



IT 112: Introduction to Programming

Dr. Manish Khare

Dr. Bakul Gohel

Lecture 23,24



Structure in Structure

- Here we will learn to handle structure in a structure in C programming language.
- We have already learned how to create and access members of a structures in the structure tutorial.
- Now, lets add a structure inside a structure and learn how to work with it.

In the following example we have a student structure. Inside it we have another structure address to hold the address detail.

```
// student structure having address structure inside
struct student {
           char name[255];
           char id[20];
           // address of the student
           struct {
                  char line1[255];
                  char line2[255];
                  char city[255];
                  char state[100];
                  char country[255];
                  char pincode[20];
                   } address;
           };
```

- In the above code snippet we can see that we have an address structure inside the student structure.
- We already know by now that to access any member of a structure variable we use the operator.
- So, if we want to access the name member of the student structure we have to write the following.

std.name

Similarly if we want to access the city member of the address structure which is inside the student structure, we have to write the following.

std.address.city

> code we will take student detail from the user and then print the output. (5)

Pointers and Structures

Here we will learn to use pointers with structures in C Programming Language.

Creating a structure

Lets start by creating a structure variable student as shown below.

```
// student structure
struct student {
  char id[15];
  char firstname[64];
  char lastname[64];
  float points;
};
```

Now we will create a student structure variable std. For this we will write the following code.

// student structure variable struct student std;

Access the members of a structure

- We know that to access a member of a structure we use the . operator.
- In the following example we are accessing the members of the student structure.

```
printf("ID: %s\n", std.id);
printf("First Name: %s\n", std.firstname);
printf("Last Name: %s\n", std.lastname);
printf("Points: %f\n", std.points);
```

Creating pointer for structure

Following is the syntax to create a pointer for a structure. struct tagName *ptrName;

So, to create a pointer for the student structure we will write the following code. struct student *ptr;

Assigning structure variable to pointer

We use the following syntax to assign a structure variable address to a pointer.

In the following example we are assigning the address of the structure variable std to the structure pointer variable ptr. So, ptr is pointing at std.

$$ptr = &std$$

Accessing the members of a structure via pointer

We use the arrow operator also known as member selection operator -> to access the members of a structure via pointer variable.

Following is the syntax for accessing members of a structure via pointer. ptrName->member

In the following example we are accessing the firstname member of the student structure via pointer variable ptr.

printf("First Name: %s", ptr->firstname);

> Code (6)

Pointers and Array of Structures

- Create an array of structure variable
- In the following example we are considering the student structure that we created in the previous tutorial and we are creating an array of student structure variable std of size 3 to hold details of three students.

```
// student structure
struct student
          char id[15];
          char firstname[64];
          char lastname[64];
          float points;
// student structure variable
struct student std[3];
```

We can represent the std array variable as following.

```
struct student {
    char id[15];
    char firstname[64];
    char lastname[64];
    float points;
};
```

	id	firstname	lastname	points
std[0]				
std[1]				
std[2]				

Create pointer variable for structure

Now we will create a pointer variable that will hold the starting address of the student structure variable std.

```
// student structure pointer variable
struct student *ptr = NULL;

// assign std to ptr
ptr = std;
```

Note! std is an array variable and the name of the array variable points at the memory location so, we are assigning it to the structure pointer variable ptr.

- Accessing each element of the structure array variable via pointer
- For this we will first set the pointer variable ptr to point at the starting memory location of std variable. For this we write ptr = std;
- Then, we can increment the pointer variable using increment operator ptr++ to make the pointer point at the next element of the structure array variable i.e., from str[0] to str[1].
- We will loop three times as there are three students. So, we will increment pointer variable twice. First increment will move pointer ptr from std[0] to std[1] and the second increment will move pointer ptr from std[1] to std[2].
- To reset the pointer variable ptr to point at the starting memory location of structure variable std we write ptr = std;.

 Code (7)

Passing structure pointer to function

- Create a structure
- In the following example are are creating a student structure.

