**Dynamic Memory Allocation :**

**Used When we want to allocate memory during run time.**

**int marks[10]; // fixed size and fixed address … No change in Memory address.**

**// fixed size. ( no change in size possible**

**We have to use <stdlib.h> hadder file for dynamic memory allocation.**

**It has 4 functions.**

1. **malloc()**
2. **calloc()**
3. **free()**
4. **realloc()**

**malloc()   
memory allocation**

**allocate the one memory block given by user. // eg. Reserves 20bytes of block**

**calloc()**

**creates number of blocks. // uses for arrays**

**free()**

**used to free the space after using malloc() or alloc()**

**realloc()**

**if used malloc() or alloc() and need to modified memory block size realloc()**

**\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**malloc()**

**creates the memory block according to given size ().**

**malloc() function Also returns the address , which points the address of the first byte in that specific block.**

**Syntax :**

**void \* maclloc(size in byte );**

**as it has void pointer as return type it can return Any type of data : int , string, char.**

**ptr = (cast\_type\*) malloc( size in byte);**

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

**you must cast the pointer according to type of data eg. Here.. (int \*)**

**here, ptr will be int type.**

**int – 2 byte**

**it contains garbage value. And here it can hold 5 int values if one int requires 2 bytes**

**in case because of the some problem if memory is not allocated by malloc() function than It will return null pointer.**

**#include<stdio.h>**

**#include<conio.h>**

**#include<stdlib.h>**

**main()**

**{**

**int n, \*ptr, sum =0, i, \*p;**

**printf(“Enter the size of array”);**

**scanf(“%d”,&n);**

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

**// ptr will point the first byte of the memory block.**

**// now we can use null pointer to see block is created or not.**

**if( ptr == NULL)**

**{**

**printf(“Error : out of Memory”);**

**exit(0);**

**}**

**p = ptr;**

**// right now both have the same address.**

**printf(“Enter the elements in Array”);**

**for( i= 1; i<=n; i++)**

**{**

**scanf(“%d”,ptr);**

**sum = sum + \*ptr;**

**ptr++;**

**}**

**printf(“Array Elements : “);**

**for(i=1; i<=n; i++)**

**{**

**Printf(“%d”,\*p);**

**p++;**

**}**

**printf(“addition is %d”, sum);**

**}**

**calloc()**

**malloc ()creates only one block.. while calloc() can create multiple blocks.**

**calloc() can be used for arrays.**

**void \*calloc( number\_of\_blocks, size for each block in bytes);**

**Syntax :**

**pointer = (Data\_Type\*) calloc(n,Size in bytes);**

**// here function calloc() returns the address of first byte of first block.**

**// malloc() has garbage value in all variables while calloc initionalize with 0.**

**// returns null pointer if block is not created successfully.**

**Example Programm for calloc() in C**

**#include<stdio.h>**

**#include<stdlib.h> // malloc(), calloc() and other functions are here in this file.**

**main()**

**{**

**int n, \*ptr, \*p, i, sum=0;**

**printf( “number of elements to be entered”);**

**scanf(“%d”,&n);**

**ptr = (int \*)calloc(n, sizeof(int));**

**p= ptr;**

**if(ptr == NULL)**

**{**

**printf(“memory block is not created successfully);**

**exit(0); // 0 means normal termination.**

**}**

**printf(“enter %d elements”,n);**

**for(i =1; i<=n; i++)**

**{**

**scanf(“%d”,ptr);**

**sum = sum + \*ptr;**

**ptr++;**

**}**

**printf(“Elements are “);**

**for(i =1; i<=n; i++)**

**{**

**printf( “%d”, \*p);**

**p++;**

**}**

**printf(“ Addition = %d”,sum);**

**free(ptr); // free can be used to free the memory so that we can use that memory in other program.**

**}**

**// calloc has 2 arguments and can create more than 1 block.**

**}**