Problem Solving With C

UE15CS151

Dynamic Memory Management

Dynamic Memory Management

The exact size of array is unknown until the compile time, i.e., time when a compiler compiles code written in a programming language into a executable form. The size of array you have declared initially can be sometimes insufficient and sometimes more than required. Dynamic memory allocation allows a program to obtain more memory space, while running or to release space when no space is required.

Although, C language inherently does not has any technique to allocated memory dynamically, there are 4 library functions under "stdlib.h" for dynamic memory allocation.

Function	**Use of the 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()	de-allocate the previously allocated space
realloc()	Change the size of previously allocated space

malloc():

Syntax : 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.

example:

```
int *p=NULL;
p = (int *) malloc(400);          //        p = (int *) malloc(100*sizeof(int));
```

This statement will allocate either 200 or 400 according to size of int 2 or 4 bytes respectively and the pointer points to the address of first byte of memory

C Program to Demonstrate the use of malloc():

Example 1:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main()
{
        char *s=NULL;                          //Null pointer
        if((s=(char *)malloc(24))==NULL)       // Checking if memory is available
        {
                printf("Out of memory\n");
        }

strcpy(s,"PES UNIVERSITY");               //Assignment
printf("%p\n",s);                         //Displaying the Address
printf("%s\n",s);

free(s);                                  //De-allocating the memory
s=NULL;
}

Output      :       PES UNIVERSITY
```

Example 2:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int *p=NULL;
        if((p=(int *)malloc(sizeof(int)*1))==NULL)
        {
                printf("Out of memory\n");
        }
printf("%p\n",p);
*p=10;
printf("%d\n",*p);
free(p);
p=NULL;
}
Output      :       Address
                    10
```

Example 3:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int *p=NULL;
        int n;
        int i;
        printf("Enter n:\n");
        scanf("%d",&n);
        if((p=(int *)malloc(sizeof(int)*n))==NULL)
        {
                printf("Out of memory\n");
        }
        printf("%p\n",p);
        for(i=0;i<n;i++)
        {
                scanf("%d",&p[i]);
        }
        printf("Array contents\n");
        for(i=0;i<n;i++)
        {
                printf("%d\n",p[i]);
        }
        free(p);
        p=NULL;
}
```

```
Output      :      Enter n : 3
                   111    222    333

                   Array Contents:
                   111    222    333
```

Example 4:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int *p=NULL;
        int n;
        int i;
        printf("Enter n:\n");
        scanf("%d",&n);
        if((p=(int *)malloc(sizeof(int)*n))==NULL)
        {
                printf("Out of memory\n");
        }
        printf("%p\n",p);
        for(i=0;i<n;i++)
        {
                scanf("%d",&p[i]);
        }
        printf("Array contents\n");
        for(i=0;i<n;i++)
        {
                printf("%d\n",p[i]);
        }
        *p= 500;
        printf("Edited Array contents\n");
        for(i=0;i<n;i++)
        {
                printf("%d\n",p[i]);
        }
        free(p);
        p=NULL;
}
```

```
Output      :       Enter n : 3
                    111     222     333

                    Array Contents:
                    111     222     333

                    Edited Array Contents:
                    500     222     333
```

Calloc():

The name calloc stands for "contiguous allocation". The only difference between malloc() and calloc() is that, malloc() allocates single block of memory whereas calloc() allocates multiple blocks of memory each of same size and sets all bytes to zero.

Syntax : ptr = (cast-type)calloc(n,element-size);

This statement will allocate contiguous space in memory for an array of n elements.

```
float *p = NULL;
p = (float *)calloc(25,sizeof(float));
```

This statement allocates contiguous space in memory for an array of 25 elements each of size of float.

Example 1:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int *p=NULL;
        int n;
        int i;
        printf("Enter n:\n");
        scanf("%d",&n);

        if((p=(int *)calloc(n,sizeof(int)*n))==NULL)
        {
                printf("Out of memory\n");
        }

        printf("%p\n",p);
        for(i=0;i<n;i++)
        {
                scanf("%d",&p[i]);
        }
```

```
            printf(“Array Contents\n”);
            for(i=0;i<n*n;i++)
            {
                    printf("%d\n",p[i]);
            }
            free(p);
            p=NULL;
    }
```

```
    Output          :       Enter n : 3
                            111     222     333

                            Array Contents:
                            111     222     333
```

free():

Dynamically allocated memory with either calloc() or malloc() does not get return on its own. The programmer must use free() explicitly to release space

Syntax : free(ptr);

realloc():

If the previously allocated memory is insufficient or more than sufficient. Then, you can change memory size previously allocated using realloc().

Syntax : ptr = realloc(ptr,size);

Example 1:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        char *s;
        if((s=malloc(6))==NULL)
        {
                printf("Out of memory\n");
        }
        strcpy(s,"Hello");
        s=(char *)realloc(s,14);
        strcpy(s,"Hello welcome");
        printf("%s\n",s);

}
```

Output : Hello welcome