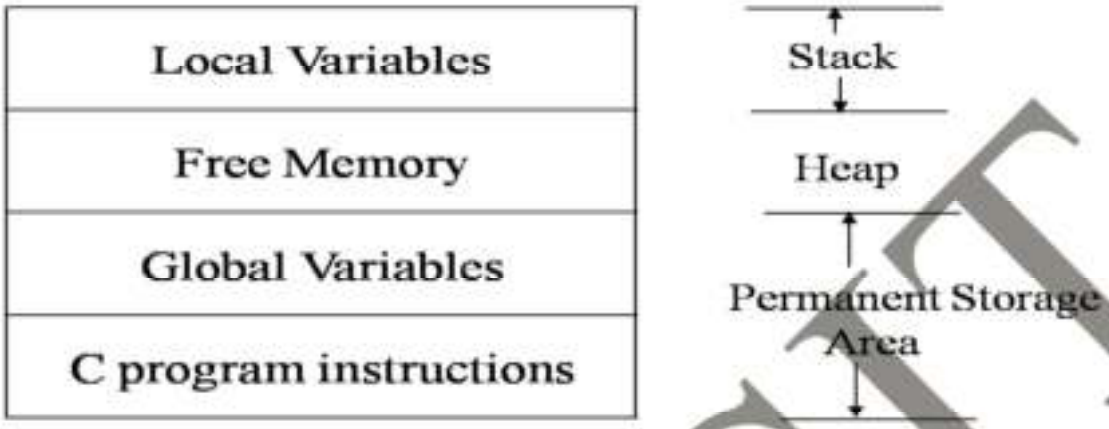
Topic of today's discussion – "Dynamic memory allocation"

Let us OBSERVE the following figure again –



Storage of a C program in memory

- Permanent Storage Area: The program instructions and the global variables are stored in this region.
- Stack: It is used for storing local variables of functions, return addresses at the function calls, arguments passed to the function. It also stores the current state of CPU.
- Heap: This is a region of free memory. It is available for dynamic allocation during execution of the program. The size of the heap keeps on changing.

REMEMBER –
Heap region of the memory is used for dynamic allocation during execution of the program.

There are **two** types of **memory allocation** –

- ✓ Static Memory Allocation.
- ✓ Dynamic Memory Allocation.

Static Memory Allocation: The **process of allocating** memory **at compile time** is known as **Static Memory Allocation**.

Example: **int marks[10];**
Here **marks** is an array with 10 elements. If we assume integer takes 2 bytes of memory to store integer data, then 20 bytes will be allocated at the compile time.

Problem of Static Memory Allocation:
If our initial judgment about the size of the array is wrong, then there will be **either** wastage **or** shortage of memory space.

Such problem can be avoided by allocating memory at run time.

Dynamic Memory Allocation:
The **process of allocating** memory **at run time** is known as **Dynamic Memory Allocation**.

C language provides **four library functions** for allocating and freeing allocated memory during program execution.

The functions are malloc( ), calloc( ), realloc( ), free( ).

The prototypes of all these functions are declared in **alloc.h** or **stdlib.h**.

Example:

```
ptr = (int *) malloc (20 * sizeof(int)) ;
```

On successful execution of this statement, a memory space in bytes equivalent to "20 times the size of an **int**" is allocated in heap and the address of the first byte of the memory allocated is assigned to the integer pointer **ptr**.

Let us NOW discuss the above **4 functions**.

1. malloc( ):

   The **malloc** function reserves a single block of memory of specified size in bytes and returns a pointer of type **void** so that it can casted to desired type.

   Example:

```
int main()
{
        int * a;
        int n;
        --------
        a = (int *) malloc (n * sizeof(int)) ;
        --------

}
```

   On successful execution of this statement, a memory space in bytes equivalent to "n times the size of an **int**" is allocated and the address of the first byte of the memory allocated is assigned to the integer pointer **a**. If there is not enough space in heap region, a **NULL pointer** is returned.

2. **calloc( ):**

   The **Calloc** function allocates multiple blocks of storage, each of the same size, and then the storage is initialized to zero.

   Example:

```
int main()
{
        int * a;
        int n;
        --------
        a = (int *) calloc (n , sizeof(int)) ;
        --------

}
```

   On successful execution of this statement, contiguous memory space is allocated for **n** blocks, each of size sizeof(**int**) bytes. All bytes are initialized to zero and a pointer to the first byte of the allocated region is

assigned to the pointer **a** of type **int**. If there is not enough space, a NULL pointer is returned.

3. **free():**

It is used to release the block(s) of memory created by **malloc( )** or **calloc( )** function. The general form is

free(**ptr**);

where **ptr** is a pointer to a memory block created dynamically.

[Note that dynamically allocated memory space is not released automatically upon function exit. Hence, in case of dynamic memory allocation, programmer MUST use free() **explicitly** to release the memory space.]

4. **realloc( ):**

If the previously dynamically allocated memory becomes insufficient then we can modify (either increase or reduce) the memory size using the **realloc( )** function.

For example, if original allocation is done by the statement

ptr = (char *) malloc(**oldsize**);

then reallocation of space may be done by the statement

ptr = (char *) realloc(**ptr, newsize**);

Program 1:

Write a C program to find average of n integer numbers given by user. Use malloc() function to store values given by user.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i,n,*p,sum=0;
    float avg;
    printf("\nHow many numbers? ");
    scanf("%d", &n);

    p = (int * ) malloc(n * sizeof(int));

    printf("\nEnter %d numbers : \n",n);
    for(i = 0; i <n; i++)
    {
        scanf("%d", p+i);
        sum += *(p+i);
    }

    avg = sum/(float)n;
    printf("\n Average of following %d numbers = %f\n", n, avg);
    for(i = 0;i < n;i++)
    {
        printf("\t%d", *(p+i));
    }
    free(p);
    return(0);
}
```

Input:

How many numbers? **4**

Enter 4 numbers :

4

2

3

1

Output:

Average of following 4 numbers = 2.500000

4 2 3 1

Compare calloc() and malloc():

calloc()	malloc()
It is used to allocate memory at run time.	It is also used to allocate memory at run time.
It reserves multiple memory blocks, each of same size in bytes and returns a pointer of type void.	It reserves a block of memory of specified size in bytes and returns a pointer of type void.
This function takes 2 arguments. For example: **calloc(n, m);** where n = number of blocks and m = size of each block in bytes	This function takes 1 argument. For example: **malloc(nb);** where nb = block size in bytes
It initializes the contents of the each block to zero.	It does not initialize.
It uses free() to de-allocate.	It also uses free() to de-allocate.
It can be resized by realloc() function.	It can also be resized by realloc().

Review Problem 1:

Use calloc() function in place of malloc() in the above program 1. Check whether you are getting same output as program 1.

Review Problem 2:

Carefully observe the following program. Now answer the following questions based on the given program –

a) State the purpose of using header file stdlib.h?
b) What is the purpose of using malloc()?
c) Specify the purpose of using realloc()?
d) Why a programmer must use free() explicitly?
e) State the outputs of the program.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
        char *s = NULL;
        s = (char *) malloc(10 * sizeof(char));
        strcpy(s,"Hello");
        printf("\nSpace created and its Content is %s\n", s);
        s = (char *) realloc(s, 25);
        printf("Space reallocated and its content is %s\n", s);
        strcpy(s,"Hello How are You");
        printf("New content of reallocated space is %s\n", s);
        free(s);
        return(0);
}
```
