

Resizing Arrays

STACK allocated arrays cannot change size!

```
int main() {  
    // Fixed size array on the stack  
    int values[5] = {1, 2, 3, 4, 5};  
  
    for (size_t i = 0; i < 5; ++i)  
        printf("%i\n", values[i]);  
  
    return 0;  
}
```

Only HEAP arrays can resize

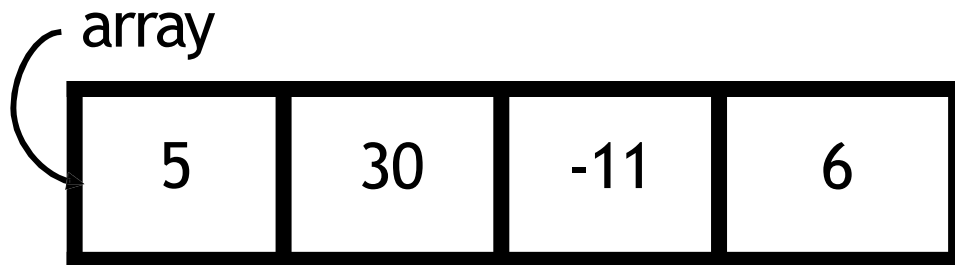
```
int main() {  
    // Dynamic array on the heap  
    int *values = calloc(5, sizeof(int));  
  
    for (size_t i = 0; i < 5; ++i)  
        printf("%i\n", values[i]);  
  
    free(values);  
  
    return 0;  
}
```

Refresher: Array Storage

- Arrays are stored in CONTIGUOUS areas of memory
 - Starting at the beginning of the array, each element in the array will follow the previous element consecutively in memory
- This is required due to the way we access array elements
 - The array variable is a pointer to the first element of the array (offset of 0)
 - Subsequent elements are found at the starting location + an offset
 - This is what happens when we request `array[n]` where $0 \leq n < \text{the array size}$.
 - `array[3]` is equivalent to `*(array + 3)`

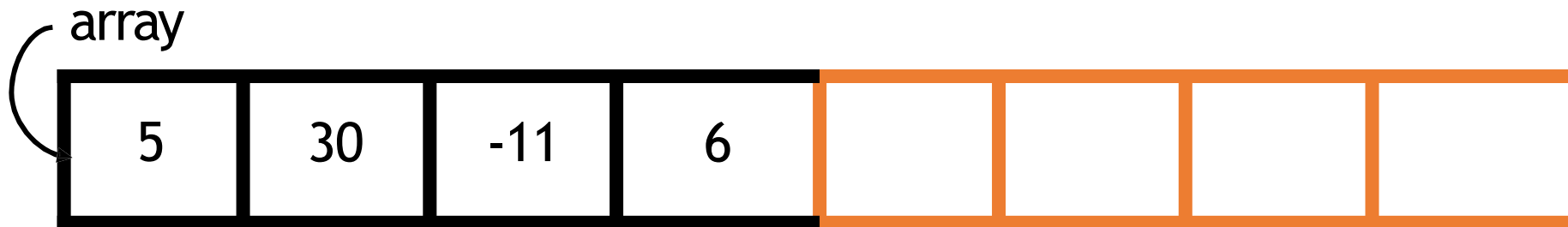
Resizing an Array

We can acquire more memory for our heap arrays in one of two ways depending on our memory layout.



