The Adventures of Malloc and New

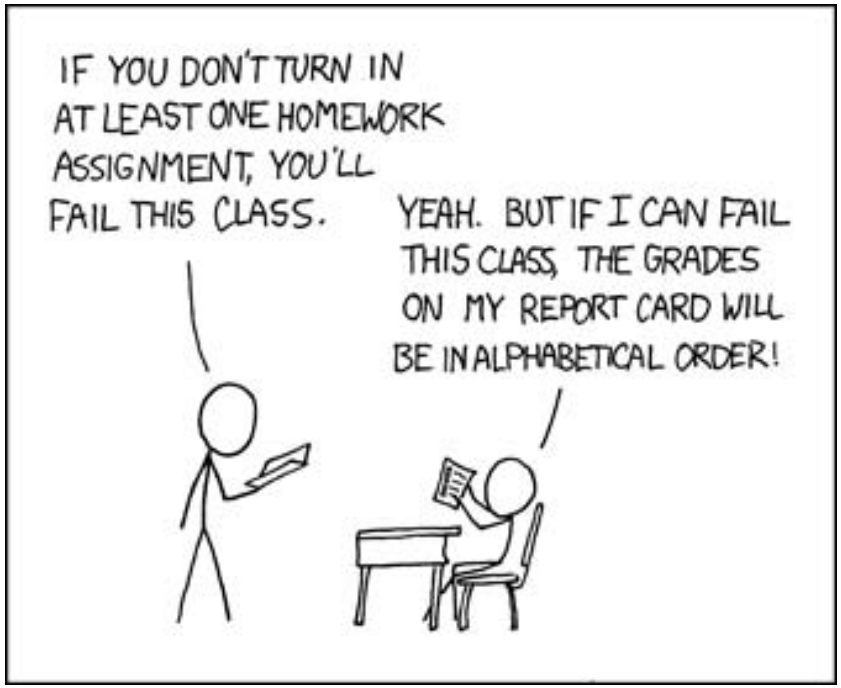
Lecture 3: Oh, Say Can You C

Eunsuk Kang and **Jean Yang**

MIT CSAIL

January 21, 2010

Homework notes



Courtesy of xkcd.com. Comic is available here: http://xkcd.com/336/

Programs must:

- Compile and run (with instructions) to receive a ✓.
- Compile and run, passing tests, to receive a ✓+.

Lecture plan

The more I C, the less I see. –Unknown.

1. Review of main concepts from previous lectures.
2. Fancier memory examples.
3. Closer look at GCC.
4. Style and tips.
5. Why C?

