Lecture 21

Pointers-3

Can we change the size of an array?

In C, the size of a statically allocated array cannot be changed once it is defined.

But sometimes we need to change the size of an array!

Problem-1

Problem: Classroom Management for Awesh

"Awesh is a teacher in a primary school, and he is responsible for managing the list of students' roll numbers in his classroom. At the beginning of the year, he takes an initial count of students and their roll numbers. As the year progresses, he knows that a fixed number of new students will be admitted in the middle of the year. Awesh wants to use dynamic memory allocation to efficiently manage the list of students' roll numbers. Help Awesh by writing a program that handles the initial list of roll numbers and then expands the list to accommodate the new students when they are admitted."

Requirements:

1) Prompt the user (Awesh) to enter the number of initial students.
2) Allocate memory dynamically for storing the roll numbers of these students using malloc.
3) Allow Awesh to enter the roll numbers of the initial students.
4) Mid-year, prompt the user to enter a fixed number of new students to be admitted.
5) Use realloc to expand the memory allocation to accommodate the new students.
6) Allow Awesh to enter the roll numbers of the new students.
7) Display the final list of all students' roll numbers.

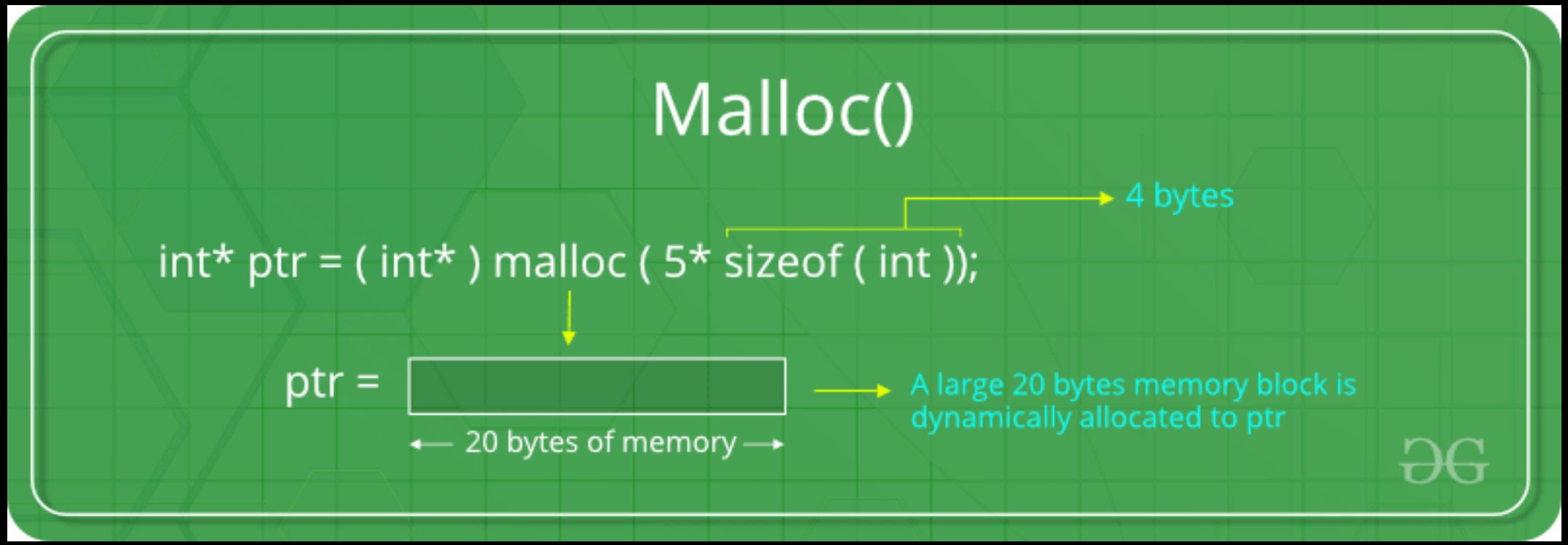
Dynamic Memory Allocation

To Change the size of an array in runtime we use dynamic memory allocation!

