Dynamic Memory Allocation in C

Memory allocation has two core types;

- Static Memory Allocation: The program is allocated memory at compile time.

  For ex :

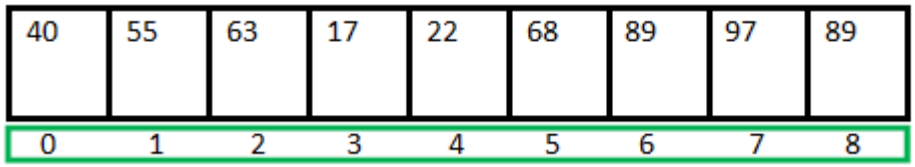
```
void main()
{
int x; //SMA
char a[5]; //SMA
}
```

- **Dynamic Memory Allocation: The programs are allocated with memory at run time.**

- Since C is a structured language, it has some fixed rules for programming.

Lets take example of Array: One of it includes changing the size of an array. “An array is collection of items stored at continuous memory locations”.

Ex:



the length (size) of the array is 9. But what if there is a requirement to change this length (size).

- *If there is a situation where only 5 elements are needed to be entered in this array. 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.*

- This procedure is referred to as Dynamic Memory Allocation 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.
- C provides some functions to achieve these tasks. 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()

C malloc() method

- **“malloc”** or **“memory allocation”** method in C 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.

Syntax:

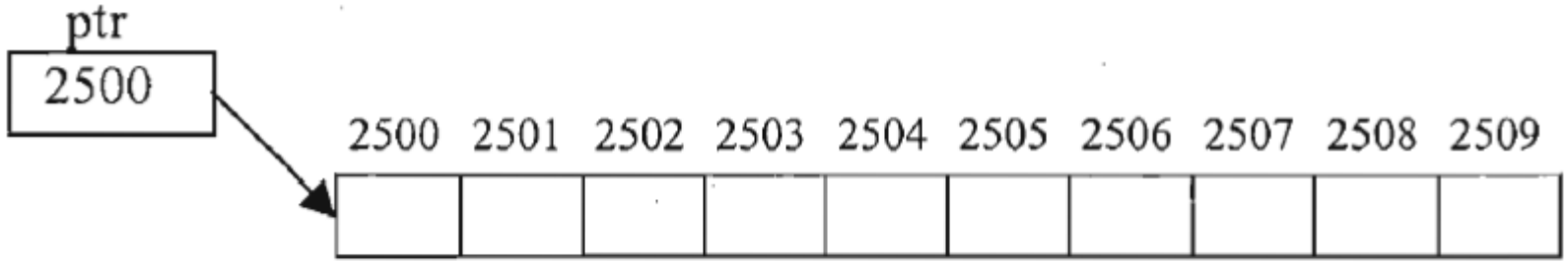
ptr = (data-type*) malloc(byte-size)

ptr = (data-type*) malloc(byte-size)

For example:

int *ptr;

ptr = (int *) malloc (l0);



- This allocates 10 contiguous bytes of memory space and the address of first byte is stored in the pointer variable ptr. This space can hold 5 integers. The allocated memory contains garbage value. We can use sizeof operator to make the program portable and more readable.

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

This allocates the memory space to hold five integer values.

If there is not sufficient memory available in heap then malloc( ) returns NULL. So we should always check the value returned by malloc( ).

```
ptr = (float *) malloc(l0*sizeof(float) );
if ( ptr = = NULL )
printf("Sufficient memory not available");
```

- NOTE: Unlike memory allocated for variables and arrays, dynamically allocated memory has no name associated with it. So it can be accessed only through pointers. We have a pointer which points to the first byte of the allocated memory and we can access the subsequent bytes using pointer arithmetic.

