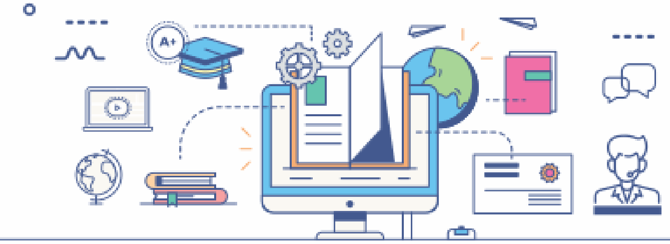
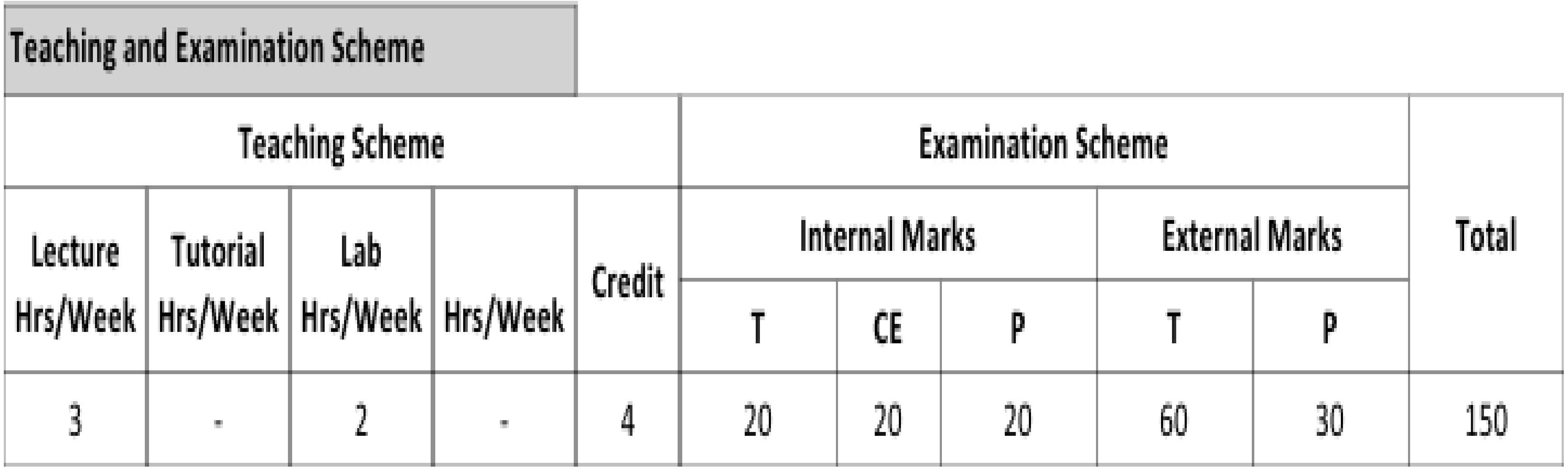
303105151 - Computational Thinking for Structured Design-2

Dr. Harish Prajapati, Assistant Professor
AI & AIDS(PIET)



Scheme of the Subject



Teaching and Examination Scheme

Teaching Scheme					Examination Scheme					Total
Lecture Hrs/Week	Tutorial Hrs/Week	Lab Hrs/Week	Hrs/Week	Credit	Internal Marks			External Marks		
					T	CE	P	T	P	
3	-	2	-	4	20	20	20	60	30	150

SEE - Semester End Examination, **CIA** - Continuous Internal Assessment (It consists of Assignments/Seminars/Presentations/MCQ Tests, etc.)

Parul® University NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

Course Outcome

After Completion of course students shall be able to :

1. Learn to use data structures concepts for realistic Problems
2. Ability to identify appropriate data structures for Solving computing problems in respective language.
3. Ability to solve problems independently and think critically.
4. Understand the concept of File Management.