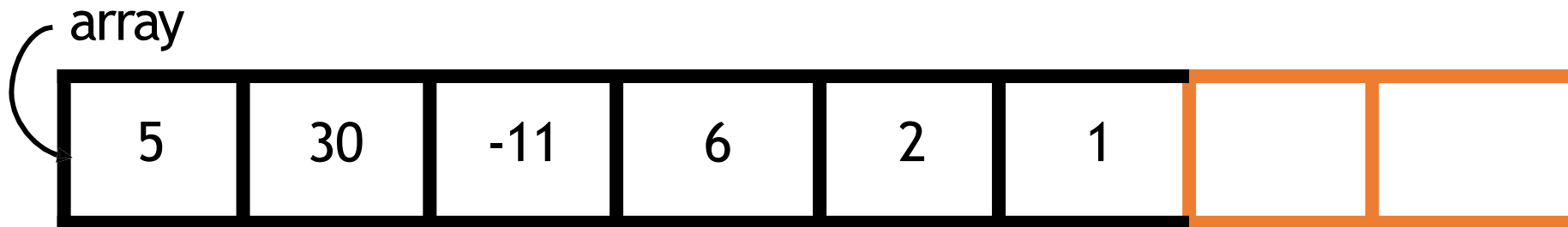
Resizing an Array: Condition #1

There is adequate free space next to the current array



Resizing an Array: Condition #1

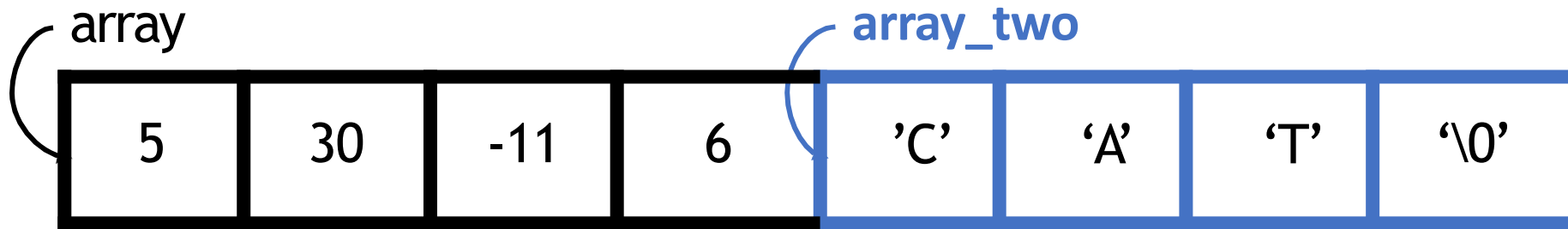
There is adequate free space next to the current array