malloc() Syntax

```
ptr = (int*) malloc(100 *
sizeof(int));
//Syntax:
ptr =
(type*)malloc(bytesize)
```

malloc() Syntax



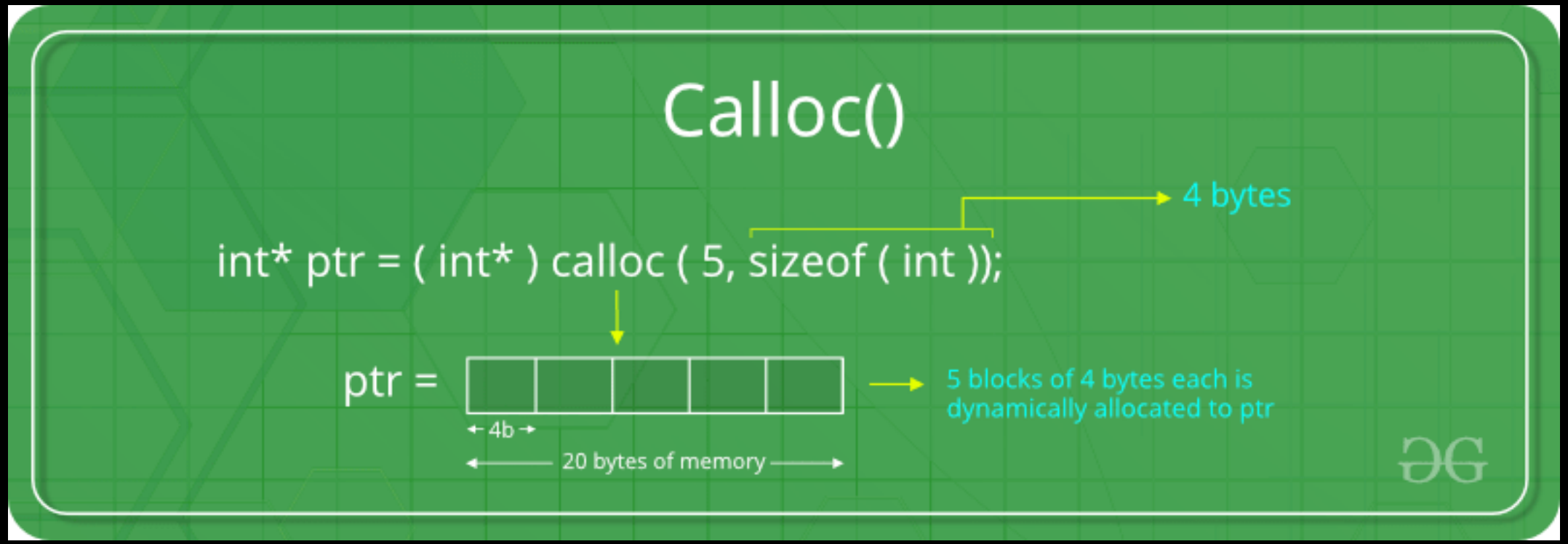
Example

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    // This pointer will hold the
    // base address of the block created
    int* ptr;
    int n, i;
    // Get the number of elements for the array
    printf("Enter number of elements:");
    scanf("%d",&n);
    printf("Entered number of elements: %d\n", n);
    // Dynamically allocate memory using malloc()
    ptr = (int*)malloc(n * sizeof(int));
    // Check if the memory has been successfully
    // allocated by malloc or not
    if (ptr == NULL) {
        printf("Memory not allocated.\n");
        exit(0);
    }
    else {
        // Memory has been successfully allocated
        printf("Memory successfully allocated using
malloc.\n");Get the elements of the array
        for (i = 0; i < n; ++i) {
            ptr[i] = i + 1;
        }
        // Print the elements of the array
        printf("The elements of the array are: ");
        for (i = 0; i < n; ++i) {
            printf("%d, ", ptr[i]);
        }
    }
    return 0;
}
```

calloc() syntax

```
//Syntax
ptr = (cast-type*)calloc(n, element-
size);
```

calloc() syntax



calloc() example

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

int main()
{

    // This pointer will hold the
    // base address of the block created
    int* ptr;
    int n, i;

    // Get the number of elements for the array
    n = 5;
    printf("Enter number of elements: %d\n", n);

    // Dynamically allocate memory using calloc()
    ptr = (int*)calloc(n, sizeof(int));

    // Check if the memory has been successfully
    // allocated by calloc or not
    if (ptr == NULL) {
        printf("Memory not allocated.\n");
        exit(0);
    }
    else {

        // Memory has been successfully allocated
        printf("Memory successfully allocated using
calloc.\n");Get the elements of the array
        for (i = 0; i < n; ++i) {
            ptr[i] = i + 1;
        }

        // Print the elements of the array
        printf("The elements of the array are: ");
        for (i = 0; i < n; ++i) {
            printf("%d, ", ptr[i]);
        }
    }

    return 0;
}
```

calloc() vs malloc()

