6.096 Lecture 1 – Introduction to C
Welcome to the Memory Jungle

Andre Kessler

January 8, 2014
Outline

1. Motivation
2. Class Logistics
3. Memory Model
4. Compiling
5. Wrap-up
def binary_search( data, N, value):
    lo, hi = 0, N - 1

    while lo < hi:
        mid = (lo + hi) / 2

        if data[mid] < value:
            lo = mid + 1
        else:
            hi = mid

    if hi == lo and data[lo] == value:
        return lo
    else:
        return N
size_t binary_search( int *data, size_t N, int value ) {
    size_t lo = 0, hi = N - 1;

    while( lo < hi ) {
        size_t mid = lo + ( hi - lo ) / 2;

        if( data[mid] < value ) {
            lo = mid + 1;
        } else {
            hi = mid;
        }
    }

    return ( hi == lo && data[lo] == value ) ? lo : N;
}
Why C or C++?

Speed

---

Motivation

Andre Kessler

6.S096 Lecture 1 – Introduction to C

January 8, 2014 5 / 26
Why C or C++?

Power

- C: direct access to memory and memory management, expressive but terse
- C++: all the power of C, plus stronger typing, object-oriented and generic programming, and more
Why C or C++?

Ubiquity

- C: operating systems, drivers, embedded, high-performance computing
- C++: large software projects everywhere
- Examples: Linux kernel, Python, PHP, Perl, C#, Google search engine/Chrome/MapReduce/etc, Firefox, MySQL, Microsoft Windows/Office, Adobe Photoshop/Acrobat/InDesign/etc, lots of financial/trading software, Starcraft, WoW, EA games, Doom engine, and much, much more
Effective Programming

Writing good, standards-compliant code is not hard.
Writing good, standards-compliant code is not hard.

Doing so will make your life much easier.
Effective Programming

Writing good, standards-compliant code is not hard.

Doing so will make your life much easier.

There is a lot of bad code out there.
Effective Programming

Writing good, standards-compliant code is *not hard*. Doing so will make your life *much easier*. There is a lot of *bad code* out there. You are *better* than that!
Anyone can write good, readable, standards-compliant code.
# Course Syllabus

<table>
<thead>
<tr>
<th>Day</th>
<th>Date</th>
<th>Topic</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/8</td>
<td>Introduction to C: memory and the compiler</td>
<td>(54-100)</td>
</tr>
<tr>
<td>2</td>
<td>1/10</td>
<td>Subtleties of C: memory, floating point</td>
<td>(10-250)</td>
</tr>
<tr>
<td>3</td>
<td>1/13</td>
<td>Guest lectures: Assembly and Secure C</td>
<td>(54-100)</td>
</tr>
<tr>
<td>4</td>
<td>1/15</td>
<td>Transition from C to C++</td>
<td>(54-100)</td>
</tr>
<tr>
<td>5</td>
<td>1/17</td>
<td>Object-oriented programming in C++</td>
<td>(32-155)</td>
</tr>
<tr>
<td>6</td>
<td>1/22</td>
<td>Design patterns and anti-patterns</td>
<td>(54-100)</td>
</tr>
<tr>
<td>7</td>
<td>1/24</td>
<td>Generic programming: templates and more</td>
<td>(54-100)</td>
</tr>
<tr>
<td>8</td>
<td>1/27</td>
<td>Projects: putting it all together</td>
<td>(34-101)</td>
</tr>
<tr>
<td>9</td>
<td>1/29</td>
<td>Projects: continued</td>
<td>(34-101)</td>
</tr>
<tr>
<td>10</td>
<td>1/31</td>
<td>Grab-bag: coding interviews, large projects</td>
<td>(34-101)</td>
</tr>
</tbody>
</table>
Grading

6 units U credit, graded Pass/Fail

- Coding assignments
  - Three assignments worth 20%, final worth 40%.
  - Submit to https://6.s096.scripts.mit.edu/grader
  - Just got the server, so don’t try signing up immediately- but it will be set up before today’s open lab at 7pm
  - Automatic instantaneous feedback

- Code reviews
  - Two reviews of code by your peers
  - More details later

To Pass

- at least 50% of available coding assignment points
- must submit both code reviews
None required.

However, the following books are on reserve at the library and may be useful as references. Highly recommended if you end up doing more C/C++ coding after this course.