We can grow our array in place and use the spare memory

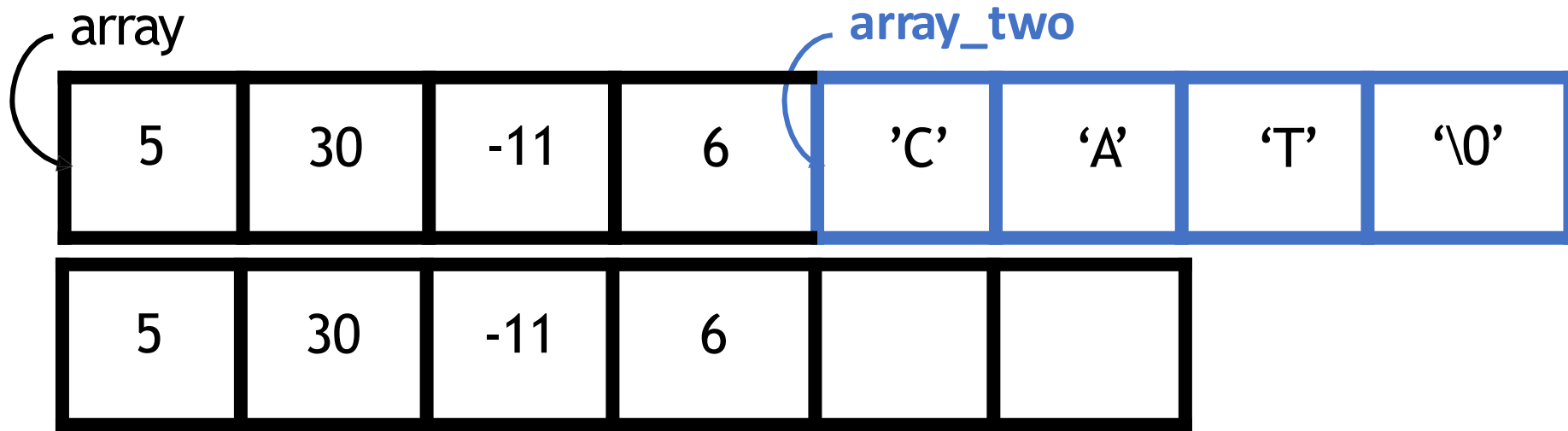
Resizing an Array: Condition #2

There is NOT enough free space next to the current array



Resizing an Array: Condition #2

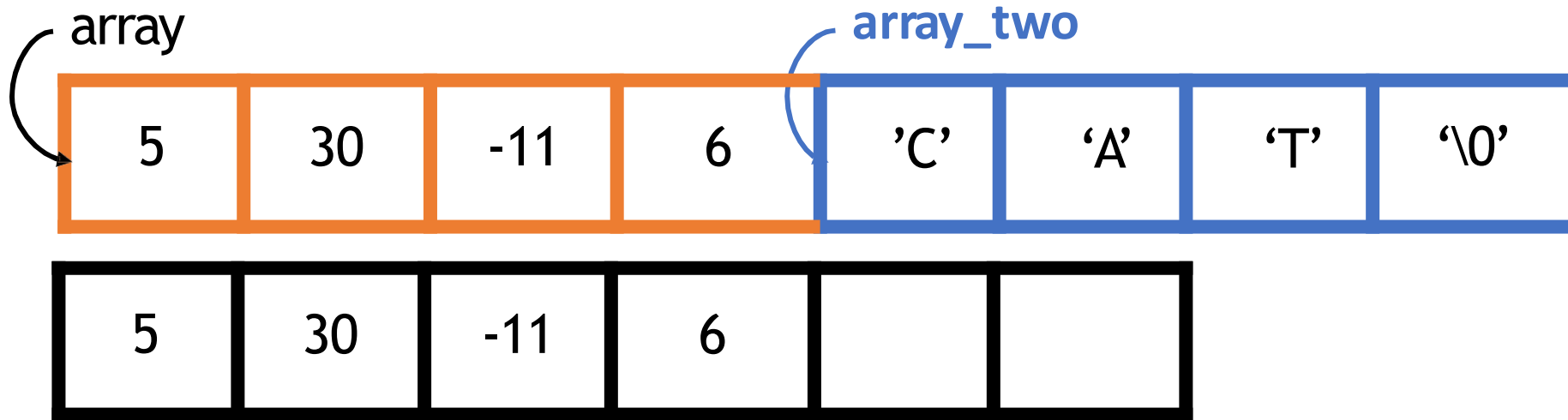
There is NOT enough free space next to the current array



1) Copy the old array to a larger memory location

Resizing an Array: Condition #2

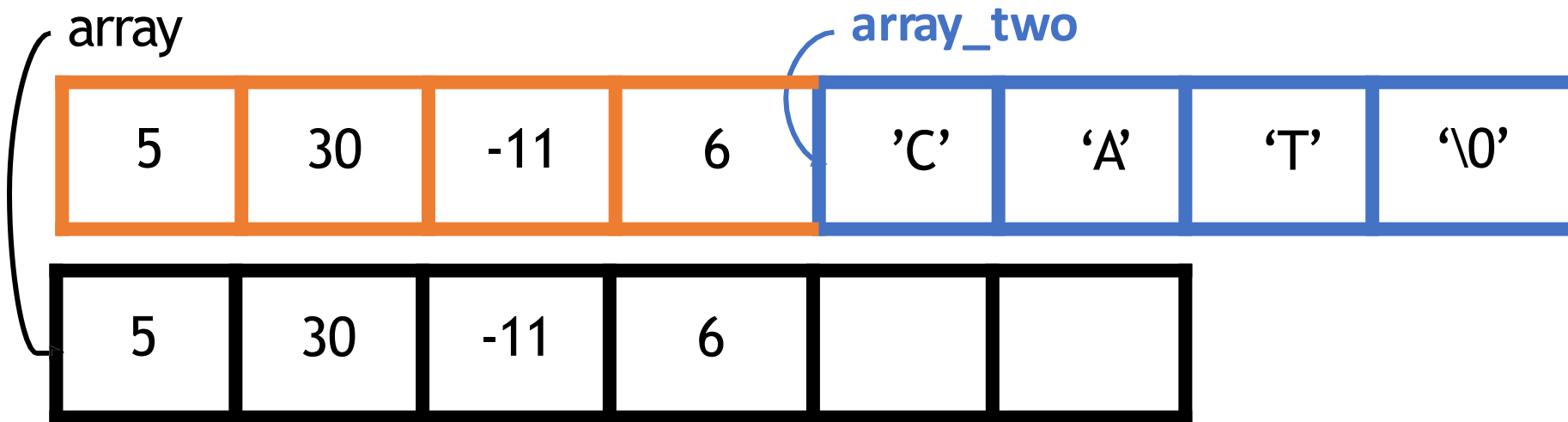
There is NOT enough free space next to the current array



