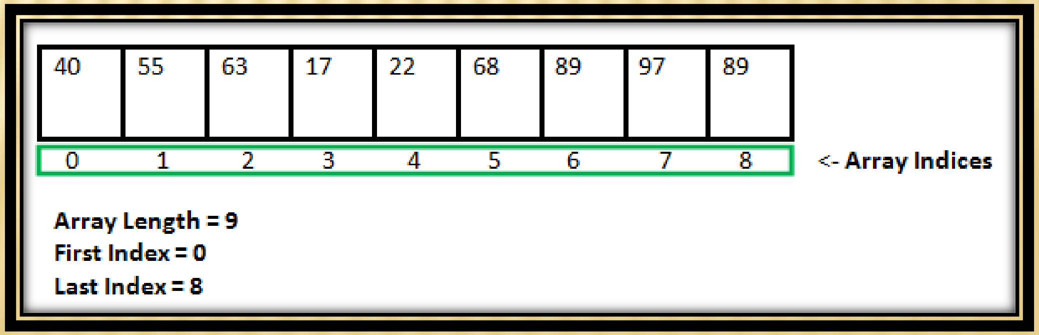
DYNAMIC MEMORY ALLOCATION

STATIC MEMORY ALLOCATION

- As we have seen, array size during declaration must be a constant value only.
- Example:

  int a[9];
- If we have a situation where we need to use only 5 elements, remaining 4 positions are a waste of memory.
- In another scenario, suppose we want to use 3 more elements beyond the 9 elements of the array. The size of existing cannot be extended from 9 to 12.



STATIC MEMORY ALLOCATION

- The previous two scenarios are the major drawbacks of static memory allocation
- Static memory allocation occurs during compilation time.
- Memory for arrays traditionally are created by static allocation.
- In case of static memory allocation, the size of an array once fixed cannot be changed.
- That is why we cannot get the required flexibility to change the size of an array as and when required.
- To do that, we must use **dynamic memory allocation**

DYNAMIC MEMORY ALLOCATION

- The process of allocating memory during **program execution** is called dynamic memory allocation.
- Allocating memory during runtime:
  - makes it more flexible and
  - reduces wastage of memory space.
- There are three main ways in which memory can be allocated dynamically:
  - malloc()
  - calloc()
  - realloc()
  - free()
- The above are built-in functions and to use them we must include “stdlib.h” in our program.

NULL POINTERS

- If pointers are uninitialized, what will be the value?
  - Garbage value
- In computer programming, “NULL” is a value exclusively assigned to pointers, to indicate that they are currently not pointing to any address.

```
int num = 10;
int *ptr1 = &num;
int *ptr2;          <-------  Uninitialized pointer
int *ptr3=NULL;     <-------  NULL pointer
```

- NULL is a constant value like 0.