CHAPTER-2

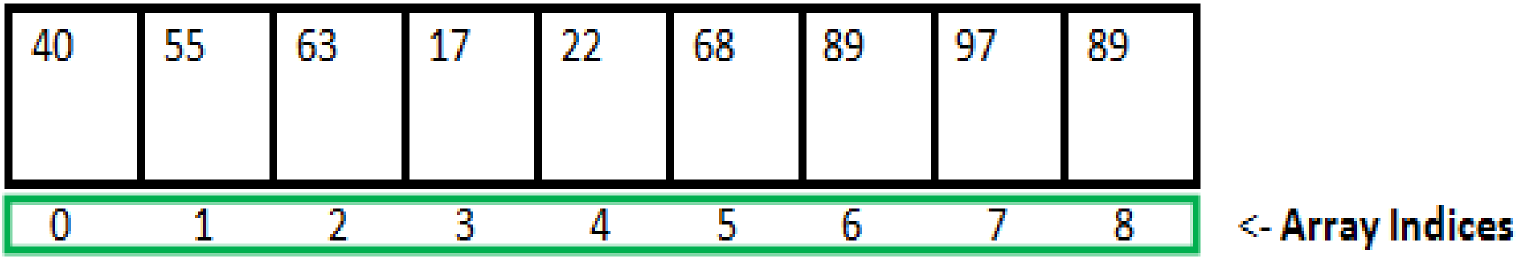
Dynamic Memory Allocation:

- Introduction
- Merits of DMA
- malloc
- calloc
- realloc
- free functions

Dynamic memory allocation

- Problem with Arrays:
- The length (size) of the array is fixed. But what if there is a requirement to change this length (size)?
- For example,
- If there is a situation where only 5 elements are needed to be entered in array of size 9. In this case, the remaining 4 indices are just wasting memory in this array. So there is a requirement to lessen the length (size) of the array from 9 to 5.
- Take another situation. In this, there is an array of 9 elements with all 9 indices filled. But there is a need to enter 3 more elements in this array. In this case, 3 indices more are required. So the length (size) of the array needs to be changed from 9 to 12.

Dynamic memory allocation



Array Length = 9

First Index = 0

Last Index = 8

Dynamic memory allocation

- This procedure is referred to as Dynamic Memory Allocation in C.
- Dynamic memory allocation is essential for efficient memory management in C.
- Therefore, C **Dynamic Memory Allocation** can be defined as a procedure in which the size of a data structure (like Array) is changed during the runtime.

Dynamic memory allocation

- There are 4 library functions provided by C defined under **<stdlib.h>** header file to facilitate dynamic memory allocation in C programming. They are:
- malloc()
- calloc()
- free()
- realloc()

Merits of DMA

1.Allocation of memory: In static memory allocation the variables are allocated permanently hence a wastage of memory is occurred, But in case of dynamic memory allocation the variables are get allocated until the program unit gets inactive. Memory allocation occurs during runtime.

2.Efficiency: The efficiency of dynamic memory allocation is more than static memory allocation and it is very flexible also.

3.Memory reusability: The previously used memory can also be reused in dynamic memory allocation, a function named “free” is used to release the memory for future use when the user need it.

4.Reallocation of memory: In static memory allocation after a memory is allocated we can’t change its size. But in dynamic memory allocation using the “realloc()” function we can change the memory block size as per our requirement.

