The Adventures of Malloc and New
Lecture 1: The Abstract Memory Machine

Eunsuk Kang and Jean Yang

MIT CSAIL

January 19, 2010
C: outdated, old, antiquated...


**Figure:** Dennis Ritche and Ken Thompson in 1972.
C: fast, faster, fastest

Figure: Benchmark times from the Debian language shootout.
Congratulations on choosing to spend your time wisely!

Figure: XKCD knows that tools are important.

Courtesy of [xkcd.com](http://xkcd.com). Original comic is available here: [http://xkcd.com/519/]
Lecture plan

1. Course goals and prerequisites.
2. Administrative details (syllabus, homework, grading).
3. High-level introduction to C.
4. C philosophy: “the abstract memory machine.”
5. How to get started with C.
6. Wrap-up and homework.
Course goal: to help proficient programmers understand how and when to use C and C++.
Background check

Expected knowledge

- Basic data structures (linked lists, binary search trees, etc.)?
- Familiarity with basic imperative programming concepts.
  - Variables (scoping, global/local).
  - Loops.
  - Functions and function abstraction.

Other knowledge

- Functional programming?
- Systems programming?
- Hardware?
- OOP with another language?
## Course syllabus

<table>
<thead>
<tr>
<th>Day</th>
<th>Date</th>
<th>Topic</th>
<th>Lecturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/19</td>
<td>Meet C and memory management</td>
<td>Jean</td>
</tr>
<tr>
<td>2</td>
<td>1/20</td>
<td>Memory management logistics</td>
<td>Jean</td>
</tr>
<tr>
<td>3</td>
<td>1/21</td>
<td>More advanced memory management</td>
<td>Jean</td>
</tr>
<tr>
<td>4</td>
<td>1/22</td>
<td>Meet C++ and OOP</td>
<td>Eunsuk</td>
</tr>
<tr>
<td>5</td>
<td>1/23</td>
<td>More advanced OOP</td>
<td>Eunsuk</td>
</tr>
<tr>
<td>6</td>
<td>1/24</td>
<td>Tricks of the trade, Q &amp; A</td>
<td>Eunsuk</td>
</tr>
</tbody>
</table>
Administrivia

Homework

- Daily homework to be submitted via the Stellar site.
- Graded $\checkmark +$, $\checkmark$, or $\checkmark -$.
- Homework $i$ will be due 11:59 PM the day after Lecture $i$; late submissions up to one day (with deductions).
- Solutions will be released one day following the due date.

Requirements for passing

- Attend lectures–sign in at back.
- Complete all 5 homework assignments with a $\checkmark$ average.
Recommended references

Books


Online resources

http://www.cprogramming.com
The C family

C

- Developed in 1972 by Dennis Ritchie at Bell Labs.
- Imperative systems language.

C++

- Developed in 1979 by Bjarne Stroustrup at Bell Labs.
- Imperative, object-oriented language with generics.

C# (outside scope of course)

- Multi-paradigm language with support for imperative, function, generic, and OO programming and memory management.
- Developed at Microsoft, release circa 2001.
Vocabulary check

- Imperative, declarative, functional
- Compiled, interpreted
- Static, dynamic
- Memory-managed
Typically, C is... 

- Compiled.
- Imperative.
- Manually memory-managed.
- Used when at least one of the following matters:
  - Speed.
  - Memory.
  - Low-level features (moving the stack pointer, etc.).
Thinking about C in terms of memory...
Layers of abstraction over memory

<table>
<thead>
<tr>
<th>Level of abstraction</th>
<th>Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directly manipulate memory</td>
<td>Assembly (x86, MIPS)</td>
</tr>
<tr>
<td>Access to memory</td>
<td>C, C++</td>
</tr>
<tr>
<td>Memory managed</td>
<td>Java, C#, Scheme/Lisp, ML</td>
</tr>
</tbody>
</table>
It’s a memory world

**Figure:** Processors read from memory, do things, and write to memory.

Eunsuk Kang and Jean Yang  
[The Adventures of Malloc and New](#)
The *heap* is a chunk of memory for the C program to use.