MALLOC

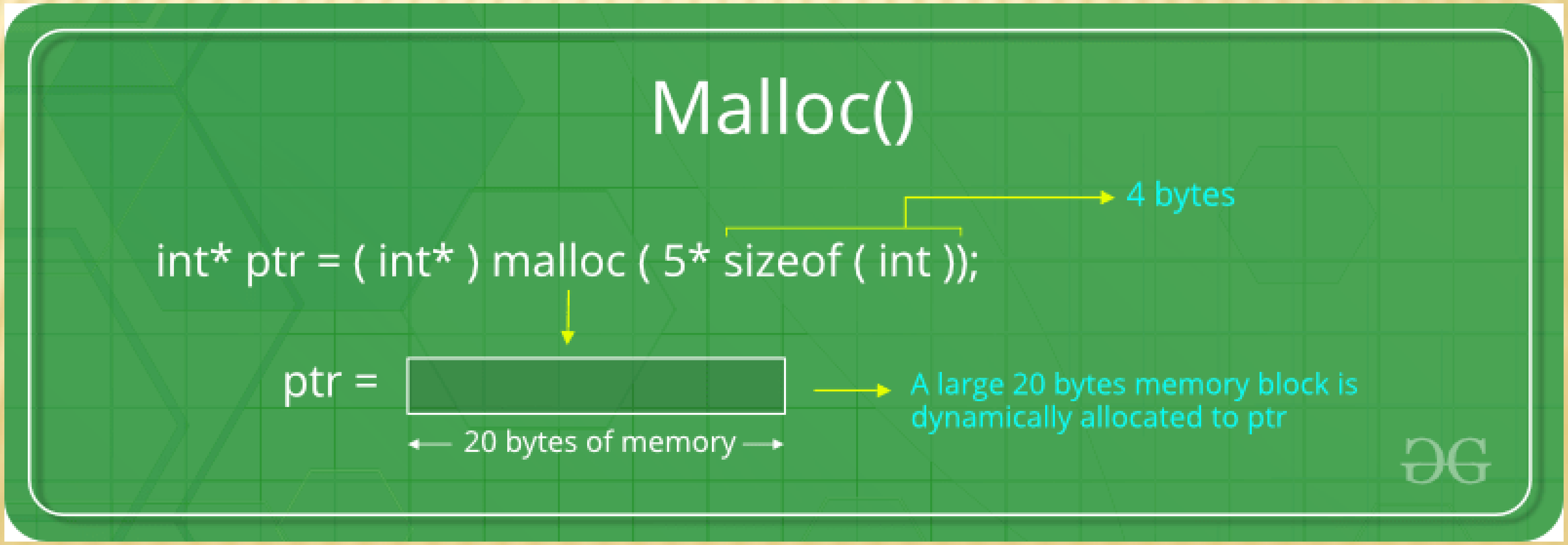
- **"malloc"** or **"memory allocation"** method is used to dynamically allocate a single large block of memory with the specified size.
- It returns a pointer of type void which can be cast into a pointer of any form.
- malloc () does not initialize the memory allocated during execution.
- It carries garbage value.
- malloc () function returns null pointer if it is unable to allocate requested amount of memory.
- Syntax:

datatype *ptr = (datatype*) malloc (size in bytes) ;

MALLOC () - EXAMPLE

int *ptr = (int*) malloc(100 * sizeof(int));

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


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

CALLOC ()

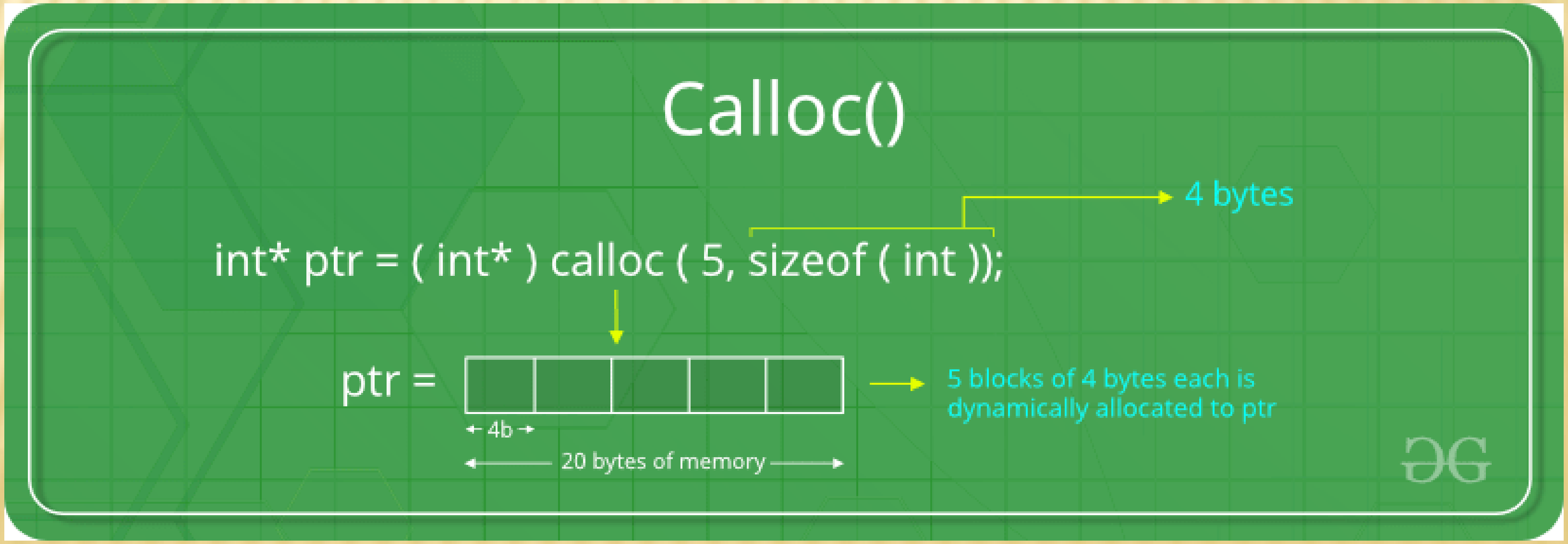
- **"calloc"** or **"contiguous allocation"** is another method for dynamic memory allocation.
- It is used to dynamically allocate the specified number of blocks of memory of the specified type.
- It initializes each block with a default value '0'.
- Syntax:

```
datatype *ptr = (datatype*) calloc (n, element-size);
```

