

Solutions for Memory-Tools

Memory-Tools

Uhoh

1.

Try something like this.

```
char *tmp ;
tmp = (char*) malloc(strlen(fname)*sizeof(char)+1);
if (NULL==NULL) {
    return store;
}
u->family_name = tmp;
```

... and the same for the next string.

2.

Example:

```
./gen-person.sh 100 | valgrind --leak-check=full ./uhoh
```

3.

We could instruct the user (the developer using our API) to check the size variable in the `user_store` struct.

4.

Since we're running out of memory we're probably facing a big problem. We've made sure we're not losing data and that we're not crashing if we're failing to allocate more. That's good enough.

It is up to the the developer of the program to take care of closing down the program.

5.

This is one solution - a bit ugly since we're using a global variable.

```

#include <stdio.h>
#include <string.h>

#include "user.h"

int close_down=0;

user_store* read_from_stdin(user_store *us){
    int last_size;

#define BUF_SIZE 100
    char buf[BUF_SIZE];
    char fname[BUF_SIZE];
    char gname[BUF_SIZE];
    while(1) {
        char *tmp = fgets(buf,BUF_SIZE,stdin);
        /*      printf ("buf: '%s' =>  (%lu) '%c'\n",
                    buf, strlen(buf),
                    buf[strlen(buf)-1]); */
        if (tmp == NULL) { break ; }
        if (buf[strlen(buf)-1]=='\n') {
            sscanf(buf, "%s , %s", fname, gname);
            /*      fprintf(stdout, "Adding: '%s' '%s'\n",
                    fname, gname);*/
            last_size=us->size;
            us = add_user(us, fname, gname);
            if ( us->size == last_size) {
                fprintf(stderr, "Uh oh...out of memory\n");
                close_down=1;
                return us;
            }
        } else {
            fprintf(stderr, "Too long name\nSeems spooky... bailing out\n");
            return us;
        }
    }
    return us;
}

int main() {

    user_store *us = new_user_store();

    /* us = add_user(us, "Diego", "Maradona"); */
    /* us = add_user(us, "Mario", "Kempes"); */

```

```

    read_from_stdin(us);

    if (!close_down) {
        print_user_store(us);
    }

    free_user_store(us);
}

```

bad memories

10.

See source code.

11.

See source code.

12.

```
gcc bad-memories1.c -o bad-memories
```

13.

```
gcc -pedantic -Wall -Werror bad-memories.c -o bad-memories
```

14.

That it leaks 4000 bytes (1000 ints of size 4 bytes!).

15.

That it leaks 4000 bytes (1000 ints of size 4 bytes!).

But it can now say where (what line) we leak memory.

16.

Add a call to free.

See source code.

17.

See source code.

18.

That you're writing on invalid memory. If you've compiled with `-g` Valgrind will tell you on what line the error can be found.