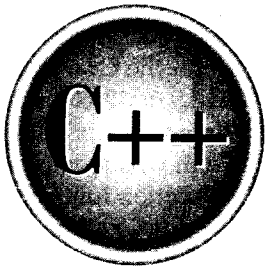
The Complete Reference



Chapter 29

The Dynamic Allocation Functions

This chapter describes the dynamic allocation functions, which were inherited from the C language. At their core are the functions **malloc( )** and **free( )**. Each time **malloc( )** is called, a portion of the remaining free memory is allocated. Each time **free( )** is called, memory is returned to the system. The region of free memory from which memory is allocated is called the *heap*. The prototypes for the dynamic allocation functions are in **<cstdlib>**. A C program must use the header file **stdlib.h**.

All C++ compilers will include at least these four dynamic allocation functions: **calloc( )**, **malloc( )**, **free( )**, **realloc( )**. However, your compiler will almost certainly contain several variants on these functions to accommodate various options and environmental differences. You will want to refer to your compiler's documentation.

While C++ supports the dynamic allocation functions described here, you will typically not use them in a C++ program. The reason for this is that C++ provides the dynamic allocation operators **new** and **delete**. There are several advantages to using the dynamic allocation operators. First, **new** automatically allocates the correct amount of memory for the type of data being allocated. Second, it returns the correct type of pointer to that memory. Third, both **new** and **delete** can be overloaded. Since **new** and **delete** have advantages over the C-based dynamic allocation functions, their use is recommended for C++ programs.

calloc

```
#include <cstdlib>
void *calloc(size_t num, size_t size);
```

The **calloc( )** function allocates memory the size of which is equal to *num* * *size*. That is, **calloc( )** allocates sufficient memory for an array of *num* objects of size *size*.

The **calloc( )** function returns a pointer to the first byte of the allocated region. If there is not enough memory to satisfy the request, a null pointer is returned. It is always important to verify that the return value is not null before attempting to use it.

Related functions are **free( )**, **malloc( )**, and **realloc( )**.

free

```
#include <cstdlib>
void free(void *ptr);
```

The **free( )** function returns the memory pointed to by *ptr* to the heap. This makes the memory available for future allocation.

It is imperative that **free( )** only be called with a pointer that was previously allocated using one of the dynamic allocation system's functions (either **malloc( )** or **calloc( )**).

Using an invalid pointer in the call most likely will destroy the memory management mechanism and cause a system crash.

Related functions are **calloc( )**, **malloc( )**, and **realloc( )**.

malloc

```
#include <cstdlib>
void *malloc(size_t size);
```

The **malloc( )** function returns a pointer to the first byte of a region of memory of size *size* that has been allocated from the heap. If there is insufficient memory in the heap to satisfy the request, **malloc( )** returns a null pointer. It is always important to verify that the return value is not null before attempting to use it. Attempting to use a null pointer will usually result in a system crash.

Related functions are **free( )**, **realloc( )**, and **calloc( )**.

realloc

```
#include <cstdlib>
void *realloc(void *ptr, size_t size);
```

The **realloc( )** function changes the size of the previously allocated memory pointed to by *ptr* to that specified by *size*. The value of *size* may be greater or less than the original. A pointer to the memory block is returned because it may be necessary for **realloc( )** to move the block in order to increase its size. If this occurs, the contents of the old block are copied into the new block—no information is lost.

If *ptr* is null, **realloc( )** simply allocates *size* bytes of memory and returns a pointer to it. If *size* is zero, the memory pointed to by *ptr* is freed.

If there is not enough free memory in the heap to allocate *size* bytes, a null pointer is returned, and the original block is left unchanged.

Related functions are **free( )**, **malloc( )**, and **calloc( )**.