Comparison of `malloc` and `calloc`

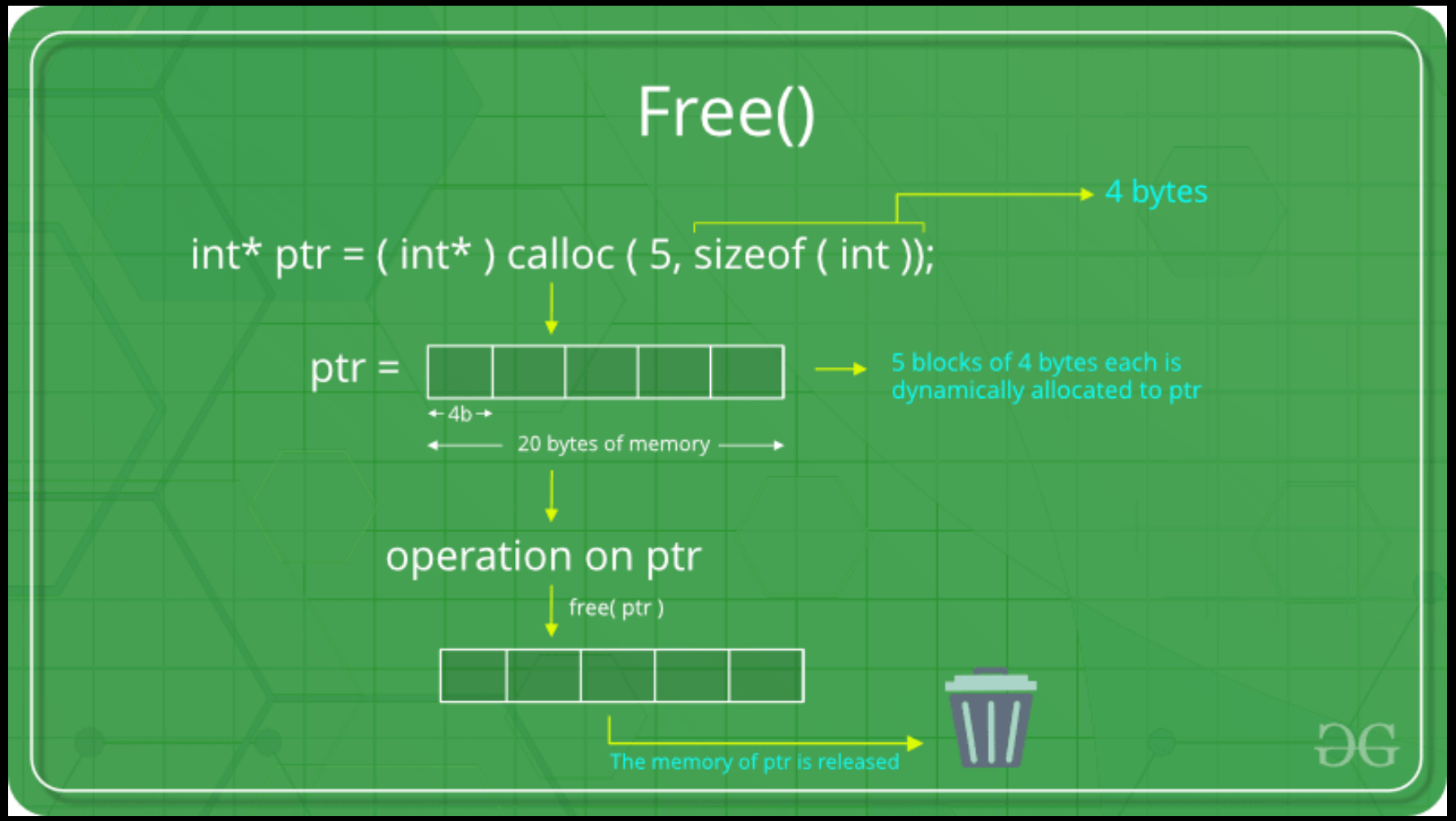
Feature	`malloc`	`calloc`
Function Prototype	`void* malloc(size_t size);`	`void* calloc(size_t num, size_t size);`
Memory Allocation	Allocates a block of memory of the specified size in bytes.	Allocates memory for an array of `num` elements, each of the specified size in bytes.
Initialization	Does not initialize the allocated memory (contains garbage values).	Initializes the allocated memory to zero.
Performance	Typically faster since it does not initialize memory.	Usually slower due to memory initialization.
Usage Example	`int *arr = (int *)malloc(5 * sizeof(int));`	`int *arr = (int *)calloc(5, sizeof(int));`
Flexibility	Flexible for allocating any amount of memory.	Best suited for allocating arrays with elements initialized to zero.
Memory Allocation Failure	Returns `NULL` if memory allocation fails.	Returns `NULL` if memory allocation fails.

