;
head->next = temp;
```

Recall: Build list



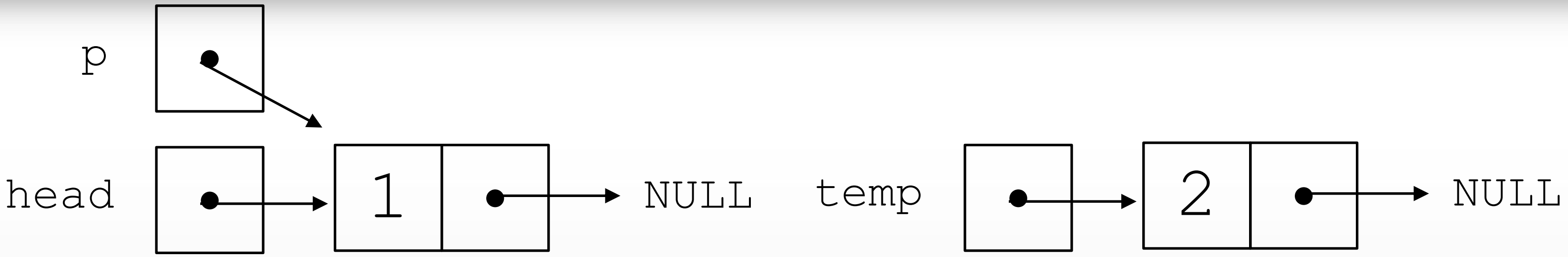
```
head = (list *)malloc(sizeof(list));
head->num = 1;
head->next = NULL;

temp = (list *)malloc(sizeof(list));
temp->num = 2;
temp->next = NULL;
head->next = temp;
```

Recall: Build list



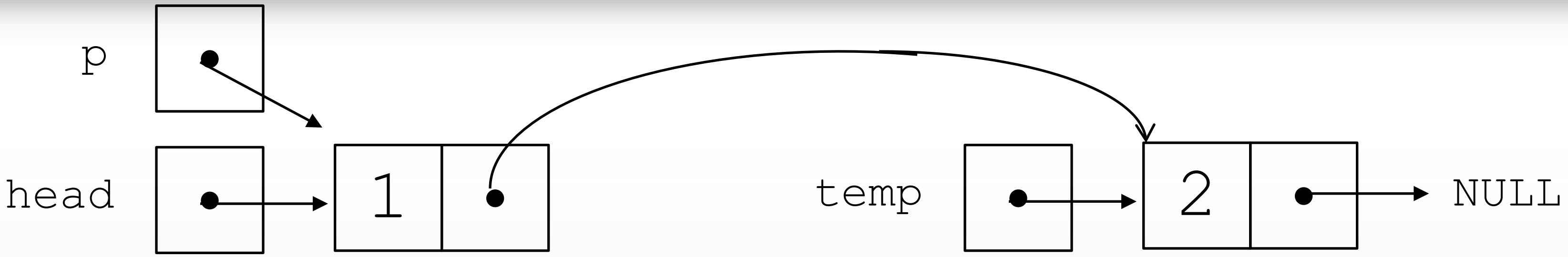
```
head = (list *)malloc(sizeof(list));
head->num = 1;
head->next = NULL;

temp = (list *)malloc(sizeof(list));
temp->num = 2;
temp->next = NULL;
head->next = temp;
```

Recall: Build list



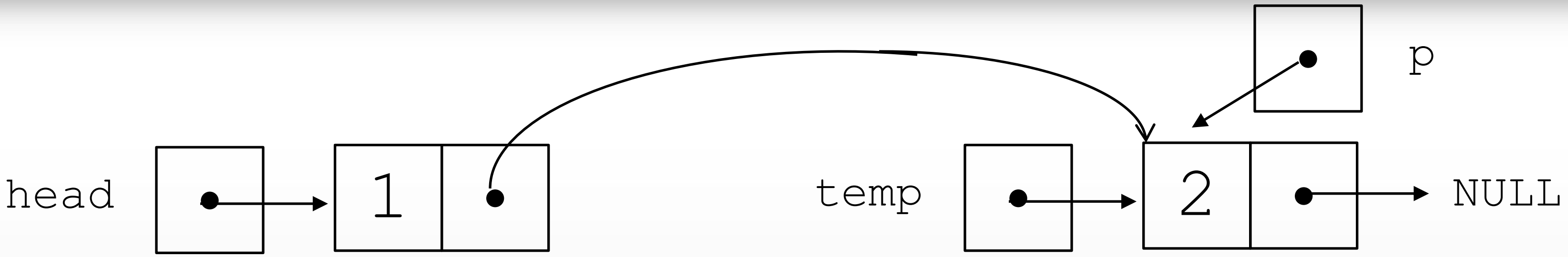
```
head = (list *)malloc(sizeof(list));
head->num = 1;
head->next = NULL;
p = head;
while(there is data){
    temp = (list *)malloc(sizeof(list));
    temp->num = data;
    temp->next = NULL;
    p->next = temp;
    p = p->next;
}
```

Recall: Build list



```
head = (list *)malloc(sizeof(list));
head->num = 1;
head->next = NULL;
p = head;
while(there is data){
    temp = (list *)malloc(sizeof(list));
    temp->num = data;
    temp->next = NULL;
    p->next = temp;
    p = p->next;
}
```

Recall: Build list

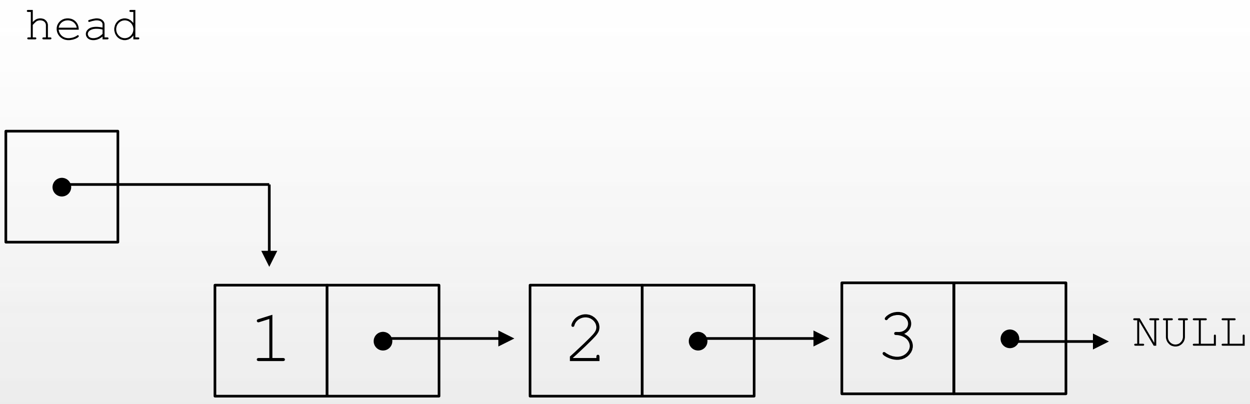


