**Dynamic Array in C**

Dynamic arrays are very useful data structures. They can be initialized with variable size at runtime. This size can be modified later in the program to expand (or) shrink the array. Unlike fixed-size arrays and Variable Length Arrays, Dynamically sized arrays are allocated in a heap. Flexible Array Members closely resemble dynamic-sized arrays. They're used in structs and provide a data member which has similar properties as dynamically sized arrays.

## **Fixed Length Arrays**

Fixed-length arrays, as the name implies, their size has to be a compile-time constant. It cannot be changed during the execution of the program.

These are not ideal in many cases. Most of the time, we need to create an array of variable sizes. Of course, we can create an array of the largest possible size, but that would be a waste of memory. VLA's address this issue by accepting runtime size for arrays.

## **Variable Length Arrays (VLA) in C**

Variable Length Arrays are arrays whose size is determined at runtime (still the size cannot be modified after initialization). They were first brought into C by C99 standard. But it was made an optional feature in later versions. They were allocated on the stack and freed when they leave the scope (just like normal arrays).

#include <stdio.h>

int main()

{

int n;

printf("Size of the array: ");

scanf("%d", &n);

int arr[n];

printf("Created an array of size %lu\n", sizeof(arr) / sizeof(arr[0]));

return 0;

}

* In fixed-length arrays, the n should be compile-time constant, whereas, here n is based on user input.

**So, Why Exactly Usage of VLA's is Discouraged?**

Even though VLAs can be initialized with dynamic size, size cannot be modified after initialization. There is no straightforward way to detect whether VLA is properly allocated. If the size is too large, the program halts with Segmentation Fault. They also generate much more Assembly code, and it's slower than static(fixed size) arrays.

There are a few restrictions on VLA usage. VLA's cannot be:

* extern
* struct members
* static
* declared with unspecified bounds

## **Introduction to Dynamic Sized Arrays**

Unlike other high-level languages (Python, JavaScript, etc), C doesn't have built-in dynamic arrays. But, It provides a way to interact with the raw memory (In other words, to shoot in the foot).

Dynamic arrays are resizable and provide random access for their elements. They can be initialized with variable size, and their size can be modified later in the program. Dynamic arrays are allocated on the heap, whereas VLAs are allocated on the stack.

It's important to note that, VLAs aren't the same as dynamic arrays. Some of the key differences are:-

* **Scope:** VLA's behave like normal arrays, and they are bounded by scope. Dynamic arrays can be used anywhere in the program, regardless of scope, until free() is called.
* **Allocation:** VLAs are allocated on the stack, whereas dynamic arrays are allocated on the heap. So, VLAs are faster than dynamic memory. Since the compiler has to cleanup memory (for dynamic array) after usage.
* **Performance:** Dynamic-sized arrays are often slow because we have to allocate required memory and deallocate manually. In terms of space, Dynamic-sized arrays are optimal because we can expand/shrink the size unlike with VLAs. Some performance benchmarks are [here](https://stackoverflow.com/questions/58848183/dynamic-array-on-stack-vla-vs-heap-performance).

## **Building Blocks**

We can use a few C functions such as malloc, free, calloc, realloc, reallocarray to implement dynamic-sized arrays.

## **Flexible Array Members (FAM)**

FAM is an array data member, with which we can initialize the array without bounds. The size of FAM is flexible and can be controlled with malloc, calloc etc. (just like dynamic arrays).

This is also standardized in C99 along with VLAs. There are a few restrictions to use FAMs:-

* There must be at least one other data member.
* The Flexible Array Member should be declared at the end of the struct.
* There can be at most one FMA in a struct (This is obvious from the fact that only one member can be at the end).

The size of the array can be decided at the time of creating an object from the struct.

**Flexible Array Member Example**

#include <stdio.h>

#include <stdlib.h>

typedef struct vector {

int size;

int arr[];

}

vector;

int main()

{

int n;

printf("Size of the arr: ");

scanf("%d", &n);

vector \*ptr = (vector \* ) malloc(sizeof(vector \* ) + n \* sizeof(int));

// This implies 'ptr->arr' can hold `n` integers.

printf("Enter the elements (space/newline separated): ");

for (int i = 0; i < n; i++)

scanf("%d", ptr -> arr + i);

printf("Given array: ");

for (int i = 0; i < n; i++)

printf("%d ", ptr -> arr[i]);

printf("\n");

free(ptr);

return 0;

}

Size of the arr: 4

Enter the elements (space/newline separated): 1 2 3 4

Given array: 1 2 3 4

* While allocating memory with malloc, note that we have added extra n \* sizeof(int) bytes. This memory is utilized by the arr data member of struct, and it also specifies the size of the Flexible array member( in this case arr).
* In this way, we can create dynamic arrays in structs.

It seems like there is no big advantage of FMAs, Because we can use pointers as a data member and allocate dynamic memory. But there is one feeble advantage. If we move the struct block, the array(data member) moves along with the struct, since the array (data member) is allocated along with the struct (and they're in the same block). Whereas, if we use a pointer as a data member, struct and array are not guaranteed to be in the same block. So, we have to move them individually.

## **Conclusion**

* Size of Variable Length Arrays can be set at runtime, but we could not change their size once set.
* Unlike static arrays, Dynamic arrays in C are allocated on the heap and we could change their size in runtime. We need to deallocate their memory after use ourselves.
* Dynamic arrays are slower than static arrays.