C Dynamic Memory Allocation

In this tutorial, you'll learn to dynamically allocate memory in your C program using standard library functions: malloc(), calloc(), free() and realloc().

As you know, an array is a collection of a fixed number of values. Once the size of an array is declared, you cannot change it.

Sometimes the size of the array you declared may be insufficient. To solve this issue, you can allocate memory manually during run-time. This is known as dynamic memory allocation in C programming.

To allocate memory dynamically, library functions are `malloc()`, `calloc()`, `realloc()` and `free()` are used. These functions are defined in the `<stdlib.h>` header file.

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.

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.

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`.

C free()

Dynamically allocated memory created with either `calloc()` or `malloc()` doesn't get freed on their own. You must explicitly use `free()` to release the space.

Syntax of free()

```
free(ptr);
```

This statement frees the space allocated in the memory pointed by `ptr`.

Example 1: malloc() and free()

```
// Program to calculate the sum of n numbers entered by the user

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

int main() {
  int n, i, *ptr, sum = 0;

  printf("Enter number of elements: ");
  scanf("%d", &n);

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

  // if memory cannot be allocated
  if(ptr == NULL) {
    printf("Error! memory not allocated.");
```

```
    exit(0);
  }

  printf("Enter elements: ");
  for(i = 0; i < n; ++i) {
    scanf("%d", ptr + i);
    sum += *(ptr + i);
  }

  printf("Sum = %d", sum);

  // deallocating the memory
  free(ptr);

  return 0;
}
```

Run Code

Output

```
Enter number of elements: 3
Enter elements: 100
20
36
Sum = 156
```

Here, we have dynamically allocated the memory for `n` number of `int`.

Example 2: calloc() and free()

```
// Program to calculate the sum of n numbers entered by the user

#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));
  if(ptr == NULL) {
    printf("Error! memory not allocated.");
    exit(0);
  }

  printf("Enter elements: ");
  for(i = 0; i < n; ++i) {
    scanf("%d", ptr + i);
    sum += *(ptr + i);
  }

  printf("Sum = %d", sum);
  free(ptr);
  return 0;
}
```

Run Code

Output

```
Enter number of elements: 3
Enter elements: 100
20
36
Sum = 156
```

C realloc()

If the dynamically allocated memory is insufficient or more than required, you can change the size of previously allocated memory using the `realloc()` function.

Syntax of realloc()

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

Here, `ptr` is reallocated with a new size `x`.

Example 3: realloc()

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

int main() {
  int *ptr, i , n1, n2;
  printf("Enter size: ");
  scanf("%d", &n1);

  ptr = (int*) malloc(n1 * sizeof(int));

  printf("Addresses of previously allocated memory:\n");
  for(i = 0; i < n1; ++i)
    printf("%pc\n",ptr + i);

  printf("\nEnter the new size: ");
  scanf("%d", &n2);

  // rellocating the memory
  ptr = realloc(ptr, n2 * sizeof(int));

  printf("Addresses of newly allocated memory:\n");
  for(i = 0; i < n2; ++i)
    printf("%pc\n", ptr + i);

  free(ptr);

  return 0;
}
```

Run Code

Output

```
Enter size: 2
Addresses of previously allocated memory:
26855472
26855476

Enter the new size: 4
Addresses of newly allocated memory:
26855472
26855476
26855480
26855484
```