- 1) Copy the old array to a larger memory location
- 2) Free the old array

Resizing an Array: Condition #2

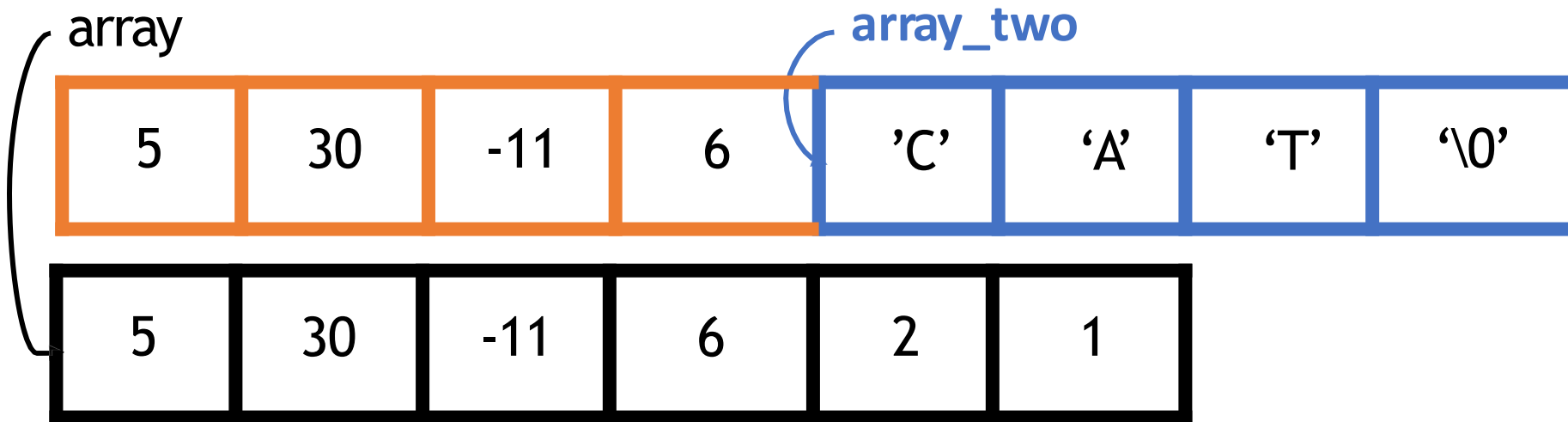
There is NOT enough free space next to the current array



- 1) Copy the old array to a larger memory location
- 2) Free the old array
- 3) Update the pointer

Resizing an Array: Condition #2

There is NOT enough free space next to the current array



- 1) Copy the old array to a larger memory location
- 2) Free the old array
- 3) Update the pointer

The realloc Function

- Good news, the realloc function will take care of this
 - `void *realloc(void *ptr, size_t size)`
 - Returns: a void pointer
 - Parameters: pointer to the original array, new size in bytes
- If there is consecutive free space next to the array realloc...
 - acquires extra space and returns the original pointer
- If there is insufficient space realloc...
 - creates a new array, copies the old data to the new array, frees old memory, and returns a pointer to the new array location

Writing a program that
dynamically resizes an array

Variables

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    size_t size = 0;

    size_t capacity = 1;

    return 0;
}
```

- size
 - number of elements that have been placed into the array
- capacity
 - total number of elements that the array can currently hold

Dynamic array

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    size_t size = 0;

    size_t capacity = 1;

    int *array = malloc(capacity * sizeof(int));

    return 0;
}
```

- Allocating memory in the heap with initial capacity

Reading from the standard input

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    size_t size = 0;

    size_t capacity = 1;

    int *array = malloc(capacity * sizeof(int));

    int input;
    while(scanf("%i", &input) == 1){
        if(size == capacity){
            capacity *= 2;
            array = realloc(array, capacity * sizeof(int));
            printf("Resized array to have capacity %zu\n", capacity);
        }
        array[size] = input;
        size++;
    }

    return 0;
}
```

- If `scanf()` is asked to read a single integer, it will
 - return 1 if it was able to read an integer, `i`
 - return 0 if there was more data from standard input but it was not an integer
 - return EOF if standard input reaches EOF
- By looping while `scanf()` returns 1
 - it will read all the integers until there is input that is
 - not an integer
 - EOF is reached.

Reading from the standard input

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    size_t size = 0;

    size_t capacity = 1;

    int *array = malloc(capacity * sizeof(int));

    int input;
    while(scanf("%i", &input) == 1){
        if(size == capacity){
            capacity *= 2;
            array = realloc(array, capacity * sizeof(int));
            printf("Resized array to have capacity %zu\n", capacity);
        }
        array[size] = input;
        size++;
    }

    return 0;
}
```