Review: stack and heap

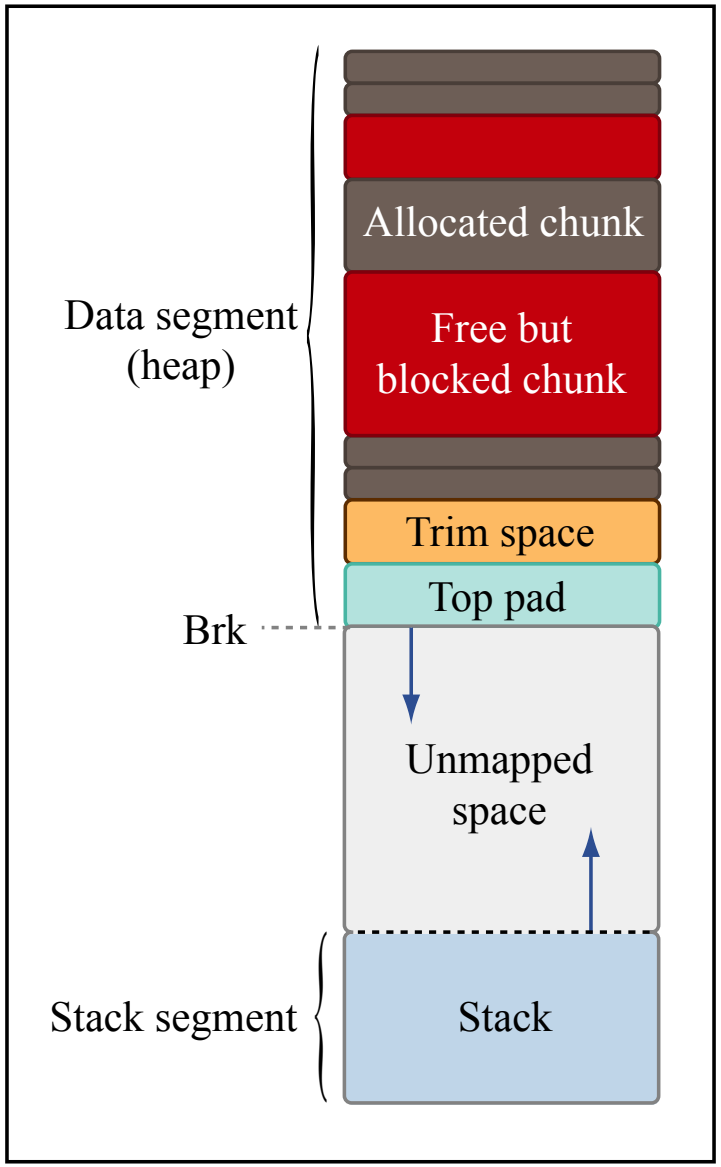


Figure by MIT OpenCourseWare.

Review: when to use pointers

- When you have to allocate memory on heap. (When is this?)
- When passing a parameter whose value you want to allow the other function to change.
- Also for efficiency–to avoid copying data structures.

Buggy field access

```
int* i = NULL;
*i = 3;
```

Segmentation fault!

```
struct pair {
  int first;
  int second;
};

struct pair* pp = NULL;
pp->first = 1;
```

Segmentation fault!

Buggy free

```
struct pair* pp = malloc(sizeof(struct pair));
pp = NULL;
free(pp);
```

Memory leak!

```
int* i = NULL;
free(i);
```

Nothing bad happens! Freeing `NULL` does nothing.

Buggy scope

Buggy example

```
int* get_array(int* len) {

  *len = 3;
  return vals;
}

int main() {
  int len, i;
  int* arr = get_array(&len);
  for (i = 0; i < len; +i) {
    printf("%d\n", arr[i]);
  }
  return 0;
}
```

Returns address of local (statically allocated) variable. (Should get warning!)

Buggy scope continued...

Buggy output

```
1
32607
-1527611392
```

Correct program

```
int* get_array(int* len) {
  *len = 3;
  int* vals = malloc(sizeof(int) * 3);
  arr[0] = 1; arr[1] = 2; arr[2] = 3;
  return arr;
}
```

Buggy initialization

```
struct pair* pp;
int i = pp->first;
```

Maybe a segmentation fault...

In-place linked list reversal

```
Element *reverse(Element *old_list)
{
    Element *new_list = NULL;

    while (old_list != NULL) {
        // Remove element from old list.
        Element *element = old_list;
        old_list = old_list->next;

        // Insert element in new list.
        element->next = new_list;
        new_list = element;
    }

    return new_list;
}
```

Courtesy of Lawrence Kesteloot. Used with permission.
Please see http://www.teamten.com/lawrence/writings/reverse_a_linked_list.html.

Constant time insert into a circular singly-linked list

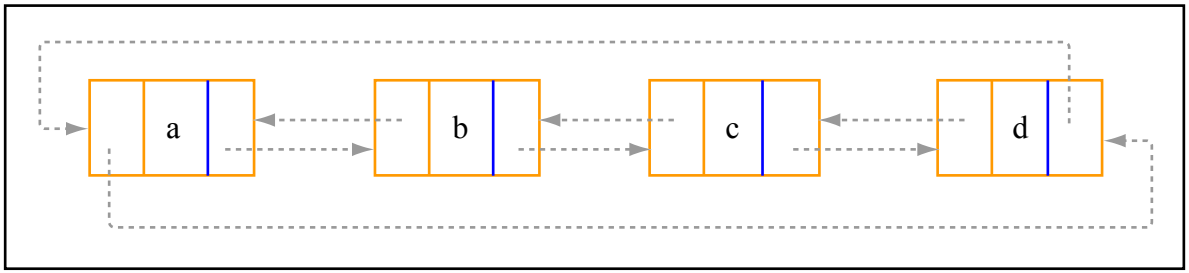


Figure by MIT OpenCourseWare.

- Circular linked list: last node has a pointer to the first node.
- Given a pointer to a node–can't change that pointer!
- Want to insert a node *before* the current one–can we do that in constant time?

A closer look at the GCC compilation process

Preprocessor

Translation of # directives.

- Translates all macros (`#DEFINE`'s) into inline C code.
- Takes `#include` files and inserts them into the code.
  - Get redefinition error if structs etc. are defined more than once!
  - Use `#ifndef` directive to define things only if they have not been defined.

```
#ifndef __HEADER_NAME
#define __HEADER_NAME
/* Header code here. */
#endif
```

Aside: #define

Compiler directive.

```
#define DEFINED_CONSTANT 3
```

```
#define increment(x) (x+1)
```

GCC continued

Parsing and translation

Translates to assembly, performing optimizations.

