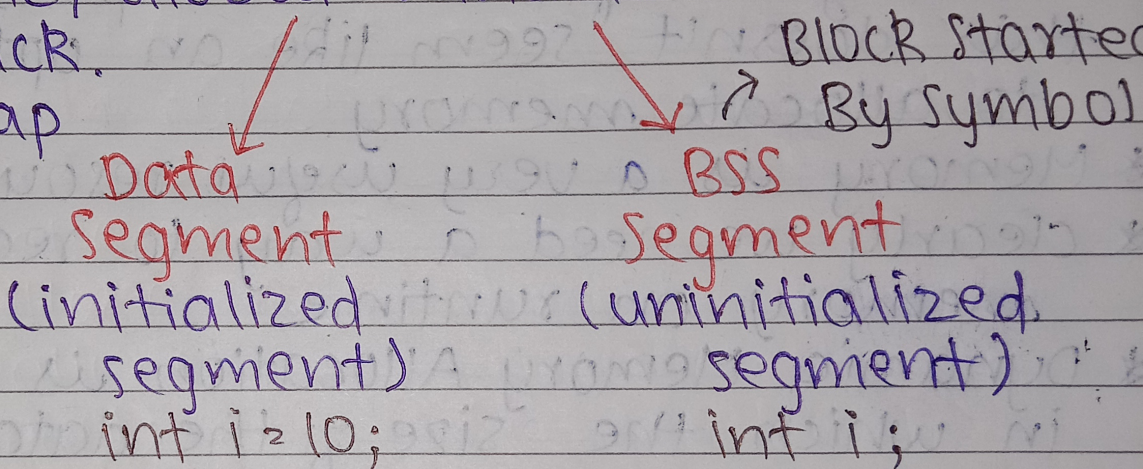
[Memory Layout of C Program]

* An statically allocated variable or array has fixed sized in memory.
* we have learnt to create big enough arrays to fit in our inputs but this doesn't seem like an optimal way to allocate memory.
* Memory is a very useful resource.
* clearly we need a way to request memory on runtime.
* Dynamic Memory Allocation is a way in which the size of the data structure can be changed during the runtime.

Static Memory Alloc	Dynamic Memory Alloc
i) Allocation is done before the program execution.	Allocation is done during the program execution.
ii) There is no memory reusability & the memory allocated can not be free.	There is memory reusability & the allocated memory can be freed when not required.
iii) Less efficient.	More efficient.

Memory assigned to a program in a typical architecture can be broken down into 4 segments :-

i) code → Text Segment.
ii) Static / Global variable
iii) Stack.
iv) Heap



Stack overflow :-

* compiler allocates some space for the stack part of the memory.
* when this space gets exhaused for some bad reason, this situation is called as stackoverflow.
* Example, Recursion with wrong or no base condition.

use of Heap :-

* There are a lot of limitations of stack (static memory allocation).
* Example, variable sized array, freeing memory no longer required, etc.
* Heap can be used flexibly by the programmer as per his needs.

How to use Heap :-

* we can create a pointer in our main function & point to a memory block in the head/heap.
* The address is stored by the local variable in the main function.
* The memory consumed will not get freed automatically in case we overwrite the pointer.

Dynamic Memory Allocation

* In dynamic memory allocation, the memory is allocated at runtime from the heap segment.

* There are 4 functions to achieve dynamic memory:-
  i) malloc()
  ii) calloc()
  iii) realloc()
  iv) free()

  } dynamic memory allocation comes under <stdlib.h> header file, i.e., we should use :-
  #include <stdlib.h>

(i) malloc() function :-

i) malloc() stands for memory allocation.
ii) It reserves a block of memory with the given amount of bytes.
iii) The return value is a void pointer to the allocated space.
iv) However, if the space is insufficient allocation of memory fails & it returns a null pointer.
v) The void pointer needs to be casted to the approached type as per the requirement.

vi) All the values at allocated memory are initialized to garbage value.

Syntax :-

```
ptr = (ptr - type *) malloc
                    (size_in_bytes)
```

Example :-

```
int *ptr;
ptr = (int *) malloc (3 * sizeof
                        (int));
```

(2) calloc() function :-

i) calloc() function stands for contigous allocation.
ii) It reserves 'n' blocks of memory with the given amount of bytes.
iii) The return value is a void pointer to the allocated space.
iv) Therefore the void pointer needs to be casted to the appropriate type as per the requirements.
v) However, if the space is insufficient, allocation of memory fails & it returns a null pointer.
vi) All the values at allocated memory are initialized to 0.

Syntax :-

```
ptr = (ptr-type *) calloc (n, size
                           in_bytes)
```

Example :-

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

* malloc() vs. calloc()

malloc()	calloc()
malloc() function creates a single block of memory of a specific size.	calloc() function assigns multiple block of memory to a single variable.
The number of args in malloc is 1.	The number of args in calloc is 2.
malloc() is faster.	calloc() is slower.
malloc() is high time efficiency.	calloc() is low time efficiency.
The memory block allocated by malloc has a garbage value.	The memory block allocated by calloc is initialized by 0.

(3) realloc() function :-

i) realloc() stands for reallocation.
ii) If the dynamically allocated memory in insufficient we can change the size of previously allocated memory using realloc() function.

Syntax :-

```
ptr = (ptr-type *) realloc (ptr,
                    new_size_in_bytes)
```

Examples :-

```
int *ptr;
ptr = (int *) calloc (4, sizeof(int));
ptr = (int *) realloc (ptr, 6);
```

→ First allocate memory using calloc() function.

→ Then reallocate more memory to the same pointer using realloc() function.

(9) free() function :-

i) free() is used to free the allocated memory.

ii) If the dynamically allocated memory is not required anymore, we can free it using free() function.

iii) This will free the memory being used by the program in the heap.

Syntax :-

```
free(ptr);
```

* Program using malloc() function

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int *ptr;
    int n;
    printf("enter the size of the array
        you want to create = ");
    scanf("%d", &n);
    ptr = (int *) malloc(n * sizeof(int))
    for(int i=0; i<n; i++)
    {
        printf("enter the value no %d
            of this array", i);
        scanf("%d", &ptr[i]);
    }
    for(int i=0; i<n; i++)
    {
        printf("the value at %d of this
            array is %d", i, ptr[i]);
    }
    return 0;
}
```

* Program using calloc() function.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int *ptr;
    int n;
    printf("enter the size of the array
       you want to create");
    scanf("%d", &n);
    ptr = (int *) calloc (n, sizeof(int));
    for(int i=0; i<n; i++)
    {
        printf("enter the value no %d
           of this array", i);
        scanf("%d",&ptr[i]);
    }
    for(int i=0; i<n; i++)
    {
        printf("the value at %d of this
           array is %d", i, ptr[i]);
    }
    return 0;
}
```