Linked Lists & Stacks

Outline

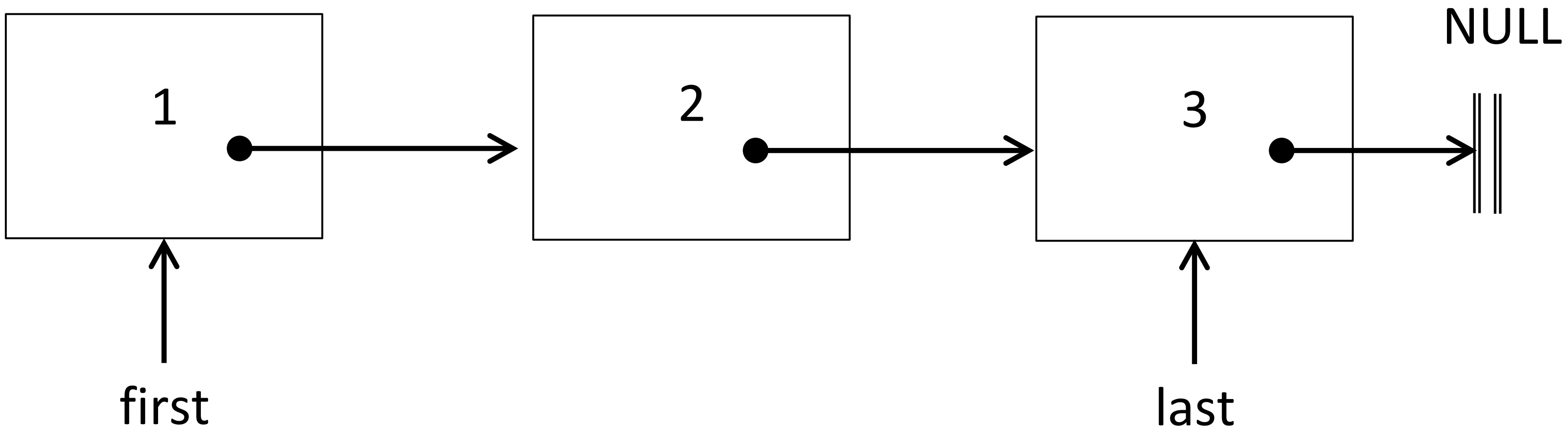
- **Linked list**
  - Example of a linked list based on the FIFO (First-In First-Out) ordering principle

- **Stack**
  - Example of a linked list based on the LIFO (Last-In First Out) ordering principle

FIFO Linked List

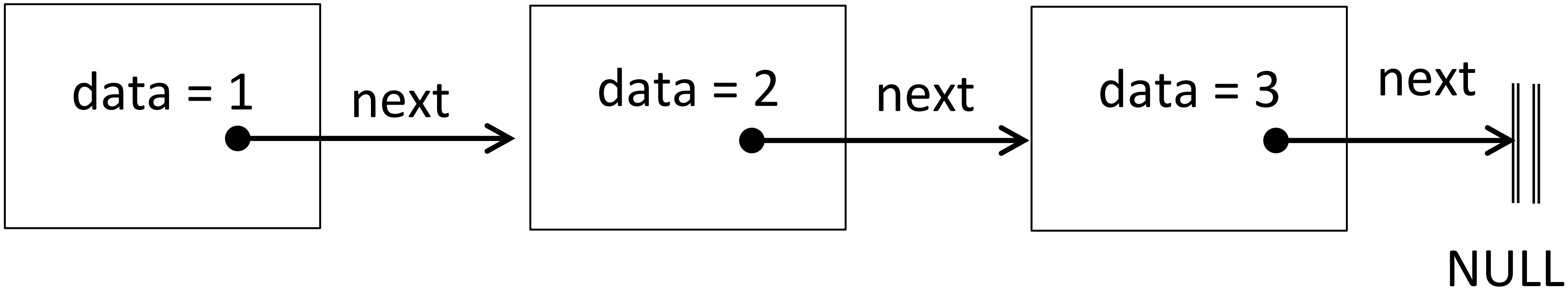
Create a Chain of Integers

- Create a chain of elements, each of them containing an integer
- Each element should be linked to the next one
- **FIFO (First-In First Out) ordering principle:** Elements should be removed from the chain in the order in which the are inserted



Structure Members can Be Self-Referential

```
struct chain_element {
   int data;
   struct chain_element *next;
};
```



Adding Elements

Example (Program chain.c)

```
struct chain_element {
  int data;
  struct chain_element* next
} chain;

int main(int) {
    int chainSize;

    struct chain_element *curr;
    struct chain_element *first;
    struct chain_element *last;

    printf("Insert number of elements\n");
    scanf("%d",&chainSize);
```

Example (Program chain.c)

```
struct chain_element {
  int data;
  struct chain_element* next
} chain;
```

```
int main(int) {
    int chainSize;

    struct chain_element *curr;
    struct chain_element *first;
    struct chain_element *last;

    printf("Insert number of elements\n");
    scanf("%d",&chainSize);
```

Example (Program chain.c)

```
struct chain_element {
  int data;
  struct chain_element* next
} chain;

int main(int) {
    int chainSize;

    struct chain_element *curr;
    struct chain_element *first;
    struct chain_element *last;

    printf("Insert number of elements\n");
    scanf("%d",&chainSize);
```

Example (Program chain.c)

chainSize = 3

```
struct chain_element {
 int data;
 struct chain_element* next
} chain;

int main(int) {
   int chainSize;

   struct chain_element *curr;
   struct chain_element *first;
   struct chain_element *last;

   printf("Insert number of elements\n");
   scanf("%d",&chainSize);
```

Example (Program chain.c)

```
for (int i = 0; i < chainSize; i++) {
    last = malloc (sizeof (chain));
    last->data = i + 1;
    last->next = NULL;
    if(i==0)
        first = last;
    else
        curr-> next = last;
    curr = last;
}
```

chainSize = 3

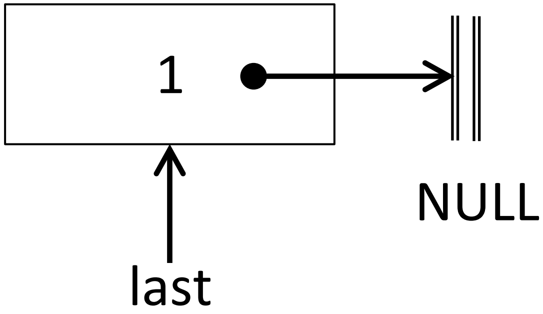
i = 0

Example (Program chain.c)

```
for (int i = 0; i < chainSize; i++) {
    last = malloc (sizeof (chain));
    last->data = i + 1;
    last->next = NULL;
    if(i==0)
        first = last;
    else
        curr-> next = last;
    curr = last;
}
```

chainSize = 3

i = 0



1

NULL

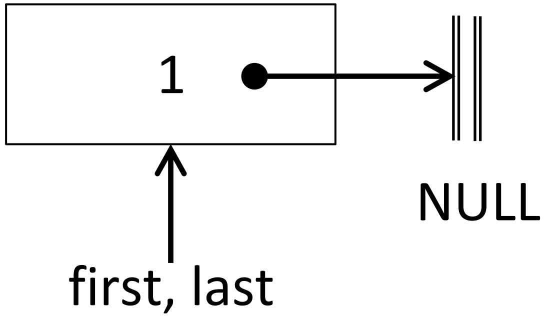
last

Example (Program chain.c)

```
for (int i = 0; i < chainSize; i++) {
    last = malloc (sizeof (chain));
    last->data = i + 1;
    last->next = NULL;
    if(i==0)
        first = last;
    else
        curr-> next = last;
    curr = last;
}
```

chainSize = 3

i = 0

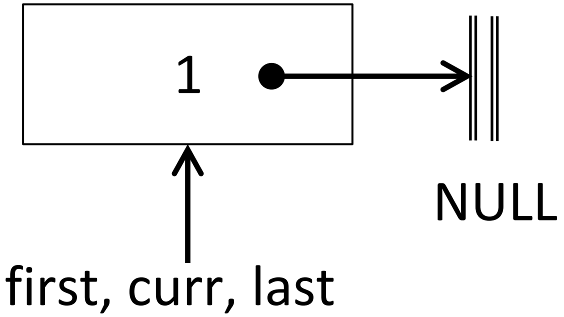


Example (Program chain.c)

```
for (int i = 0; i < chainSize; i++) {
    last = malloc (sizeof (chain));
    last->data = i + 1;
    last->next = NULL;
    if(i==0)
           first = last;
    else
           curr-> next = last;
    curr = last;
}
```

chainSize = 3

i = 0

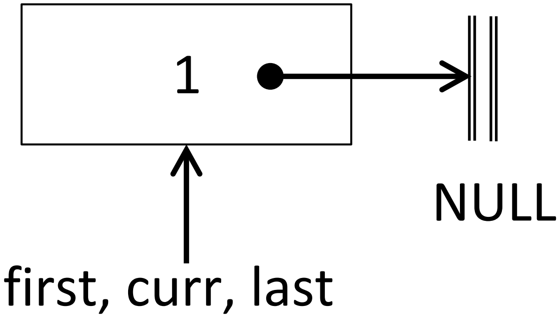


Example (Program chain.c)

```
for (int i = 0; i < chainSize; i++) {
    last = malloc (sizeof (chain));
    last->data = i + 1;
    last->next = NULL;
    if(i==0)
        first = last;
    else
        curr-> next = last;
    curr = last;
}
```

chainSize = 3

i = 1

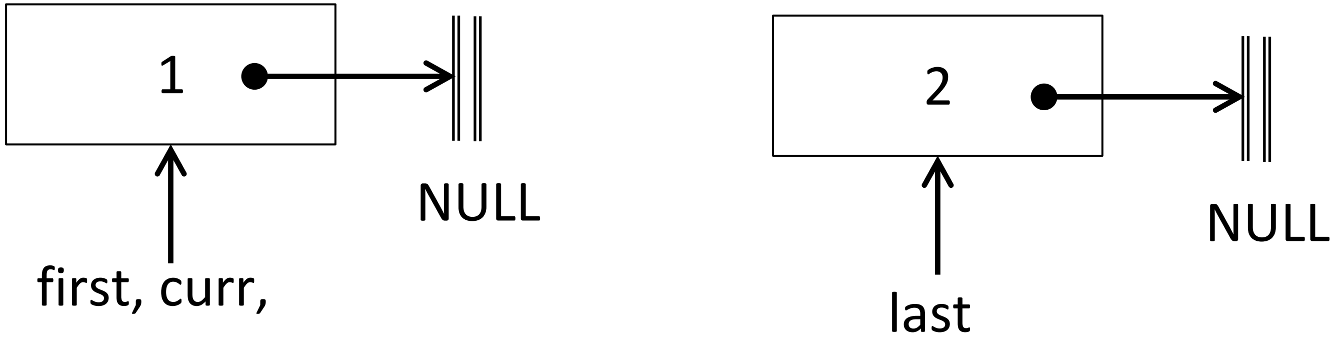


Example (Program chain.c)

```
for (int i = 0; i < chainSize; i++) {
    last = malloc (sizeof (chain));
    last->data = i + 1;
    last->next = NULL;
    if(i==0)
        first = last;
    else
        curr-> next = last;
    curr = last;
}
```

chainSize = 3

i = 1

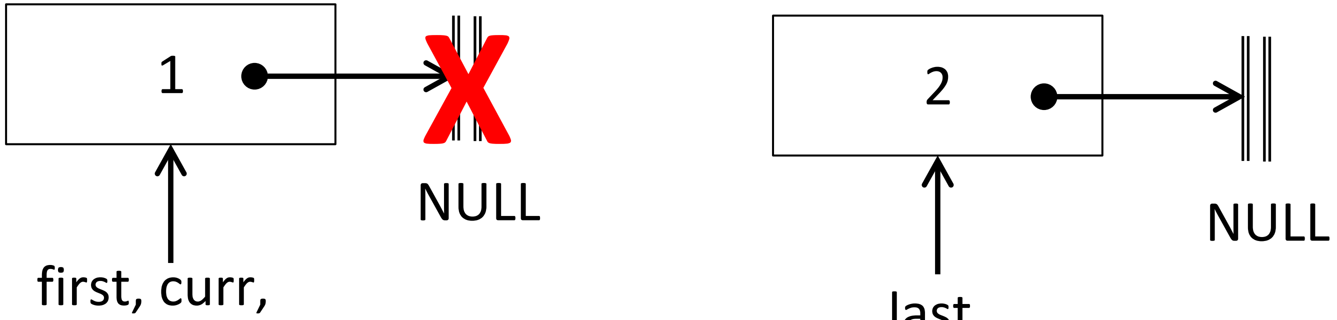


Example (Program chain.c)

```
for (int i = 0; i < chainSize; i++) {
    last = malloc (sizeof (chain));
    last->data = i + 1;
    last->next = NULL;
    if(i==0)
        first = last;
    else
        curr-> next = last;
    curr = last;
}
```

chainSize = 3

i = 1

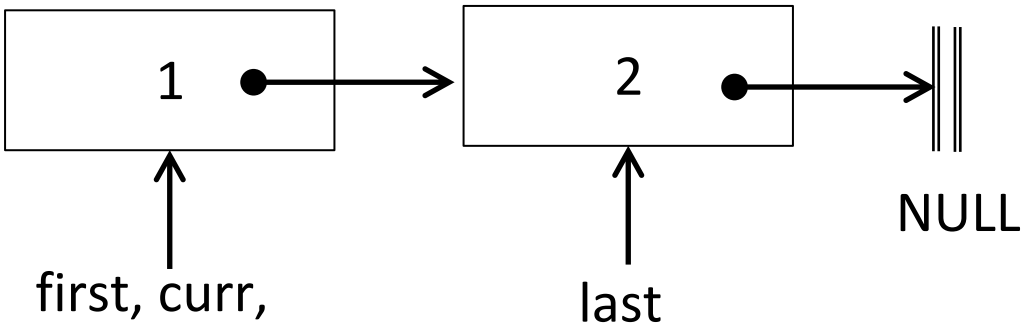


Example (Program chain.c)

```
for (int i = 0; i < chainSize; i++) {
    last = malloc (sizeof (chain));
    last->data = i + 1;
    last->next = NULL;
    if(i==0)
        first = last;
    else
        curr-> next = last;
    curr = last;
}
```

chainSize = 3

i = 1

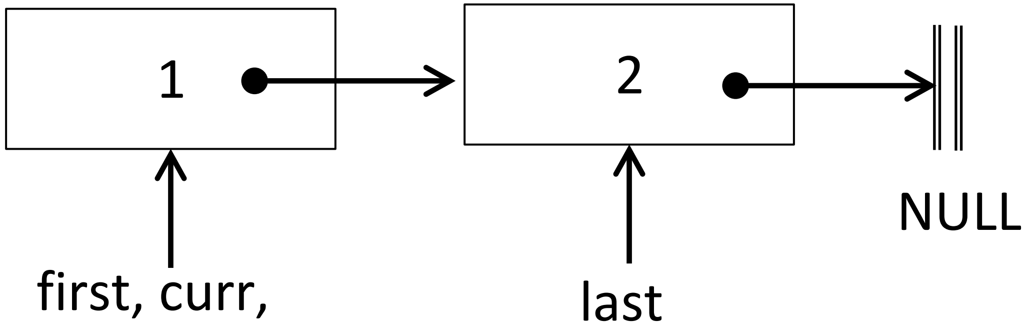


Example (Program chain.c)

```
for (int i = 0; i < chainSize; i++) {
    last = malloc (sizeof (chain));
    last->data = i + 1;
    last->next = NULL;
    if(i==0)
           first = last;
    else
           curr-> next = last;
    curr = last;
}
```

chainSize = 3

i = 1

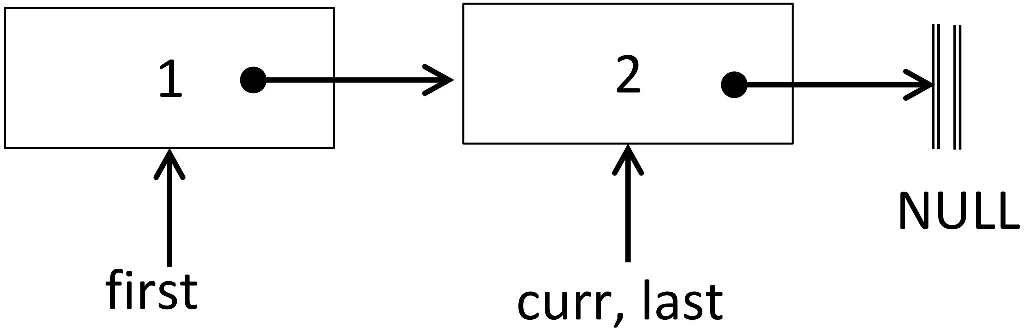


Example (Program chain.c)

```
for (int i = 0; i < chainSize; i++) {
    last = malloc (sizeof (chain));
    last->data = i + 1;
    last->next = NULL;
    if(i==0)
            first = last;
    else
            curr-> next = last;
    curr = last;
}
```