CALLOC() - EXAMPLE

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

- This statement allocates contiguous space in memory for 25 elements each with the size of float.
- The address of the first block is returned by the calloc function.



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

QUICK EXERCISE: PREDICT THE VALUE OF SUM

```
int* ptr;
int n, i, sum = 0;
ptr = (int*)calloc(5, sizeof(int));
if (ptr == NULL)
{
printf("Memory not allocated.\n");
 }
else
{
 for (i = 0; i < 5; ++i)
{
 ptr[i] = i + 1;
sum +=ptr[i];
}
```

Output:

```
15
```

REALLOC()

- Let’s say we have allocated some memory using malloc() and calloc(), but later we find that memory is too large or too small.
- The realloc() function is used to resize allocated memory without losing old data.
- Syntax:

  datatype *new_ptr = (datatype*) realloc (old_ptr, new_size in bytes);

- The realloc() function accepts two arguments,
  - old_ptr is a pointer to the first byte of memory that was previously allocated using malloc() or calloc() function.
  - new_size specifies the new size of the block in bytes, which may be smaller or larger than the original size.

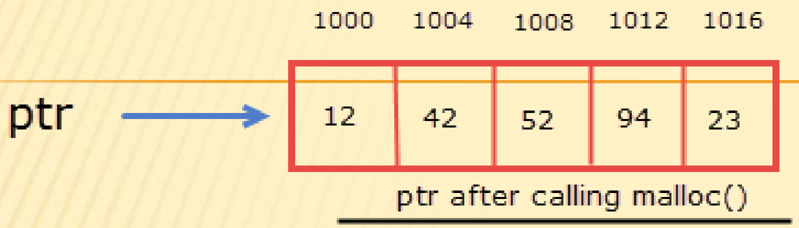
EXAMPLE: REALLOC()

- Assume we have allocated memory for 5 integers using malloc().
- Suppose that later we want to increase the size of the allocated memory to store 6 more integers.
- To do that, we have to allocate additional 6 *sizeof(int) bytes of memory.

```
int *p;
p = (int*)malloc(5*sizeof(int)); // allocate memory for 5 integers
p = (int*)realloc(p, 11*sizeof(int));
```

- realloc() function only allocates 6 * sizeof(int) bytes next to already used bytes.
- It is important to note that in doing so old data is not lost but newly allocated bytes are uninitialized.

MEMORY REPRESENTATION OF REALLOC()

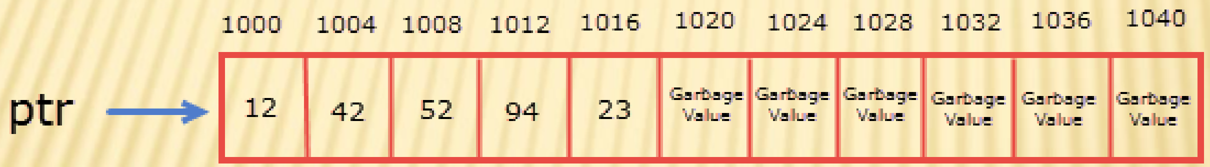


ptr after calling malloc()

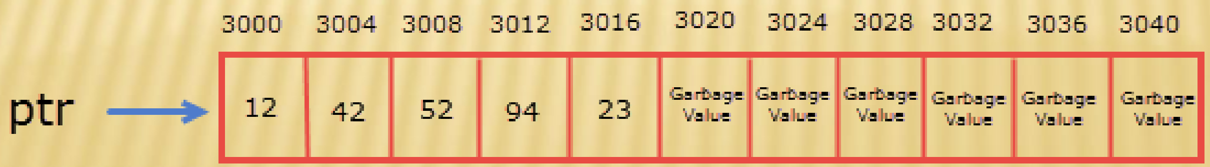
p = (int)realloc(p, 11*sizeof(int));*

Now two conditions may arise:

1st case: If sufficient memory is available after address 1016, then the address of ptr doesn't change.