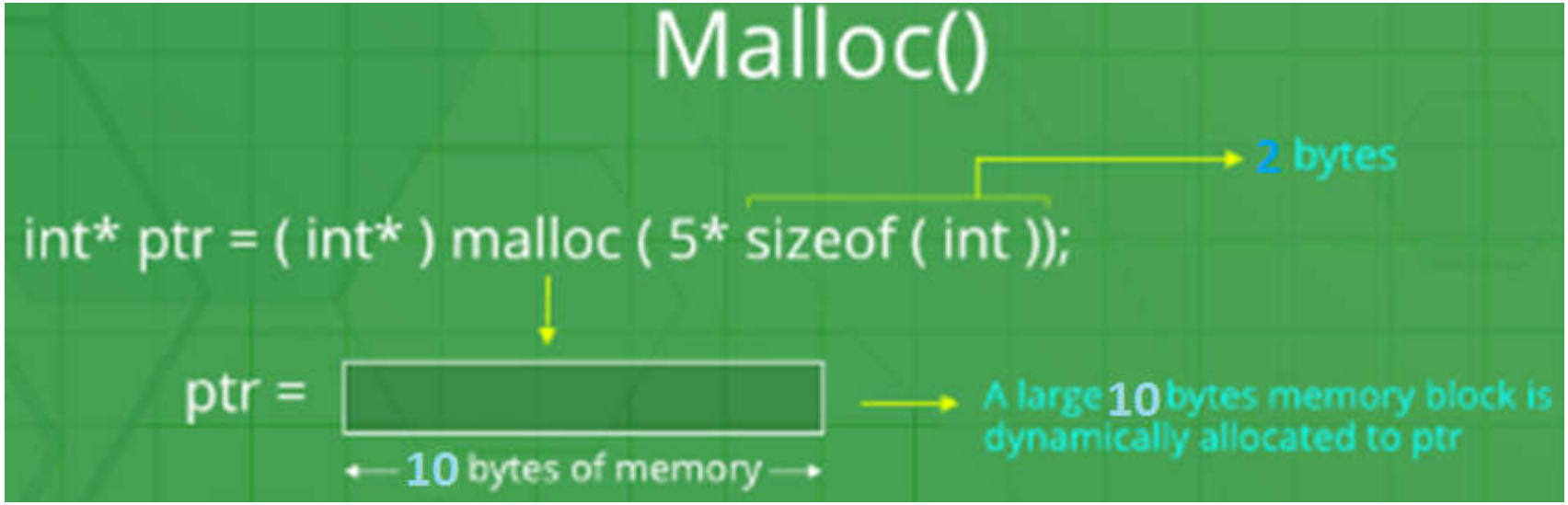
For Example:

ex: ptr = (int*)malloc(n*sizeof(int));

For Example:

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

Since the size of int is 2 bytes, this statement will allocate 200 bytes of memory. And, the pointer ptr holds the address of the first byte in the allocated memory.



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

malloc_method.c

malloc_method2.c

malloc_method.c

```
#include <stdio.h>
#include <stdlib.h>
int main()
{       int *ptr;
        int n,i,byte;
        printf("\n Enter size of memory required: ");
        scanf("%d",&byte);
        printf("\n Enter total number of elements of array: ");
        scanf("%d",&n);
        // Dynamically allocate memory using malloc()
        ptr = (int*)malloc(byte);
        // Check if the memory has been successfully
        // allocated by malloc or not
        if(ptr==NULL)
        { printf("\n Memory not allocated.\n");
                exit(0);  }
        else
        {       // Memory has been successfully allocated
                printf("\n Memory successfully allocated using malloc.\n");

                // Get the elements of the array
                printf("\n enter array elements:\n");
                for(i=0; i<n; i++)
                                scanf("%d",ptr+i);
                // Print the elements of the array
                printf("\n The elements of the array are: ");
                for(i=0; i<n; i++)
                printf(" %d",*(ptr+i));
        }
        return 0;
}
```

malloc_method2.c

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

int main()  {
        int *ptr;
        int n,i;
        printf("\n Enter number of elements: ");
        scanf("%d",&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
                    printf("enter array elements:\n");
                    for(i=0; i<n; i++)
                                scanf("%d",ptr+i);