Assembler

Translates assembly to machine instructions.

Linking

- **Static.** For each function called by the program, the assembly to that function is included directly in the executable, allowing function calls to directly address code.
- **Dynamic.** Function calls call a Procedure Linkage Table, which contains the proper addresses of the mapped memory.

Some helpful compiler options

Enforcements and warnings	
`-ansi`	Tells compiler to enforce ANSI C standards.
`-pedantic`	More pedantic ANSI with warnings.
`-Wall`	Shows all warnings.
`-W`	Show some warnings (return without a value, etc.).
Compilation/output	
`-c`	Compile and assemble, but do not link.
`-S`	Stop after compilation; do not assemble.
`-E`	Stop after preprocessing; do not compile.
`-o` [file]	Put the output binary in [file].
Profiling	
`-pg`	Compile with profiling information.

```
#define _ F--->00 || F-OO--;
long F=00,OO=00;
main(){F_OO();printf("%1.3f\n", 4.*-F/OO/OO);}F_OO()
{
                  _-_-_-_
            _-_-_-_-_-_-_-_-_
         _-_-_-_-_-_-_-_-_-_-_-_
      _-_-_-_-_-_-_-_-_-_-_-_-_-_-_
    _-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
   _-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
  _-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
  _-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
   _-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
    _-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
      _-_-_-_-_-_-_-_-_-_-_-_-_-_-_
         _-_-_-_-_-_-_-_-_-_-_-_-_
            _-_-_-_-_-_-_-_-_-_
                  _-_-_-_
}
```

gcc pi.c -E

```
long F=00,OO=00;
main(){F_OO();printf("%1.3f\n", 4.*-F/OO/OO);}F_OO()
{
                F-->00 || F-OO--;-F-->00 || F-OO--;-F-->00
                    || F-OO--;-F-->00 || F-OO--;
        F-->00 || F-OO--;-F-->00 || F-OO--;-F-->00 || F-
            OO--;-F-->00 || F-OO--;-F-->00 || F-OO--;-F
            -->00 || F-OO--;-F-->00 || F-OO--;-F-->00 ||
             F-OO--;-F-->00 || F-OO--;
    F-->00 || F-OO--;-F-->00 || F-OO--;-F-->00 || F-OO
        --;-F-->00 || F-OO--;-F-->00 || F-OO--;-F-->00
         || F-OO--;-F-->00 || F-OO--;-F-->00 || F-OO--;-
        F-->00 || F-OO--;-F-->00 || F-OO--;-F-->00 || F
        -OO--;-F-->00 || F-OO--;
  F-->00 || F-OO--;-F-->00 || F-OO--;-F-->00 || F-OO
      --;-F-->00 || F-OO--;-F-->00 || F-OO--;-F-->00 ||
       F-OO--;-F-->00 || F-OO--;-F-->00 || F-OO--;-F
      -->00 || F-OO--;-F-->00 || F-OO--;-F-->00 || F-OO
      --;-F-->00 || F-OO--;-F-->00 || F-OO--;-F-->00 ||
       F-OO--;
```

Build systems: a quick flavor of `make`

- Way to manage compilation for large systems.
- Automatically determines which parts of large programs need to be recompiled.
- Simple makefile consists of rules of the following form:

```
target ... : prerequisites ...
        command
        ...
        ...
```

- Build system by running `make` from command line.

A simple C makefile

Makefile

This uses the implicit rules GNU Make defines for compiling C.

```
CC=gcc
CFLAGS=-Wall
main: main.o hello_fn.o
clean:
  rm -f main main.o hello_fn.o
```

Compiling with make

```
$ make
gcc -Wall   -c -o main.o main.c
gcc -Wall   -c -o hello_fn.o hello_fn.c
gcc   main.o hello_fn.o   -o main
$ ./main
"Hello world!"
```

Profiling: gprof

1. Compile with the profiling option.
   `gcc single_linked_list.c test_sll.c -o sll -pg`
2. Run the program–this will produce a file gmon.out (unless otherwise specifiedi) containing profiling information.
   `./sll`
3. Run `gprof` on the binary to get the profiling information. (Can suppress/select specific functions.)
   `gprof sll > profile_output.txt`

Sample profiling output

Flat profile

```
%   cumulative  self              self     total
time   seconds  seconds calls  Ts/call  Ts/call  name
0.00      0.00     0.00     6     0.00     0.00  print_sll
0.00      0.00     0.00     5     0.00     0.00  make_node
0.00      0.00     0.00     4     0.00     0.00  insert_val
0.00      0.00     0.00     2     0.00     0.00  delete_val
```

Call tree

Also shows more detailed call tree information, sorted by total amount of time spent in each function and its children.

