Semi-automatic malloc-free library – Programmers' documentation

The Semi-automatic malloc-free library (SA lib) is intended to provide a simpler way of memory management to developers. The library defines new versions of the functions malloc, calloc, realloc and free with the "sa_" prefix. In addition to these, it also defines a function called "sa_freeall(void)", which frees every pointer allocated by the library so far.

How it works:

Basically, the SA lib collects the pointers allocated by its functions in a linked list, that's how it knows which pointers need to be freed. The "sa_freeall" function makes the library useful, since you don't have to free the dynamically allocated memory space in a function at each return point, you only have to call "sa_freeall" at a point, where you certainly don't need the memory space allocated by the SA lib functions anymore.

However**, SA lib has no effect on the pointers given by the standard memory allocating functions.** In many cases, it's much better to use the standard functions, since you might not want some of the allocated memory space to be affected by mass-freeing. In fact, for any memory space that needs to be freed individually, you should use the standard library functions, since "sa_free" function needs to find the given pointer in the linked list too, and remove it from there as well. Doing it too much will result in a significant loss of efficiency.

The "sa_add function"

There are many cases, when you use a function which allocates memory space using the standard functions, but you do not want to call "free" for that pointer individually. That's when you can use the "sa_add(void *ptr)" function, which allows you to add a pointer to the linked list built by SA lib upon getting it from the called function, and you don't need to call free for it at each return point.

Warnings:

1. When memory space is allocated by SA lib at the first time, it puts a hook, which guarantees, that the "sa_freeall" function will be called upon exit. **However, it does not prevent long-term memory leaks** when you build a more complex program, it might run out of memory while running. **Do not forget to call "sa_freeall" periodically, especially in programs, that could run infinitely.**
2. **Be aware of your pointers**, don't call SA lib functions for standard dynamic pointers and vice versa. **However, you can turn a standard dynamic pointer into an SA lib pointer with "sa_add".**
3. **Avoid using "sa_realloc" (and realloc in general) too much**, it could make your program less efficient. Build linked lists instead, or use buffering. For the latter, it might be better to use standard malloc and free functions, until the array reaches its final size, then it could be turned into a SA pointer by "sa_add".
4. **Do not call "sa_free" too much**, it certainly makes your program less efficient, since SA lib has to find the given pointer in its linked list, and delete it from there as well.
5. **Like I mentioned, sometimes it's better to use the standard functions for memory allocations.**
6. **It's much easier to familiarize yourself with the library, if you look at the code.** It has quite a simple concept.

Happy coding!