chainSize =3

i = 1

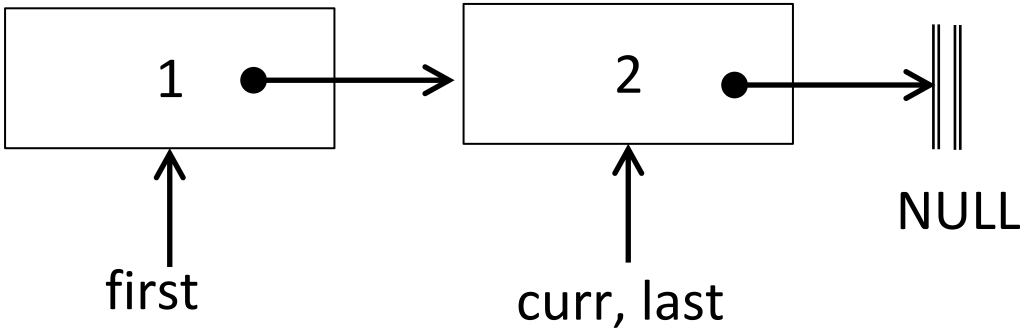


Example (Program chain.c)

```
for (int i = 0; i < chainSize; i++) {
    last = malloc (sizeof (chain));
    last->data = i + 1;
    last->next = NULL;
    if(i==0)
        first = last;
    else
        curr-> next = last;
    curr = last;
}
```

chainSize =3

i = 2

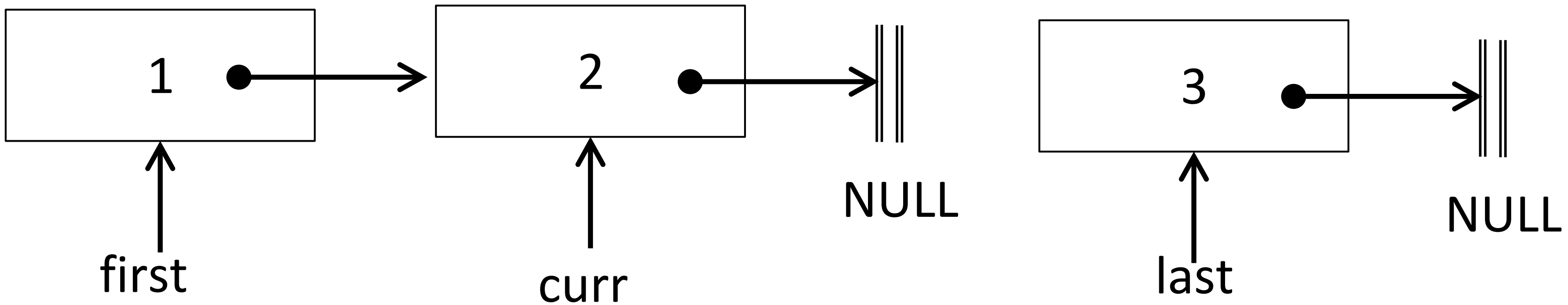


Example (Program chain.c)

```
for (int i = 0; i < chainSize; i++) {
    last = malloc (sizeof (chain));
    last->data = i + 1;
    last->next = NULL;
    if(i==0)
          first = last;
    else
          curr-> next = last;
    curr = last;
}
```

chainSize =3

i = 2

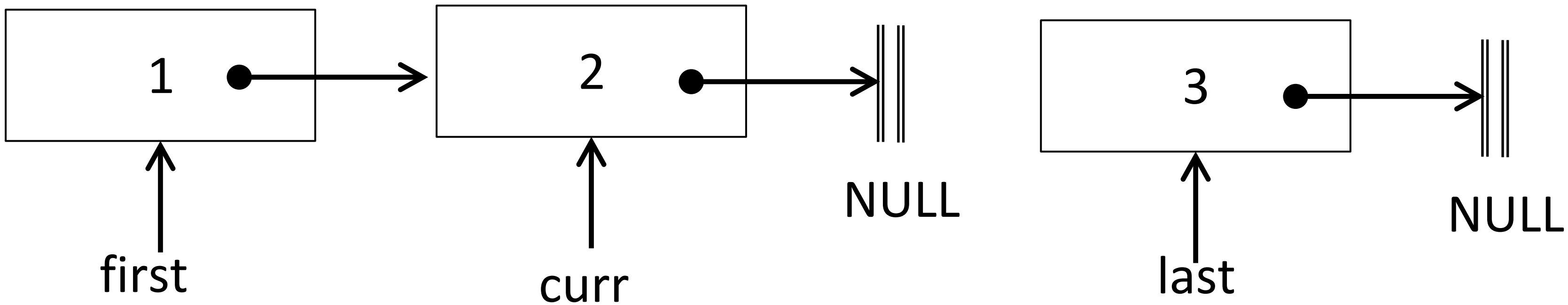


Example (Program chain.c)

```
for (int i = 0; i < chainSize; i++) {
    last = malloc (sizeof (chain));
    last->data = i + 1;
    last->next = NULL;
    if(i==0)
        first = last;
    else
        curr-> next = last;
    curr = last;
}
```

chainSize =3

i = 2

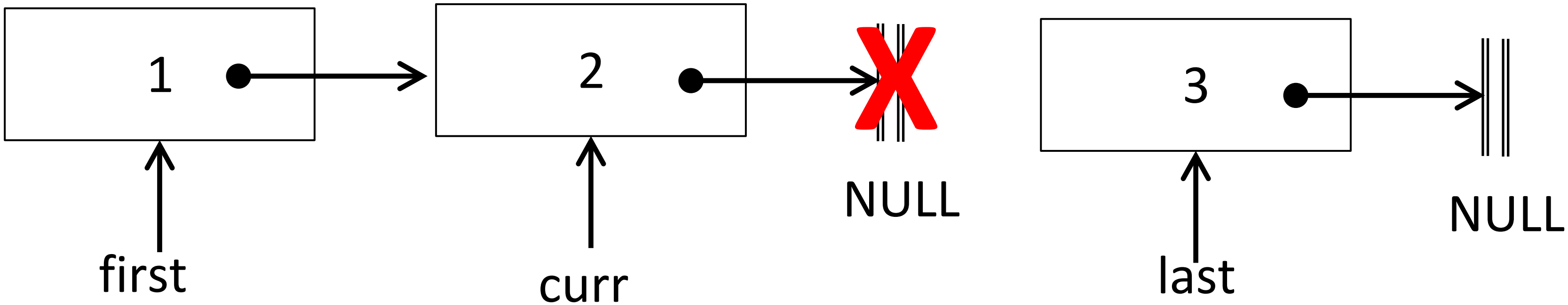


Example (Program chain.c)

```
for (int i = 0; i < chainSize; i++) {
    last = malloc (sizeof (chain));
    last->data = i + 1;
    last->next = NULL;
    if(i==0)
           first = last;
    else
           curr-> next = last;
    curr = last;
}
```

chainSize =3

i = 2

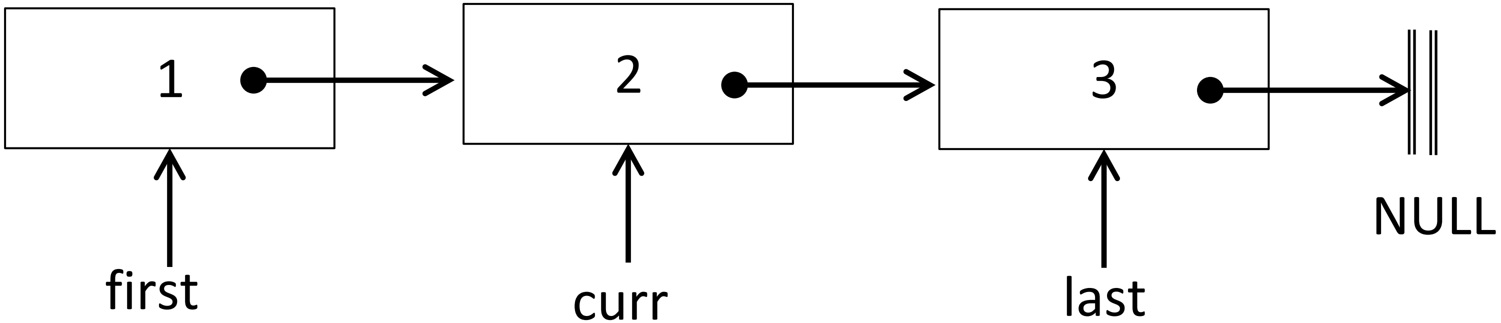


Example (Program chain.c)

```
for (int i = 0; i < chainSize; i++) {
    last = malloc (sizeof (chain));
    last->data = i + 1;
    last->next = NULL;
    if(i==0)
        first = last;
    else
        curr-> next = last;
    curr = last;
}
```

chainSize =3

i = 2

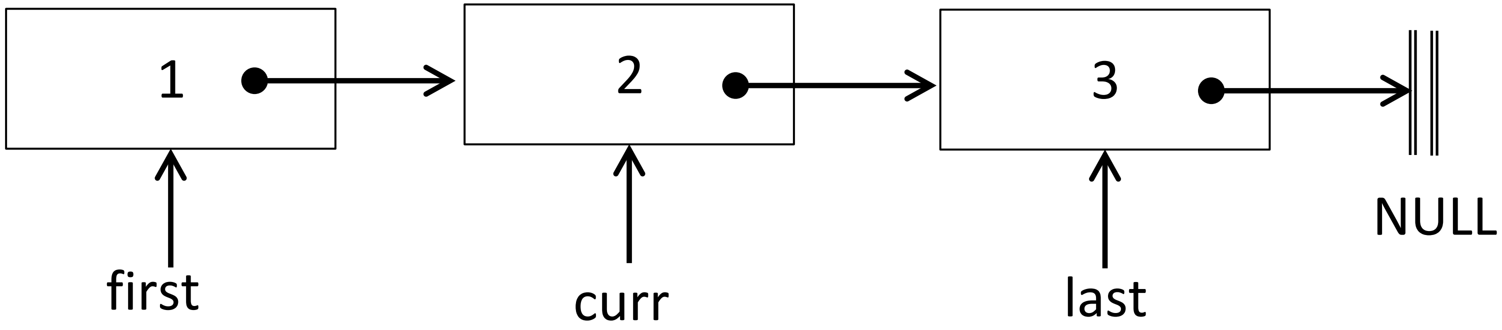


Example (Program chain.c)

```
for (int i = 0; i < chainSize; i++) {
    last = malloc (sizeof (chain));
    last->data = i + 1;
    last->next = NULL;
    if(i==0)
          first = last;
    else
          curr-> next = last;
    curr = last;
}
```

chainSize =3

i = 2

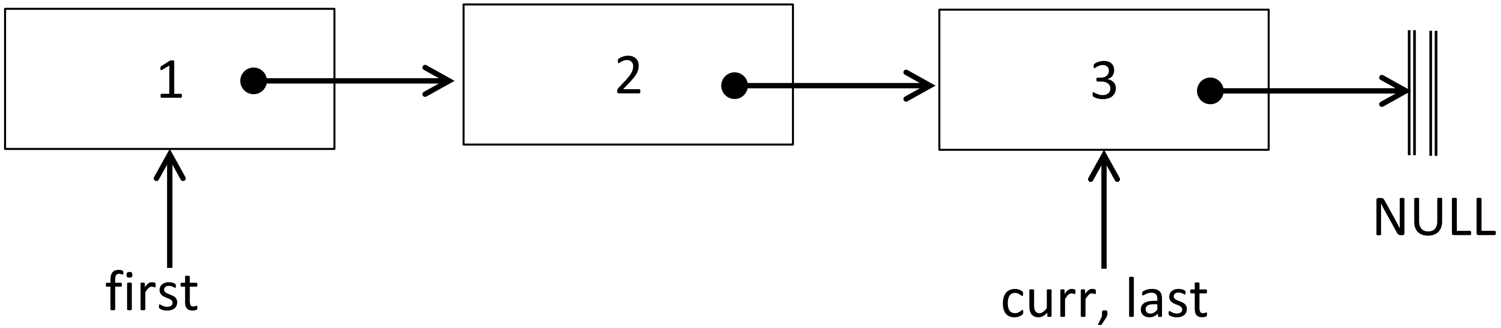


Example (Program chain.c)

```
for (int i = 0; i < chainSize; i++) {
    last = malloc (sizeof (chain));
    last->data = i + 1;
    last->next = NULL;
    if(i==0)
           first = last;
    else
           curr-> next = last;
    curr = last;
}
```

chainSize =3

i = 2

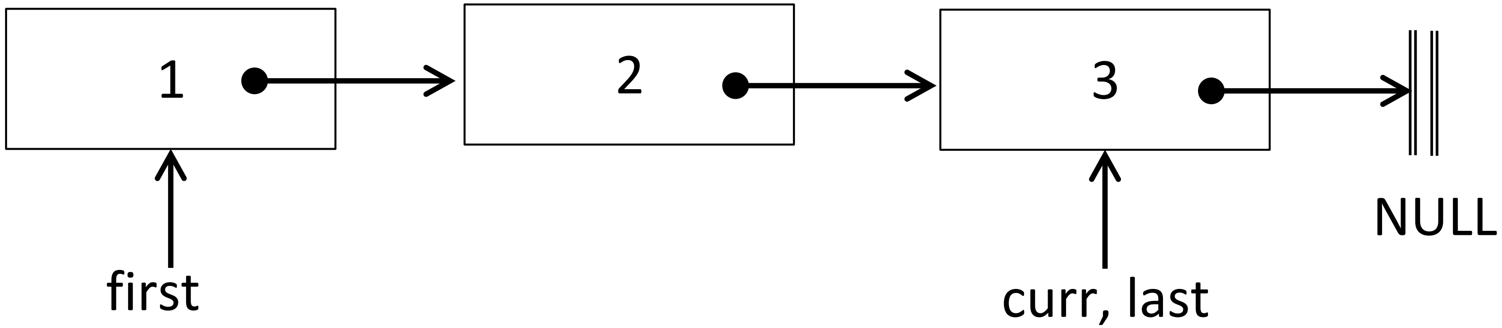


Example (Program chain.c)

```
for (int i = 0; i < chainSize; i++) {
    last = malloc (sizeof (chain));
    last->data = i + 1;
    last->next = NULL;
    if(i==0)
        first = last;
    else
        curr-> next = last;
    curr = last;
}
```

chainSize =3

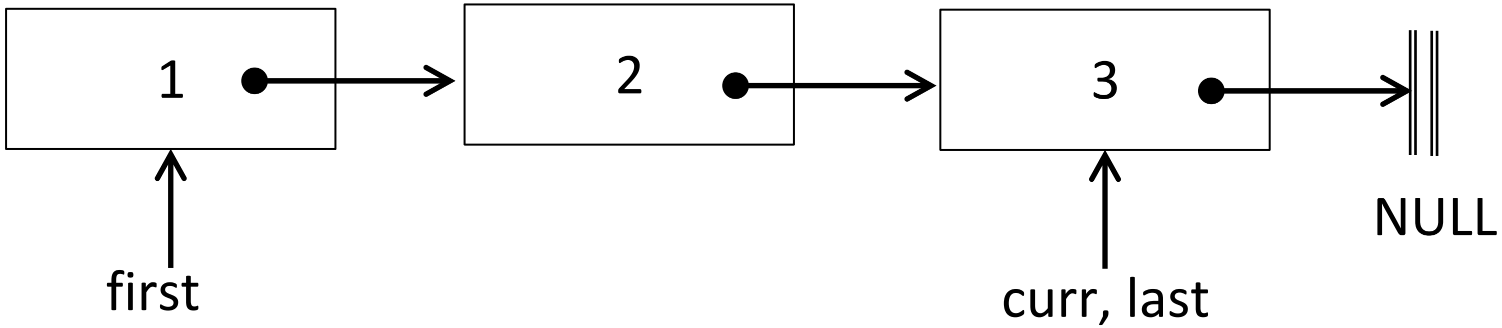
i = 3



Traversing the List & Printing Its Elements

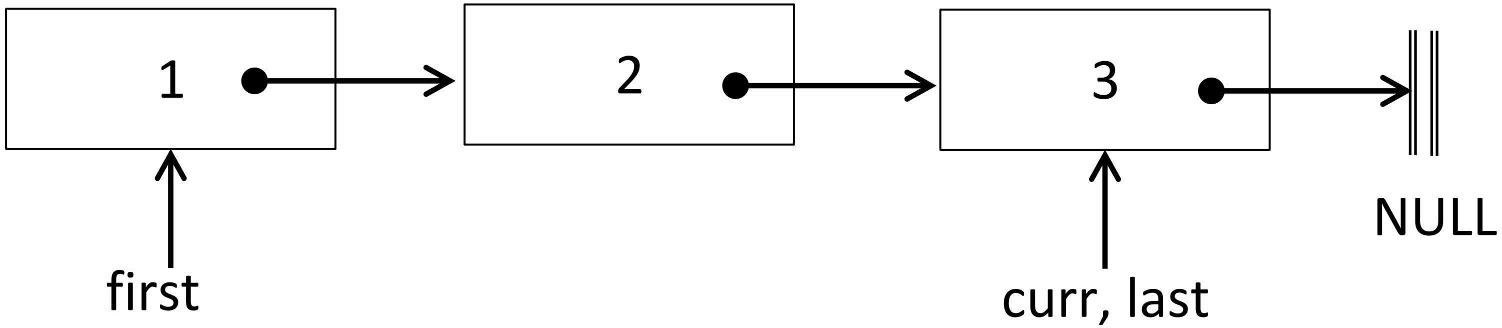
Example (Program chain.c)

```
curr = first;
while (curr != NULL) {
    printf ("Chain num %d -> ", curr->data);
    curr = curr->next;
}
```



