# Suggestions for Assignment 2 (Parts 1-2)

# Outline

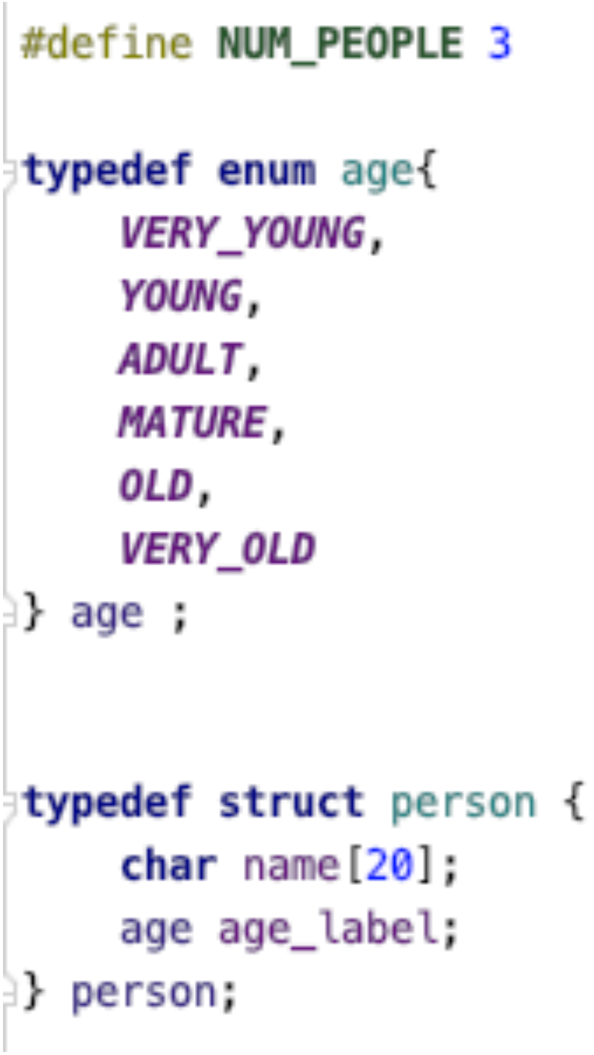
- **How to initialize players?**
  - Example program about how to initialize objects that are complex data structures.

- **How to merge stacks?**
  - Example about how to merge stacks (with pseudo code).

# Initialize Objects that are Complex Data Structures

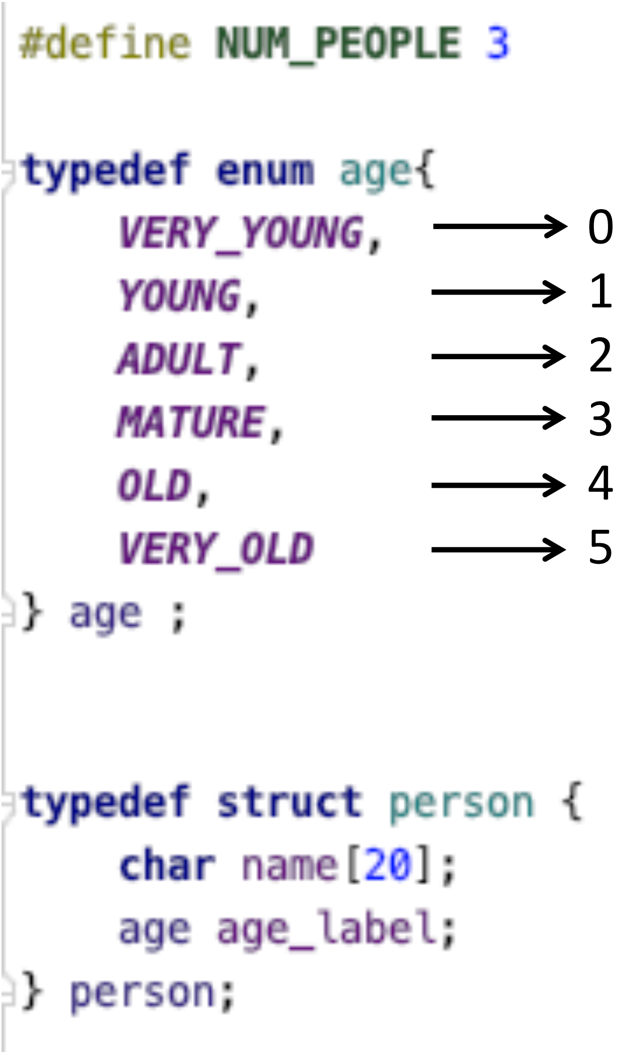
(code provided in *Week 10 >> init_struct.c*)

# Imagine a data structure representing a Person



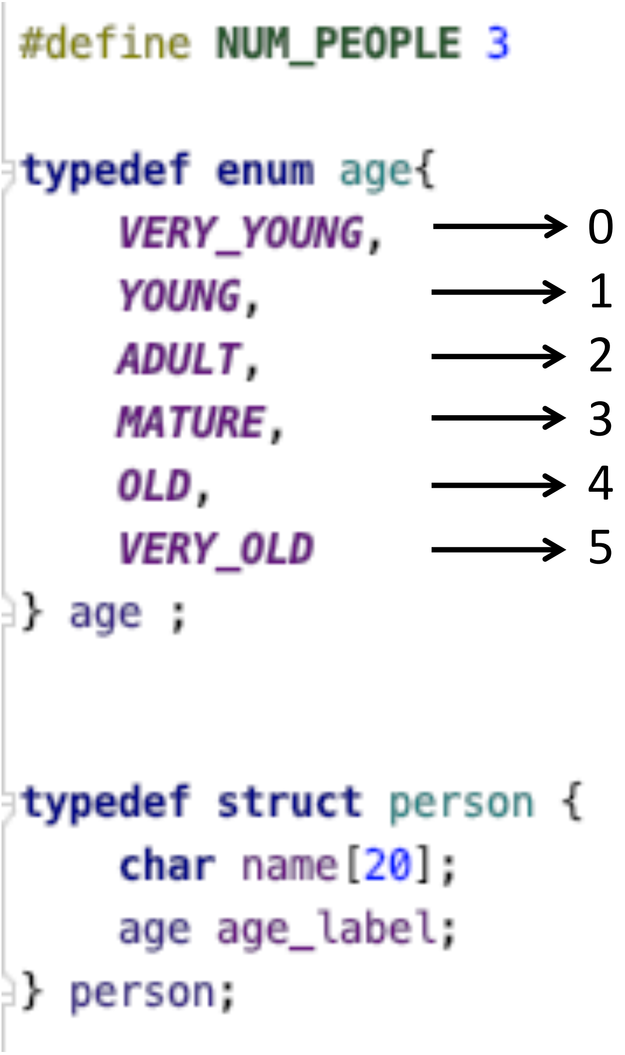
```c
#define NUM_PEOPLE 3

typedef enum age{
    VERY_YOUNG,
    YOUNG,
    ADULT,
    MATURE,
    OLD,
    VERY_OLD
} age ;


typedef struct person {
    char name[20];
    age age_label;
} person;
```

# Imagine a data structure representing a Person

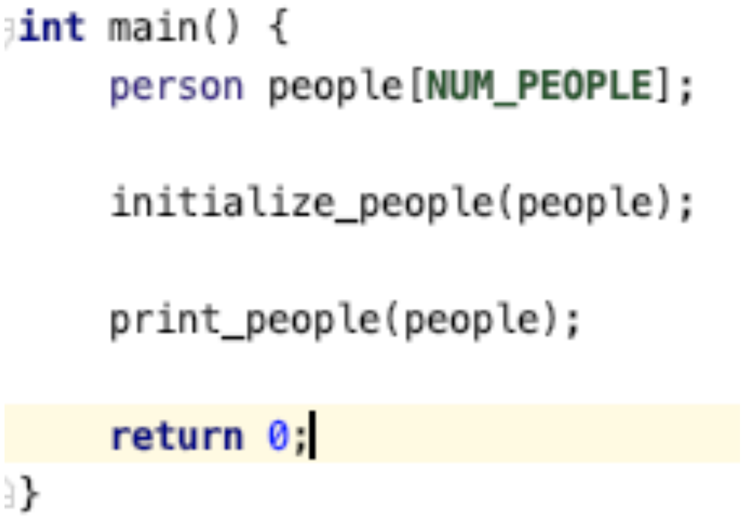


```
#define NUM_PEOPLE 3

typedef enum age{
    VERY_YOUNG,    ⟶ 0
    YOUNG,         ⟶ 1
    ADULT,         ⟶ 2
    MATURE,        ⟶ 3
    OLD,           ⟶ 4
    VERY_OLD       ⟶ 5
} age ;


typedef struct person {
    char name[20];
    age age_label;
} person;
```

# Imagine a data structure representing a Person

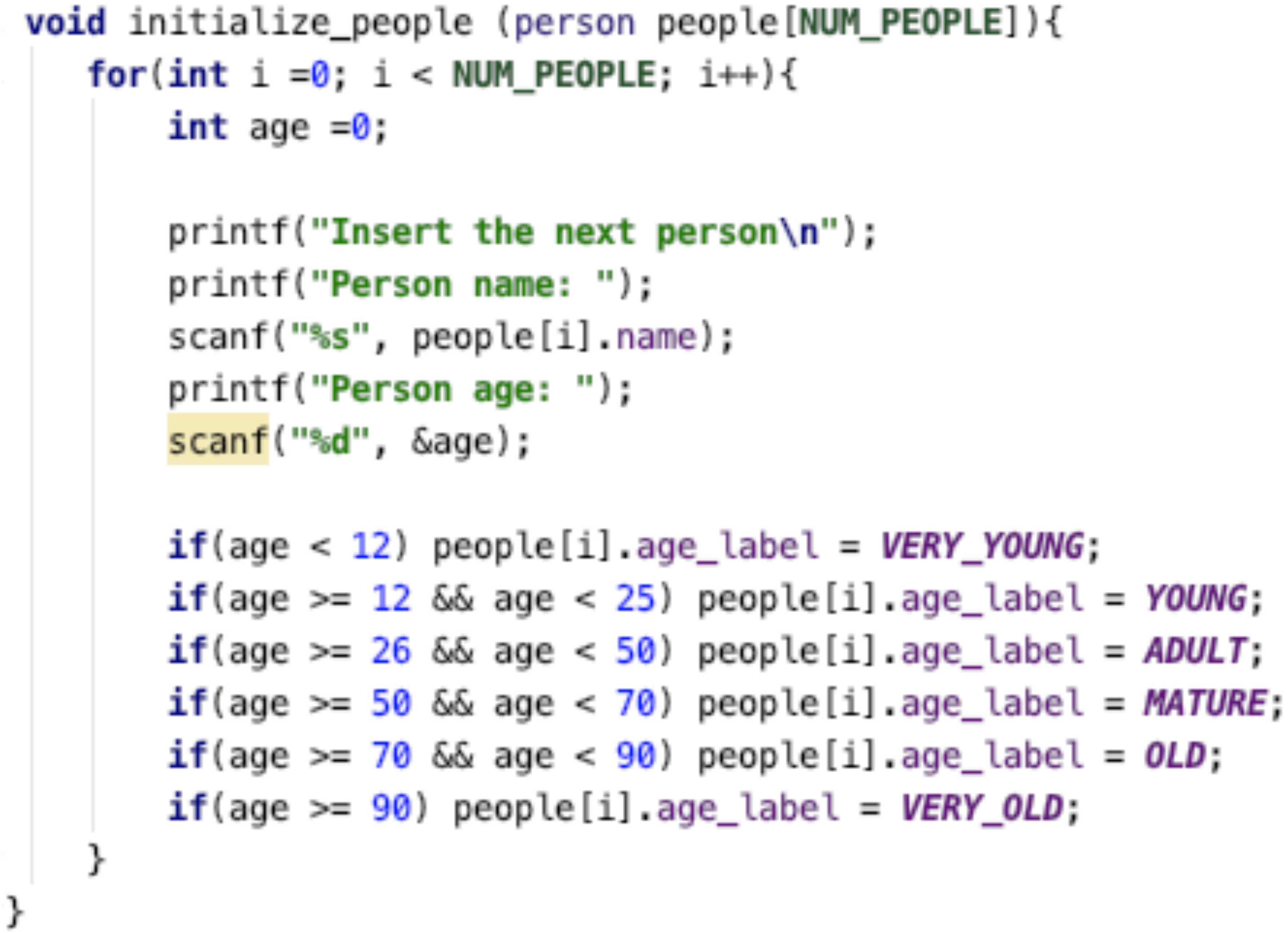


```
#define NUM_PEOPLE 3

typedef enum age{
    VERY_YOUNG,     ⟶ 0
    YOUNG,          ⟶ 1
    ADULT,          ⟶ 2
    MATURE,         ⟶ 3
    OLD,            ⟶ 4
    VERY_OLD        ⟶ 5
} age ;


typedef struct person {
    char name[20];
    age age_label;
} person;
```



