DYNAMIC DATA STRUCTURE

Data Structure

A data structure is the logical and mathematical model of a particular organization of data.

Two types:

1. Static data structure - an organization or collection of data in memory that is fixed in size.

   Examples: Array, struct

2. Dynamic data structures – an organization or collection of data in memory that has the flexibility to grow or shrink in size, enabling a programmer to control exactly how much memory is utilized.

   Examples: Linked list, stack, queue, binary trees

Static data structures are ideal for storing a fixed number of data items, but they lack the dynamic data structure's flexibility to consume additional memory if needed or free up memory when possible for improved efficiency.

Stack and Heap Memory

Stack is used for static memory allocation and Heap for dynamic memory allocation, both stored in the computer's RAM.

Variables allocated on the stack are stored directly to the memory and access to this memory is very fast, and its allocation is dealt with when the program is compiled. The stack is always reserved in a LIFO (last-in-first-out) order; the most recently reserved block is always the next block to be freed. This makes it really simple to keep track of the stack, freeing a block from the stack is nothing more than adjusting one pointer.

Variables allocated on the heap have their memory allocated at run time and accessing this memory is a bit slower, but the heap size is only limited by the size of virtual memory. You can allocate a block at any time and free it at any time.

Ordinary variables are stored in the stack, while memory allocated using malloc and calloc are in the heap.

Dynamic Memory Allocation

To allocate memory dynamically, the following functions under the stdlib.h library can be used:

1. malloc()
   The name "malloc" stands for memory allocation.
   The malloc() function reserves a block of memory of the specified number of bytes. And, it returns a pointer to void which can be casted into pointers of any form.
   Syntax:

   ```
   ptr = (castType*) malloc (size);
   ```

   Example:

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

   The above example dynamically allocates memory for an int.

2. calloc()
   The name "calloc" stands for contiguous allocation.
   The malloc() function allocates memory and leaves the memory uninitialized. Whereas, the calloc() function allocates memory and initializes all bits to zero.
   Syntax:
   ptr = (castType*) calloc (n,size);
   Example:
   ptr = (int*) calloc (n,sizeof(int));
   The above example dynamically allocates and initializes 10 elements of type int.
3. free – deallocates memory
   Syntax:
   free(ptr);

Dynamic Array

In general, an array is a collection of a fixed memory locations. You can only change it during design time. But C is capable of changing the size of an array during run-time which is known as dynamic memory allocation.

Example:

Dynamically create an array of 10 integers.

First declare a pointer variable of type int (only for array of integers). *Why?* Because, memory allocated in the heap doesn't have a name, so the only way to access it is to have a static variable (like ptr) that will point to the elements in the heap.

```
int * ptr;
```

To allocate,

```
ptr = (int*) malloc(sizeof(int) * 10);
```

To dissect some parts (highlighted) and explain:

ptr = (int*) malloc(sizeof(int) * 10);

- gets the size of the type to be allocated

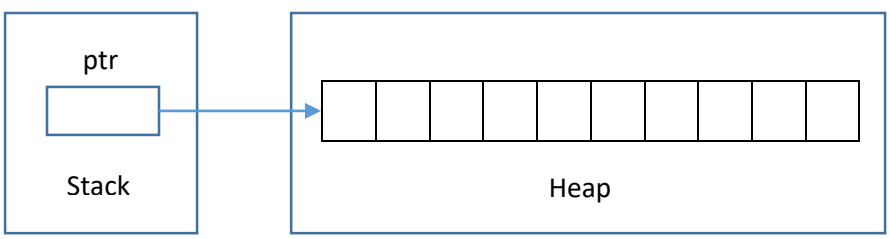
ptr = (int*) malloc(sizeof(int) * 10);

- size of the array

ptr = (int*) malloc(sizeof(int) * 10);

- Remember that malloc returns a void*, so we cast it with the appropriate type.

Illustration on the memory representation:



Another example:

Create an array of 100 double values.

Answer:

double *ptr;

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

Here is a sample program to demonstrate the use of a dynamic array:

```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10

int main(int argc, char *argv[]) {
    int *ptr, i;
    ptr = (int*) malloc(sizeof(int) * SIZE);
    for(i=0;i<SIZE;i++)
        scanf("%d",&ptr[i]);
    for(i=0;i<SIZE;i++)
        printf("%d ",ptr[i]);
    int sum=0;
    for(i=0;i<SIZE;i++)
        sum+=ptr[i];
    printf(" = %d",sum);
    return 0;
}
```

The only difference between a static array and dynamic array is how they are created. Access and other processes are the same because of the subscript notation of a pointer.

Therefore, if you are going to pass a dynamic array to a function, do the same as how a static array is done. For example,

```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10

int main(int argc, char *argv[]) {
    int *ptr;
    int count = 0;
    ptr = (int*) malloc(sizeof(int) * SIZE);
    add(ptr, &count, 5);
    add(ptr, &count, 10);
    add(ptr, &count, 15);
    display(ptr, count);
    return 0;
}
```