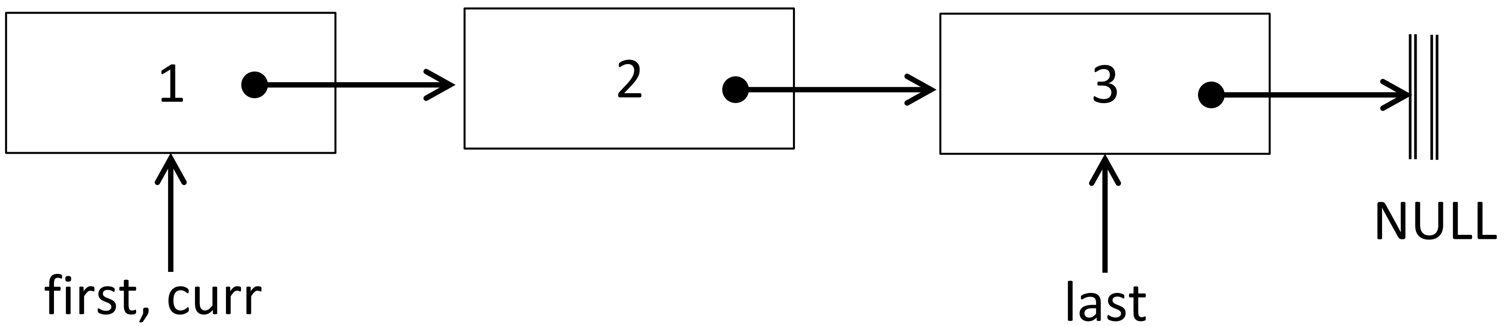
Example (Program chain.c)

```
curr = first;
while (curr != NULL) {
   printf ("Chain num %d -> ", curr->data);
   curr = curr->next;
}
```



Example (Program chain.c)

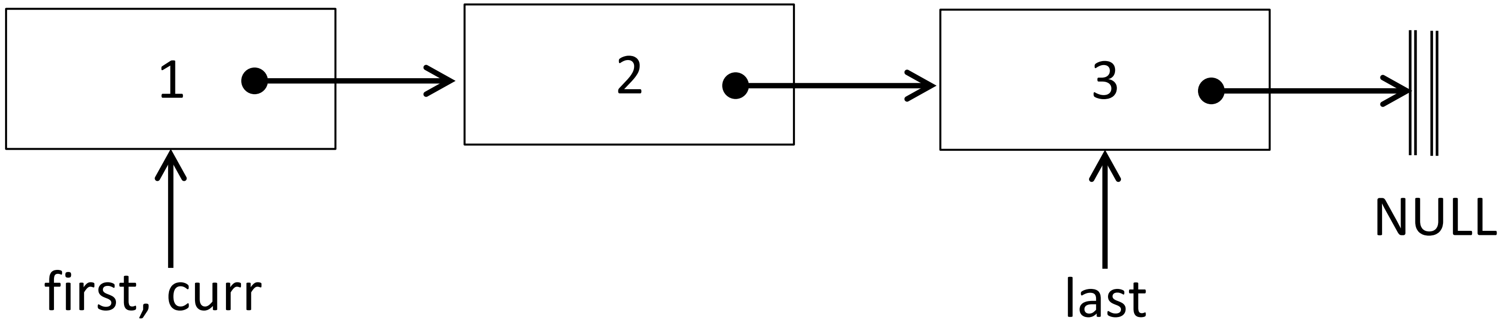
```
curr = first;
while (curr != NULL) {
    printf ("Chain num %d -> ", curr->data);
    curr = curr->next;
}
```



Example (Program chain.c)

```
curr = first;
while (curr != NULL) {
    printf ("Chain num %d -> ", curr->data);
    curr = curr->next;
}
```

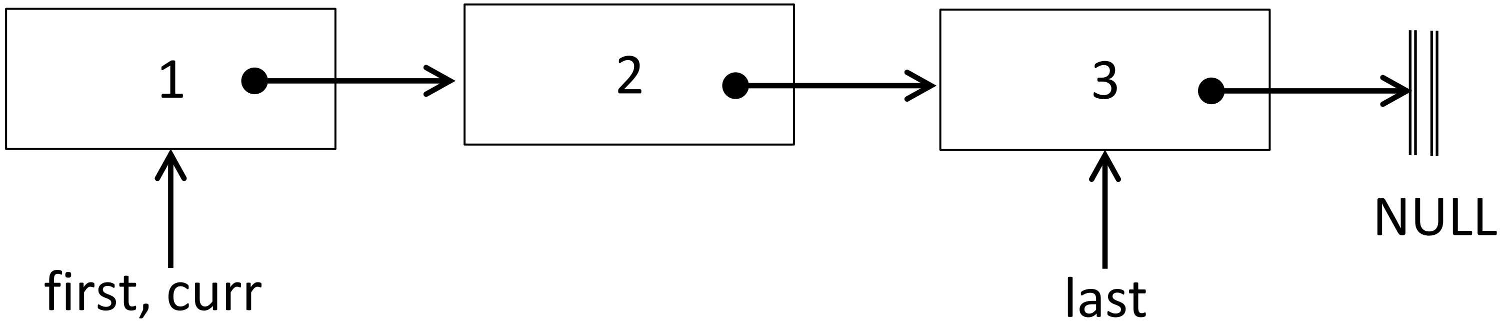
Chain num 1 ->



Example (Program chain.c)

```
curr = first;
while (curr != NULL) {
    printf ("Chain num %d -> ", curr->data);
    curr = curr->next;
}
```

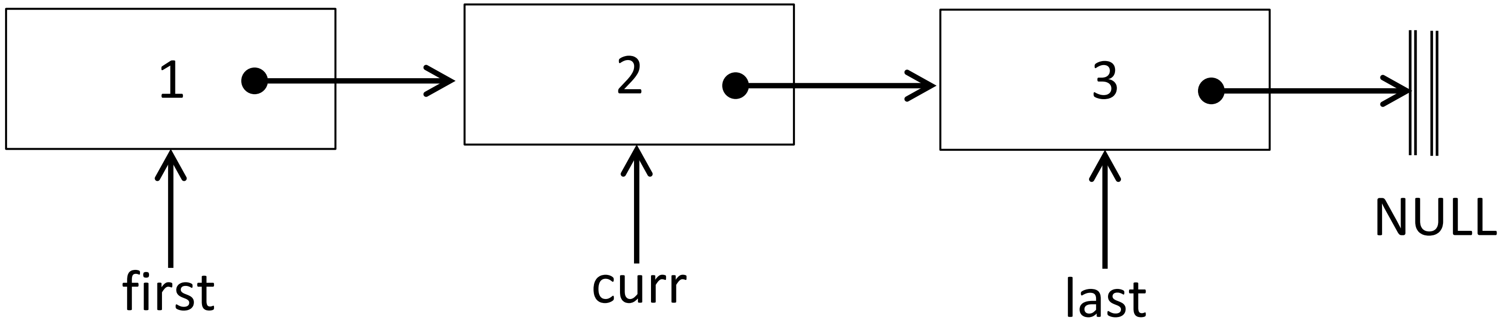
Chain num 1 ->



Example (Program chain.c)

```
curr = first;
while (curr != NULL) {
    printf ("Chain num %d -> ", curr->data);
    curr = curr->next;
}
```

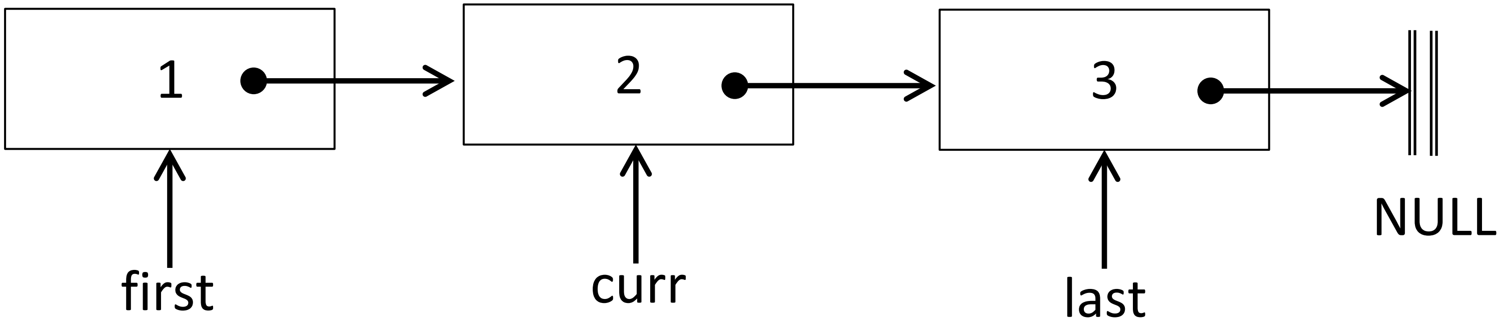
```
Chain num 1 ->
```



Example (Program chain.c)

```
curr = first;
while (curr != NULL) {
   printf ("Chain num %d -> ", curr->data);
   curr = curr->next;
}
```

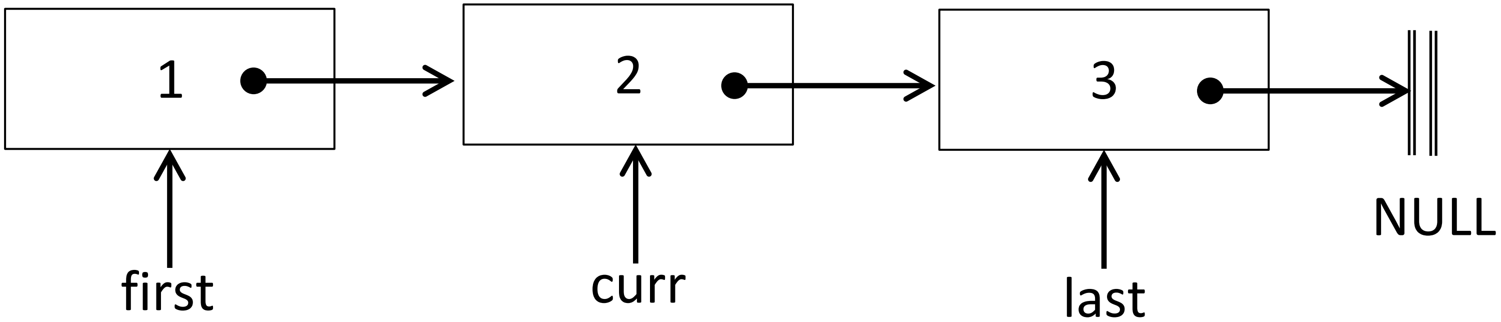
```
Chain num 1 ->
```



Example (Program chain.c)

```
curr = first;
while (curr != NULL) {
   printf ("Chain num %d -> ", curr->data);
   curr = curr->next;
}
```

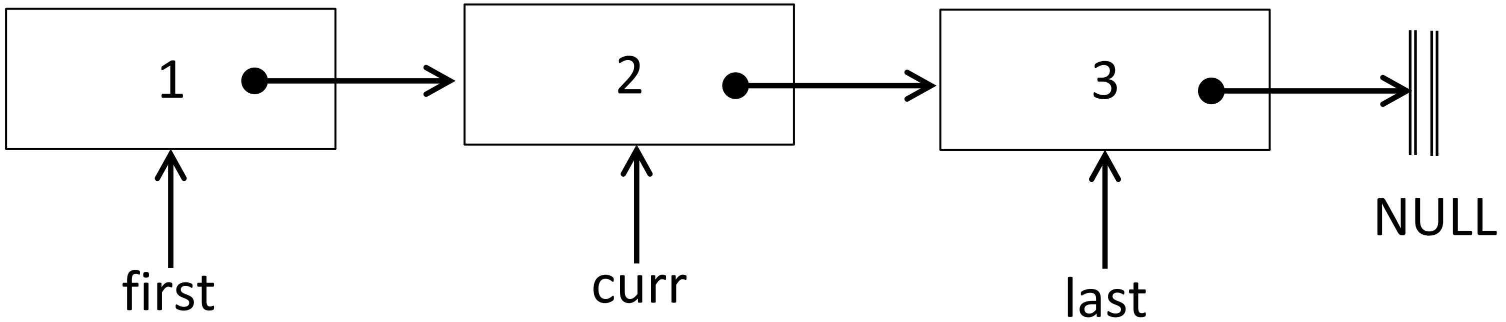
Chain num 1 -> Chain num 2 ->



Example (Program chain.c)

```
curr = first;
while (curr != NULL) {
    printf ("Chain num %d -> ", curr->data);
    curr = curr->next;
}
```

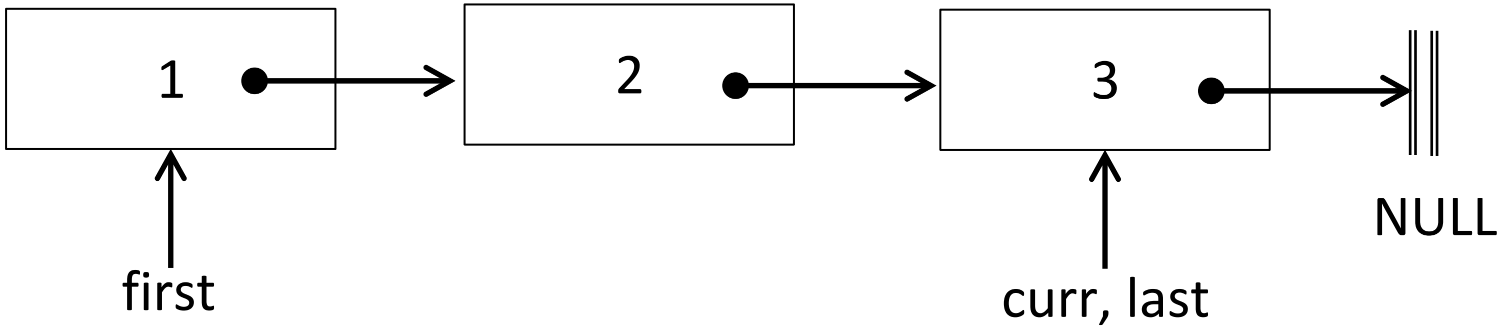
Chain num 1 -> Chain num 2 ->



Example (Program chain.c)

```
curr = first;
while (curr != NULL) {
    printf ("Chain num %d -> ", curr->data);
    curr = curr->next;
}
```

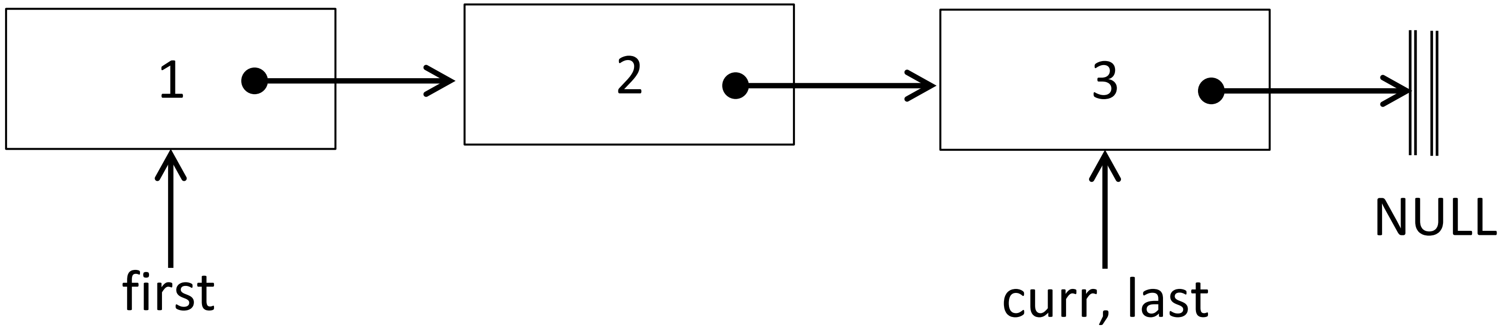
```
Chain num 1 -> Chain num 2 ->
```



Example (Program chain.c)

```
curr = first;
while (curr != NULL) {
   printf ("Chain num %d -> ", curr->data);
   curr = curr->next;
}
```

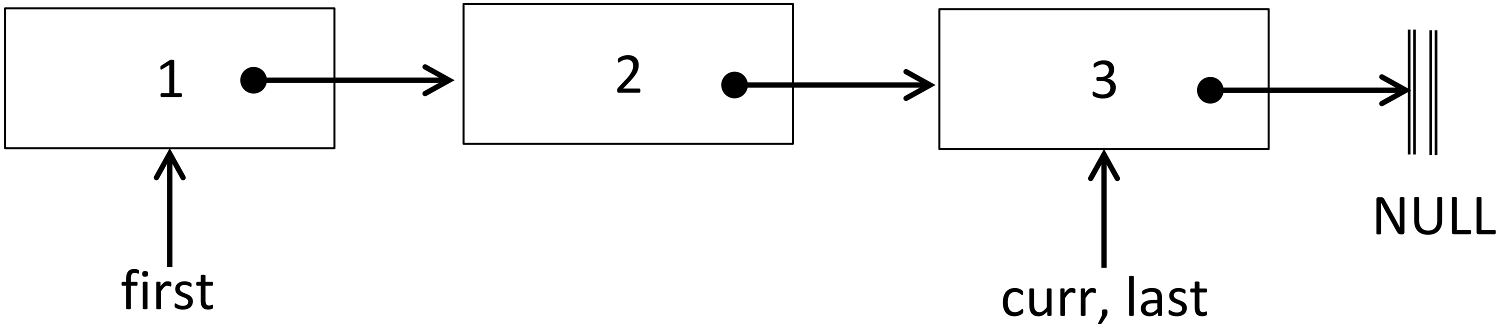
```
Chain num 1 -> Chain num 2 ->
```



Example (Program chain.c)

```
curr = first;
while (curr != NULL) {
    printf ("Chain num %d -> ", curr->data);
    curr = curr->next;
}
```