                    // 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;
}
```

C calloc() method

- **“calloc”** or **“contiguous allocation”** method in C is used to dynamically allocate the specified number of blocks of memory of the specified type.
- The calloc( ) function is used to allocate multiple blocks of memory. It is somewhat similar to malloc( ) function.
- It takes two arguments. The first argument specifies the number of blocks and the second one specifies the size of each block.
- It initializes each block with a default value ‘0’.

Syntax:

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

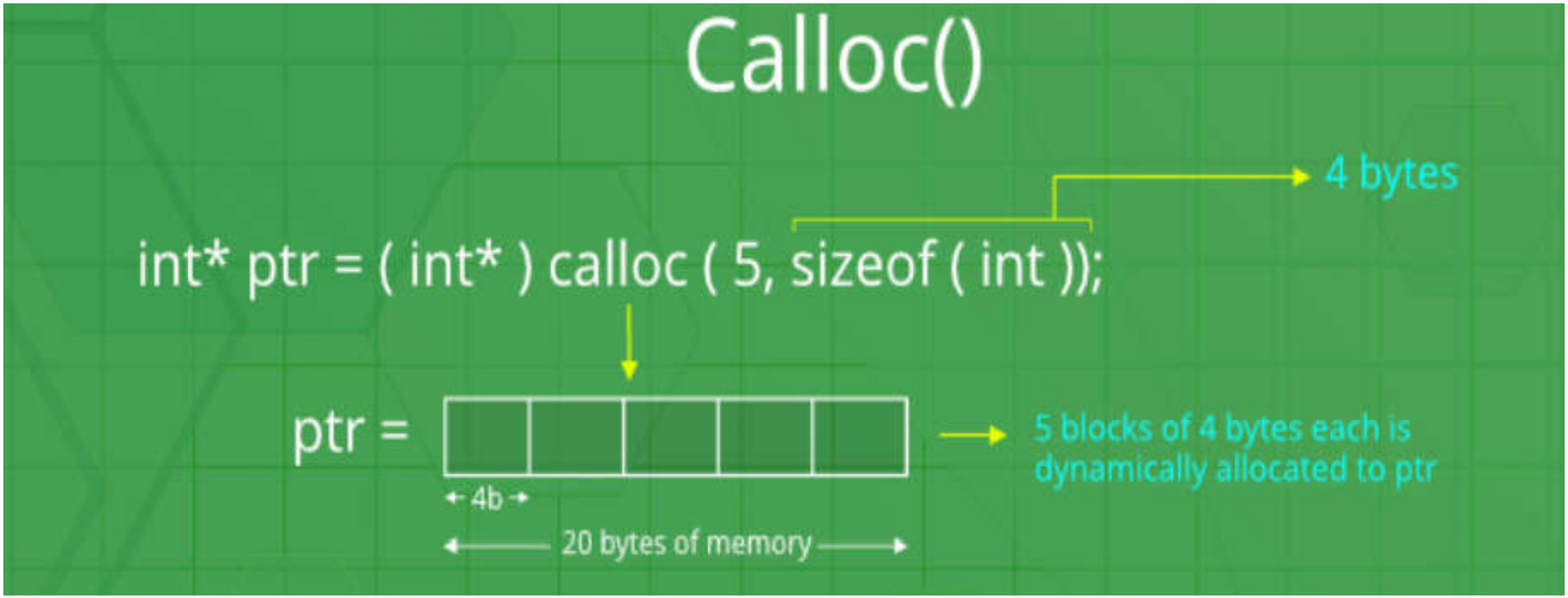
- For Example:

ptr = (int *) calloc (5 , sizeof(int));

This allocates 5 blocks of memory, each block contains 2 bytes

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

This 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_method.c

calloc_method.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;
}
```

