Difference between Static Memory Allocation and Dynamic Memory allocation

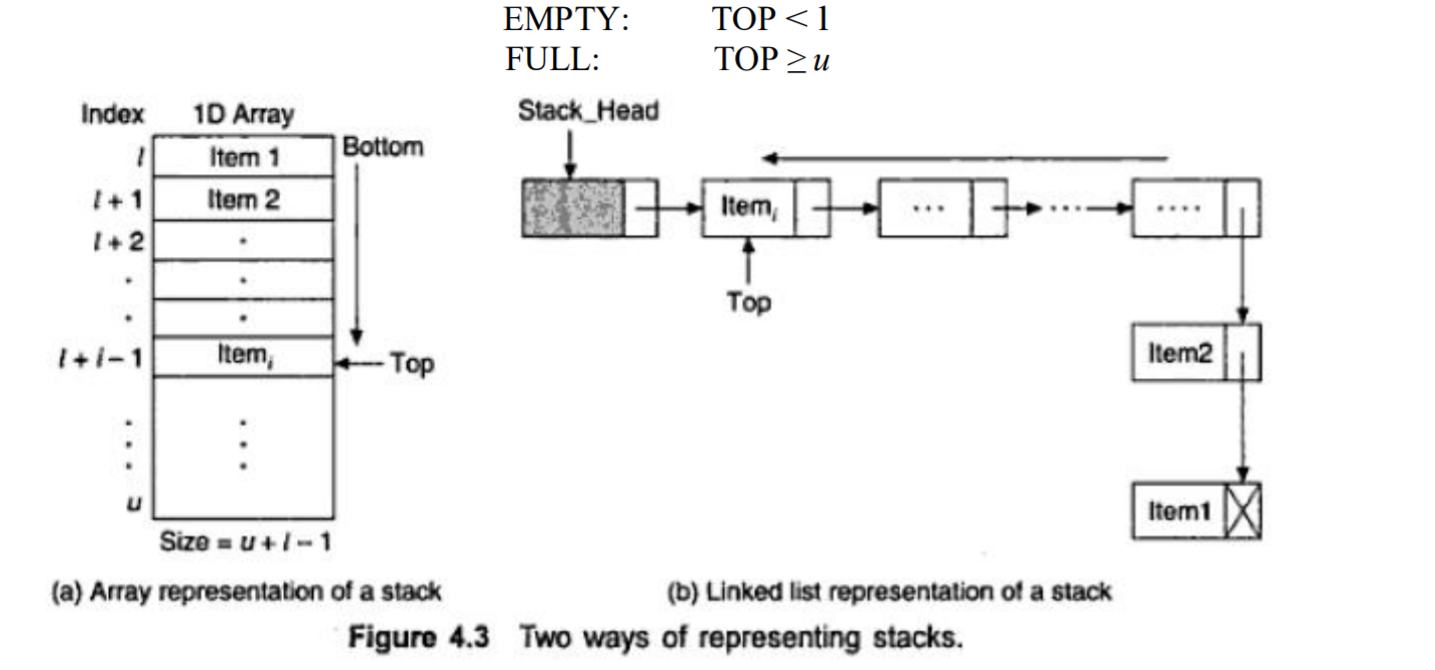
|  |  |  |
| --- | --- | --- |
| **S.No** | **Static Memory Allocation** | **Dynamic Memory Allocation** |
| 1 | In the static memory allocation, variables get allocated permanently. | In the Dynamic memory allocation, variables get allocated only if your program unit gets active. |
| 2 | Static Memory Allocation is done before program execution. | Dynamic Memory Allocation is done during program execution. |
| 3 | It uses [stack](http://www.geeksforgeeks.org/stack-data-structure/) for managing the static allocation of memory | It uses [heap](https://www.geeksforgeeks.org/heap-data-structure/) for managing the dynamic allocation of memory |
| 4 | It is less efficient | It is more efficient |
| 5 | In Static Memory Allocation, there is no memory re-usability | In Dynamic Memory Allocation, there is memory re-usability and memory can be freed when not required |
| 6 | In static memory allocation, once the memory is allocated, the memory size cannot change. | In dynamic memory allocation, when memory is allocated the memory size can be changed. |
| 7 | In this memory allocation scheme, we cannot reuse the unused memory. | This allows reusing the memory. The user can allocate more memory when required. Also, the user can release the memory when the user needs it. |
| 8 | In this memory allocation scheme, execution is faster than dynamic memory allocation. | In this memory allocation scheme, execution is slower than static memory allocation. |
| 9 | In this memory is allocated at compile time. | In this memory is allocated at run time. |
| 10 | In this allocated memory remains from start to end of the program. | In this allocated memory can be released at any time during the program. |
| 11 | **Example:** This static memory allocation is generally used for [array](https://www.geeksforgeeks.org/introduction-to-arrays/). | **Example:** This dynamic memory allocation is generally used for [linked list](http://www.geeksforgeeks.org/data-structures/linked-list/). |

**REPRESENTATION OF A STACK**

A stack may be represented in the memory in various ways. There are two main ways: using a one-dimensional array and a single linked list.

**Array Representation of Stacks:** First we have to allocate a memory block of sufficient size to accommodate the full capacity of the stack. Then, starting from the first location of the memory block, the items of the stack can be stored in a sequential fashion.

In Figure, Itemi denotes the ith item in the stack; l and u denote the index range of the array in use; usually the values of these indices are 1 and SIZE respectively. TOP is a pointer to point the position of the array up to which it is filled with the items of the stack. With this representation, the following two ways can be stated:



**Linked List Representation of Stacks**: Although array representation of stacks is very easy and convenient but it allows the representation of only fixed sized stacks. In several applications, the size of the stack may vary during program execution. An obvious solution to this problem is to represent a stack using a linked list. A single linked list structure is sufficient to represent any stack. Here, the DATA field is for the ITEM, and the LINK field is, as usual, to point to the next' item. Above Figure b depicts such a stack using a single linked list.