**Recommended**

*The C Programming Language* by B. Kernighan and D. Ritchie ("K&R")

*The C++ Programming Language, 4th ed.* by Bjarne Stroustrup

*Effective C++, More Effective C++,* and *Effective STL* by Scott Meyers
The Minimal C Program

nothing.c: takes no arguments, does nothing, returns 0 ("exit success")

```c
int main(void) {
    return 0;
}
```

1. To compile: `make nothing`
2. Previous step produced an executable named `nothing`
3. To run: `./nothing`
The Minimal C Program

nothing.c: takes no arguments, does nothing, returns 0 ("exit success")

```c
int main(void) {
    return 0;
}
```

1. To compile: `make nothing`
2. Previous step produced an executable named `nothing`
3. To run: `.\nothing`
nothing.c: takes no arguments, does nothing, returns 0 ("exit success")

```c
int main(void) {
    return 0;
}
```

1. To compile: make nothing
2. Previous step produced an executable named nothing
3. To run: ./nothing

But you probably have higher aspirations for your programs...
Hello, world!

`hello.c`: takes no arguments, prints “Hello, world!”,
returns 0

```c
int main(void) {
    return 0;
}
```
Hello, world!

hello.c: takes no arguments, prints “Hello, world!” , returns 0

```c
#include <stdio.h>

int main(void) {
    return 0;
}
```
Hello, world!

`hello.c`: takes no arguments, prints “Hello, world!”, returns 0

```c
#include <stdio.h>

int main(void) {
    printf("Hello, world!\n");
    return 0;
}
```
Hello, world!

hello.c: takes no arguments, prints “Hello, world!”

```c
#include <stdio.h>

int main(void) {
    printf( "Hello, world!\n" );
    return 0;
}
```

1. To compile: make hello
2. Previous step produced an executable named hello
3. To run: ./hello
Hello, world!

hello.c: takes no arguments, prints “Hello, world!” , returns 0

```c
#include <stdio.h>

int main(void) {
    printf( "Hello, world!\n" );
    return 0;
}
```

1. To compile: make hello
2. Previous step produced an executable named hello
3. To run: ./hello
4. Hello, world!
Hello, world!

hello.c: takes no arguments, prints “Hello, world!”, returns 0

```c
#include <stdio.h>

int main(void) {
    printf( "Hello, world!\n" );
    return 0;
}
```

1. To compile: make hello
2. Previous step produced an executable named hello
3. To run: ./hello
4. Hello, world!
How do you get at this information about memory?
How do you get at this information about memory?

Through pointers; that is, the & and * operators
Pointers

How do you get at this information about memory?

Through pointers; that is, the & and * operators

```c
int a = 5; The address of a is &a.
int *a_ptr = &a; Read declarations from right to left. See it this way: “*a_ptr is declared to be of type int.”
```
How do you get at this information about memory?

Through pointers; that is, the `&` and `*` operators

```c
int a = 5; The address of a is &a.
int *a_ptr = &a; Read declarations from right to left.
See it this way: “*a_ptr is declared to be of type int.”
```

You can apply `&` to any addressable value (“lvalue”)

```c
return &5;
// error: lvalue required as unary ‘&’ operand
```
It’s all about the memory

```c
int a = 5;
int *a_ptr = &a;
```

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Value</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;a</td>
<td>0x7fff6f641914</td>
<td>a</td>
</tr>
<tr>
<td>&amp;a_ptr</td>
<td>0x7fff6f641918</td>
<td>a_ptr</td>
</tr>
</tbody>
</table>

Note: definitely a 64-bit machine, since the addresses are larger than $2^{32}$. 
It’s all about the memory

```c
int a = 5;
int *a_ptr = &a;
```

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Value</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;a</td>
<td>0x7fff6f641914</td>
<td>0x000000000005</td>
</tr>
<tr>
<td>&amp;a_ptr</td>
<td>0x7fff6f641918</td>
<td>0x??????????????</td>
</tr>
</tbody>
</table>

Note: definitely a 64-bit machine, since the addresses are larger than $2^{32}$. 
It’s all about the memory

```c
int a = 5;
int *a_ptr = &a;
```