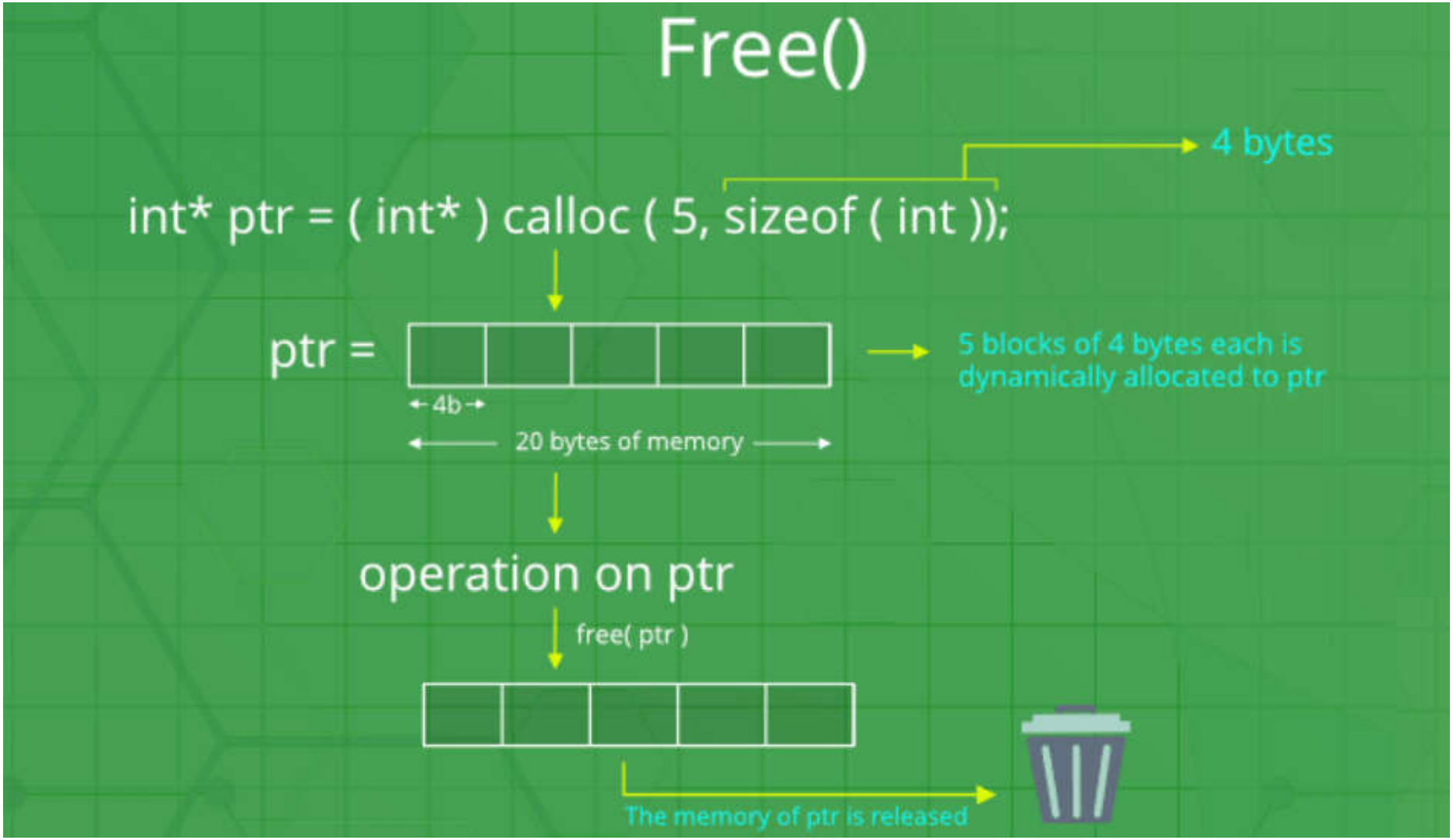
C free() method

- The dynamically allocated memory is not automatically released; it will exist till the end of program.
- If we have finished working with the memory allocated dynamically, it is our responsibility to release that memory so that it can be reused.
- The function free( ) is used to release the memory space allocated dynamically.
- The memory released by free( ) is made available to the heap again and can be used for some other purpose.

