C Dynamic Memory Allocation

C malloc()

The name "malloc" stands for memory allocation.

The `malloc()` function reserves a block of memory of the specified number of bytes. And, it returns a pointer of `void` which can be casted into pointers of any form.

The malloc() function allocates single block of requested memory.

It doesn't initialize memory at execution time, so it has garbage value initially.

It returns NULL if memory is not sufficient.

The syntax of malloc() function is given below:

Syntax of malloc()

```
ptr = (castType*) malloc(size);
```

Example

```
ptr = (float*) malloc(100 * sizeof(float));
```

The above statement allocates 400 bytes of memory. It's because the size of `float` is 4 bytes. And, the pointer *ptr* holds the address of the first byte in the allocated memory

The expression results in a `NULL` pointer if the memory cannot be allocated.

Let's see the example of malloc() function.

```
1.  #include<stdio.h>
2.  #include<stdlib.h>
3.  int main(){
4.    int n,i,*ptr,sum=0;
5.      printf("Enter number of elements: ");
6.      scanf("%d",&n);
7.      ptr=(int*)malloc(n*sizeof(int));  //memory allocated using malloc
8.      if(ptr==NULL)
9.      {
10.         printf("Sorry! unable to allocate memory");
11.         exit(0);
12.     }
```

```
13.     printf("Enter elements of array: ");
14.     for(i=0;i<n;++i)
15.     {
16.         scanf("%d",ptr+i);
17.         sum+=*(ptr+i);
18.     }
19.     printf("Sum=%d",sum);
20.     free(ptr);
21. return 0;
22. }
```

Output

```
Enter elements of array: 3
Enter elements of array: 10
10
10
Sum=30
```

C calloc()

The name "calloc" stands for contiguous allocation.

The `malloc()` function allocates memory and leaves the memory uninitialized. Whereas, the `calloc()` function allocates memory and initializes all bits to zero.

Syntax of calloc()

```
ptr = (castType*)calloc(n, size);
```

Example:

```
ptr = (float*) calloc(25, sizeof(float));
```

The above statement allocates contiguous space in memory for 25 elements of type `float`.

The calloc() function allocates multiple block of requested memory.

It initially initialize all bytes to zero.

It returns NULL if memory is not sufficient.

The syntax of calloc() function is given below:

1. ptr=(cast-type*)calloc(number, byte-size)

Let's see the example of calloc() function.

```
#include<stdio.h>
#include<stdlib.h>
int main(){
 int n,i,*ptr,sum=0;
    printf("Enter number of elements: ");
    scanf("%d",&n);
    ptr=(int*)calloc(n,sizeof(int));  //memory allocated using calloc
    if(ptr==NULL)
    {
        printf("Sorry! unable to allocate memory");
        exit(0);
    }
    printf("Enter elements of array: ");
    for(i=0;i<n;++i)
    {
        scanf("%d",ptr+i);
        sum+=*(ptr+i);
    }
    printf("Sum=%d",sum);
    free(ptr);
return 0;
}
```

Output

```
Enter elements of array: 3
Enter elements of array: 10
10
10
Sum=30
```

realloc

“realloc” or **“re-allocation”** method in C is used to dynamically change the memory allocation of a previously allocated memory. In other words, if the memory previously allocated with the help of malloc or calloc is insufficient, realloc can be used to **dynamically re-allocate memory**. re-allocation of memory maintains the already present value and new blocks will be initialized with default garbage value.

Syntax:

```
ptr = realloc(ptr, newSize);
```

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

int main()
{

    // This pointer will hold the
    // base address of the block created
    int* ptr;
    int n, i;

    // Get the number of elements for the array
    n = 5;
    printf("Enter number of elements: %d\n", n);

    // Dynamically allocate memory using calloc()
    ptr = (int*)calloc(n, sizeof(int));

    // Check if the memory has been successfully
    // allocated by malloc or not
    if (ptr == NULL) {
```

```
        printf("Memory not allocated.\n");
        exit(0);
}
else {

        // Memory has been successfully allocated
        printf("Memory successfully allocated using calloc.\n");

        // Get the elements of the array
        for (i = 0; i < n; ++i) {
                ptr[i] = i + 1;
        }

        // Print the elements of the array
        printf("The elements of the array are: ");
        for (i = 0; i < n; ++i) {
                printf("%d, ", ptr[i]);
        }

        // Get the new size for the array
        n = 10;
        printf("\n\nEnter the new size of the array: %d\n", n);
```

```
        // Dynamically re-allocate memory using realloc()
        ptr = realloc(ptr, n * sizeof(int));

        // Memory has been successfully allocated
        printf("Memory successfully re-allocated using realloc.\n");

        // Get the new elements of the array
        for (i = 5; i < n; ++i) {
            ptr[i] = i + 1;
        }

        // Print the elements of the array
        printf("The elements of the array are: ");
        for (i = 0; i < n; ++i) {
            printf("%d, ", ptr[i]);
        }

        free(ptr);
    }
    getch();
    return 0;
}
```

Output:

```
Enter number of elements: 5
```

```
Memory successfully allocated using calloc.
The elements of the array are: 1, 2, 3, 4, 5,

Enter the new size of the array: 10
Memory successfully re-allocated using realloc.
The elements of the array are: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
```