Dynamic memory allocation

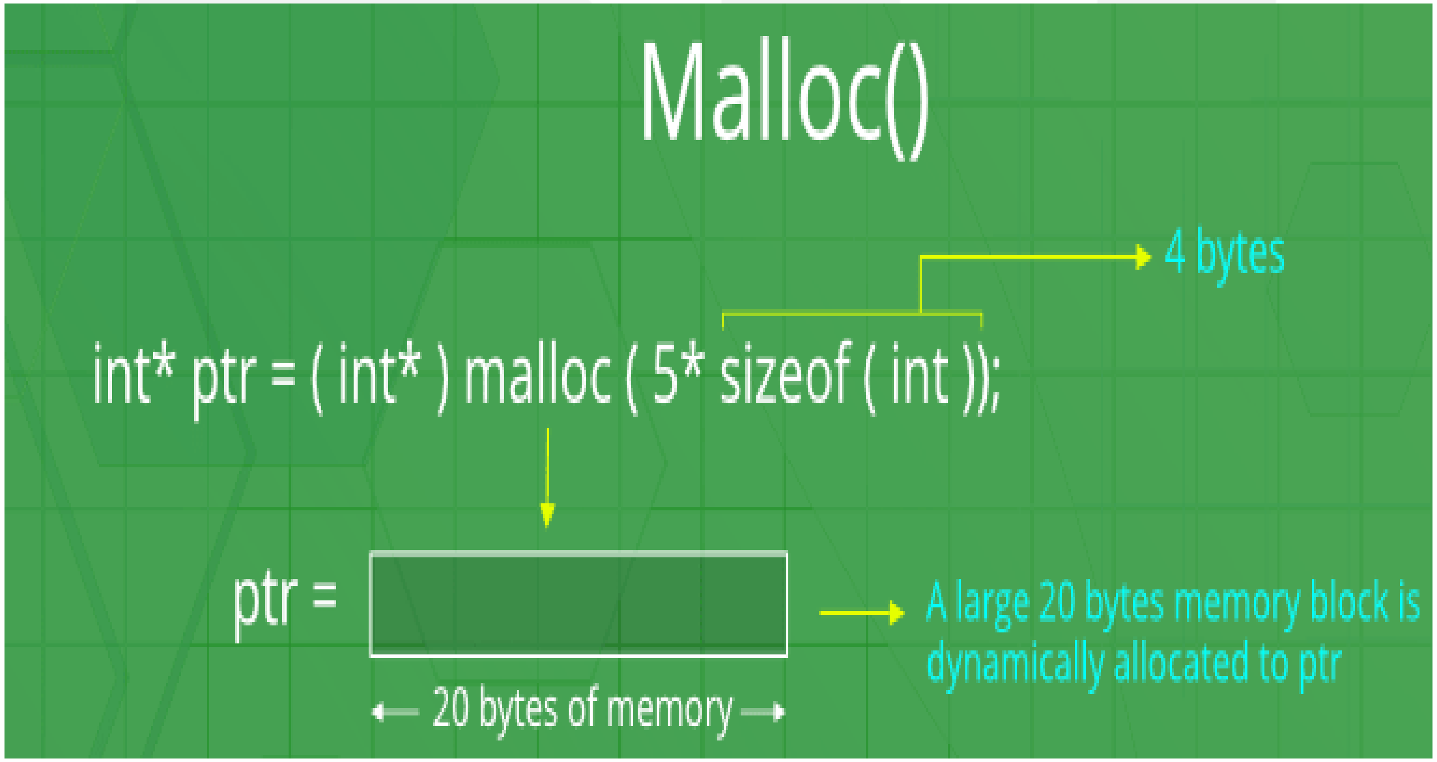
- There are 4 library functions provided by C defined under **<stdlib.h>** header file to facilitate dynamic memory allocation in C programming. They are:
- malloc()
- calloc()
- free()
- realloc()

malloc()

- The **“malloc”** or **“memory allocation”** method in C is used to dynamically allocate a single large block of memory with the specified size.
- **Syntax of malloc() in C**
- ptr = (cast-type*) malloc(byte-size)
- For Example:
- ***ptr = (int*) malloc(100 * sizeof(int));***

  Since the size of int is 4 bytes, this statement will allocate 400 bytes of memory. And, the pointer ptr holds the address of the first byte in the allocated memory.

malloc()


Malloc()
4 bytes
ptr =
20 bytes of memory
A large 20 bytes memory block is dynamically allocated to ptr

```
int* ptr = ( int* ) malloc ( 5* sizeof ( int ));
```

Parul® University NAAC A++ ACCREDITED UNIVERSITY

Example of malloc()

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
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");
```

Example of malloc()

```
// Get the elements of the array
      for (i = 0; i < n; ++i) {
         ptr[i] = 3*i+5;
      }

