Content 34

Dynamic Memory Allocation Malloc Calloc Realloc & Free()

For the allocation of memory using the heap, we have four functions:

* Malloc
* Calloc
* Realloc
* Free

#### malloc():

 malloc stands for memory allocation. As can be guessed by its name, it requests memory from the heap and returns a pointer to the memory. The pointer is of the void type, so that we can typecast it for any variables. All the values at the allocation time are initialized to garbage values. Its syntax is simple as we have to provide the memory space along with the size we want in bytes.

#### Syntax:

ptr = (ptr-type\*) malloc(size\_in\_bytes)

**For example:**

int \*ptr;

ptr = (int\*) malloc (3\* sizeof(int))

**Note:** We are using the sizeof() function here because the size of int may differ in different systems, so to be on the safe side.

#### calloc():

calloc stands for contiguous allocation. It also requests memory from the heap and returns a pointer to the memory and has the same functionality as malloc(), two main differences, though. We have to send as parameters the number of blocks needed along with their size. The second difference is a major one. That is, in calloc(), the values at the allocation time are initialized to 0 instead of garbage value.

**Syntax:**

ptr = (ptr-type\*) calloc(n,size\_in\_bytes)

**For example:**

int \*ptr;

ptr = (int\*) malloc (10, sizeof(int))

1. **values are initialized by 0 so this is the extra overhead in calloc().**
2. **So don't use calloc() when you don't want intialized memory with 0.**
3. **and I think this would be extra to make 0 and this will create inefficiency in our Program.**

#### realloc():

realloc stands for reallocation. It is used in cases where the dynamic memory is insufficient or wants to increase the already allocated memory to store more data. Its syntax is simple as we just have to overwrite the memory already allocated as a parameter in the function while providing the data related to the pointer.

**Syntax:**

ptr = (ptr-type\*) realloc(ptr,new\_size\_in\_bytes)

**For example:**

ptr = (int\*) realloc (ptr, 5\* sizeof(int))

#### free():

As we discussed earlier, while discussing the disadvantages of dynamic memory allocation that we have to free up the allocated memory space manually as there is no automatic procedure for that. So free is used to free up the space occupied by the allocated memory. Its syntax is the easiest of all, as we have to send the pointer as a parameter inside the function.

**Syntax:**

free(ptr)

#### Summary:

We use dynamic memory to make the program more efficient as we can define the memory allocation ourselves, thus reducing the program's size, which will have a direct effect on its execution timing. We can allocate dynamic memory fully by using only the four functions given above. Dynamic memory allocation has its advantages and disadvantages, but in order to become a great programmer, we must learn all the concepts completely.

**Code1 For malloc() Function:**

#include <stdio.h>

#include <stdlib.h> /\* This header file is used for Dynamic

Allocation which contains functions like malloc(), calloc(), realloc and free();\*/

// Program for using malloc() function

int main()

{

    int \*ptr;

    ptr = (int \*)malloc(3 \* sizeof(int));

    for (int i = 0; i < 3; i++)

    {

        printf("Enter the value %d: ", i);

        scanf("%d", &ptr[i]); //array is created here

    }

    for (int i = 0; i < 3; i++) **//but here if I make i<4 than it will gie Garbage value**

    {

        printf("\n\nThe value at %d of Array is: %d \n", i, ptr[i]);

    }

    return 0;

}

**Output:**

Enter the value 0: 1

Enter the value 1: 1

Enter the value 2: 1

The value at 0 of Array is: 1

The value at 1 of Array is: 1

The value at 2 of Array is: 1

//garbage may be anything

**Code2 for malloc() but here i am taking n input of element of arrays from user to show**

**memory is allocated at Run Time:**

//here i am taking n input of element of arrays from user to show memory is allocated at Run Time

#include <stdio.h>

#include <stdlib.h> /\* This header file is used for Dynamic

Allocation which contains functions like malloc(), calloc(), realloc and free();\*/

int main()

