C - MEMORY MANAGEMENT

This chapter will explain dynamic memory management in C. The C programming language provides several functions for memory allocation and management. These functions can be found in the **<stdlib.h>** header file.

S.N.	Function and Description
1	**void *calloc(int num, int size);** This function allocates an array of **num** elements each of which size in bytes will be **size**.
2	**void free(void *address);** This function release a block of memory block specified by address.
3	**void *malloc(int num);** This function allocates an array of **num** bytes and leave them initialized.
4	**void *realloc(void *address, int newsize);** This function re-allocates memory extending it upto **newsize**.

Allocating Memory Dynamically

While doing programming, if you are aware about the size of an array, then it is easy and you can define it as an array. For example to store a name of any person, it can go max 100 characters so you can define something as follows:

```
char name[100];
```

But now let us consider a situation where you have no idea about the length of the text you need to store, for example you want to store a detailed description about a topic. Here we need to define a pointer to character without defining how much memory is required and later based on requirement we can allocate memory as shown in the below example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
   char name[100];
   char *description;

   strcpy(name, "Zara Ali");

   /* allocate memory dynamically */
   description = malloc( 200 * sizeof(char) );
   if( description == NULL )
   {
      fprintf(stderr, "Error - unable to allocate required memory\n");
   }
   else
   {
      strcpy( description, "Zara ali a DPS student in class 10th");
   }
   printf("Name = %s\n", name );
   printf("Description: %s\n", description );
}
```

When the above code is compiled and executed, it produces the following result.

```
Name = Zara Ali
```

```
Description: Zara ali a DPS student in class 10th
```

Same program can be written using **calloc()** only thing you need to replace malloc with calloc as follows:

```
calloc(200, sizeof(char));
```

So you have complete control and you can pass any size value while allocating memory unlike arrays where once you defined the size can not be changed.

Resizing and Releasing Memory

When your program comes out, operating system automatically release all the memory allocated by your program but as a good practice when you are not in need of memory anymore then you should release that memory by calling the function **free()**.

Alternatively, you can increase or decrease the size of an allocated memory block by calling the function **realloc()**. Let us check the above program once again and make use of realloc() and free() functions:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
   char name[100];
   char *description;

   strcpy(name, "Zara Ali");

   /* allocate memory dynamically */
   description = malloc( 30 * sizeof(char) );
   if( description == NULL )
   {
      fprintf(stderr, "Error - unable to allocate required memory\n");
   }
   else
   {
      strcpy( description, "Zara ali a DPS student.");
   }
   /* suppose you want to store bigger description */
   description = realloc( description, 100 * sizeof(char) );
   if( description == NULL )
   {
      fprintf(stderr, "Error - unable to allocate required memory\n");
   }
   else
   {
      strcat( description, "She is in class 10th");
   }

   printf("Name = %s\n", name );
   printf("Description: %s\n", description );

   /* release memory using free() function */
   free(description);
}
```

When the above code is compiled and executed, it produces the following result.

```
Name = Zara Ali
Description: Zara ali a DPS student.She is in class 10th
```

You can try above example without re-allocating extra memory and strcat() function will give an error due to lack of available memory in description.