Dynamic Memory Allocation in C

Dynamic memory allocation in C allows programs to request memory during runtime, providing flexibility in managing memory usage. This is especially useful when the amount of memory needed cannot be determined beforehand.

Key Functions for Dynamic Memory Allocation

1. **malloc()**:
   - **Purpose**: Allocates a specified number of bytes and returns a pointer to the first byte of the allocated memory.
   - **Prototype**: `void *malloc(size_t size);`
   - **Usage**:

     ```
     c
     Copy code
     int *ptr = (int *)malloc(10 * sizeof(int));
     if (ptr == NULL) {
         // Handle memory allocation failure
     }
     ```

2. **calloc()**:
   - **Purpose**: Allocates memory for an array of elements, initializes all bytes to zero, and returns a pointer to the first byte of the allocated memory.
   - **Prototype**: `void *calloc(size_t num, size_t size);`
   - **Usage**:

     ```
     c
     Copy code
     int *ptr = (int *)calloc(10, sizeof(int));
     if (ptr == NULL) {
         // Handle memory allocation failure
     }
     ```

3. **realloc()**:
   - **Purpose**: Resizes a previously allocated block of memory, expanding or contracting the amount of memory.
   - **Prototype**: `void *realloc(void *ptr, size_t size);`
   - **Usage**:

     ```
     c
     Copy code
     ptr = (int *)realloc(ptr, 20 * sizeof(int));
     if (ptr == NULL) {
         // Handle memory allocation failure
     }
     ```

4. **free()**:
   - **Purpose**: Deallocates previously allocated memory, making it available for future allocations.
   - **Prototype**: `void free(void *ptr);`
   - **Usage**:

```
c
Copy code
free(ptr);
ptr = NULL; // Avoid dangling pointer
```

Example Program

Here's an example program demonstrating the use of these functions:

```
c
Copy code
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *array;
    int n, i;

    // Allocate memory using malloc
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    array = (int *)malloc(n * sizeof(int));
    if (array == NULL) {
        printf("Memory allocation failed\n");
        return 1;
    }

    // Initialize and print array
    printf("Enter %d integers: ", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &array[i]);
    }

    printf("You entered: ");
    for (i = 0; i < n; i++) {
        printf("%d ", array[i]);
    }
    printf("\n");

    // Resize the memory block using realloc
    printf("Enter the new number of elements: ");
    scanf("%d", &n);

    array = (int *)realloc(array, n * sizeof(int));
    if (array == NULL) {
        printf("Memory reallocation failed\n");
        return 1;
    }

    // Initialize and print resized array
    printf("Enter %d integers: ", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &array[i]);
    }

    printf("You entered: ");
    for (i = 0; i < n; i++) {
        printf("%d ", array[i]);
    }
```

```
    printf("\n");

    // Deallocate memory
    free(array);

    return 0;
}
```

Summary

- **Dynamic Memory Allocation**: Provides flexibility to allocate memory as needed during program execution.
- **Functions**:
  - `malloc()`: Allocates a specified amount of memory.
  - `calloc()`: Allocates memory for an array and initializes all bytes to zero.
  - `realloc()`: Resizes an allocated memory block.
  - `free()`: Deallocates allocated memory.
- **Error Handling**: Always check if the memory allocation functions return `NULL` to handle allocation failures gracefully.
- **Memory Management**: It is crucial to `free` dynamically allocated memory to avoid memory leaks.