      // 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;
}
```

```
Enter number of elements:8
Entered number of elements: 8
Memory successfully allocated using
malloc.
The elements of the array are:
5,8,11,14,17,20,23,26
```

calloc()

“calloc” or **“contiguous allocation”** method in C is used to dynamically allocate the specified number of blocks of memory of the specified type.
it is very much similar to malloc() but has two different points and these are:
It initializes each block with a default value ‘0’.
It has two parameters or arguments as compare to malloc().

calloc()

Syntax of calloc() in C

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

here, n is the no. of elements and element-size is the size of each element.

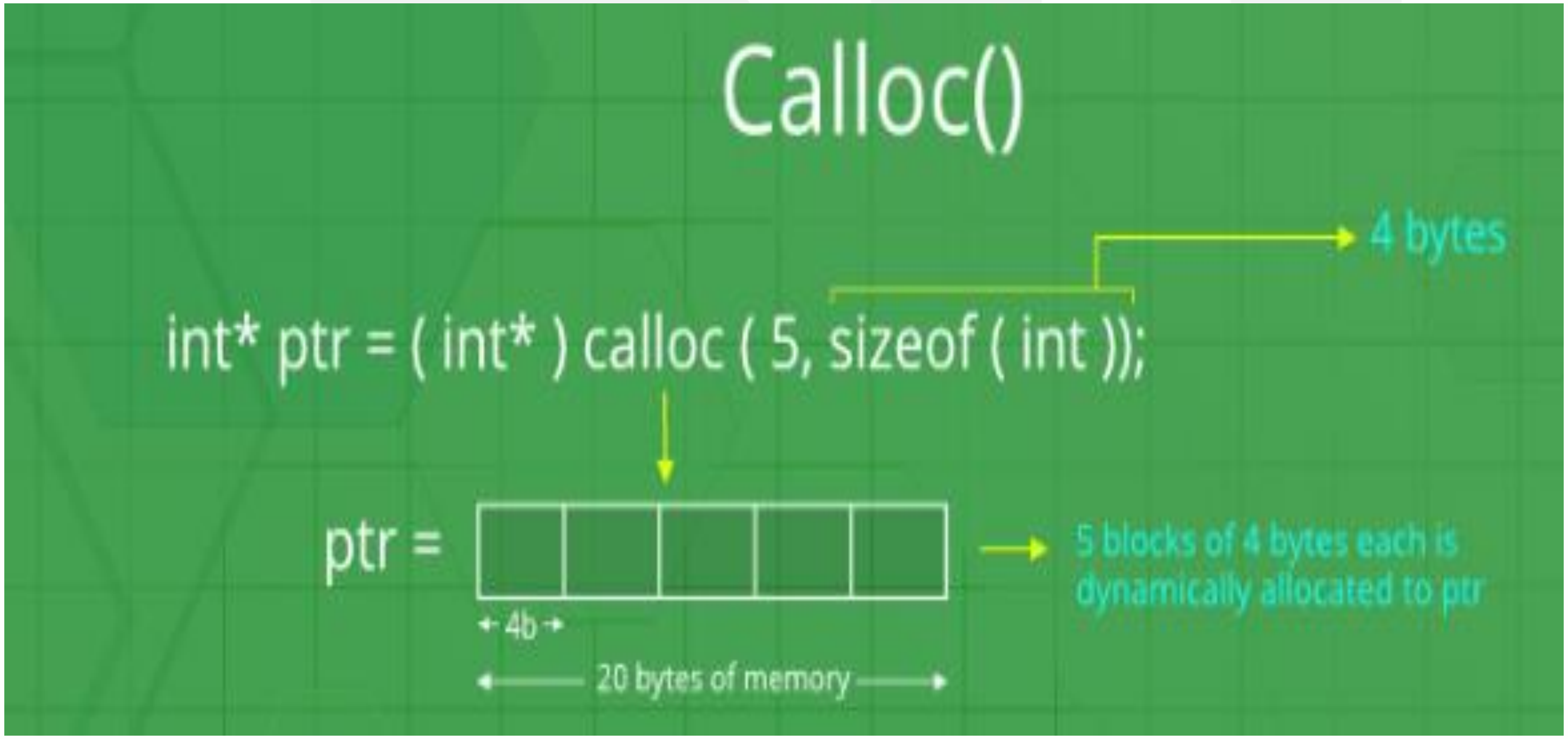
For Example:

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

This statement allocates contiguous space in memory for 25 elements each with the size of the float.

If space is insufficient, allocation fails and returns a NULL pointer.

calloc()


Calloc()
4 bytes
int* ptr = (int*) calloc (5, sizeof (int));
ptr =
4b
20 bytes of memory
5 blocks of 4 bytes each is dynamically allocated to ptr

calloc()

Example of calloc() in C

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
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");
```

calloc()

```
// Get the elements of the array
        for (i = 0; i < n; ++i) {
            ptr[i] = 5*i + 8;
        }

        // 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;
}
```

```
 Enter number of elements: 5
Memory successfully allocated
using calloc.
The elements of the array are:
8,13,18,23,28
```

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.

