02

Structure and Self-referential Structure, Dynamic memory allocation and Project planning

DATA STRUCTURE LAB

SESSION - 02

CSE135 | DAFFODIL INTERNATIONAL UNIVERSITY

Dynamic Memory Allocation:

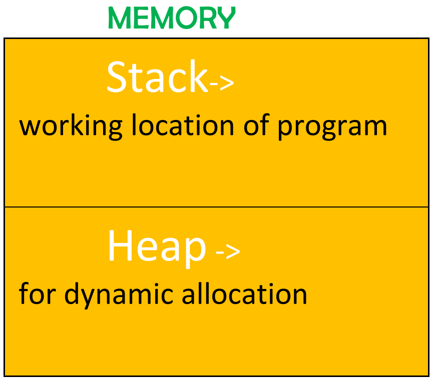
It’s the process that allows your program to obtain more memory space while running, or to release memory if that is not required. In brief dynamic memory allocation allows you to manually handle memory space for your program.

In C language there are 4 library functions under **"stdlib.h"** header file for dynamic memory allocation.

Function	Use of function
malloc()	Allocates requested size of bytes and returns a pointer first byte of allocated space
calloc()	Allocates space for an array elements, initializes to zero and then returns a pointer to memory
free()	deallocate the previously allocated space
realloc()	Change the size of previously allocated space

We normally use malloc() function for Dynamic Memory Allocation.

- This malloc function allocate memory in Byte (from the “Heap”)



MEMORY

Stack->
working location of program

Heap ->
for dynamic allocation

Memory in a program is divided into two parts:

- **The stack:** All variables declared inside the function will take up memory from the stack.
- **The heap:** This is unused memory of the program and can be used to allocate the memory dynamically when program runs.

Syntax of malloc()

```
ptr = (cast-type*) malloc(byte-size);
```

Here, ***ptr*** is pointer of cast-type. The **malloc()** function returns a pointer to an area of memory with size of byte size. If the space is insufficient, allocation fails and returns NULL pointer.

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

This statement will allocate either n*2 or n*4 according to size of int 2 or 4 bytes respectively and the pointer points to the address of first byte of memory.

**Simple program:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int *p , n = 10 , i;
    p = (int*)malloc(n*sizeof(int));   //here we allocate memory in heap
    for(i = 0 ; i< n ; i++)
    {
        p[i] = i;
    }
    for(i = 0; i<n ; i++)
    {
        printf("%d\n",p[i]);
    }
    return 0;
}
```

SELF REFERENTIAL STRUCTURE

If a structure contains one or more pointers to itself (a structure of same type) as its members, then the structure is said to be a self-referential structure, that is, a structure that contains a reference to its own structure type.

In brief, One of the member of the structure is pointer of the same type.

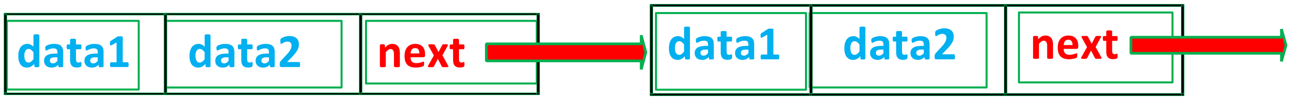
For example:

```
struct node                  //{here marked portion is “data type}
{
  int data1;
  int data2;
  struct node *next; //{here red marked portion is “self referential
member”}
};
```

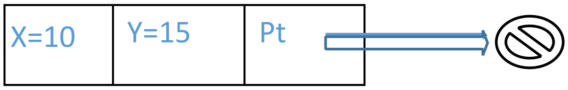
Here, the structure '*node*' contains a pointer named '*next*', which is of the same type (struct node) as the structure it contains in (struct node).

The above illustrated structure prototype describes one node that comprises of two logical segments. One of them stores data/information and the other one is a pointer indicating where the next component can be found. .Several such inter-connected nodes create a chain of structures.

The following figure depicts the composition of such a node.



#simple program on creating nodes and assigning data:



```
#include<stdio.h>

#include<stdlib.h>

struct Node

{

    int x,y;

    struct Node *next;

};

typedef struct Node  node;

int main()

{

    node *head;

    head=(node*)malloc(sizeof(node));

    head->x=10;

    head->y=15;

    head->pt=NULL;

    printf("%d %d\n",head->x,head->y);

    return 0;

}
```