CSCI [4|6]730:
A C Refresher or Introduction

Hello Word!

~/Ctest/

How do I learn C?

In addition to syntax you need to learn:

- the Tools.
- the Libraries.
- And the Documentation.

Diving In: A Simple C Program
1-hello-word.c

```
/* header files go up here -- specifies headers needed for routines */
/* note that C comments are enclosed within a slash
and a star, and may wrap over lines */
// but if you use the latest gcc, two slashes will work too, like C++
#include <stdio.h>  /* prototypes processed by cpp */








/* main returns an integer */
int main( int argc, char *argv[] )
{
printf( "hello, world\n" );
return(0);   /* returns 0 by conventions indicates all went well */
}
```

declarations

functions()

main()

Diving In: A Simple C Program
1-hello-word.c

```
/* header files go up here -- specifies headers needed for routines */
/* note that C comments are enclosed within a slash
and a star, and may wrap over lines */
// but if you use the latest gcc, two slashes will work too, like C++
#include <stdio.h>  /* prototypes processed by cpp */

/* main returns an integer */
int main( int argc, char *argv[] )
{
/* printf is our output function; by default it writes to standard out */
/* printf returns an integer, but we ignore it here */
/*1 [stout]   >& redirect stout and stderr */
/* >& /dev/null - suppress all output */ /*(cat f1 > myout) >& myerror */

printf( "hello, world\n" );
/* return 0 by conventions indicates all went well */
return(0);
}
```

declarations

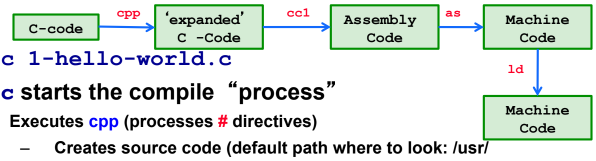
functions()

main()

*.c File Name

- Naming the program (e.g., 1-hello-world.c, main.c)
  - » Arbitrary – Not Like in Java where file name is connected with file content (class name).
  - » Constraint: Need to end with a *.c'

How to Compile and Run a
C-program: 1-hello-world.c



- gcc 1-hello-world.c
- gcc starts the compile "process"
  1. Executes cpp (processes # directives)
     - – Creates source code (default path where to look: /usr/include)
  2. Compilation (cc1)
     - – Transforms C code to assembly code
  3. Assembler (as) runs
     - – Transforms assembly code to machine code
  4. Linker (ld) runs
     - – Links code together to create the final executable
- ./a.out
  - » int main(int argc, char *argv[])

1

argv[1]=NULL (end)

Compile Command line & ‘flags’

```
● prompt> gcc -o first first.c # -o
  lets you specify the executable name
● prompt> gcc -Wall first.c # -Wall
  gives much better warnings
● prompt> gcc -g first.c # use -g to
  enable debugging with gdb
● prompt> gcc -O first.c # use -O to
  turn on optimization
```

Linking Libraries

- Example: `fork()` requires a library, namely the C-library. The C library is *automatically* linked, so all we need then is :
  - » The ‘including’ the right `#include` file “<>”, -i, -I to to find the prototype of the function (return type, date types of parameters).
  - » How to find out:
    - – `man fork`
  - » CAVEAT: the controversial and dreaded `LD_LIBRARY_PATH`
  - » http://www.cs.uga.edu/~maria/classes/1730-Spring-2006/gcc-getstarted.txt
  - » May fix (e.g., readline) problems

Lets say that again….

- fork() requires the C-library (`clib`). The C library is *automatically* linked in, so all we need then is :
  - » How do you know what to include?
  - » `man fork`
  - » BUT – Wait a minute why a library - Fork is a system call! [a request of ‘service’ by the OS from the application]
    - – `C library provides C –wrappers for all system calls – which simply traps into the OS`
    - – The ‘real’ system call in Linux e.g., is `sys_fork()`

Other Libraries: The Math Library

- gcc [flag ...] file ... -lm [library ...]
- #include <math.h>
  - » In /usr/lib
  - » Statically linked .a (compile time)
    - – Combines code (copies) directly into executable
  - » Dynamically linked shared library .so (run time)
    - – Smaller code base (can be shared by multiple processes)
    - – A reference and only links when needed, smaller code base (some work), hooks in code triggers the run time system to load in the library, only when needed
  - » /usr/libm.a & /usr/libm.so
  - » Link editor searches for library in a certain order.
  - » -lm directory path include) and –L(directory path)

