Unit VI – Part 04
(Dynamic Memory Allocation)

Lecture Slide

AS2023

Objectives

By the end of this session, students will be able to:

- Define and understand pointer to a pointer
- To change the memory size using Dynamic Memory Allocation

POINTER TO A POINTER

- C language allows to use pointers that point to pointers
- To declare pointers to pointers, just add an asterisk (*) for each level of reference

```
int y=4;
int *p; //pointer to an integer
int **pt; //pointer to a pointer to an integer
p=&y;
pt=&p;
```

NULL POINTER

- A NULL pointer is a special pointer value that does not point anywhere
- It does not point to any valid memory address
- To declare a Null pointer, a constant known as NULL is used

```
int *ptr=NULL;
```

- NULL is defined in several header files like <stdio.h>, <stdlib.h> and <string.h>

TYPE CASTING

- A way to convert a constant or variable from one data type to another data type

Example 1:

```
int x=9,y=6;
float s=(float)x+(float)y;
printf("%f",s);
```

Example 2:

```
float const  C=3.14;
printf("%d",(int)C);
```

Dynamic Memory Allocation

- The process of allocating memory to the variable during execution of the program or at run time
- This allows the proper utilization of memory efficiently
  - Only the required amount of memory is reserved

Dynamic Memory Allocation

- An array is a collection of a fixed number of values. Once the size of an array is declared, you cannot change it.
- Sometimes the size of the array you declared may be insufficient.
- To solve this issue, you can allocate memory manually during run-time. This is known as dynamic memory allocation in C programming.
- To allocate memory dynamically, library functions are `malloc()`, `calloc()`, `realloc()` and `free()` are used. These functions are defined in the `<stdlib.h>` header file.

malloc()

- The name ***"malloc"*** stands for memory allocation.
- The ***malloc()*** function reserves a block of memory of the specified number of bytes. And, it returns a pointer of void which can be cast into pointers of any form.

  Syntax:

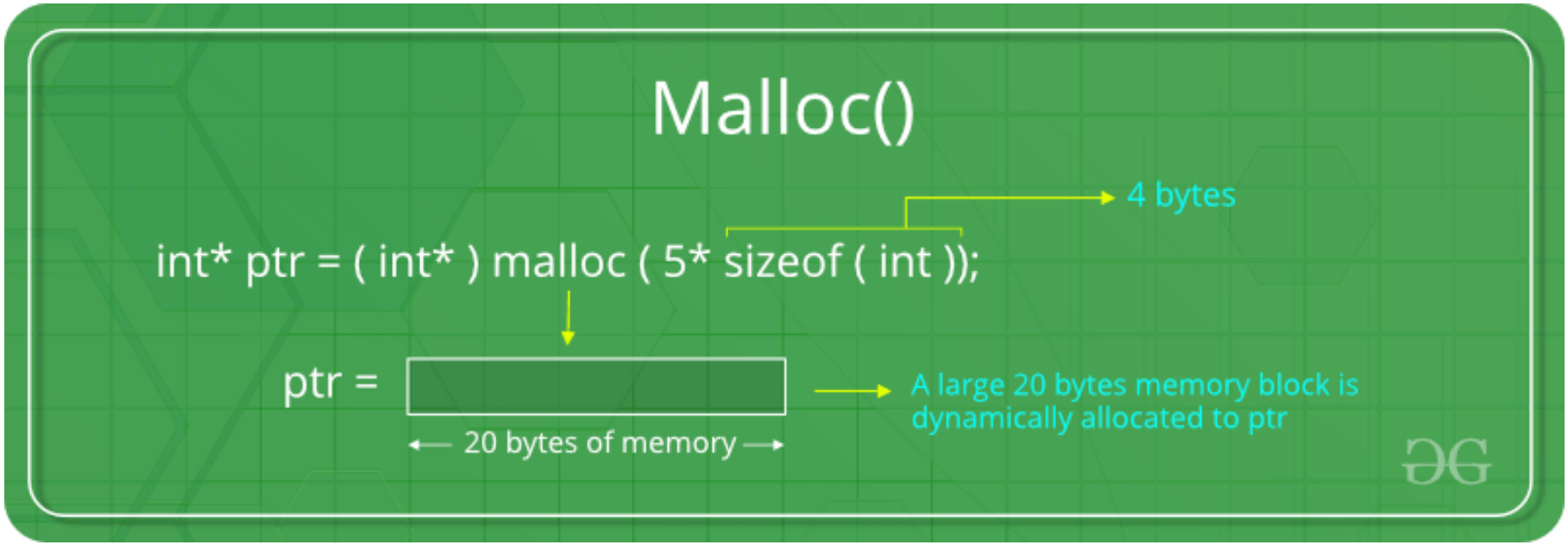
```
ptr = (castType*) malloc(size);
```

  Example:

```
ptr = (float*) malloc(100 * sizeof(float));
```

- The above statement allocates 400 bytes of memory. It's because the size of float is 4 bytes. And, the pointer ***ptr*** holds the address of the first byte in the allocated memory.