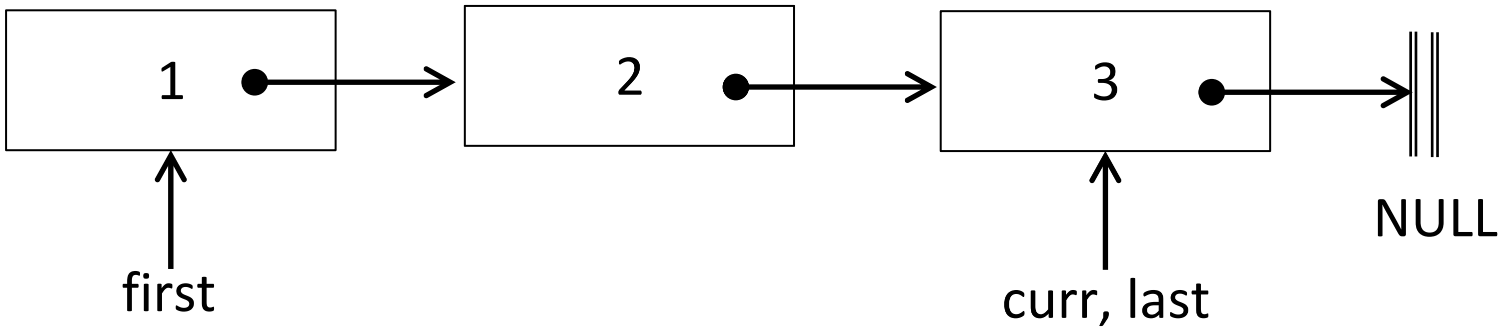
```
Chain num 1 -> Chain num 2 -> Chain num 3 ->
```



Example (Program chain.c)

```
curr = first;
while (curr != NULL) {
    printf ("Chain num %d -> ", curr->data);
    curr = curr->next;
}
```

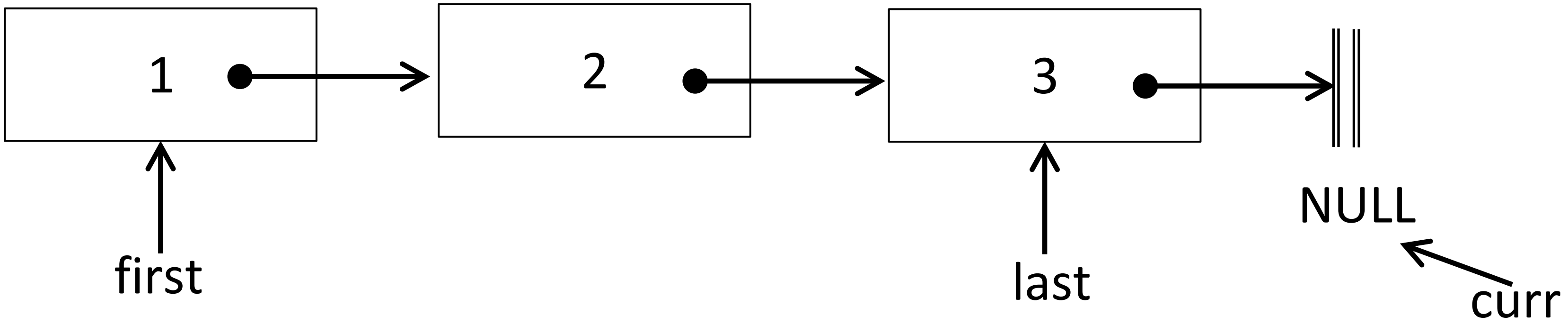
```
Chain num 1 -> Chain num 2 -> Chain num 3 ->
```



Example (Program chain.c)

```
curr = first;
while (curr != NULL) {
    printf ("Chain num %d -> ", curr->data);
    curr = curr->next;
}
```

```
Chain num 1 -> Chain num 2 -> Chain num 3 ->
```

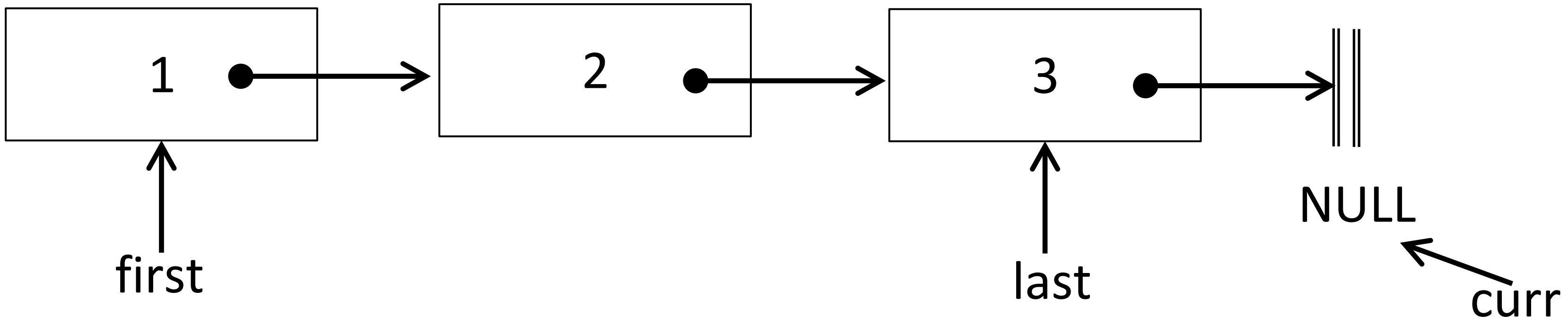


Example (Program chain.c)

```
curr = first;
while (curr != NULL) {
    printf ("Chain num %d -> ", curr->data);
    curr = curr->next;
}
```

When curr == NULL it means we reached the end of the list

Chain num 1 -> Chain num 2 -> Chain num 3 ->



Removing Elements

Releasing Memory

- Once the data is no longer needed it should be released back into the heap for later use

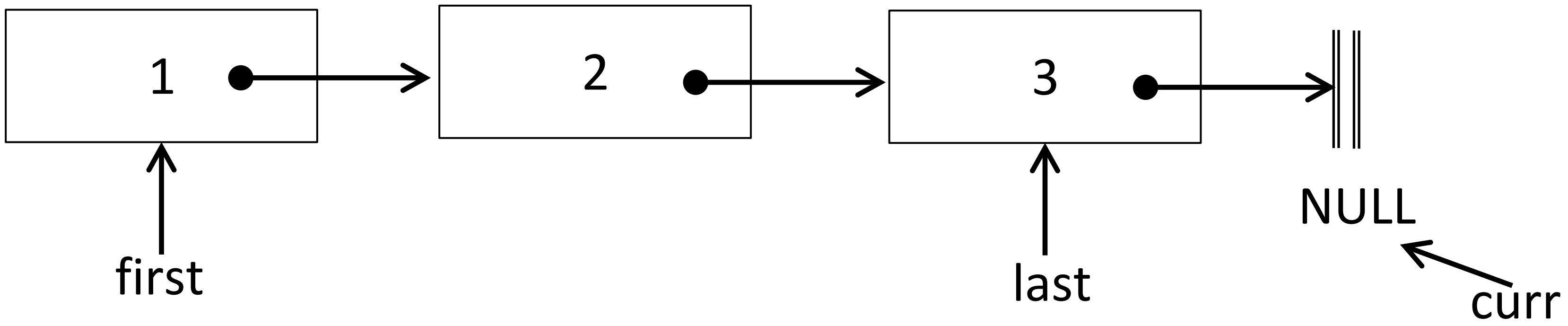
- This is done using the free function, passing it the same address that was returned by malloc

```
void free (void *);
```

- If allocated data is not freed the program might run out of heap memory and be unable to continue

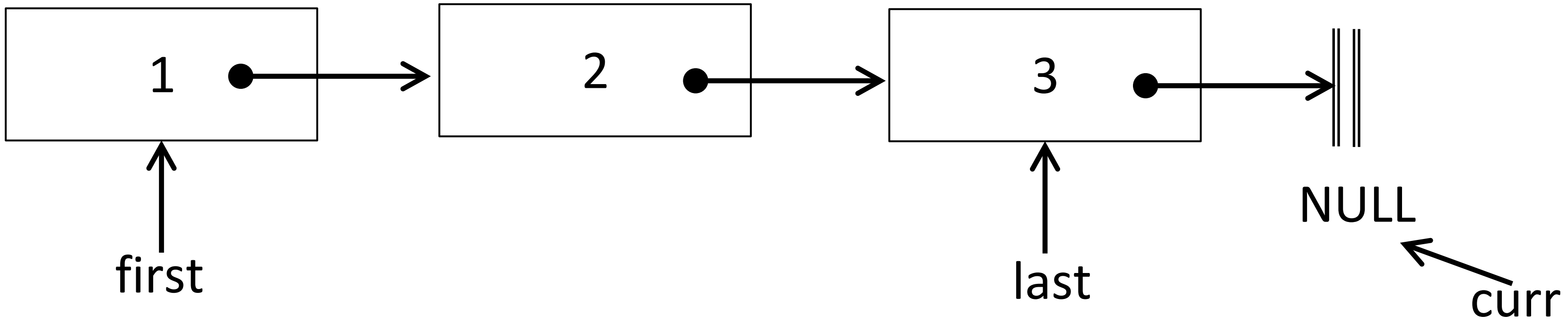
Example (Program chain.c)

```
printf("\n\n");
curr = first;
while (curr != NULL) {
   printf ("freeing %d ->", curr->data);
   first= curr->next;
   free(curr);
   curr = first;
 }
```



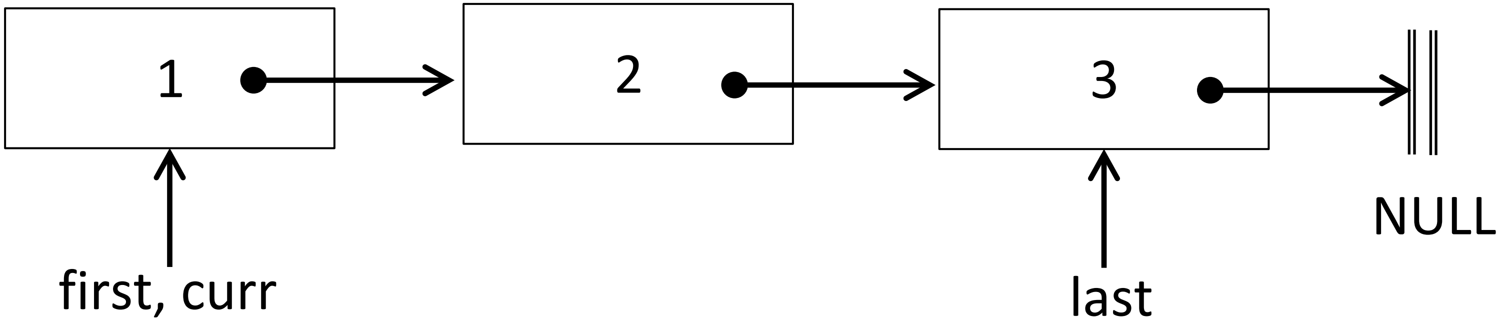
Example (Program chain.c)

```
printf("\n\n");
curr = first;
while (curr != NULL) {
   printf ("freeing %d ->", curr->data);
   first= curr->next;
   free(curr);
   curr = first;
 }
```



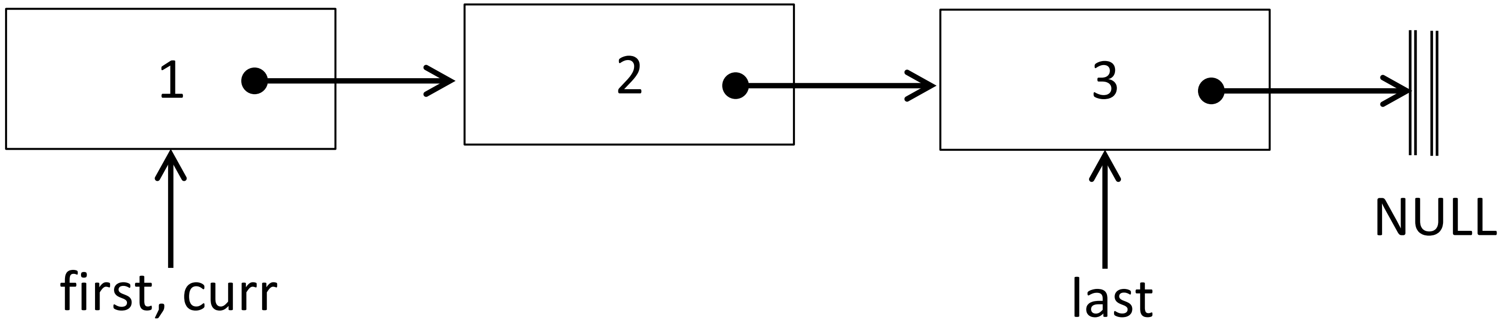
Example (Program chain.c)

```
printf("\n\n");
curr = first;
while (curr != NULL) {
   printf ("freeing %d ->", curr->data);
   first= curr->next;
   free(curr);
   curr = first;
 }
```



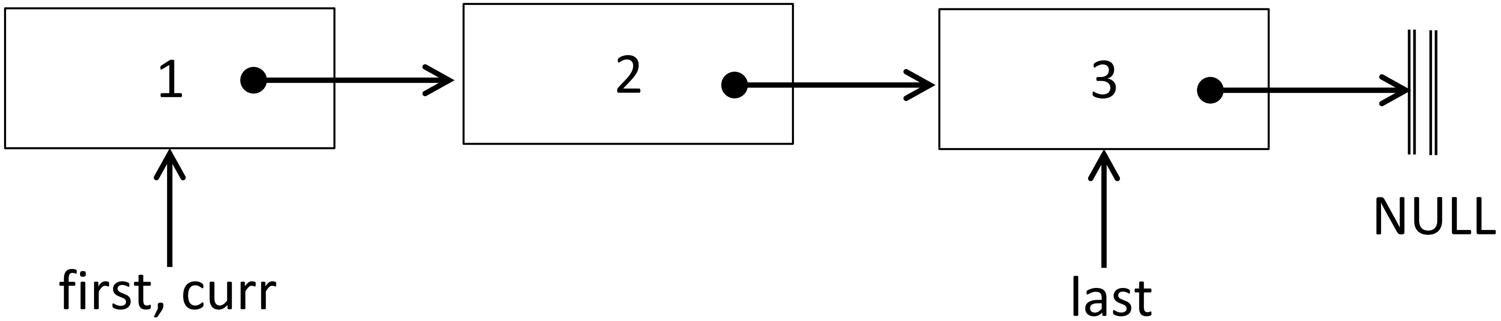
Example (Program chain.c)

```
printf("\n\n");
curr = first;
while (curr != NULL) {
   printf ("freeing %d ->", curr->data);
   first= curr->next;
   free(curr);
   curr = first;
 }
```



Example (Program chain.c)

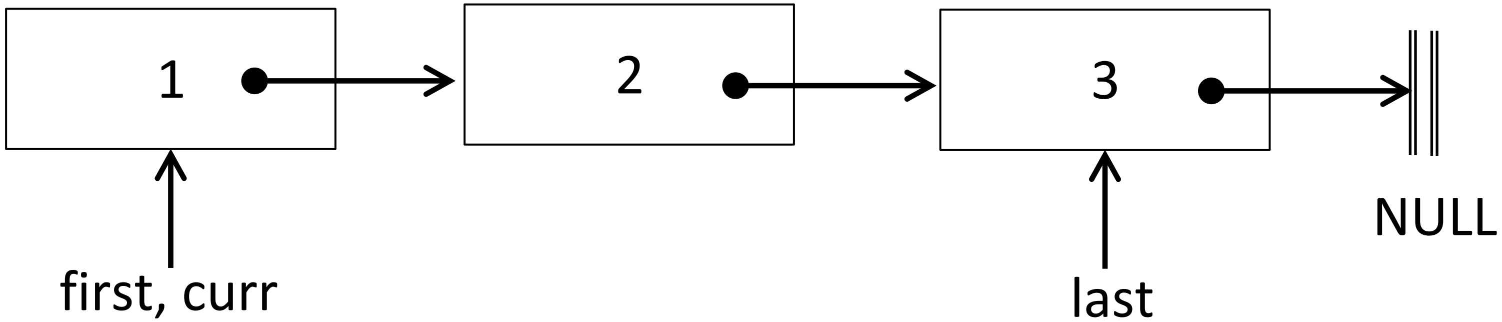
```
printf("\n\n");
curr = first;
while (curr != NULL) {
   printf ("freeing %d ->", curr->data);
   first= curr->next;
   free(curr);
   curr = first;
 }
```



Example (Program chain.c)

```
printf("\n\n");
curr = first;
while (curr != NULL) {
   printf ("freeing %d ->", curr->data);
   first= curr->next;
   free(curr);
   curr = first;
 }
```

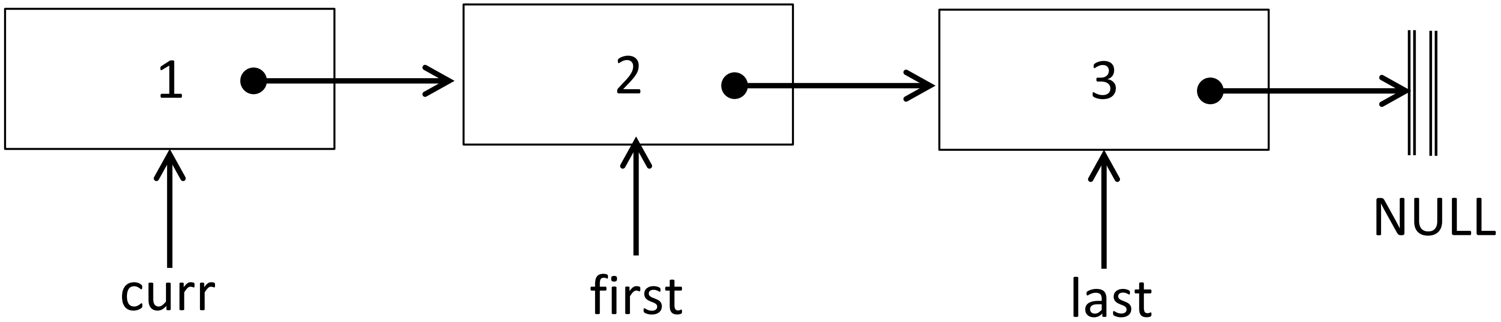
freeing 1 ->



Example (Program chain.c)

```
printf("\n\n");
curr = first;
while (curr != NULL) {
   printf ("freeing %d ->", curr->data);
   first= curr->next;
   free(curr);
   curr = first;
 }
```