Function declaration to accept structure pointer

- Following is the syntax of the function declaration that accepts structure pointer. returnType functionName(struct tagName *);
- returnType is the return type of the function functionName. If the function is not returning anything then set it to void. The function takes structure tagName pointer.
- In the following example we are creating two function declarations that takes address of student structure.

```
void getDetail(struct student *);
void displayDetail(struct student *);
```

- Creating an array of structure variable
- We will now create an array of student structure variable by writing the following code.

// student structure variable
struct student std[3];

So, we have created an array of student structure variable of size 3 to store the detail of three students.

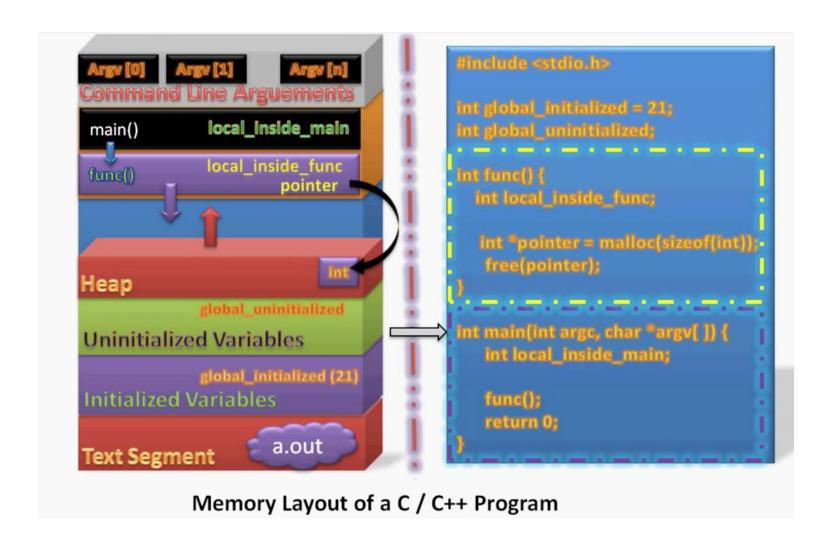
We can represent the std array variable as follows.



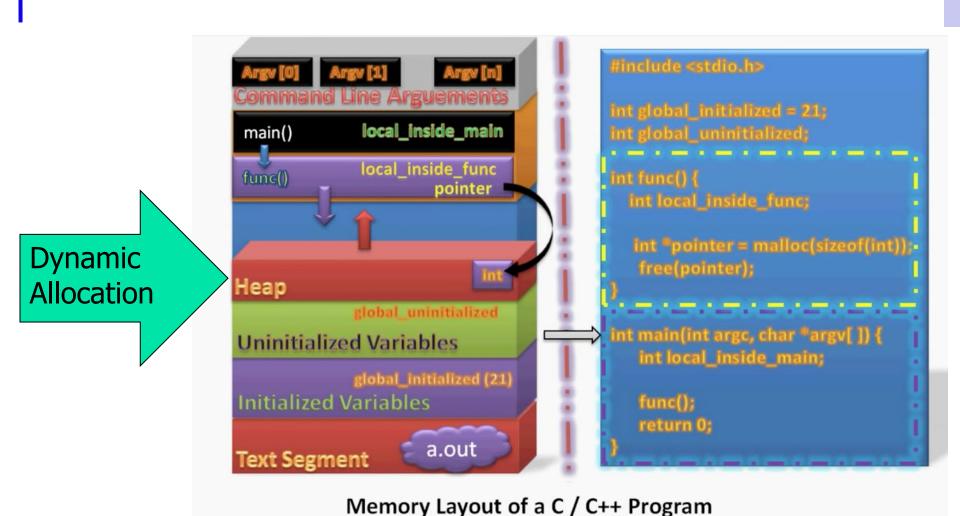
DYNAMIC list(array)

- YOU NEED TO IMPLIMENT ARRAY (like) VARIABLE, BUT SIZE IS NOT KNOWN and SIZE IS DYNAMICALLY CAHNGING
- For example, customer list, inventory and many more....
- ➤ What will your approach? Array?
 - Array elements need to be stored at contiguous memory locations
 - Run time resizing of the array is expensive
 - If array is not full,