free()

“**free**” method in C is used to dynamically **de-allocate** the memory. The memory allocated using functions malloc() and calloc() is not de-allocated on their own. Hence the free() method is used, whenever the dynamic memory allocation takes place. It helps to reduce wastage of memory by freeing it.

free() syntax

free() syntax


Free()
4 bytes
int* ptr = (int*) calloc (5, sizeof (int));
ptr =
5 blocks of 4 bytes each is dynamically allocated to ptr
4b
20 bytes of memory
operation on ptr
free(ptr)
The memory of ptr is released

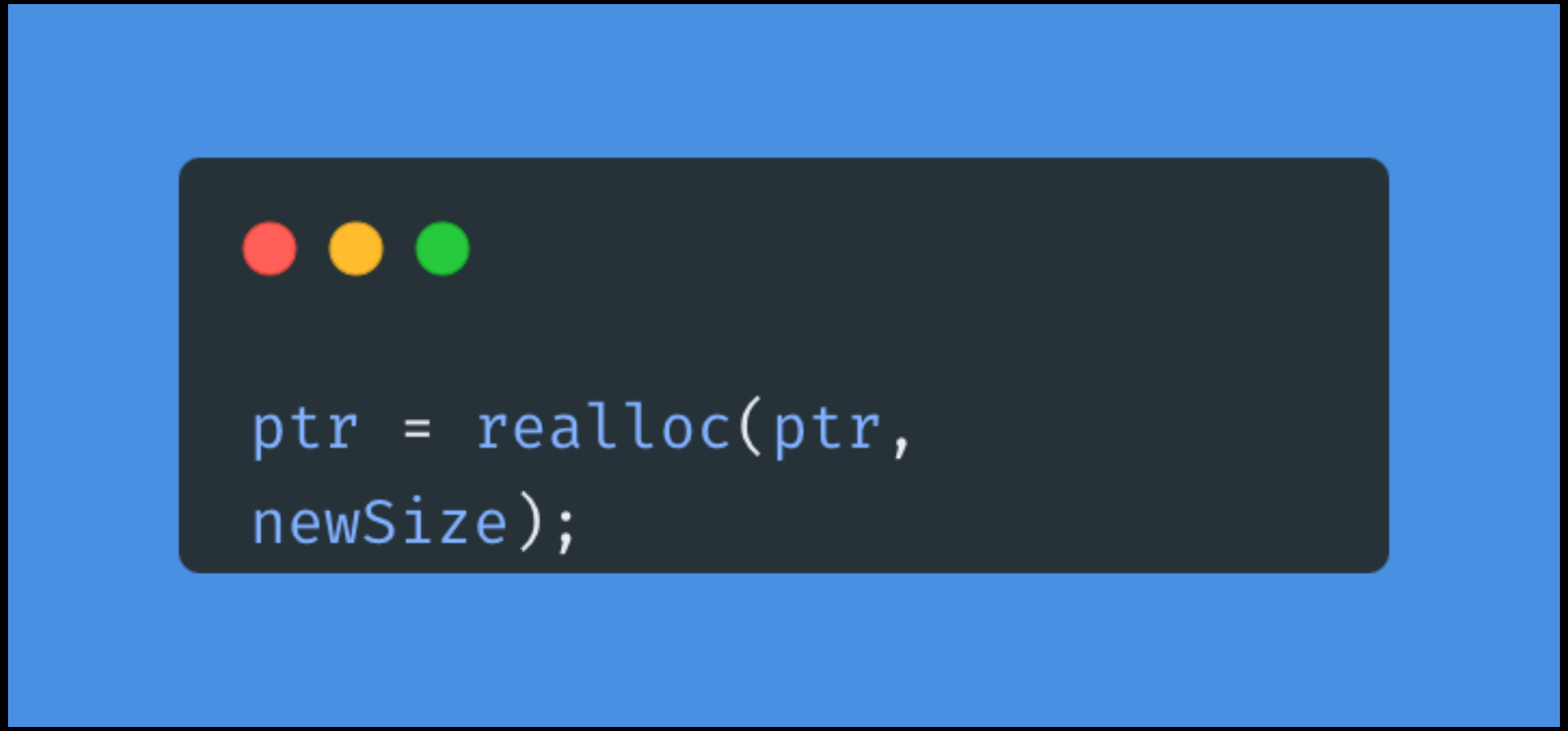
Free Example

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

int main()
{

    // This pointer will hold the
    // base address of the block created
    int *ptr, *ptr1;
    int n, i;