```
head = (list *)malloc(sizeof(list));
head->num = 1;
head->next = NULL;
p = head;
while(there is data){
    temp = (list *)malloc(sizeof(list));
    temp->num = data;
    temp->next = NULL;
    p->next = temp;
    p = p->next;
}
```

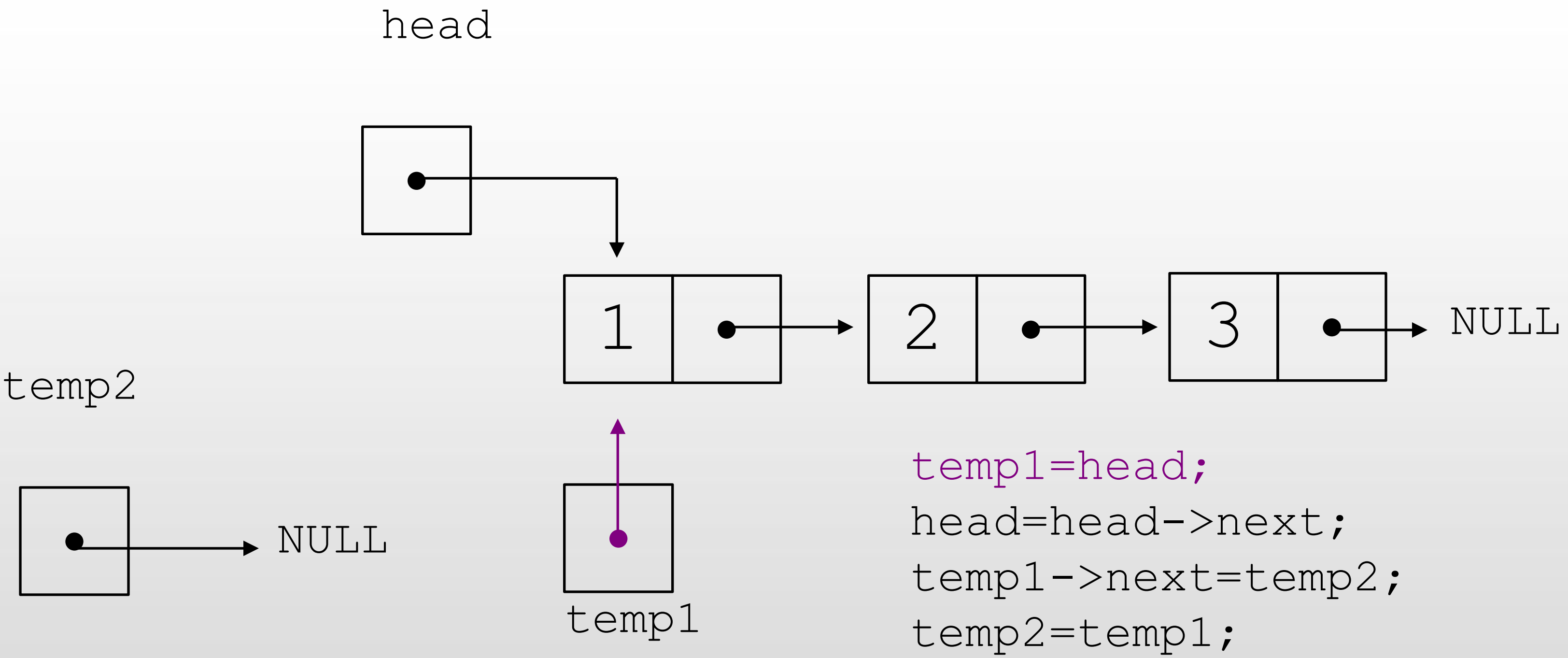
Invert a singly-linked list

```
void invert_list(list *head){
    list *temp1=NULL, *temp2==NULL;