For example: free(ptr);

Here ptr is a pointer variable that contains the base address of a memory block created by malloc( ) or calloc( ). Once a memory location is freed it should not be used.

We should not try to free any memory location that was not allocated by malloc( ), calloc( ) or realloc( ).


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.c

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
        // This pointer will hold the  // base address of the block created
        int *p, *p1;
        int n, i;
        // Get the number of elements for the array
        printf("Enter number of elements: ");
        scanf("%d",&n);
        // Dynamically allocate memory using malloc()
        p=(int*)malloc(n*sizeof(int));
        // Dynamically allocate memory using calloc()
        p1=(int*)calloc(n, sizeof(int));
        // Check if the memory has been successfully
        // allocated by malloc or not
        if(p == NULL || p1 == 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(p);
                printf("Malloc Memory successfully freed.\n");

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

                // Free the memory
                free(p1);
                printf("Calloc Memory successfully freed.\n");
        }

        return 0;
}
```

C realloc() method

- **“realloc”** or **“re-allocation”** method in C is used to dynamically change the memory allocation of a previously allocated memory.
- If we want to increase or decrease the memory allocated by malloc() or calloc(). The function realloc( ) is used to change the size of the memory block.
- It alters the size of the memory block without losing the old data.

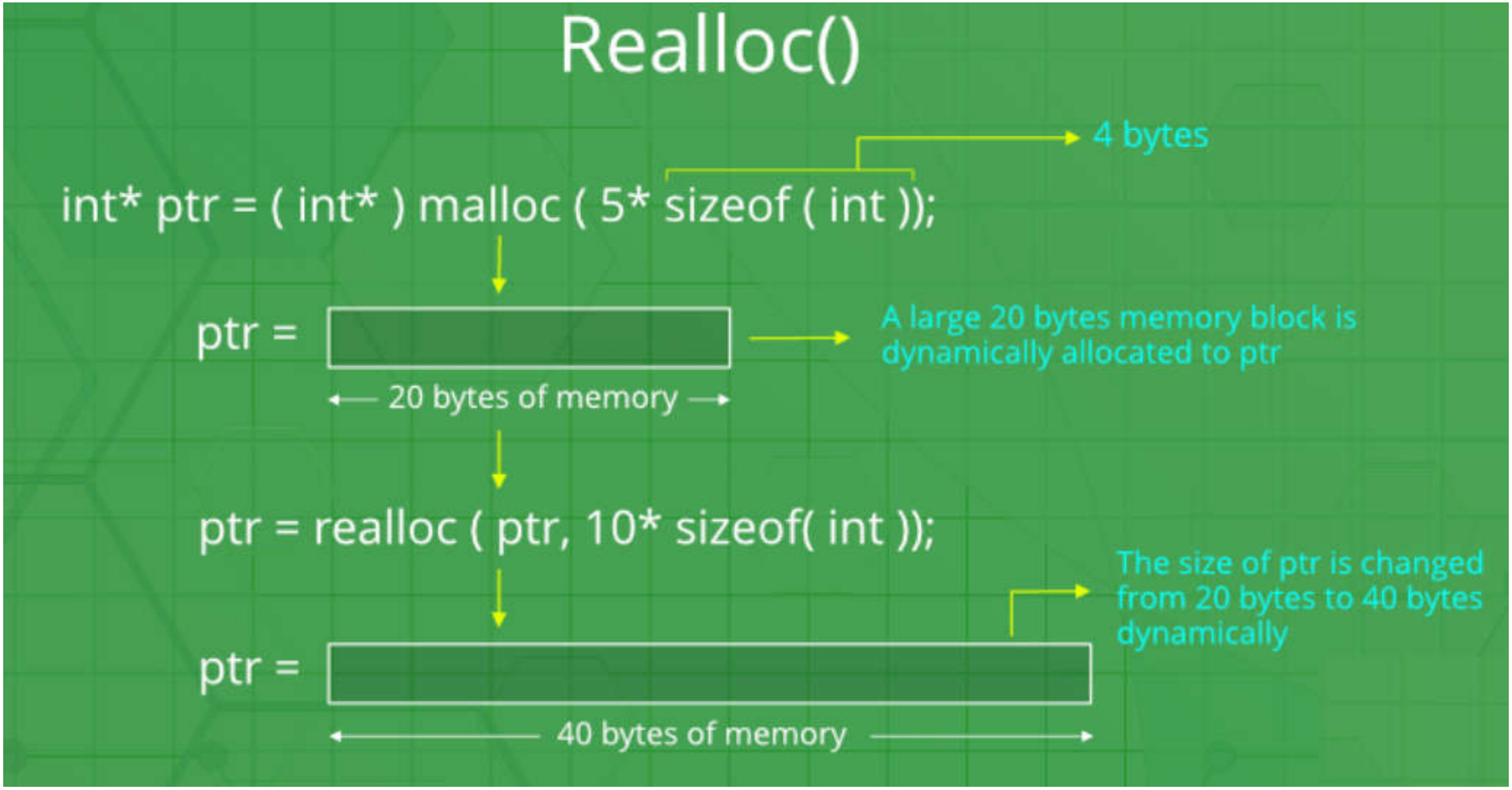
we can use realloc( ) as**:**