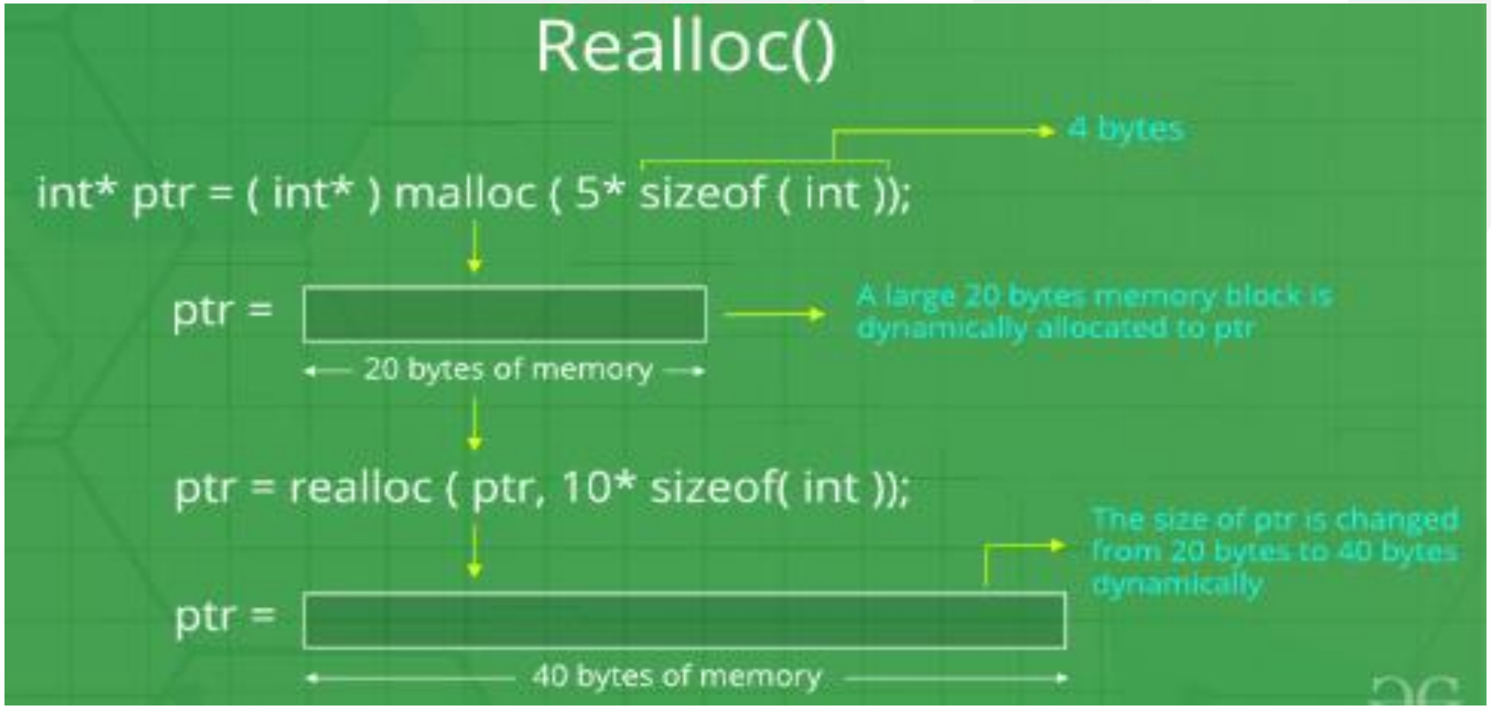
re-allocation of memory maintains the already present value and new blocks will be initialized with the default garbage value.

realloc()

Syntax of realloc() in C

ptr = realloc(ptr, newSize);

where ptr is reallocated with new size 'newSize'.


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

realloc()

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
int* ptr;
    int n, i;
    // Get the number of elements for
the array
    printf("Enter number of elements:
%d\n", n);
n = 5;
// 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);
    }
else {
        // Memory has been successfully
allocated
        printf("Memory successfully
allocated using calloc.\n");
```

realloc()

```
// 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
printf("\n\nEnter the new size of the
array: %d\n", n);
        n = 10;
```

```
// Dynamically re-allocate memory
using realloc()
        ptr = (int*)realloc(ptr, n *
sizeof(int));

          if (ptr == NULL) {
          printf("Reallocation Failed\n");
          exit(0);
        }
/ Memory has been successfully
allocated
        printf("Memory successfully re-
allocated using realloc.\n");
```

DIGITAL LEARNING CONTENT

realloc()

```
// 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;
}
```

```
Enter number of elements: 5
Memory successfully allocated using
calloc.
The elements of the array are: 1, 2, 3, 4, 5,