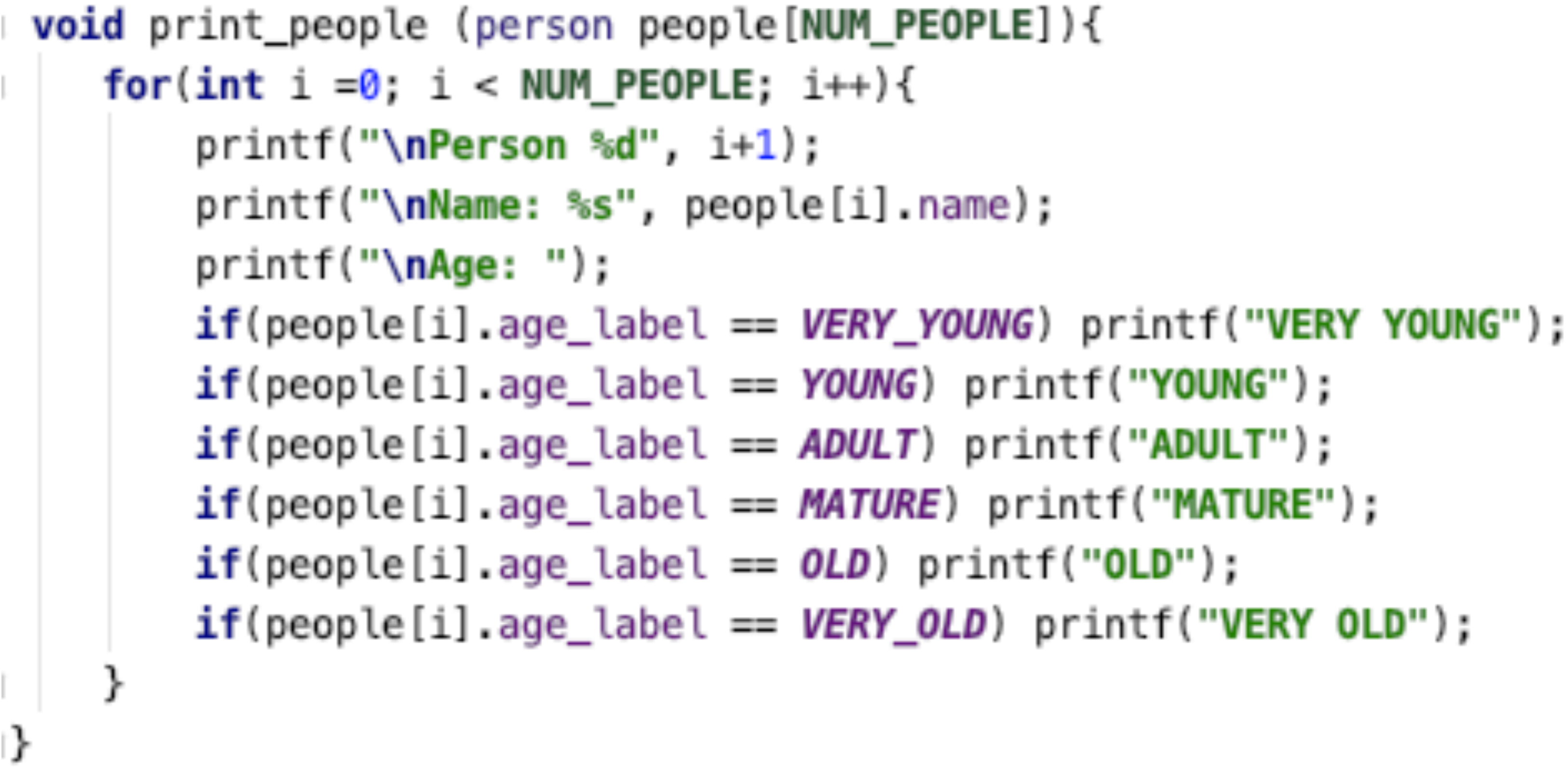
```
int main() {
    person people[NUM_PEOPLE];

    initialize_people(people);

    print_people(people);

    return 0;
}
```

# Initialize People



```
void initialize_people (person people[NUM_PEOPLE]){
    for(int i =0; i < NUM_PEOPLE; i++){
        int age =0;

        printf("Insert the next person\n");
        printf("Person name: ");
        scanf("%s", people[i].name);
        printf("Person age: ");
        scanf("%d", &age);

        if(age < 12) people[i].age_label = VERY_YOUNG;
        if(age >= 12 && age < 25) people[i].age_label = YOUNG;
        if(age >= 26 && age < 50) people[i].age_label = ADULT;
        if(age >= 50 && age < 70) people[i].age_label = MATURE;
        if(age >= 70 && age < 90) people[i].age_label = OLD;
        if(age >= 90) people[i].age_label = VERY_OLD;
    }
}
```

# Print People



```
void print_people (person people[NUM_PEOPLE]){
    for(int i =0; i < NUM_PEOPLE; i++){
        printf("\nPerson %d", i+1);
        printf("\nName: %s", people[i].name);
        printf("\nAge: ");
        if(people[i].age_label == VERY_YOUNG) printf("VERY YOUNG");
        if(people[i].age_label == YOUNG) printf("YOUNG");
        if(people[i].age_label == ADULT) printf("ADULT");
        if(people[i].age_label == MATURE) printf("MATURE");
        if(people[i].age_label == OLD) printf("OLD");
        if(people[i].age_label == VERY_OLD) printf("VERY OLD");
    }
}
```

# How to merge stacks?

# Stack

An ordered collection of items where the addition of new items and the removal of existing items always takes places at the same end (the top).

# Stack

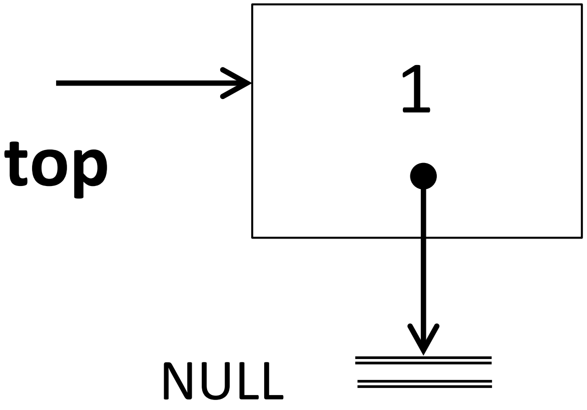
An ordered collection of items where the addition of new items and the removal of existing items always takes places at the same end (the top).

- **LIFO (last-in first-out) ordering principle**: the most recently added item is in the top position and it should be removed first.

# Stack

An ordered collection of items where the addition of new items and the removal of existing items always takes places at the same end (the top).

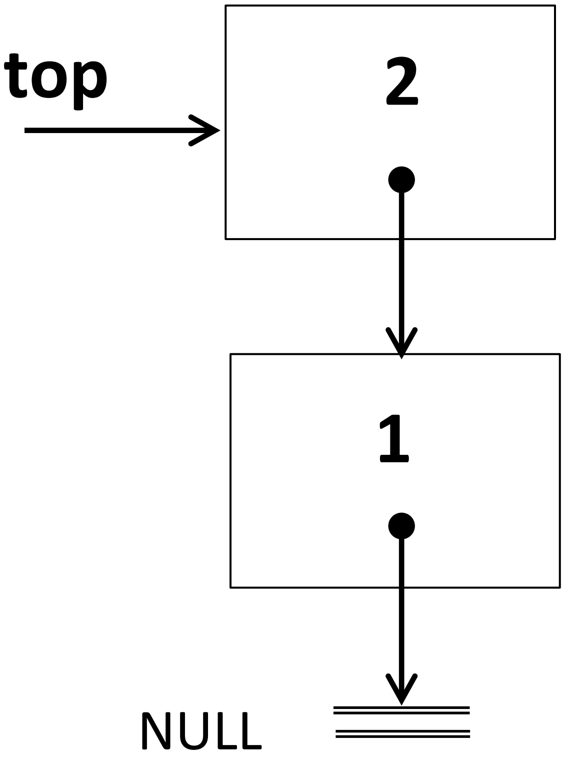
- **LIFO (last-in first-out) ordering principle**: the most recently added item is in the top position and it is to be removed first



# Stack

An ordered collection of items where the addition of new items and the removal of existing items always takes places at the same end (the top).

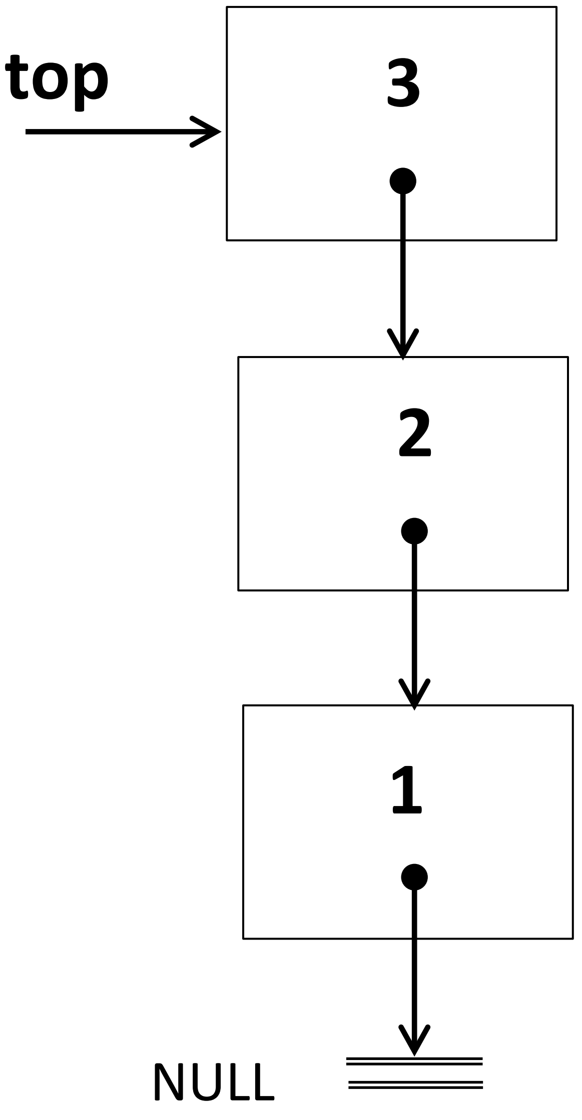
- **LIFO (last-in first-out) ordering principle**: the most recently added item is in the top position and it is to be removed first



# Stack

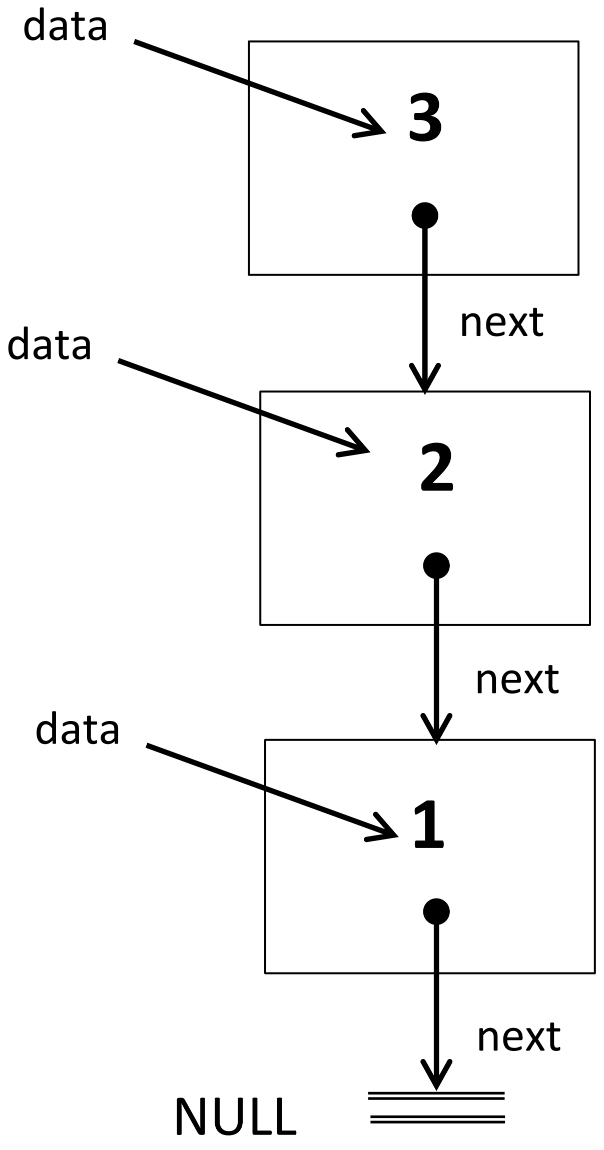
An ordered collection of items where the addition of new items and the removal of existing items always takes places at the same end (the top).