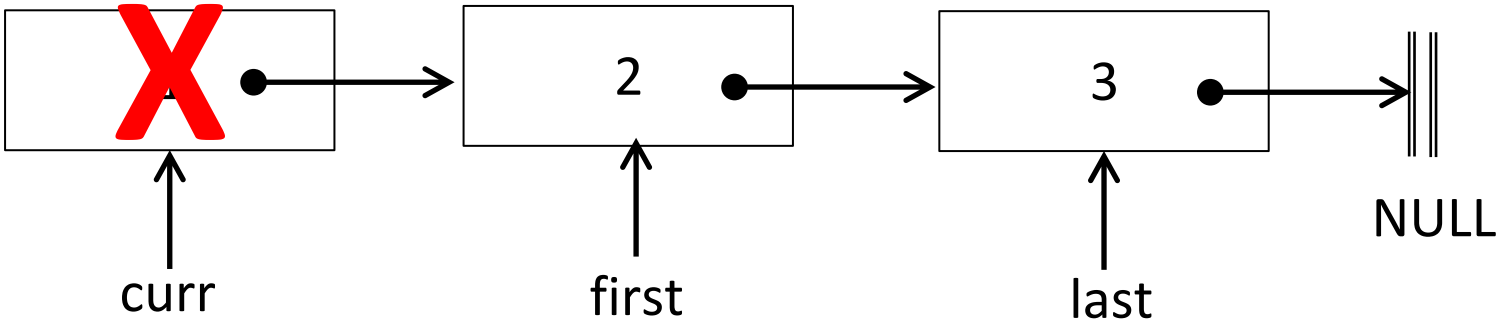
freeing 1 ->



Example (Program chain.c)

```
printf("\n\n");
curr = first;
while (curr != NULL) {
   printf ("freeing %d ->", curr->data);
   first= curr->next;
   free(curr);
   curr = first;
 }
```

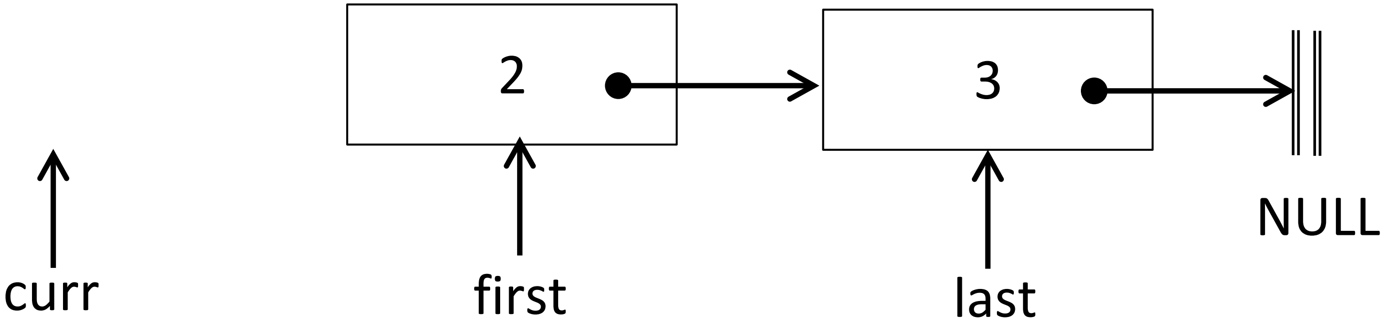
freeing 1 ->



Example (Program chain.c)

```
printf("\n\n");
curr = first;
while (curr != NULL) {
   printf ("freeing %d ->", curr->data);
   first= curr->next;
   free(curr);
   curr = first;
 }
```

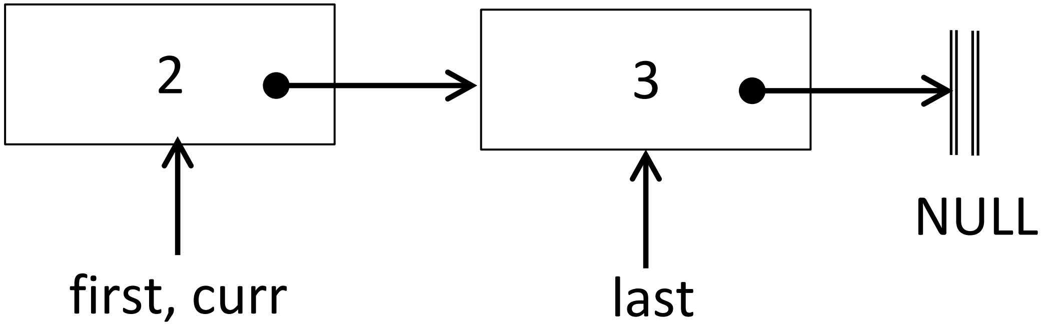
```
freeing 1 ->
```



Example (Program chain.c)

```
printf("\n\n");
curr = first;
while (curr != NULL) {
   printf ("freeing %d ->", curr->data);
   first= curr->next;
   free(curr);
   curr = first;
 }
```

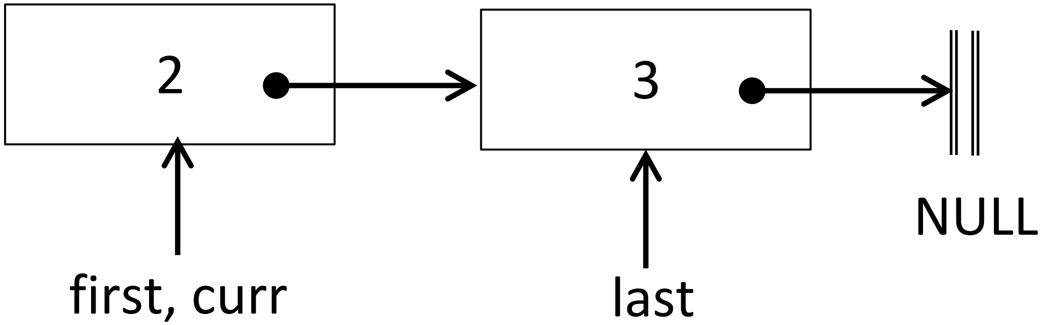
freeing 1 ->



Example (Program chain.c)

```
printf("\n\n");
curr = first;
while (curr != NULL) {
   printf ("freeing %d ->", curr->data);
   first= curr->next;
   free(curr);
   curr = first;
 }
```

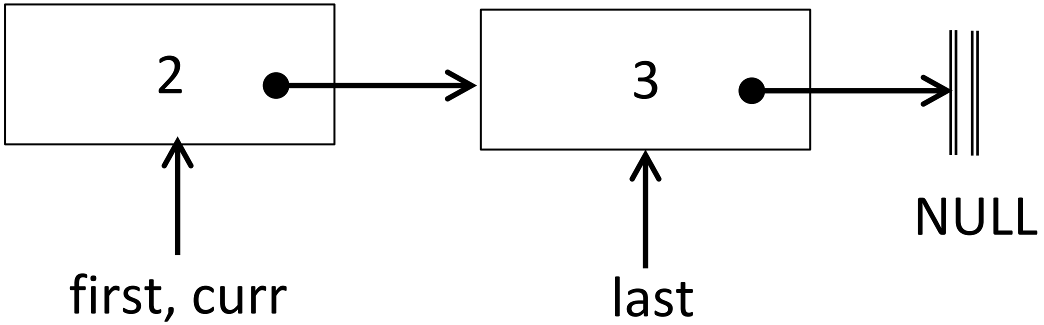
freeing 1 ->



Example (Program chain.c)

```
printf("\n\n");
curr = first;
while (curr != NULL) {
   printf ("freeing %d ->", curr->data);
   first= curr->next;
   free(curr);
   curr = first;
 }
```

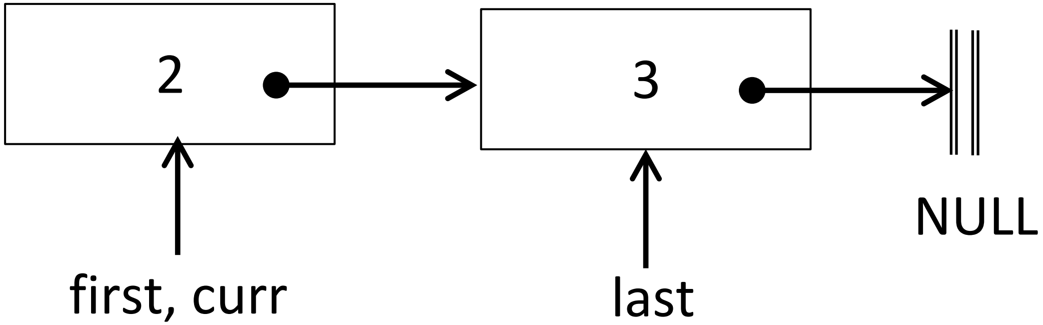
```
freeing 1 ->
```



Example (Program chain.c)

```
printf("\n\n");
curr = first;
while (curr != NULL) {
   printf ("freeing %d ->", curr->data);
   first= curr->next;
   free(curr);
   curr = first;
 }
```

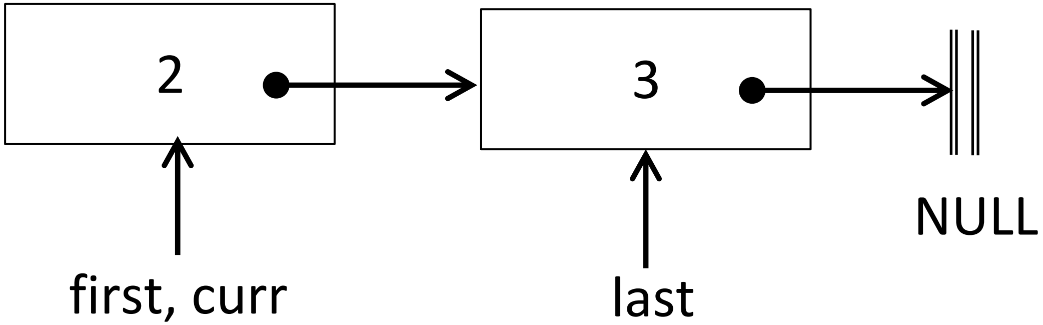
freeing 1 -> freeing 2 ->



Example (Program chain.c)

```
printf("\n\n");
curr = first;
while (curr != NULL) {
   printf ("freeing %d ->", curr->data);
   first= curr->next;
   free(curr);
   curr = first;
 }
```

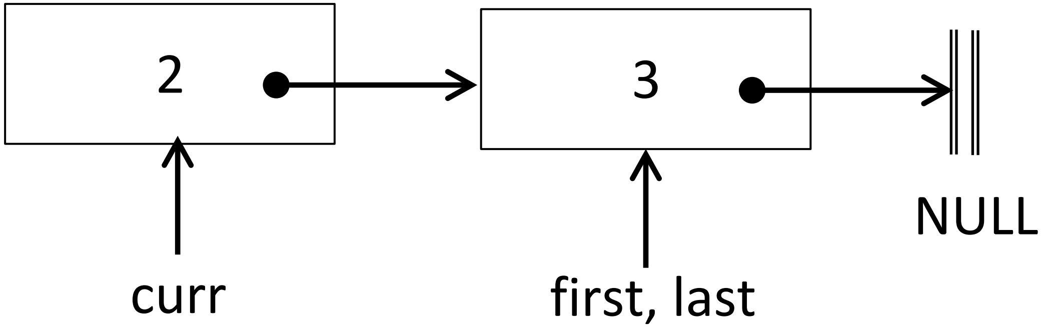
```
freeing 1 -> freeing 2 ->
```



Example (Program chain.c)

```
printf("\n\n");
curr = first;
while (curr != NULL) {
   printf ("freeing %d ->", curr->data);
   first= curr->next;
   free(curr);
   curr = first;
 }
```

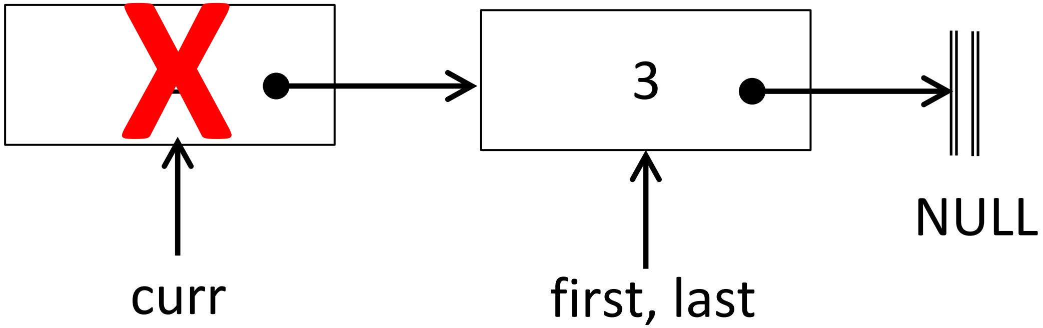
freeing 1 -> freeing 2 ->



Example (Program chain.c)

```
printf("\n\n");
curr = first;
while (curr != NULL) {
   printf ("freeing %d ->", curr->data);
   first= curr->next;
   free(curr);
   curr = first;
 }
```

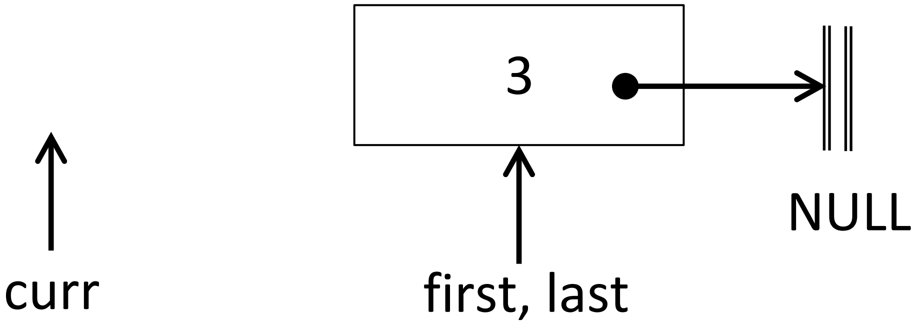
freeing 1 -> freeing 2 ->



Example (Program chain.c)

```
printf("\n\n");
curr = first;
while (curr != NULL) {
   printf ("freeing %d ->", curr->data);
   first= curr->next;
   free(curr);
   curr = first;
 }
```

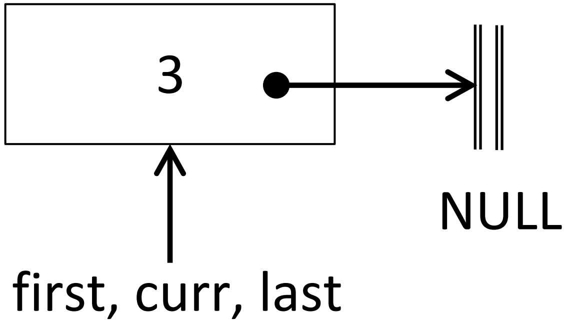
freeing 1 -> freeing 2 ->



Example (Program chain.c)

```
printf("\n\n");
curr = first;
while (curr != NULL) {
   printf ("freeing %d ->", curr->data);
   first= curr->next;
   free(curr);
   curr = first;
 }
```

```
freeing 1 -> freeing 2 ->
```

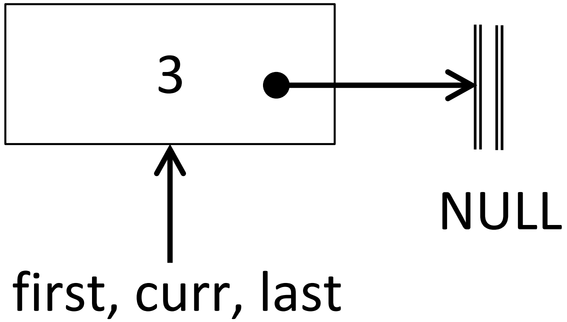


Example (Program chain.c)

```
printf("\n\n");
curr = first;
while (curr != NULL) {
   printf ("freeing %d ->", curr->data);
   first= curr->next;
   free(curr);
   curr = first;
 }
```

```
freeing 1 -> freeing 2 ->
```

… continues until all elements of the chain are deleted and first, curr, and last will all point to NULL



Linked Lists vs Array

- A linked list can only be accessed **sequentially**
  - To find the 5th element, for instance, you must start from the head and follow the links through 4 other nodes

- **Advantages of linked lists**
  - Dynamic size
  - Easy to add additional nodes as needed
  - Easy to add and remove nodes from the middle of the list

- **Advantages of using arrays**
  - Can easily and quickly access arbitrary elements

Stack

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

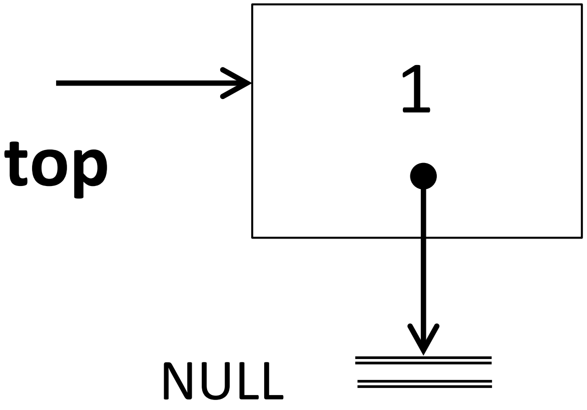
An ordered collection of items where the addition of new items and the removal of existing items always takes places at the same end (the top).