malloc()


Malloc()
4 bytes
int* ptr = (int*) malloc (5* sizeof (int));
ptr =
A large 20 bytes memory block is dynamically allocated to ptr
20 bytes of memory

calloc()

- The name ***"calloc"*** stands for contiguous allocation.
- The ***malloc()*** function allocates memory and leaves the memory uninitialized. Whereas, the ***calloc()*** function allocates memory and initializes all bits to zero.

Syntax:

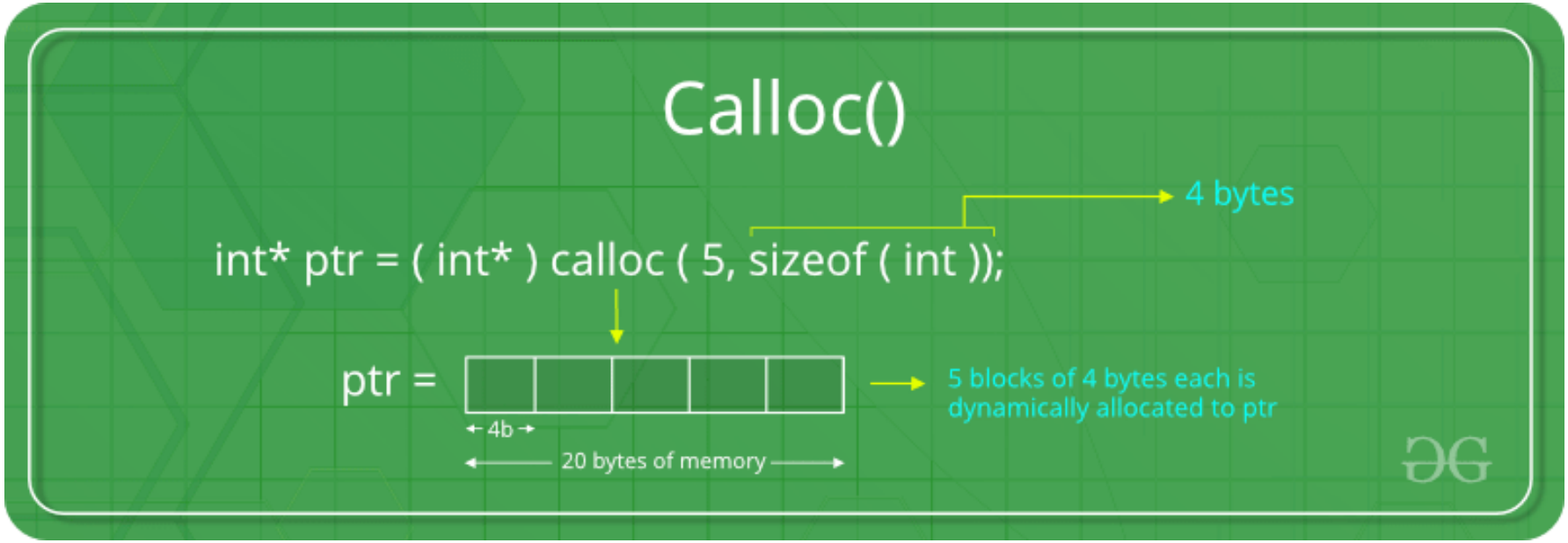
```
ptr = (castType*)calloc(n, size);
```

Example:

```
ptr = (float*) calloc(25, sizeof(float));
```

- The above statement allocates contiguous space in memory for 25 elements of type `float`.

calloc()