- **LIFO (last-in first-out) ordering principle:** the most recently added item is in the top position and it is to be removed first



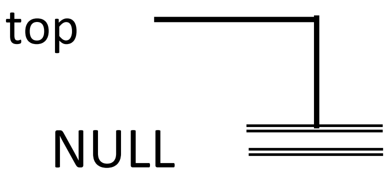
# Structure Members

```
struct stack_elem{
    int data;
    struct stack_elem *next;
} stack;
```



# Example stack

```
int main(int argc, char** argv) {

    struct stack_elem *top = NULL;
    struct stack_elem *curr = NULL;

    top = push(1, top);
    printf("Stack Data: %d\n", top->data);

    top = push(2, top);
    printf("Stack Data: %d\n", top->data);

    top = push(3, top);
    printf("Stack Data: %d\n", top->data);

    top = pop(top);
    top= pop(top);
    top= pop(top);

}
```

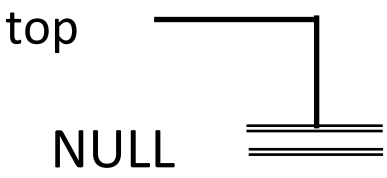


# Example stack

```
struct stack_elem * push(int value, struct stack_elem *top){
   struct stack_elem *curr = top;
   top = malloc(sizeof(stack));
   top->data = value;
   top->next = curr;
   return top;
}
```

main.c

```
 top = push(1, top);
 printf("Stack Data: %d\n", top->data);

top = push(2, top);
 printf("Stack Data: %d\n", top->data);

 top = push(3, top);
 printf("Stack Data: %d\n", top->data);
```

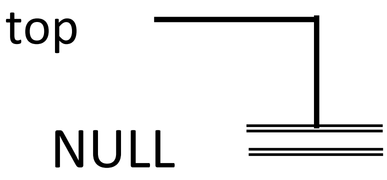


# Example stack

```
struct stack_elem * push(int value, struct stack_elem *top){
   struct stack_elem *curr = top;
   top = malloc(sizeof(stack));
   top->data = value;
   top->next = curr;
   return top;
}
```

main.c

```
top = push(1, top);
printf("Stack Data: %d\n", top->data);

top = push(2, top);
printf("Stack Data: %d\n", top->data);

top = push(3, top);
printf("Stack Data: %d\n", top->data);
```

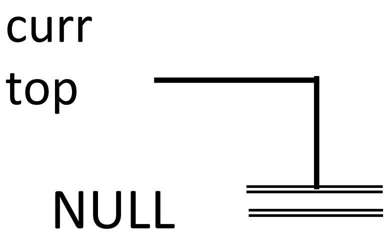


# Example stack

```
struct stack_elem * push(int value, struct stack_elem *top){
    struct stack_elem *curr = top;
    top = malloc(sizeof(stack));
    top->data = value;
    top->next = curr;
    return top;
}
```

main.c

```
 top = push(1, top);
 printf("Stack Data: %d\n", top->data);

top = push(2, top);
 printf("Stack Data: %d\n", top->data);

 top = push(3, top);
 printf("Stack Data: %d\n", top->data);
```



curr

top

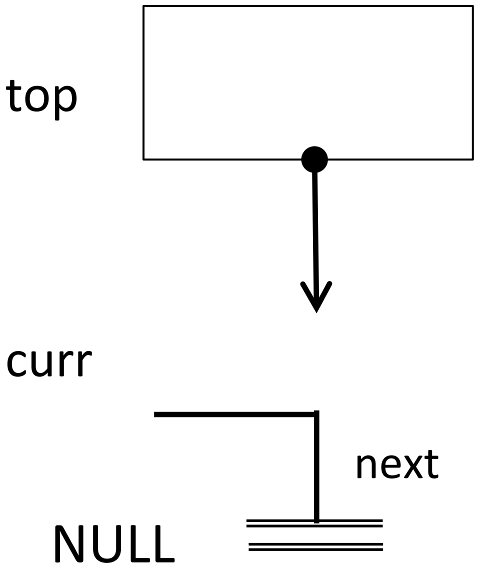
NULL

# Example stack

```
struct stack_elem * push(int value, struct stack_elem *top){
    struct stack_elem *curr = top;
    top = malloc(sizeof(stack));
    top->data = value;
    top->next = curr;
    return top;
}
```

main.c

```
top = push(1, top);
printf("Stack Data: %d\n", top->data);

top = push(2, top);
printf("Stack Data: %d\n", top->data);

top = push(3, top);
printf("Stack Data: %d\n", top->data);
```

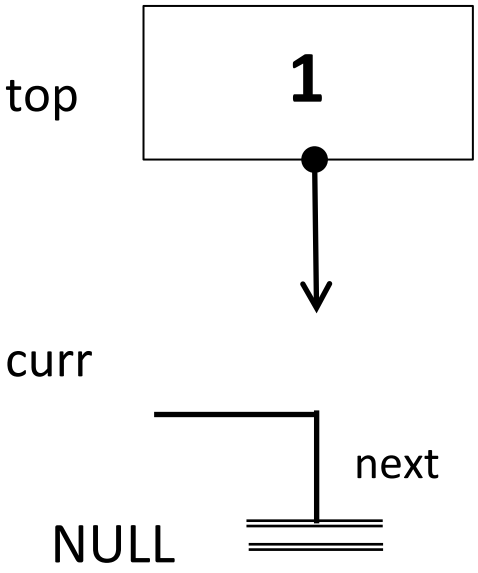


# Example stack

```
struct stack_elem * push(int value, struct stack_elem *top){
    struct stack_elem *curr = top;
    top = malloc(sizeof(stack));
    top->data = value;
    top->next = curr;
    return top;
}
```

main.c

```
 top = push(1, top);
 printf("Stack Data: %d\n", top->data);

top = push(2, top);
 printf("Stack Data: %d\n", top->data);

 top = push(3, top);
 printf("Stack Data: %d\n", top->data);
```

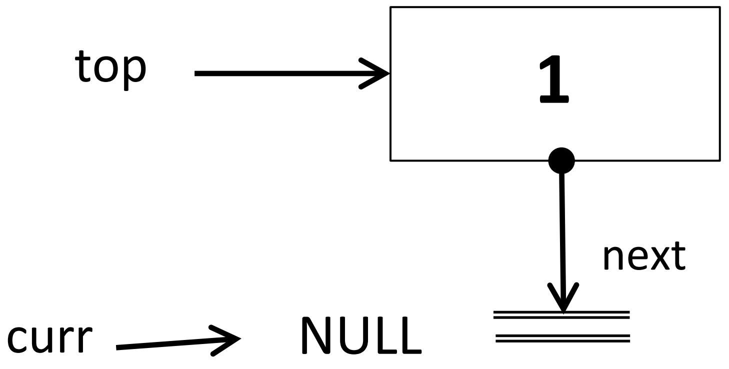


# Example stack

```
struct stack_elem * push(int value, struct stack_elem *top){
   struct stack_elem *curr = top;
   top = malloc(sizeof(stack));
   top->data = value;
   top->next = curr;
   return top;
}
```

main.c

```
 top = push(1, top);
 printf("Stack Data: %d\n", top->data);

top = push(2, top);
 printf("Stack Data: %d\n", top->data);

 top = push(3, top);
 printf("Stack Data: %d\n", top->data);
```

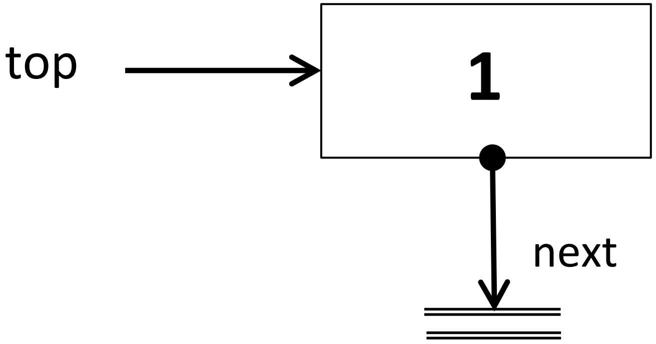


# Example stack

```
struct stack_elem * push(int value, struct stack_elem *top){
    struct stack_elem *curr = top;
    top = malloc(sizeof(stack));
    top->data = value;
    top->next = curr;
    return top;
}
```

main.c

```
top = push(1, top);
printf("Stack Data: %d\n", top->data);

top = push(2, top);
printf("Stack Data: %d\n", top->data);

top = push(3, top);
printf("Stack Data: %d\n", top->data);
```

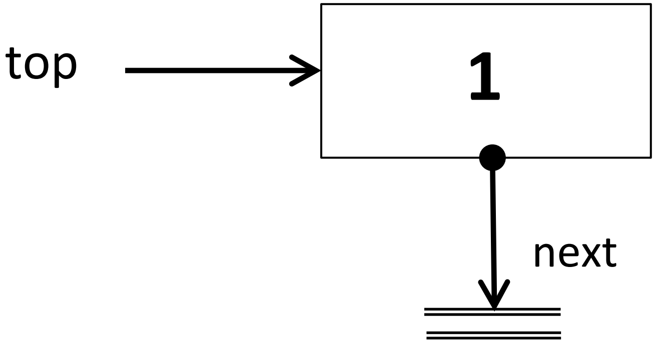


# Example stack

```
struct stack_elem * push(int value, struct stack_elem *top){
    struct stack_elem *curr = top;
    top = malloc(sizeof(stack));
    top->data = value;
    top->next = curr;
    return top;
}
```

main.c

```
top = push(1, top);
printf("Stack Data: %d\n", top->data);

top = push(2, top);
printf("Stack Data: %d\n", top->data);

top = push(3, top);
printf("Stack Data: %d\n", top->data);
```

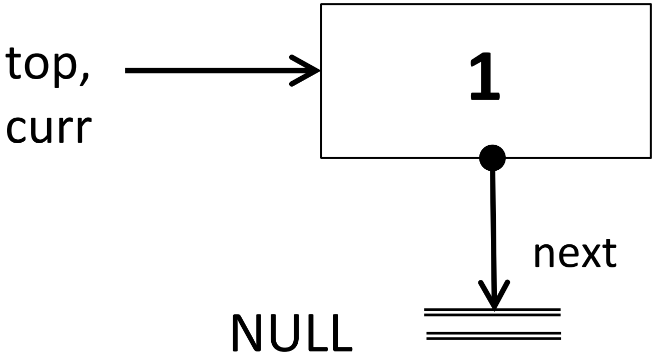


# Example stack

```
struct stack_elem * push(int value, struct stack_elem *top){
    struct stack_elem *curr = top;
    top = malloc(sizeof(stack));
    top->data = value;
    top->next = curr;
    return top;
}
```

main.c

```
 top = push(1, top);
 printf("Stack Data: %d\n", top->data);

top = push(2, top);
 printf("Stack Data: %d\n", top->data);

 top = push(3, top);
 printf("Stack Data: %d\n", top->data);
```