    // Get the number of elements for the array
    n = 5;
    printf("Enter number of elements: %d\n", n);

    // Dynamically allocate memory using malloc()
    ptr = (int*)malloc(n * sizeof(int));

    // Dynamically allocate memory using calloc()
    ptr1 = (int*)calloc(n, sizeof(int));

    // Check if the memory has been successfully
    // allocated by malloc or not
    if (ptr == NULL || ptr1 == NULL) {
        printf("Memory not allocated.\n");
        exit(0);
    }
    else {

        // Memory has been successfully allocated
        printf("Memory successfully allocated using malloc.\n");

        // Free the memory
        free(ptr);
        printf("Malloc Memory successfully freed.\n");

        // Memory has been successfully allocated
        printf("\nMemory successfully allocated using
calloc.\n");
        // Free the memory
        free(ptr1);
        printf("Calloc Memory successfully freed.\n");
    }

    return 0;
}
```

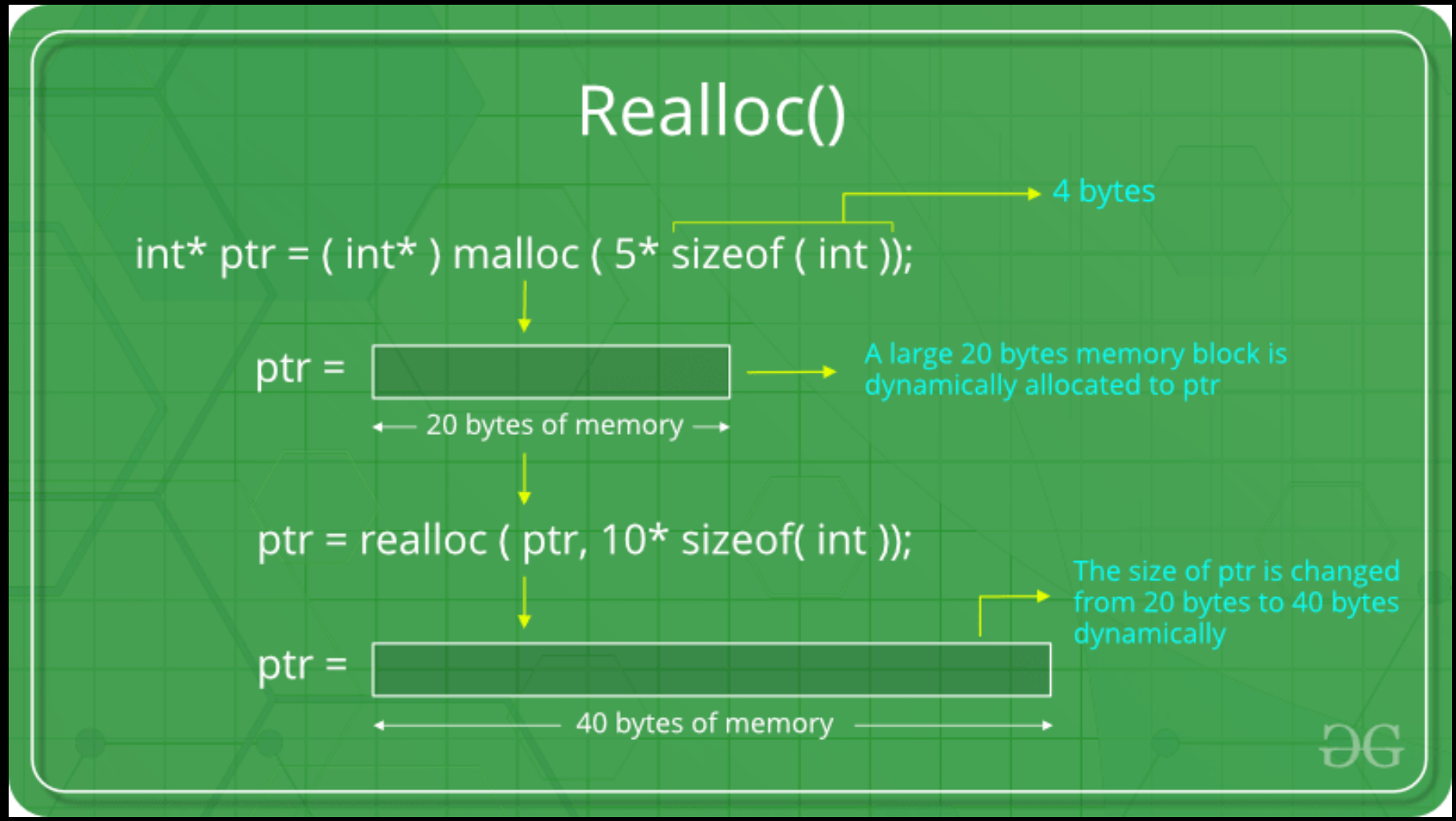
realloc()

“realloc” or **“re-allocation”** method in C is used to dynamically change the memory allocation of a previously allocated memory. In other words, if the memory previously allocated with the help of malloc or calloc is insufficient, realloc can be used to **dynamically re-allocate memory**. re-allocation of memory maintains the already present value and new blocks will be initialized with the default garbage value.“realloc” or “re-allocation” method in C is used to dynamically change the memory allocation of a previously allocated memory. In other words, if the memory previously allocated with the help of malloc or calloc is insufficient, realloc can be used to dynamically re-allocate memory. re-allocation of memory maintains the already present value and new blocks will be initialized with the default garbage value.

realloc() syntax



realloc() syntax



Realloc() example

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

int main()
{

    // This pointer will hold the
    // base address of the block created
    int* ptr;
    int n, i;

    // Get the number of elements for the array
    n = 5;
    printf("Enter number of elements: %d\n", n);

    // Dynamically allocate memory using calloc()
    ptr = (int*)calloc(n, sizeof(int));

    // Check if the memory has been successfully
    // allocated by malloc or not
