6. S096 Lecture 1 – Introduction to C
Welcome to the Memory Jungle

Andre Kessler
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
```c
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

Graph of program speed across language implementations removed due to copyright restrictions.
Source: [http://benchmarksgame.alioth.debian.org/u64q/which-programs-are-fastest.php](http://benchmarksgame.alioth.debian.org/u64q/which-programs-are-fastest.php).
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
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>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction to C: memory and the compiler</td>
</tr>
<tr>
<td>2</td>
<td>Subtleties of C: memory, floating point</td>
</tr>
<tr>
<td>3</td>
<td>Guest lectures: Assembly and Secure C</td>
</tr>
<tr>
<td>4</td>
<td>Transition from C to C++</td>
</tr>
<tr>
<td>5</td>
<td>Object-oriented programming in C++</td>
</tr>
<tr>
<td>6</td>
<td>Design patterns and anti-patterns</td>
</tr>
<tr>
<td>7</td>
<td>Generic programming: templates and more</td>
</tr>
<tr>
<td>8</td>
<td>Projects: putting it all together</td>
</tr>
<tr>
<td>9</td>
<td>Projects: continued</td>
</tr>
<tr>
<td>10</td>
<td>Grab-bag: coding interviews, large projects</td>
</tr>
</tbody>
</table>
Grading

6 units U credit, graded Pass/Fail

- Coding assignments
  - Three assignments worth 20%, final worth 40%
  - 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
Textbooks

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
nulling.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!”, 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!`
Pointers

How do you get at this information about memory?

Through pointers; that is, the & and * operators

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”)

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>Identifier</th>
<th>Memory Address</th>
<th>Value</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>0x0000000000000005</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 of C data types removed due to copyright restrictions.

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 `Makefile` to this directory.
- To compile a file `filename.c`, just run “make filename”.
What happens when we compile?

```c
#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.
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

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

Class on Friday

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

Questions?