

# Programming with C I

Fangtian Zhong  
CSCI 112

Gianforte School of Computing  
Norm Asbjornson College of Engineering  
E-mail: [fangtian.zhong@montana.edu](mailto:fangtian.zhong@montana.edu)

# Dynamic Array In C

- 🛡️ A dynamic array in C is a versatile and powerful data structure that provides the flexibility to allocate memory at runtime, allowing for the dynamic resizing of the array during program execution.
- 🛡️ Unlike static arrays, which have a fixed size determined at compile time, dynamic arrays can adapt to changing requirements, making them an essential tool in C programming for managing and manipulating data efficiently.

# Dynamic Array In C


➤ However, we can create a dynamic array with the help of the following methods:

- Using malloc() Function
- Using calloc() Function
- Resizing Array Using realloc() Function

# Dynamic Array Using malloc() Function

## Syntax

```
ptr = (cast-type*) malloc(byte-size);
```

 Therefore, we can produce a dynamic array of any type by allocating a single block of memory of a particular size and thus typecasting the returned pointer to the pointer of the returned type.

## Example

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

# Dynamic Array Using malloc() Function

Here, they have used a dynamic array of type **int** and size **100 elements**.

## Example

```
// C program to create dynamic array using malloc() function

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

int main()
{
    // address of the block created hold by this pointer
    int* ptr;
    int size;

    // Size of the array
    printf("Enter size of elements:");
    scanf("%d", &size);
```

# Dynamic Array Using malloc() Function

## Example

```
// Memory allocates dynamically using malloc()
ptr = (int*)malloc(size * sizeof(int));

// Checking for memory allocation
if (ptr == NULL) {
    printf("Memory not allocated.\n");
}
else {

    // Memory allocated
    printf("Memory successfully allocated using "
           "malloc.\n");

    // Get the elements of the array
    for (int j = 0; j < size; ++j) {
        ptr[j] = j + 1;
    }
}
```

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

return 0;
}
```

# Dynamic Array Using malloc() Function

## output

Enter size of elements:5

Memory successfully allocated using malloc.

The elements of the array are: 1, 2, 3, 4, 5,

# Dynamic Array Using calloc() Function



**Dynamic Memory Allocation and Initialization:** **calloc** is used to dynamically allocate a specified number of blocks of memory, each of a specified type. Unlike **malloc**, which does not initialize the memory, **calloc** initializes each block with a default value of 0. This ensures that the allocated memory is zero-initialized from the start.



# Dynamic Array Using calloc() Function

## Syntax

```
ptr = (cast-type*)calloc(n, element-size);
```

## Example

```
ptr = (int*) calloc(5, sizeof(float));
```



**The example provided below illustrates how to create a dynamic array using the calloc() method.**

# Dynamic Array Using calloc() Function

## Example

```
// C program to create dynamic array using calloc()
function

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

int main()
{

    // address of the block created hold by this pointer
    int* ptr;
    int size;

    // Size of the array
    printf("Enter size of elements:");
    scanf("%d", &size);
```

```
// Memory allocates dynamically using calloc()
ptr = (int*)calloc(size, sizeof(int));

// Checking for memory allocation
if (ptr == NULL) {
    printf("Memory not allocated.\n");
}
else {

    // Memory allocated
    printf("Memory successfully allocated using "
           "malloc.\n");

// Get the elements of the array
for (int j = 0; j < size; ++j) {
    ptr[j] = j + 1;
}
```

# Dynamic Array Using calloc() Function

## Example




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

return 0;
}
```

## output

```
Enter size of elements:6
Memory successfully allocated using malloc.
The elements of the array are: 1, 2, 3, 4, 5, 6,
```

# Dynamically Resizing Array Using realloc() Function

-  Used to change the size of previously allocated memory.
-  Adapt to changing memory requirements. The function takes a pointer to the old memory block and the new size in bytes as arguments.
-  It automatically copies the data from the old block to the new one if necessary.

# Dynamically Resizing Array Using realloc() Function

## Syntax

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

## Example

```
/ C program to resize dynamic array using realloc()
// function

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

int main()
{

    // address of the block created hold by this pointer
    int* ptr;
    int size = 5;
```

# Dynamically Resizing Array Using realloc() Function

## Example

```
// Memory allocates dynamically using calloc()
ptr = (int*)calloc(size, sizeof(int));

if (ptr == NULL) {
    printf("Memory not allocated.\n");
    exit(0);
}
else {
    printf("Memory successfully allocated using "
           "calloc.\n");
}

// inserting elements
for (int j = 0; j < size; ++j) {
    ptr[j] = j + 1;
}
```

```
printf("The elements of the array are: ");
for (int k = 0; k < size; ++k) {
    printf("%d, ", ptr[k]);
}

printf("\n");

size = 10;

int *temp = ptr;
```

# Dynamically Resizing Array Using realloc() Function

## Example

```
// using realloc
ptr = realloc(ptr, size * sizeof(int));
if (!ptr) {
    printf("Memory Re-allocation failed.");
    ptr = temp;
}
else {
    printf("Memory successfully re-allocated using "
           "realloc.\n");
}
```

```
// inserting new elements
for (int j = 5; j < size; ++j) {
    ptr[j] = j + 10;
}

printf("The new elements of the array are: ");
for (int k = 0; k < size; ++k) {
    printf("%d, ", ptr[k]);
}
return 0;
}
```

# Dynamically Resizing Array Using realloc() Function

## output

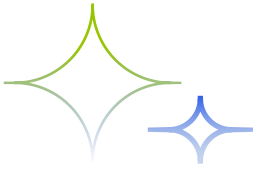
Memory successfully allocated using calloc.

The elements of the array are: 1, 2, 3, 4, 5,

Memory successfully re-allocated using realloc.

The new elements of the array are: 1, 2, 3, 4, 5, 15, 16, 17, 18, 19,





# THE END

Fangtian Zhong  
CSCI 112

Gianforte School of Computing  
Norm Asbjornson College of Engineering  
E-mail: [fangtian.zhong@montana.edu](mailto:fangtian.zhong@montana.edu)