2nd case: If sufficient memory is not available after address 1016, then the realloc() function allocates memory somewhere else in the heap and copies the all content from old memory block to the new memory block. In this case the address of ptr changes.



QUICK EXERCISE

- Initialize an array of 4 elements by using malloc() /calloc() and input the values.
- Resize the array to increase the size to a total of 9 integers. Initialize the newly allocated values.
- Print all the elements of the array.

SOLUTION

```
int *p, i, n;
printf("Initial size of the array is 4\n\n");
p = (int*)calloc(4, sizeof(int));
if(p==NULL)
{   printf("Memory allocation failed");
    exit(1); // exit the program
}
for(i = 0; i < 4; i++)
{   printf("Enter element at index %d: ", i);
    scanf("%d", &p[i]);
}
printf("\nIncreasing the size of the array by 5 elements ...\n ");
p = (int*)realloc(p, 9 * sizeof(int));
if(p==NULL)
{   printf("Memory allocation failed");
    exit(1); // exit the program
}
printf("\nEnter 5 more integers\n\n");
for(i = 4; i < 9; i++)
{   printf("Enter element at index %d: ", i);
    scanf("%d", &p[i]);  }
printf("\nFinal array: \n\n");
for(i = 0; i < 9; i++)
    printf("%d ", *(p+i) );
```

OUTPUT

```
Initial size of the array is 4

Enter element at index 0: 11
Enter element at index 1: 22
Enter element at index 2: 33
Enter element at index 3: 44

Increasing the size of the array by 5 elements ...

Enter 5 more integers

Enter element at index 4: 1
Enter element at index 5: 2
Enter element at index 6: 3
Enter element at index 7: 4
Enter element at index 8: 5

Final array:

11 22 33 44 1 2 3 4 5
```

FREE()

- When you declare a variable using a basic data type, the C compiler automatically allocates memory space for the variable in a pool of memory called the **stack.**
- Similarly when the program is over, all memory is deallocated by the compiler.
- When declaring a basic data type or an array, the memory is automatically managed.
- Memory allocated through dynamic memory allocation functions are done in the **heap** part of the RAM.
- In dynamic memory allocation, you have to deallocate memory explicitly.
- If not done, you may encounter out of memory error.
- **The free()** function is called to release/deallocate memory.
- By freeing memory in your program, you make more available for use later.

EXAMPLE: FREE()

```
#include <stdio.h>
int main()
{
int* ptr = malloc(10 * sizeof(*ptr));
if (ptr != NULL)
{
*(ptr + 2) = 50;
printf("Value of the 2nd integer is %d",*(ptr + 2));
}
free(ptr);
}
```

SUMMARY OF DYNAMIC MEMORY ALLOCATION

Static Memory Allocation	Dynamic Memory Allocation
Memory is allocated at compile time.	Memory is allocated at run time.
Memory is allocated in the stack part of the RAM.	Memory is allocated the heap part of the RAM
Memory can't be increased while executing program.	Memory can be increased while executing program.
Memory is destroyed/ deallocated automatically	Deallocation has to be done using free()
Used in array.	Used in linked list.

SUMMARY OF DYNAMIC MEMORY ALLOCATION

Functions	Syntax	Description
malloc	(datatype*)malloc(size);	Allocates the specified number of bytes user requested.
calloc	(datatype*) calloc(n, size of each block);	Allocates the specified number of bytes and initialize them to zero
realloc	(datatype*) realloc(ptr, size);	Modify the previously allocated memory block (increase or decrease memory size and reallocate)
free	–	Release previously allocated memory block

QUICK EXERCISE: ASSUME n=5

```
int i, n, d=8;
int *element;
printf("Enter total number of elements: ");
scanf("%d", &n);
element = (int*) calloc(n,sizeof(int));
if(element != NULL)
{
for(i = 0; i < n; i++)
{ *(element +i ) = d* 3;
d- -;
}
for(i = 1; i < n; i++)
{
if(*element > *(element+i))
*element = *(element+i);
}
 printf("%d", *element);}
```

Output:

```
12
```