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

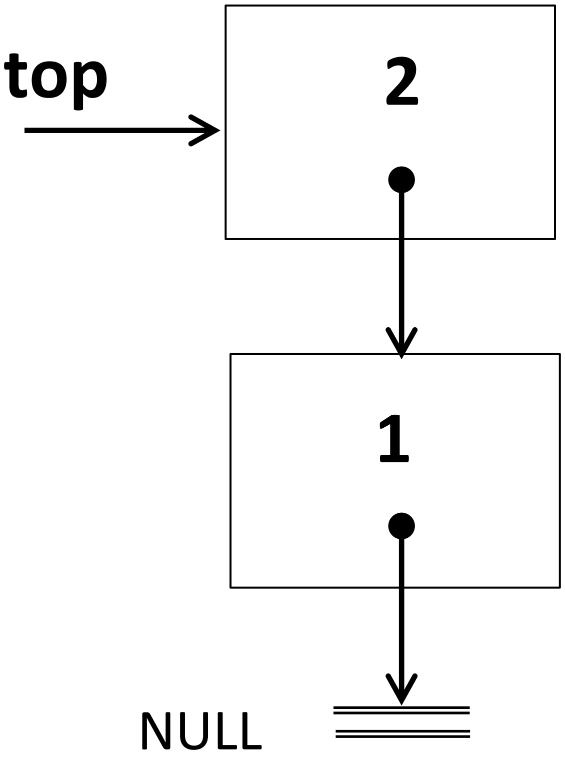
- **LIFO (last-in first-out) ordering principle**: the most recently added item is in the top position and it is to be removed first



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).

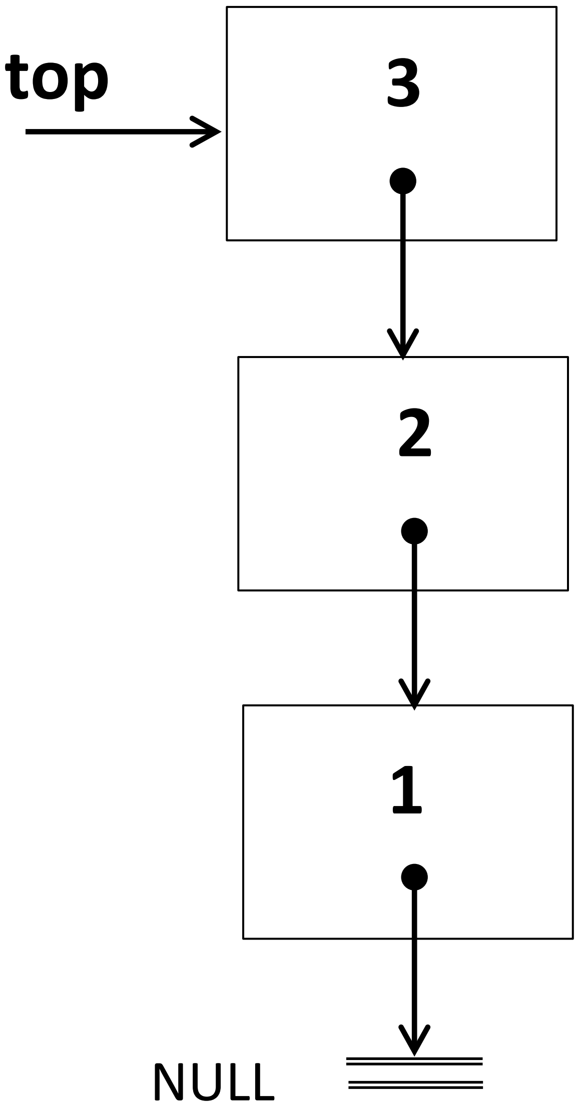
- **LIFO (last-in first-out) ordering principle**: the most recently added item is in the top position and it is to be removed first



Stack

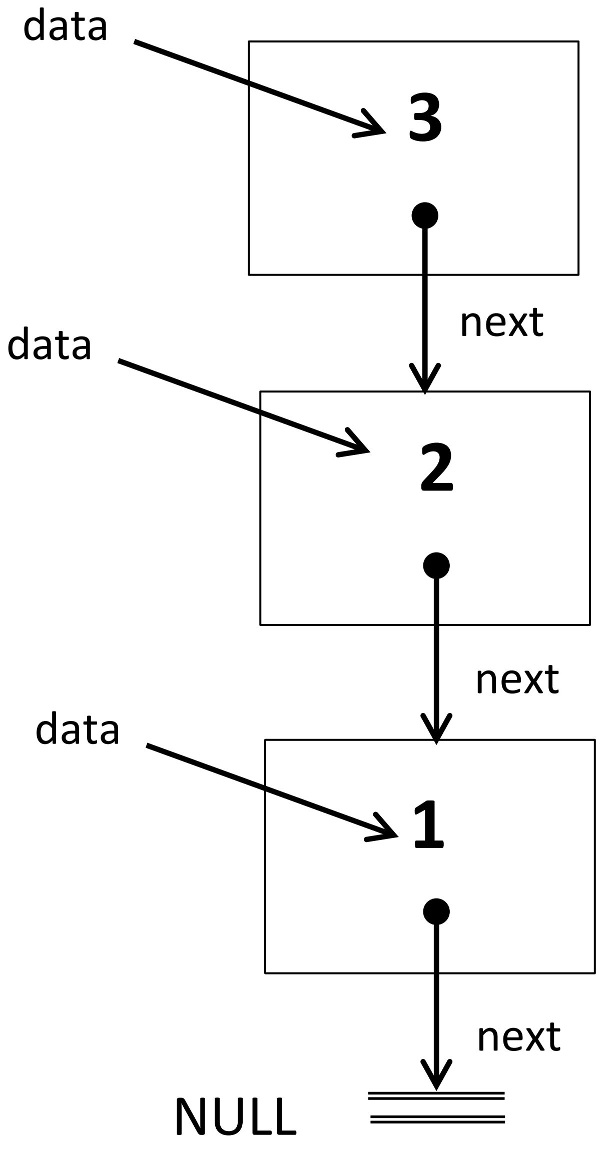
An ordered collection of items where the addition of new items and the removal of existing items always takes places at the same end (the top).

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



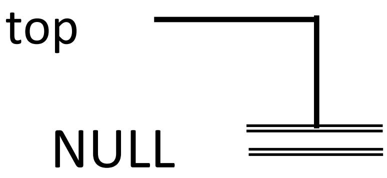
Structure Members

```
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);

}
```

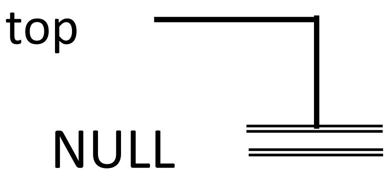


Example stack

```
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);
```



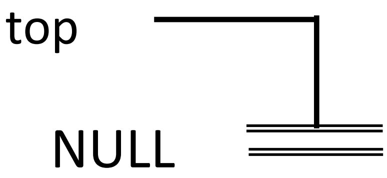
Example stack

```
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);
```



top

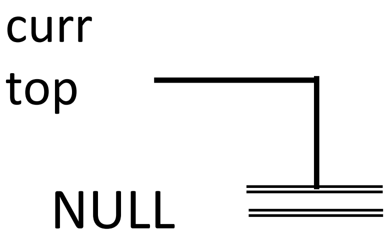
NULL

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);
```



curr

top

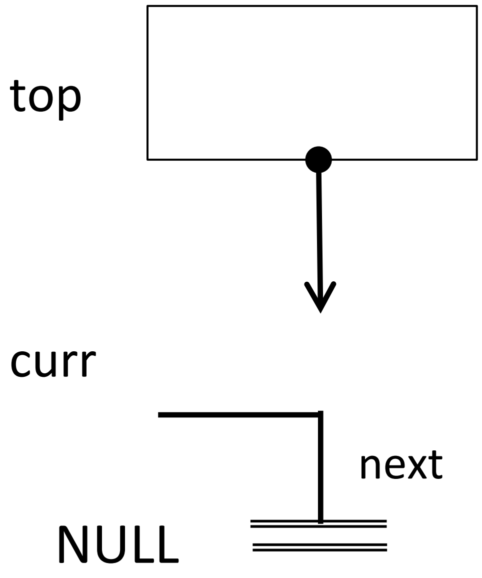
NULL

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);
```

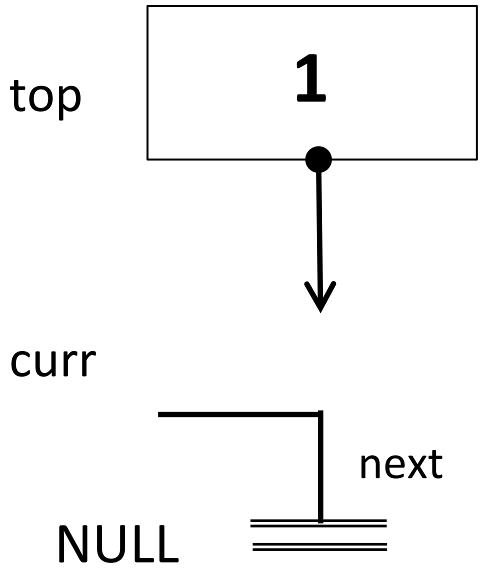


Example stack

```
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);
```

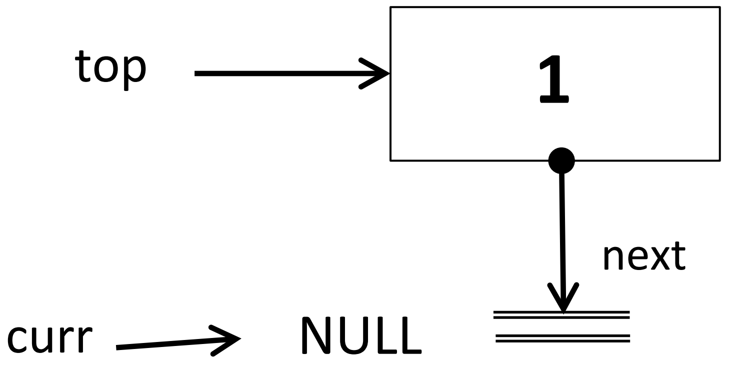


Example stack

```
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

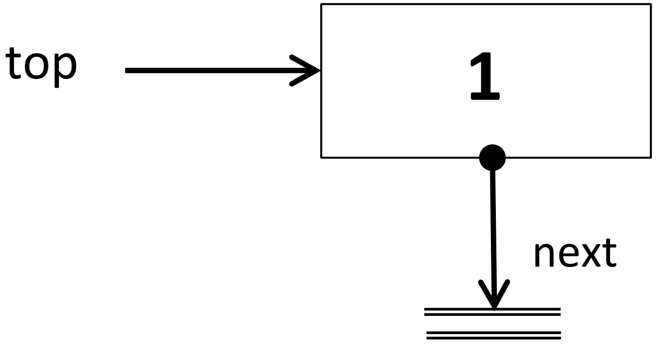
```
struct stack_elem * push(int value, struct stack_elem *top){
    struct stack_elem *curr = top;
    top = malloc(sizeof(stack));
    top->data = value;
    top->next = curr;
    return top;
}
```

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);
```

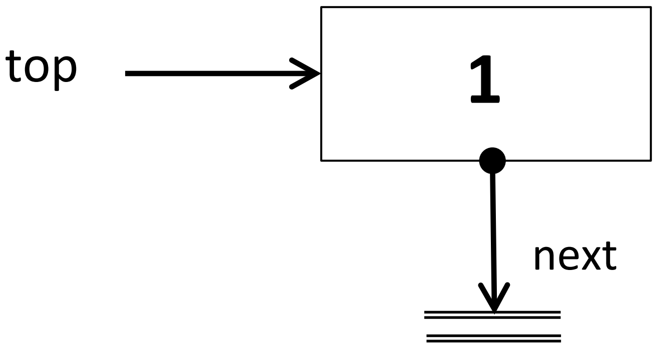


Example stack

```
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);
```

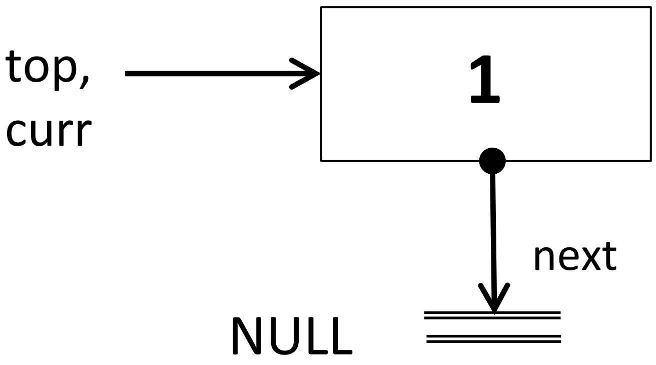


Example stack

```
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);
```



top,
curr

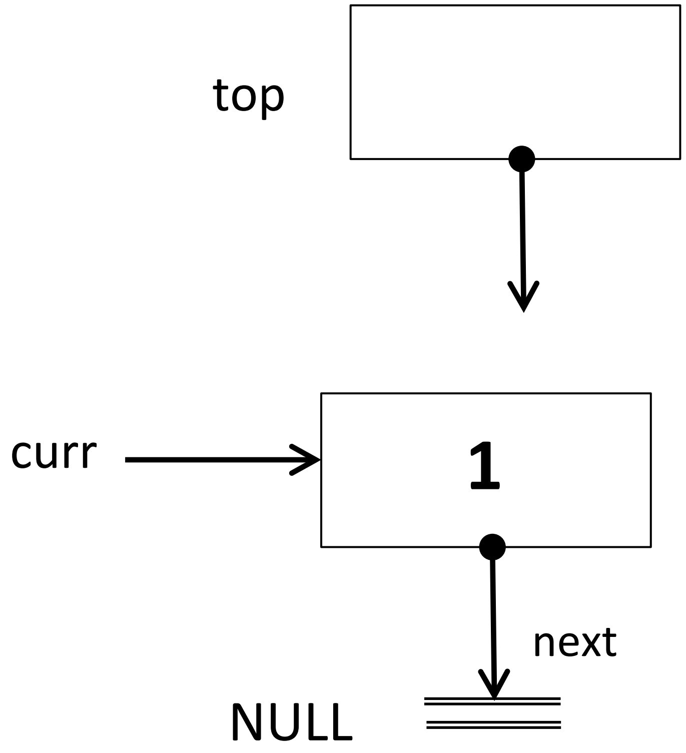
1

next

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

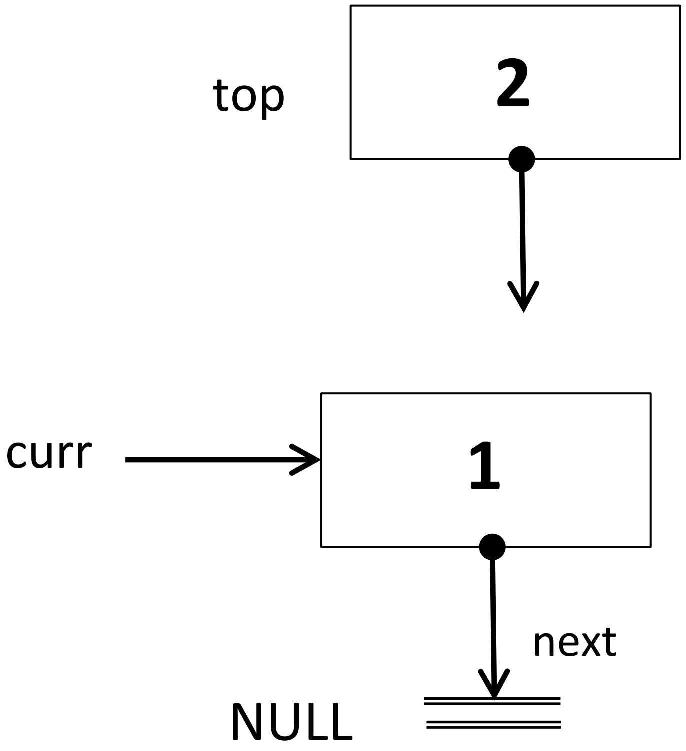
```
 top = push(1, top);
 printf("Stack Data: %d\n", top->data);

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);
```