Calloc()
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

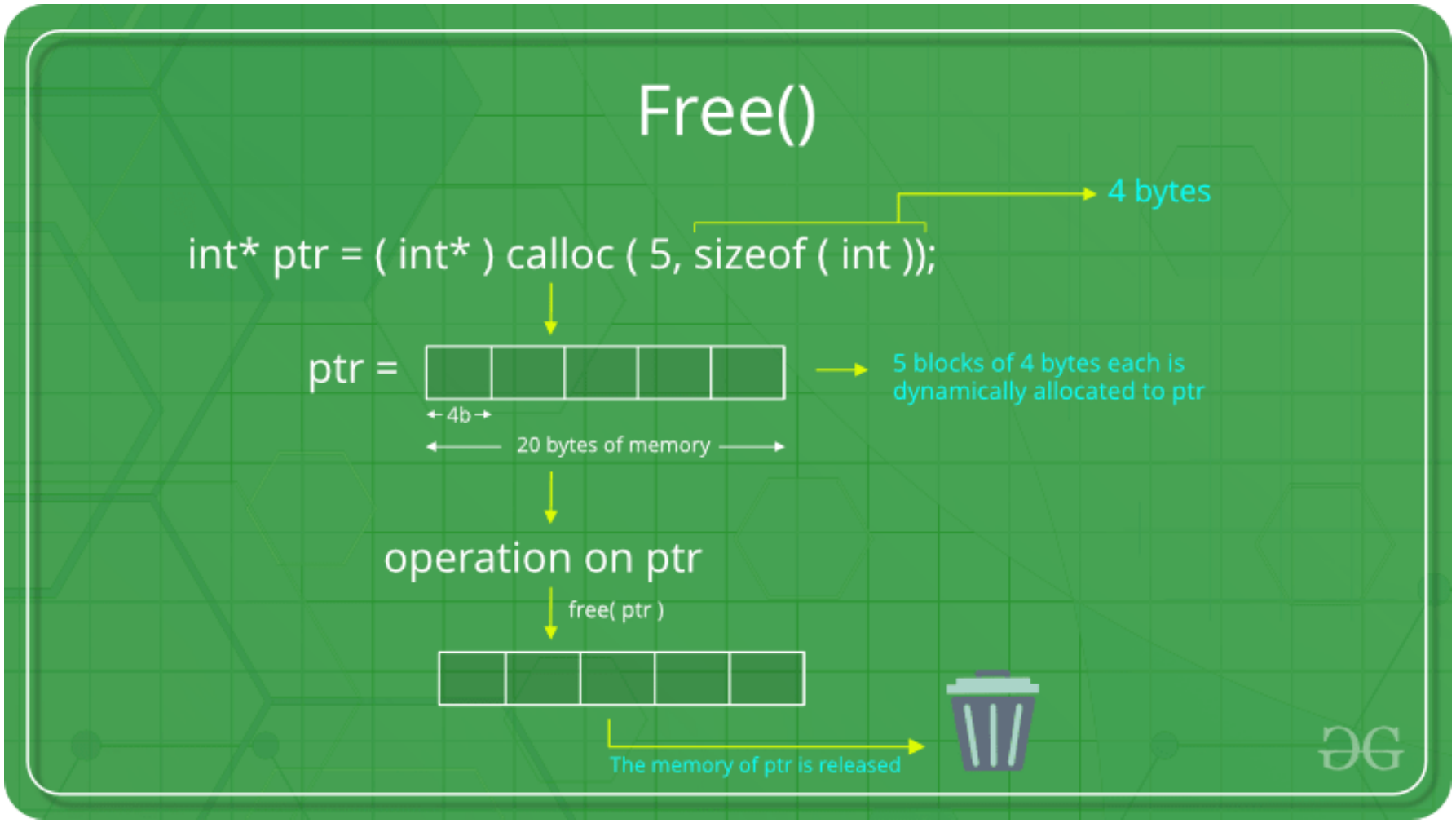
free()

- Dynamically allocated memory created with either `calloc()` or `malloc()` doesn't get freed on their own. You must explicitly use `free()` to release the space.
- Syntax of free():

```
free(ptr);
```

- This statement frees the space allocated in the memory pointed by *ptr*.

free()