```
ptr = (int*) realloc ( ptr , newsize );
```

where ptr is reallocated with new size 'newSize'.

This function takes two arguments, first is a pointer to the block of memory that was previously allocated by malloc( ) or calloc( ) and second one is the new size for that block.

- This statement allocates the memory space of newsize bytes, and the starting address of this memory block is stored in the pointer variable ptr.
- The newsize may be smaller or larger than the old size. If the newsize is larger, then the old data is not lost and the newly allocated bytes are uninitialized.
- The starting address contained in ptr may change if there is not sufficient memory at the old address to store all the bytes consecutively. Then this function moves the contents of old block into the new block and the data of the old block is not lost.
- On failure, realloc( ) returns NULL.


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

relloc1.c

```
/*program to understand the use of realloc () function*/
#include<stdio.h>
#include<stdlib.h>
main ( )
{
int i,*ptr;
ptr=(int *)malloc(5*sizeof(int));
      if (ptr==NULL)
      {
      printf ("Memory not available\n");
      exit(1);
      }

      printf ("Enter 5 integers ") ;
      for(i=0;i<5;i++)
      scanf("%d",ptr+i);
      ptr= (int *) realloc (ptr, 9 *sizeof (int) ); /*Allocate memory for 4 more integers*/
if (ptr==NULL)
{
printf ("Memory not available\n");
exit(1);
}
      printf ("Enter 4 more integers: ");
      for(i=5;i<9;i++)
      scanf("%d",ptr+i);
            for(i=0;i<9;i++)
            printf ("%d ",*(ptr+i));
}
```

realloc.c

```
/ *program to understand the use of realloc () function* /
#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);
	}
```

realloc.c

```
else {
        printf("Memory successfully allocated using calloc.\n");
                for (i = 0; i < n; ++i) {
                ptr[i] = i + 1;
                }
                printf("The elements of the array are: ");       // Print the elements of the array
                for (i = 0; i < n; ++i) {
                                printf("%d, ", ptr[i]);
                }
                n = 10;
                printf("\n\nEnter the new size of the array: %d\n", n);

                ptr = realloc(ptr, n * sizeof(int));

                printf("Memory successfully re-allocated using realloc.\n");

                for (i = 5; i < n; ++i) {
                ptr[i] = i + 1;
                }
                printf("The elements of the array are: ");
                for (i = 0; i < n; ++i) {
                printf("%d, ", ptr[i]);
                }
                free(ptr);
        }
        return 0;  }
```