- Can think of it as a giant array.
- Access heap using special *pointer* syntax.
- The whole program has access to the heap\(^a\).

\(^a\)Depending on what the operating system allows

<table>
<thead>
<tr>
<th>Addr.</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>0xbee</td>
<td>0xbeef</td>
</tr>
<tr>
<td>0xbf4</td>
<td>0xfeed</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>
Manual memory management

Goals

• Want to allow the program to be able to designate chunks of memory as currently in use.
• Want to be able to re-designate a piece of memory as “freed” when the program is done with it.

C support
Standard library (stlib.h) has malloc and free functions.
The other C memory: the stack

C functions get allocated on the *stack*.

- Functions are “pushed on” to the stack when called.
- Functions are “popped off” the stack when they return.
- Functions can access any memory below the current top of the stack.
Memory layout: process context

Figure by MIT OpenCourseWare.

Eunsuk Kang and Jean Yang

The Adventures of Malloc and New
Getting started with C

Photograph removed due to copyright restrictions.

**Figure:** IBM 29 card punch, introduced late 1964.
Using C

1. Obtain a C compiler (GCC recommended—more instructions on site for downloading GCC or using it on MIT servers.)
2. Write a simple C program.

```c
#include <stdio.h>    /* Headers to include. */

int main() {
}
```

3. Compile: gcc -o run_hello hello.c
4. Run: ./run_hello
Functions

```c
void print_sum(int arg1, int arg2) {
    int sum = arg1 + arg2;

    /* Printf is a special function taking variable number of arguments. */
    printf("The sum is %d\n", sum);

    /* The return is optional. */
    return;
}

/* Each executable needs to have a main function with type int. */
int main() {
    print_sum(3, 4);
    return 0;
}
```
Local and global variables

```c
int x;
int y, z;
x = 1;

/* Functions can have local variables. */
void foo() {
    int x;
    x = 2;
}

/* Arguments are locally scoped. */
void bar(int x) {
    x = 3;
}
```
# Conditionals

```c
int foo(int x) {
    /* C has the usual boolean operators. */
    if (3 == x) {
        return 0;
    }
}

int bar() {
    /* Note that conditions are integer type, where 1 is true! */
    if (1) {
        return 0;
    }
}
```
Loops

**For loops**

```c
void foo() {
    int i;
    for (i = 1; i < 10; ++i) {
        printf("%d\n", i);
    }
}
```

**While loops**

```c
void bar() {
    int lcv = 0;
    while (lcv < 10) {
        printf("%d\n", lcv);
        ++lcv;
    }
}
```
When can we call what?

Each function needs to be *declared* (but not necessarily *defined*) before we call it.

```c
/* Declaration. */
void print_sum(int, int);

/* Each executable needs to have a main function with
type int. */
int main() {
    print_sum(3, 4);
    return 0;
}

/* Definition. */
void print_sum(int arg1, int arg2) {
}
```
Including headers

Header definitions allow us to use things defined elsewhere.

- **Header files** (.h files) typically contain *declarations* (variables, types, functions). Declarations tell the compiler “these functions are defined somewhere.”
- Function *definitions* typically go in .c files.
- Angle brackets indicate library header files; quotes indicate local header files.

```
#include <stdio.h>    /* Library file. */
#include "mylib.h"    /* Local file. */
```

- The compiler’s `-I` flag indicates where to look for library files (`gcc -I [libdir] -o [output] [file]`).
Until tomorrow...

Homework (due tomorrow)

- Get a C compiler up and running.
- Compile and run “Hello world.” Make a small extension to print the system time.
- Play around with gdb and valgrind.
- 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

For information about citing these materials or our Terms of Use, visit http://ocw.mit.edu/terms.