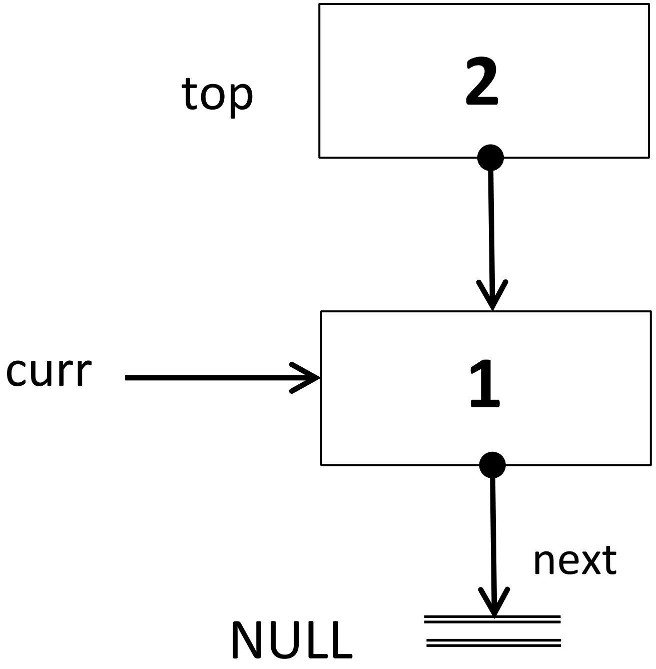


Example stack

```
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);
```

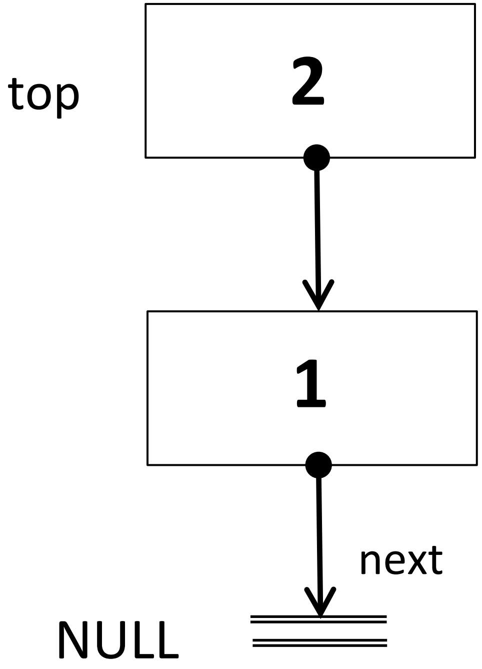


Example stack

```
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);
```

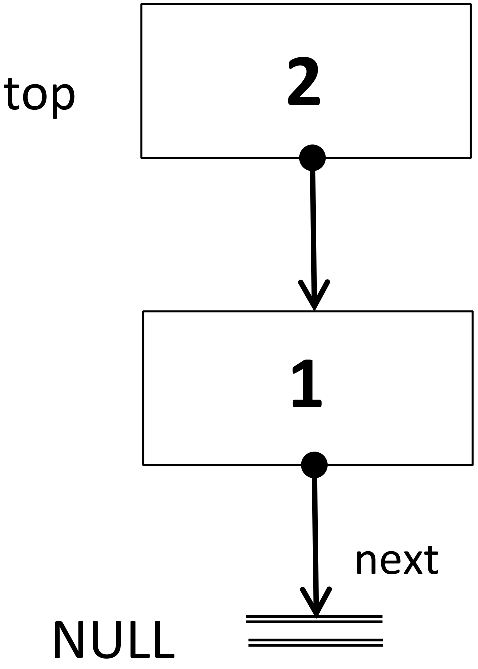


Example stack

```
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);
```

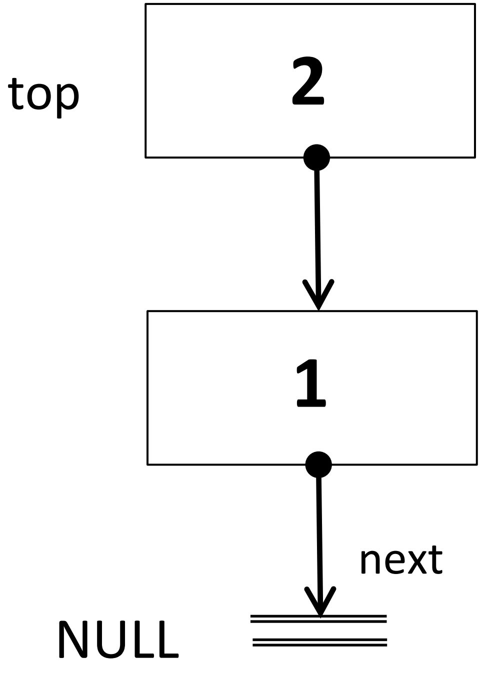


Example stack

```
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);
```

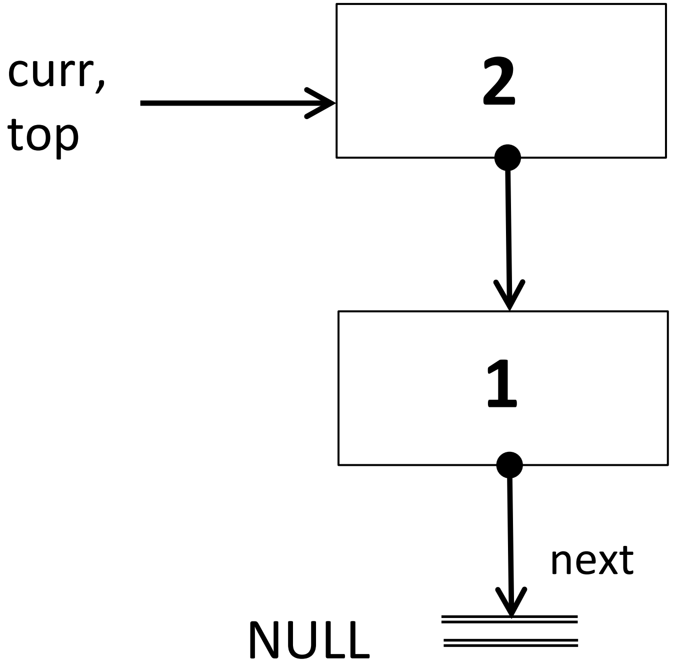


Example stack

```
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);
```

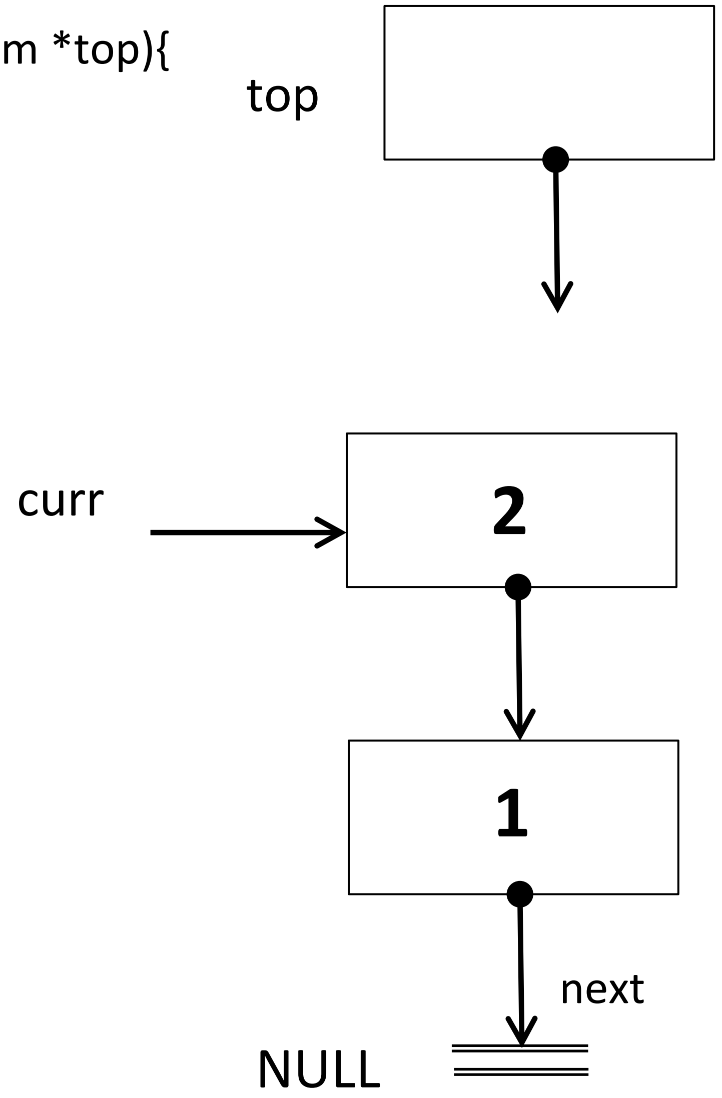


Example stack

```
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);
```

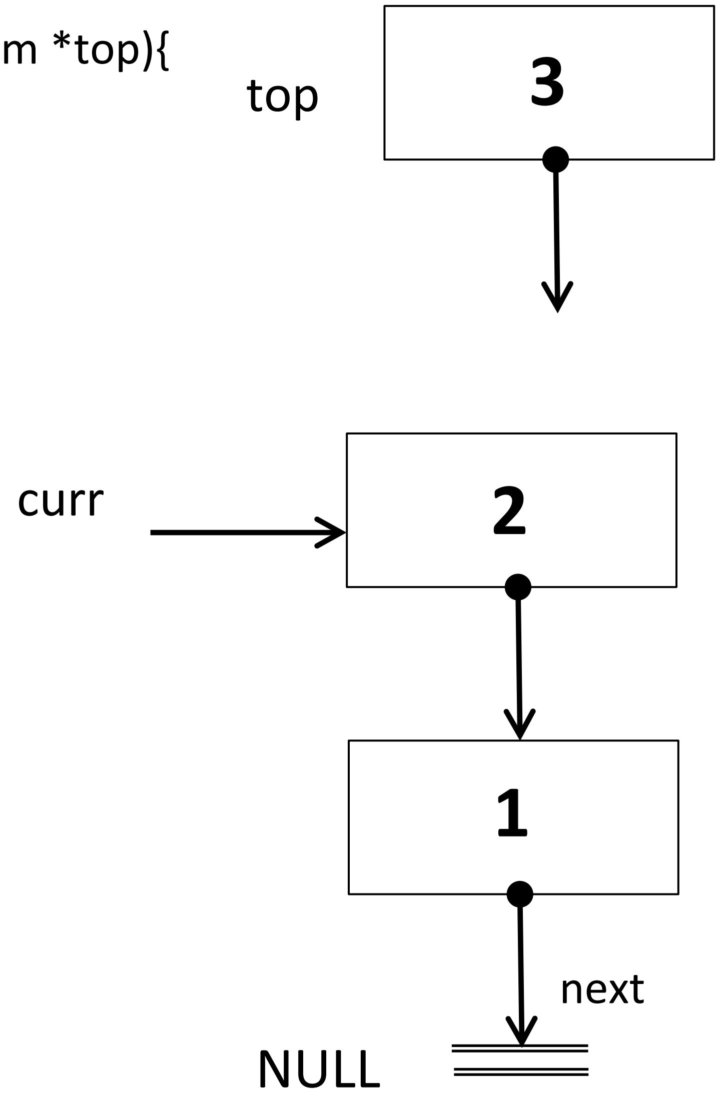


Example stack

```
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);
```

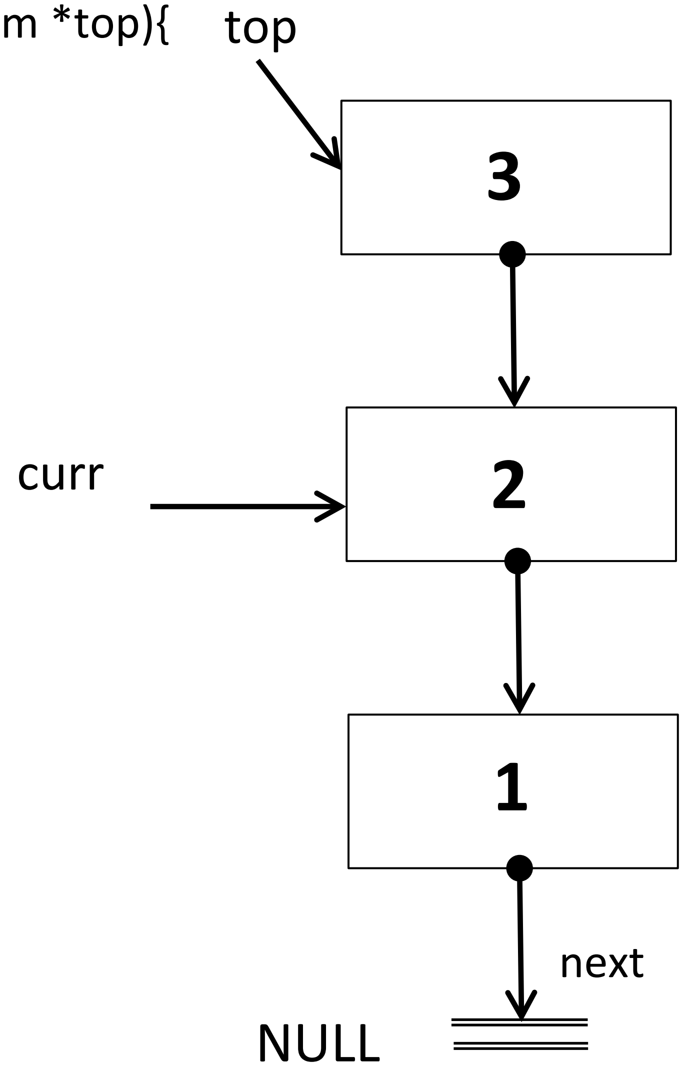


Example stack

```
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);
```

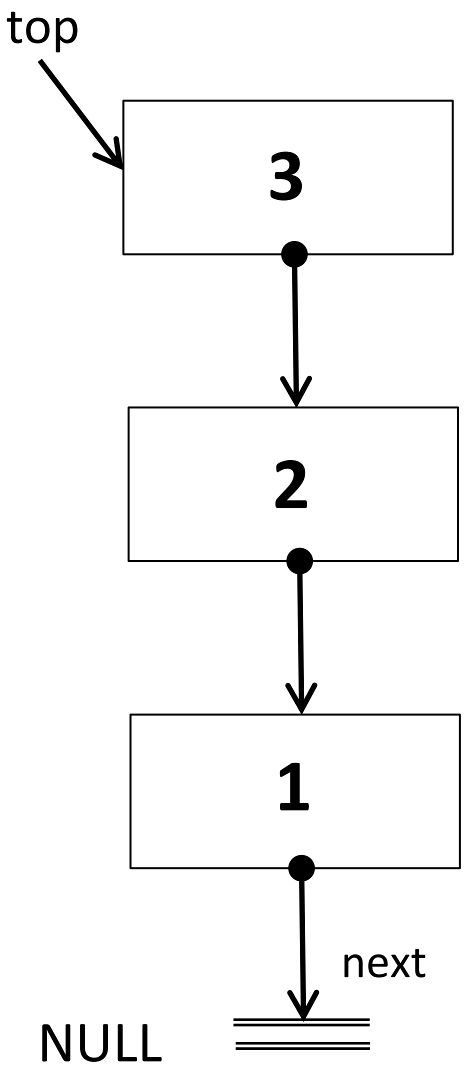


Example stack

```
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);
```



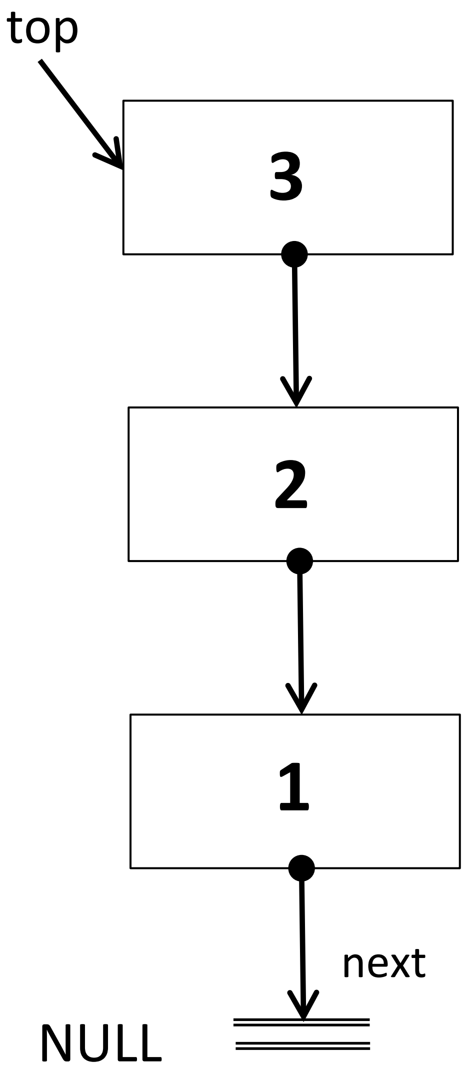
Removing Elements

Pop Elements from the Stack

```
struct stack_elem * pop(struct stack_elem *top){
    struct stack_elem *curr = top;
    if(curr!=NULL){
        top = curr->next;
        printf("Stack Data: %d\n", curr->data);
        free(curr);
    }
    return top;
}
```

main.c

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

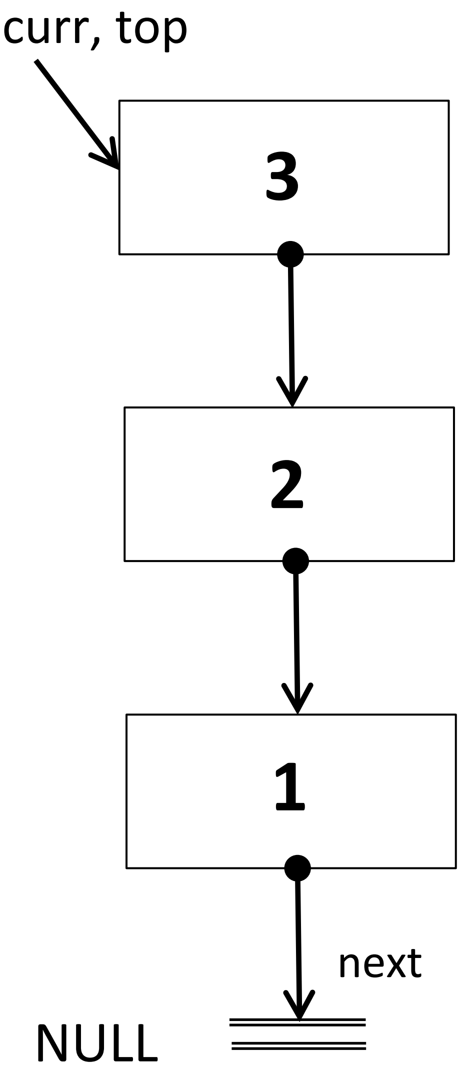


Pop Elements from the Stack

```
struct stack_elem * pop(struct stack_elem *top){
    struct stack_elem *curr = top;
    if(curr!=NULL){
        top = curr->next;
        printf("Stack Data: %d\n", curr->data);
        free(curr);
    }
    return top;
}
```

