**Data Structures**

Dynamic Memory Allocation

**Static Memory:**

* Also called as Fixed memory.
* We cannot modify the allocated memory.
* We cannot change(Increment or Decrement) the allocated memory by using particular functions.

Examle:- int a,int a[100],char s[100].

#include<stdio.h>  
**int** main(){  
 **int** n,i=0;  
 scanf("%d",&n);  
 **int** array[n];  
 **while**(i<n) {  
 printf("%d",i+1);  
 i++;  
 }  
}

**What is Dynamic Memory Allocation:**

* Sometimes the size of an array can be less or more than required size.
* Dynamic Memory allocation allows a program to gain more memory space or to release it if it's not required at the time of program execution(running).
* C language doesn’t have its own technique of Allocating memory dynamically,but it have some library functions for Dynamic Memory Allocation and they are “malloc()”,”calloc()”,”realloc()”,”free()”.

|  |  |
| --- | --- |
| function | About the Function |
| malloc() | Allocates required amount of memory and return the pointer to the first byte(returns the address of f.b) |
| calloc() | Similar to malloc ,but initializes all the elements of allocated memory to zero. |
| realloc() | Increases or decreases the previously allocated space. |
| free() | It free up(release) the Dynamically allocated memory. |

**Malloc:**

* malloc function:- void\* malloc(size\_t size)
* return a void pointer which means a generic pointer which can be typecasted into required pointer.
* Instead of giving size of a datatype directly as an argument of the function malloc().we can use the function sizeof(“datatype”).

#include<stdio.h>  
#include<stdlib.h>  
  
**typedef struct** date{  
 **int** day;  
 **char** month[50];  
 **int** year;  
}date;  
**int** main(){  
 date \*DOB=malloc(**sizeof**(date));  
 **return** 0;  
}

Pointer to structure

Return type void pointer

* We have to type cast the void pointer into the other pointer.

=>date \*DOB=(date\*)malloc(sizeof(data));

* General syntax for allocating memory Dynamically through malloc is

(type-cast\*)malloc(n\*sizeof(datatype)).

#include<stdio.h>  
#include<stdlib.h>  
  
**int** main(){  
 **int** \*array=(**int**\*)malloc(3\***sizeof**(**int**));  
 \*array=1;  
 \*(array+1)=2;  
 \*(array+2)=3;  
 printf("%d %d %d",array[0],array[1],array[2]);  
}

* Malloc function return the address of the first byte of allocated memory.

#include<stdio.h>  
#include<stdlib.h>  
  
**int** main(){  
 **int** i=0,n;  
 scanf("%d",&n);  
 **int** \*array=(**int**\*)malloc(n\***sizeof**(**int**));  
 **while**(i<n){  
 array[i]=i+1;  
 printf("%d ",array[i]);  
 i++;  
 }  
 **return** 0;  
}

**Calloc:**

* Very similar to malloc but differs in syntax and initialization.

(type-cast\*)calloc(number,sizeof(datatype));

#include<stdio.h>  
#include<stdlib.h>  
  
**int** main(){  
 **int** i=0,n;  
 scanf("%d",&n);  
 **int** \*array=(**int**\*)calloc(n,**sizeof**(**int**));  
 **while**(i<n){  
 printf("%d ",array[i]);  
 i++;  
 }  
 **return** 0;  
}

1. 0 0 0 0

* Here calloc() has initialized all the elements with 0.

**realloc():**

* pointer=realloc(pointer, new\_size);

**int** \*array=(**int**\*)calloc(n,**sizeof**(**int**));  
array=(**int**\*)realloc(array,2\*n);

* here we have re-allocated the memory by increasing its size two times.

**free():**

* Syntax:- free(pointer);

**int** main(){  
 **int** i=0,n;  
 scanf("%d",&n);  
 **int** \*array=(**int**\*)calloc(n,**sizeof**(**int**));  
 array=(**int**\*)realloc(array,2\*n);  
 **while**(i<n){  
 array[i]=i+1;  
 i++;  
 }  
 i=0;  
 free(array);  
 **while**(i<n){  
 printf("%d ",array[i]);  
 i++;  
 }  
 **return** 0;  
}

* Output -some garbage values or a compilation error.