    while(head!=NULL){
        temp1=head;
        head=head->next;
        temp1->next=temp2;
        temp2=temp1;
    }
    head=temp2;
}
```

Invert a singly-linked list

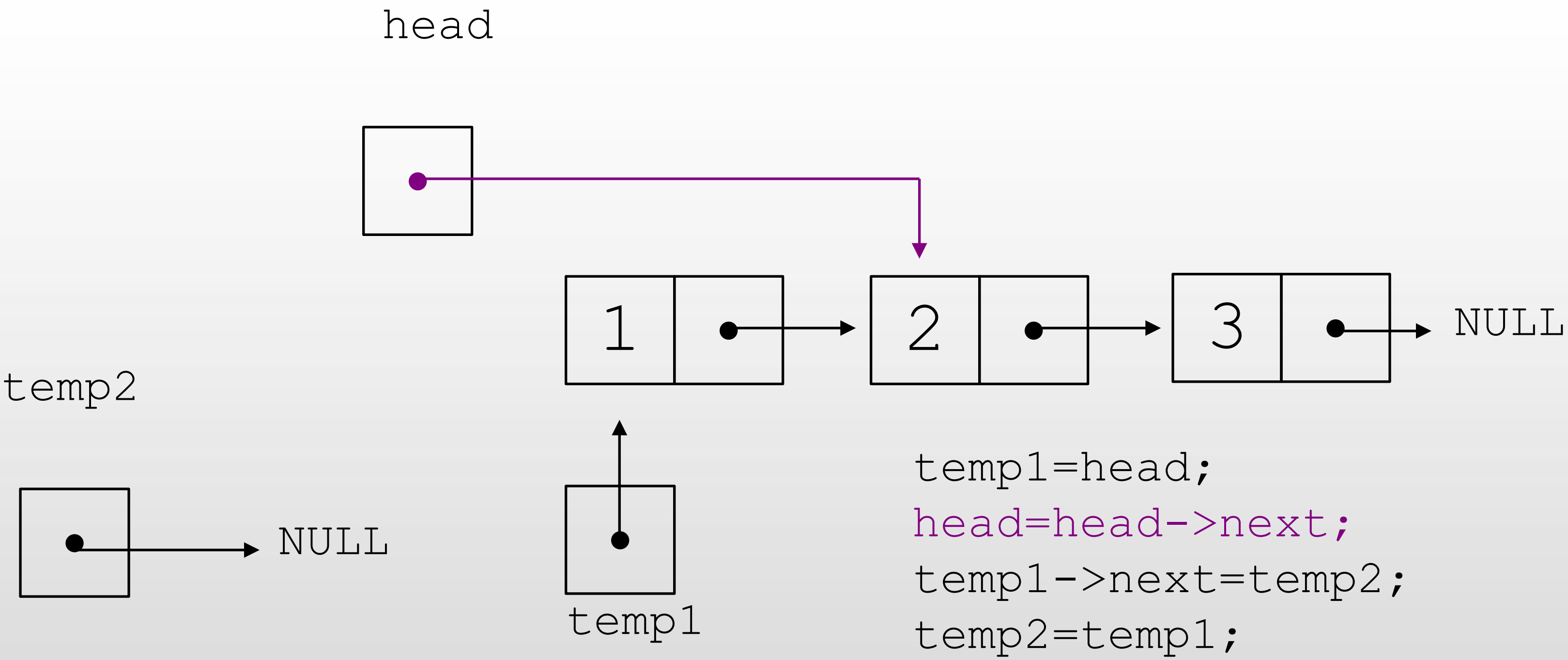


Invert a singly-linked list



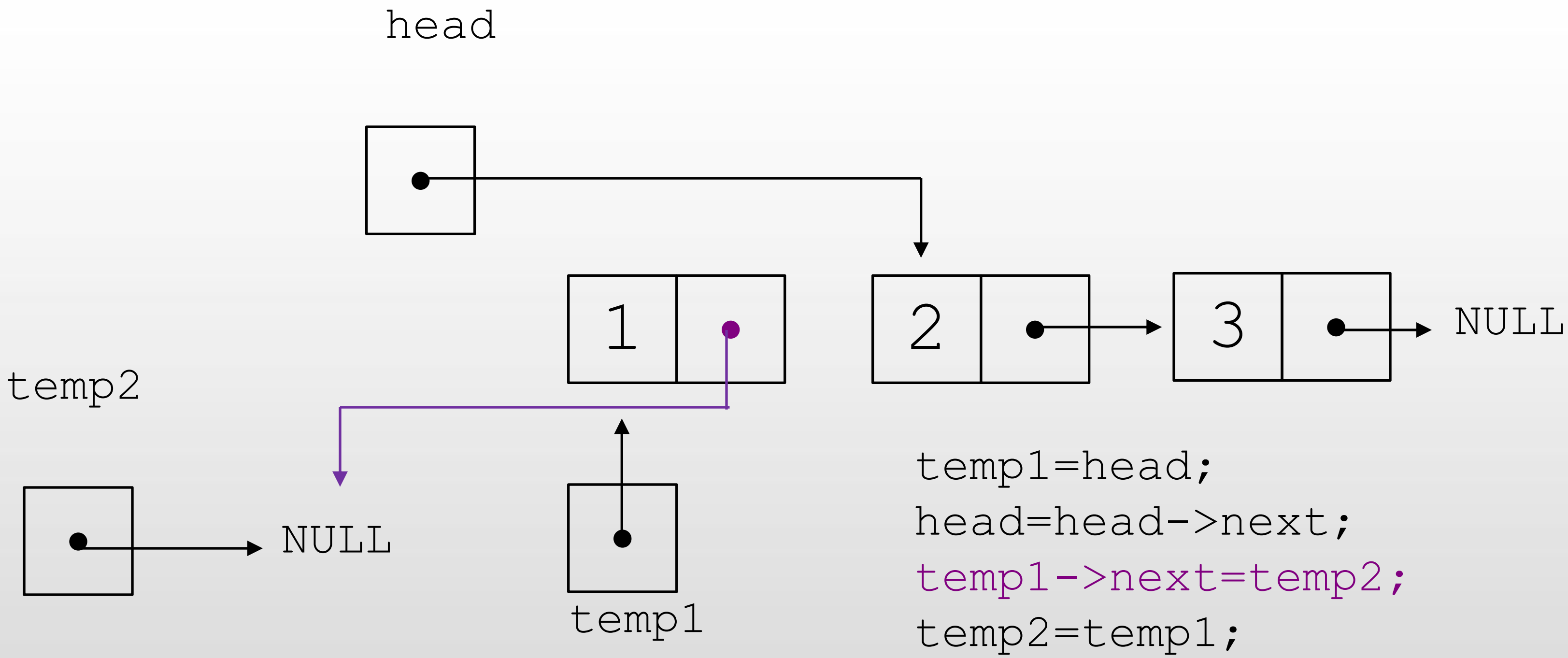
```
temp1=head;
head=head->next;
temp1->next=temp2;
temp2=temp1;
```

Invert a singly-linked list



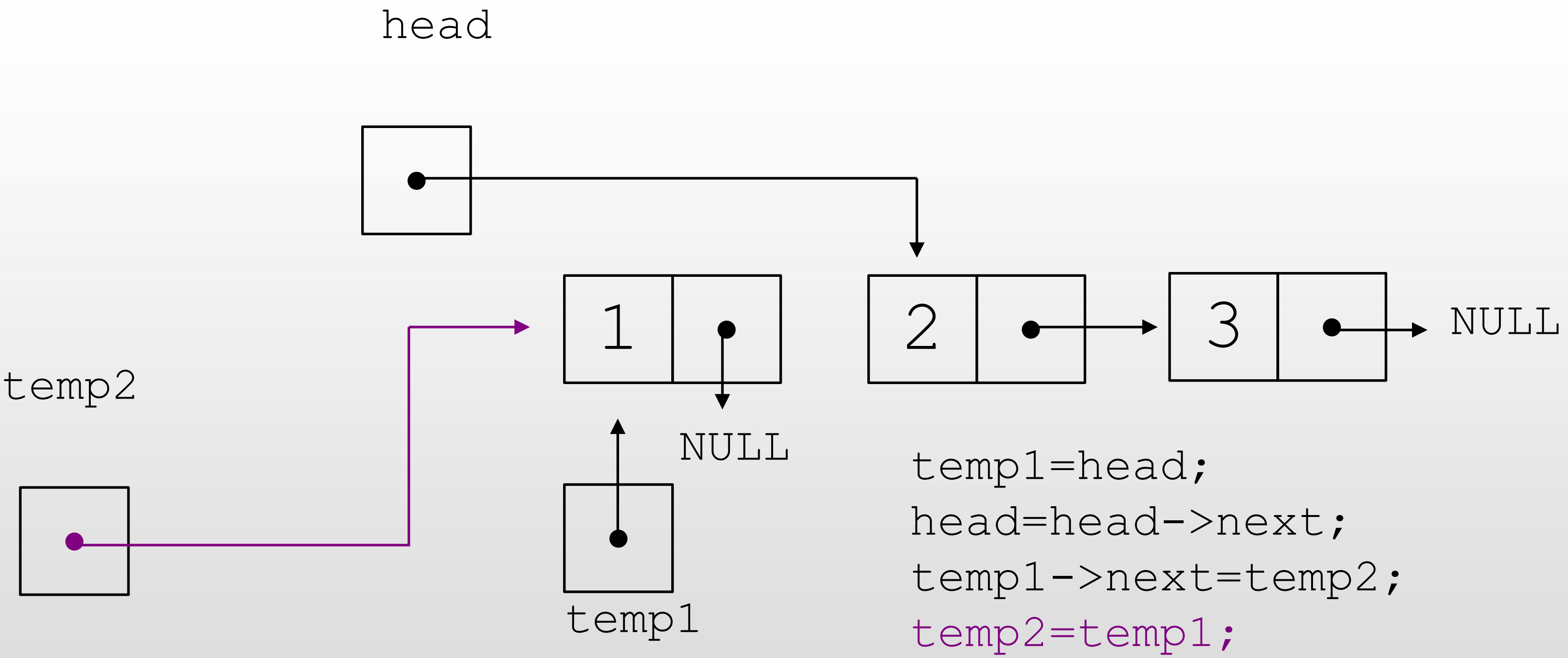
```
temp1=head;
head=head->next;
temp1->next=temp2;