main.c

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

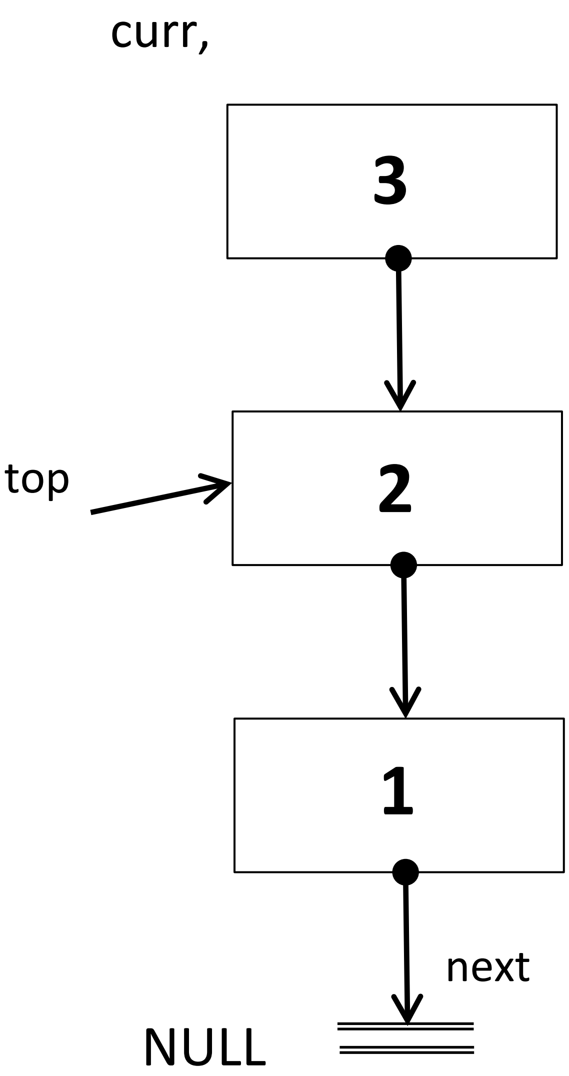


Pop Elements from the stack

```
struct stack_elem * pop(struct stack_elem *top){
    struct stack_elem *curr = top;
    if(curr!=NULL){
        top = curr->next;
        printf("Stack Data: %d\n", curr->data);
        free(curr);
    }
    return top;
}
```

main.c

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



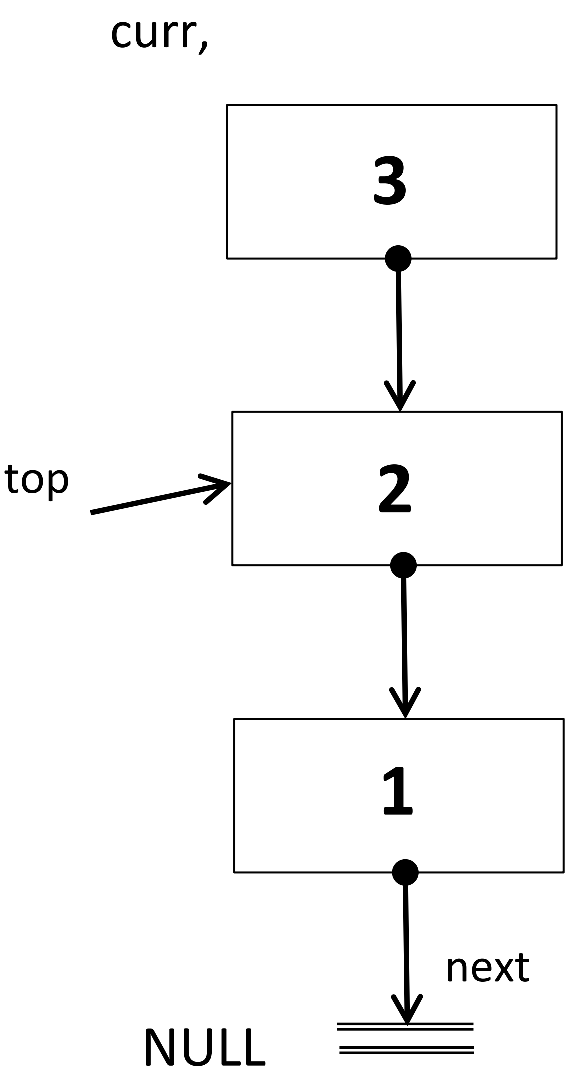
Pop Elements from the stack

```
struct stack_elem * pop(struct stack_elem *top){
    struct stack_elem *curr = top;
    if(curr!=NULL){
        top = curr->next;
        printf("Stack Data: %d\n", curr->data);
        free(curr);
    }
    return top;
}
```

main.c

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

```
Stack Data: 3
```



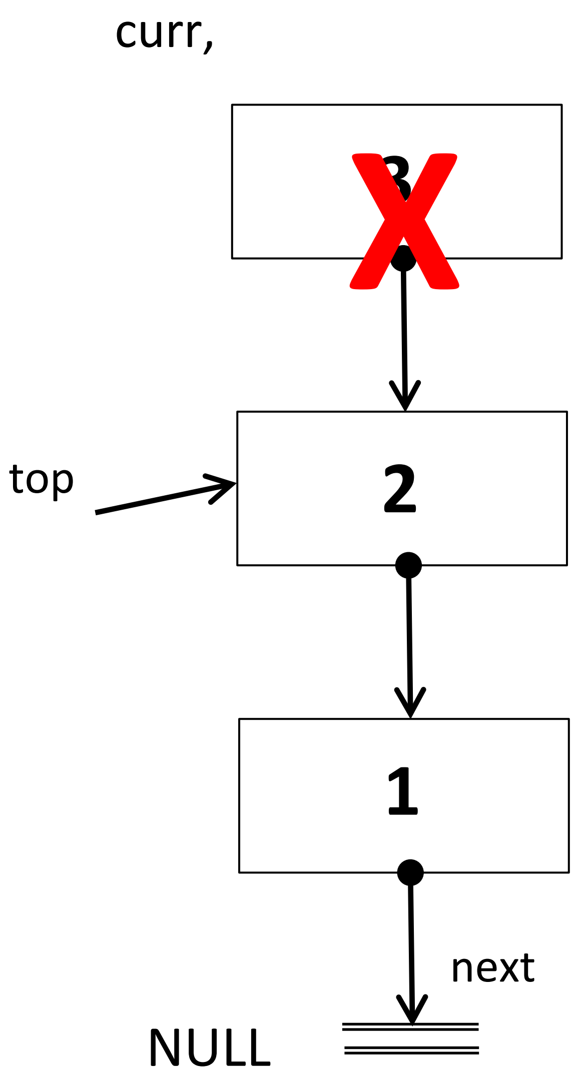
Pop Elements from the stack

```
struct stack_elem * pop(struct stack_elem *top){
    struct stack_elem *curr = top;
    if(curr!=NULL){
        top = curr->next;
        printf("Stack Data: %d\n", curr->data);
        free(curr);
    }
    return top;
}
```

main.c

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

```
Stack Data: 3
```



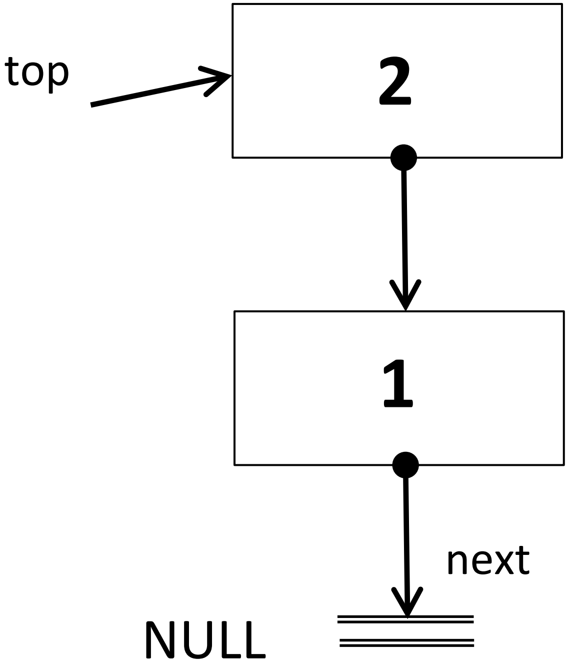
Pop Elements from the stack

```
struct stack_elem * pop(struct stack_elem *top){
    struct stack_elem *curr = top;
    if(curr!=NULL){
        top = curr->next;
        printf("Stack Data: %d\n", curr->data);
        free(curr);
    }
    return top;
}
```

main.c

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

Stack Data: 3



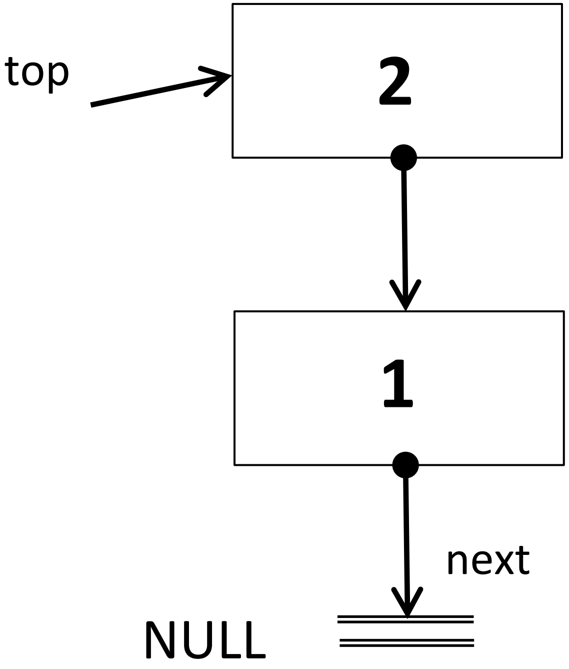
Pop Elements from the stack

```
struct stack_elem * pop(struct stack_elem *top){
    struct stack_elem *curr = top;
    if(curr!=NULL){
        top = curr->next;
        printf("Stack Data: %d\n", curr->data);
        free(curr);
    }
    return top;
}
```

main.c

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

```
Stack Data: 3
Stack Data: 2
```



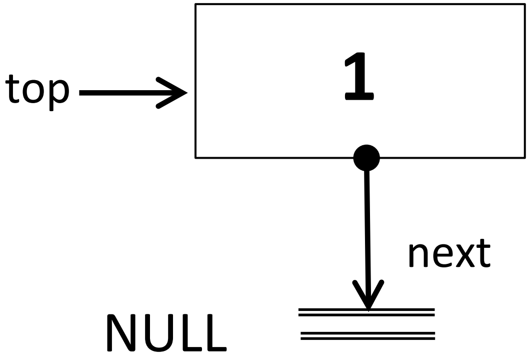
Pop Elements from the stack

```
struct stack_elem * pop(struct stack_elem *top){
    struct stack_elem *curr = top;
    if(curr!=NULL){
        top = curr->next;
        printf("Stack Data: %d\n", curr->data);
        free(curr);
    }
    return top;
}
```

main.c

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

```
Stack Data: 3
Stack Data: 2
```



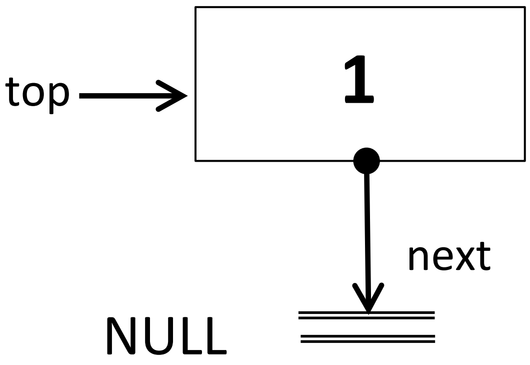
Pop Elements from the stack

```
struct stack_elem * pop(struct stack_elem *top){
    struct stack_elem *curr = top;
    if(curr!=NULL){
        top = curr->next;
        printf("Stack Data: %d\n", curr->data);
        free(curr);
    }
    return top;
}
```

main.c

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

```
Stack Data: 3
Stack Data: 2
```



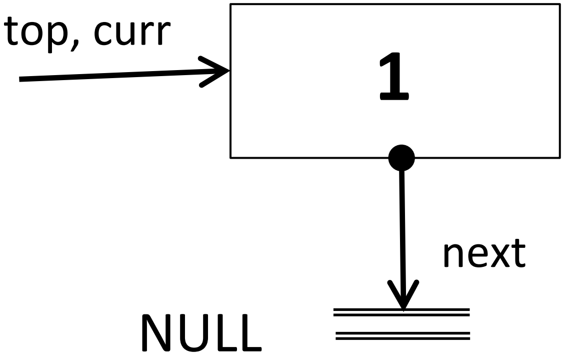
Pop Elements from the stack

```
struct stack_elem * pop(struct stack_elem *top){
    struct stack_elem *curr = top;
    if(curr!=NULL){
        top = curr->next;
        printf("Stack Data: %d\n", curr->data);
        free(curr);
    }
    return top;
}
```

main.c

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

```
Stack Data: 3
Stack Data: 2
```



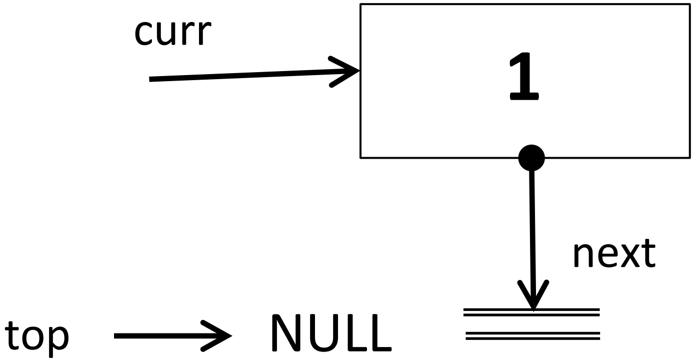
Pop Elements from the stack

```
struct stack_elem * pop(struct stack_elem *top){
    struct stack_elem *curr = top;
    if(curr!=NULL){
        top = curr->next;
        printf("Stack Data: %d\n", curr->data);
        free(curr);
    }
    return top;
}
```

main.c

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

```
Stack Data: 3
Stack Data: 2
```



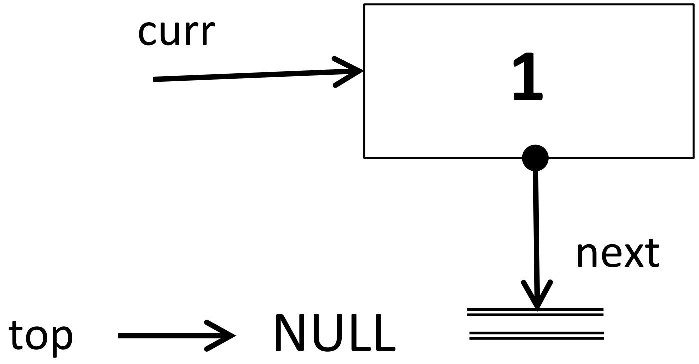
Pop Elements from the stack

```
struct stack_elem * pop(struct stack_elem *top){
    struct stack_elem *curr = top;
    if(curr!=NULL){
        top = curr->next;
        printf("Stack Data: %d\n", curr->data);
        free(curr);
    }
    return top;
}
```

main.c

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

```
Stack Data: 3
Stack Data: 2
Stack Data: 1
```



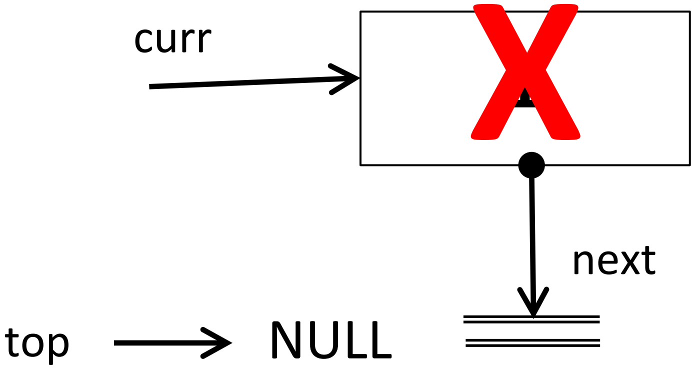
Pop Elements from the stack

```
struct stack_elem * pop(struct stack_elem *top){
    struct stack_elem *curr = top;
    if(curr!=NULL){
        top = curr->next;
        printf("Stack Data: %d\n", curr->data);
        free(curr);
    }
    return top;
}
```

main.c

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

```
Stack Data: 3
Stack Data: 2
Stack Data: 1
```



Pop Elements from the stack

```
struct stack_elem * pop(struct stack_elem *top){
    struct stack_elem *curr = top;
    if(curr!=NULL){
        top = curr->next;
        printf("Stack Data: %d\n", curr->data);
        free(curr);
    }
    return top;
}
```

main.c

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

```
Stack Data: 3
Stack Data: 2
Stack Data: 1
```

When top == NULL it means we reached the end of the stack

top → NULL