    if (ptr == NULL) {
        printf("Memory not allocated.\n");
        exit(0);
    }
```

```c
else {
        // Memory has been successfully allocated
        printf("Memory successfully allocated using calloc.\n");
        // Get the elements of the array
        for (i = 0; i < n; ++i) {
            ptr[i] = i + 1;
        }
        // Print the elements of the array
        printf("The elements of the array are: ");
        for (i = 0; i < n; ++i) {
            printf("%d, ", ptr[i]);
        }
        // Get the new size for the array
        n = 10;
        printf("\n\nEnter the new size of the array: %d\n", n);
        // Dynamically re-allocate memory using realloc()
        ptr = (int*)realloc(ptr, n * sizeof(int));
        // Memory has been successfully allocated
        printf("Memory successfully re-allocated using
realloc.\n");
        // Get the new elements of the array
        for (i = 5; i < n; ++i) {
            ptr[i] = i + 1;
        }
        // Print the elements of the array
        printf("The elements of the array are: ");
        for (i = 0; i < n; ++i) {
            printf("%d, ", ptr[i]);
        }
        free(ptr);
    }
    return 0;
}
```

Solution To The Problem

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    int initial_count, new_count;
    printf("Enter the number of initial students: ");
    scanf("%d", &initial_count);
    // Allocate memory for the initial list of students' roll numbers
    int *students = (int *)malloc(initial_count * sizeof(int));
    if (students == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
    printf("Enter the roll numbers of the initial students:\n");
    enter_roll_numbers(students, initial_count);
    printf("Enter the number of new students to be admitted mid-year: ");
    scanf("%d", &new_count);
    // Reallocate memory to accommodate new students
    students = (int *)realloc(students, (initial_count + new_count) *
sizeof(int));
    if (students == NULL) {
        printf("Memory reallocation failed!\n");
        exit(1);
    }
    printf("Enter the roll numbers of the new students:\n");
    enter_roll_numbers(students + initial_count, new_count);
    printf("Final list of all students' roll numbers:\n");
    print_roll_numbers(students, initial_count + new_count);
    // Free the allocated memory
    free(students);
    return 0;
}
```

