Dynamic memory allocation

It is also possible to initialize pointers using free memory. This allows dynamic allocation of memory. It is useful for setting up structures such as linked lists or data trees where you don't know exactly how much memory will be needed at compile time, so you have to get memory during the program's execution. We'll look at these structures later, but for now, we'll simply examine how to request memory from and return memory to the operating system.

The function malloc( ), residing in the stdlib.h header file, is used to initialize pointers with memory from free store, named heep (a section of memory available to all programs). Function malloc( ) works just like any other function call. The argument to malloc( ) is the amount of memory requested (in bytes), and malloc( ) gets a block of memory of that size and then returns a pointer to the block of memory allocated.

Since different variable types have different memory requirements, we need to get a size for the amount of memory malloc( ) should return. So we need to know how to get the size of different variable types. This can be done using the operator sizeof, which takes an expression and returns its size. For example, sizeof(int) would return the number of bytes required to store an integer.

```
#include <stdlib.h>

int *p = (int*)malloc(sizeof(int));
```

This code set p to point to a memory address of size int. The memory that is pointed to becomes unavailable to other programs. This means that the careful coder should free this memory at the end of its usage lest the memory be lost to the operating system for the duration of the program (this is often called a memory leak because the program is not keeping track of all of its memory).

Note that it is slightly cleaner to write malloc( ) function by taking the size of the variable pointed to by using the pointer directly:

```
int *p =(int*)malloc(sizeof(*p));
```

What's going on here? sizeof(*p) will evaluate the size of whatever we would get back from dereferencing p; since p is a pointer to an int, *p would give us an int, so sizeof(*p) will return the size of an integer. So why do this? Well, if we change p to point to something else like a float, then we don't have to go back and correct the malloc( ) call to use sizeof(float). Since p would be pointing to a float, *p would be a float, so sizeof(*p) would still give the right size!

The free function returns memory to the operating system.

```
free(p);
```

After freeing a pointer, it is a good idea to reset it to point to NULL. When NULL is assigned to a pointer, the pointer becomes a NULL pointer, in other words, it points to nothing. By doing this, when you do something foolish with the pointer (it happens a lot, even with experienced programmers), you find out immediately instead of later, when you have done considerable damage.

The concept of the NULL pointer is frequently used as a way of indicating a problem--for instance, malloc( ) returns NULL when it cannot correctly allocate memory. You want to be sure to handle this correctly--sometimes your operating system might actually run out of memory and give you this value!