temp2=temp1;
```

Invert a singly-linked list



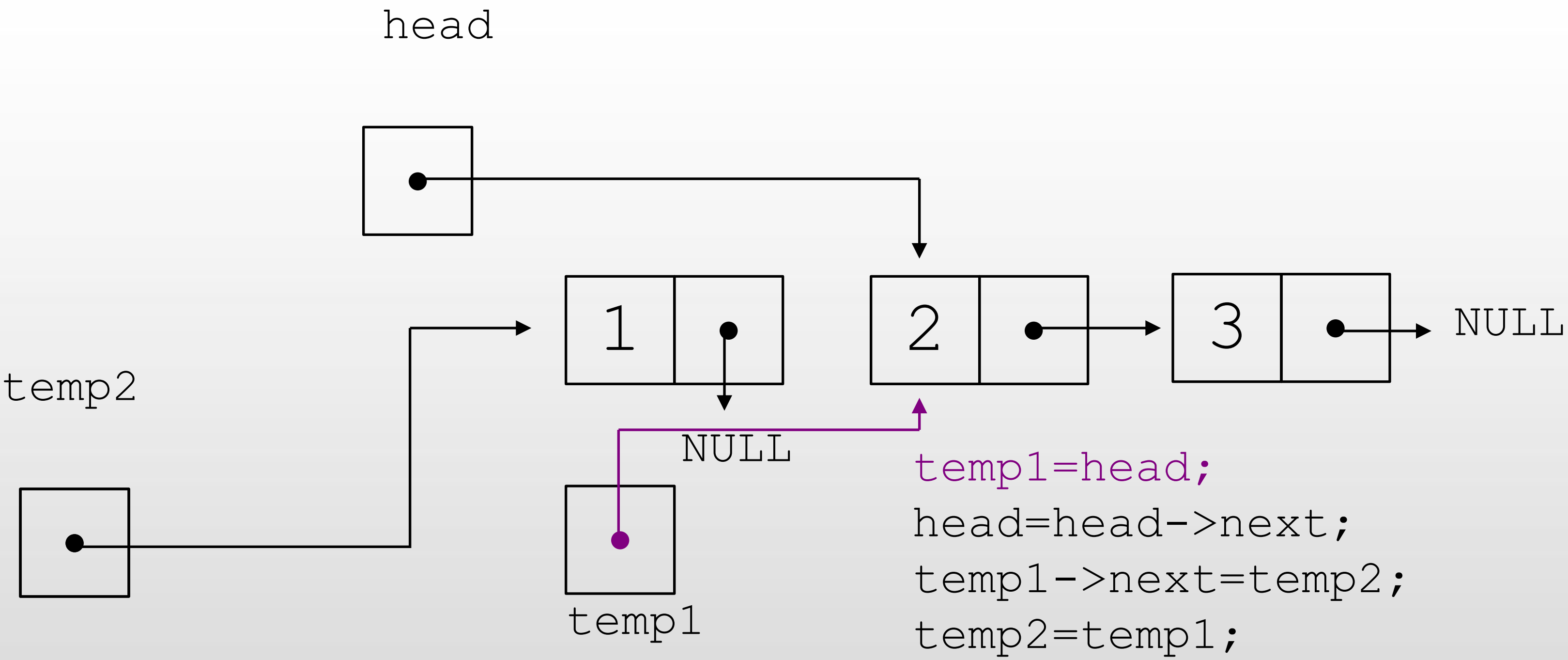
```
temp1=head;
head=head->next;
temp1->next=temp2;
temp2=temp1;
```

Invert a singly-linked list



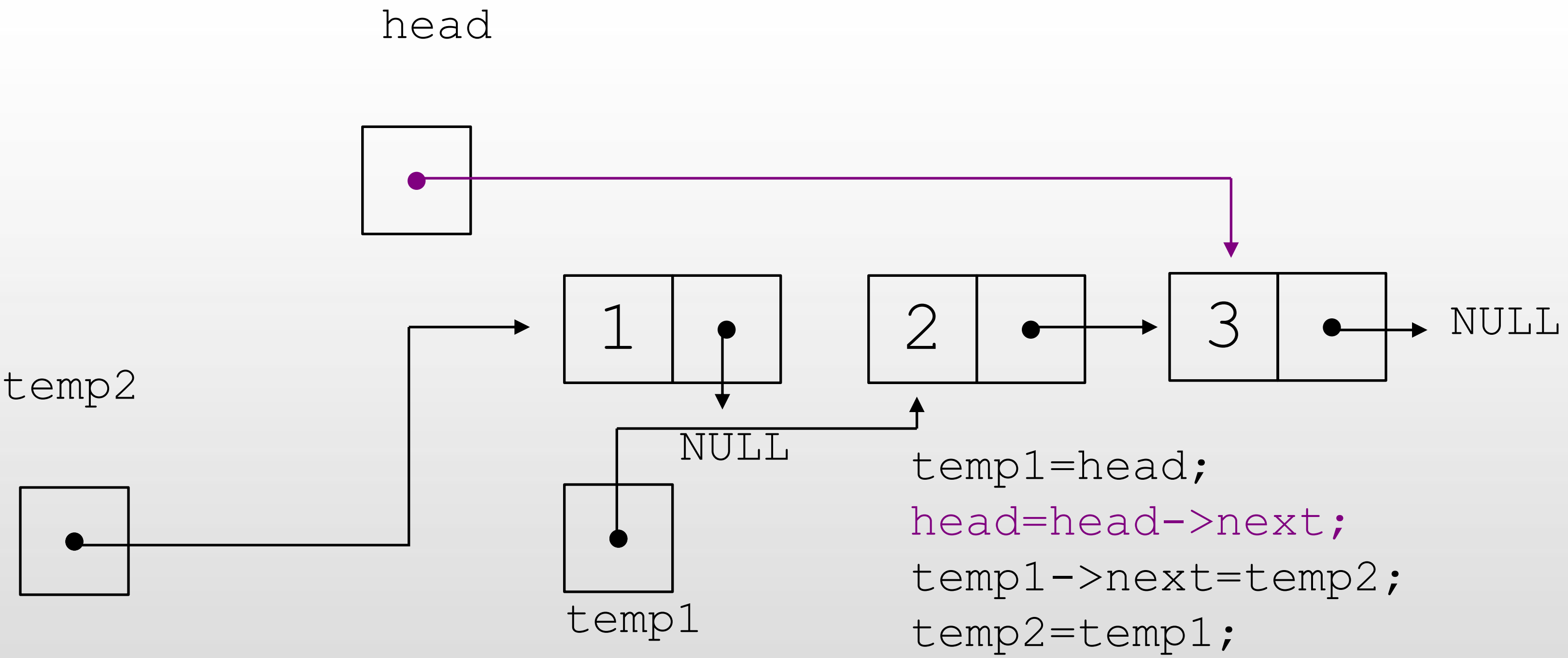
```
temp1=head;
head=head->next;
temp1->next=temp2;
temp2=temp1;
```

Invert a singly-linked list



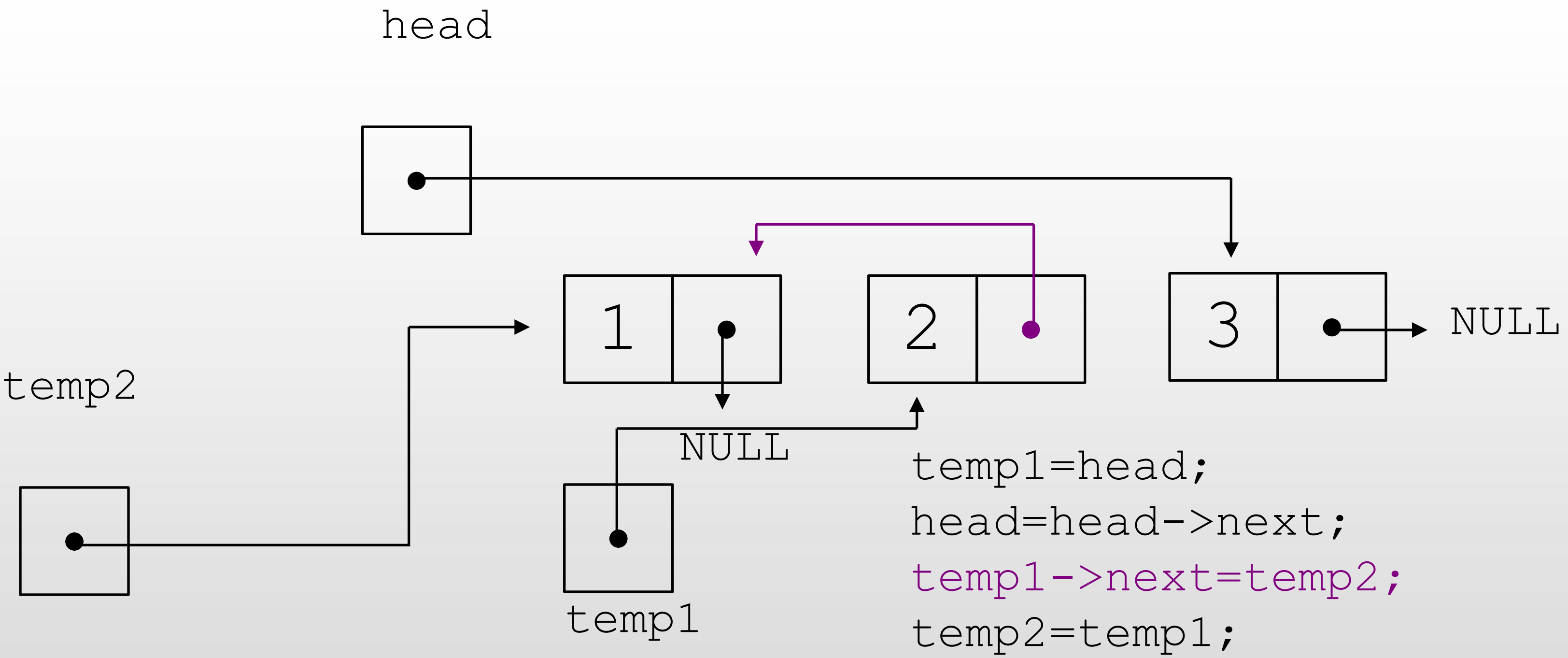
```
temp1=head;
head=head->next;
temp1->next=temp2;
temp2=temp1;
```

Invert a singly-linked list