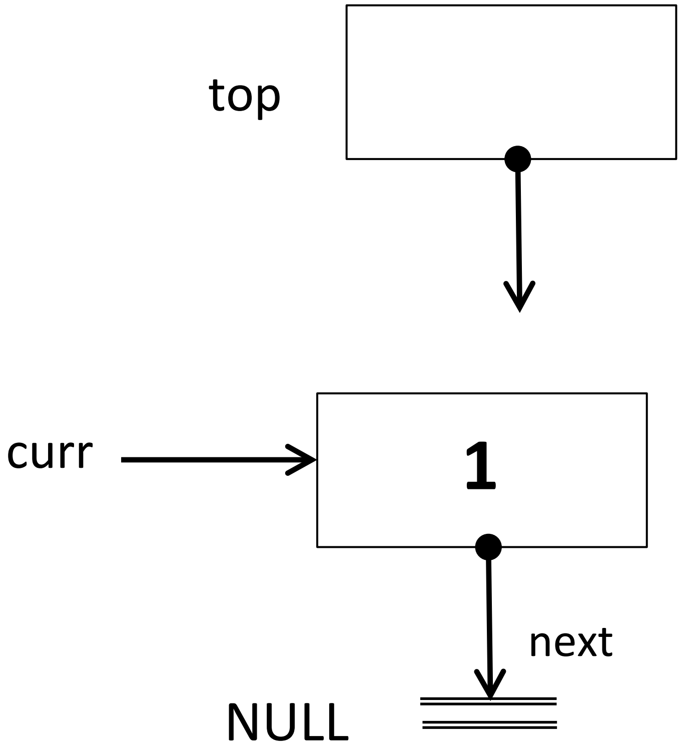


# Example stack

```
struct stack_elem * push(int value, struct stack_elem *top){
    struct stack_elem *curr = top;
    top = malloc(sizeof(stack));
    top->data = value;
    top->next = curr;
    return top;
}
```

main.c

```
top = push(1, top);
printf("Stack Data: %d\n", top->data);

top = push(2, top);
printf("Stack Data: %d\n", top->data);

top = push(3, top);
printf("Stack Data: %d\n", top->data);
```

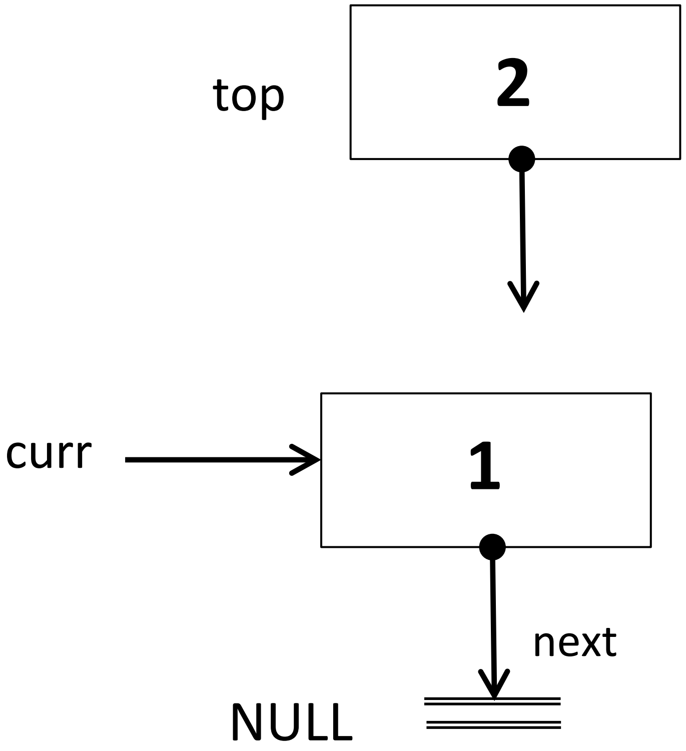


# Example stack

```
struct stack_elem * push(int value, struct stack_elem *top){
   struct stack_elem *curr = top;
   top = malloc(sizeof(stack));
   top->data = value;
   top->next = curr;
   return top;
}
```

main.c

```
top = push(1, top);
printf("Stack Data: %d\n", top->data);

top = push(2, top);
printf("Stack Data: %d\n", top->data);

top = push(3, top);
printf("Stack Data: %d\n", top->data);
```

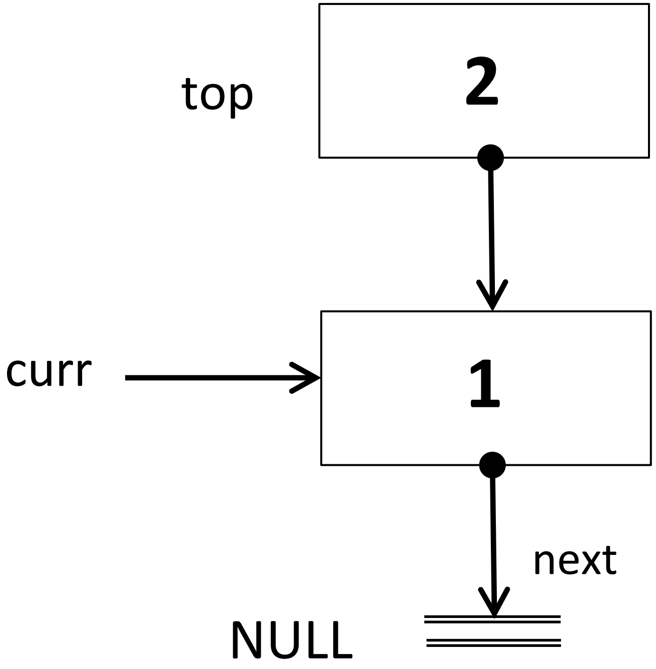


# Example stack

```
struct stack_elem * push(int value, struct stack_elem *top){
    struct stack_elem *curr = top;
    top = malloc(sizeof(stack));
    top->data = value;
    top->next = curr;
    return top;
}
```

main.c

```
top = push(1, top);
printf("Stack Data: %d\n", top->data);

top = push(2, top);
printf("Stack Data: %d\n", top->data);

top = push(3, top);
printf("Stack Data: %d\n", top->data);
```

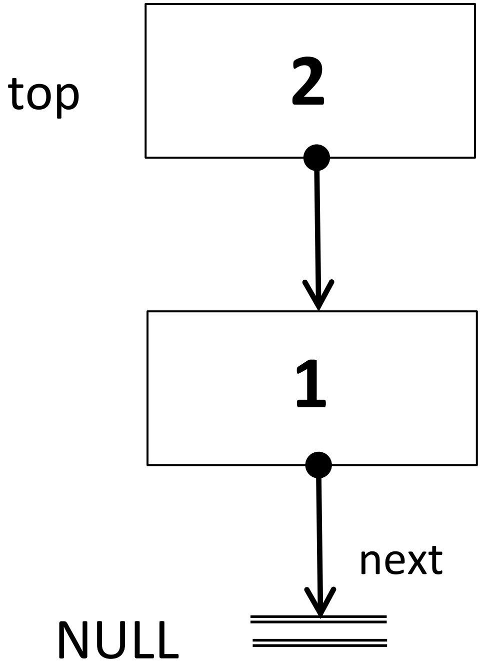


# Example stack

```
struct stack_elem * push(int value, struct stack_elem *top){
    struct stack_elem *curr = top;
    top = malloc(sizeof(stack));
    top->data = value;
    top->next = curr;
    return top;
}
```

main.c

```
top = push(1, top);
printf("Stack Data: %d\n", top->data);

top = push(2, top);
printf("Stack Data: %d\n", top->data);

top = push(3, top);
printf("Stack Data: %d\n", top->data);
```

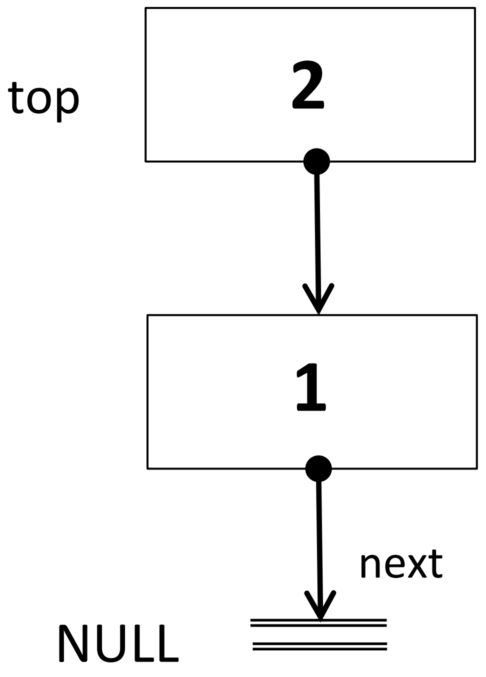


# Example stack

```
struct stack_elem * push(int value, struct stack_elem *top){
   struct stack_elem *curr = top;
   top = malloc(sizeof(stack));
   top->data = value;
   top->next = curr;
   return top;
}
```

main.c

```
top = push(1, top);
printf("Stack Data: %d\n", top->data);

top = push(2, top);
printf("Stack Data: %d\n", top->data);

top = push(3, top);
printf("Stack Data: %d\n", top->data);
```



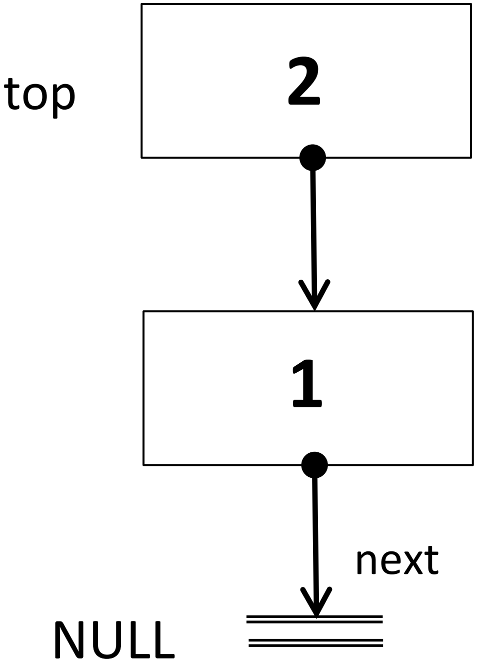
# Example stack

```
struct stack_elem * push(int value, struct stack_elem *top){
    struct stack_elem *curr = top;
    top = malloc(sizeof(stack));
    top->data = value;
    top->next = curr;
    return top;
}
```

main.c

```
 top = push(1, top);
 printf("Stack Data: %d\n", top->data);

top = push(2, top);
 printf("Stack Data: %d\n", top->data);
```

```
 top = push(3, top);
 printf("Stack Data: %d\n", top->data);
```

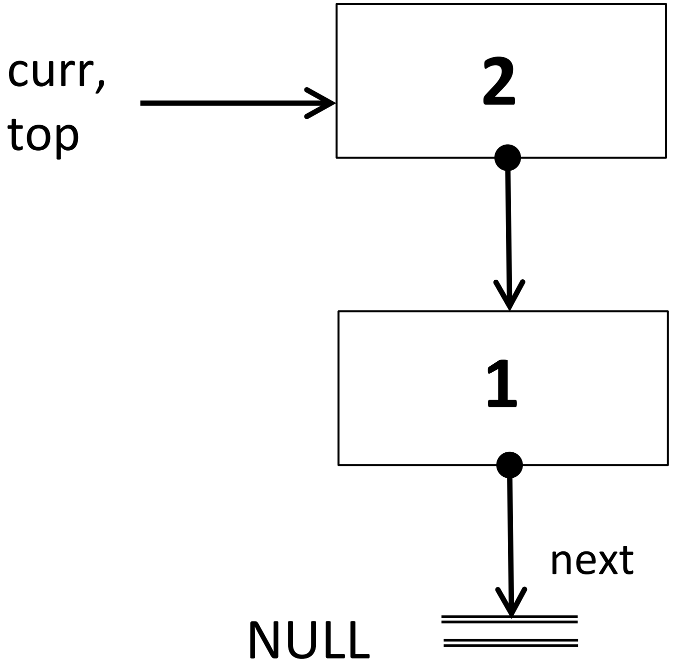


# Example stack

```
struct stack_elem * push(int value, struct stack_elem *top){
    struct stack_elem *curr = top;
    top = malloc(sizeof(stack));
    top->data = value;
    top->next = curr;
    return top;
}
```