Dynamic Allocation of Memory



Dynamic Allocation of Memory

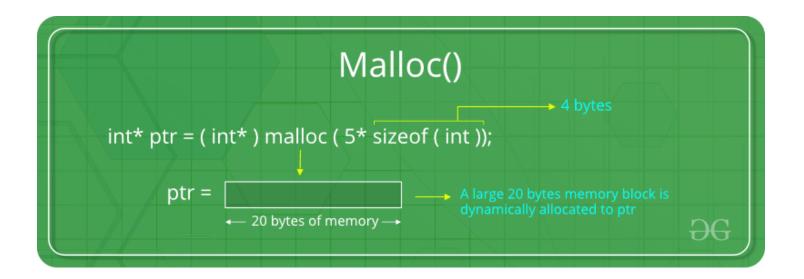


Malloc()

void * malloc(unsigned Int NumBytes)

Typical Use

DataType * ptr = (DataType*) malloc(byte-size)

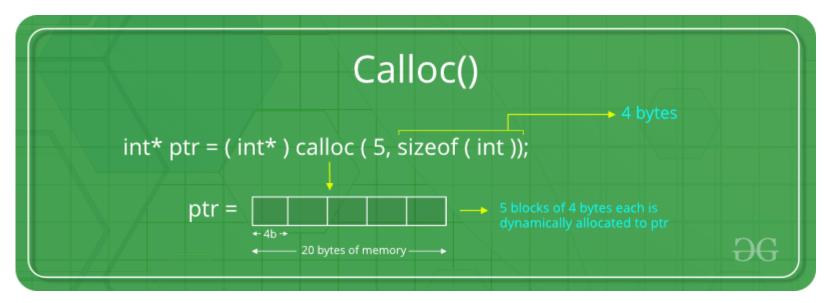


Calloc()

void * malloc(unsigned Int num_elements, unsigned Int element size)

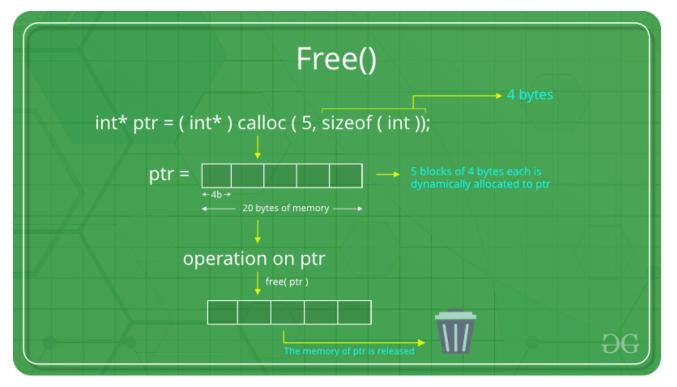
Typical Use

DataType * ptr = (DataType*) calloc(N, sizeof(DataType))



free()

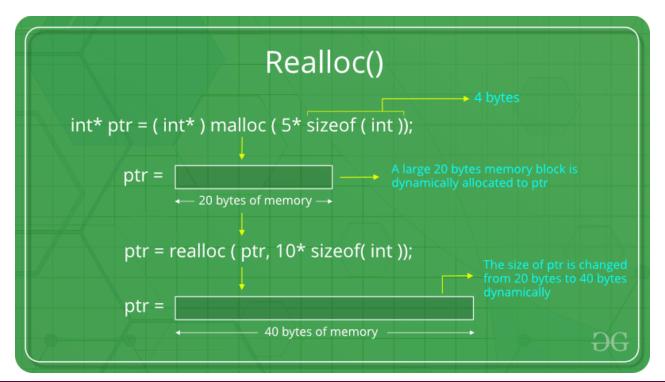
De-allocate the memory (allocated by malloc() and calloc()) ones it's use is over to prevent wastage of memory free(ptr)



Realloc()

- Re-allocation or extention of already allocated memory
- maintains the already present value

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



Dynamic Memory Allocation

Whenever space is insufficient during memory allocation malloc(),calloc(), and realloc() return NULL value