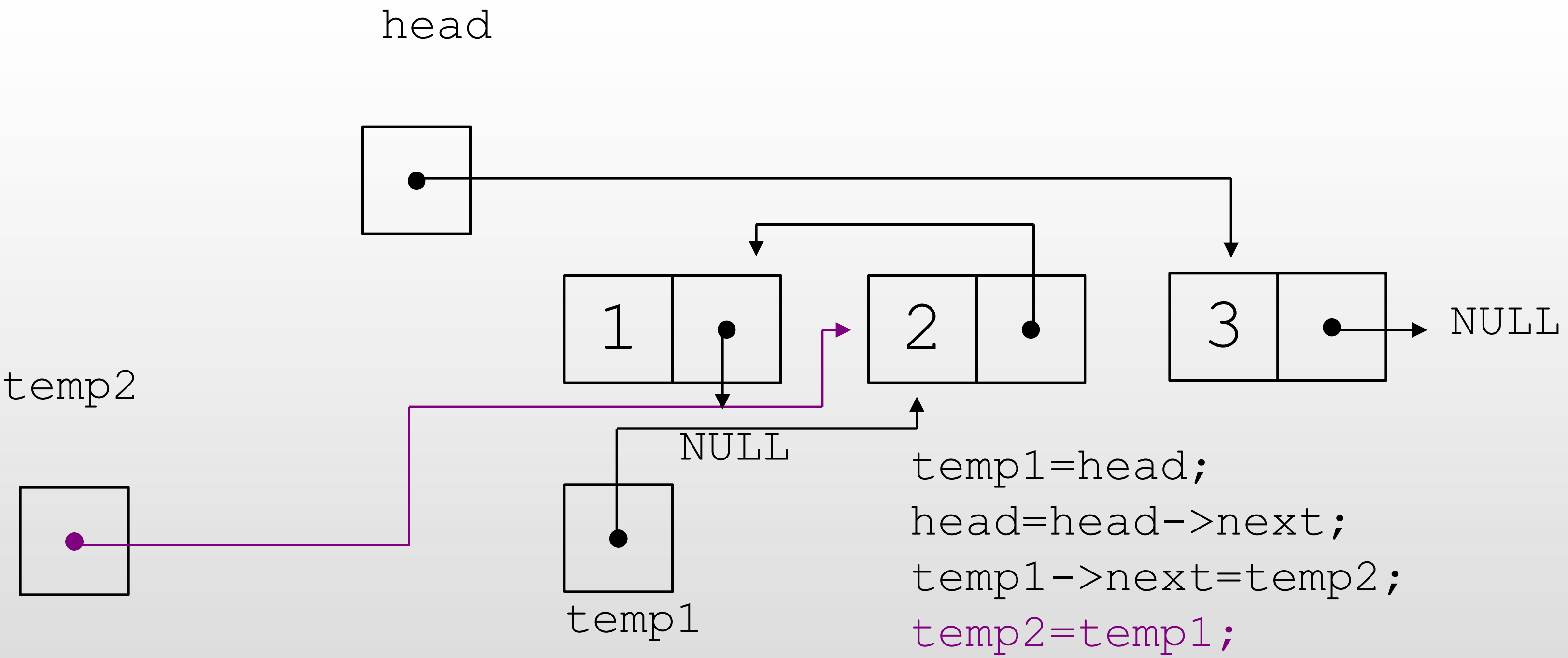
```
temp1=head;
head=head->next;
temp1->next=temp2;
temp2=temp1;
```

Invert a singly-linked list



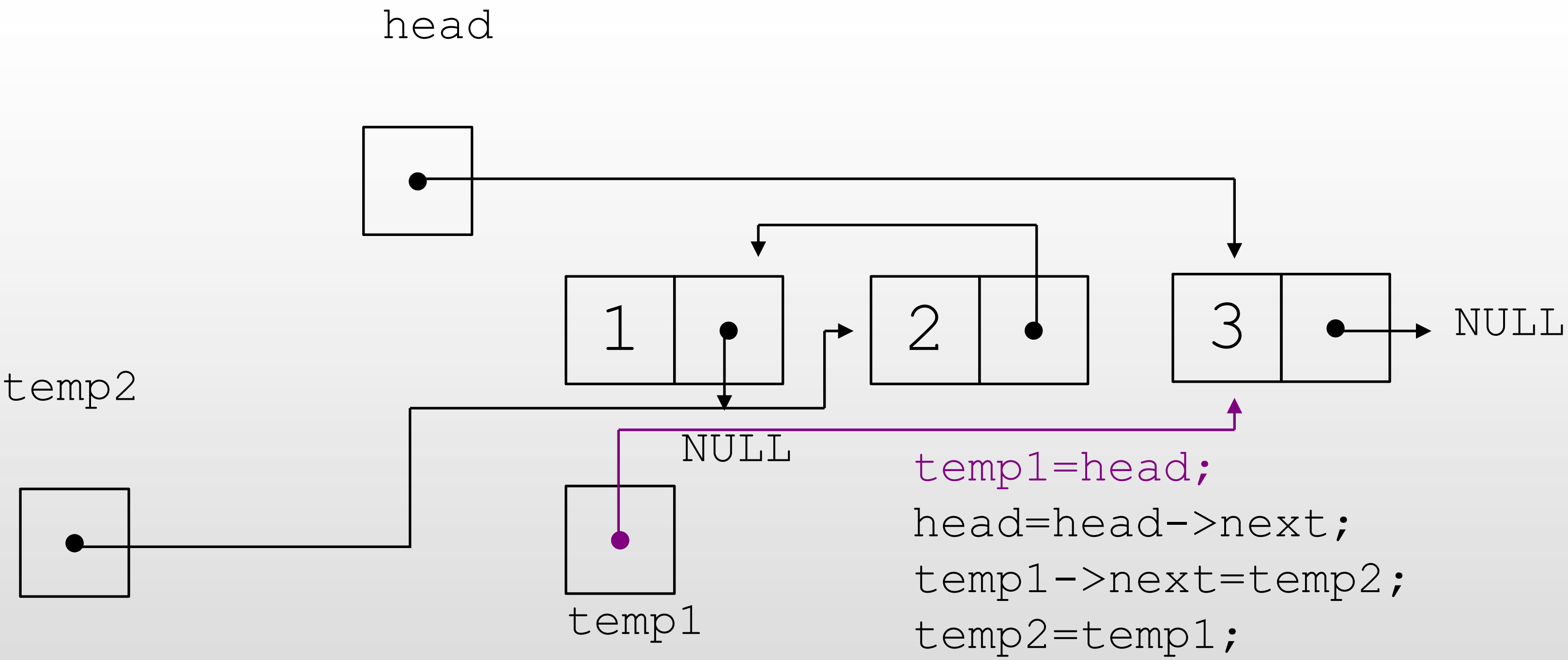
```
temp1=head;
head=head->next;
temp1->next=temp2;
temp2=temp1;
```

Invert a singly-linked list



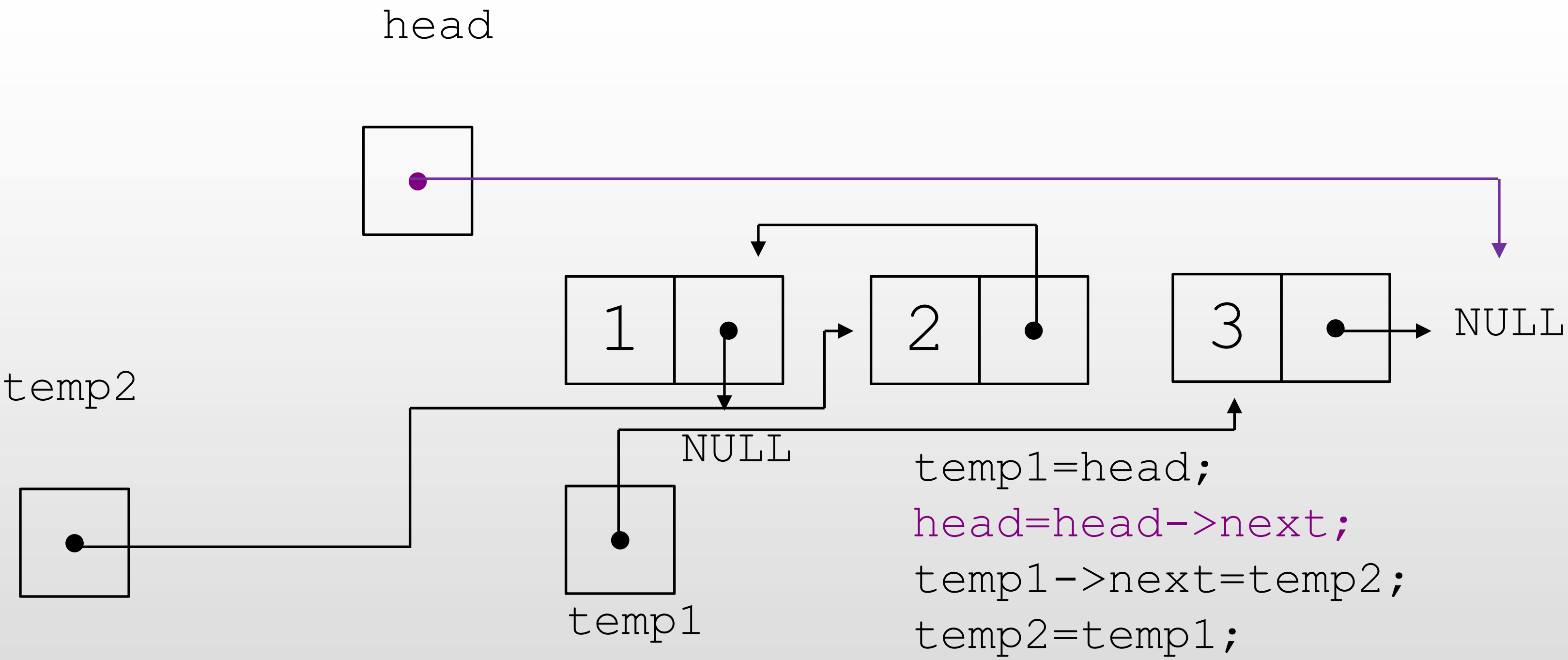
```
temp1=head;
head=head->next;
temp1->next=temp2;
temp2=temp1;
```

Invert a singly-linked list



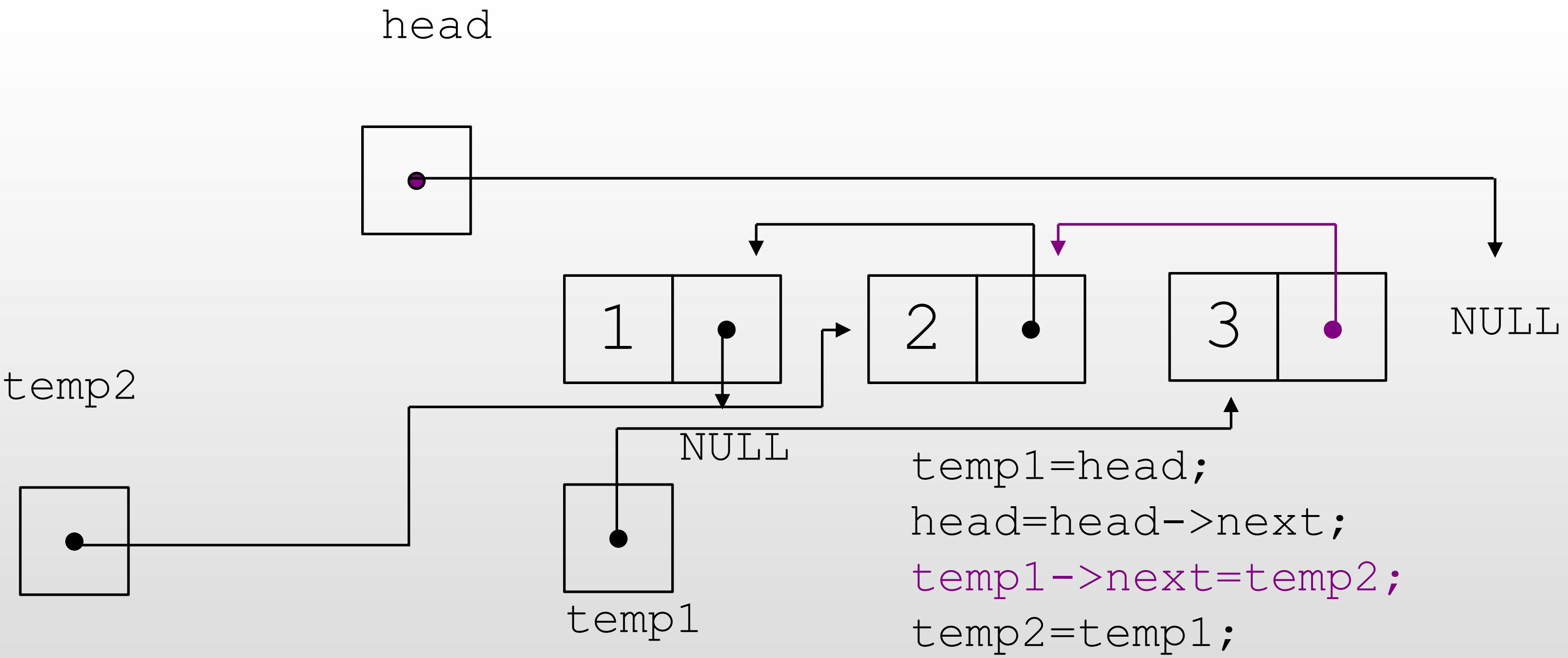
```
temp1=head;
head=head->next;