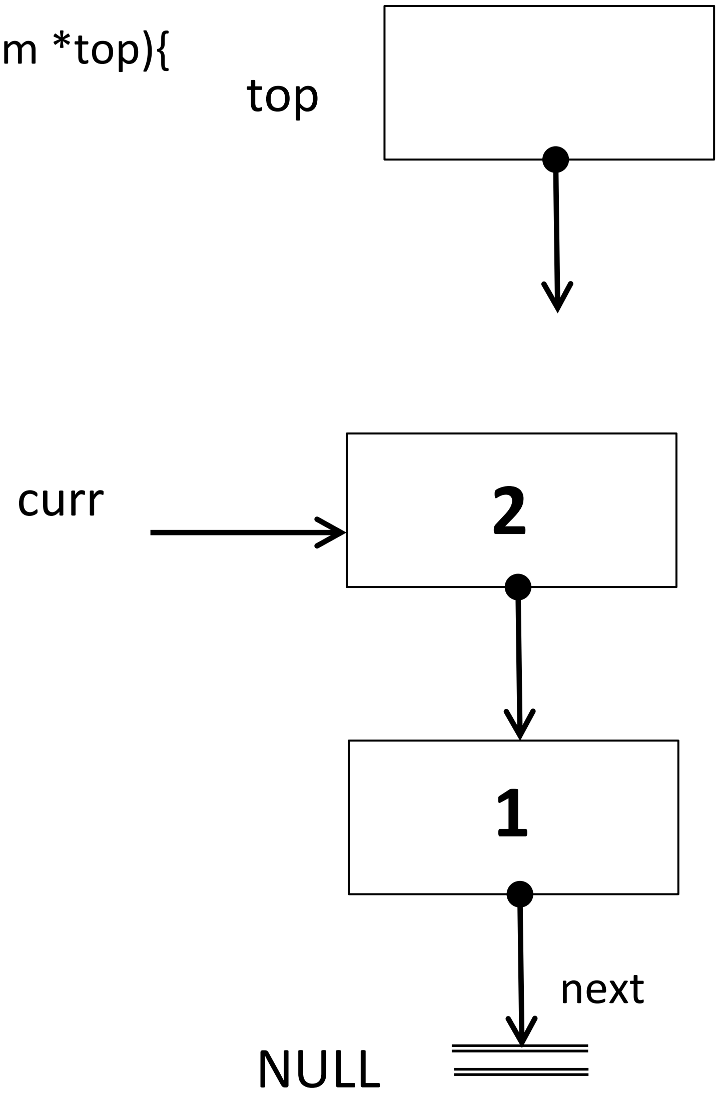
main.c

```
 top = push(1, top);
 printf("Stack Data: %d\n", top->data);

top = push(2, top);
 printf("Stack Data: %d\n", top->data);

 top = push(3, top);
 printf("Stack Data: %d\n", top->data);
```



# Example stack

```
struct stack_elem * push(int value, struct stack_elem *top){
    struct stack_elem *curr = top;
    top = malloc(sizeof(stack));
    top->data = value;
    top->next = curr;
    return top;
}
```



main.c

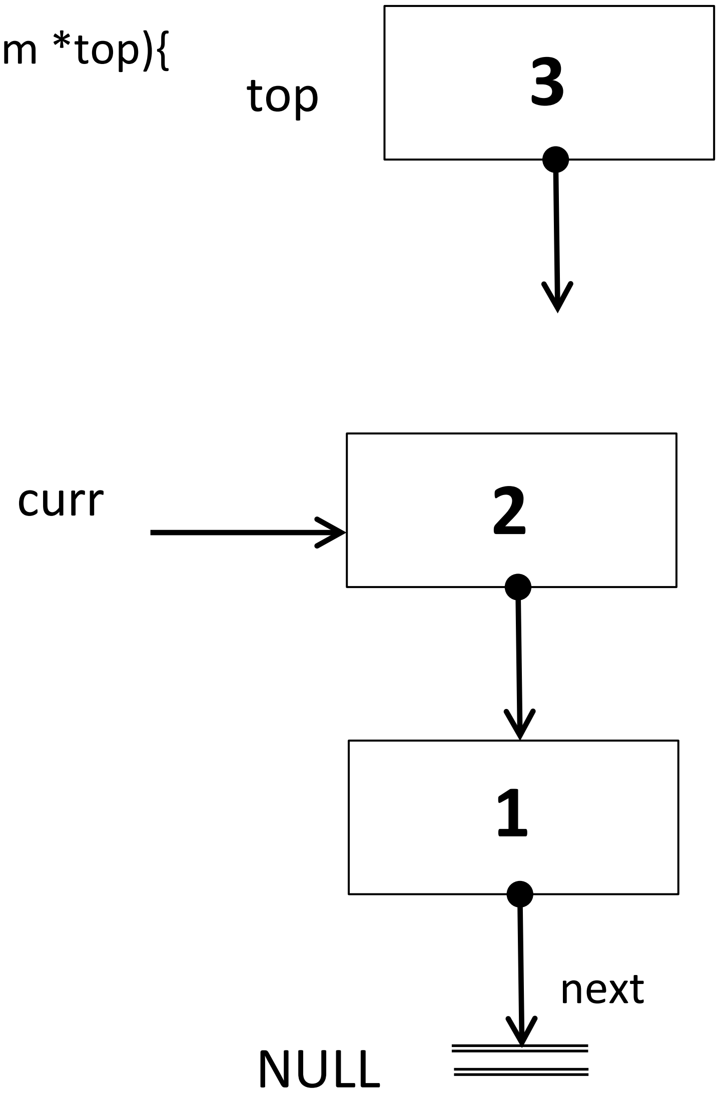
```
top = push(1, top);
printf("Stack Data: %d\n", top->data);

top = push(2, top);
printf("Stack Data: %d\n", top->data);

top = push(3, top);
printf("Stack Data: %d\n", top->data);
```

# Example stack

```
struct stack_elem * push(int value, struct stack_elem *top){
   struct stack_elem *curr = top;
   top = malloc(sizeof(stack));
   top->data = value;
   top->next = curr;
   return top;
}
```

main.c

```
top = push(1, top);
printf("Stack Data: %d\n", top->data);

top = push(2, top);
printf("Stack Data: %d\n", top->data);

top = push(3, top);
printf("Stack Data: %d\n", top->data);
```

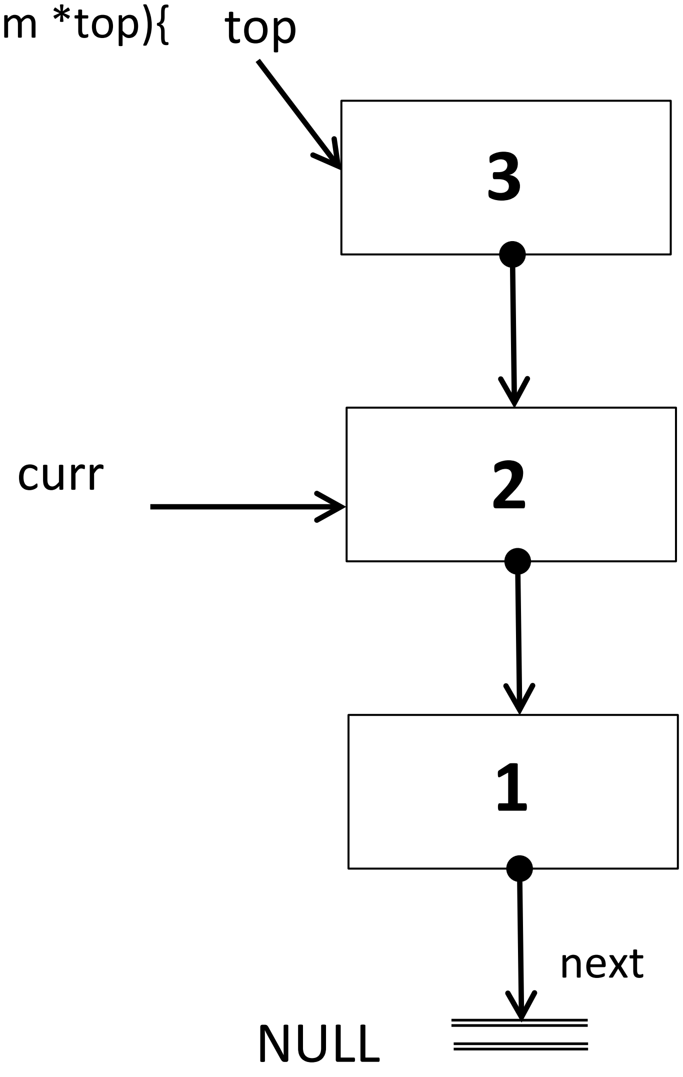


# Example stack

```
struct stack_elem * push(int value, struct stack_elem *top){
    struct stack_elem *curr = top;
    top = malloc(sizeof(stack));
    top->data = value;
    top->next = curr;
    return top;
}
```

main.c

```
top = push(1, top);
printf("Stack Data: %d\n", top->data);

top = push(2, top);
printf("Stack Data: %d\n", top->data);

top = push(3, top);
printf("Stack Data: %d\n", top->data);
```

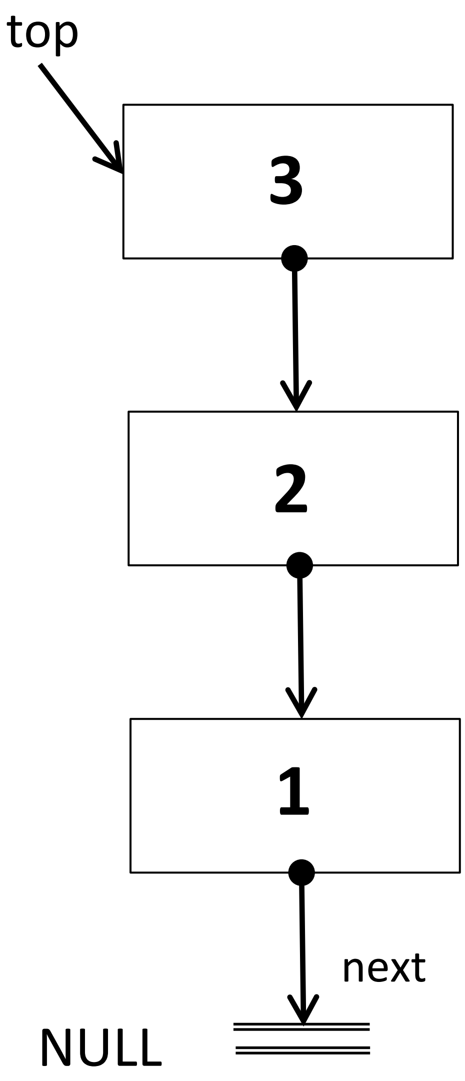


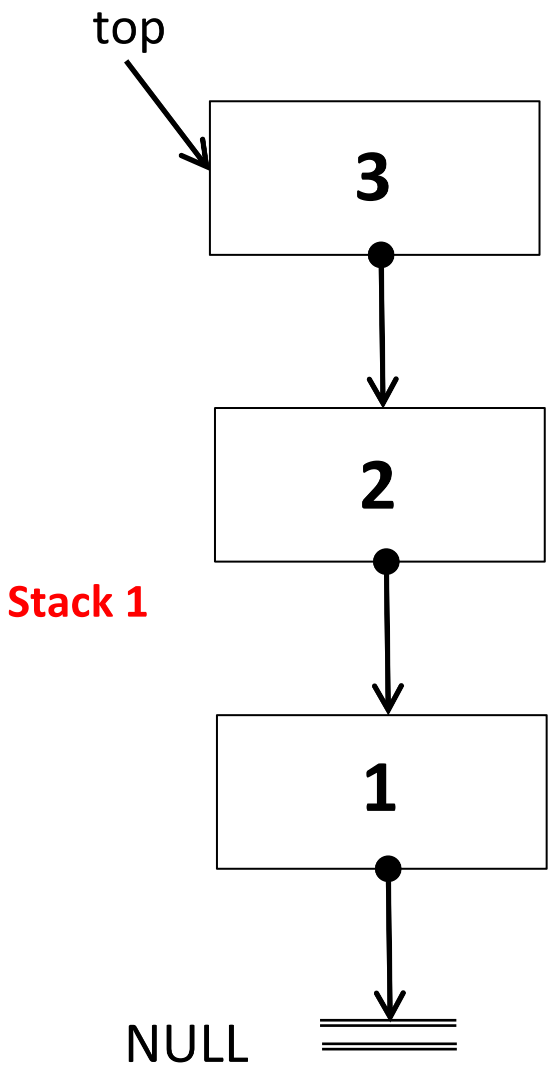
# Example stack

```
struct stack_elem * push(int value, struct stack_elem *top){
    struct stack_elem *curr = top;
    top = malloc(sizeof(stack));
    top->data = value;
    top->next = curr;
    return top;
}
```

