

C, Memory

CS 2130: Computer Systems and Organization 1

Xinyao Yi Ph.D.

Assistant Professor





Announcements

- Homework 8 due tonight on Gradescope
- Homework 9 available this afternoon
- Lab 10 tomorrow: Memory Errors
- Lab 11 next Tuesday (can check off for full credit by 12/5)



Common Memory Bugs (reading)

you should read it in detail by lab and will see a lot of these there.



List Example

makelist, destroylist and append.

You should have seen all in the homework.

O. header file. (list.h).

#ifndef __LIST_H__ #define __LIST_H__

typedef struct {
 unsigned length;
 int *array;
} List:

void append(List *list, int item);

List *makeList(); void destroyList(List *);

#endif

it's okay for compiler.

For better reading: (List *list)

3. typedef struct {

This begins a structure definition. typedef means you will give this struct a type name (List) later.

4. unsigned length;

This declares a field named length.

Type: unsigned (same as unsigned int).

It stores the number of elements currently in the list.

Because it is unsigned, it can never be

negative.
5. int *list:

This declares a field named list. Type: int * — a pointer to an integer. This pointer will point to a dynamically

allocated array of integers.
This is where the actual list elements are stored

1. #ifndef LIST H

#ifndef means "if not defined "

This checks whether the macro __LIST_H_ has not been defined yet.

It is the start of an include guard, which prevents this header from being included multiple times.

2. #define __LIST_H__

This defines the macro __LIST H .

Now the compiler knows that this header has been included once.

Everything between this line and #endif will only be included one time, even if you #include it repeatedly.

void append(List *list, int item);

int item — the integer to append.

This is a function declaration.

Parameters: List *list — a pointer to the list you want to modify.

Purpose: Append item to the end of the list.

7. List *makeList();

This is another function declaration.

Return type: List * — a pointer to a newly created list.

Parameter: a List * (the name is omitted, but the type is given).

Purpose: Allocate a new List and initialize it.

8. void destroyList(List *);

Function declaration.

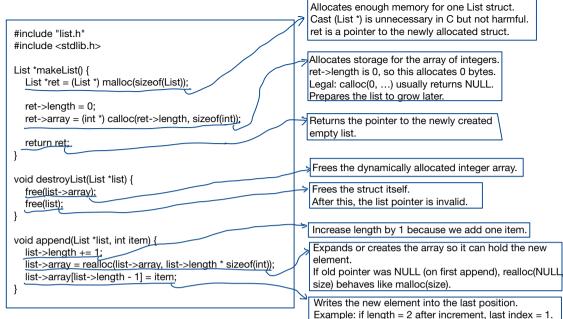
Returns nothing (void).

Purpose: Free all memory used by a List.

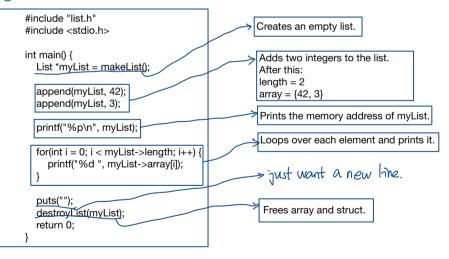
9. #endif

This matches the #ifndef at the top.
It ends the include guard.
Ensures the header file is only included once during compilation.





(3) uselist.c



■ University fVirginia

- 1. man man: ask for the manual on the manual
- 2) synopsis: very short overview.
- 3. something underlined, it's linking it back to something at the top.
- (4). It's divided in sections and there are 9 sections of the manual.

More on man pages

- (3. Mostly going to be in section 3. (Library calls).
- (B). Examples of how to use the man.
- O. "/": slash for search. "/example". it will find example on the page. "n" will be the next, and "shift" is the previous one.
- (0. ty "man printf", "man 3 printf" (check the printf in section 3), then seach for Page 5

examples, you can find some examples for printf.

19. If you're not quite sure what you're looking for, use "-k" to do larger searches.

MIVERSITY VIRGINIA

- (1), man -k "square not" (find the instruction using keyword) very fast. it tells you some information of square not like sqrt function.
- (1), man sqrt: This is the function of square root.
- (D. man -f is equivalent to "whatis" (I know what I want, for example, I want to check printf of Section 3. (man 3 printf)
 - (3). man -k: search all things in man, find where this word appears, (very slow).

All the processes running on the machine: ps - A less
How many things are running? ps - A | wc - L (All of them have a view of)

MIVERSITY of VIRGINIA

Processes

-> memory as if they were the only program

hunning in memory.

Process - approximately what we think of as a "running program"

- Operating System effectively has a giant array of processes started since computer turned on
- Try ps -A
- Has access to all memory (but only its own!)
- Operating System maintains data structure about each process
 - · What program is running, who ran it, when it started, ...
 - · Array of "file like objects"