temp1->next=temp2;
temp2=temp1;
```

Invert a singly-linked list



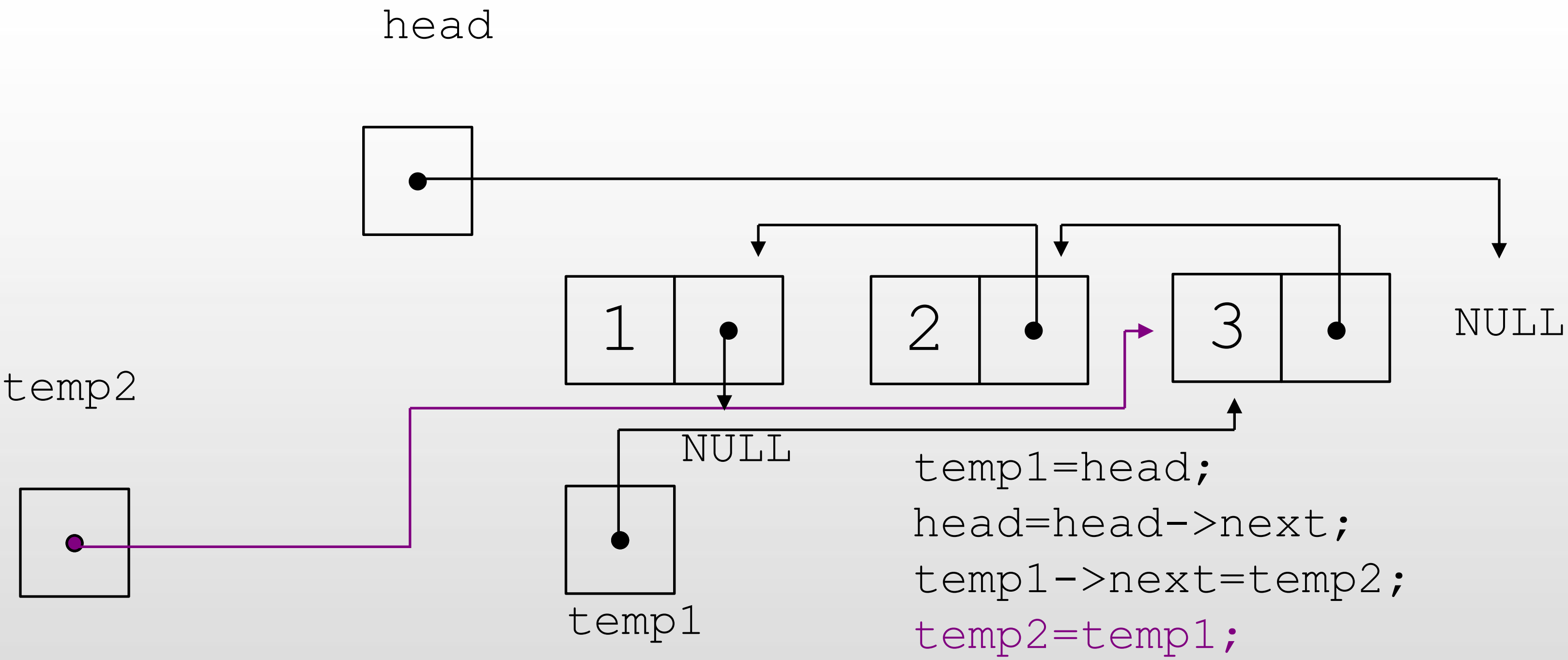
```
temp1=head;
head=head->next;
temp1->next=temp2;
temp2=temp1;
```

Invert a singly-linked list



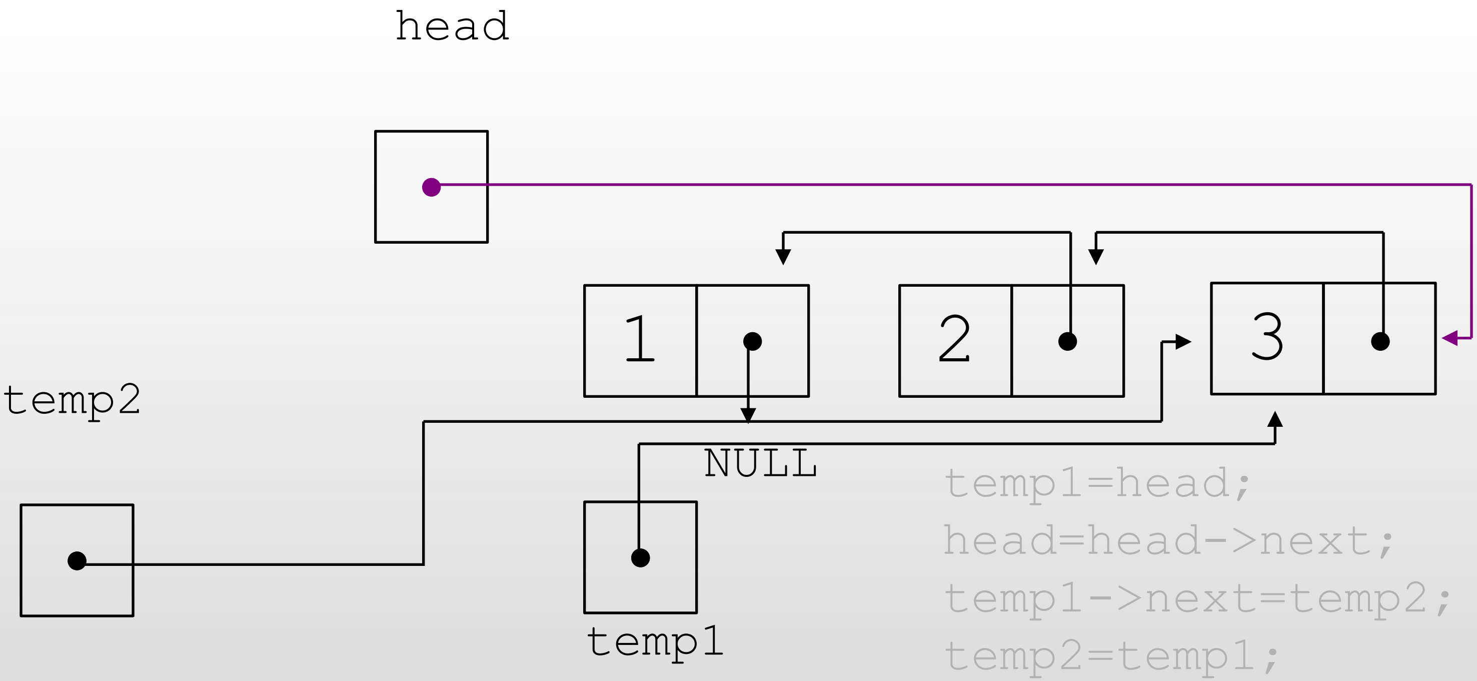
```
temp1=head;
head=head->next;
temp1->next=temp2;
temp2=temp1;
```

Invert a singly-linked list



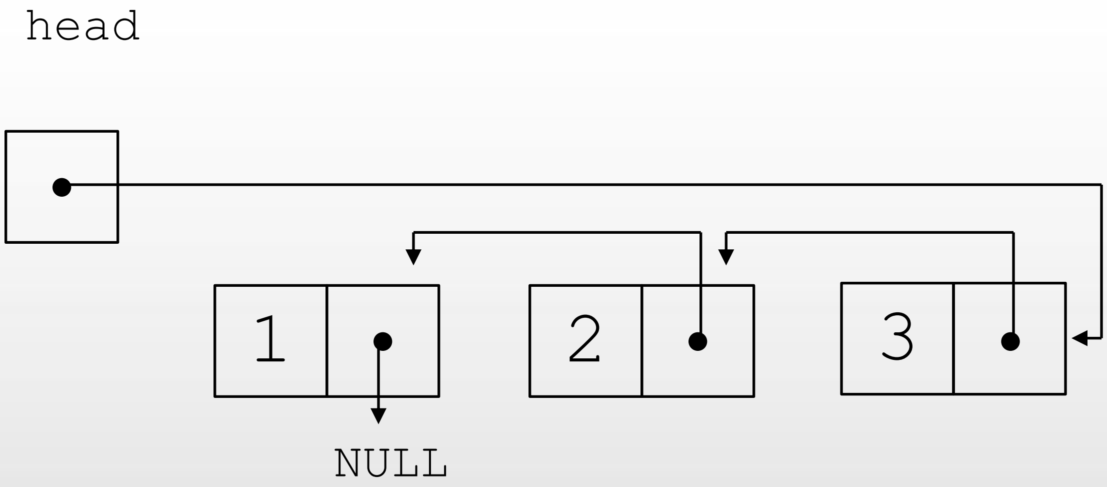
```
temp1=head;
head=head->next;
temp1->next=temp2;
temp2=temp1;
```

Invert a singly-linked list