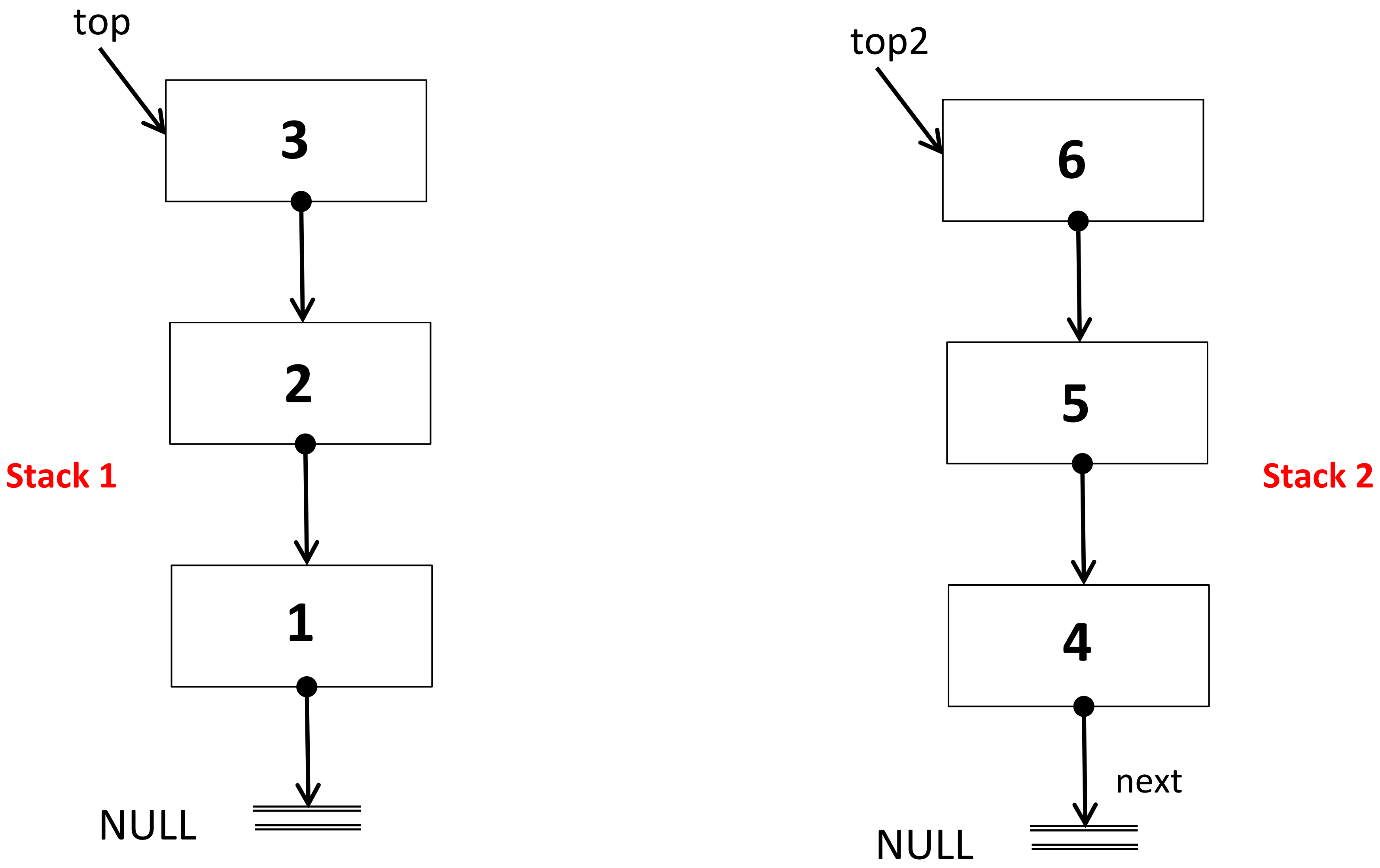
main.c

```
top = push(1, top);
printf("Stack Data: %d\n", top->data);

top = push(2, top);
printf("Stack Data: %d\n", top->data);

top = push(3, top);
printf("Stack Data: %d\n", top->data);
```