<table>
<thead>
<tr>
<th></th>
<th>Memory Address</th>
<th>Value</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;a</td>
<td>0x7fff6f641914</td>
<td>0x000000000005</td>
<td>a</td>
</tr>
<tr>
<td>&amp;a_ptr</td>
<td>0x7fff6f641918</td>
<td>0x7fff6f641914</td>
<td>a_ptr</td>
</tr>
</tbody>
</table>

Note: definitely a 64-bit machine, since the addresses are larger than $2^{32}$. 
C Data Types

For the bit counts, we're assuming a 64-bit system.

char (8)
short (16), int (32),
   long (64), long long (64+)
float (32), double (64), long double (80)
## C Data Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Size in bits</th>
<th>Format</th>
<th>Value range</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Approximate</strong></td>
<td><strong>Exact</strong></td>
</tr>
<tr>
<td>character</td>
<td>8</td>
<td>signed (one's complement)</td>
<td>-127 to 127</td>
</tr>
<tr>
<td></td>
<td></td>
<td>signed (two's complement)</td>
<td>-128 to 127</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unsigned</td>
<td>0 to 255</td>
</tr>
<tr>
<td>integral</td>
<td>16</td>
<td>signed (one's complement)</td>
<td>-32767 to 32767</td>
</tr>
<tr>
<td></td>
<td></td>
<td>signed (two's complement)</td>
<td>-32768 to 32767</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unsigned</td>
<td>0 to 6,55 · 10^4</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>signed (one's complement)</td>
<td>-2,147,483,647 to 2,147,483,647</td>
</tr>
<tr>
<td></td>
<td></td>
<td>signed (two's complement)</td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unsigned</td>
<td>0 to 4,294,967,295</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>signed (one's complement)</td>
<td>-9,223,372,036,854,775,807 to 9,223,372,036,854,775,807</td>
</tr>
<tr>
<td></td>
<td></td>
<td>signed (two's complement)</td>
<td>-9,223,372,036,854,775,808 to 9,223,372,036,854,775,808</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unsigned</td>
<td>0 to 18,446,744,073,709,551,615</td>
</tr>
</tbody>
</table>
| floating point | 32    | IEEE-754                                     | ± 3.4 · 10^±38  \((-7\text{ digits})\) | • min subnormal: ± 1.401,298,4 · 10^{-47}  
|          |              |                                            | • min normal: ± 1.175,494,3 · 10^{-38}  
|          |              |                                            | • max: ± 3.402,823,4 · 10^{38}  |
|          | 64           | IEEE-754                                     | ± 1.7 · 10^±38  \(\sim 15\text{ digits})| • min subnormal: ± 4.940,656,458,412 · 10^{-324}  
|          |              |                                            | • min normal: ± 2.225,073,858,507,201,4 · 10^{-308}  
|          |              |                                            | • max: ± 1.797,693,134,862,315,7 · 10^{308}  |

Development Environment

- We officially support development with gcc on Linux.
  - If you don’t have a computer running Linux, then that’s what today’s lab time is devoted to.
  - Some options: SSH with PuTTY, Cygwin, Xcode on Mac
- Create a directory dev/
- Copy the file http://web.mit.edu/6.s096/www/dev/Makefile to this directory.
- To compile a file filename.c, just run “make filename”.
What happens when we compile?

```
#include <stdio.h>

int do_thing( float a, float b ) {
    /* do things */
}

void call(void) {
    /* do stuff */
    do_thing( a, b );
    /* do more */
}

int main(void) {
    call();
    return 0;
}
```
What happens when we compile?

- Three functions `main`, `call`, and `do_thing`.
- **Object code** is produced for each
- When we run: the object code is loaded into memory
- Each function that is called is in memory, somewhere.
Examples

Time for some examples!
With great power comes great responsibility

- C is focused on speed; always checking array bounds/memory access would slow you down.
- Simple typo `for( int i = 0; i <= N; ++i )` can cause corruption
- Memory corruption can cause totally unexpected, hard-to-debug behavior at worst
- At best: Segmentation fault (core dumped)
- (At least it’s more obvious!)
“C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do, it blows your whole leg off.”

— Bjarne Stroustrup, creator of the C++ programming language
Wrap-up & Friday

Open lab from 7-9pm today in 32-044

- Bring your laptops, get a C programming environment working
- Test out the automatic grader

Class on Friday is in 10-250 at 2pm.

- Will cover floating point arithmetic, memory management, and headers in more depth.

Questions?

- I’m available after class, later during lab, or on Piazza.