CSE 333 – SECTION 1

code/compile/link/run process
Your TAs

- Matthew Kang
  - mkang204@cs.washington.edu
- Ricardo Martin
  - rmartin@cs.washington.edu
- Reid Zhang
  - zzt124@cs.washington.edu
- Renshu Gu
  - renshugu@cs.washington.edu

- Office hours will be posted soon

Contact us:
- Use the staff email instead of our individual emails
- cse333-staff@cs.washington.edu
Questions, Comments, Concerns?

- Talk to us
- Use the discussion board
Course website

- [https://courses.cs.washington.edu/courses/cse333/15au](https://courses.cs.washington.edu/courses/cse333/15au)

- Links to:
  - lecture slides
  - code discussed in lectures
  - assignments, exercises (including due dates)
  - various C/C++/Linux resources (e.g. [http://www.cplusplus.com](http://www.cplusplus.com))
Upcoming Due Dates

• Exercise 0 – Due Friday October 2\textsuperscript{nd}, 2015 @ 11:15 AM
Refresher on C

• General purpose programming language
  • Created in 1972 by Dennis Ritchie
  • most recently updated in 1999 (C99), 2011 (C11)

• Often used in low-level system programming
• Procedural (not object-oriented)
• Supports use of pointer arithmetic
• Provides facilities for managing memory
• C passes all of its arguments by value
  • Pass-by-reference is simulated by passing the address of a variable
C workflow

1. Source files (.c, .h) - Editor (emacs, vi) or IDE (eclipse)
2. foo.c - Compilation
3. foo.o - Object files (.o)
4. bar.c - Linking
5. bar.o - Linking
6. libZ.a - Static libraries
7. libc.so - Shared libraries
8. bar - Executable
9. execute, debug, profile, ...
10. process

Diagram showing the workflow of C programming, including the steps of editing, compiling, linking, and executing a program with examples of source files, object files, and static and shared libraries.
Let’s Go Through The Workflow!
From C to machine code

C source file (dosum.c)

```c
int dosum(int i, int j) {
    return i+j;
}
```

C compiler (gcc -S)

Assemble:
```
dosum:
pushl %ebp
movl %esp, %ebp
movl 12(%ebp), %eax
addl 8(%ebp), %eax
popl %ebp
ret
```

Machine code (dosum.o)

```
80483b0:  55
89 e5 8b 45
0c 03 45 08
5d c3
```
Skipping assembly language

- Most C compilers generate .o files (machine code) directly
- i.e., without actually saving the readable .s assembly file
Multi-file C programs

```c
#include <stdio.h>

int dosum(int i, int j) {
    return i+j;
}

int main(int argc, char **argv) {
    printf("%d \n", dosum(1,2));
    return 0;
}
```

- This "prototype" of `dosum()` tells GCC about the types of `dosum`'s arguments and its return value.
- `dosum()` is implemented in `dosum.c`.

C source file (dosum.c)

C source file (sumnum.c)
Multi-file C programs

C source file (dosum.c)

```c
int dosum(int i, int j) {
    return i+j;
}
```

why do we need this #include?

C source file (sumnum.c)

```c
#include <stdio.h>

int dosum(int i, int j);

int main(int argc, char **argv) {
    printf("%d\n", dosum(1,2));
    return 0;
}
```

where is the implementation of printf?
Compiling multi-file programs

- Multiple object files are **linked** to produce an executable
  - standard libraries (libc, crt1, ...) are usually also linked in
  - a library is just a pre-assembled collection of .o files
Object files

- `sumnum.o, dosum.o` are **object files**
- Each contains machine code produced by the compiler
- Each might contain references to external symbols
  - variables and functions not defined in the associated `.c` file
  - e.g., `sumnum.o` contains code that relies on `printf()` and `dosum()`,
    but these are defined in `libc.a` and `dosum.o`, respectively.
- Linking resolves these external symbols while smooshing together object files and libraries
Any Questions?