```
temp1=head;
head=head->next;
temp1->next=temp2;
temp2=temp1;

head=temp2;
```

Invert a singly-linked list



Print a singly linked list

```
void print_list(list *head){
    list *temp;

    temp=head;
    while(______________){
        printf(“%i”,temp->num);
        __________________________
    }
}
```

Print a singly linked list

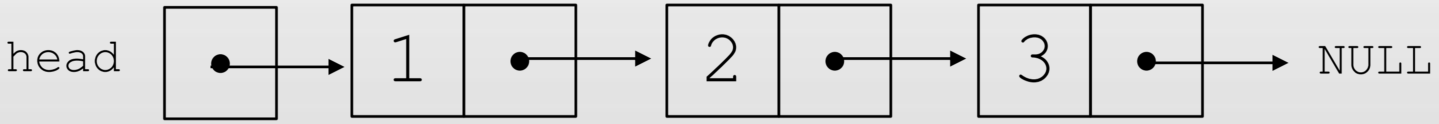
```
void print_list(list *head){
    list *temp;

    temp=head;
    while(temp!=NULL){
        printf(“%i”,temp->num);
        temp=temp->next;
    }
}
```

Circular linked list

- similar in structure to linear linked list
- difference: the next node pointer of the last node points to the first node instead of NULL

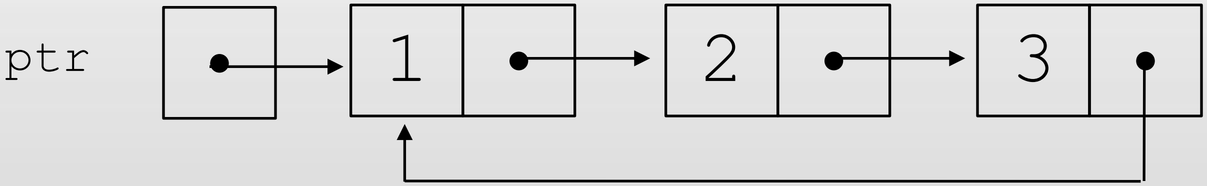
Singly linked list template

```
typedef struct node{
    int num;
    struct node *next;
}list;

list *head;
```



Circular linked list

```
typedef struct node{
    int num;
    struct node *next;
}list;

list *ptr;
```



Recall: Build list

```
    head = (list *)malloc(sizeof(list));
    head->num = 1;
    head->next = NULL;
    p = head;
    while(there is data){
        temp = (list *)malloc(sizeof(list));
        temp->num = data;
        temp->next = NULL;
        p->next = temp;
        p = p->next;
    }
```

Creating Circular linked list

```
ptr = (list *)malloc(sizeof(list));
ptr->num = 1;
ptr->next = NULL;
p = ptr;
while(there is data){
    temp = (list *)malloc(sizeof(list));
    temp->num = data;
    temp->next = NULL;
    p->next = temp;
    p = p->next;
}
```

Creating Circular linked list