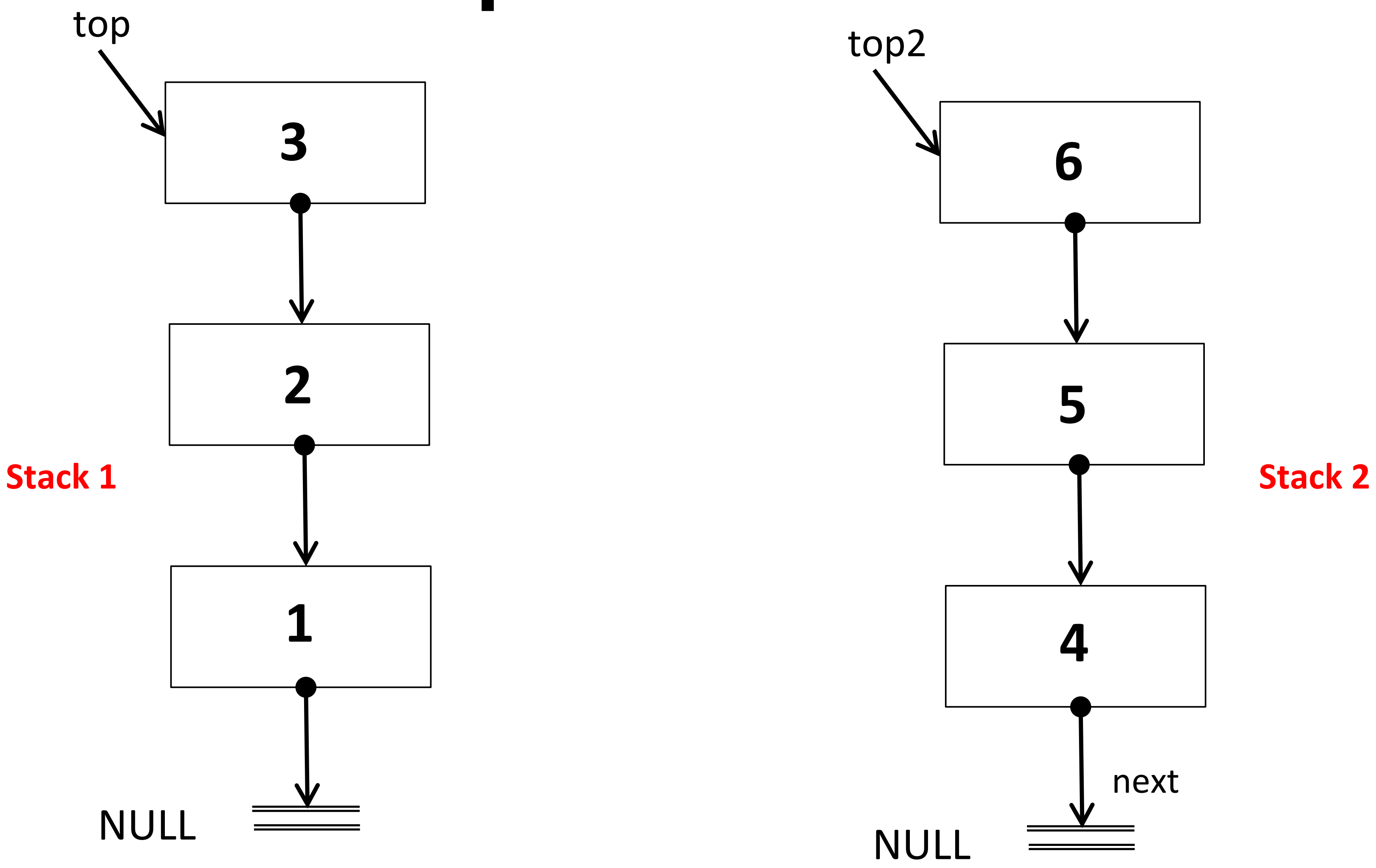



top
3
2
Stack 1
1
NULL

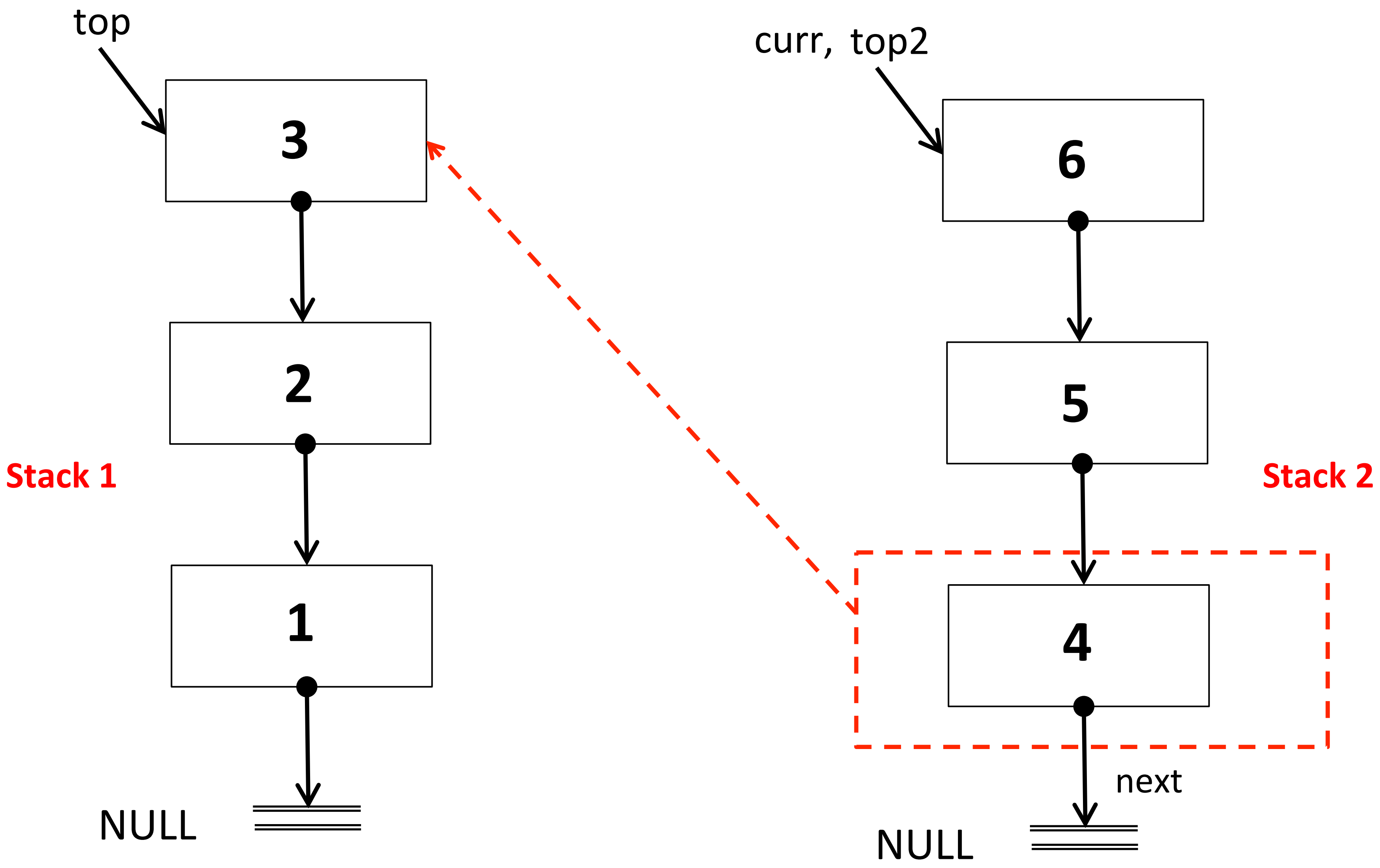
# Imagine we created 2 stacks…



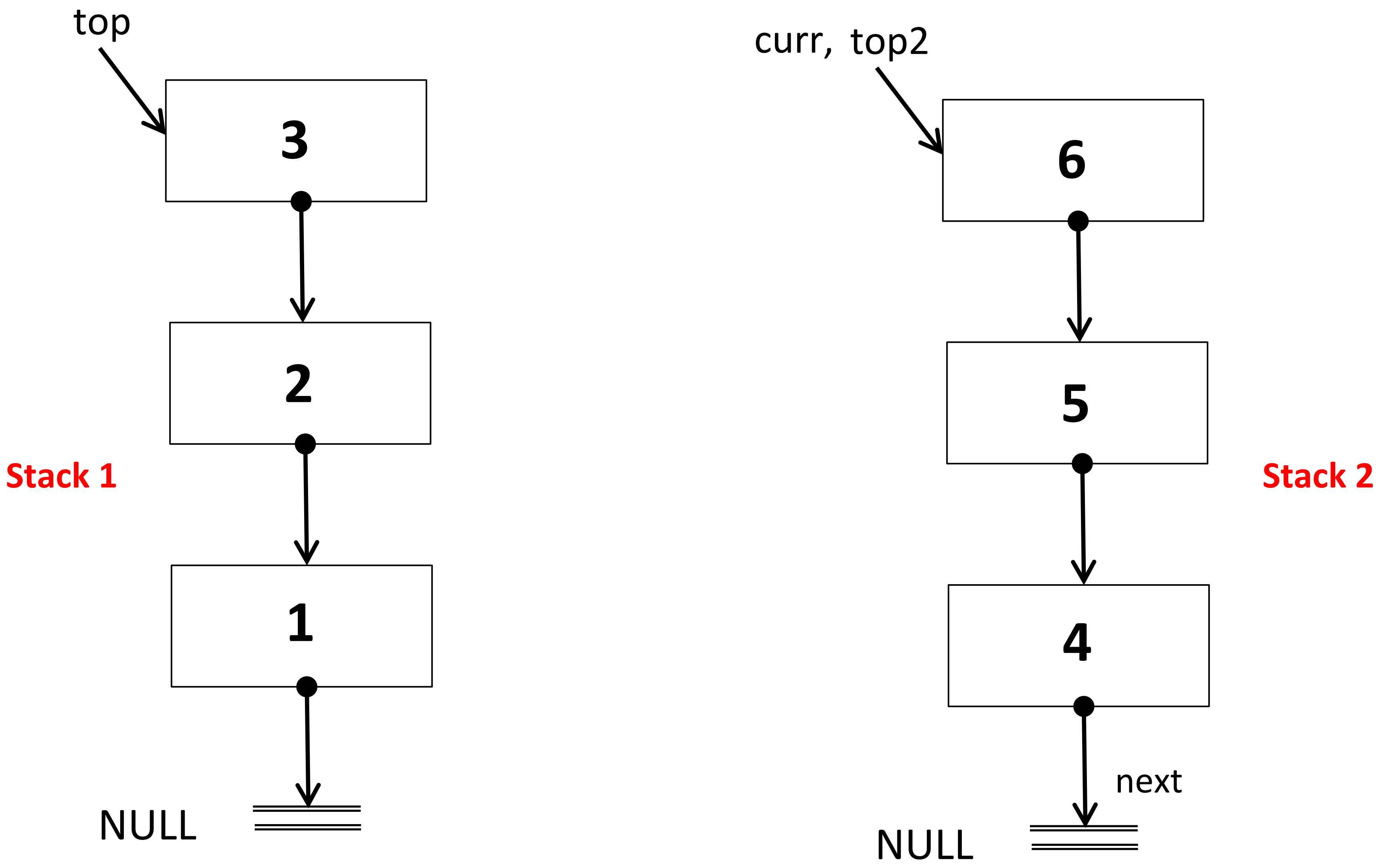
# … And we want to put stack 2 on top of stack 1



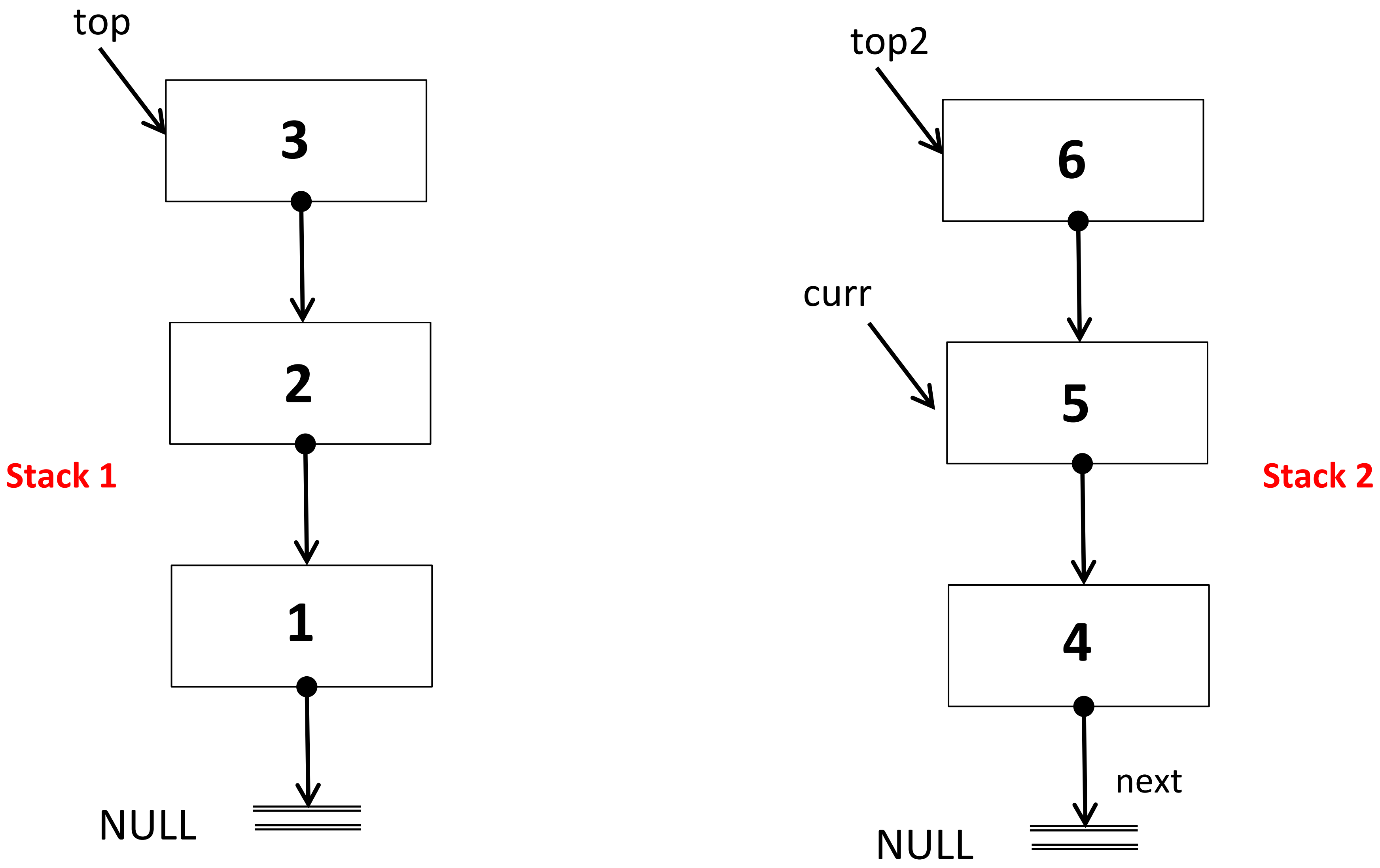
**the last element of the Stack 2 should have its pointer *“next”* to the top of Stack 1**



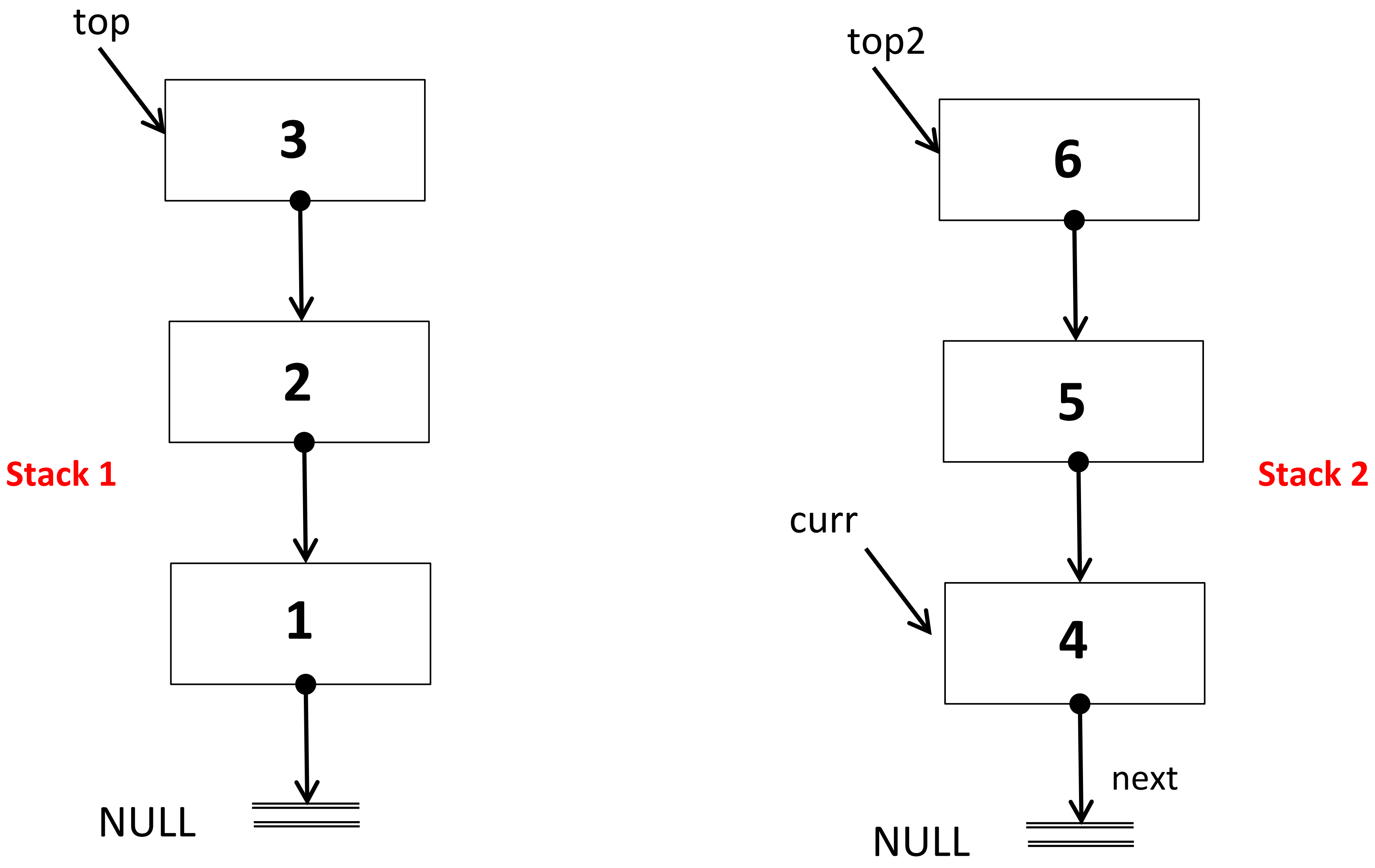
```
while(curr -> next != NULL)
    curr = curr-> next;
```