- Need to grow the array if the number of elements in the array is equal to the current capacity
- Doubling the capacity each time the array grows helps avoid calling `realloc()` many times if the array gets large

Reading from the standard input

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    size_t size = 0;

    size_t capacity = 1;

    int *array = malloc(capacity * sizeof(int));

    int input;
    while(scanf("%i", &input) == 1){
        if(size == capacity){
            capacity *= 2;
            array = realloc(array, capacity * sizeof(int));
            printf("Resized array to have capacity %zu\n", capacity);
        }
        array[size] = input;
        size++;
    }

    return 0;
}
```

- Need to assign the return value of `realloc()` back to the variable `array`, because the address of the array may have changed

Test run

```
1
2
Array has been resized to hold 2 values.
3
Array has been resized to hold 4 values.
4
5
Array has been resized to hold 8 values.
6
7
8
9
Array has been resized to hold 16 values.
```

input	size (beginning of iteration)	capacity	size (end of iteration)
1	0	1	1
2	1	2	2
3	2	4	3
4	3	4	4
5	4	8	5
6	5	8	6
7	6	8	7
8	7	8	8
9	8	16	9

Test run

```
(base) zwkbhowmiknb02:Downloads kbhowmik$ ./realloc
1
2
Resized array to have capacity 2
3
Resized array to have capacity 4
4
5
Resized array to have capacity 8
6
7
8
9
Resized array to have capacity 16
h
```

- Input that is not an integer

Printing out the elements of the array

```
int input;
while(scanf("%i", &input) == 1){
    if(size == capacity){
        capacity *= 2;
        array = realloc(array, capacity * sizeof(int));
        printf("Resized array to have capacity %zu\n", capacity);
    }
    array[size] = input;
    size++;
}

for(size_t i = 0; i < size; i++){
    printf("%i\n", array[i]);
}

return 0;
}
```

```
(base) zwkbhowmiknb02:Downloads kbhowmik$ ./realloc
1
2
Resized array to have capacity 2
3
Resized array to have capacity 4
4
5
Resized array to have capacity 8
10
2
3
4
5
```

Printing the elements till 'capacity' is reached

```
while(scanf("%i", &input) == 1){  
    if(size == capacity){  
        capacity *= 2;  
        array = realloc(array, capacity * sizeof(int));  
        printf("Resized array to have capacity %zu\n", capacity);  
    }  
    array[size] = input;  
    size++;  
}  
  
for(size_t i = 0; i < capacity; i++){  
    printf("%i\n", array[i]);  
}  
  
return 0;  
}
```

Printing the elements till 'capacity' is reached

```
1  
2  
Resized array to have capacity 2  
3  
Resized array to have capacity 4  
4  
5  
Resized array to have capacity 8  
hello
```

```
1  
2  
3  
4  
5  
0  
0  
0
```

Freeing up heap allocated memory

```
for(size_t i = 0; i < capacity; i++){  
    printf("%i\n", array[i]);  
}  
  
free(array);  
  
return 0;  
}
```

Adjusting the array size once input is finished

```
array = realloc(array, size * sizeof(int));
```

```
for(size_t i = 0; i < size; i++){  
    printf("%i\n", array[i]);  
}
```

```
free(array);
```

```
return 0;
```

```
}
```

NULL pointer

- malloc() and realloc() will return NULL if they were unable to allocate memory.
- NULL is a special value used to indicate that a pointer points to nothing.
- If you try to dereference a pointer that has the value NULL you will get a segmentation fault.

NULL Pointer

```
int main(){
    size_t size = 0;

    size_t capacity = 1;

    int *array = malloc(capacity * sizeof(int));

    if(array == NULL){
        printf("Unable to allocate memory.\n");
        return 1;
    }
```

```
int input;
while(scanf("%i", &input) == 1){
    if(size == capacity){
        capacity *= 2;
        array = realloc(array, capacity * sizeof(int));

        if (array == NULL){
            printf("Unable to realloc to a new capacity of %zu.\n", capacity);
            return 1;
        }

        printf("Resized array to have capacity %zu\n", capacity);
    }
    array[size] = input;
    size++;
}
```