```
ptr = (list *)malloc(sizeof(list));
ptr->num = 1;
ptr->next = ptr;
p = ptr;
while(there is data){
    temp = (list *)malloc(sizeof(list));
    temp->num = data;
    temp->next = NULL;
    p->next = temp;
    p = p->next;
}
```

Creating Circular linked list

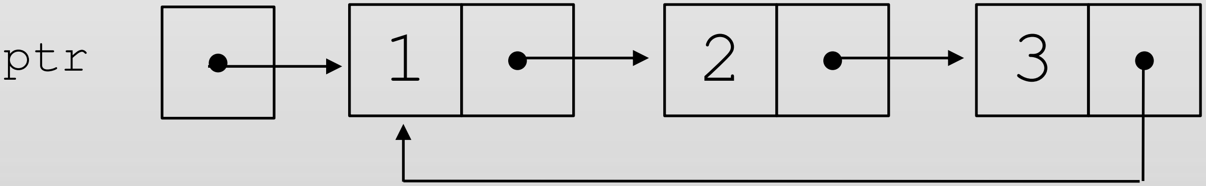
```
ptr = (list *)malloc(sizeof(list));
ptr->num = 1;
ptr->next = ptr;
p = ptr;
while(there is data){
    temp = (list *)malloc(sizeof(list));
    temp->num = data;
    temp->next = ptr;
    p->next = temp;
    p = p->next;
}
```

Print a singly linked list

```
void print_list(list *head){
    list *temp;

    temp=head;
    while(temp!=NULL){
        printf(“%i”,temp->num);
        temp=temp->next;
    }
}
```

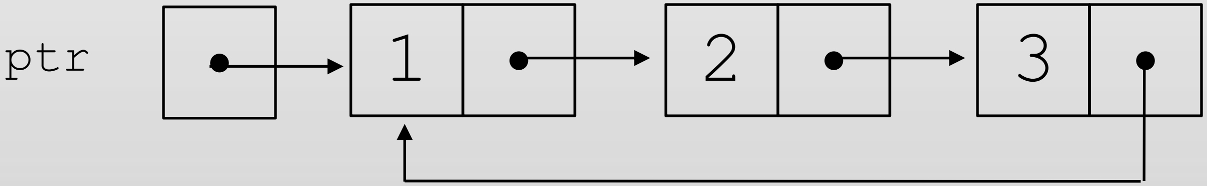
Print a circular linked list

```
void print_list(list *ptr){
    list *temp;

    temp=ptr;
    while(temp!=NULL){
        printf("%i",temp->num);
        temp=temp->next;
    }
}
```



Print a circular linked list

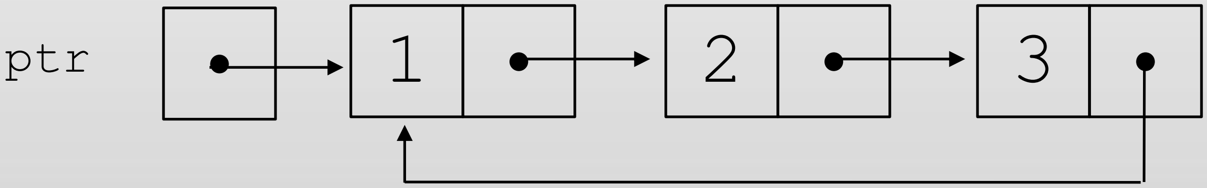
```
void print_list(list *ptr){
    list *temp;

    temp=ptr;
    while(temp!=ptr){  ?
        printf(“%i”,temp->num);
        temp=temp->next;
    }
}
```



Print a circular linked list

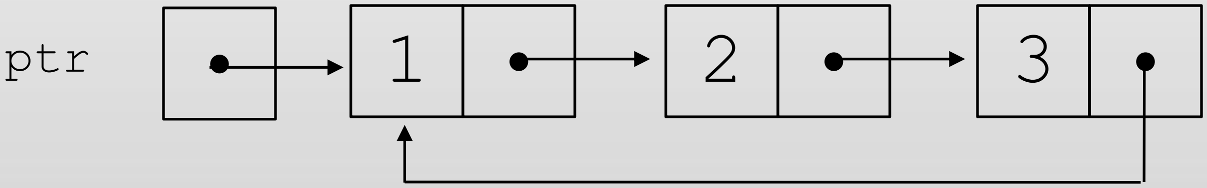
```
void print_list(list *ptr){
    list *temp;

    temp=ptr;
    while(temp->next!=ptr){
        printf("%i",temp->num);
        temp=temp->next;
    }
}
```

?



Print a circular linked list

```
void print_list(list *ptr){
    list *temp;

    temp=ptr;
    do {
        printf("%i",temp->num);
        temp=temp->next;
    } while(temp!=ptr);
}
```

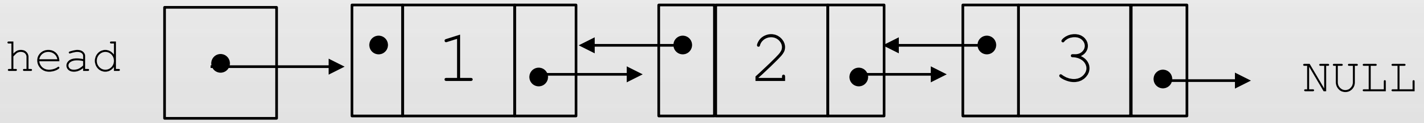


Doubly Linked list

- each node has two pointers, one pointing to the previous node in the list, another pointing to the next node
- allows forward and backward movements

Doubly linked list template

```
typedef struct node{
    int num;
    struct node *prev, *next;
}list;

list *head;
```



Recall: Build list

```
head = (list *)malloc(sizeof(list));
head->num = 1;
head->next = NULL;
p = head;
while(there is data){
    temp = (list *)malloc(sizeof(list));
    temp->num = data;
    temp->next = NULL;
    p->next = temp;
    p = p->next;
}
```

Creating Doubly linked list

```
head = (list *)malloc(sizeof(list));
head->num = 1;
head->prev = NULL;
head->next = NULL;
p = head;
while(there is data){
    temp = (list *)malloc(sizeof(list));
    temp->num = data;
    temp->prev = p;
    temp->next = NULL;
    p->next = temp;
    p = p->next;
}
```

Print a doubly linked list

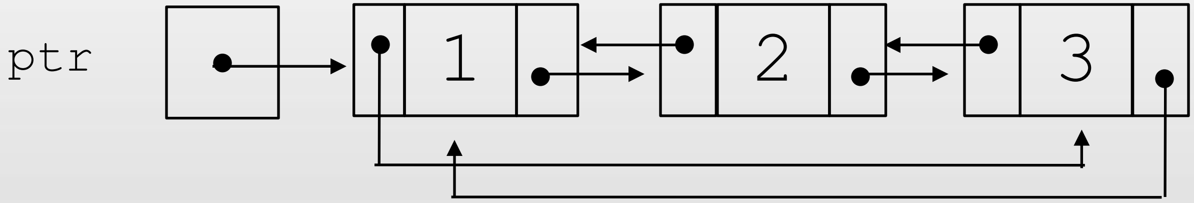
```
void print_list(list *head){
    list *temp;

    temp=head;
    while(temp!=NULL){
        printf(“%i”,temp->num);
        temp=temp->next;
    }
}
```

Print inverse - doubly linked list

```
void print_list(list *head){
    list *temp;

    temp=head;
    while(temp->next!=NULL)
        temp=temp->next;

    while(temp!=NULL){
        printf(“%i”,temp->num);
        temp=temp->prev;
    }
}
```

Circular Doubly Linked list

- right pointer of the last node points to the first node and the left pointer of the first node points to the last node



Creating Circular Doubly linked list

```
ptr = (list *)malloc(sizeof(list));
ptr->num = 1;
ptr->prev = NULL;
ptr->next = NULL;
p = ptr;
while(there is data){
    temp = (list *)malloc(sizeof(list));
    temp->num = data;
    temp->prev = p;
    temp->next = NULL;
    p->next = temp;
    p = p->next;
}
```

Creating Circular Doubly linked list

```
ptr = (list *)malloc(sizeof(list));
ptr->num = 1;
ptr->prev = ptr;
ptr->next = ptr;
p = ptr;
while(there is data){
    temp = (list *)malloc(sizeof(list));
    temp->num = data;
    temp->prev = p;
    temp->next = ptr;
    ptr->prev = temp;
    p->next = temp;
    p = p->next;
}
```

Print a circular doubly linked list

```
void print_list(list *ptr){
    list *temp;

    temp=ptr;
    do {
        printf(“%i”,temp->num);
        temp=temp->next;
    } while(temp!=ptr);
}
```