```c
void enter_roll_numbers(int *students, int count) {
    for (int i = 0; i < count; i++) {
        printf("Enter roll number of student %d: ", i + 1);
        scanf("%d", &students[i]);
    }
}

void print_roll_numbers(int *students, int count) {
    for (int i = 0; i < count; i++) {
        printf("Student %d Roll Number: %d\n", i + 1,
students[i]);
    }
}
```

Problem-1 Extended

What if we wanted to store the name of the students?

2D array Dynamic allocation

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

int main()
{
    int r = 3, c = 4, i, j, count;

    int** arr = (int**)malloc(r * sizeof(int*));
    for (i = 0; i < r; i++)
        arr[i] = (int*)malloc(c * sizeof(int));

    // Note that arr[i][j] is same as *(*(arr+i)+j)
    count = 0;
    for (i = 0; i < r; i++)
        for (j = 0; j < c; j++)
            arr[i][j] = ++count; // OR *(*(arr+i)+j) =
++count
    for (i = 0; i < r; i++)
        for (j = 0; j < c; j++)
            printf("%d ", arr[i][j]);

    /* Code for further processing and free the
    dynamically allocated memory */

    for (int i = 0; i < r; i++)
        free(arr[i]);

    free(arr);

    return 0;
}
```

Problem-1 Extended Solution

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

int main() {
    int initial_count, new_count;
    printf("Enter the number of initial students: ");
    scanf("%d", &initial_count);

    // Allocate memory for the initial list of students
    char **students = (char **)malloc(initial_count * sizeof(char *));
    if (students == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }

    printf("Enter the names of the initial students:\n");
    enter_names(students, initial_count);

    printf("Enter the number of new students to be admitted mid-year: ");
    scanf("%d", &new_count);

    // Reallocate memory to accommodate new students
    students = (char **)realloc(students, (initial_count + new_count) * sizeof(char
*));if (students == NULL) {
        printf("Memory reallocation failed!\n");
        exit(1);
    }

    printf("Enter the names of the new students:\n");
    enter_names(students + initial_count, new_count);

    printf("Final list of all students:\n");
    print_names(students, initial_count + new_count);

    // Free the allocated memory
    free_memory(students, initial_count + new_count);

    return 0;
}
```

```c
void enter_names(char **students, int count) {
    char buffer[100];
    for (int i = 0; i < count; i++) {
        printf("Enter name of student %d: ", i + 1);
        scanf("%s", buffer);
        students[i] = (char *)malloc((strlen(buffer) + 1) *
sizeof(char));if (students[i] == NULL) {
            printf("Memory allocation failed!\n");
            exit(1);
        }
        strcpy(students[i], buffer);
    }
}

void print_names(char **students, int count) {
    for (int i = 0; i < count; i++) {
        printf("Student %d: %s\n", i + 1, students[i]);
    }
}

void free_memory(char **students, int count) {
    for (int i = 0; i < count; i++) {
        free(students[i]);
    }
    free(students);
}
```

Problem-1 Extension-2

Problem: Classroom Management for Awesh (Multiple Classes)

"Awesh is a teacher responsible for managing the roll numbers of students in multiple classes. At the beginning of the year, he records the number of classes and the number of students in each class along with their roll numbers. After a year, the school decides to add more classes, and new students are admitted to each class. Awesh needs to update his records to include these new classes and students. Help Awesh by writing a program that handles the initial input and then expands to accommodate the new classes and students."

Requirements:

1) Prompt the user (Awesh) to enter the number of initial classes.
2) For each class, prompt the user to enter the number of students and their roll numbers.
3) Use dynamic memory allocation (malloc and realloc) to store the roll numbers for each class.
4) After a year, prompt the user to enter the number of new classes and the number of new students for each class.
5) Expand the memory allocation to include the new classes and students.
6) Display the final list of all classes and students' roll numbers.

Problem-1 Extension-2 Solution

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

void enter_roll_numbers(int *students, int count);
void print_classes(int **classes, int *students_count, int class_count);
void free_memory(int **classes, int *students_count, int class_count);

int main() {
    int initial_classes, new_classes;

    // Enter the number of initial classes
    printf("Enter the number of initial classes: ");
    scanf("%d", &initial_classes);

    // Allocate memory for the initial classes and their student counts
    int **classes = (int **)malloc(initial_classes * sizeof(int *));
    int *students_count = (int *)malloc(initial_classes * sizeof(int));
    if (classes == NULL || students_count == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }

    // Enter the number of students and their roll numbers for each
clasfor (int i = 0; i < initial_classes; i++) {
        printf("Enter the number of students in class %d: ", i + 1);
        scanf("%d", &students_count[i]);

        classes[i] = (int *)malloc(students_count[i] * sizeof(int));
        if (classes[i] == NULL) {
            printf("Memory allocation failed!\n");
            exit(1);
        }

        printf("Enter the roll numbers for class %d:\n", i + 1);
        enter_roll_numbers(classes[i], students_count[i]);
    }
```

```c
    // Enter the number of new classes to be added
    printf("Enter the number of new classes to be added: ");
    scanf("%d", &new_classes);

    // Reallocate memory to accommodate the new classes
    classes = (int **)realloc(classes, (initial_classes + new_classes) * sizeof(int *));
    students_count = (int *)realloc(students_count, (initial_classes + new_classes) *
sizeof(int));
    if (classes == NULL || students_count == NULL) {
        printf("Memory reallocation failed!\n");
        exit(1);
    }

    // Enter the number of students and their roll numbers for each new class
    for (int i = initial_classes; i < initial_classes + new_classes; i++) {
        printf("Enter the number of students in new class %d: ", i + 1);
        scanf("%d", &students_count[i]);

        classes[i] = (int *)malloc(students_count[i] * sizeof(int));
        if (classes[i] == NULL) {
            printf("Memory allocation failed!\n");
            exit(1);
        }

        printf("Enter the roll numbers for new class %d:\n", i + 1);
        enter_roll_numbers(classes[i], students_count[i]);
    }

    // Print the final list of all classes and their students' roll numbers
    printf("Final list of all classes and their students' roll numbers:\n");
    print_classes(classes, students_count, initial_classes + new_classes);

    // Free the allocated memory
    free_memory(classes, students_count, initial_classes + new_classes);

    return 0;
}
```

