6.S096 Lecture 1 – Introduction to C

Welcome to the Memory Jungle

Andre Kessler

Outline

1. Motivation
2. Class Logistics
3. Memory Model
4. Compiling
5. Wrap-up

First Example (Python)

```python
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
```

First Example (C)

```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.

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.

Doing so will make your life much easier.

There is a lot of bad code out there.

You are better than that!

Effective Programming

Anyone can write good, readable, standards-compliant code.

Course Syllabus

Day	Topic
1	Introduction to C: memory and the compiler
2	Subtleties of C: memory, floating point
3	Guest lectures: Assembly and Secure C
4	Transition from C to C++
5	Object-oriented programming in C++
6	Design patterns and anti-patterns
7	Generic programming: templates and more
8	Projects: putting it all together
9	Projects: continued
10	Grab-bag: coding interviews, large projects

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 Stroustrop
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”)

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

1. To compile: `make nothing`
2. Previous step produced an executable named `nothing`
3. To run: `./nothing`
4. Surprise! Does 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

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

	Memory Address	Value	Identifier
&a	0x7fff6f641914	0x????????????	a
&a_ptr	0x7fff6f641918	0x????????????	a_ptr

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

It's all about the memory

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

	Memory Address	Value	Identifier
&a	0x7fff6f641914	0x000000000005	a
&a_ptr	0x7fff6f641918	0x????????????	a_ptr

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

It's all about the memory

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

	Memory Address	Value	Identifier
&a	0x7fff6f641914	0x000000000005	a
&a_ptr	0x7fff6f641918	0x7fff6f641914	a_ptr

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.

Courtesy of http://en.cppreference.com/w/cpp/language/types

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 `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 Stroustrop, 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?

6.S096 Effective Programming in C and C++

IAP 2014