Multiple Files (`hw.c, helper.c Makefile2`)

```
prompt> gcc –o hw hw.c helper.c -lm
```

Problem: Remake everything (2 programs here) every time, even if the change is only in hw.c

Approach: Separate 2 step compilation process that only re-compiles source files that have been modified

- Create object files then link *.o files
- Then link these files into an executable

Separate Compilation

```
# note that we are using -Wall for
  warnings and -O for optimization
prompt> gcc -Wall -O -c hw.c
prompt> gcc -Wall -O -c helper.c
prompt> gcc -o hw hw.o helper.o -lm
```

- -c flag produces an object file
  - Machine level code (not executable)
  - Need to link to make an executable

```
prompt> gcc -o hw hw.c helper.c -lm
```

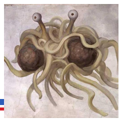
Make & Makefiles

- Make make things easier to handle the compilation process.

```
target: prerequisite1 prerequisite2
  command1
  command2
```

- Target usually the name of executable of (1) the object file or (2) the action (like clean)

Make – Makefiles (be aware of the dreaded white space phenomena)



```
hw: hw.o helper.o
   gcc -o hw hw.o helper.o -lm
hw.o: hw.c
  gcc -O -Wall -c hw.c
helper.o: helper.c
  gcc -O -Wall -c helper.c
clean:
  rm -f hw.o helper.o hw
```

OK what is going on here?

```
hw: hw.o helper.o
   gcc -o hw hw.o helper.o -lm
hw.o: hw.c
  gcc -O -Wall -c hw.c
helper.o: helper.c
  gcc -O -Wall -c helper.c
clean:
  rm -f hw.o helper.o hw
```

- Goes to target hw (first target) need the prerequisites
- Check them in turn (according to date) and see if they need to be re-made

Make macros

- Also you can create macros:
  - » CC = gcc
  - » OBJECTS = data.o main.o
  - » Project1: $(OBJECTS)
- Examples of Special macros
  - » CC, CFLAGS (compiler, and compiler flags)
  - » $@ short cut for full name of current target

```
%.o : %.c
     $(CC) –c –o $@ $(CFLAGS)
```

Debugging

```
#include <stdio.h>
struct Data {
int x;
};
int main( int argc, char *argv[] )
{
struct Data *p = NULL;
printf("%d\n", p->x);
}
```

Debugging

- `gcc –g  –o 3-buggy  3-buggy.c`
- `{odin:maria:428} 3-buggy`
- `Segmentation Fault(coredump)`
- `gdb 3-buggy`
  - `run`
  - `print p`
  - `break main`

GDB

- `(gdb) help`
- `Help running`
- `Help files`
- `Help breakpoints`

Man

- man XXX
- man -k

Lets get going: Create A Child
2-lets-fork.c

```
#include <stdio.h>  /* printf */
#include <unistd.h> /* fork is defined here */

pid_t childpid = 0 ;  /* descriptive variables makes code readable */
int main( int argc, char *argv[] )
{
printf( "I have no children, but I need one\n" );
if( (childpid = fork()) == 0 )
        {
        printf("\nHello from child\n");
        fflush(stdout);
        }
  else
        { /* what is childpid? Here? */
        printf("\nHello from parent\n");
        fflush(stdout);
        printf("(A) my child (%d) is on his own -- exiting \n", childpid );
        }
  /* printf("(B) my child (%d) is on his own -- exiting \n", childpid ); */
return(0);  // well that was fun!
}
```

declarations

functions()

main()

- Parent 'waiting' for the child add a wait
  - 3-lets-fork.c

The *Ultimate* C Reference Guides

- *"The C book" or the "K & R Book":*
  - *The C Programming Language, by Brian Kernighan and Dennis Ritchie (thin, concise and all you really need…)*
- *The GDB Booklet*
  - *Debugging with GDB: The GNU Source-Level Debugger, by Richard M. Stallman, Roland H. Pesch*
    - *http://sourceware.org/gdb/current/onlinedocs/gdb.html*
- *The Unix System Programming Book*
  - *Advanced Programming in the UNIX Environment, by W. Richard Stevens*