{

    int \*ptr, n;

    printf("Enter the size of Array you want: ");

    scanf("%d", &n);

ptr = (int \*)malloc(3 \* sizeof(int));

    for (int i = 0; i < n; i++)

    {

        printf("Enter the value %d: ", i);

        scanf("%d", &ptr[i]); //array is created here

    }

    for (int i = 0; i < n; i++)

    {

        printf("\n\nThe value at %d of Array is: %d \n", i, ptr[i]);

    }

    return 0;

}

**Code3 for calloc() function:**

// Here using Calloc() function

 int main()

{

    int \*ptr, n;

    printf("Enter the size of Array you want: ");

    scanf("%d", &n);

ptr = (int \*)calloc(n,sizeof(int)); //here No of blocks are

also send along with size

    for (int i = 0; i < n; i++)

    {

        printf("Enter the value %d: ", i);

        scanf("%d", &ptr[i]); //array is created here

    }

    for (int i = 0; i < n; i++)

    {

        printf("\n\nThe value at %d of Array is: %d \n", i, ptr[i]);

    }

    return 0;

}

**Output:**

Output will be same as malloc() above.

**Code for calloc() function when I the input It will print something like:**

// Here using Calloc() function

 int main()

{

    int \*ptr, n;

    printf("Enter the size of Array you want: ");

    scanf("%d", &n);

ptr = (int \*)calloc(n,sizeof(int)); //here No of blocks are

also send along with size

    // for (int i = 0; i < n; i++)

    // {

    //     printf("Enter the value %d: ", i);

    //     scanf("%d", &ptr[i]); //array is created here

    // }

    for (int i = 0; i < n; i++)

    {

        printf("The value at %d of Array is: %d \n", i, ptr[i]);

    }

    return 0;

}

**Output:**

Enter the size of Array you want: 4

The value at 0 of Array is: 0

The value at 1 of Array is: 0

The value at 2 of Array is: 0

The value at 3 of Array is: 0

1. **values are initialized by 0 so this is the extra overhead in calloc().**
2. **So don't use calloc() when you don't want intialized memory with 0.**
3. **and I think this would be extra to make 0 and this will create inefficiency in our Program.**

**Code for realloc() function:**

#include <stdio.h>

#include <stdlib.h>

/\* This header file is used for Dynamic

Allocation which contains functions like malloc(), calloc(), realloc and free();\*/

// Here using realloc() function

int main(int argc, char const \*argv[])

{

    int \*ptr, n;

    printf("Enter number of elemens you want: ");

    scanf("%d", &n);

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

    for (int i = 0; i < n; i++)

    {

        printf("Enter element %d of the array: ", i);

        scanf("%d", &ptr[i]);

    }

    for (int i = 0; i < n; i++)

    {

        printf("\nThe element %d of the array is: %d\n", i, ptr[i]);

    }

// using realloc()

    printf("\n\nEnter number of elemens you want for New array: ");

    scanf("%d", &n);

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

    for (int i = 0; i < n; i++)

    {

        printf("Enter element %d of the New array: ", i);

        scanf("%d", &ptr[i]);

    }

    for (int i = 0; i < n; i++)

    {

        printf("\nThe element %d of the New array is: %d\n", i, ptr[i]);

    }

    return 0;

}

**Output:**

Enter number of elemens you want: 4

Enter element 0 of the array: 1

Enter element 1 of the array: 2

Enter element 2 of the array: 3

Enter element 3 of the array: 4

The element 0 of the array is: 1

The element 1 of the array is: 2

The element 2 of the array is: 3

The element 3 of the array is: 4

Enter number of elemens you want for New array: 2

Enter element 0 of the New array: 1

Enter element 1 of the New array: 2

The element 0 of the New array is: 1

The element 1 of the New array is: 2

Free() is very easy if I write between any above program that free(ptr) then it would free memory.