War stories

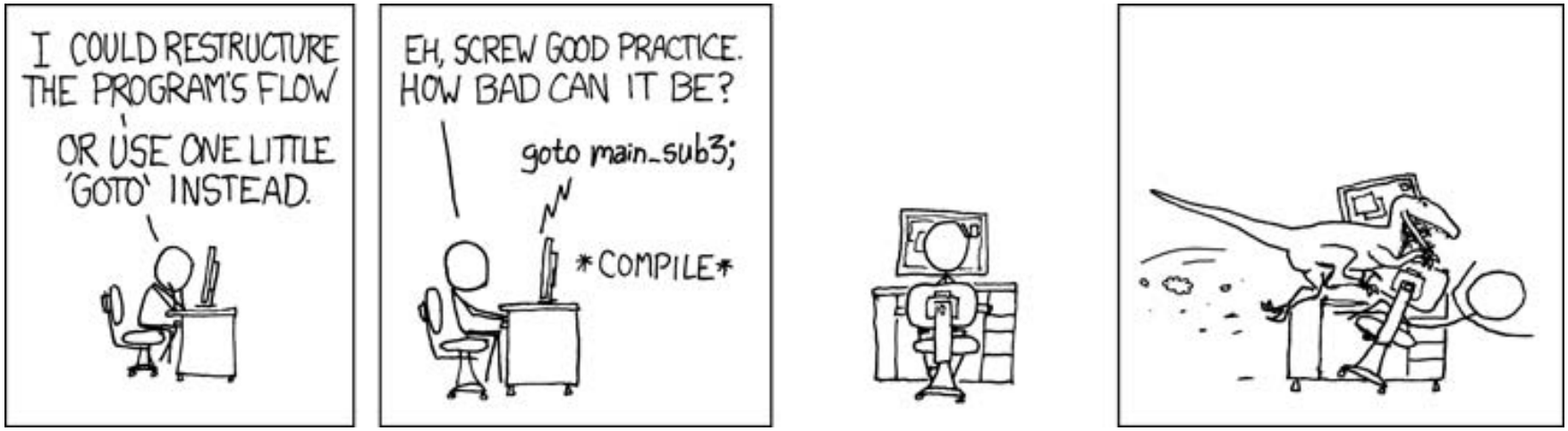


Figure: With great power comes great responsibility.

Courtesy of xkcd.com. Comic is available here: http://xkcd.com/229/

- Linker woes–why you want namespaces.
- You can link anything!

Write principled code in C

Cartoon of lemming falling off a cliff removed due to copyright restrictions.

I will not be a lemming and follow the crowd over the cliff and into the C. –John (Jack) Beidler

Array indexing

Recall that and array of size n with objects of type t is just a block of memory of size `sizeof`$(t) * n$.

Element	1	2	3	n
Array (arr [])	arr[0]	arr[1]	arr[2]	arr[$n-1$]
Pointer (arr*)	*arr	*(arr+1)	*(arr+2)	*(arr + $n+1$)

Style guidelines

- Test for equality with the constant on the left-hand side.

```
if (3 == x) /* rather than (x == 3) {
    ...
}
```

- Initialize pointers to `NULL`.

```
int* p = NULL;
int* q;    /* Unitialized, q will point to junk. */
```

- Use pre-increment unless post-increment is necessary.

```
++i;       /* Pre-increment. */
i++;       /* Post-increment. */
```

Where is the overhead?

In memory-managed languages

- **Garbage collection**–figuring out what can be garbage collected and reclaiming that memory.
- Overhead from GC's *conservative* approximations of what is still in use.
- Reducing object allocation mitigates this problem.

In C

- Each **allocation** takes time because the allocator has to search for a piece of sufficiently large memory each time.
- Reducing the number of allocations mitigates this problem.

C is not just for thrill-seekers

Speed

- Slowdowns from compiling from higher-level languages.
- Personal anecdote: randomized simulations (C vs. Python).

Memory

- Memory overhead (20%+) necessary for garbage collection.
- Personal anecdote: 400x speedup from reducing object creation in OCaml program.

Access to dangerous, low-level parts of system

- Accessing hardware devices as memory.
- Mucking with the registers (stack pointer).

Until tomorrow...

Homework (due Saturday)

- Rewrite the binary search tree assignment using an array.
- Please submit a `.zip` or `.tar.gz` file prefaced by [username].hw[number].
- More details on the course website.

Questions?

- The course staff will be available after class.

6.088 Introduction to C Memory Management and C++ Object-Oriented Programming

January IAP 2010