Enter the new size of the array: 10
Memory successfully re-allocated using
realloc.
The elements of the array are: 1, 2, 3, 4, 5,
6, 7, 8, 9, 10,
```

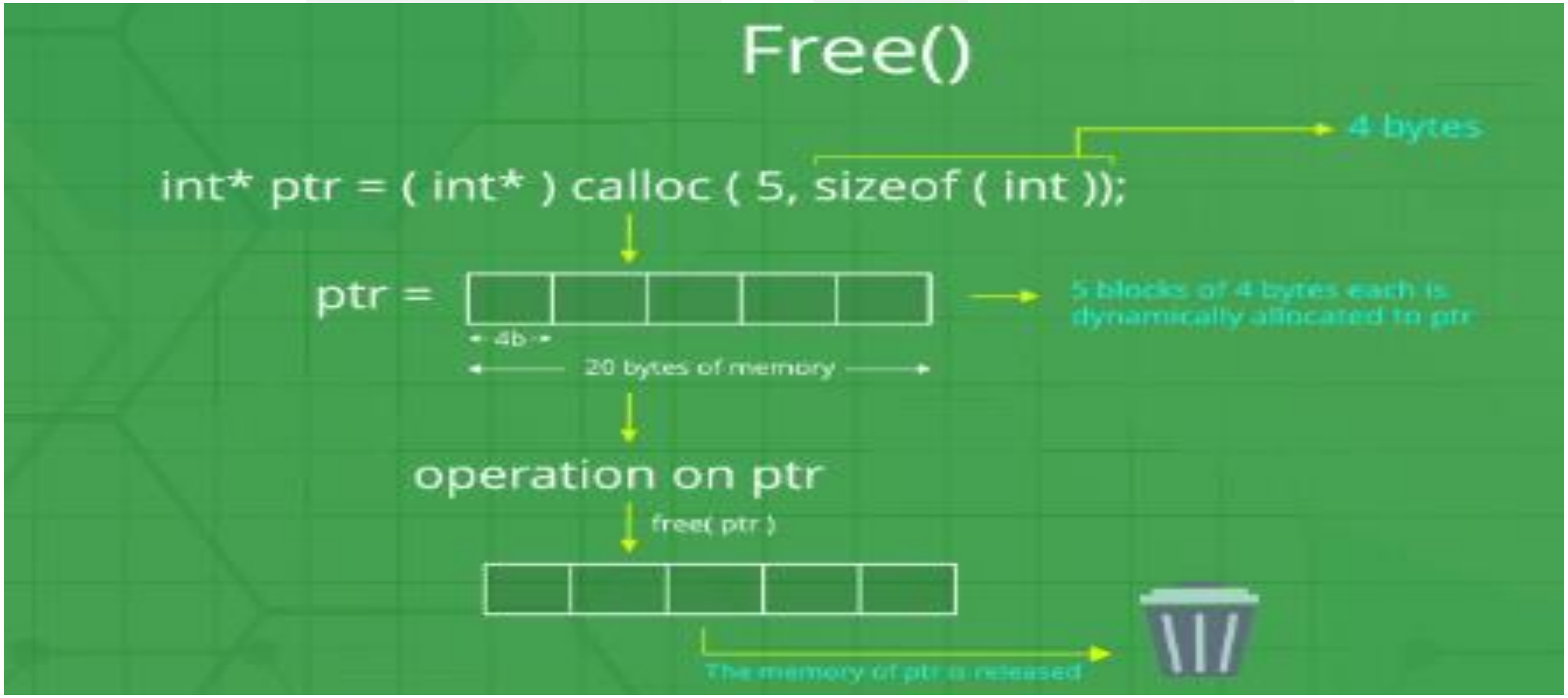
free()

- **“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 of free() in C

free(ptr);


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

DIGITAL LEARNING CONTENT

free()

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
        int *ptr, *ptr1;
    int n, i;
    // Get the number of elements for the
array
    n = 5;
    printf("Entered 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()

```
// 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;
}
```

```
Enter number of elements: 5
Memory successfully allocated using malloc.
Malloc Memory successfully freed.

Memory successfully allocated using calloc.
Calloc Memory successfully freed.
```

Application of C

Operating systems: C is widely used for developing operating systems such as Unix, Linux, and Windows.

Embedded systems: C is a popular language for developing embedded systems such as microcontrollers, microprocessors, and other electronic devices.

System software: C is used for developing system software such as device drivers, compilers, and assemblers.

Networking: C is widely used for developing networking applications such as web servers, network protocols, and network drivers.

Database systems: C is used for developing database systems such as Oracle, MySQL, and PostgreSQL.

Application of C

Gaming: C is often used for developing computer games due to its ability to handle low-level hardware interactions.

Artificial Intelligence: C is used for developing artificial intelligence and machine learning applications such as neural networks and deep learning algorithms.

Scientific applications: C is used for developing scientific applications such as simulation software and numerical analysis tools.

Financial applications: C is used for developing financial applications such as stock market analysis and trading systems.

Thank
You

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in