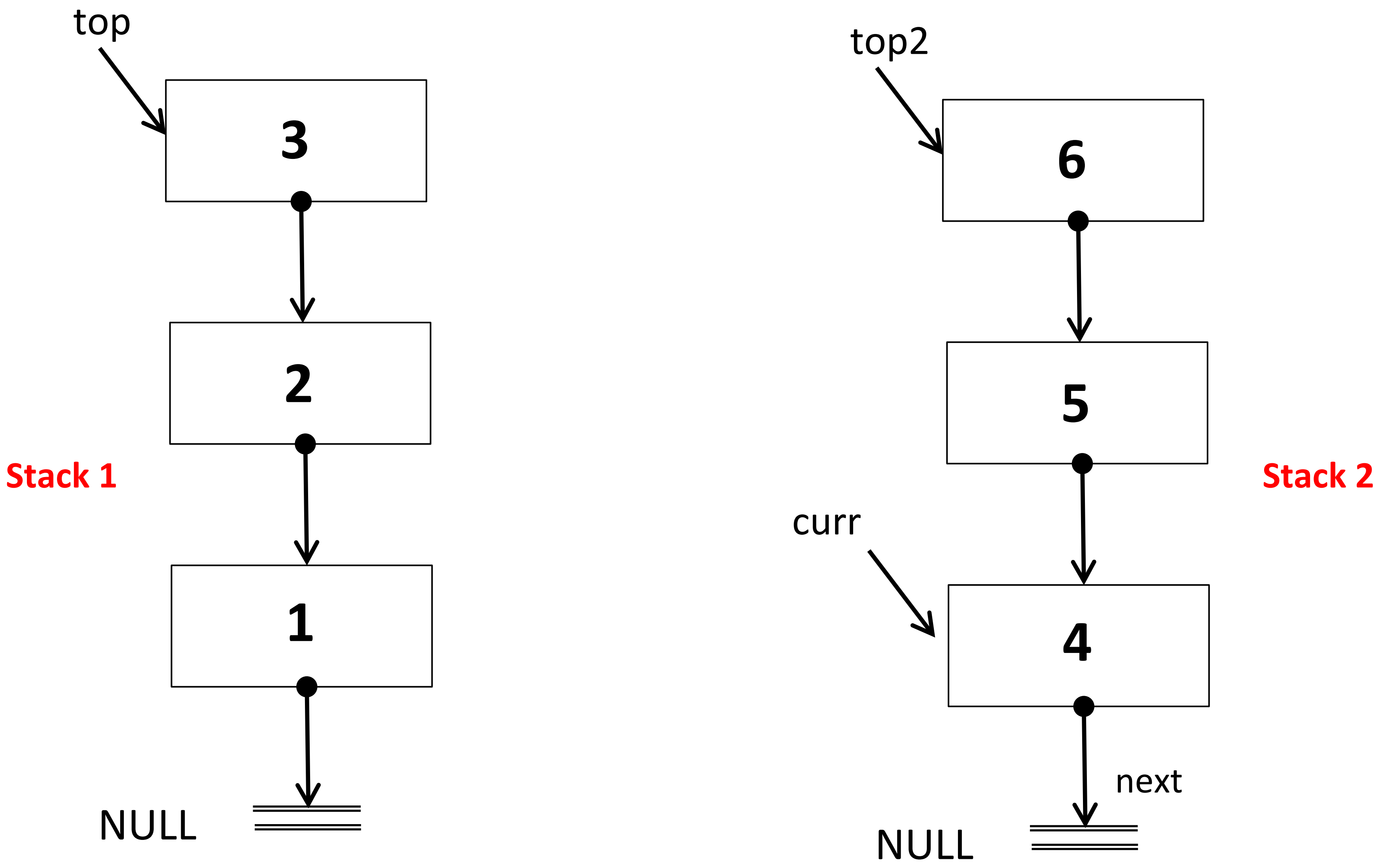
```
while(curr -> next != NULL)
    curr = curr-> next;
```



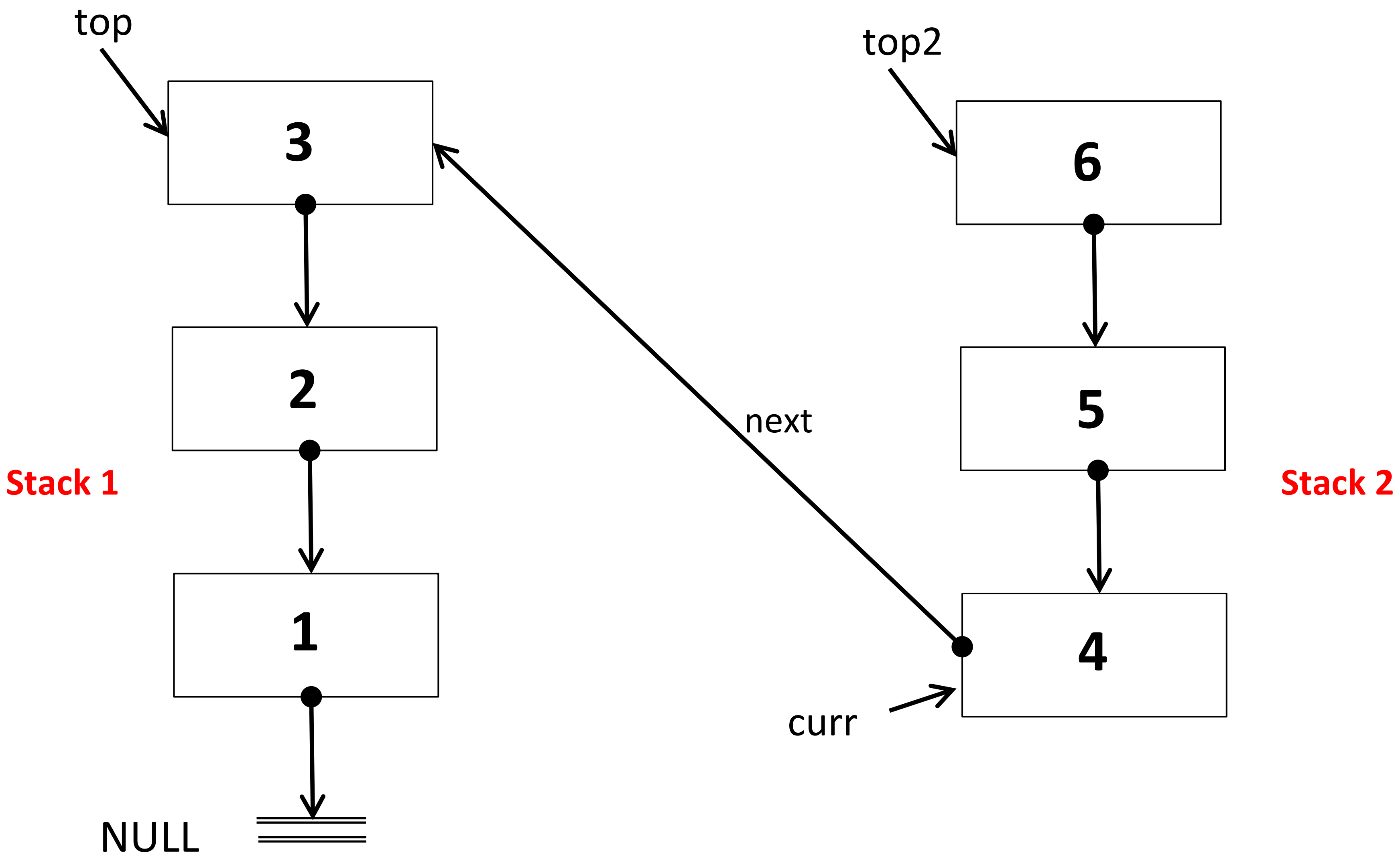
```
while(curr -> next != NULL)
    curr = curr-> next;
```

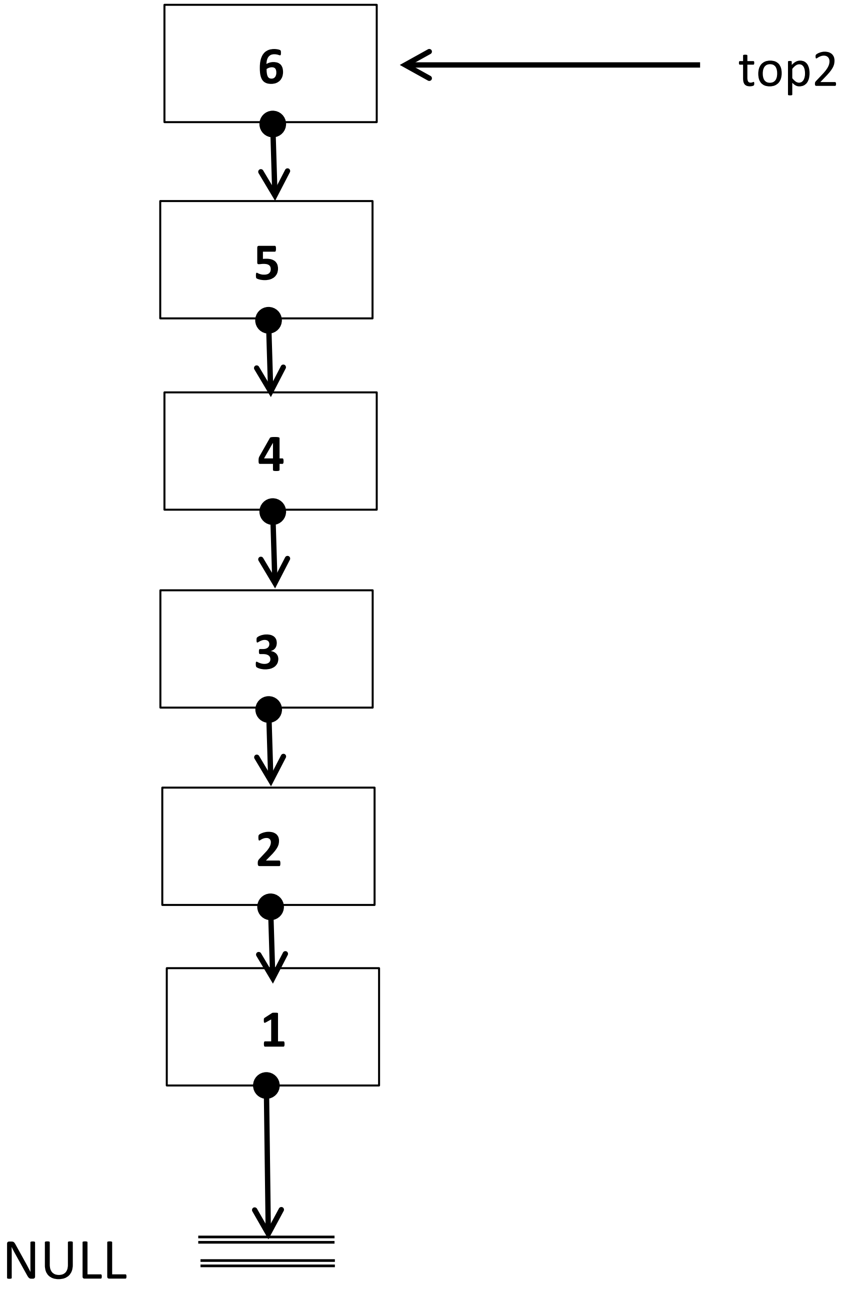


```
curr->next = top;
```



```
curr->next = top;
```




6
top2
5
4
3
2
1
NULL

# Recap

- **Initialize a Data Structure**
  - Initialize each memebr of the data structure

- **Stack**
  - Recap on how to create a LIFO stack
  - How to merge 2 stacks