Dynamic Memory Tips

Writing programs that use dynamic memory can lead to difficult debugging sessions. By following the suggestions below, you can save yourself a great deal of time. The tips assume that you are writing code that involves linked lists, so that you've set up a struct to support the linked coding.

1) Before changing the value of a pointer variable (by assignment or via ***new*** operator):

   Be sure that the pointer variable to be used is not already pointing to some previously allocated memory location. If it is, and no other pointer variables are aimed at the memory location, you should use the ***delete*** operator to avoid turning the memory location into garbage (inaccessible memory which is a memory leak).

2) After using *new*:

   Remember, the fields in a struct created by *new* need to be initialized. Uninitialized pointers are very dangerous!

3) Before using *delete*:

   Check to make sure that no other pointers are aimed at the node to be deleted. If there are, you may want to set them to NULL before invoking *delete*. If you don't, these pointers will be "dangling".

4) When writing functions:

   Be careful to delete any unwanted dynamic storage before exiting the function. If you don't, it will become garbage and create a memory leak.

   Keep in mind that if a pointer is passed by value, the function can't permanently change the value of the pointer, but it can permanently change the data pointed at by the pointer.

If your code does not work, **don't start hacking!** It is well known that linked list code cannot be hacked into submission! If the fix isn't obvious, get a printout, retreat to a quiet place and trace the code!

The debugger has good support for writing and tracing linked list code. Keep in mind, however, that the debugger will not show you the entire linked list. Typically, you would want to watch such values as "`List->Data`" and or "`List->Link->Data`".