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

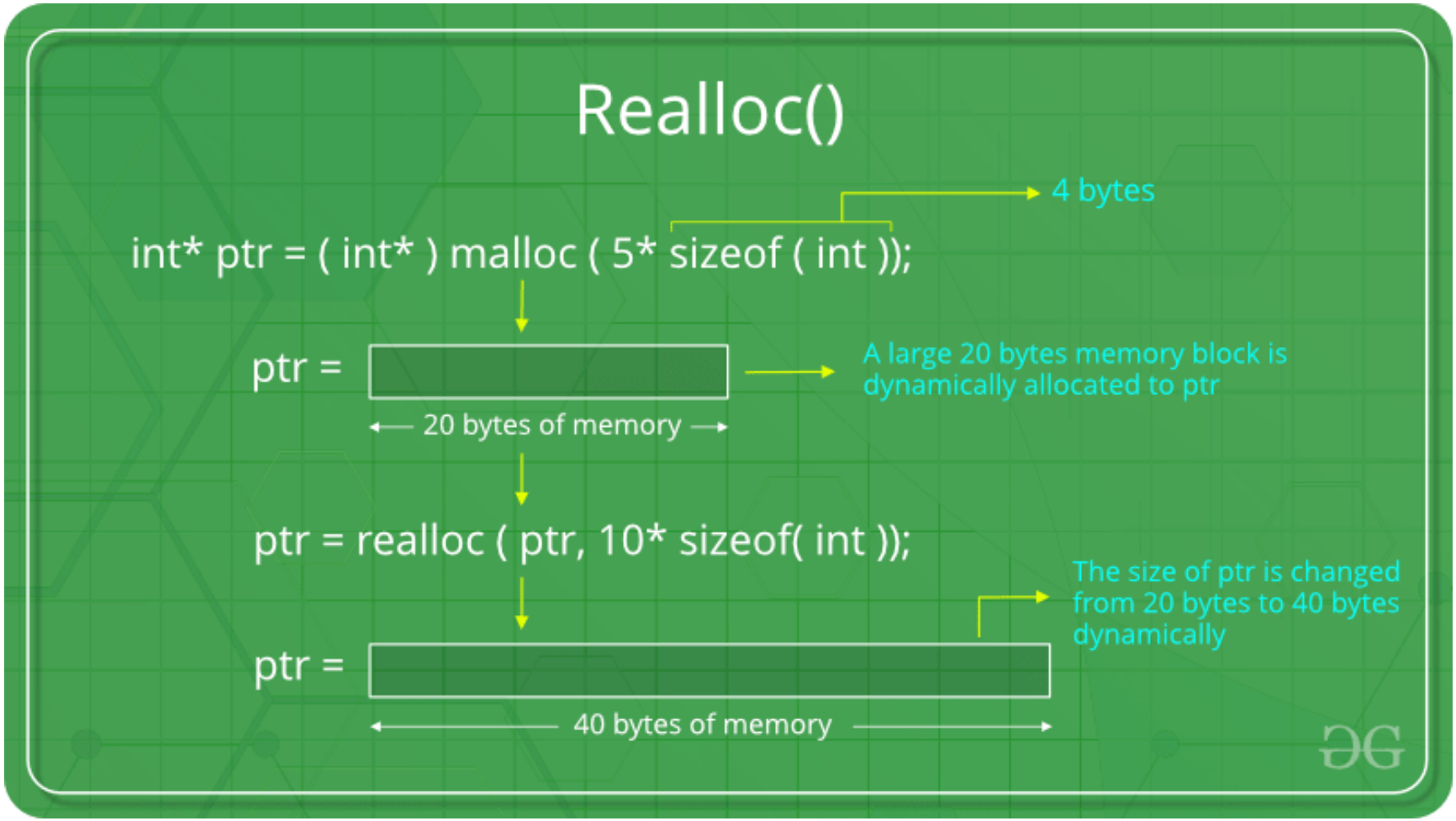
realloc()

- If the dynamically allocated memory is insufficient or more than required, you can change the size of previously allocated memory using the *realloc()* function.
- Alters the size of previously allocated memory
- Syntax of realloc()

```
ptr = realloc(ptr, x);
```

- Here, *ptr* is reallocated with a new size *x*.

realloc()


Realloc()
4 bytes
int* ptr = (int*) malloc (5* sizeof (int));
ptr =
A large 20 bytes memory block is dynamically allocated to ptr
20 bytes of memory
ptr = realloc (ptr, 10* sizeof(int));
The size of ptr is changed from 20 bytes to 40 bytes dynamically
ptr =
40 bytes of memory

Example: Memory Allocation

```
// Program to calculate the sum of n numbers entered by the user
#include <stdio.h>
#include <stdlib.h>
int main() {
        int n, i, *ptr, sum = 0;
        printf("Enter number of elements: ");
        scanf("%d", &n); ptr = (int*) malloc(n * sizeof(int));
        // if memory cannot be allocated
        if(ptr == NULL) {
                printf("Error! memory not allocated.");
                exit(0);
        }
        printf("Enter elements: ");
        for(i = 0; i < n; ++i) {
                scanf("%d", ptr + i);
                sum += *(ptr + i);
        }
        printf("Sum = %d", sum);
        // deallocating the memory
        free(ptr);
        return 0;
}
```

Home Work

1. Find the Largest Element in a Dynamically Allocated Memory
2. Calculate the sum of ***n*** numbers entered by the user but later change the numbers to ***n-1*** size before entering $(n-2)^{th}$ number.

Thank you