Problem-1 Extension-2 Solution

```c
void enter_roll_numbers(int *students, int count) {
    for (int i = 0; i < count; i++) {
        printf("Enter roll number of student %d: ", i + 1);
        scanf("%d", &students[i]);
    }
}

void print_classes(int **classes, int *students_count, int class_count) {
    for (int i = 0; i < class_count; i++) {
        printf("Class %d:\n", i + 1);
        for (int j = 0; j < students_count[i]; j++) {
            printf("  Student %d Roll Number: %d\n", j + 1, classes[i]
[j]);    }
    }
}

void free_memory(int **classes, int *students_count, int class_count) {
    for (int i = 0; i < class_count; i++) {
        free(classes[i]);
    }
    free(classes);
    free(students_count);
}
```

Problem

Problem: Dynamic Memory Allocation for a List of Numbers

You are required to create a program that allows the user to input a list of integers. The program should dynamically allocate memory for the list of numbers and resize the array as needed when more numbers are added.

Requirements:

Dynamic Array of Integers:

Use a dynamic array to store integers.

The array should expand as more numbers are added.

Functions:

add_number: Adds a new number to the array.

print_numbers: Prints all the numbers in the array.

free_memory: Frees all dynamically allocated memory.

User Interface:

The program should provide a simple text-based interface for adding numbers and displaying the list of numbers.

Example Program Flow:

Menu:

Add Number

Display Numbers

Exit

Add Number:

Prompt the user to enter a number.

Allocate memory for the new number and add it to the array.

Display Numbers:

Display the list of numbers.

Exit:

Before exiting, free all dynamically allocated memory.

Solution To The Problem

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    int *numbers = NULL;
    int size = 0;
    int capacity = 0;
    int choice;
    while (1) {
        printf("\nMenu:\n");
        printf("1. Add Number\n");
        printf("2. Display Numbers\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                add_number(&numbers, &size, &capacity);
                break;
            case 2:
                print_numbers(numbers, size);
                break;
            case 3:
                free_memory(numbers);
                return 0;
            default:
                printf("Invalid choice! Please try
again.\n");
    }
    return 0;
}
```

```c
void add_number(int *numbers, int *size, int *capacity) {
    if (*size == *capacity) {
        *capacity = (*capacity == 0) ? 1 : *capacity * 2;
        *numbers = realloc(*numbers, *capacity *
sizeof(int));
        if (*numbers == NULL) {
            printf("Memory allocation failed!\n");
            exit(1);
        }
    }

    printf("Enter number: ");
    scanf("%d", &(*numbers)[*size]);
    (*size)++;
}

void print_numbers(int *numbers, int size) {
    printf("Numbers in the array:\n");
    for (int i = 0; i < size; i++) {
        printf("%d ", numbers[i]);
    }
    printf("\n");
}

void free_memory(int *numbers) {
    free(numbers);
}
```